UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Framework para Comércio Eletrônico, via Internet Móvel, mediado por Agente de Software

Diogo Sobral Fontes

Orientador: Prof. Dr. Antonio Francisco do Prado

São Carlos – SP, Fevereiro de 2002

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Framework para Comércio Eletrônico, via Internet móvel, mediado por Agente de Software

Diogo Sobral Fontes

Dissertação apresentada como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

F683fc

Fontes, Diogo Sobral.
Framework para comércio eletrônico, via Internet móvel, mediado por agente de software / Diogo Sobral Fontes. -- São Carlos : UFSCar, 2005.
93p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2002.

1. Internet (Redes de computação). 2. Framework (Programa de computador). 3. Agentes de software inteligentes. I. Título.

CDD: 004.67 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

FRAMEWORK PARA COMÉRCIO ELETRÔNICO, VIA INTERNET MÓVEL, MEDIADO POR AGENTE DE SOFTWARE

Diogo Sobral Fontes

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Membros da Banca:

Prof. Dr. Antonio Francisco do Prado
(Orientador – DC/UFSCar)

Prof. Dr. Sérgio Donizetti Zorzo
(DC/UFSCar)

Prof. Dr. Roberto da Silva Bigonha
(DCC/UFMG)

São Carlos
Fevereiro/2002

À minha querida esposa Sueli,
minha princesinha Dani e
meu amigão Diego,
pelo carinho e compreensão.

AGRADECIMENTOS

A Deus, em primeiro lugar, que me deu saúde, inteligência e colocou-me com as pessoas certas para ajudar-me neste trabalho.

Ao prof. Prado, meu orientador, por ter me apoiado, procurando sempre me incentivar, visando meu amadurecimento como aluno e pesquisador.

Ao incansável Marcelo Sant'Anna, pelo espírito de perseverança e dedicação à pesquisa que transmitiu durante o período em que trabalhamos juntos.

A minha mãe que sempre me apoiou e aos meus irmãos por participarem deste trabalho, mesmo que de forma indireta.

A todos os amigos da UFSCar, com destaque especial para os queridos amigos, Paula, Eli, Maris, Mário, Xuxa, Murakami, Wesley, Calebe, Adail, Iolanda, Ricardo, Val, Balbino, Vinicius e Eduardo, os amigos da PUC-Rio, Daflon, Guga, Léo, Felipe, Alexandre, e ao aluno e amigo Mindu, os quais, de uma forma ou de outra, tiveram uma participação muito importante neste trabalho de mestrado.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A tecnologia emergente de agentes de software vem se tornando popular nos últimos anos, devido ao aumento das aplicações para Internet. Agentes de software foram inicialmente utilizados para filtrar informações, encontrar pessoas com interesses similares e automatizar procedimentos repetitivos. Atualmente eles estão sendo utilizados em aplicações para comércio eletrônico, promovendo uma revolução na forma de conduzir as negociações entre pessoas e empresas. Essa tecnologia automatiza diversos estágios tradicionais do processo de compra e venda pela *Internet*, minimizando os custos e eliminando barreiras geográficas. Este projeto pesquisou o desenvolvimento de um *Framework* para Comércio Eletrônico, via Internet Móvel, mediado por Agentes de Software (FCEMAS). O *framework* baseia-se em padrões de projeto e soluções para Internet Móvel usando o WAP (*Wireless Application Protocol).* Foram construídos componentes, implementados com a tecnologia *Enterprise Java Beans* (EJB), para aplicações do domínio de farmácia, que permitiram validar o *framework.*

The emergent technology of software agents is becoming popular in the last years, due to the increasing of Internet applications. Software Agents were initially used to filter information, find people with similar interests and to automate repetitive processes. Nowadays, they are being used in e-commerce applications, making a revolution in the way of leading the negotiations between people and companies.
This technology automates several traditional steps of the purchase and sale process for Internet, reducing the costs and eliminating geographical barriers.
This research presents a proposal for the development of an E-Commerce Framework, via Mobile Internet, mediated by Software Agents (FCEMAS). The framework is based on project patterns and Mobile Internet solutions using WAP (*Wireless Application Protocol).*

Capítulo 1

Introdução

O comércio eletrônico já é uma das grandes utilizações da Internet, causando mudanças na forma como as empresas devem se estruturar para esta nova realidade e criando um mercado com características marcantes que não podem ser desconsideradas. Entre as principais características deste novo mercado estão: velocidade de adaptação às novas tecnologias, personalização e globalização.

A velocidade de adaptação é a capacidade de as empresas adaptarem-se mais rapidamente às tecnologias emergentes de forma a acompanhar a evolução natural do mercado. A personalização reflete a facilidade com que a empresa se relaciona com o usuário final. Empresas interessadas em comércio eletrônico devem considerar o fato de que a utilização dos serviços é fortemente individualizada, uma vez que existe um único usuário à frente de cada computador e, conseqüentemente, todos os serviços devem ser “customizados” e “entendidos” como se fossem feitos sob medida para cada um. Finalmente a globalização, apesar de não ser uma regra de atuação em um mercado global, deve ser considerada, uma vez que não existem mais fronteiras geográficas para o comércio de produtos e serviços. Assim, as empresas devem estar preparadas para uma eventual atuação em um mercado globalizado. Estas características fazem da tecnologia de agentes de software, também chamados "empregados digitais", uma tecnologia promissora para aplicações de comércio eletrônico.

A necessidade de automatizar diferentes etapas do comércio eletrônico, particularmente o processo de negociação entre compradores e vendedores faz com que a tecnologia de agentes de software seja propícia às aplicações de comércio eletrônico. A utilização de agentes aumenta,

consideravelmente, com relação às aplicações que interagem com usuários distribuídos pela Internet, incluindo os que utilizam dispositivos móveis.

Considerando que ainda existe uma carência de sistemas que suportem esta nova forma de comércio eletrônico, este trabalho apresenta um *framework* para construção de sistemas para comércio eletrônico, mediado por agentes de software, responsáveis por realizar as negociações, via *Web ou* WAP (*Wireless Application Protocol)* [Wap01], baseados no princípio da reutilização, para criação de mercados virtuais.

Um mercado virtual é um sistema voltado para Internet onde os agentes de software interagem e negociam em nome de seus respectivos usuários para comprar, vender ou encontrar produtos e serviços de seus interesses. Nesse tipo de mercado, os usuários são potenciais compradores ou vendedores [Fon01a].

A reutilização é um princípio essencial na área de Engenharia de Software para garantir a redução de esforços e custos no processo de desenvolvimento de sistemas de software. Na tecnologia orientada a objetos, a reutilização de software pode ser assegurada com a adoção de padrões, *frameworks* de domínios específicos, e com componentes de software previamente testados.

Padrões são soluções provadas para um problema ocorrido repetidamente dentro de um certo contexto, aumentando o grau de reutilização de uma aplicação [Fon01a]. Um *framework* é caracterizado por um conjunto de classes abstratas e concretas que colaboram para prover uma estrutura para implementar uma aplicação [Dso98]. Os *frameworks* podem ser classificados em horizontais, quando destinados a prover uma infra-estrutura comum a muitos domínios de problema, e em verticais, quando destinados à construção de aplicações em um domínio de problema específico.

Os trabalhos desta pesquisa são apresentados conforme segue: Capítulo 2 apresenta uma síntese da revisão bibliográfica a respeito das principais técnicas e tecnologias envolvidas na construção do *framework*, conceitos sobre *frameworks*; conceitos básicos sobre Agentes de Software; linguagem Java e os padrões Java*Servlet e Enterprise Java Beans* (EJB); protocolo *Wireless Application Protocol* e o método *Catalysis.* Capítulo 3 descreve o *framework* proposto. Capítulo 4 apresenta o desenvolvimento do domínio de aplicações. Capítulo 5 descreve a

reutilização do *framework* através de um estudo de caso. Finalmente, Capítulo 6 apresenta as conclusões, as contribuições e os trabalhos futuros.

Capítulo 2

Principais Técnicas

São abordadas neste capítulo as principais fontes bibliográficas estudadas para o desenvolvimento desta pesquisa. A seção 2.1 apresenta os conceitos sobre *frameworks*; a seção 2.2 apresenta os conceitos sobre Padrões de Projeto; a seção 2.3 apresenta os conceitos sobre Agentes de Software; a seção 2.4 apresenta a Linguagem Java, os padrões *Java Servlets e Enterprise Java Beans (EJB);* a seção 2.5 apresenta o protocolo *Wireless Aplication Protocol;* a seção 2.6 apresenta a abordagem de Comércio Eletrônico, utilizando Agentes de Software, e a seção 2.7 apresenta o método *Catalysis.*

2.1 *Frameworks*

A Engenharia de Software procura melhorar a qualidade dos processos e produtos de software com a redução dos esforços e custos da produção. Dentre as técnicas empregadas para atingir estes objetivos, destacam-se os *frameworks.*

Um *framework* é uma estrutura de classes inter-relacionadas, que constitui uma implementação inacabada, para um conjunto de aplicações de um domínio. Existem diversas definições de *framework* como as de:

- Jonhson [Jon96] – "*framework* é um projeto, reutilizável em todo ou parte de um sistema, que é representado por um conjunto de classes abstratas e por um modelo que representam as interações de suas instâncias";

- Gamma [Gam95] – "*framework* é um conjunto de classes relacionadas que suportam o reutilização em projetos com classes específicas do mesmo domínio. Considera-se não apenas uma hierarquia de classes, mas uma aplicação em miniatura, completa, com estrutura dinâmica e estática bem definidas"; e
- D'Souza [Dso98] – "*framework* é um conjunto de classes abstratas e concretas, que colaboram para prover uma estrutura para implementar uma aplicação".

Em outras palavras, seguindo as práticas mais comuns em engenharia de software, um *framework* pode ser descrito como um projeto orientado a objetos abstratos adaptáveis às diversas necessidades das aplicações. Deste modo, um *framework* funciona como um modelo para a construção de aplicações ou subsistemas dentro de um domínio. Todas as aplicações construídas a partir de um mesmo *framework* apresentam a mesma estrutura, diferenciando-se em seu comportamento, de acordo com a aplicação. Isto torna as aplicações desenvolvidas a partir do *framework,* mais fáceis de serem mantidas e mais consistentes para os desenvolvedores de sistemas, que não precisam recriar diferentes padrões de arquitetura para suas aplicações.

Um *framework* pode fornecer um grau de reutilização de até 80% [Pre99], pois além de fornecer reutilização de código, que é oferecido por classes e objetos, também oferece reutilização de projeto, liberando o desenvolvedor dos aspectos comuns daquele domínio de aplicação. A reutilização de projeto permite a reutilização em larga escala, que é uma condição necessária para o aumento de produtividade na atividade de desenvolvimento de software. Além disso, a reutilização de projeto é mais importante que a de código, porque o projeto é mais difícil de criar do que o código [Ral98].

Neste ponto, é importante diferenciar um *framework* de bibliotecas de classes orientadas a objetos e bibliotecas de procedimentos. Um *framework* corresponde a um projeto de alto nível, consistindo de classes abstratas e concretas que especificam uma arquitetura para aplicações. Uma biblioteca de classes é apenas um conjunto de classes, não necessariamente relacionadas, que fornece um conjunto de serviços disponibilizados através da interface pública de suas classes. De maneira similar, uma biblioteca de procedimentos fornece uma série de serviços através de chamadas às suas rotinas. Nenhuma das duas bibliotecas permite a reutilização de projeto.

O modo como os *framework*s operam também é diferente de outras abordagens. Em um *framework* bem-projetado, o desenvolvedor especifica somente o código de métodos de classes específicas que corresponderão à sua aplicação. O próprio *framework* irá se encarregar de chamar

este código quando for necessário. Em bibliotecas de classes ou procedurais, além do código da aplicação propriamente dita, o desenvolvedor deve especificar o fluxo de execução e a estrutura do programa.

Segundo Taligent [Tal96], as principais diferenças entre bibliotecas de classes e *frameworks* são mostras na Tabela 1.

Bibliotecas de Classes	**Frameworks**
conjunto de classes instanciadas pelos desenvolvedores	fornecem adaptações para subclasses, ou seja, classes que podem ser estendidas através da herança
acesso a funções pré-definidas	estende funções pré-definidas
nenhum fluxo de controle pré-definido	fluxo de controle de execução onde é definido o interrelacionamento das classes e as regras de como deve ser a seqüência de execução e suas dependências
nenhuma interação pré-definida	define interações entre objetos (inter-relacionamento entre as classes)
nenhum comportamento padrão	fornecem comportamento padrão

Tabela 1 – Comparação da biblioteca de classes e *frameworks*

Um *framework* pode ser classificado de acordo com a visibilidade da sua estrutura interna: *black box* e *white box*; e também, de acordo com a sua aplicabilidade: *framework* horizontal ou *framework* vertical [Fay97] .

- ***framework black box* -** disponibiliza somente uma interface, ou seja, a assinatura dos métodos públicos. O modo como é implementada a funcionalidade interna não é visível ao usuário. Este encapsulamento facilita seu uso, pois o usuário não necessita conhecer o funcionamento interno. Mas reduz sua flexibilidade, pois não permite ao usuário adaptar novas características [Joh98];
- ***framework white box*** - tem a sua estrutura interna visível ao usuário. Esta abertura de código permite que o usuário do *framework* estude sua funcionalidade e

posteriormente adapte o *framework* a suas necessidades [Bos97]. O uso de um *framework white box* requer que o desenvolvedor conheça a funcionalidade interna de suas classes, exigindo maior conhecimento da sua estrutura;

- ***framework* vertical** - desenvolvido para um domínio de aplicação específico. Para exemplificar, pode-se considerar um *framework* para controle de produção industrial. Neste domínio de problema, existem características comuns as várias indústrias. O *framework* é então construído de forma que atenda as características comuns, deixando as específicas para serem implementadas em aplicações construídas a partir dele; e
- ***framework* horizontal** - pode ser aplicado a qualquer domínio de problema, pois é desenvolvido de forma genérica sem preocupação com domínios específicos. Exemplos de *frameworks horizontais* são os que implementam infra-estruturas de comunicação de dados, de interface de usuários e de gerenciamento de dados (persistência de objetos). Em alguns casos, *frameworks* horizontais implementam padrões de arquitetura de software ("architectural patterns"), como por exemplo, o MVC (Model-View-Control) [Bus95].

A utilização de *framework*s fornece uma série de vantagens à atividade de desenvolvimento de software, destacando-se:

- Melhor desempenho no desenvolvimento de novas aplicações tornando-o mais rápido e mais barato, uma vez que se utilizam componentes pré-fabricados e pré-testados. O desenvolvedor não precisa descobrir novas classes e projetar interfaces. Basicamente, ele somente necessita reescrever o comportamento de métodos específicos de determinadas classes. Além disso, a estrutura do programa já está especificada e o fluxo de execução definido;
- Suporte para a reutilização de código e projeto. Normalmente, a reutilização de código é obtida através de polimorfismo e/ou herança. A reutilização de projeto é obtida a partir do próprio *framework*, uma vez que ele foi desenvolvido como uma solução genérica para uma determinada categoria de problemas;

- Redução dos custos de manutenção. Os *framework*s fornecem a maior parte do código necessário às aplicações, tornando menores os custos de manutenção. Devido à herança, quando um erro em um *framework* é corrigido, ou uma característica adicionada, os benefícios são imediatamente estendidos às novas classes; e
- O desenvolvimento de um *framework* baseado em um *framework* já existente permite aos desenvolvedores a reutilização de conceitos básicos, gastando menos tempo. Além disso, os desenvolvedores concentram-se em atividades específicas e mais complexas.

Outra técnica combinada para a construção do *Framework para Comércio Eletrônico* mediado por Agentes de Software é a de padrões de projeto, apresentado a seguir.

2.2 Padrões de Projeto

A utilização de padrões no desenvolvimento de software tem emergido como uma das mais promissoras abordagens para a melhoria da qualidade de software.

O conceito de padrões foi proposto pelo arquiteto Alexander [Ale77] para construção de arquiteturas e planejamento urbano. Na área de software, foram recentemente popularizados para apoiar uma larga variedade de domínios relacionados às atividades de desenvolvimento de projetos.

A utilização de padrões em sistemas complexos de softwares permite que soluções, previamente testadas, sejam reutilizadas, tornando o sistema mais compreensível, flexível, fácil de desenvolver e manter. O objetivo do uso de padrões de software é a disseminação das soluções de desenvolvimento de software já existentes [Joh92].

Padrões são soluções de um problema previamente testadas e aprovadas. Os desenvolvedores baseiam-se nesses padrões para melhor resolverem seus problemas, aumentado o grau de reutilização de uma aplicação [Fon01b, San00]. Alexander [Ale77] descreve que um padrão é constituído de três partes: um *Contexto*, um *Problema*, e uma *Solução*.

O *Contexto* descreve o domínio onde o problema ocorre; o *Problema* é uma questão que ocorre repetidamente e que deve ser solucionada e a *Solução* mostra como resolver o problema.

Buschmann [Bus95] divide os padrões em três categorias que representam diferentes níveis de abstração:

- *Padrões Arquiteturais (Architectural Patterns):* descrevem um esquema de organização estrutural e fundamental para um sistema de software. Eles fornecem um conjunto de subsistemas pré-definidos, especificando as responsabilidades e definindo regras e diretrizes para organizar os relacionamentos entre eles;
- *Padrões de Projeto (Design Patterns):* fornecem um esquema para refinar os subsistemas, os componentes de sistemas de software ou o relacionamento entre eles. Os mecanismos de cooperação entre componentes são descritos e definidos para encontrar soluções para os problemas de projeto em um contexto específico. Gamma [Gam95] define que um padrão de projeto nomeia, abstrai e identifica o essencial da estrutura de um projeto comum, tornando-o útil para criação de um projeto Orientado a Objeto reusável; e
- *Padrões em nível de Idiomas (Idioms):* especificam como implementar os aspectos particulares dos componentes e seus relacionamentos adequadamente às características da linguagem de programação.

Um Padrão de Projeto descreve uma estrutura comum para criar um projeto orientado a objetos reutilizáveis, baseados em soluções práticas. O termo padrão de projeto refere-se freqüentemente a qualquer padrão que diretamente trata de arquitetura de software, de projeto ou de implementação.

Buschmann [Bus95] também divide os padrões em três grupos que representam o grau de relacionamento entre os padrões:

- *Catálogo de Padrões (Patterns Catalog):* descrevem vários padrões, cada um de uma forma relativamente independente, para problemas comuns de projeto;
- *Sistemas de Padrões (Patterns Systems):* descrevem os relacionamentos entre os padrões e como estão conectados. Alexander [Ale77] usa o termo Linguagem (*language*) em comparação com Sistema; e

- *Linguagens de Padrões (Patterns Languages):* descrevem soluções para todos os problemas de projeto que podem ocorrer em um domínio.

Enquanto um Catálogo de Padrões representa um conjunto de padrões relacionados por um número pequeno de categorias, na qual não é necessário associá-los, mas que podem possuir relações entre si, um Sistema de Padrões representa um conjunto de padrões organizados em grupos e subgrupos para apoiar a construção e evolução de arquiteturas. Já as Linguagens de Padrões definem um conjunto de padrões associados, formando uma estrutura organizacional, conectados através de nodos em níveis, produzindo arquiteturas completas. Linguagens de Padrões evoluem a partir de sistemas de padrões, por meio de um processo de implantação de novos padrões e de regras para o relacionamento entre os mesmos.

O uso de padrões neste trabalho foi de extrema relevância, uma vez que foram aplicados os padrões *Singleton, State, Strategy, Object Pool* [Gam95], *Server Socket, Cliente Socket* [Gran99], *Persisten Object* [Yod98,Cag99] e *FrontControll* [Sun01].

Outra técnica combinada neste trabalho de pesquisa é a de Agentes de Software, apresentada a seguir.

2.3 Agentes de Software

Atualmente, a pesquisa na área de agentes de software divide-se em duas linhas principais. A primeira caracteriza-se pela pesquisa de agentes de software com habilidades na área de Inteligência Artificial. Estes agentes, muitas vezes denominados "agentes inteligentes", são capazes de realizar inferências, apreender com o comportamento do usuário. Baseados nestes dados, os agentes alteram sua linha de execução, caracterizando, assim, um comportamento autônomo e adaptativo.

Em contraste, a segunda linha de pesquisa enfatiza agentes de software capazes de realizar tarefas bem definidas, de forma automática, sem a utilização de conceitos da área de Inteligência Artificial como, por exemplo, software automático (robôs) de busca por conteúdos específicos na *Web* e agentes administradores de caixas postais, sentinelas de pontos de rede, entre outros.

Apesar de ser um conceito já razoavelmente difundido, e empregado em diversas áreas de pesquisa da Ciência da Computação, a definição de agentes de software ainda não é clara e varia, dependendo da área na qual se aplica. A proliferação de agentes de software em áreas como Interface com Usuário, Banco de Dados, Redes, Computação Gráfica, e Engenharia de Software provocou a utilização deliberada do termo "agente".

Alguns programas são denominados "agentes" simplesmente por:

- terem a capacidade de executar remotamente em outras máquinas,
- informarem ao usuário quando o sistema de arquivos está com problemas,
- informarem problemas de comunicação em nós de uma LAN (Local Area Network); e
- filtrarem informações desejadas na *Web*, aliviando a sobrecarga de informação.

Segundo Franklin [Fra96], as principais características da maioria dos agentes são:

- **autonomia** - os agentes funcionam sem a intervenção direta de operadores de qualquer tipo e possuem algum tipo de controle sobre suas ações e seus estados internos;
- **pró-atividade** - os agentes pró-ativos tomam a iniciativa para atingir os seus objetivos, não se limitando a responder estímulos do ambiente;
- **reatividade** - os agentes percebem seu ambiente, que pode ser o mundo físico, um usuário através de uma interface gráfica, uma coleção de outros agentes, a Internet ou tudo isto combinado, e respondem aos estímulos recebidos deste ambiente;
- **continuidade temporal** - os agentes são processos em execução contínua, que tanto podem estar ativos, quanto adormecidos;
- **mobilidade** - os agentes, ao longo do tempo, estão habilitados a movimentarem-se em uma rede, ocupando seus diferentes nodos;
- **benevolência** - os agentes não devem possuir objetivos conflitantes. Cada agente irá sempre tentar fazer exatamente o que lhe for solicitado;
- **racionalidade** - os agentes, dentro do alcance de suas crenças, agirão de forma a atingir seus objetivos e nunca contra os mesmos;

- **adaptabilidade** - os agentes devem adaptar-se aos hábitos, métodos de trabalho e preferências de seus usuários; e
- **colaboração** - os agentes têm capacidade para interagir com outros agentes, alcançando os seus objetivos, através da ajuda mútua. Agentes podem trabalhar em conjunto, cooperando para atingir seus objetivos. Agentes de compra e venda podem interagir em um ambiente multiagente, com o objetivo de realizar uma negociação, interessante para ambas as partes. Agentes de busca podem cooperar trocando informações com outros agentes, via *Web,* otimizando o trabalho de busca.

Para Franklin [Fra96], autonomia, pró-atividade, reatividade e continuidade temporal são características essenciais de um agente de software. No escopo deste trabalho de pesquisa, agentes de softwares são definidos como entidades que automatizam o processo de compra e venda na *Web*, capazes de realizar tarefas específicas e bem definidas de forma autônoma, sem a necessidade de intervenção do usuário. Dentre as características apresentadas, **reatividade**, **autonomia** e **cooperação** [Fon01a, Fon01b, Bia01] foram consideradas como requisitos essenciais para a modelagem dos agentes de software .

2.4 Linguagem Java

Java é uma linguagem de programação Orientada a Objetos que está sendo utilizada pelas facilidades no desenvolvimento de aplicações que são executadas na Internet em múltiplas plataformas. A linguagem foi desenvolvida pela *Sun* [Sun01] entre os anos de 1990 e 1995. Foi a primeira linguagem que permitiu a inclusão de som e animação em uma página da *Web*. Em vez de apenas ler um artigo ou visualizar uma figura na *Web* os usuários podem, agora, interagir em tempo real com os objetos da página *Web* [Sun01].

Como uma linguagem Orientada a Objetos, Java busca facilitar a manutenção, reusabilidade e extensibilidade do sistema, tornando os programas mais fáceis de serem entendidos e lidos do que os desenvolvidos segundo a abordagem tradicional. Java integra várias tecnologias recentes como: programação concorrente, sistemas distribuídos, orientados a objetos, protocolos da Internet e outras áreas da informática.

As principais características de Java são:

- **Orientação a objetos**: praticamente, tudo em Java é uma classe, um serviço ou um objeto;
- **Independência de plataforma:** o programa Java é compilado em um formato que pode ser lido e executado por interpretadores em diferentes plataformas como Windows 95, Windows NT e Solaris;
- **Segurança**: como foi desenvolvida depois do fenômeno do vírus de computador, Java incorpora elementos de segurança que proíbem que programas elaborados nessa linguagem introduzam vírus, apaguem ou modifiquem arquivos; e
- **Multitarefa:** um simples programa em Java pode ter muitas tarefas sendo processadas independente e continuamente.

Na Figura 1, apresenta-se um diagrama com os componentes Java representados por blocos e as suas respectivas ligações representadas por setas.

Durante a compilação e execução de um programa Java, o arquivo fonte (<*arq*>.*java*) é submetido ao compilador Java, o **javac**. Este compilador gera um arquivo (<*arq*>.*class*) em *Bytecode*, que será submetido à Máquina Virtual Java (*Java Virtual Machine* - JVM). A JVM permitirá a execução do programa em diferentes plataformas por estar baseada no *Bytecode*, o que torna a linguagem portável e com uma arquitetura neutra. Outras linguagens, como C++ e Pascal, estão presas a um conjunto de instruções nativas de uma plataforma de hardware e software específica.

Ligado diretamente à JVM pode-se observar o Compilador JIT (*Just In Time*). O JIT realiza a compilação do <arq.class> em *Bytecodes*, gerando o código de máquina nativo e específico de cada plataforma. Bibliotecas específicas da plataforma atual são utilizadas na execução do programa. Isso oferece uma melhor performance para a JVM durante a execução dos programas.

Outro componente oferecido pela linguagem Java é um conjunto padronizado de bibliotecas de classes, ligado a JVM, que suporta um determinado serviço como:

- criação de GUI (*Graphics User Interface*);
- controle de dados multimídia, e;
- comunicação em rede.

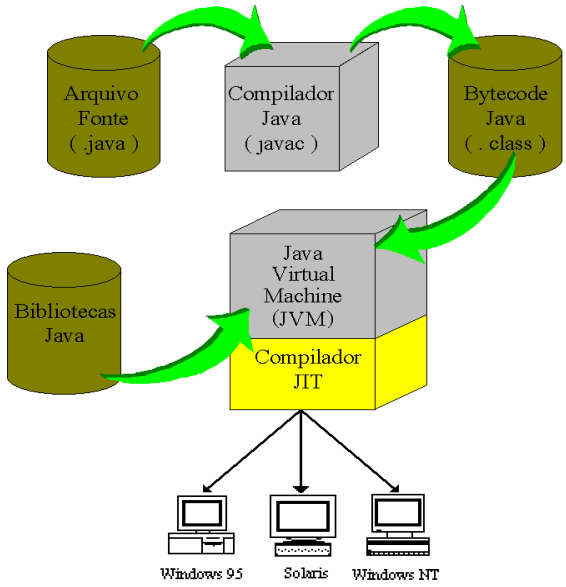


Figura 1- Componentes Java

A linguagem Java é capaz de gerar dois tipos de programas: *Applets* e Aplicações. *Applets* são programas que são inseridos em uma página *Web* para serem executados após o início de um ou vários eventos como o de pressionar o teclado e o botão do mouse. Em uma página da *Web* pode haver vários *applets* que são executados independentemente um do outro, ou ainda, ocorrer uma interação entre esses *applets* (troca de mensagens, por exemplo).

Os *applets* necessitam da utilização de *browsers* que são softwares que permitem sua apresentação, por exemplo como o Netscape. Para que o *browser* execute um *applet*, é necessário "chamar" este programa através de um documento HTML. Uma Aplicação não conta com todos os recursos de um *applet* e também não precisa de um *browser* para ser executado, pois a própria Máquina Virtual Java executa-o diretamente.

Nesta pesquisa, a implementação do sistema utilizará um dos padrões específicos de Java, o *Java Servlets* [Hun99]. A próxima seção apresenta os principais conceitos de *servlets*.

2.4.1 Java Servlets

Java *Servlet* é uma extensão da plataforma Java que permite aos desenvolvedores adicionarem, nos servidores *Web*, mecanismos que estendam sua funcionalidade [Sun01]. Os *Servlets* substituem com grandes vantagens as atuais aplicações CGI utilizadas na maioria dos servidores *Web* [Mos99]. Programas CGI são comumente implementados para controlar a interação do usuário com o servidor de rede.

Algumas vantagens dos *Servlets* sobre as aplicações CGI são destacadas:

- ***performance:*** são carregados apenas uma vez no Servidor, e para cada nova requisição simultânea, é gerado um novo *thread* no *Servlet*. O método `init()` do *Servlet* é invocado apenas na primeira vez que a classe é carregada. As aplicações baseadas em programas CGI carregam um novo programa para cada requisição no Servidor. Assim se ocorrem 50 requisições simultâneas, 50 programas iguais serão carregados na memória;
- ***independência de plataforma:*** podem ser executados em qualquer plataforma sem serem reescritos e até mesmo compilados novamente [Mos99],e;
- ***geração dinâmica de páginas HTML:*** podem ser instalados em servidores *Web* para processar informações transmitidas via HTTP a partir de formulários HTML. As aplicações podem incluir acesso a banco de dados ou comunicação com outros *Servlets* [Mos99].

A Figura 2 mostra em alto nível o fluxo do processo da aplicação Cliente para o Servlet:

1. O Cliente envia uma requisição para o servidor;
2. O servidor envia a requisição para o Servlet;
3. O *Servlet* constrói a resposta e retorna para o servidor. Esta resposta é construída dinamicamente e é dependente da requisição do cliente;
4. O servidor envia a resposta para o Cliente.

Para criar um *Servlet*, utiliza-se a API `javax.servlet` para definir a interface *Servlet* que apresenta todos os métodos para estabelecer a comunicação com o Cliente. Todo *Servlet* deve ser uma implementação desta interface, ou ser uma subclasse de uma classe implementada. Os

Servlets são instalados em servidores *Web*, utilizando o protocolo HTTP para enviar e receber informações. A classe que implementa a interface *Servlet* é a `HttpServlet`, a qual apresenta os métodos para tratar os vários tipos de requisições (`POST, GET, PUT, DELETE`, e outros).

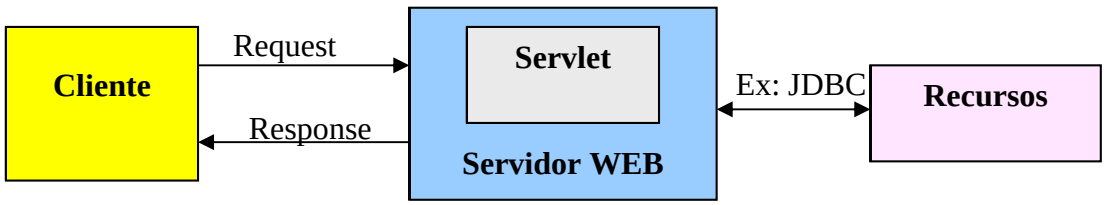


Figura 2 – Fluxo do Processo do Cliente para o Servlet

A comunicação entre um *Servlet* e um Cliente HTML é feita por dois objetos instanciados das classes: o **`ServletRequest`** e o **`ServletResponse`.** O `ServletRequest` encapsula as funções de comunicação do Cliente no Servidor, permitindo que o *Servlet* receba dados como o conteúdo de um formulário HTML. O `ServletResponse` encapsula os dados do Servidor no Cliente, manipula o protocolo HTTP e especifica informações de cabeçalho da conexão. A interface `HttpServletRequest` é uma extensão da interface `ServletRequest` e o `HttpServletResponse` é uma extensão da interface `ServletResponse`.

O ciclo de vida do *Servlet* consiste de três métodos principais: `init(), service()` `e destroy().`

O método `init()` inicializa o *Servlet* e os logs de inicialização. Nele ocorrem a leitura de configurações estabelecidas pelo sistema, a inicialização de estruturas de dados e, se necessário, estabelece a conexão com um banco de dados, que será utilizado pelo *Servlet* durante a sua execução. Este método é chamado apenas uma vez, ficando alocado na memória do Servidor, e não será executado novamente a menos que o *Servlet* seja recarregado.

Após o término da inicialização, o *Servlet* está apto a receber e processar requisições de acesso de Clientes. O processamento dos pedidos ocorre no método `service().` O método `service()` recebe informações do Cliente, realiza o processamento e envia a resposta para o Cliente. Para cada requisição simultânea, o *Servlet* cria um *thread*, automaticamente para executar o método `service().` Isto exige alguns cuidados para garantir que o *Servlet* será

executado corretamente com vários *thread* simultâneos. Por exemplo, se o Cliente irá atualizar uma variável compartilhada, esta variável deve ser sincronizada para garantir que não ocorrerá a interferência entre os *threads*.

O método `destroy()` é chamado antes que o *Servlet* seja finalizado a fim de possibilitar a liberação dos recursos que estão alocados ao *Servlet*.

2.4.2 Enterprise JavaBeans

Enterprise JavaBeans (EJB) é uma tecnologia que facilita o desenvolvimento de aplicações *multi-tier*, utilizando especificação de modelos de componentes orientados à transação e baseados em servidores escritos em Java [Sun01]. Com a tecnologia EJB, os componentes que implementam as regras lógicas do negócio são disponibilizados no servidor EJB. O servidor EJB tem um sistema próprio de execução de componentes, que manipula as complexidades do sistema, como: execuções de *threads*, transações, gerenciamento de estado e recursos compartilhados.

Para a construção de um componente EJB as seguintes características são consideradas:

1. *Ambiente de execução EJB*

   A especificação de um componente EJB é armazenada em um "*container*". O *container* é responsável em fornecer aos EJBs serviços de: suporte às transações, suporte à persistência e gerenciamento de múltiplas instâncias de um dado *bean*. Depois de compilado um EJB é executado em qualquer plataforma.

2. *Camada Middle-Tier*

   Enterprise JavaBeans, como outros componentes *Middle-Tier*, são usados para encapsular regras de negócio. Um EJB típico consiste em métodos que encapsulam as regras de negócio.

3. *Processamento de transações*

   Um *container* EJB suporta transações, cujo tipo é especificado na criação do *bean*. Quando o *bean* é executado, o *container* lê essas informações e fornece o suporte necessário.

Os tipos possíveis de transação são as que requerem que o:

- Cliente deixe a transação aberta;
- *Container* comece uma nova transação quando o *bean* é chamado; e
- *Bean* gerencie sua própria transação.

4. *Estado do bean*

Muitas aplicações para a Internet são desenvolvidas utilizando *Common Gateway Interface* (CGI), por outro lado, podem manter o estado com várias chamadas a métodos, como ocorre com as aplicações *Web*. O *container* EJB mantém-se atualizado das mudanças de estado do *bean*.

5. *Responsabilidades do container EJB para:*

- Suportar a execução concorrente de vários *beans*;
- Manter-se informado de seus estados; e
- Fornecer suporte a transação.

Um sistema EJB é dividido em três camadas lógicas: o Cliente, o Servidor EJB, e a Base de Dados ou outro mecanismo de persistência.

Esta é uma arquitetura lógica e não física, pois as três camadas podem ou não residir na mesma máquina. Por exemplo:

- O Servidor EJB e a Base de Dados podem residir na mesma máquina. Este caso pode acontecer se o Servidor EJB embutir em sua funcionalidade a persistência de dados.
- O Cliente e o Servidor EJB podem residir na mesma máquina. Este caso ocorre quando um *bean* EJB faz uma chamada a outro *bean* EJB no mesmo *container*.
- Combinando os dois casos acima, as três camadas podem estar no mesmo computador.

A Figura 3 mostra a arquitetura EJB, na qual cada uma das camadas Cliente, Servidor EJB e Base de Dados, residem em uma máquina diferente.

Os papéis do EJB em cada uma dessas camadas são:

- Um programa Cliente faz uma chamada ao EJB remoto. Na camada Cliente, o programa sabe como encontrar o Servidor EJB e como interagir com o objeto que reside no Servidor EJB;
- Os componentes EJB ficam na camada intermediária (*Middle Tier)*, dentro de um *container*, que reside no **Servidor EJB** e;
- A Base de Dados reside na terceira camada. *Beans* EJB podem acessar diretamente a Base de Dados, via *Java Database Connectivity* (JDBC), ou indiretamente pelo *container*.

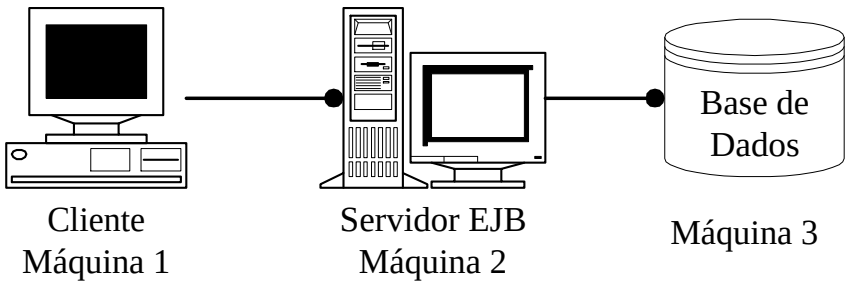


Figura 3 – Arquitetura Física do Sistema EJB

A Figura 4 mostra, em alto nível, a arquitetura do EJB onde o:

- *Bean* EJB reside no *container*;
- Cliente nunca se comunica diretamente com o *bean* EJB, ele se comunica pelas *Interfaces Home* e *Remote*.
- Servidor EJB manipula as requisições do *container* que passa para o Cliente.

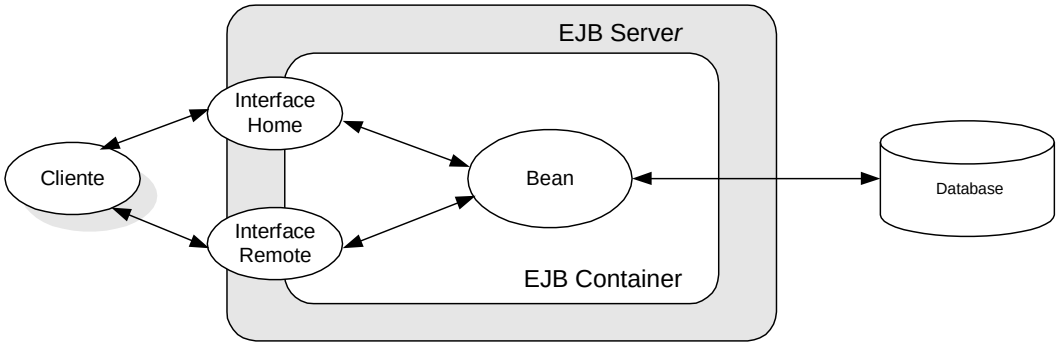


Figura 4 – Arquitetura EJB

O **Servidor EJB** fornece *containers* com serviços de baixo nível, como por exemplo, a conexão com a rede.

O **Container EJB** é o ambiente em que o EJB é executado. Serve, principalmente, como *buffer* entre um EJB e o mundo real. Os Clientes não se conectam diretamente ao *bean* EJB, e sim, por intermédio das interfaces do *container* o qual passa as requisições do Cliente para o *bean*. O *container* também disponibiliza os *bean*s e seus serviços de suporte: às transações, ao gerenciamento de múltiplas instâncias, à persistência e à segurança. O *Suporte às Transações* é possível, pois os *containers* EJB fornecem aos *beans* EJB as propriedades Atomicidade, Consistência, Isolamento e Durabilidade (ACID). As transações em outros ambientes são limitadas por "*begin"* e "*commit*". O "*begin*" notifica o gerente da transação que a mesma já começou, e o "*commit*" ordena que o gerente da transação persista os dados. Em outra abordagem para gerenciamento de transação chamada *Gerenciamento de Transação Declarativa***,** o desenvolvedor do *bean* pode especificar o tipo de transação que o *bean* necessita. Quando o *bean* é executado, o *container* lê o *Deployment Descriptor* associado ao *bean* EJB e, automaticamente, fornece o suporte à transação. EJB fornece seis possíveis modos de gerenciamento de transação:

- TX_NOT_SUPPORTED, notifica que o *bean* não suporta nenhum tipo de transação e não pode ser usado em uma transação;
- TX_BEAN_MANAGED, notifica que o *bean* gerencia sua própria transação. Este gerenciamento é usado quando o *bean* pode manipular diretamente sua transação;

- TX_REQUIRED, notifica que quando o Cliente tem uma transação aberta, deve chamar um método do *bean*, sendo este executado dentro da transação do Cliente;
- TX_SUPPORTS, notifica que se o Cliente tem uma transação em progresso, quando ele invoca um método do *bean*, a transação corrente é usada, não sendo necessário criar uma nova transação;
- TX_REQUIRES_NEW, notifica que o *bean* deve sempre começar uma nova transação; e
- TX_MANDATORY, notifica que o *bean* necessita que o Cliente tenha uma transação aberta antes de usá-lo; e caso a transação não esteja disponível, ocorre um erro. O *Suporte ao Gerenciamento de Múltiplas Instâncias* é implementado descrevendo as classes EJB como uma única *thread*, sendo acessada por um único Cliente. *Containers* EJB também fornecem *suporte à persistência* de *beans*. A responsabilidade do gerenciamento de transação, concorrência, segurança e persistência pelo *container* liberam o desenvolvedor do *bean* para concentrar nas regras de negócio de sua aplicação não se preocupando com estes detalhes operacionais. No ***Suporte a Segurança*** do EJB, o principal mecanismo usado é o Acesso à Lista de Controle (*Access Control List* – ACL). ACL é uma lista de pessoas ou grupos autorizados a acessar partes específicas da funcionalidade de um *bean*. O desenvolvedor pode especificar ACLs para um *bean* como um todo, ou para qualquer método do *bean*. Somente usuários incluídos na ACL podem acessar o *bean* `ACL-PROTECTED`.

A implementação de um *bean* EJB deve fornecer uma **Interface *Home*** e uma **Interface *Remote*** para o *bean*. O *container* é responsável por gerar a implementação dessas interfaces.

A **Interface *Home*** serve como ponto inicial de contato para um Cliente. Quando o Cliente procura um *bean* EJB, o serviço *naming service* retorna uma referência para o objeto que implementa a Interface *Home* do *bean*.

A **Interface *Remote*** estende a Interface *EJBObject*, esta interface estende a interface *Remote* do pacote RMI. Para criar uma interface *Remote* de um *bean EJB*, cria-se uma interface que herde a interface *EJBObject*, adicionando as assinaturas dos métodos da regra de negócio do bean.

A **Interface EJB** é o ponto comum aos *beans* que associam os atributos de transação e pode ser do tipo *SessionBean* ou *EntityBean*.

A Interface **SessionBean** tem seu tempo de vida de acordo com a sessão do Cliente. Normalmente de vida curta. Os *bean*s do tipo *Session* podem ser **Statefull** ou **Stateless.** *Statefull* é um *bean* que muda de estado durante a execução, de acordo com as suas variáveis de instância. ***Stateless*** é um *bean* que não mantém qualquer informação de um método que chama outro, e não possui variável de instância. Todos os métodos declarados nos *beans Sessions* são métodos **callback**, que são chamados pelo *container* para notificar ao EJB *Session* sobre a ocorrência de algum evento.

Uma Interface *Home* do *bean Session* deve fornecer um ou mais métodos `create()`, para criação de instâncias de objetos EJB. Um método `create()` pode receber quaisquer argumentos aceitos pelo *Remote Method Invocation* (RMI) [Sun01], mas precisa retornar um tipo que deve ser encontrado na interface *remote* do seu objeto EJB. RMI é um pacote em Java que contém classes para chamada de métodos remotos, na arquitetura *Client/Server*. Este pacote foi utilizado no EJB para tornar mais simples e transparente a distribuição de objetos em um programa Java. Um Cliente RMI interage com objetos no Servidor através da Interface *Remote* do objeto servidor. Utiliza-se um *stub* para chamar um método da classe remota. Este *stub* é automaticamente gerado pelo EJB.

Quando um Cliente chama o método `create()`, na Interface *Home* do *bean*, o *container* cria um objeto e, então, chama um método da classe EJB chamado `ejbCreate()`. O desenvolvedor tem de implementar o método `ejbCreate()`, na sua classe, para cada método `create()` que foi declarado na Interface *Home*. Um método `ejbCreate()`deve ter um tipo de retorno *void*, e argumentos correspondentes ao do método `create()`declarado na Interface *Home*.

Um *bean Session* implementa a Interface **SessionSynchronization** para gerenciar sua transação, caso não implemente a Interface *UserTransaction*. Todos os métodos da interface *SessionSynchronization* são *callbacks*.

As Interfaces **EntityBeans** implementam *beans* de vida longa, que existem nas sessões dos Clientes, são compartilhados por vários Clientes e permanecem ativos por tempo

indeterminado. Tipicamente, *beans Entity* permanecem em depósitos de dados persistentes. Assim como um *bean Session* deve implementar a interface *SessionBean*, um *bean Entity* deve implementar a Interface *EntityBean*.

Beans Entity são associados a uma tabela na Base de Dados. O estado interno de um *EJB Entity* ativo deve ser sincronizado com seus dados depois que uma transação é concluída. A especificação EJB suporta duas abordagens para gerenciar esta sincronização: Persistência ***Bean-Managed*** e ***Container-Managed***. Na persistência *Bean-Managed*, o desenvolvedor do *bean* deve escrever a base de dados necessária para armazenar e acessar objetos persistentes. Para implementar persistência *Bean-Managed* usam-se os métodos: `ejbLoad()` e `ejbStore()`. Na persistência *Container-Managed*, não é necessário sincronizar a base de dados. No *Deployment Descriptor* do *bean Entity*, especificam-se os campos que o *container* precisa gerenciar.

Na arquitetura EJB, o Cliente compromete-se com as tarefas de encontrar, acessar e destruir o *bean.*

Clientes EJB, implementados em RMI, utilizam a API JNDI para localizar a Interface *Home* do *bean*. Os métodos `find()` ou `create()` da Interface *Home* do *bean* retornam uma referência à Interface *Remote* do *bean*.

Mais detalhes sobre a tecnologia EJB podem ser obtidos em Roman [Rom99].

2.5 Wireless Application Protocol (WAP)

O protocolo de comunicação móvel, *Wireless Application Protocol* (WAP) [Wap01], foi desenvolvido pelo consórcio firmado entre quatro companhias: Nokia, Ericcson, Motorola e a Unwired Planet, atualmente conhecida como *Phone.com*. O objetivo deste consórcio é a convergência da Internet com a telefonia celular, fornecendo especificações para o desenvolvimento de aplicações e serviços que operam sobre redes de comunicação sem fio, permitindo, assim, que aparelhos portáteis como telefones móveis, *pagers*, *PDAs (Personal Digital Assistant)* e outros, tenham acesso à Internet. Conforme mostra a Figura 5, o modelo WAP é similar ao modelo *Web*, onde as aplicações e os seus conteúdos são endereçados através

de uma *Uniform Resource Locator* (URL), como nos protocolos para *Internet*. O usuário solicita uma URL através de um dispositivo móvel (1); o dispositivo envia a URL para um gateway WAP ou servidor WAP (2), o gateway WAP converte a solicitação WAP para uma série de pacotes HTTP (*Hypertext Transfer Protocol)*, e os envia para um servidor *Web* (3). Se a URL solicita um arquivo, o servidor Web recupera-o e adiciona um cabeçalho HTTP. Caso a URL especifique uma aplicação CGI (*Common Gateway Interface), Java Servlet* ou qualquer outro tipo, esta é executada pelo servidor *Web* (4), em seguida, o servidor *Web* retorna uma resposta em WML (*Wireless Markup Language)* [Wap01] (5), o *gateway* ou servidor WAP verifica o cabeçalho HTTP e o conteúdo da resposta, convertendo-a para WAP e enviando-a para o seu destinatário (6), e finalmente, o dispositivo móvel recebe a resposta de sua requisição, e o conteúdo é apresentado no visor para o usuário (7).

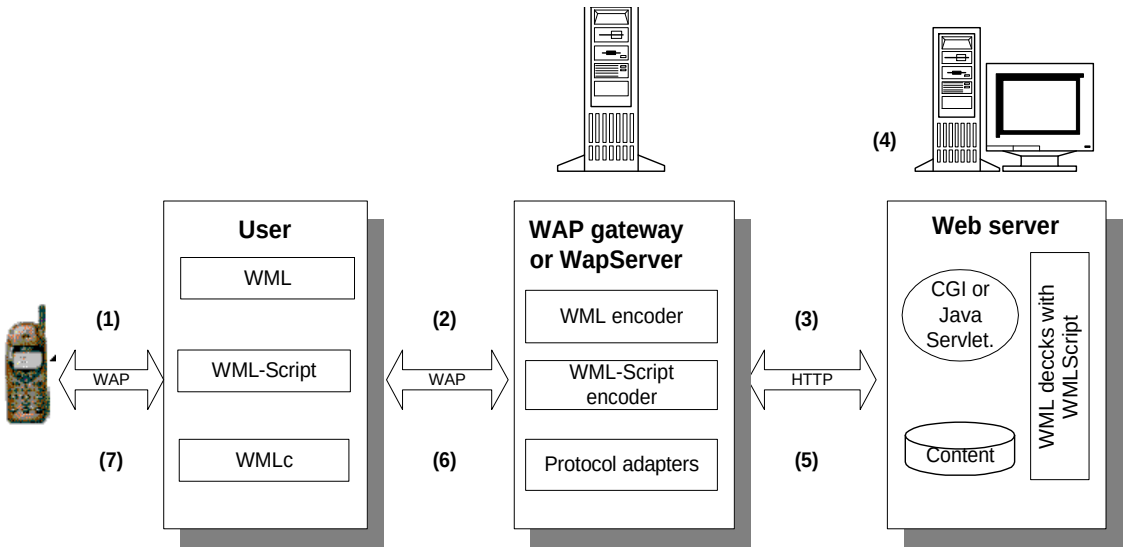


Figura 5– Modelo de Funcionamento Wap

WML é a linguagem de *markup* desenvolvida pelo consórcio WAP, que utiliza o padrão XML [W3c97] (*eXtensible Markup Language*). A linguagem WML suporta a criação de páginas dinâmicas ou estáticas, utilizadas em pequenos dispositivos para comunicação móvel, tais como celulares digitais e *pagers*. A principal característica de WML é o suporte a textos, imagens e comandos para formatação de páginas. Além disso, estas informações podem ser agrupadas em cartões WML (*cards*), que por sua vez, são organizados sob a forma de *decks*.

Um *deck* WML é similar a uma página e pode ser identificado através de uma URL, de tal modo que torna possível à navegação através de outros cartões e *decks.* Cartões *e decks* podem ser referenciados em *scripts* e *links* similares aos utilizados em HTML. WML suporta *scripts* desenvolvidos na linguagem WMLScript, de maneira similar aos *scripts* desenvolvidos em *JavaScript* inseridos em páginas HTML.

A Figura 6 mostra um simples *deck* WML, que contém um único cartão e o seu resultado. A primeira linha especifica o número da versão do XML; a segunda linha especifica o *Standardized Generalized Markup Language* (SGML); a terceira linha especifica o endereço onde se localiza a definição da estrutura de documento denominado, DTD XML (*Document Type Definition*) para documentos WML; a quarta linha indica o início do documento WML; as linhas seguintes, definem o conteúdo do cartão que será mostrado ao usuário, com *tags* de início e de fim (<card> e </card>).

```
<?xml version="1.0"?>                                          <!-- 01 -->
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"            <!-- 02 -->
"http://www.wapforum.org/DTD/wml_1.1.xml">                     <!-- 03 -->
<wml>                                                          <!-- 04 -->
        <card id="card1" title="FCEMAS via WAP">               <!-- 05 -->
                <p>                                            <!-- 06 -->
                   Id:     <input name="Id" title="Id" />       <!-- 07 -->
                   Login: <input type="login"                  <!-- 08 -->
                                 title = "Login"               <!-- 09 -->
                                 name="login" format="M*M"/>   <!-- 10 -->
                </p>                                           <!-- 11 -->
        </card>                                                <!-- 12 -->
</wml>                                                         <!-- 13 -->
```

Figura 6– Exemplo de *Wireless Markup Language* (WML)

2.6 Comércio Eletrônico Mediado por Agentes de Software

A capacidade de automatizar tarefas diversas, incluindo as etapas do processo de negociação entre compradores e vendedores, faz com que a tecnologia de agentes de software seja propícia às aplicações de comércio eletrônico [Gut98].

Apesar de possuírem muitas diferenças, os diversos mecanismos de negociação utilizados em aplicações de comércio eletrônico possuem um subconjunto das etapas identificadas como

fundamentais no Modelo de Comportamento de Compra do Consumidor (*Consumer Buying Behavior*), definidas por Maes [Mae99]:

- **identificação da necessidade** – este estágio caracteriza-se pela identificação da necessidade de consumo do usuário comprador. Sistemas com essa característica, nesta etapa, disponibilizam informações sobre produtos para estimular o consumo;
- **procura por produto** – nesta etapa, o sistema provê informações diversas, ajudando o consumidor a avaliar a melhor escolha com relação à variedade de produtos disponíveis para sua necessidade de consumo. Como resultado dessa etapa, o consumidor define o subconjunto de produtos que satisfazem sua necessidade;
- **procura por vendedor** – utilizando o conjunto de produtos definidos como satisfatório pelo consumidor, na etapa anterior, o sistema deve auxiliar a busca pelo vendedor mais adequado utilizando informações específicas dos vendedores. Baseando-se em critérios definidos pelo consumidor, como por exemplo, prazo de entrega, preço, reputação e forma de pagamento, o sistema deve ajudar a identificar o conjunto de vendedores aptos a vender o produto desejado;
- **negociação** – essa etapa determina os termos da transação comercial. O processo de negociação varia em duração e complexidade, dependendo do mercado em questão. A negociação pode ser multidimensional, ou seja, pode levar em consideração não somente o preço, mas também outras dimensões de importância para o consumidor, como prazo de entrega e forma de pagamento;
- **compra e venda** – nessa etapa, o consumidor realiza, efetivamente, o pagamento pelo produto adquirido. A entrega deste produto deve ser realizada sinalizando o término de uma negociação bem sucedida; e
- **avaliação** – sendo essa a última etapa, a avaliação consiste no retorno de informação do consumidor ou do vendedor avaliando todo o processo de negociação. Com essa avaliação, é possível classificar vendedores e consumidores por reputação e qualidade de serviço, o que pode ser utilizado também como uma nova dimensão de negociação.

Dentre os mecanismos, usados para automatizar essas etapas da negociação, destacam-se os agentes de softwares [Mae99], que proporcionam:

- **diminuição da sobrecarga de informação** - o elevado número de informações disponíveis na Internet prejudica o processo de decisão. A utilização de agentes permite a criação de um filtro automático de informações que deixa passar apenas as informações mais relevantes para o usuário [Gut98]; e
- **diminuição da necessidade de interação do usuário com o sistema** - numa negociação, o agente de software pode aliviar o usuário de tarefas que exigiriam sua interação com o sistema, tornando o processo mais rápido e independente;
    - redução de tempo - o tempo gasto para a realização de transações comerciais pode ser drasticamente reduzido com a utilização dos agentes de software, que realizam as tarefas de procura por vendedor, produto e negociação; e
    - comodidade para o usuário - a possibilidade de delegar tarefas para agentes de software, permite a criação de aplicações, que oferecem um nível maior de conforto e comodidade para o usuário.

Além dessas vantagens, inerentes às aplicações de comércio eletrônico, os agentes de software são indicados para aplicações em que o usuário tem pouca interação com o sistema, devido à falta de recursos tecnológicos ou mesmo de tempo, como no caso das aplicações que utilizam a *Internet* móvel [Fon01a]. Com a possibilidade de se ter acesso à *Internet*, por meio de dispositivos móveis como celulares e PDAs (*Personal Digital Assistants),* as aplicações *wireless* enfrentam dois principais obstáculos:

- escassez de recursos dos dispositivos de acesso móvel - celulares e PDAs são dispositivos com limitações de memória, pouca capacidade computacional e interfaces visuais para entrada e saída de dados limitada; e
- diminuição do tempo de interação - o usuário por estar em movimento e não mais à frente de um computador pessoal, tem um tempo menor para interagir com o sistema.

Com os agentes de software podem ser criadas aplicações que reduzem o uso do dispositivo móvel, delegando grande parte das tarefas para serem executadas pelos próprios agentes de software.

2.7 Método *Catalysis*

Catalysis [Dso98] é um método recente de desenvolvimento de software Orientado a Objetos, que integra técnicas para construção de softwares baseados em componentes e *frameworks*.

O método *Catalysis* começou em 1992 como uma formalização do método OMT [Rum91] e teve influências dos métodos da segunda geração como *Fusion* [Col94] e UML [Fow97, Boo99]. *Catalysis* suporta as características das tecnologias Orientadas a Objetos recentes como Java, CORBA e DCOM [Dco96] e sua notação é baseada na *Unified Modeling Language* (UML) [Fow97, Boo99].

Catalysis tem como base os princípios: *abstração*, *precisão* e *componentes "plug-in"*. O princípio *abstração* orienta o desenvolvedor na busca dos aspectos essenciais do sistema, dispensando detalhes que não são relevantes para o contexto do sistema. O princípio *precisão* tem como objetivo descobrir erros e inconsistências na modelagem, e o princípio *componentes "plug-in"* suporta a reutilização de componentes para construir outros componentes [Fuk00].

As principais características de *Catalysis* são:

- *desenvolvimento baseado em componentes:* definem interfaces flexíveis para um conjunto de componentes de uma arquitetura, independentes da implementação;
- *integridade e precisão:* definem modelos de negócio precisos em vários níveis de abstração que garantem um refinamento explícito até o nível de implementação;
- *reutilização:* baseia-se em padrões e *frameworks*, tornando o seu processo de desenvolvimento mais claro e adaptável em muitos contextos; e
- *expansível:* define consistência, refinamento e desenvolvimento incremental entre as camadas *Domínio do Problema*, *Especificação dos Componentes* e *Projeto Interno dos Componentes*.

O processo de desenvolvimento de software em *Catalysis*, como mostra a Figura 7 , é dividido em três camadas lógicas, denominados níveis de desenvolvimento: ***Domínio do Problema***, ***Especificação dos Componentes*** e ***Projeto Interno dos Componentes***,

correspondendo às atividades tradicionais do ciclo de vida do software: Análise de Requisitos, Projeto e Implementação, que são executadas de forma incremental e evolutiva.

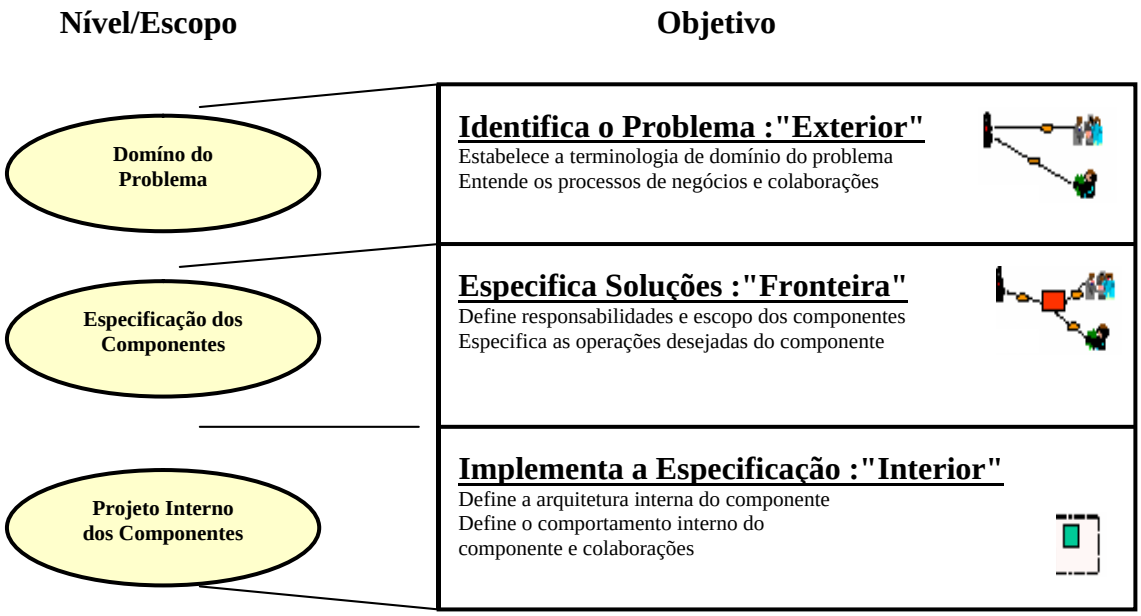


Figura 7 – Níveis de Desenvolvimento de *Catalysis*

No nível *Domínio do Problema*, é enfatizada a identificação dos requisitos do sistema no domínio do problema, especificando "o quê" o sistema deve fazer para solucionar o problema, definindo o *Modelo de Negócio*. Identificam-se os tipos de objetos e ações, agrupando-os em diferentes visões por áreas de negócio, definindo o contexto do domínio do problema. Neste nível, utilizam-se técnicas de entrevista coletiva e informal com o usuário, relatando-as com *Storyboards* e/ou *Mind maps. Storyboards* são representações de diferentes situações e cenários no domínio do problema. *Mind maps* são representações estruturadas dos termos importantes do domínio do problema, relatados na entrevista pelos usuários [Dso98, Fuk00].

Uma vez definido o contexto do domínio do problema, os requisitos coletados são transportados para o *Modelo de Colaboração* "*as-is*" e "*as-be*", representando a coleção de ações e os objetos participantes. As ações de um propósito comum são agrupadas dentro de uma colaboração. O modelo "*as-is*" descreve como é o sistema atual e o modelo "*as-be*" descreve como será o sistema proposto. Em seguida, usando o princípio do refinamento, o *Modelo de Colaboração* é mapeado para o *Modelo de Casos de Uso*, que representam os atores e suas

interações com o sistema. Ainda neste nível, pode-se utilizar *Object Constraint Language* (OCL) para detalhar as especificações sem ambigüidades e o dicionário de dados para especificar cada tipo encontrado.

No nível *Especificação dos Componentes*, são enfatizadas: a identificação, o comportamento e as responsabilidades dos componentes. Esse nível tem início com o mapeamento dos diagramas obtidos no *Domínio do Problema* para o *Modelo de Tipos* que especifica o comportamento dos objetos, mostrando os atributos e as operações dos tipos de objetos, sem se preocupar com a implementação. Refina-se o *Modelo de Tipos*, listando as ações de cada tipo. A partir do *Modelo de Tipos* e do *Modelo de Caso de Uso* especifica-se os *Modelos de Interações* que descrevem as seqüências das ações entre objetos relacionados. Os *Modelos de Interações* são representados por *Diagramas de Seqüência* e *Diagramas de Estado*. O *Diagrama de Seqüência* ressalta a seqüência de interações entre os objetos ao longo do tempo e o *Diagrama de Estado* representa uma descrição gráfica de um conjunto de estados e transições.

No nível do *Projeto Interno dos Componentes*, define-se "como" serão implementados os requisitos especificados para os componentes do sistema, preocupando-se com a distribuição física destes [Fuk00]. Neste nível, utiliza-se a técnica *Diagrama de Classes* para representar as classes com seus atributos, operações e relacionamentos, e técnicas como o *Diagrama de Componentes*, para representar a Arquitetura das plataformas Física e Lógica também são usadas nesta camada.

Catalysis apresenta dois modelos específicos para representar *frameworks*, o ***Modelo do Framework*** e o ***Modelo da Aplicação***. O *Modelo do Framework* representa a especificação estática do *framework*, identificando os tipos com os seus atributos e relacionamentos. Neste modelo são mostrados os tipos que podem ser estendidos, indicando-os com os sinais "<" e ">" (*placeholders)*, e *Modelo da Aplicação* representa a dependência dos tipos do *framework* com os tipos estendidos na aplicação.

Os níveis do processo de desenvolvimento do software em *Catalysis:* ***Domínio do Problema, Especificação dos Componentes*** e ***Projeto Interno dos Componentes***, são mostrados na Figura 13.

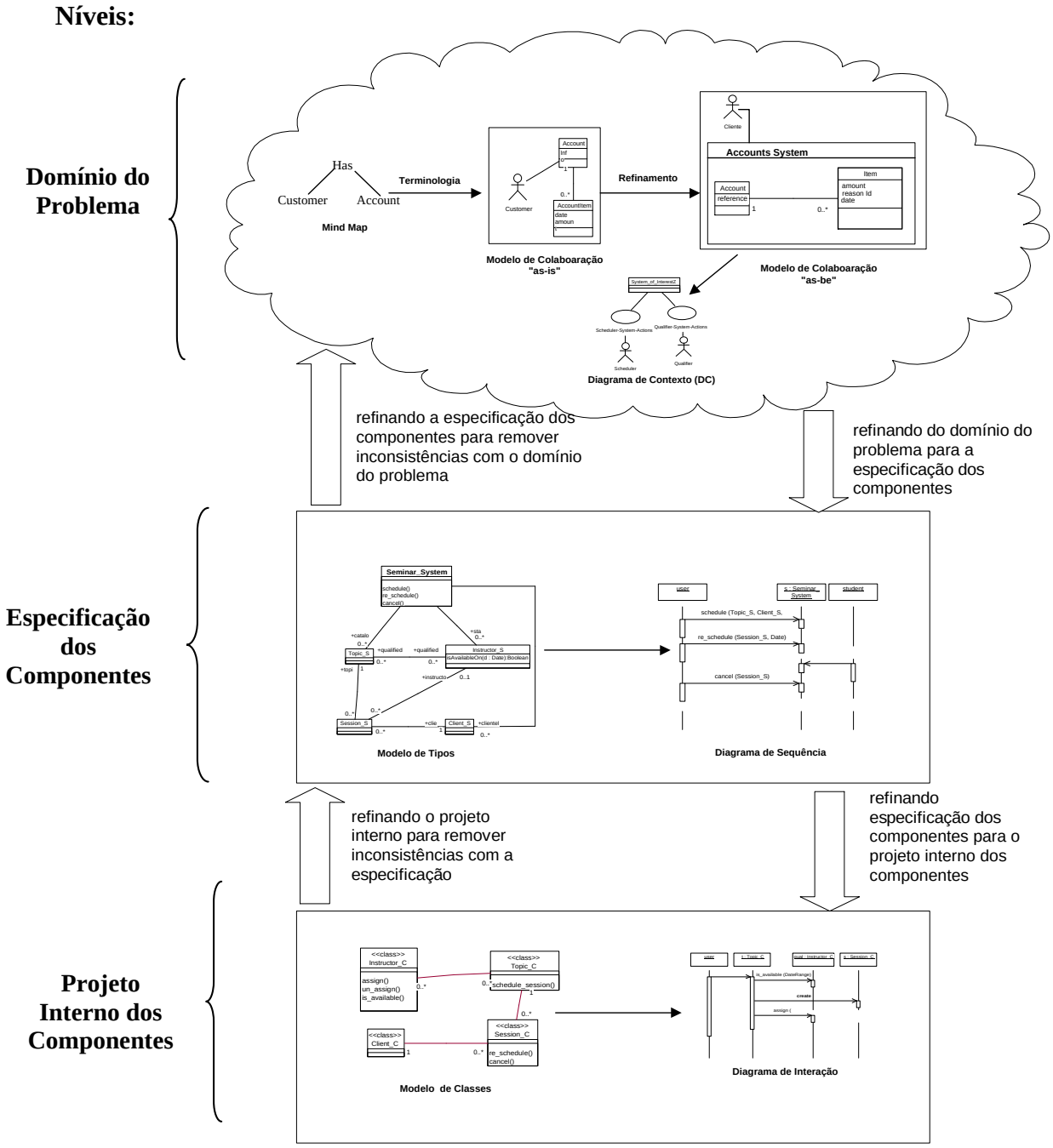


Figura 8 – Níveis do Processo de Desenvolvimento *Catalysis*

Catalysis enfatiza também, o princípio de reutilização com o uso de componentes executáveis, que são representados por interfaces polimórficas encapsuladas.

A próxima seção apresenta o *framework* para Comércio Eletrônico, via Internet Móvel, mediado por Agentes de Software, objeto deste trabalho de pesquisa.

Capítulo 3

Framework para Comércio Eletrônico, via Internet Móvel, mediado por Agente de Software

Esta pesquisa apresenta um *framework*, denominado FCEMAS, para o domínio de comércio eletrônico, via Internet Móvel, mediado por Agentes de Software, que suporta a construção de aplicações de compra e venda de produtos. O *framework* visa facilitar o processo de desenvolvimento de software, aumentando a produtividade, diminuindo os custos e tempo de construção das aplicações, dando maior flexibilidade de manutenção.

O *framework* foi modelado segundo o método *Catalysis*, e implementado na linguagem Java. Utiliza agentes de software para realizar o processo de compra e venda de produtos via Internet móvel. Para validar o *framework* foram construídos componentes de comércio eletrônico, do domínio de farmácia, que reutilizam classes do *framework*. Estudos de caso foram usados para testar o protótipo construído e fornecer "*feed-back*", para corrigir os erros, em todas as fases do ciclo de vida do *framework*.

3.1 Arquitetura do *Framework*

O *Framework* para Comércio Eletrônico, via *Internet* Móvel, mediado por Agentes de Software é um *framework* vertical do tipo *white box*.

As aplicações construídas a partir do *framework* terão uma arquitetura *Multi-Tiered*. A arquitetura *Multi-Tiered* é utilizada no desenvolvimento de sistemas distribuídos para garantir uma separação e reutilização de suas funcionalidades e facilitar a manutenção.

As aplicações construídas a partir do *framework* podem ser executadas via *Web*, em um *browser* como *Netscape ou Internet Explorer*, ou em dispositivos móveis como telefones celulares ou *Personal Digital Assitent* (PDAs). A Figura 9 mostra a arquitetura conceitual do *framework*, destacando, em sombreado, suas camadas: **Controle**, **Cadastro** e **Negociação**, **Agente**, **Persistência** e **Suporte**.

Aplicações de diferentes domínios, Farmácia, Veículos, e outras, podem ser construídas, fazendo reutilização do FCEMAS, e são executadas com diferentes sistemas operacionais, Windows NT, Linux, Unix, e outras, graças a Java Virtual Machine [Sun01].

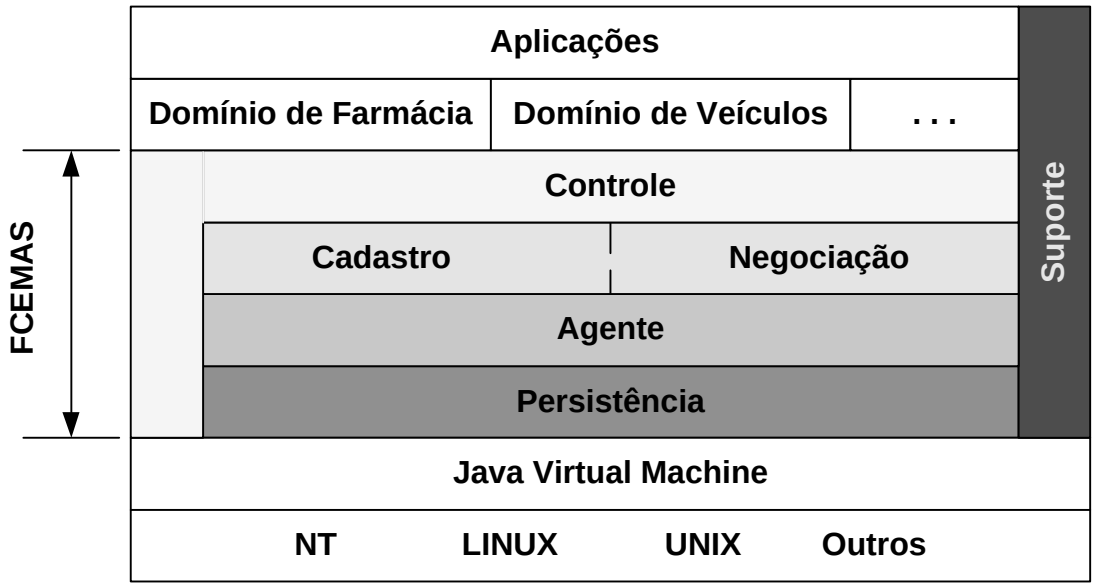


Figura 9 – Arquitetura do FCEMAS

Pela disposição das camadas pode-se ter uma idéia de suas interações. As **Aplicações** dos diferentes **Domínios,** criadas a partir do *framework*, reutilizam componentes das diferentes camadas do FCEMAS. Assim, determinada aplicação de um domínio comunica-se com camada **Controle,** que recebe as requisições da aplicação e retorna suas respostas.

A camada **Controle** é responsável por tratar as requisições das aplicações e estabelecer a comunicação com as demais camadas. A camada dividida em **Cadastro e Negociação** trata as regras gerais de negócio para as aplicações do comércio eletrônico. A camada **Agente** fornece os serviços para a realização do comércio eletrônico mediado por agentes. A camada **Persistência** realiza a persistência dos objetos em banco de dados relacional, como: Oracle, DB2, Sybase e outros. A camada **Suporte** interage com as demais camadas para fornecer serviços de apoio relacionados com o tratamento de exceção e de comunicação com os agentes de software.

Esta arquitetura assemelha-se à estrutura do *framework San Francisco* [Ibm00] da *International Business Machines* (IBM).

Para a construção do *framework*, foi definido um processo de desenvolvimento, que segue o modelo espiral do ciclo de vida de um software [Pre00], realizado em quatro passos: **Definir Domínio do Problema**, **Especificar *Framework*, Projetar *Framework* e Implementar *Framework***, conforme mostra a Figura 10.

As linhas horizontais tracejadas separam os passos segundo os níveis do método *Catalysis* [Dso98], para a construção de *frameworks:* **Domínio do Problema, Especificação do *Framework* e Projeto Interno do *Framework***

O diagrama da Figura 10 segue a notação de Ross [Ros77], conforme resume a legenda colocada à direita, onde em cada caixa tem-se uma atividade, e na face esquerda têm-se as entradas, à direita, as saídas, na parte inferior, os mecanismos de execução, e na superior, os controles que orientam a execução da atividade.

Todas as atividades são executadas pelo engenheiro de software com apoio da MVCASE [Bar99].

Parte-se dos requisitos do *framework*, e na primeira atividade define-se o domínio do problema, especificando seus requisitos. Este primeiro passo é orientado pelas técnicas de *Catalysis* e Agentes de Software.

As saídas deste passo são a entrada para o próximo passo, onde é especificado o *framework*, conforme o segundo nível de *Catalysis.* Padrões de projeto são usados para garantir maior flexibilidade e reutilização das especificações.

O *framework* especificado é então projetado, considerando agora os requisitos não funcionais, da plataforma definida para o projeto e implementação.

Finalmente faz-se a implementação do *framework* projetado na linguagem Java, conforme mostra, à direita da Figura 10, o processo de desenvolvimento é baseado no método *Catalysis* para desenvolvimento de *frameworks.*

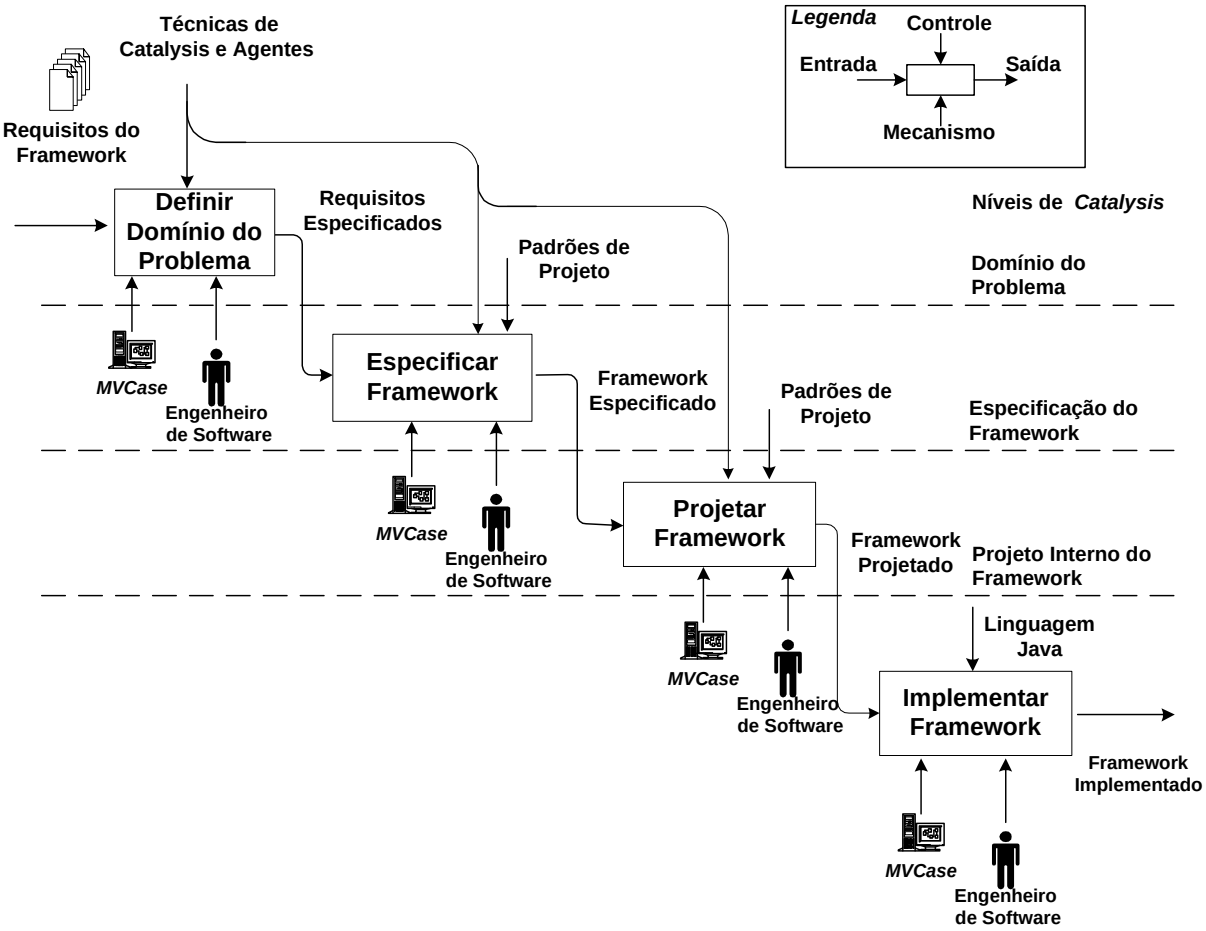


Figura 10 – Estratégia de Desenvolvimento do *Framework*

Segue-se uma apresentação de cada passo da estratégia para a construção do *Framework.*

3.2 Definir Domínio do Problema

Para definir os requisitos do *framework*, foram analisadas várias aplicações tradicionais de comércio eletrônico, como por exemplo as lojas virtuais: www.lojasamericans.com [Loj99], www.carsale.uol.com.br [Uol99], www.submarino.com.br [Sub99] e outras. Também foram analisadas aplicações de comércio eletrônico, mediado por agentes de software, como por exemplo: O Kasbah do grupo Software *Agents* do *Méd Lab* no MIT [Ame99], que tem uma arquitetura multiagente para compra e venda de livros, e o MAGMA do *Minnesota Agent Marketplace Architecture*, um sistema multiagente para comércio eletrônico desenvolvido na Universidade de Minnesota [Tsv96], para compra e venda de computadores. Estas aplicações foram analisadas para identificar as principais características das aplicações de comércio eletrônico.

Neste *primeiro passo* do processo de desenvolvimento, é enfatizado o entendimento do problema, especificando-se "o quê" o *framework* deve atender para solucionar o problema. Após o levantamento dos requisitos do sistema usando técnicas como *storyboards* ou *mind-maps*, visando representar as diferentes situações e cenários do domínio do problema, definiu-se o domínio do *framework*.

Os requisitos identificados são especificados em Modelos de Colaboração das aplicações [Boo00, Dso98, Lar99], representando a coleção de ações e os objetos participantes. Em seguida, os modelos de colaborações são refinados em Diagramas de Casos de Uso [Boo00, Dso98, Lar99].

Figura 11 mostra o ***Modelo de Colaboração*** especificado para o *framework*, onde são identificados os atores e suas principais ações. Conforme o método *Catalysis* [Dso98], os atores são representados por retângulos e suas ações por elipses. Para o FCEMAS foram identificados os atores: Comprador, Vendedor, Administrador e Aplicação. Suas principais ações são: Tratar Agente, Tratar Negociação e Gerenciar Aplicação.

Os atores Comprador e Vendedor representam os usuários das aplicações que desejam comprar ou vender produtos através da *Web*. O principal objetivo do uso dos agentes de compra e venda, nas aplicações desenvolvidas a partir do FCEMAS, é liberar o comprador das atividades

de procura por produto, vendedor e negociação atribuindo-as aos agentes. Para realizar o processo de negociação os agentes de software, de venda e de compra, comunicam-se entre si, e realizam a negociação de acordo com a estratégia selecionada pelo usuário.

O ator Administrador tem acesso às negociações realizadas pelos agentes de software, solicita relatórios de usuários que utilizam o sistema, aprova o cadastro de usuários e outras atividades pertinentes ao gerenciamento da aplicação.

O ator Aplicação representa as aplicações desenvolvidas a partir do FCEMAS.

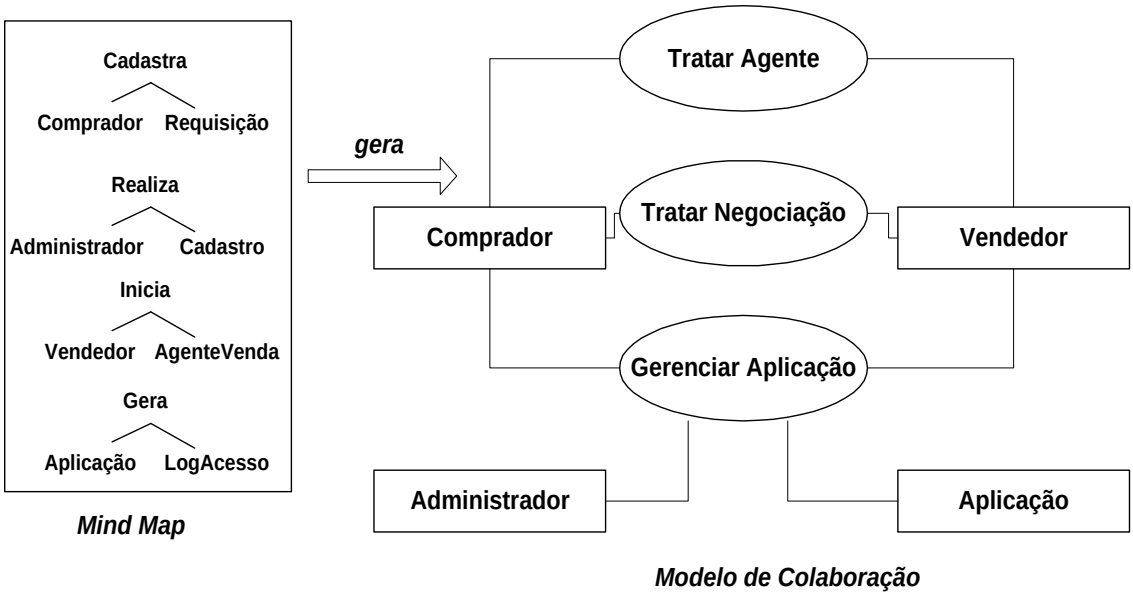


Figura 11 – Primeiro passo da Estratégia: Definir Domínio do Problema

Em seguida, o modelo de colaboração é refinado e particionado em Diagramas de Casos de Uso [Boo00, Dso98, Lar99], conforme as ações identificadas, visando diminuir a complexidade do *framework*.

A ação Tratar Agente foi refinada nos Casos de Uso: IniciarAgente, DefinirAgente, DefinirAgenteCompra, DefinirAgenteVenda, PararAgente e ConsultarAtividadesDosAgentes, como mostra a Figura 12.

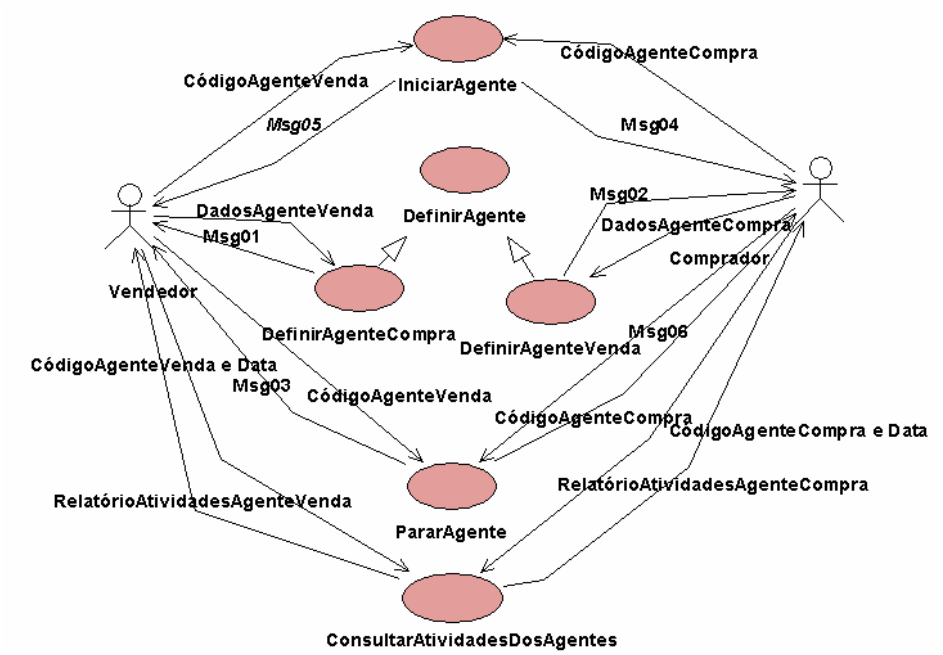


Figura 12– Diagrama de Casos de Uso – Tratar Agente

A ação Tratar Negociação foi refinada nos Casos de Uso: CadastrarOrdemCompra, ConsultarOrdemCompra, ConsultarRequisição, RealizarNegociação, CadastrarRequisição, EnviarRequisição, ReceberRequisição e TratarRequisição como mostra a Figura 13.

A ação Gerenciar Aplicação foi refinada nos Casos de Uso: CadastrarEmpresa, CadastrarGerente, CadastrarFornecedor, CadastrarProduto, CadastrarAdministrador e CadastrarCatalogo, GerarSenhaAcesso, GerarRelatórioProduto, GerarRelatórioCatálogo, GerarRelatórioLogAcesso e GerarRelatórioEmpresa, como mostra. Figura 14.

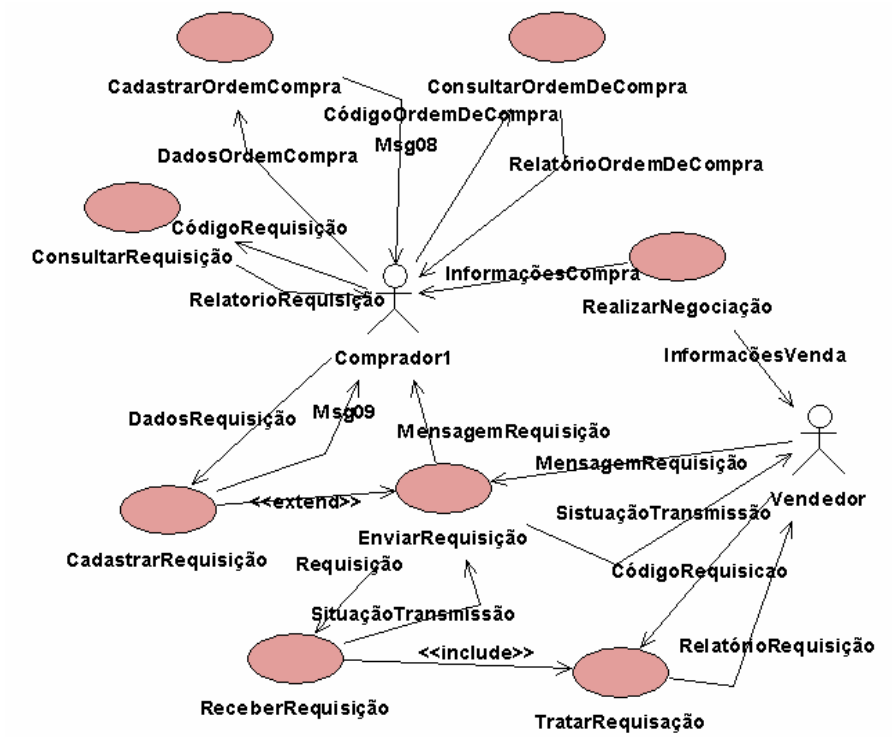


Figura 13 – Diagrama de Casos de Uso – Tratar Negociação

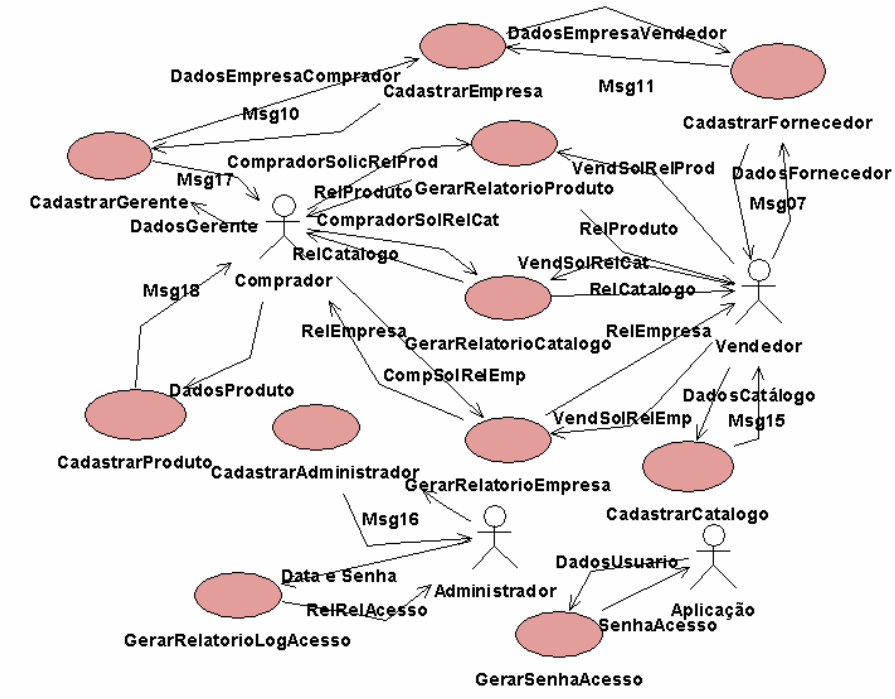


Figura 14 – Diagrama de Casos de Uso – Gerenciar Aplicação

3.3 Especificar *Framework*

Este passo corresponde ao ***segundo nível*** de *Catalysis*, onde é descrito o comportamento externo do *framework* de uma forma não ambígua. Deste modo, o engenheiro de software refina as especificações do Domínio do Problema do nível anterior, visando obter as especificações dos *framework.*

Esse passo tem início com o mapeamento das especificações dos *Mind-Maps*, Modelos de Colaboração e Casos de Uso, do passo anterior, para o *Modelo de Tipos* [Boo00, Dso98, Lar99], que especifica o comportamento dos objetos, mostrando atributos e operações dos tipos de objetos, porém sem se preocupar com a implementação. Os nomes que estão entre *placeholders* ("<" e ">"), indicam os tipos que podem ser utilizados nas aplicações desenvolvidas a partir do *Framework.*

Ainda neste passo, pode-se utilizar o dicionário de dados e a *Object Constraint Language* (OCL) [Boo00, Lar99], para detalhar as especificações.

Para o Diagrama de Casos de Uso **Tratar Cadastro**, foram identificados os tipos: `Pessoa`, `Empresa`, `Administrador`, `Gerente`, `Vendedor`, `Fornecedor`, `Produto` e `Catálogo`, como mostra a Figura 15.

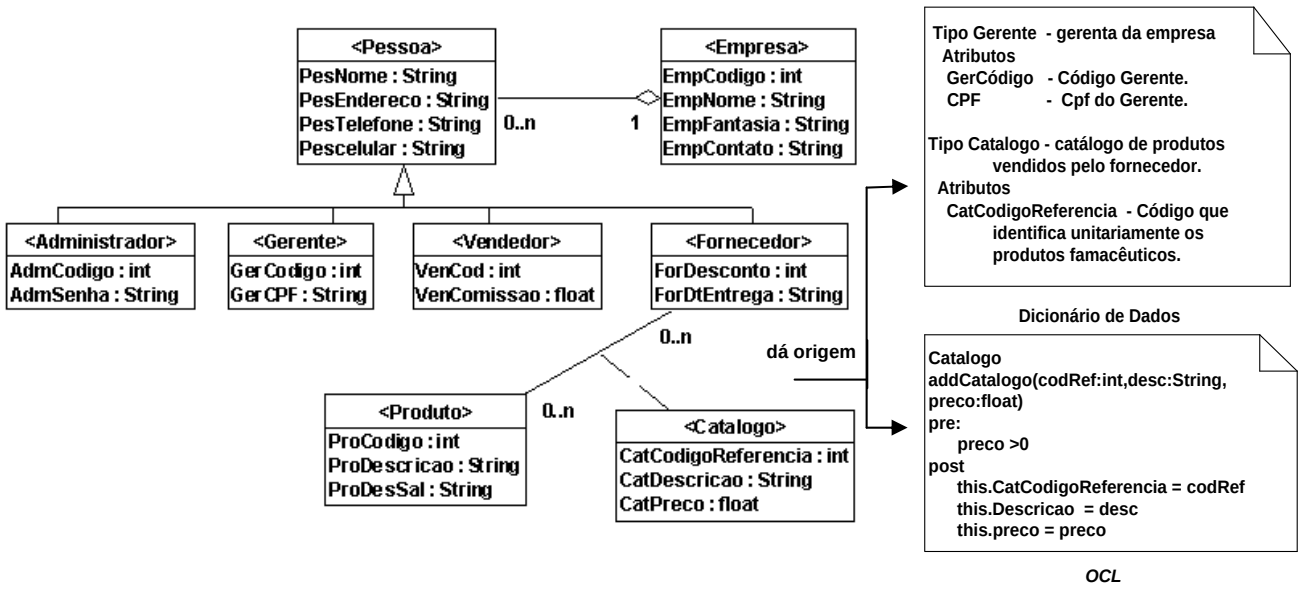


Figura 15 – Modelo de Tipos– Tratar Cadastro

Para o Diagrama de Casos de Uso **Tratar Agente** foram identificados os tipos responsáveis por realizar o processo de negociação, como mostra a Figura 16.

Os tipos `Agente`, `AgenteColaboração`, `Estado` e `Estratégia` são responsáveis por realizar o processo de negociação. Um agente pode estar em diferentes estados e utilizar diferentes estratégias de negociação.

O tipo `Agente` tem atributos básicos comuns a todos os agentes, como por exemplo, nome, código e o período para realizar as suas atividades, definidos pelos atributos data de criação e data de término.

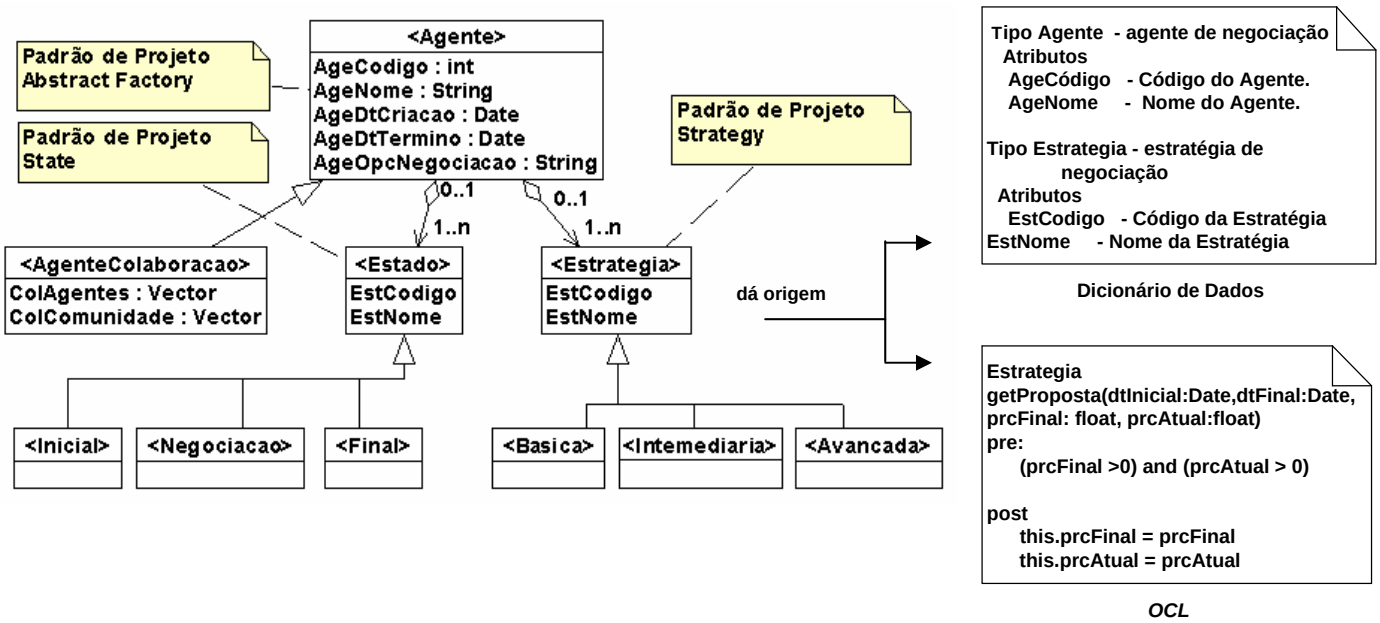


Figura 16 – Modelo de Tipos– Tratar Agente

O tipo `AgenteColaboração` é empregado em sistemas multiagentes [Fer99]. Funciona como um *blackboard* [Fer99], onde os agentes compradores e vendedores anunciam suas intenções de compra ou venda. Este tipo é um agente, com funções específicas, que suporta a comunicação entre agentes, para encontrar parceiros de negociação.

O modelo mostra que o agente pode ter diferentes estratégias de negociação. Assim, para realizar o processo de negociação pelos agentes de compra e venda foram especificadas três estratégias de negociação [Chv96], baseadas no preço do produto e no prazo estipulado para as

negociações, representadas pelos tipos `Básica`, `Intermediária` e `Avançada`, respectivamente. Na estratégia Básica, a modificação do valor inicial da negociação, em relação ao valor final, é constante ao longo do tempo, usando uma função linear. Na estratégia Intermediária, o agente de software procura manter o valor do produto negociado próximo ao valor inicial, mas em pouco tempo aproxima-se do valor final, permitindo a finalização do processo de negociação em um período de tempo menor, usando uma função quadrática. Na estratégia Avançada, o agente de software procura manter o valor inicial pelo maior tempo possível, alterando-o rapidamente, quando o período de negociação aproxima-se do final, usando uma função cúbica.

A Figura 17 mostra a variação do preço em relação ao tempo para as estratégias Básica, Intermediária e Avançada.

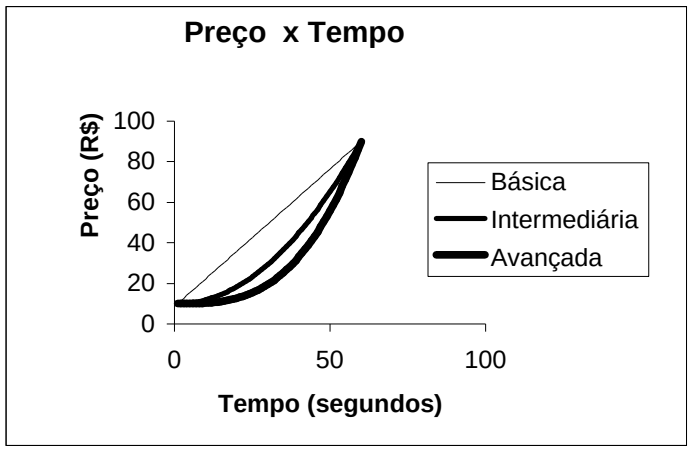


Figura 17 – Estratégias de Negociação

Para modelar as estratégias de negociação, utilizou-se o padrão de projeto *Strategy* [Gam95], que permite alterar o comportamento do agente de software de acordo com a estratégia selecionada. O uso deste padrão permite que novas estratégias de negociação possam ser adicionadas, sem alterar a estrutura dos agentes de software.

Um agente também pode estar em diferentes estados. O tipo `Estado` define o estado corrente de execução do agente de software. Para modelar os estados, utilizou-se o padrão de

projeto *State* [Gam95]. O tipo `Estado` foi especializado nos tipos `Inicial`, `Negociação`, e `Final`. No estado Inicial, o agente de software procura por possíveis agentes compradores ou vendedores. No estado Negociação, o agente comunica-se com outros agentes, ou com seu usuário, para realizar a negociação. No estado Final, tem-se a conclusão da negociação.

Para o Diagrama de Casos de Uso **Tratar Negociação** foram identificados os tipos que representam as intenções de compra e venda entre comprador e vendedor, representados pelos tipos `OrdemCompra` e `Catálogo`. O gerente elabora a sua ordem de compra, em seguida dispara o agente de compra que irá procurar pela melhor oferta. As melhores ofertas negociadas pelo agente de compra são armazenadas no tipo `Cotação`. Em seguida, o Gerente seleciona as melhores cotações e prepara a sua requisição, que é transmitida para o fornecedor. A Figura 18 mostra o modelo de tipos deste pacote.

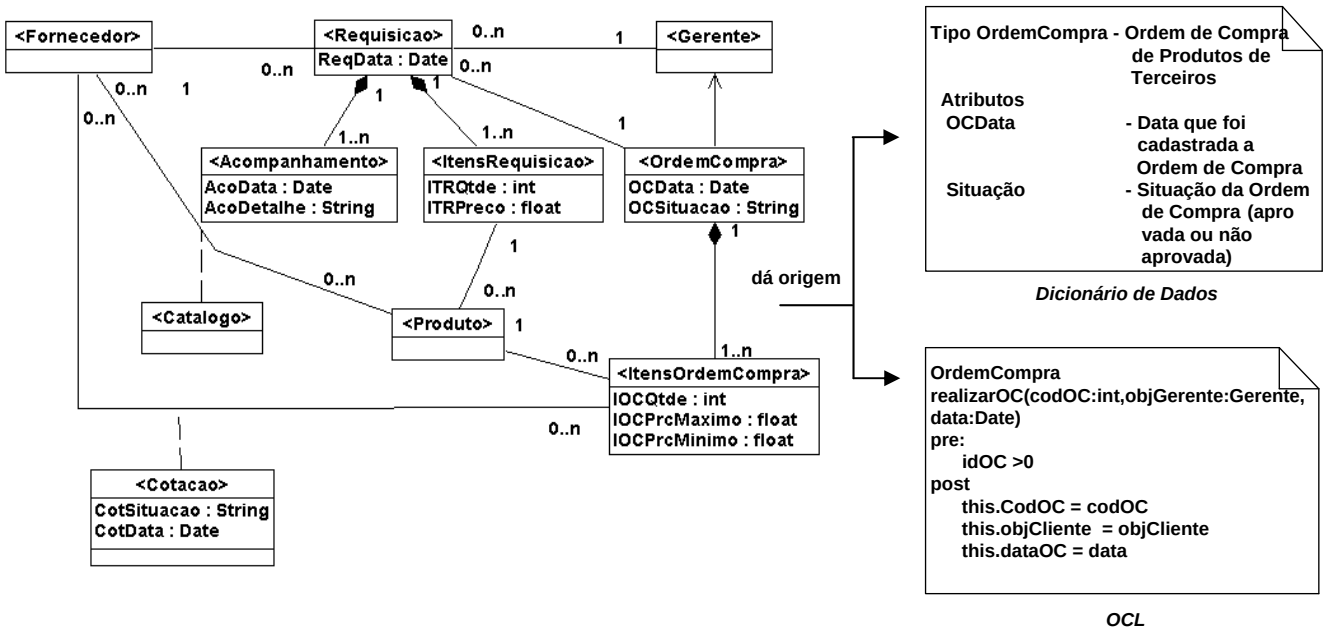


Figura 18 - Modelo de Tipos – Tratar Negociação

Ainda no Diagrama de Casos de Uso **Tratar Negociação** foram identificados os tipos responsáveis por receber e enviar as requisições do comprador para o vendedor, que foram agrupados no **Modelo de Tipos – Tratar Comunicação**, conforme mostra a Figura 19.

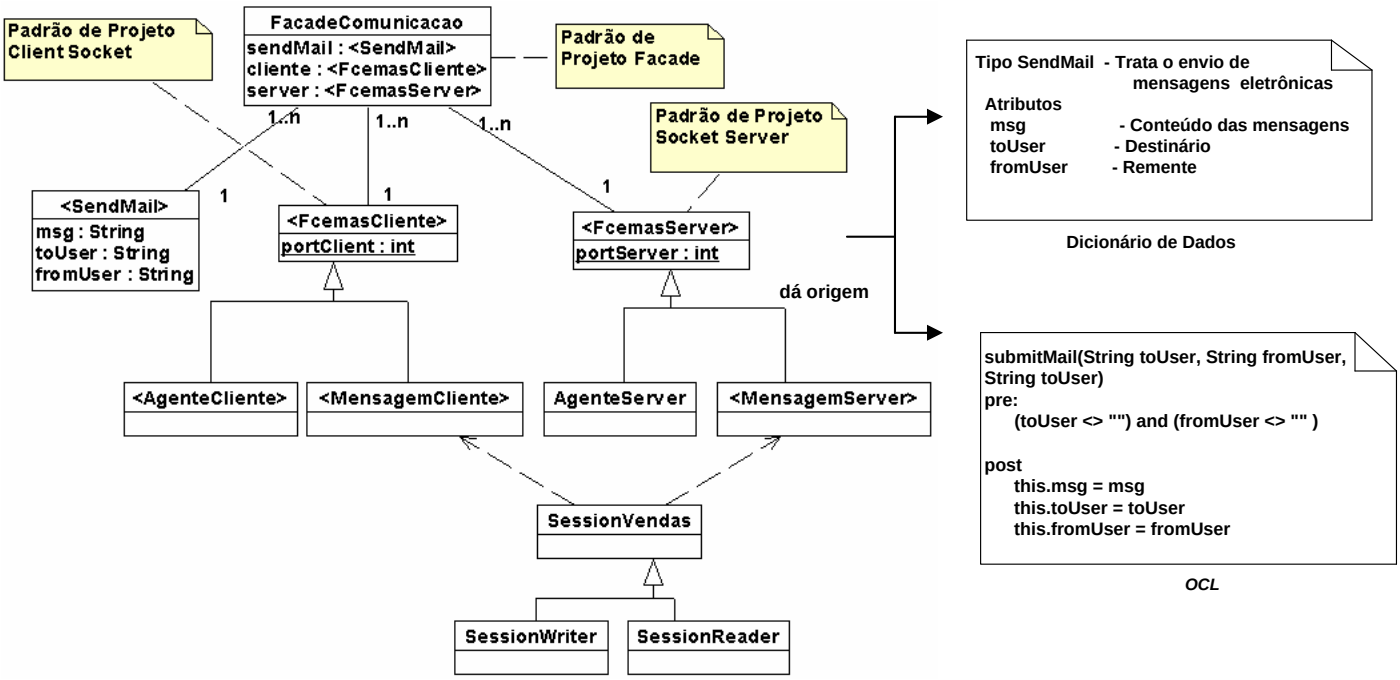


Figura 19 - Modelo de Tipos – Tratar Comunicação

O tipo `FacadeComunicacao` faz uso do padrão de projeto *Facade* [Gam95], o qual disponibiliza uma única interface de acesso. O tipo `SendMail` é responsável pelo envio de mensagens eletrônicas, *e-mails*, para os compradores e vendedores, pode ser utilizado pelos agentes de software para enviar informações a seus usuários ou pelas aplicações desenvolvidas pelo FCEMAS, como por exemplo: senha de acesso.

Para permitir que a Requisição fosse transmitida para o vendedor, foi definido um servidor de mensagens, que é representado pelos tipos `FcemasServer` e `FcemasCliente`. O tipo `FcemasServer` baseia-se no padrão de projeto *ServerSocket* [Gra99], que descreve a lógica utilizada pelos servidores para se comunicarem com seus clientes. O tipo `FcemasCliente` baseia-se no padrão de projeto *ClientSocket* [Gra99], que descreve a lógica implementada pelos clientes. Para permitir que as aplicações dos vendedores ou compradores possam enviar ou receber mensagens, foi definido o tipo `SessionVendas`. O tipo

`SessionWriter` é responsável por preparar as mensagens, no formato XML [Xml01], para serem transmitidas. O tipo `SessionReader` é responsável por ler as mensagens recebidas, no formato XML [Xml01].

Para controlar o acesso das aplicações desenvolvidas a partir do FCEMAS, foram identificados os tipos: `ServletControle`, `ControleHTTP` e `ControleWap`, como mostra a Figura 20. O tipo `ServletControle` baseia-se no padrão de projeto do catálogo J2EE, *Front Controller* [Sun01], que centraliza o gerenciamento das aplicações *Web*. O tipo `ControleHTTP` trata as requisições dos *browsers Web*, e o tipo `ControleWap` trata as requisições dos dispositivos móveis.

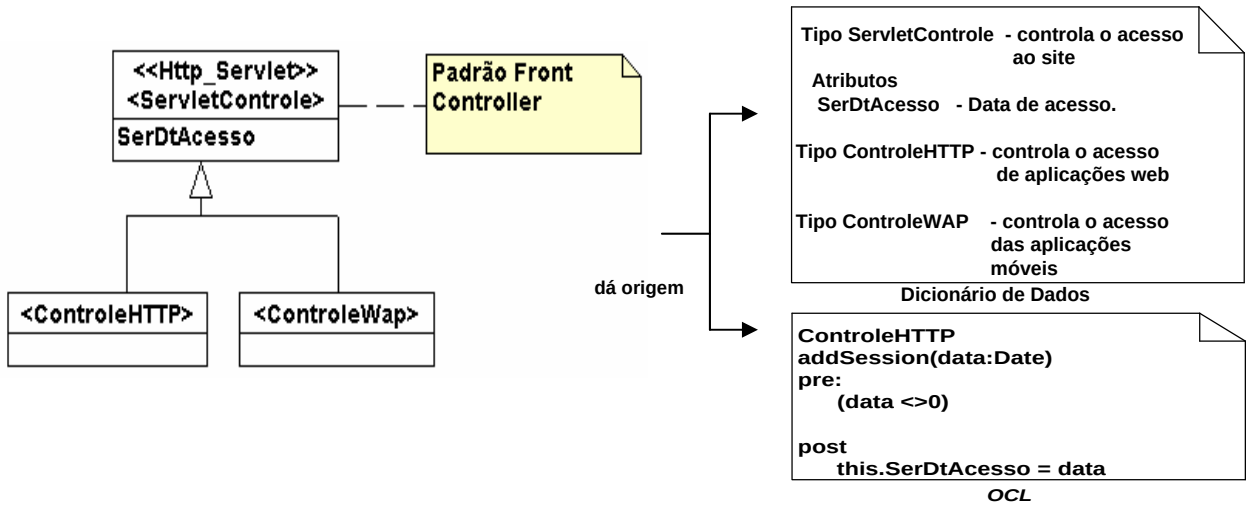


Figura 20 - Modelo de Tipos – Controlar Sistema

Devido ao grande número de tipos e para uma melhor visualização, os *Modelos de Tipos* foram agrupados em pacotes, especificando-se o *Modelo de Pacotes do Framework*, para o segundo passo da estratégia, como mostra a Figura 21. Os tipos responsáveis por controlar o acesso das aplicações foram agrupados no pacote `Controle`, e os tipos responsáveis pela comunicação foram agrupados no pacote `Comunicacao`.

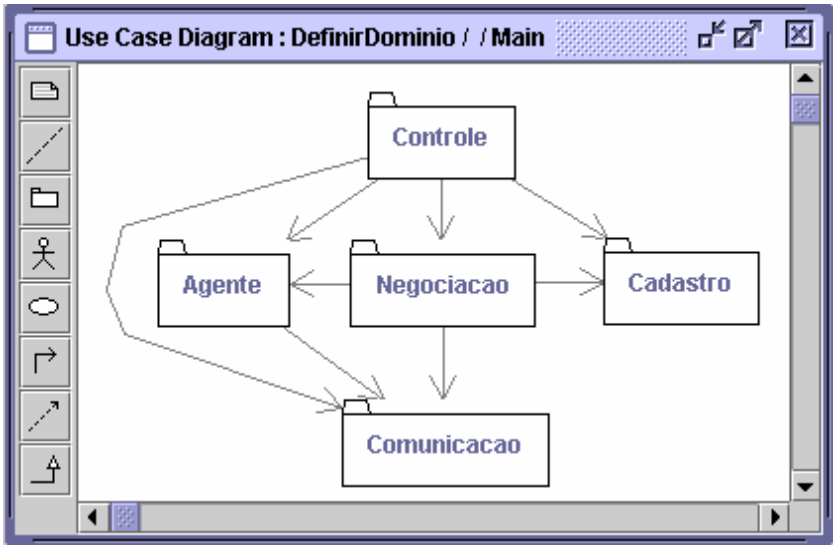


Figura 21 – Modelo de Pacotes do *Framework*

3.4 Projetar *Framework*

Neste passo, o Engenheiro de Software faz o projeto interno do *framework*. Agora, os detalhes de implementação são importantes, destacando-se: segurança, persistência, arquitetura distribuída e a linguagem de implementação.

Como passo inicial, refinam-se os *Modelos de Tipos*, agrupados no *Modelo de Pacotes do Framework*, para um Diagrama de Classes, onde são modeladas as classes e os seus relacionamentos. Os tipos agrupados nos pacotes são refinados, buscando um melhor entendimento das funcionalidades. A Figura 22 mostra o *Modelo de Classes* obtido do refinamento do pacote Agente. Neste passo, o Engenheiro de Software preocupa-se com a parte física das classes, identificando os atributos, métodos e relacionamentos. As classes sombreadas foram adicionadas nesta fase.

A interface de projeto `Runnable` do pacote `java.lang` [Sun01] é adicionada ao projeto. Esta suporta a construção de *threads*. Através deste mecanismo, múltiplas linhas de execução simultâneas são criadas, dando autonomia aos agentes de software.

A Figura 22 mostra a classe abstrata `Agente` com os seus principais métodos que devem ser implementados pelas classes `AgenteCompra`, `AgenteVenda` e `Colaboracao`.

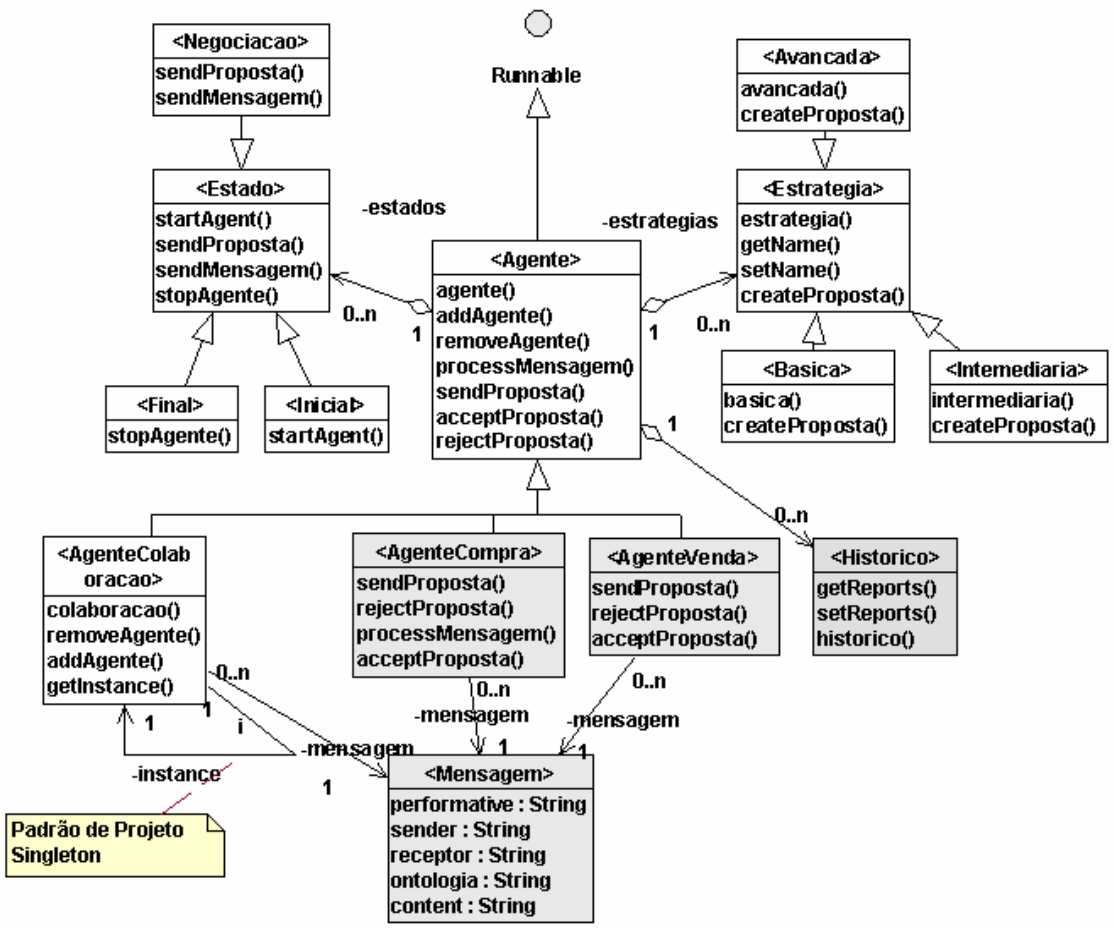


Figura 22 – Modelo de Classes do Pacote Negociação

Os atributos da classe `Agente` foram omitidos para uma melhor visualização.

- `agente` – constrói o agente;
- `addAgente` – adiciona o agente ao mercado virtual;
- `removeAgente` – remove o agente do mercado virtual;
- `startNegociacao` – inicia o processo de negociação;
- `processMensagem` – analisa as mensagens recebidas dos agentes;
- `sendProposta` – envia uma proposta de negociação;

- `acceptProposta` – concorda com a proposta; e
- `rejectProposta` – recusa a proposta.

A troca de informações entre os agentes de software é feita através da linguagem Knowledge Query Manipulation KQML [Fin96], representada pela classe `Mensagem`. A KQML possui um conjunto de mensagens pré-definidas, denominadas *performatives*, que definem as ações as serem executadas pelos agentes de software. A Tabela 02 mostra algumas das mensagens utilizadas na comunicação entre os agentes de software.

Os principais atributos da classe `Mensagem` para suportar a troca de mensagens entre os agentes são:

- `performative` – ação a ser executa pelo agente;
- `sender` – emissor da mensagem;
- `receptor` – destinatário da mensagem;
- `ontology` – termos comuns da linguagem; e
- `content` – conteúdo da mensagem

Performatives KQML	Descrição
accept-offer	Agente receptor aceita a oferta
Register	Agente emissor informa sua presença no mercado virtual
unregister	Agente emissor retira-se do mercado virtual
recommend-one	Agente emissor quer saber quem pode responder a sua mensagem
recommend-all	Agente emissor quer saber todos os que podem responder a sua mensagem
deny	Agente receptor informa que não tem o produto solicitado
reject-offer	Agente receptor rejeita a oferta
make-offer	Agente emissor faz oferta

Tabela 2 – Mensagens KQML trocadas entre os agentes de software

Outra preocupação deste trabalho foi com a persistência dos objetos. Segundo Yorder e outros [Yod98], desenvolvedores de software orientados a objetos que usam banco de dados relacional, geralmente gastam muito tempo na implementação dos objetos persistentes, devido às diferenças entre os dois paradigmas. Uma solução para facilitar o mapeamento de objetos para banco de dados relacionais é a utilização do padrão de projeto *Persistent Object* combinado dos padrões CRUD (*Create, Read, Update, Delete*), SQL *Code Description, Connection Manager e Table Manager* [Yod98,Cag99], *Singleton, Objetc Pool* e *Facade* [Gam95]. A Figura 23 mostra o modelo de classes do padrão *Persistent Object*.

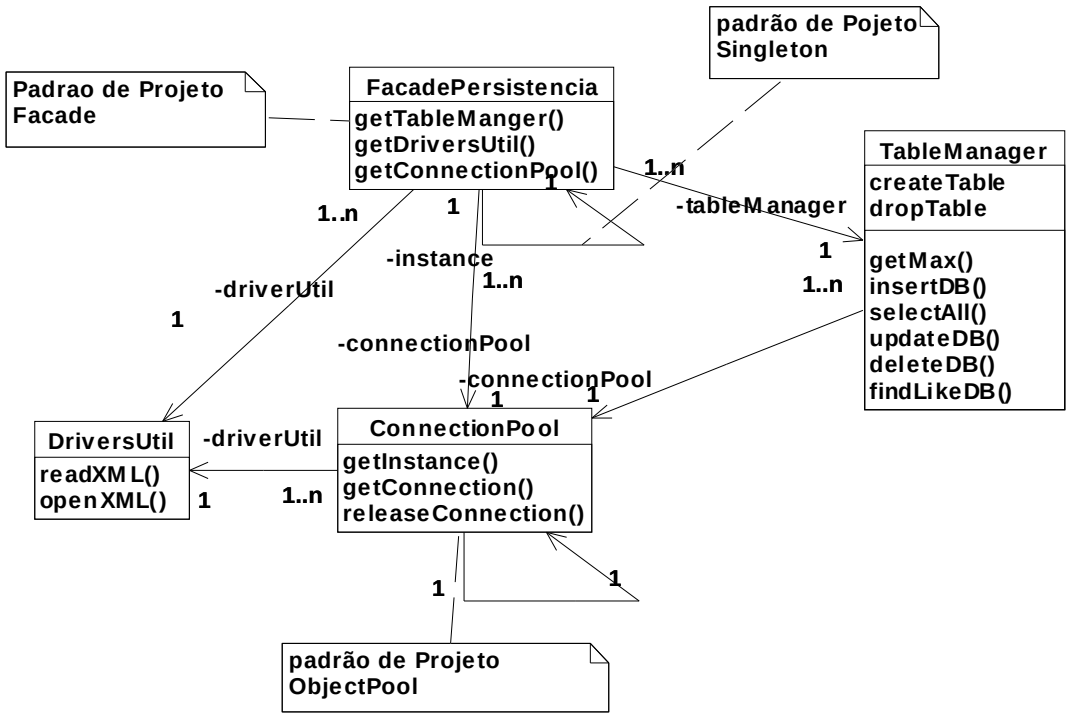


Figura 23 – Modelo de Classes do Pacote Persistência

A classe `FacadePersistencia` combina os padrões *Singleton* e *Facade*. O padrão *Facade* define uma interface de acesso às aplicações que realizam a persistência dos dados. O padrão *Singleton* permite que uma classe tenha uma única instância. As operações do padrão

CRUD realizam a inserção, remoção e recuperação de registros no banco de dados e utilizam o padrão *SQL Code Description*, para especificar comandos SQL. Esses padrões formam combinados na classe `TableManager` que realiza o mapeamento dos objetos no banco de dados relacional, gerando dinamicamente os comandos SQL. A classe `DriversUtil` carrega os drivers de acesso do banco de dados. Definidos na linguagem *eXtensible Markup Language* (XML) [Xml01]. A classe `ConnectionPool`, com base no padrão *Object Pool*, gerencia o número de conexões com o banco de dados. As classes do padrão *Persistent Object*, que foram agrupadas no pacote Persistência

Devido ao grande número de classes e para melhor visualização, os Modelos de Classes são agrupados em pacotes, especificando-se o ***Modelo de Pacotes do Framework***. A Figura 24 mostra o Modelo de Pacotes do *Framework*. O pacote do *framework*, denominado FCEMAS, contém os dois grandes: `Principal` e `Suporte`. O FCEMAS reutiliza classes dos pacotes `java` e `javax` [Sun01].

O pacote `Principal` foi estruturado numa arquitetura de três camadas: *Interface, Business Service* e *Data Service*. A *Interface* é responsável por tratar as requisições das aplicações distribuídas. As regras do negócio, do comércio eletrônico, ficam na camada *Business Service*. A camada *Data Service* provê o acesso ao Banco de Dados. O pacote `Principal` é composto dos pacotes: `Controle`, `Cadastro`, `Negociação`, `Agente` e `Persistência`.

O pacote `Suporte` fornece a infra-estrutura e os serviços básicos para construção de aplicações. É composto dos pacotes: `Comunicação` e `Excecao`. O pacote `Comunicacao` contém classes que suportam a comunicação dos agentes de software com seus usuários, utiliza os serviços do pacote `javax.mail` [Har00] para a troca de mensagens eletrônicas, e do pacote `java.net` [Har00], para o envio e confirmação das requisições e compra de produtos. O pacote `Excecao` que contém classes para tratamento de exceções.

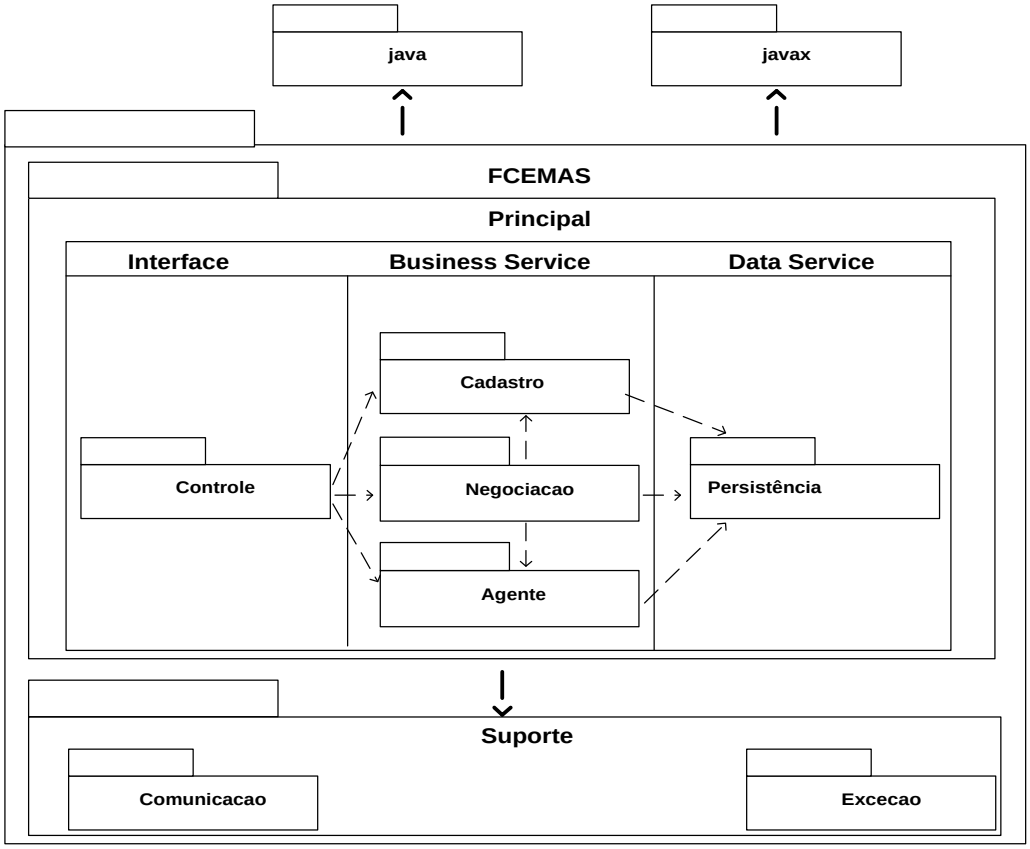


Figura 24 –Modelo de Pacotes do *Framework*

3.5 Implementar *Framework*

Por último, após o projeto do *framework*, fez-se a implementação do *framework.*

A *MVCASE* é usada para gerar o código a partir dos modelos projetados. Refinamentos nas assinaturas dos métodos e nos atributos das classes podem ser necessários nesta fase. O comportamento dos métodos é especificado diretamente na linguagem Java, alvo da implementação.

A Figura 25 mostra parte do código da classe AgenteCompra gerado pela *MVCASE* .

```
. . .
public class AgenteCompra extend Agente{
. . .
   public void processMensagem(Mensagem msg) {
     . . .
     // agente vendedor aceita a oferta
     if (msg.getPerformative().equals("accept-offer" )) {
           String emissor = msg.getSender;
           // armazena as informacoes
           addCotacao(msg.getProduto(), emissor, msg.getValorNeogiciado) ;
           . . .
    }
      // vendedor solicita uma contra oferta
     if (msg.getPerformative().equals("make-offer" )) {
           // baseado na sua estrágia cria-se uma nova proposta
           valorOferta = estrategia.createProposta(dtInicial,dtFinal,vrIni,
                                                   vrFim);
           Message newMsg =  new Message(...);
          sendProposta(newMsg) ;    }
```

Figura 25 – Implementação da Classe AgenteCompra – Pacote Negociação

A Figura 26 mostra parte do código da classe `TableManager` do pacote persistência.

```
pubic class TableManager {
             . . .
             public  ResultSet  findLikeDB(String  dbName,  String  tableName,  Vector
             . . .clause, Vector parameter){
                    . . .
                        // Realiza a geração dinâmica dos comandos SQL
             findlikeSQL = "SELECT * FROM " + tableName + " WHERE ";
             for (int i = 0; i <= contador;i++){
                    // obtem o tipo de objeto que está armazenado em parameter
                    tipo = (parameter.elementAt(i)).getClass().getName();
                    // compara o tipo do objeto lido,
                    if (tipo.equals("java.lang.String"))
                           findlikeSQL  =  findlikeSQL  +  clause.elementAt(i)  +  "'"  +
                    parameter.elementAt(i) + "'";
                    else
                           findlikeSQL = findlikeSQL + clause.elementAt(i) +
                           parameter.elementAt(i);
             }
          // realiza a conexão com o Banco de Dados
             conn = connPool.getConnection(dbName);
             ResultSet rs = connPool.getConnection(driver).
                           createStatement().executeQuery(findlikeSQL);
             return rs;
                    }
                    . . .
}
```

Figura 26 – Implementação da Classe TableManager – Pacote Persitência

Após este passo tem-se, o *framework* implementado, com suas classes disponíveis para reutilização.

Para mostrar a reutilização do *framework*, segue-se a apresentação da criação do domínio de aplicações, para comércio eletrônico, neste caso, de produtos farmacêuticos, que reutiliza o FCEMAS.

Capítulo 4

Desenvolvimento de Componentes reutilizando o FCEMAS

A partir do FCEMAS é possível desenvolver componentes dos domínios de aplicações para comércio eletrônico, que reutilizam suas classes. A construção de um domínio é realizada em quatro passos: **Definir Domínio do Problema**, **Especificar Componentes, Projetar Componentes e Implementar Componentes**, conforme mostra a Figura 27.

A idéia de utilizar o Desenvolvimento Baseado em Componentes (DBC) visa principalmente facilitar a reutilização dos componentes nas diferentes aplicações.

Parte-se, neste caso dos requisitos comuns às aplicações de um domínio do problema, para construir componentes que possam ser reutilizados pelas aplicações deste domínio.

As linhas horizontais tracejadas separam os passos da estratégia segundo os níveis de *Catalysis:* **Domínio do Problema, Especificação dos Componentes e Projeto Interno dos Componentes**, mostrados à direita da Figura 27.

Da mesma forma que na construção do *framework*, o Engenheiro de Software conta com a *MVCASE* [Bar99] como principal mecanismo de execução de cada passo do desenvolvimento.

Parte-se dos requisitos da aplicação do Domínio do Problema e baseado nas técnicas de *Catalysis* e na reutilização do FCEMAS, especifica-se os requisitos do Domínio do Problema.

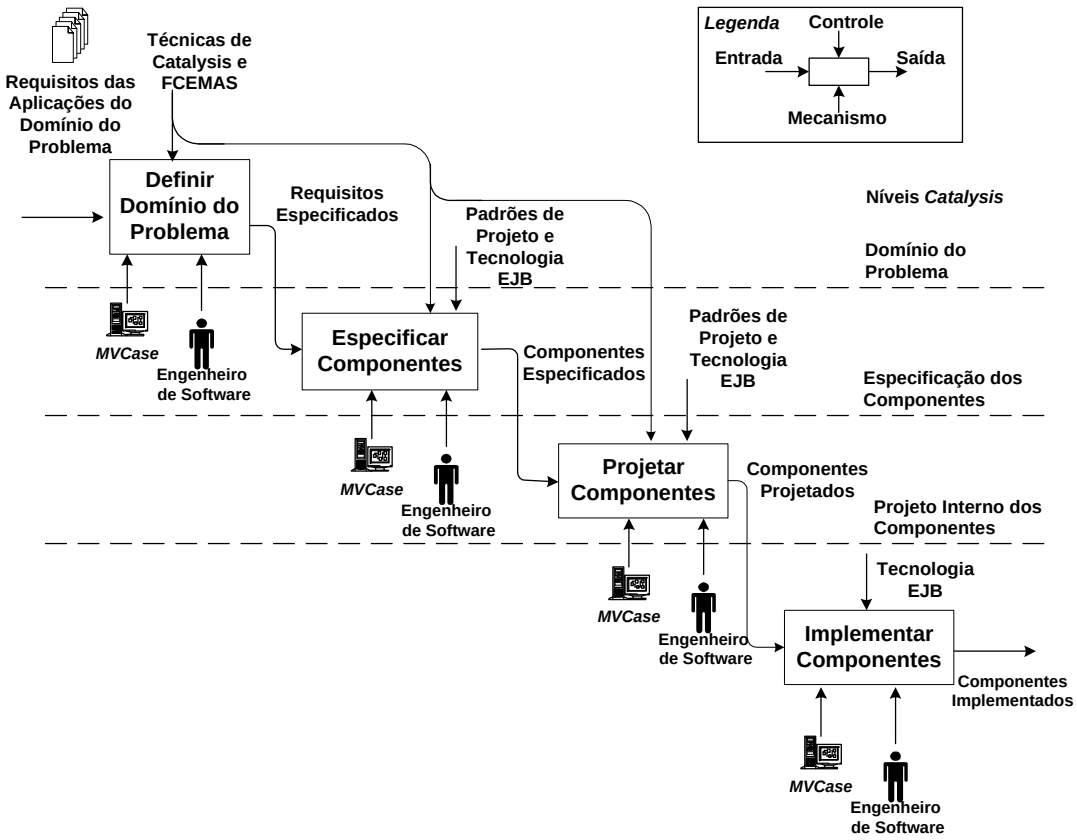


Figura 27 –Desenvolvimento de Componentes Distribuídos de um Domínio de Aplicações

Em seguida, os requisitos especificados do Domínio do Problema são modelados em componentes, baseado nos padrões de projeto [Gam95, Gra99, Sun01] e da tecnologia EJB [Sun01, Rom99], resultando nos componentes especificados.

No passo Projetar Componentes, define-se a plataforma de projeto e adicionam-se os requisitos não funcionais para obter o Projeto Interno dos Componentes. Finalmente os

componentes projetados são implementados em EJB, para serem reutilizados pelas diferentes aplicações do domínio construído.

Segue-se uma apresentação de cada passo da estratégia para a construção dos componentes de um Domínio de Aplicações.

O domínio de farmácia foi usado como exemplo para explicar detalhes de cada passo do processo de desenvolvimento. Pretende-se dessa forma, definir uma estratégia que pode ser seguida para a construção de outros domínios, que também estarão reutilizando o FCEMAS.

4.1 Definir Domínio do Problema

Trata-se de um domínio de aplicações relacionadas com o controle de distribuição de produtos farmacêuticos, via *Web* ou *Wap*. O controle se divide em dois grandes módulos. O primeiro é responsável por Controlar e Atender os Pedidos dos Revendedores, e compreende o cadastro de: fornecedor, vendedor, gerente, produto, grupo, laboratório, substância, pedido e ordem de compra. Outras funcionalidades deste módulo são: aprovar o fornecedor, controlar o estoque de produtos e disparar o processo de compras de produtos de terceiros, mediado pelos agentes de software. O segundo módulo é responsável por fornecer informações, sobre: a situação atual dos pedidos, as cotações dos produtos, a situação das requisições de compra, os históricos dos fornecedores, e as atividades dos agentes de software.

Neste passo, o Engenheiro de Software, partindo do arquivo `ListaCasosDeUsoFCEMAS.doc`, analisa os Casos de Uso do FCEMAS, e determina quais

***Diagramas de Casos de* Uso** serão reutilizados e quais devem ser adicionados. A Tabela 3 lista os principais Casos de Uso definidos. Os Casos de Uso sombreados são do FCEMAS e os demais são específicos do domínio do problema.

Caso de Uso	Descrição	Entrada	Resposta
CadastrarOrdemCompra	Comprador cadastra Ordem de Compra	DadosOrdemCompra	Msg08
IniciarAgente	Comprador inicia Agente de Venda	CódigoAgenteCompra	Msg05
IniciarAgente	Vendedor inicia Agente de Venda	CódigoAgenteCompra	Msg07
RealizarNegociação	Agentes de Compra e Venda realizam o Processo de Negociação	É hora de realizar negociação	Informações de Venda e Informações de Venda
GerarSenhaAcesso	Aplicação gera Senha	DadosUsuario	Senha
DefinirAgenteCompra	Comprador determina a estratégia de Negociação e o período para o Agente realizar a Negociação	DadosAgenteCompra	Msg01
DefinirAgenteVenda	Vendedor determina a estratégia de Negociação e o período para o Agente realizar a Negociação	DadosAgenteVenda	Msg02
CadastrarRequisição	Comprador Cadastra Requisição	DadosRequisição	Msg08
EnviarRequisição	Comprador Envia a Requisição de Acordo com as Negociações Realizadas pelos Agentes	Requisição	MensagemRequisição
ReceberRequisição	Fornecedor Recebe a Requisição	Requisição	PosiçãoTransmissão
TratarRequisição	Fornecedor Realiza o Processamento da Requisição	CódigoRequisição	RelatórioRequisição
CadastrarGrupo	Gerente Cadastra Grupos dos Medicamentos	DadosGrupo	Msg30
CadastrarLaboratório	Gerente Cadastra Laboratórios dos Medicamentos	DadosLaboratorio	Msg31
CadastrarSubstância	Gerente Cadastra Substância dos Medicamentos	DadosSubstancia	Msg32

Tabela 3 – Lista de Casos de Uso

Partindo do arquivo `ListaUseCaseFCEMAS.doc`, o Engenheiro de Software, utilizando a *MVCASE* e reutilizando o FCEMAS, constrói o ***Diagrama de Casos de Uso*** do domínio da aplicação**,** como mostra a Figura 28. Os casos de usos sombreados são do FCEMAS, os demais são específicos da aplicação.

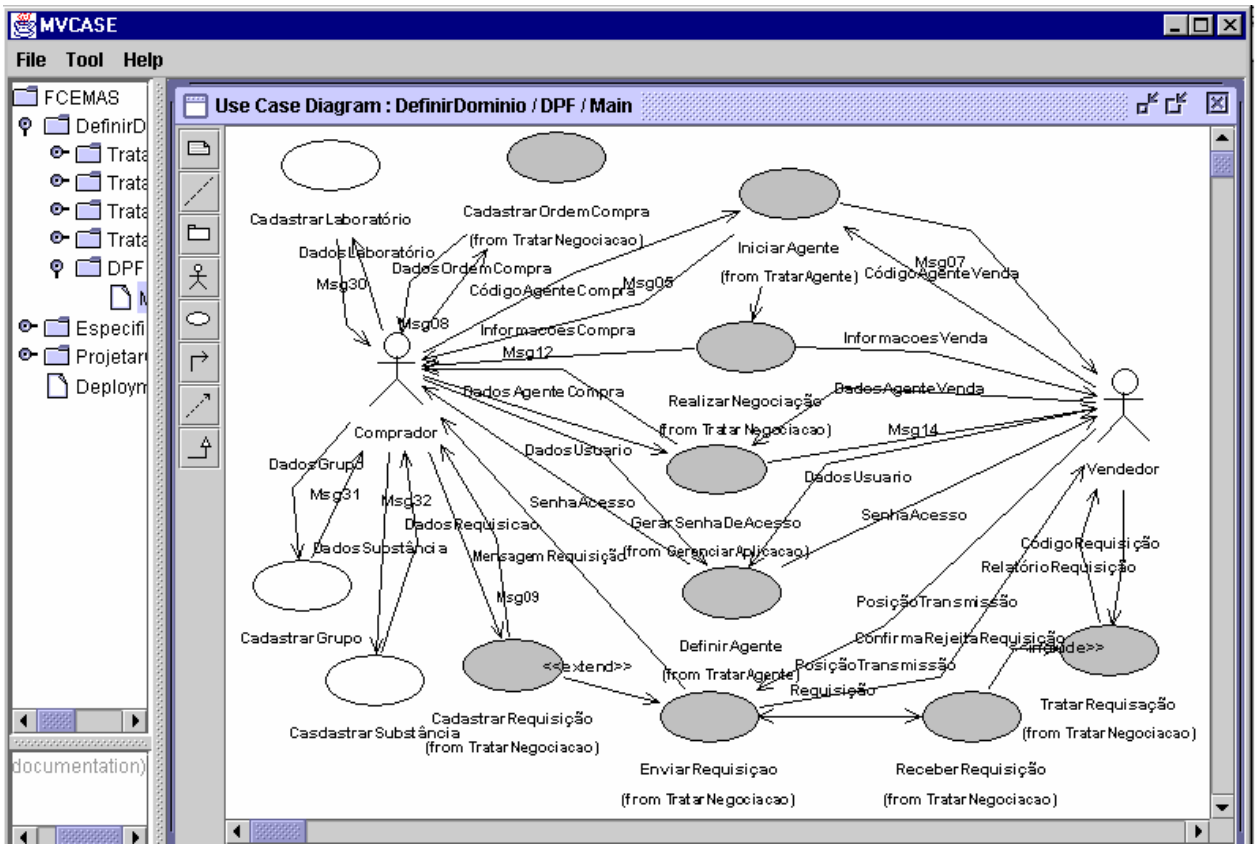


Figura 28 – Diagrama de Casos de Uso Tratar Cadastro - Domínio de Farmácia

4.2 Especificar Componentes

Este passo corresponde ao ***segundo nível*** de *Catalysis*, onde é descrito o comportamento externo do sistema de uma forma não ambígua. O Engenheiro de Software refina as

especificações do Domínio do Problema do nível anterior, visando obter as especificações dos componentes distribuídos.

Esse passo tem início com o mapeamento das especificações Modelos de Casos de uso, do passo anterior para o *Modelo de Tipos* [Boo00, Dso98, Lar99].

A Figura 29 mostra o Modelo de Tipos do Domínio das Aplicações. Os tipos `Laboratório`, `Substância` e `Grupo` são específicos do domínio, e os demais, do FCEMAS.

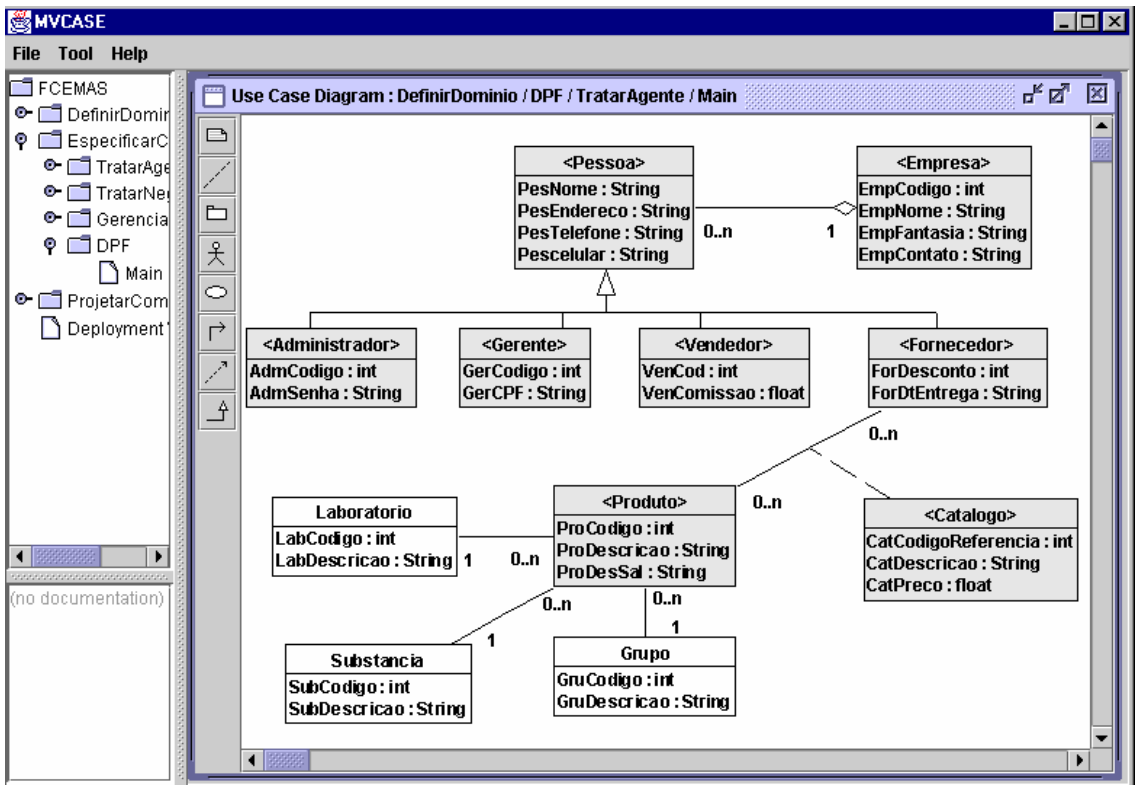


Figura 29 – Modelo de Tipos do Domínio das Aplicações

Os Casos de Uso são documentados através de descrições textuais e de Diagrama de Seqüência que detalham seus comportamentos. A Figura 30 mostra o Diagrama de Seqüência do curso normal do Caso de Uso CadastrarSubstância.

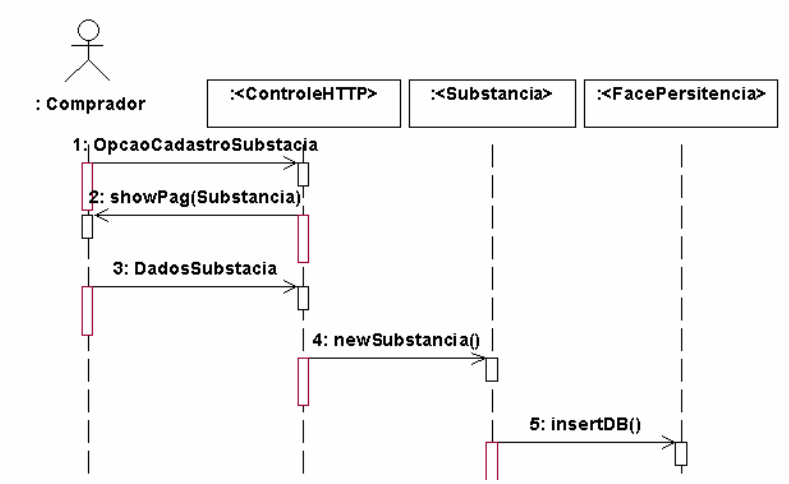


Figura 30 – Diagrama de Seqüência - Cadastrar Substância

Para as aplicações do domínio de farmácia foi adicionada uma nova estratégia de negociação. Esta estratégia utiliza uma função logarítmica para determinar o valor do produto a ser negociado pelos agentes de software.

Para o Diagrama de Casos de Uso Definir Agente foi identificado o novo tipo: `Logarítmica`, como mostra a Figura 31. Os tipos sombreados são do FCEMAS, os demais são específicos do domínio da aplicação.

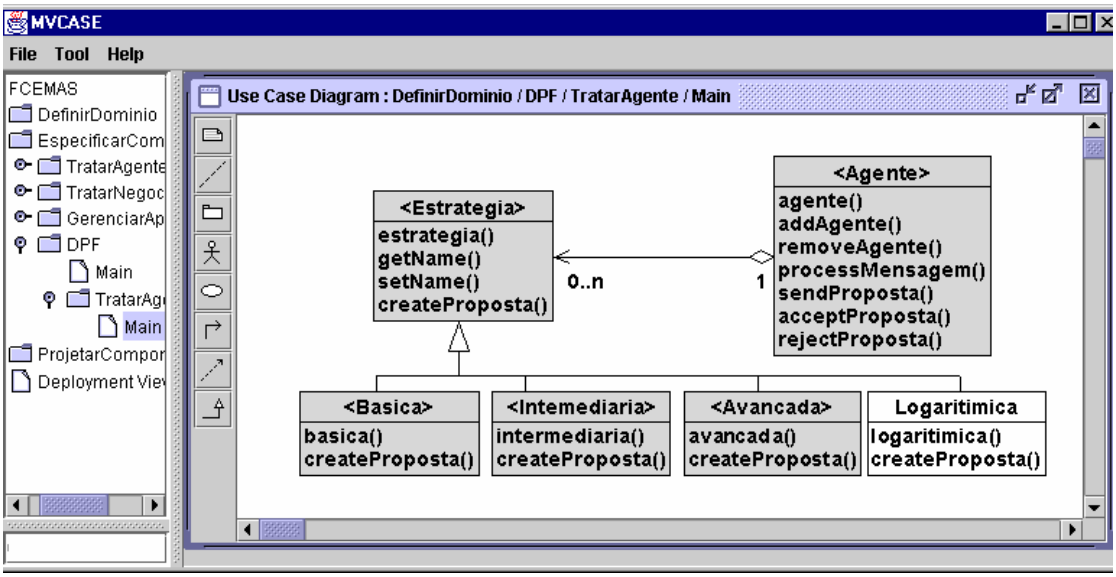


Figura 31 – Modelo de Tipos – Tratar Agente

4.3 Projetar Componentes

Neste passo, o Engenheiro de Software faz o projeto interno dos componentes, conforme o terceiro nível de *Catalysis.* Agora, os detalhes de implementação são importantes, destacando-se: segurança, persistência, arquitetura distribuída e a linguagem de implementação.

Como passo inicial, refina-se o *Modelo de Tipos*, em um Diagrama de Classes, onde são modeladas as classes, os seus relacionamentos e as interfaces dos componentes das aplicações.

Para os tipos agrupados nos pacotes `GerenciarAplicação` e `TratarNegociacao`, foram construídos os componentes EJB do tipo *Entity*. Para cada Caso de Uso, foram construídos componentes EJB do tipo *Session.*

A Figura 32 mostra, à esquerda, parte do *Modelo de Tipos - TratarNegociacao*, onde o tipo `OrdemCompra` dá origem ao componente distribuído `OrdemCompraBean` na plataforma EJB. Através da opção gerar **Entity Beans** da *MVCASE*, o Engenheiro de Software pode gerar os componentes. No caso, foi gerado o componente EJB do tipo **Entity Bean,** cujas interfaces são **OrdemCompra** e **OrdemCompraHome**. Além das interfaces, são gerados os métodos relacionados com o ciclo de vida do *bean, como* `ejbCreate` e `ejbRemove`. Caso necessário, são também criados classes para identificar o *bean* persistido em banco de dados relacional, como `OrdemCompraPK`.

O procedimento é repetido para gerar os demais componentes *Entity* ou *Session*. A Tabela 04 mostra os componentes do tipo *Entity Bean*, e a Tabela 05 mostra os componentes do tipo *session bean.*

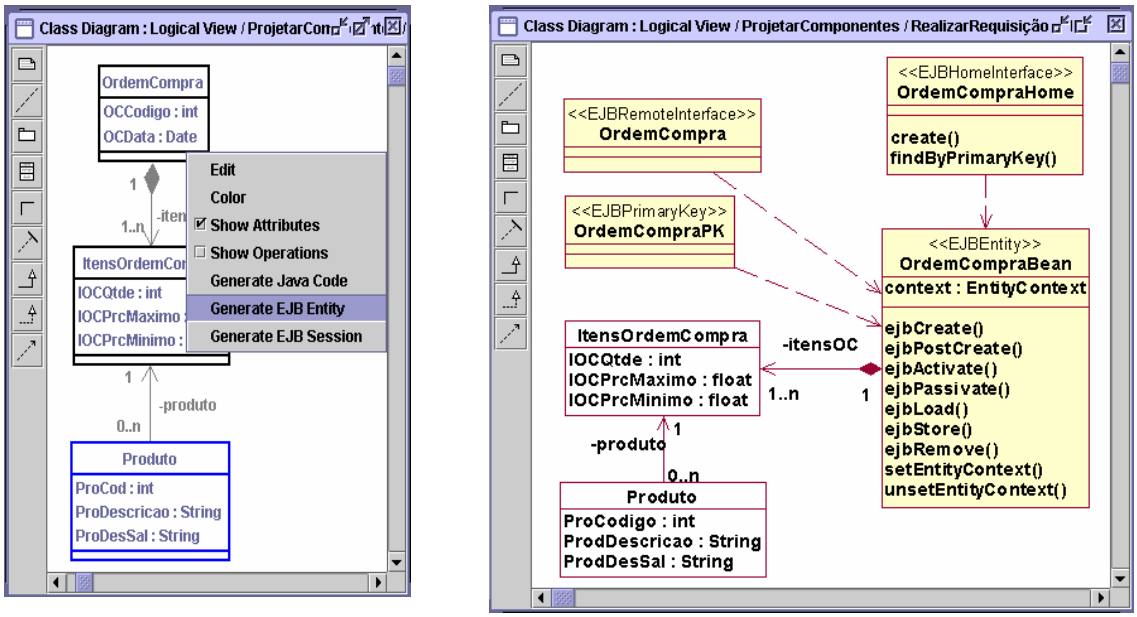


Figura 32 – Projeto do Componente EJB – Ordem de Compra

Componentes do Tipo Entity		
Pacote Gerenciar Aplicação		**Pacote Tratar Negociação**
PessoaHome Pessoa PessoaPK PessoaBean	EmpresaHome Empresa EmpresaPK EmpresaBean	RequisicaoHome Requsicao RequsicaoPK RequisicaoBean
AdministradorHome Administrador AdminitradorPK AdministradorBean	GerenteHome Gerente GerentePK GerenteBean	ItensRequisicaoHome ItensRequisicao ItensRequisicaoPK ItensRequisicaoBean
VendedorHome Vendedor VendedorPK VendedorBean	FornecedorHome Fornecedor FornecedorPK FornecedorBean	OrdemCompraHome OrdemCompra OrdemCompraPK OrdemCompraBean
LaboratorioHome Laboratorio LaboratorioPK LaboratorioBean	ProdutoHome Produto ProdutoPK ProdutoBean	ItensOrdemCompraHome ItensOrdemCompra ItensOrdemCompraPK ItensOrdemCompraBean
CatalogoHome Catalogo CatalogoPK CatalogoBean	SubstanciaHome Substancia SubstanciaPK SubstanciaBean	AcompanhamentoHome Acompanhamento AcompanhamentoPK AcompanhamentoBean

GrupoHome Grupo GrupoPK GrupoBean	ProdutoCompostoHome ProdutoComposto ProdutoCompostoPK ProdutoCompostoBean	

Tabela 4 – Listagem dos Componentes Entiy Bean

Componentes do tipo Session Bean		
Pacote GerenciarAplicação		**Pacote TratarNegociação**
CadastrarPessoaHome CadastrarPessoa CadastrarPessoaBean	CadastrarEmpresaHome CadastrarEmpresa CadastrarEmpresaBean	CadastrarRequisicaoHome CadastrarRequsicao CadastrarRequisicaoBean
CadastrarAdministradorHome CadastrarAdministrador CadastrarAdministradorBean	CadastrarGerenteHome CadastrarGerente CadastrarGerenteBean	CadastrarItensRequisicaoHome CadastrarItensRequisicao CadastrarItensRequisicaoBean
CadastrarVendedorHome CadastrarVendedorPK CadastrarVendedorBean	CadastrarFornecedorHome CadastrarFornecedor CadastrarFornecedorBean	CadastrarOrdemCompraHome CadastrarOrdemCompra CadastrarOrdemCompraBean
CadastrarLaboratorioHome CadastrarLaboratorio CadastrarLaboratorioBean	CadastrarProdutoHome CadastrarProduto CadastrarProdutoBean	CadastrarItensOrdemCompraHome CadastrarItensOrdemCompra CadastrarItensOrdemCompraBean
CadastrarCatalogoHome CadastrarCatalogo CadastrarCatalogoBean	CadastrarSubstanciaHome CadastrarSubstancia CadastrarSubstanciaBean	CadastrarAcompanhamentoHome CadastrarAcompanhamento AcompanhamentoBean
CadastrarGrupoHome CadastrarGrupo CadastrarGrupoBean	CadastrarProdutoCompostoHome CadastrarProdutoComposto CadastrarProdutoCompostoBean	

Tabela 5 – Listagem dos Componentes Session

Para os tipos agrupados no pacote `Agente`, foram definidos os componentes: Agente e Estratégia, e suas interfaces, como mostra a Figura 33.

Para os tipos agrupados no pacote `Controle`, foi definido o componente Controle, como mostra a Figura 34.

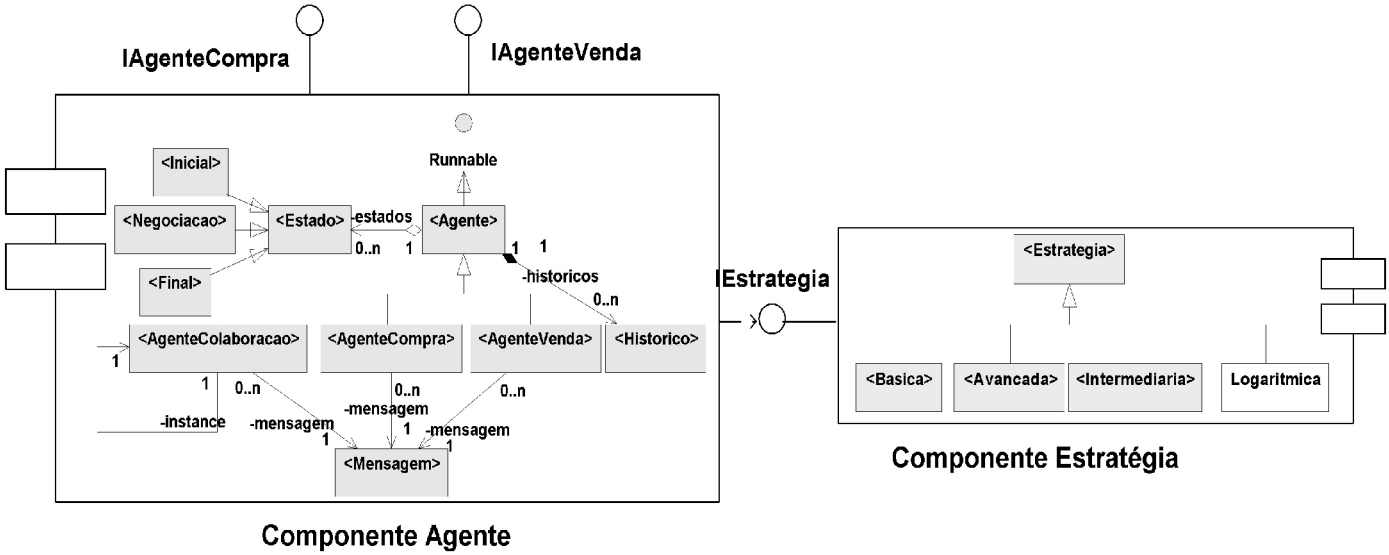


Figura 33 – Componentes do Pacote Agente

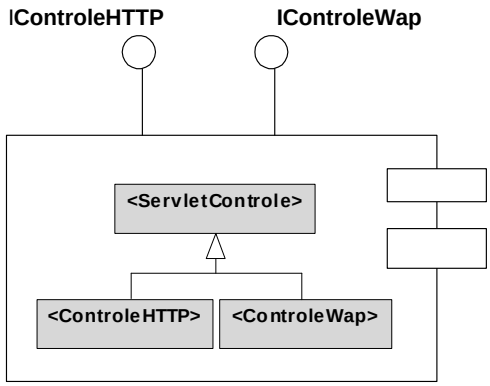


Figura 34 – Componente Controle

Os componentes foram agrupados em pacotes e representados em Diagramas de Componentes. A Figura 35 mostra um diagrama de componentes com os componentes:

`OrdemCompraBean`, `GerenteBean`, `ItensOrdemCompraBean`, `SubstanciaBean`, `ProdutoBean`, `LaboratorioBean` e `GrupoBean`.

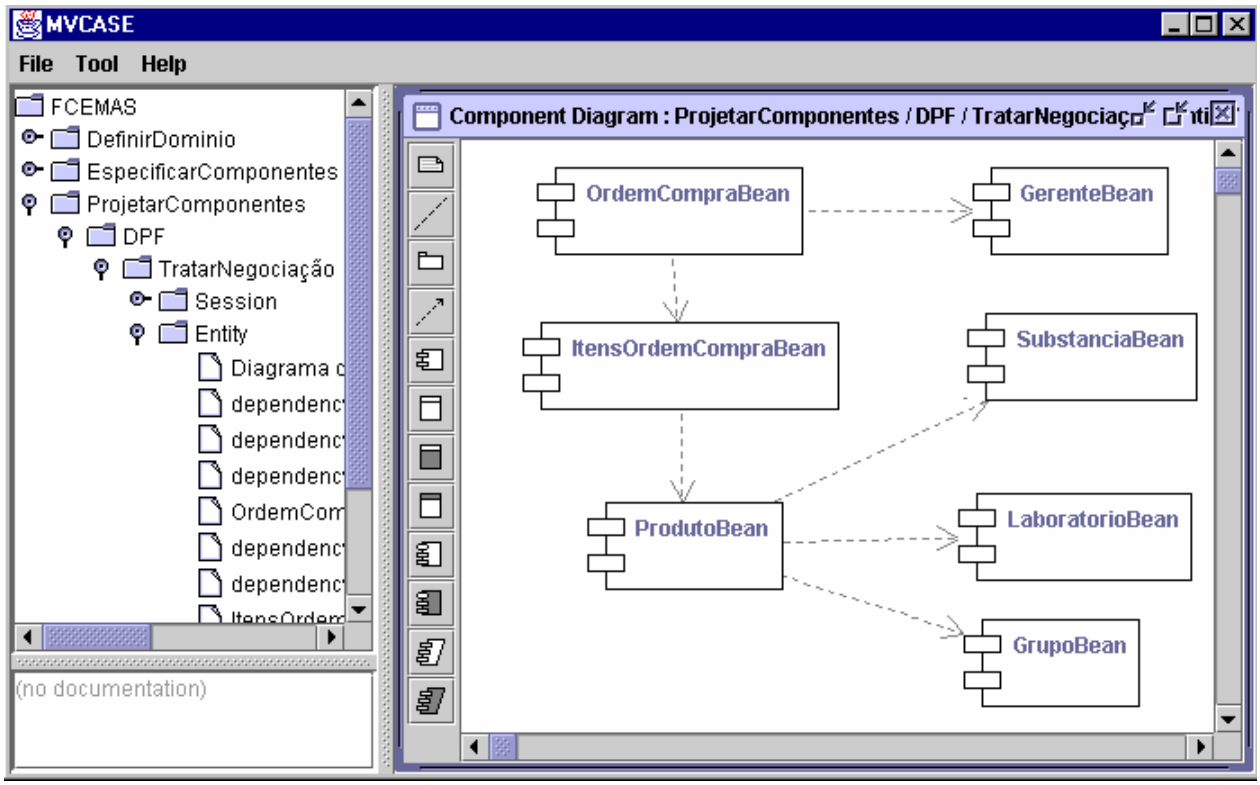


Figura 35 – Diagrama de Componentes

Outras técnicas podem ser utilizadas neste passo da estratégia, como por exemplo, os Diagramas de Colaboração [Boo00, Dso98, Lar99], que refinam os Diagramas de Seqüência do nível anterior.

A Figura 36 resume os principais artefatos e a seqüência de atividades do passo Projetar Componentes, que incluem:

- Criação do Diagrama de Classes de Componentes, com o reutilização do FCEMAS;

- Criação do Diagrama de Colaboração a partir do refinamento do Diagrama de Seqüência, e;
- Criação do Diagrama de Componentes.

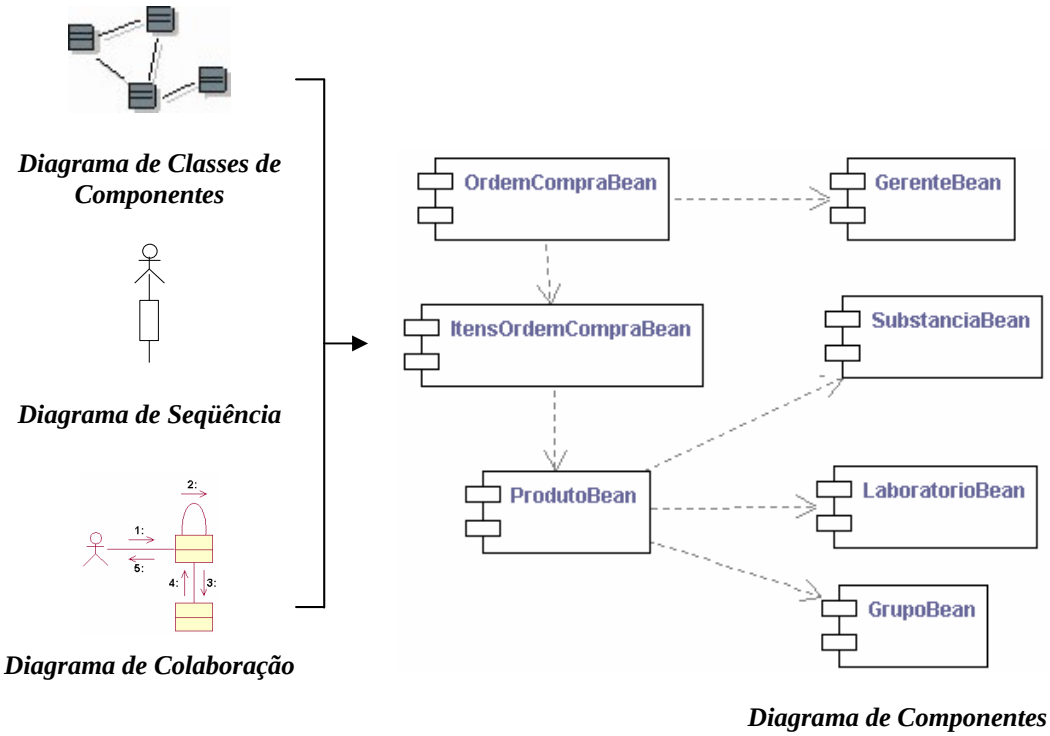
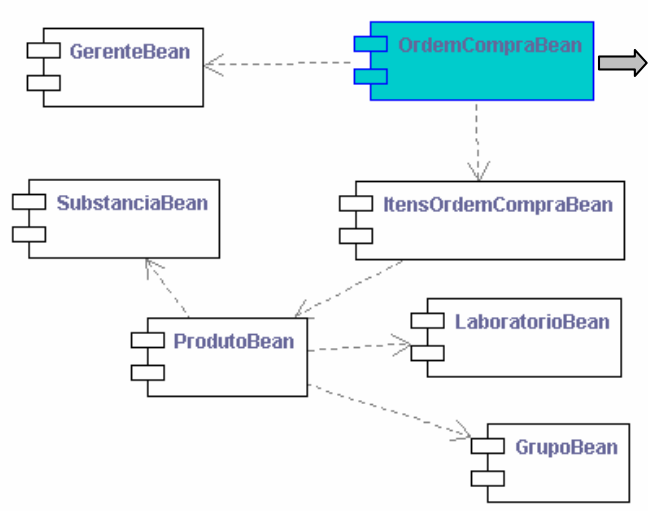


Figura 36 – Terceiro passo da Estratégia: Projetar Componentes

4.4 Implementar Componentes

Por último, após o projeto dos componentes, o Engenheiro de Software determina na *MVCASE* que se faça a geração automática dos *stubs* e *skeletons*, e das interfaces dos componentes do tipo EJB.

A Figura 37 mostra, à esquerda, alguns dos componentes projetados, e à direita, parte do código gerado pela *MVCASE* usando a tecnologia Java/EJB.




```
public class OrdemCompraBean implements
EntityBean {

  public                    java.util.Vector
ref_ItensOrdemCompra;

  private int OCCodigo;

  private GerenteHome gerenteHome;

  private EntityContext context;

. . .

  public void setOCCodigo(int p_OCCodigo) {

      this.OCCodigo = p_OCCodigo;}

  public  void  setEntityContext(EntityContext
p_context)   {

      this.context = p_context; }

  public OrderCpmpraPK ejbCreate

     (int    p_OCCodigo,    Date    p_OCData,
GerenteBean p_OCGerente)

throws CreateException {

     this.OCCodigo = p_OCCodigo;

. . .

 return  null;}

…   }
```

Figura 37 - Quarto passo da Estratégia: Implementar Componentes

Uma vez construído os componentes do domínio, pode-se desenvolver aplicações que reutilizam os componentes construídos.

Segue-se a apresentação de um estudo de caso, de uma aplicação do domínio de Comércio Eletrônico para produtos farmacêuticos, que reutilizam os componentes do Domínio de Farmácia.

Capítulo 5

Desenvolvimento de Aplicações para Comércio Eletrônico, via Internet Móvel, Mediado por Agentes de Software

Este capítulo apresenta uma estratégia para o desenvolvimento de aplicações que reutilizam os componentes construídos para o domínio de farmácia.

Embora a estratégia esteja instanciada para o Domínio de Farmácia, ela pode ser aplicada a outros domínios semelhantes, que têm aplicações de comércio eletrônico.

Trata-se de uma aplicação de compra de produtos farmacêuticos de terceiros. Inicialmente o usuário comprador realiza o cadastro da ordem de compra, de acordo com as suas necessidades. Utilizando um dispositivo móvel, telefone celular, o usuário comprador inicia as atividades do agente de compra. Com base na estratégia definida pelo usuário, o agente de compra procura por agentes de venda, que possam atender as necessidades de seu usuário, procurando obter a melhor forma de negociação, baseados em suas estratégias.

Conforme mostra a Figura 38, a estratégia de desenvolvimento da aplicação é semelhante à adotada para o desenvolvimento dos componentes do domínio, exceto que agora a instância é a aplicação.

Nesse caso, os componentes construídos, do Domínio de Distribuição de Produtos Farmacêuticos (DPF), para Comércio Eletrônico, a partir do FCEMAS são reutilizados no desenvolvimento da aplicação. Novamente a *MVCASE* auxilia o Engenheiro de Software no desenvolvimento da aplicação. Parte-se agora dos requisitos da aplicação e no final obtém-se a aplicação implementada com componentes, para ser executada em plataforma distribuída. A mesma tecnologia EJB é usada para desenvolver a aplicação, adicionada a outras tecnologias específicas para a aplicação, como no caso da *Web.*

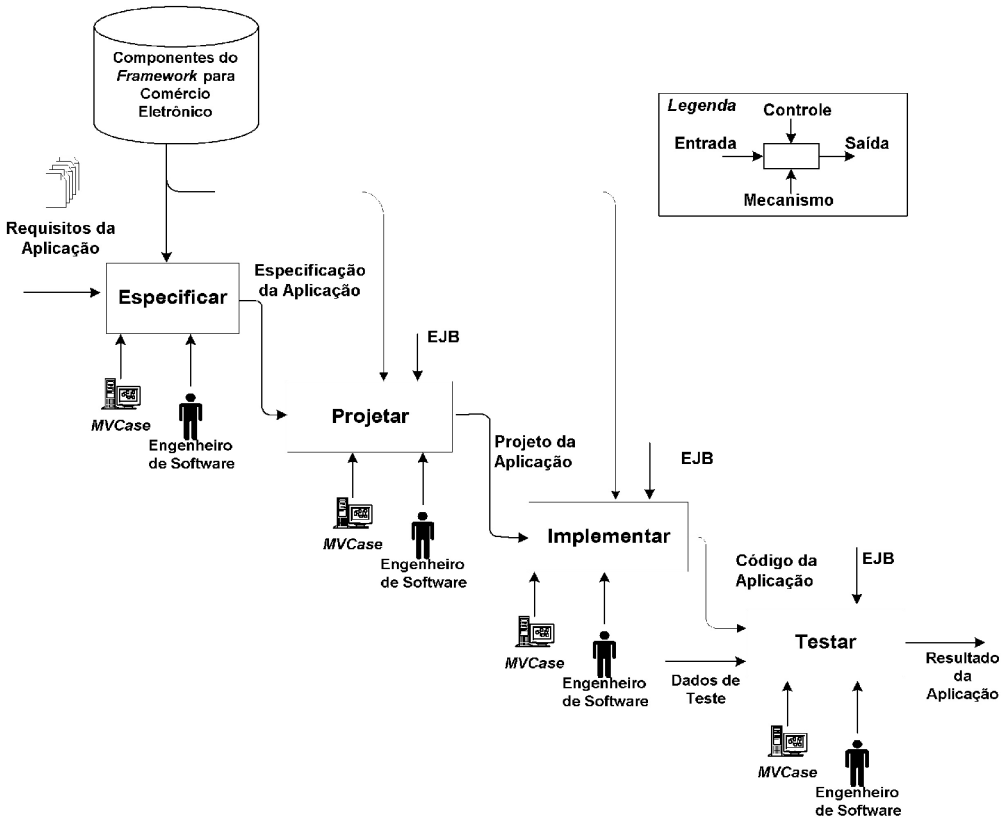


Figura 38 – Desenvolvimento de aplicações usando componente distribuídos do *Framework*

Segue-se uma apresentação de cada passo do desenvolvimento da aplicação.

5.1 Especificar

Este passo tem início com o entendimento do problema, identificando-se os requisitos da aplicação para solucioná-lo.

Em seguida, após o levantamento e identificação dos requisitos da aplicação, o Engenheiro de Software, na ferramenta *MVCASE*, especifica os requisitos com técnicas UML [Boo00]. A Figura 39 mostra, por exemplo, o Diagrama do caso de uso *IniciarAgenteCompra*, e o respectivo diagrama de seqüência do seu curso normal.

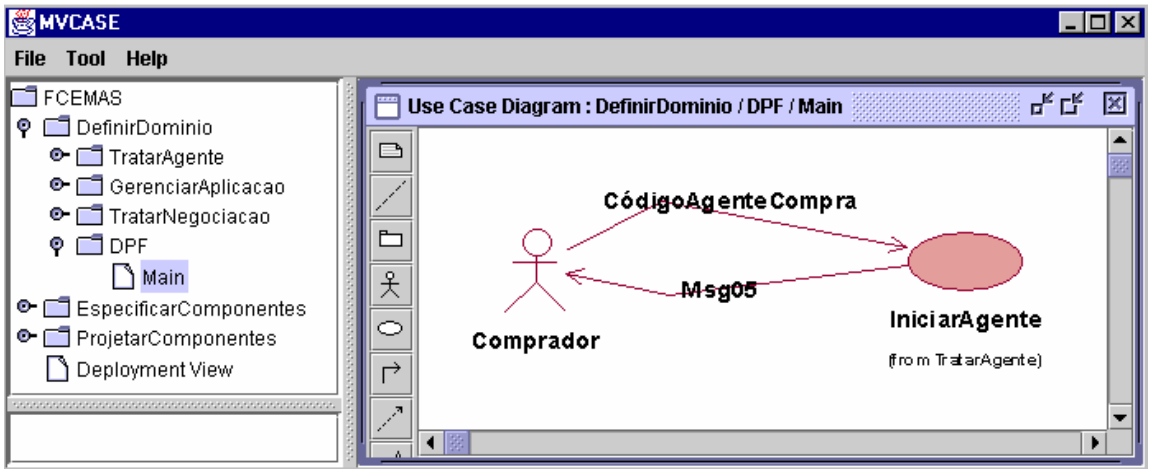


Diagrama de Caso de Uso

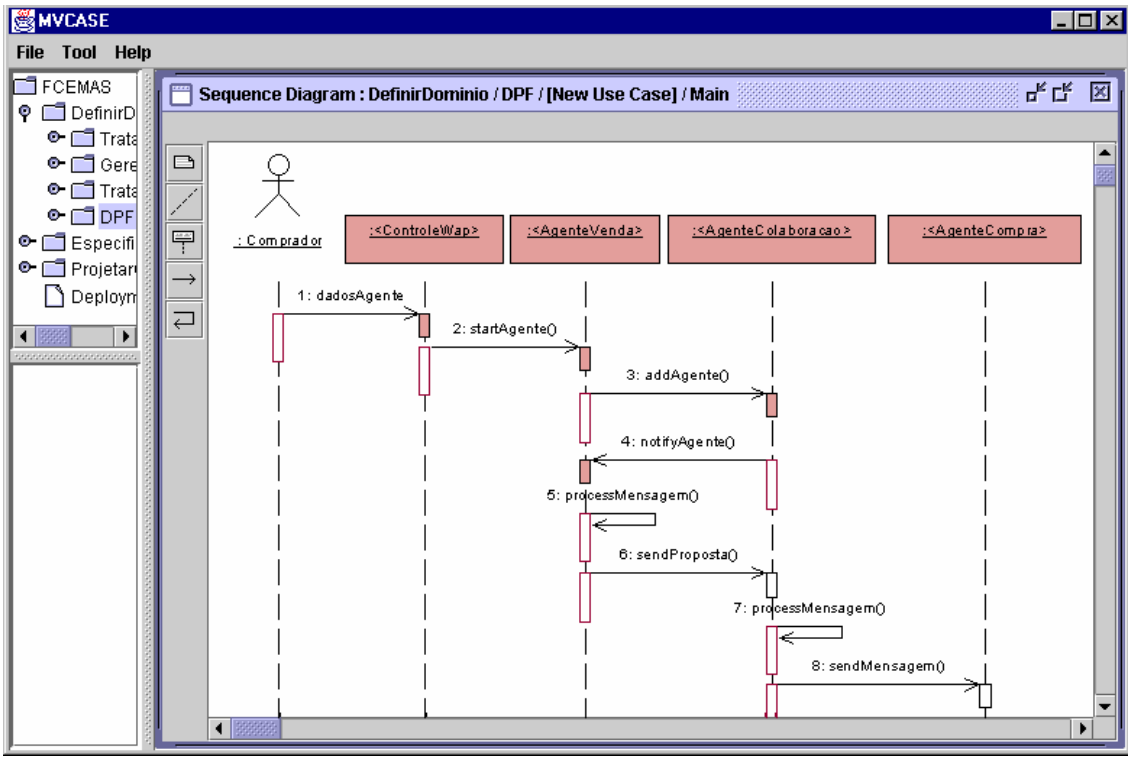


Diagrama de Seqüência

Figura 39– Diagrama de Caso de Uso e de Seqüência da Aplicação

Outras técnicas, como os Diagramas de Colaboração e de Estados, podem utilizados nesta fase de Especificação da Aplicação.

5.2 Projetar

Com base nas especificações obtidas do passo anterior, o Engenheiro de Software projeta a aplicação incluindo os requisitos não funcionais relacionados com: segurança da aplicação, arquitetura distribuída, e linguagem de implementação.

Neste passo, são também definidos os relacionamentos dos componentes, através de interfaces e dependências entre os componentes selecionados. Caso seja necessário, o Engenheiro de Software pode reutilizar os tipos definidos no FCEMAS e os componentes existentes ou criar novos componentes.

A Figura 40 mostra o Modelo da Aplicação, que representa a dependência dos componentes do domínio com os tipos da aplicação.

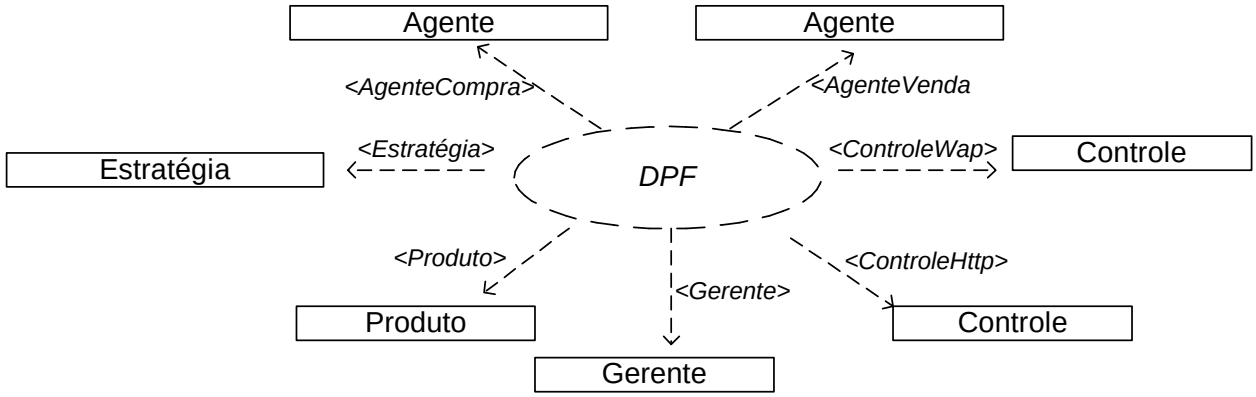


Figura 40 – Modelo da Aplicação

Em seguida, importa-se, na *MVCASE,* os componentes que serão reutilizados. A Figura 41 mostra no *browser* da *MVCASE* o relacionamento entre o pacote DPF e o reuso de alguns dos componentes criados para o desenvolvimento da aplicação.

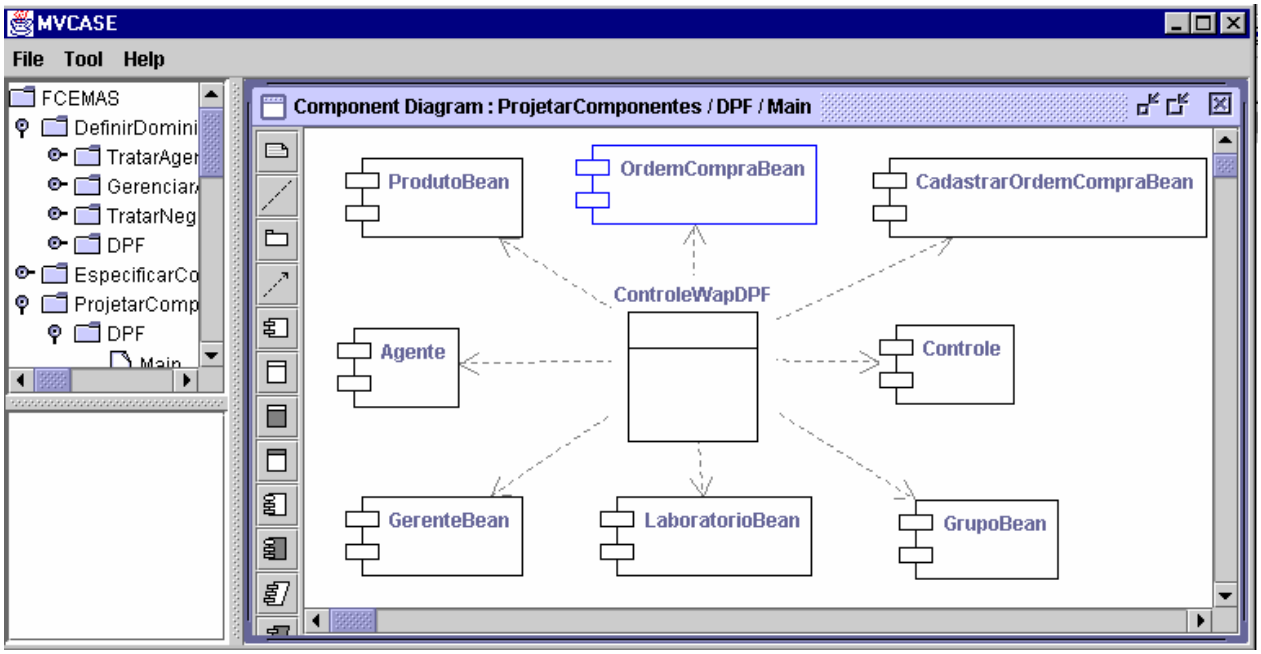


Figura 41 - Relacionamento entre os Componentes

Tendo projetado a aplicação, o Engenheiro de Software faz a sua implementação da aplicação, e em seguida faz o *deployment* dos componentes EJB da aplicação [Boo00].

5.3 Implementar

Com base nos componentes projetados na *MVCASE* no passo anterior, o Engenheiro de Software realiza a geração automática do código *Java*. A Figura 42 mostra, parte do código para a execução na *Web*, da classe `ControleWapDPF`.

A classe `ControleWapDPF` reutiliza os componentes do domínio de farmácia. Definidos a partir do FCEMAS e gerados pela *MVCASE*.

```
public class ControleWapDPF implements  IControleWap  {
    private GerenteHome gerenteHome;
   private. CadastrarOrdemCompraHome ocHome;
. . .
   public void init(ServletConfig config) throws ServletException {
            super.init(config);
            try         { // localiza os componentes EJB
              ctx = new InitialContext();
              Object obj1 = ctx.lookup("fcems.ejb.dbf.GerenteBean");
              gerenteHome = (GerenteHome) PortableRemoteObject.narrow(obj1, gerentetHome.class);
              Object obj2 = ctx.lookup("fcems.ejb.dbf.CadastrarOrdemCompraBean ");
              ocHome = (CadastrarOrdemCompraHome) PortableRemoteObject.narrow(obj2,CadastrarOrdemCompra.class);
            . . .
            } catch (Exception e) {
                                e.printStackTrace(System.out);
   }
   public void startAgente(int codGerente, int codAgenteCompra) {
            try{ // adiciono o agente no mercado virtual, para realizar a negociação
              . . .
              AgenteCompra agenteCompra = new AgenteCompra();
            ..agenteCompra.startAgente(codAgenteCompra);
            . . .
            } catch (Exception e) {
                        System.out.println(“Error startAgent --> ” +  e);
            }
            . . .
}
```

Figura 42 –Código da Classe ControleWapDPF

5.4 Testar

Neste passo, o Engenheiro de Software realiza os testes com a aplicação, para verificar se ocorreram erros. Os testes ficam facilitados considerando que os componentes reutilizados já foram previamente testados.

Para testar a aplicação adotou-se uma plataforma com Servidor Web *IBM HTTP Server*, Servidor de aplicação *IBM WebSphere*, banco de dados DB2, sistema operacional WindowsNT, *browser* Explorer e simulador WAP W3Gate.

A Figura 43 mostra a página *Web*, em que o usuário prepara a Ordem Compra para ser negociada pelo agente de compra, informado o preço máximo e mínimo para cada item.

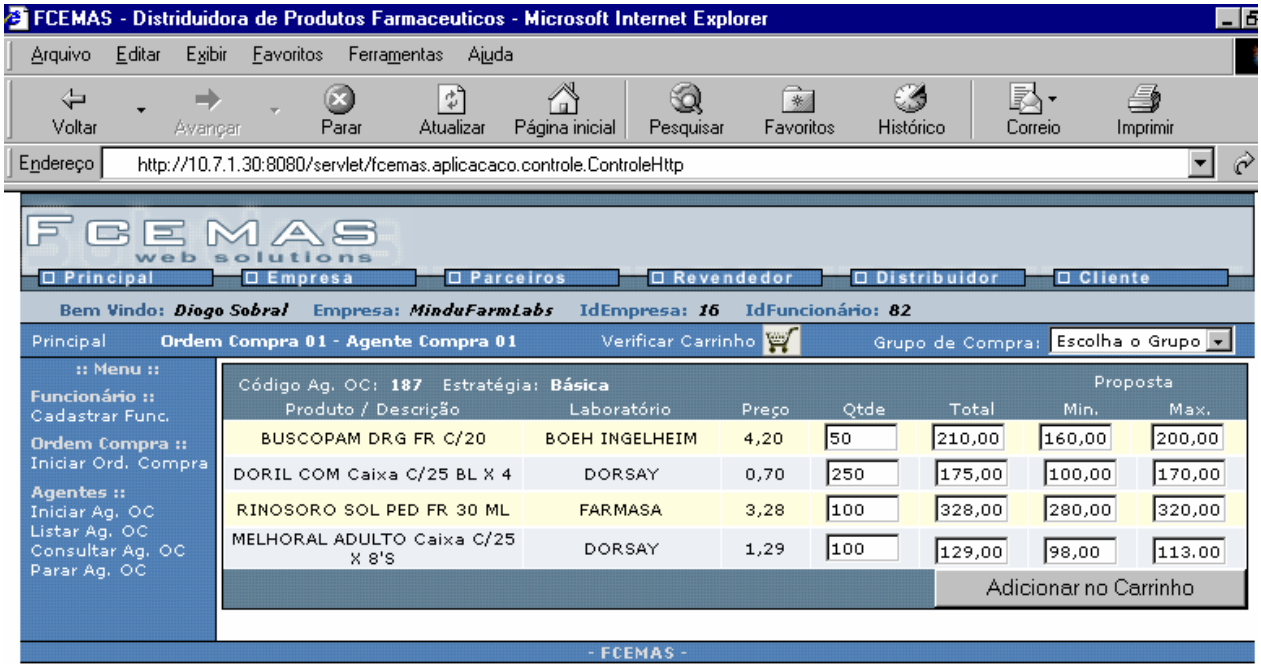


Figura 43 – Página da Aplicação - Ordem de Compra

Figura 44 mostra o ambiente de execução da aplicação, numa arquitetura de três camadas com a Aplicação Cliente, as Regras Negócio dos componentes EJB disponíveis pelos servidores e o Banco de Dados, onde são armazenados os dados da aplicação.

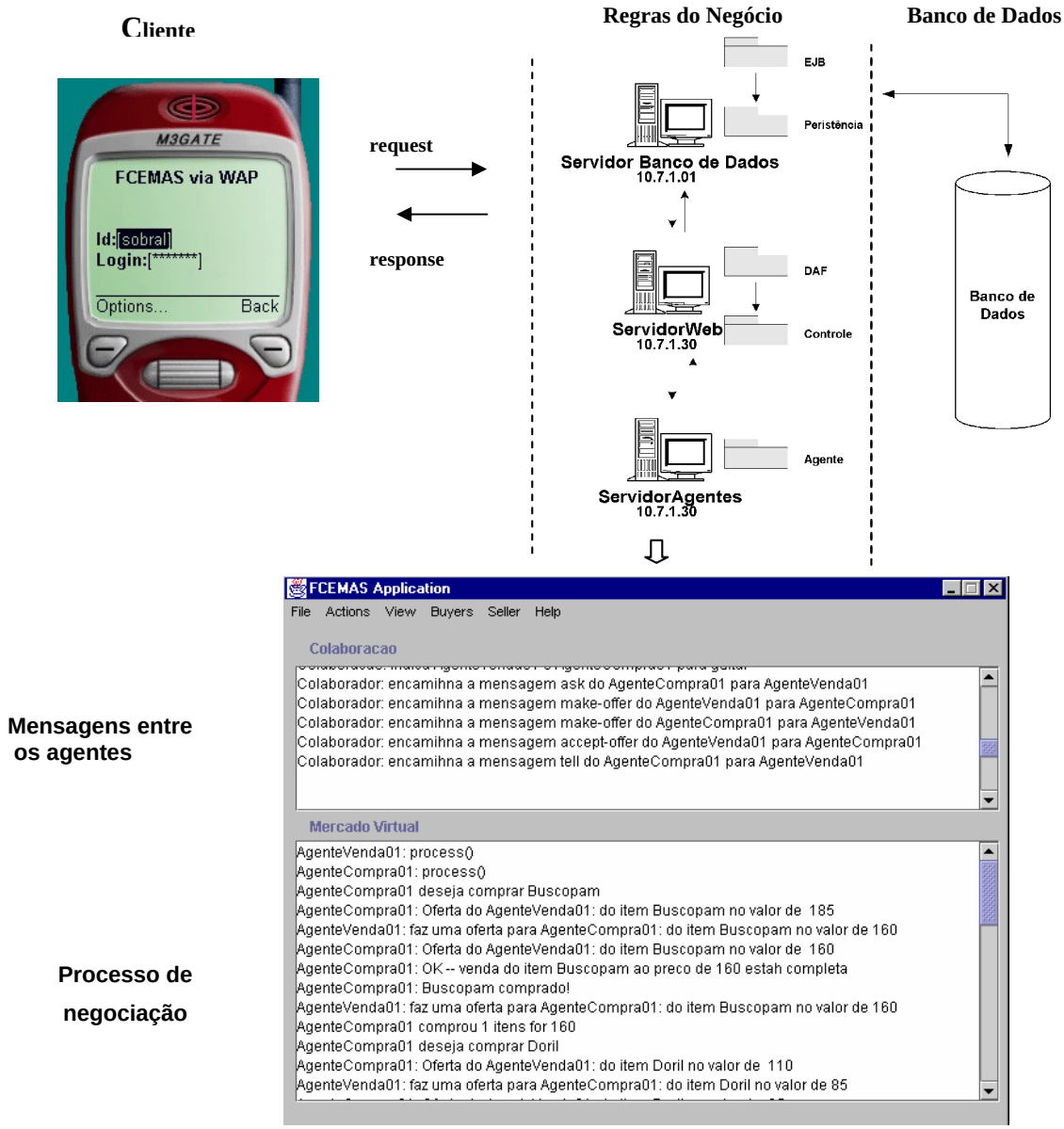


Figura 44 - – Teste da Aplicação Realizar Negociação

Conclusão

A Internet tem se mostrado um importante e crescente meio para a realização de negócios. Estudos realizados por Institutos de Pesquisa como *Forrester Reseach, International Data Corp. e Nielsen Media Research* [Gut99], demonstram que o número de pessoas que utilizam a *web*, para comprar ou vender tem crescido bastante. Contudo, existem poucas estruturas de suporte a esta nova forma de comércio, que hoje se resume, em grande parte, ao anúncio e encomenda de produtos.

Partindo destas idéias, este trabalho apresentou um *framework* para construção de sistemas para comércio eletrônico, via Internet Móvel, mediado por agentes de software, responsáveis por realizar as negociações.

A utilização do *framework*, como estrutura básica do desenvolvimento de uma aplicação, resulta numa economia de custos e tempo, além de facilitar a manutenção. A arquitetura multicamada facilita a integração do *framework* com os diferentes domínios de aplicações.

A reutilização de componentes, disponíveis nos diferentes domínios de aplicações de comércio eletrônico, combinado com a inteligência dos agentes de software, facilita o desenvolvimento das aplicações, principalmente de comércio eletrônico, que utiliza a vasta rede *Internet*, agora ainda maior com acesso via WAP.

O desenvolvimento de *frameworks* e componentes para domínios de aplicações demanda tempo, experiência e muita técnica em seu desenvolvimento, desde a análise do domínio do problema até a sua implementação e reutilização.

Frameworks e componentes devem ser bem documentados, para facilitar a sua utilização, nas aplicações. Para melhorar a documentação, utilizou-se *Catalysis* como método de desenvolvimento e técnicas UML para representar as especificações. A ferramenta *MVCASE* auxilia o Engenheiro de Software para produzir os diferentes artefatos que documentam o *framework* e os componentes.

As estratégias de desenvolvimento utilizadas orientam o Engenheiro de Software nas diferentes etapas do ciclo de vida do *framework* e dos componentes.

Agentes de Software foram utilizados para automatizar as tarefas do comércio eletrônico.

Apesar de o projeto estar direcionado para comércio eletrônico entre empresas, as técnicas apresentadas podem ser utilizadas em outros segmentos de mercado, como por exemplo para o comércio eletrônico entre empresa e consumidor.

Além das vantagens da reutilização de componentes, no desenvolvimento das aplicações, destaca-se a flexibilidade oferecida para os usuários, através da delegação de tarefas para serem executadas pelos agentes, diminuindo a necessidade de interação constante do usuário com a aplicação.

6.1 Principais Contribuições

As principais contribuições deste projeto de pesquisa são listadas a seguir:

a) Construção de um *framework*, FCEMAS, para comércio eletrônico;

b) Definição de uma arquitetura multicamada que integra o *framework* com componentes de diferentes domínios de aplicações, reutilizando os componentes conforme suas necessidades. Esta arquitetura diminui a redundância de código e facilita a manutenção;

c) Definição da estratégia para o desenvolvimento do *framework* e de componentes do domínio da aplicação de comércio eletrônico;

d) Comprovação da idéia de se utilizar agentes de software para facilitar o comércio eletrônico, diminuindo a necessidade de interação constante dos compradores e vendedores;

e) Utilização de padrões de projeto que auxiliam no desenvolvimento do *framework*.

6.2 Trabalhos Futuros

Seguindo o modelo espiral no processo de desenvolvimento de software [Pres00], algumas idéias são apresentadas para dar continuidade a este trabalho:

- Adicionar mecanismos de reputação ao *framework*, permitindo que futuras aplicações possam fazer uso da reputação de compradores e vendedores na avaliação de negociações.
- Adicionar aspectos de segurança ao *framework*. Esta é uma característica importante para *frameworks* de comércio eletrônico e não foi foco neste trabalho de pesquisa.
- Integração de um mecanismo de pagamento eletrônico, que permitirá a criação de aplicações com suporte à etapa de pagamento do comércio eletrônico.
- Aplicar o FCEMAS para atender aplicações para a comercialização de produtos distintos com a finalidade de identificar os principais atributos a serem modelados para os diferentes produtos. Este trabalho traria o aprendizado sobre como deve ser realizada a modelagem de produtos para aplicações de comércio eletrônico, nas quais agentes de software são utilizados. As novas aplicações permitirão avaliar as vantagens de se reutilizar componentes de *frameworks* comparado ao desenvolvimento de software tradicional.

6.3 Contribuições Acadêmicas

- Components for Eletronic Commerce using Software, Agents. Fontes, D.S, Bianchini, C. P., Prado, A. F., Sant'Anna, M., World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida, USA, 22-25 de Julho de 2001, Volume XVI, pág 246-251.

- Development of Applications for Information Research using Software Agents, Fontes, D.S, Bianchini, C. P., Prado, A. F., Sant'Anna, M., LAPTEC' 2001 Segundo Congresso de Lógica Aplicada à Tecnologia, Brasil, 12-14 de Novembro de 2001, Volume II, pág. 51-58.
- Agentes de Software Inteligentes Bianchini, C. P., Fontes, D. S., Prado, A. F., Sant'Anna, M.II Workshop on Artificial Intelligence - JCC'2001. Punta Arenas, Chile.05-09 de Novembro, 2001. Documentação em CD-ROM.

- *Framework* para Ensino a Distância mediado por Agentes de Software. Sanches, I., Fontes, D.S., Prado, A. F. INTERTEC'2001 VII Conferência Internacional de Educação em Engenharia e Tecnologia. Brasil, 17-20 de Março de 2002, a ser publicado em Fevereiro de 2002.

REFERÊNCIAS BIBLIOGRÁFICAS

[Ale77] ALEXANDER, C. A Pattern Language. Oxford University Press, 1977.

[Ale79] ALEXANDER, C. The Timelles way of building. Oxford University Press, 1979.

[Ame99] AMEC – Agent Mediated e-Commerce. http://ecommerce.media.mit.edu/. Acessado em Março de 1999.

[Bar99] BARRÉRE, T. S. *CASE com Múltiplas Visões de Requisitos de Software e Implementação Automática em Java - MVCASE*, Tese de Mestrado, UFSCar, 1999.

[Boo99] BOOCH, G; et. al.; 1999. The Unified Modeling Language – User Guide. Addison Wesley, USA.

[Bos97] BOSCH, J. Adapting Object-Oriented Components. In: ECOOP, 1997, Jyväskylä. Proceedings [S.l.]: Springer-Verlag, 1997.

[Bra77] BRADSHAW, J. M., *Sofware Agents*, AAAI Press/The MIT Press, 1997.

[Bus95] BUSCHMANN, F. et al. A System of Patterns. Chichester: John Wiley & Sons, 1995.

[Cag99] CAGNIN, M.I; Penteado R.S, "Reengenharia com Uso de Padrões de Projeto" XIII Simpósio Brasileiro de Engenharia de Software, Outubro 1999.

[Chv96] CHVE A.; Mães P.; "Kasbah: An Agent Marketplace for Buying and Selling Goods" Proceedings of the First International Conference on the Practical Application of Intellligent Agents and Multi-Agent Techonology (PAAM´96). London, UK, April 1996.

[Col94] COLEMAN, D; et. al.; 1994. Object-oriented Development – the Fusion Method. Prentice Hall, New Jersey.

[Cun87] CUNNIMGHAM, W.; Beck, K. Using Pattern Language for Object-Oriented Program. OOPSLA-87 workshop on the Specification and Design for Object-Oriented Programming.

[Dco96] DCOM; 1996. DCOM 1.0 Network Working Group. *Distributed Component Object Model Protocol.*

[Dso98] D'SOUZA, D.; WILLS, A. Objects, Components and Frameworks with UML – The Catalysis Approach. USA:Addison Wesley, 1998.

[Fay97] FAYAD, M.; SCHIMIDT, D. Object-Oriented Application Frameworks. Communications of the ACM, New York, v.40, n.10, p71-77, Oct. 1997.

[Fin96] FININ, T., Labrou, Y., Mayfield, J. KQML as an Agent Commnunication Language:. J. Bradshaw (ed.), Software Agents, MIT Press, 1996.

[Fer99] FERBER, J. *Multi-Systems: An Introduction to Distributed Artifical Intelligence.* Addison-Wesley, 1999.

[Fon01a] FONTES, D. S., Components for Eletronic Commerce using Software, Agents., World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida, USA, 22-25 de Julho de 2001, Volume XVI, pág 246-251.

[Fon01b] FONTES, D.S, Bianchini, C. P., Prado, A. F., Sant'Anna, Development of Applications for Information Research using Software Agents, M., LAPTEC' 2001 Segundo Congresso de Lógica Aplicada à Tecnologia, Brasil, 12-14 de Novembro de 2001, Volume II, pág. 51-58.

[Fow97] FOWLER, M.; 1997. UML Destilled - Applying the Standard Object Modeling Language. Addison Wesley, England.

[Fra96] FRANKLIN, S., GRAESSER, A., "*Is it an agent or just a program?: a taxonomy for automonous agents*", In: INTERNATIONAL WORKSHOP ON AGENT THRORIES, ARCHITECTURE AND LANGUAGES, 3, 1996. http://www.msci.memphis.edu/~franklin/AgentProg.html, acessado em 15/01/2000.

[Fuk00] FUKUDA, A. P.; 2000. Refinamento Automático de Sistemas Orientados a Objetos Distribuídos. Dissertação de Mestrado – Programa de Pós-Graduação em Ciência da Computação – Departamento de Computação – UFSCar, São Carlos, São Paulo – Brasil.

[Gam95] GAMMA, E., Helm, R., Johnson, R., Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented software.* Addison –Wesley, 1995.

[Gran99] GRAND, M. *Patterns in JavaI,* Addison –Wesley, 1999.

[Gut98] GUTTMAN, R. and P. MAES. *"Agent-mediated Integrative Negotiation for Retail Eletronic Commerce" Proceedings of the Workshop on Agent Mediated Eletronic Trading* (AMET'98). May 1998.

[Gut99] GUTTMAN, R., Maes P., Moukas, A., *Agents that Buy and Sell.* Communications the ACM, Vol. 42, Nº 3, Mar 1999.

[Har99] HAROLD, E.R. *"Java Network Programming"* O'Reilly & Associates, 2000

[Hun99] HUNTER, J.,GRAWFORD., W. *Java Servlet Programming. O'Reilly, 1999.*

[Ibm00] IBM Software; 1999. IBM San Franscico Framework. URL:http://www-4.ibm.com/software/ad/sanfrancisco/concepts.html. Acessado em Dezembro de 2000.

[Joh92] JOHNSON, R.; 1992. Documenting Frameworks Using Patterns. *Sigplan Notices – Trabalho Apresentado na OOPSLA*., New York, v.27, n. 10, Oct.

[Jon96] JOHNSON, R. "*How to Develop Frameworks"*. Tutorial Notes of ECOOP 96. Linz, Austria: July 1996.

[Joh98] JOHNSON, R.; FOOTE, B. "Designing Reusable Classes. *Journal of Object-Oriented Programming"*, [S.l.], v.1, n.2, p32-35, June/July 1998.

[Lar99] LARMAN, C, Utilizando UML e Padrões. Prentice Hall , Inc, 1999.

[Loj99] www.lojasamericanas.com.br. Acessado em Janeiro de 2000.

[Mae99] MAES, P. GUTTMAN, R. and A. MOUKAS. "*Agents that Buy and Sell: Transforming Commerce as we Know It*" Communications of the ACM, October 1999/Vo. 42, No.10.

[Mos99] MOSS, Karl. "*Java Servlets*." NewYork: McGray-Hill, 1999.

[Pre99] PREE, W.; SIKORA,H. "*Creation and Reuse of flexible bean architectures in business applications"*. Disponível por http://members.magnet.at/it-transfer/23_executive2.html em (20 nov. 1999).

[Pre00] PRESSMAN, R. S. Software Engineering: A Practitioner's Approach. 5th Edition, june 2000.

[Ral98] RALPH, E. JOHNSON and BRIAN FOOTE. "*Designing reusable classes.* Journal of Object-Oriented Programming", 1(2):22—35, Jun 1988.

[Rom99] ROMAN, E., Mastering Enterprise JavaBeans. Addison –Wesley, 1999.

[Ros77] Ross, Douglas T. Structured Analysis (SA): A language for Communicating Ideas. IEEE Transaction on Software Engineering, Jan 1977.

[Rum91] RUMBAUGH, J.; 1991. *Object-Oriented Modeling and Design.* Prentice-Hall, Englewood Cliffs.

[San00] SANCHES, I. C.; 2000. Framework para Ensino a Distância via Web. *Qualificação de Mestrado – Programa de Pós-Graduação em Ciência da Computação – Departamento de Computação – UFSCar*, São Carlos, São Paulo –Brasil.

[Sub99] www.submarino.com.br. Acessado em Janeiro de 1999.

[Sun01] SUN MICROSYSTEMS. "*Tutoriais Java"*, URL: http://www.java.sun.com, 2001

[Tal96] TALIGENT. "*Leveraging object-oriented frameworks"*. Taligent Inc. white paper. Disponível por http://www.taligent.com (18 mar. 1999).

[Tsy96] TSVETOVATYYM, M., Mobasher, B., Gini, M. And Wieckowski, Z. “MAGMA: An Agent-Based Virtual Market for Eletronic Commerce”, Dep. Of Computer Science University of Minnesota, Minneapolis, MN55455, 1996.

[Uol99] http://carsalel.com.br. Acessado em Janeiro de 1999.

[W3c97] W3C “*Extensible Markup Language*” World Wibe Web Consortiun (W3C) Working Draft, 1997.

[Wap01] WAP Forum, URL: http://www.wapforum.com, 2001.

[Xml01] XML. Extensible Markup Language (XML) 1.0 Second Edition. Disponível: site URL: http://www.w3.org/TR/2000/REC-xml-2000-10-06. Consultado em 10/07/2001.

[Yod98] YODER, J.M.; JOHNSON, R.E.; WILSON, Q.D. “Connecting Business Objects to Relational Database” Conference on the Pattern Languages of Programs, 5, Monticello-IL, EUA. Proceeding. 1988.

This document was created with Win2PDF available at http://www.daneprairie.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.