UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Ambiente Integrado de Modelagem Distribuída para Sistemas de Informação na Internet

por

LEANDRO BENTO POMPERMAIER

Dissertação submetida à avaliação, como requisito
parcial para a obtenção do grau de
Mestre em Ciências da Computação

Prof. Roberto Tom Price
Orientador

Porto Alegre, março de 1999.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Pompermaier, Leandro Bento
Ambiente Integrado de Modelagem Distribuída para Sistemas de Informação na Internet / por Leandro Bento Pompermaier - Porto Alegre: PPGC da UFRGS, 1999.

107p.:il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS,1999. Orientador: Price, Roberto Tom.

1. Engenharia de Software. 2. CASE. 3.Ambiente de Desenvolvimento 4.Editores Diagramáticos. I. Price, Roberto Tom. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Profa. Wrana Panizzi
Pró-Reitor de Pós-Graduação: Prof. Franz Rainer Semmelmann
Diretor do Instituto de Informática: Prof. Philippe Oliver Alexandre Navaux
Coordenadora do PPGC: Profa. Carla Maria Dal Sasso Freitas
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

"É melhor tentar e falhar que preocupar-se e ver a vida passar.
É melhor tentar, ainda que em vão, que sentar-se fazendo nada até o final.
Eu prefiro na chuva caminhar que em dias tristes em casa me esconder.
Prefiro ser feliz embora louco, que em conformidade viver..."
(Martin Luther King)

"We are drowning in information but
starved of knowledge" (John Naisbitt)

Agradecimentos

Durante o curso várias idéias foram revisadas, pesquisas foram realizadas, organizadas e dirigidas para que as informações tomassem forma e chegasse a conclusão desta dissertação. Sem o apoio do meu orientador, freqüentemente acreditando em mim e mostrando-se um lugar mais alto para se alcançar, não seria possível ter chegado até aqui. Por isto agradeço a orientação, amizade, respeito e contribuições valiosas para este trabalho ao prof. Roberto Tom Price.

Se em um caminho árduo e estressante como este não houvesse momentos de descontração e alegria, não conseguiria me manter são. Por isto agradeço a todos que riram de minhas piadas (às vezes até sem graça). Principalmente ao pessoal da Biblioteca, e a alguns colegas do mestrado que de certa forma "me aguentaram".

Somente uma boa conversa e boas polêmicas constróem amizades. Por isso gostaria de agradecer a Confraria da Quarta, em especial às meninas (Denise Stringhini, Isabel Harb Manssour, Simone Vicari e Magda Leyser) pelos momentos de alegria e beleza proporcionados pelo charme, inteligência e elegância.

Uma idéia, muitas vezes, não é como uma luz que se acende de repente do nada. Em alguns momentos parece que nem vai acender. Sem o incentivo na hora certa pode-se pensar até em trocar de lâmpada. Por isto agradeço a todos os colegas que de certa forma não deixaram que as idéias se apagassem.

Em toda caminhada é indispensável o apoio que se recebe das instituições dirigidas por pessoas que investem, acreditando, em nós. Por isto agradeço ao Instituto de Informática da UFRGS, ao CPGCC, aos professores e ao CNPq.

Mas também em toda a caminhada é preciso que as pessoas que nos acompanham olhem na mesma direção. Por isto agradeço aos meus pais, Cláudio e Cleusa, e a todos os meus familiares por me entenderem e apoiarem.

Sumário

Lista de Abreviaturas

ADS	Ambiente de Desenvolvimento de *Software*
AWT	*Abstract Window Toolkit*
BMP	*BitMaP*
CASE	*Computer Aided Software Engineering*
CSCW	*Computer Supported Cooperative Work*
CORBA	*Common Object Request Broker Architecture*
DD	*Data Dictionary*
EDI	Editor Diagramático para a Internet
ER	Entidade Relacionamento
HTML	*HyperText Markup Language*
GIF	*Graphics Interface Format*
JDK	*Java Development Kit*
JDBC	*Java Data Base Connectivity*
JFS	*Java File System*
JPEG	*Joint Photographic Experts Graphic*
MVC	*Model/View/Controller*
OMT	*Object Modeling Technique*
OOHDM	*Object-Oriented Hypertext Design Model*ing
RMI	*Remote Method Invocation*
SGML	*Standard Generalized Markup Language*
SII	Sistemas de Informação na Internet
URL	*Uniform Resource Locator*
WWW	*World Wide Web*

Lista de Figuras

Lista de Tabelas

Resumo

O objetivo principal desta dissertação explora alguns aspectos relacionados ao desenvolvimento colaborativo de sistemas de informação na Internet. É apresentado o Editor Diagramático na Internet (EDI), que suporta a especificação colaborativa de aplicações. Este editor utiliza tecnologia e funcionalidade dos hiperdocumentos, oferecendo características como: compartilhamento de informações, colaboração entre vários autores e várias visões dos dados conceituais armazenados. EDI foi implementado utilizando a linguagem de programação Java e projetada de forma genérica para permitir a criação de editores de diferentes notações diagramáticas.

Este trabalho propõe a utilização de anotações em documentos de desenvolvimento de sistemas de informação na Internet. Estas anotações auxiliam no desenvolvimento colaborativo de sistemas, tornando o processo mais colaborativo e com um produto resultante de qualidade superior. As anotações estão baseadas em dois tipos de usuários: o usuário proprietário, responsável pela criação do documento, e o usuário colaborador, que inclui anotações nos documentos. Anotações (identificadas por uma específica cor) podem ser de inclusão, alteração, remoção de conceitos (visões) ou registro de comentários.

Palavras-Chave: Engenharia de Software, Ambientes de Desenvolvimento de Software, WWW, Ferramentas CASE, Anotações, Autoria Colaborativa, Colaboração na WWW.

TITLE: "INTEGRATED ENVIRONMENT FOR DISTRIBUTED MODELING OF WEB INFORMATION SYSTEMS"

Abstract

The main goal of this work is to explore some issues related to collaborative development of information systems on the Internet. A Diagrammatic Editor on the Internet (EDI) that supports collaborative specification of applications is described. This editor uses hyperdocument technology and funcionalities, offering features such as information sharing, collaboration between several authors, multiple views of the stored conceptual data, among others. EDI was implemented using the Java language, and designed with the purpose of being generic to enable the easy creation of specific editors for different diagrammatic notations.

The use of annotations for the joint development of information systems on the Internet is proposed. With these annotations the development process becomes more collaborative and the quality of the final product may increase. Annotations are based on two types of users: the owner, who is an author responsible for the creation of a document, and the collaborator, who makes annotations on those documents. Annotations (identified by one specific colour) can be of inclusion, change and removal of concepts (views), or recording of comments.

Keywords: Software Engineering, Software Development Environment, CASE Tools, WWW, Annotations, Collaborative Authoring, and Collaboration on WWW

1 Introdução

A engenharia de *software* tem como objetivo viabilizar maior produtividade na construção de sistemas e qualidade dos artefatos de *software*. Diversas foram as metodologias que surgiram com estas perspectivas, fornecendo descrições de como o *software* deve ser criado e mantido [ORT95]. Em geral, estas descrições estão baseadas principalmente no conjunto de documentos produzidos durante o ciclo de vida, definindo a estrutura e os relacionamentos desses documentos.

O processo de desenvolvimento de *software* envolve uma contínua produção e transformação de notações, desde a especificação de requisitos até a produção de código executável [PER92]. Estas produções e transformações devem ser suportadas pelas ferramentas que fazem parte ou não de Ambientes de Desenvolvimento de *Software* (ADS).

Nas metodologias existentes, várias ferramentas e notações são utilizadas para atender as fases de desenvolvimento [PER92]. O que ocorre com estas ferramentas é o isolamento ou independência uma das outras, dificultando o relacionamento entre os documentos utilizados e/ou gerados durante este processo.

Atualmente, um dos grandes desafios para qualquer desenvolvedor de software é projetar um sistema que reutilize ao máximo códigos e projetos já existentes [TRE94]. A orientação a objetos foi crescendo como uma das respostas a este problema de reuso. Com a construção de *frameworks*, o reuso pode ser ainda melhorado, em domínios de aplicações específicos. Um *framework* é constituído por um conjunto de classes que abstraem as características gerais de um domínio de aplicação e codificam uma estrutura de controle que invoca métodos que devem ser implementados por subclasses.

Com o surgimento de novas tecnologias, como o ambiente *World Wide Web* (WWW), os aplicativos de *software* deverão satisfazer requisitos mais abrangentes [UDE96], utilizando arquiteturas de dados mais complexas envolvendo visualizações por *browser*, servidores de documentos e de bases de dados, sistemas de informações integrados ao *workgroup* das organizações e trabalho em grupo na geração dos documentos.

O ambiente WWW, capaz de representar hiperdocumentos de forma distribuída através da Internet, é baseado em *HyperText Markup Language* (HTML). HTML é uma linguagem baseada em marcas totalmente definida utilizando-se do padrão *Standard Generalized Markup Language* (SGML) - um padrão internacional para o armazenamento e intercâmbio de informações em ambientes de sistemas abertos.

Para toda esta filosofia, o paradigma de orientação a objetos tem se desenvolvido com a intenção de apresentar propostas mais rebuscadas que as dos paradigmas tradicionais. Suas promessas são: distribuir incrementos, cruzar uma extensa faixa de tecnologia e difundir a nova geração de arquiteturas de *software* [WIN93][MEY88].

Na Internet sistemas de informação são construídos fazendo acesso às bases de dados através de interfaces de navegadores (*browsers*). Através destas interfaces,

informações são sucessivamente manipuladas por diferentes usuários permitindo, ainda a estes usuários, diferentes visualizações destas informações.

O desenvolvimento de Sistemas de Informação na Internet (SII) requer, contudo, a modelagem de diversos componentes como, por exemplo, a modelagem estrutural, visual e caminhos de navegação dos documentos, bem como aspectos de atualização e acesso à bases de dados. Os SII requerem, também, a modelagem das bases de dados, pois essas possuem um vínculo direto com a base de documentos, peça fundamental de um SII.

Para modelar todos estes aspectos são necessárias ferramentas de edição desses modelos, ferramentas de consistência entre os modelos, ferramentas de transformação e dicionários de dados constituindo, assim, um ambiente de desenvolvimento de *software*.

Pode-se imaginar que, para desenvolver SII, seja conveniente que o ambiente de desenvolvimento de *software* também esteja baseado na Internet. Ambientes de desenvolvimento de *software* na Internet poderão apresentar as seguintes características: cooperação de autoria, coordenação entre usuários, diversas visualizações das informações manipuladas, segurança e ambiente aberto.

Um ambiente de desenvolvimento de *software* na Internet é, então, um ambiente de desenvolvimento que, além de suportar o ciclo de desenvolvimento de *software*, emprega mecanismos de cooperação entre os diferentes usuários podendo, assim, formar *workgroups* para o desenvolvimento de diversas tarefas.

Um ambiente de desenvolvimento de *software* na Internet possui uma base de dados que é comum a todos os usuários. Este ambiente pode garantir a coerência e a consistência das informações durante todo o ciclo de desenvolvimento garantindo, também, visualização dessas informações através de diferentes notações diagramáticas, conforme preferência do usuário.

1.1 Objetivos

O desenvolvimento de sistemas complexos normalmente é uma tarefa realizada por um grupo de técnicos especialistas que participam das distintas etapas do desenvolvimento do produto. Em cada fase do desenvolvimento são gerados diversos tipos de documentos. Essa criação, normalmente, é realizada com o compartilhamento de informações e cooperação entre tarefas em um ambiente onde usuários trocam idéias, experiências ou sugerem soluções para problemas específicos.

O presente trabalho pretende explorar o desenvolvimento colaborativo de sistemas de informação na Internet. Os aspectos a serem abordados podem resumir-se como:

⇨ analisar as características de um ADS na Internet;

⇨ identificar as principais funcionalidade de um ADS na Internet, para que este possa auxiliar o desenvolvimento colaborativo;

⇨apresentar uma experimentação na construção de ferramentas de apoio a um ambiente integrado de modelagem distribuída com suporte colaborativo a execução das tarefas encontradas nas diversas etapas do ciclo de desenvolvimento de *software*.

Especificamente, foi desenvolvido um protótipo de um editor de diagramas de notações para definição de sistemas de informação na Internet, com suporte ao desenvolvimento colaborativo de sistemas de informação na Internet.

1.2 Organização do Texto

Este trabalho está organizado da seguinte forma: no capítulo 2 são apresentados ambientes de desenvolvimento de *software*. São descritos, nesse capítulo, as principais características de ambientes de desenvolvimento, ferramentas de auxílio ao desenvolvimento e conceitos de *workgroup*, principalmente sua relação com o desenvolvimento de *software*.

No capítulo 3, são apresentadas algumas considerações sobre o processo de desenvolvimento de sistemas de informação na Internet. São discutidos conceitos que devem ser utilizados na especificação de um sistema de informação que utiliza o ambiente WWW.

No capítulo 4, são apresentados alguns conceitos que devem ser considerados na implementação de uma ferramenta de apoio ao desenvolvimento para a Internet como, por exemplo, conceito de visualização da ferramenta, armazenamento de dados e compartilhamento de informações. Neste capítulo, também é apresentado um levantamento de estado da arte sobre ambientes de desenvolvimento, ferramentas de apoio ao desenvolvimento e ferramentas divulgadas na WWW para a utilização na Internet.

No capítulo 5, são discutidas as características que um ambiente de desenvolvimento de sistemas de informação deve possuir. São mostradas as características e funcionalidades do editor colaborativo.

No capítulo 6, é apresentado o projeto e implementação do editor colaborativo que constitui em um ambiente de modelagem diagramática na Internet. São descritos aspectos de utilização e da estrutura da ferramenta como, por exemplo, cooperação e detalhes de implementação.

Finalmente, no capítulo 7 são apresentadas as conclusões do trabalho.

2 Ambientes de Desenvolvimento de *Software*

2.1 Ambientes de Desenvolvimento

Um ambiente de desenvolvimento de *software* (ADS) supre o usuário com ferramentas necessárias para sistematicamente desenvolver e dar manutenção ao *software* [GIM94][NAG96][PER91]. ADS referenciam produtos que tem como objetivo suportar ou automatizar a maior parte ou todo o processo de desenvolvimento, fornecendo ferramentas que assistam desde a fase de análise até a manutenção de uma aplicação de *software*.

O ciclo (ou processo) de desenvolvimento de *software* consiste em várias etapas bem definidas que, seguidas à risca, dão condições para a construção de *software* bem projetado e de fácil manutenção. É muito instrutivo imaginar o desenvolvimento de *software* como um ciclo em vez de um processo linear, como mostra a Figura 2.1. Os processos normalmente têm um fim, mas o desenvolvimento de *software* às vezes não. O sistema precisa, quase sempre, de uma atividade de manutenção constante e, se obtiver êxito, pode ser necessário desenvolvimento contínuo para ampliar suas possibilidades.

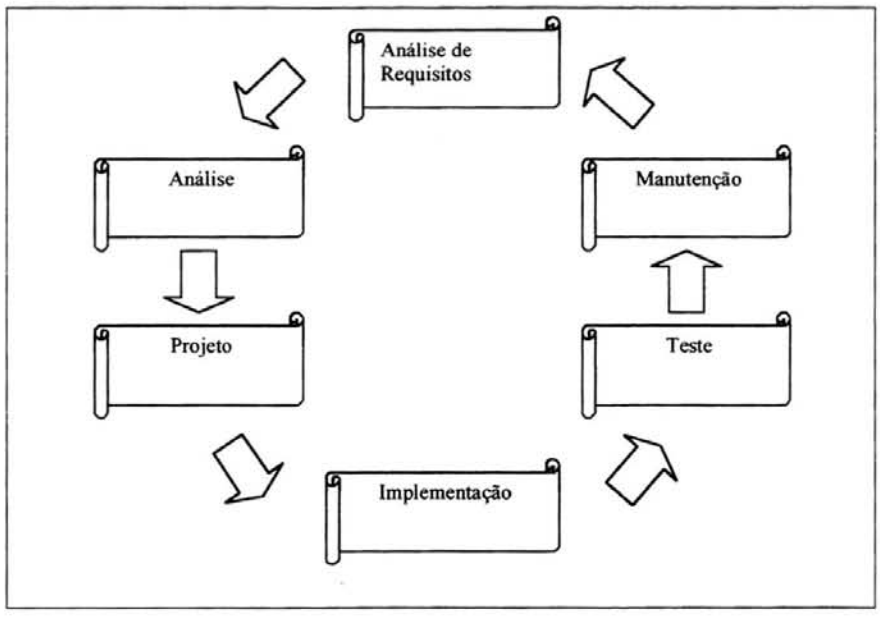

FIGURA 2.1 - Ciclo de Desenvolvimento de *Software*

Os primeiros ADS foram criados essencialmente para assistir a fase de programação fornecendo ferramentas como compiladores, depuradores, editores, etc. Porém com o avanço da programação gráfica, ferramentas que suportam métodos gráficos de análise e projeto foram desenvolvidas. Estes ambientes permitiram a produção de *software* de complexidade cada vez maior, gerando novos requisitos que os ambientes deveriam satisfazer.

Por isso, o conceito de ambiente de desenvolvimento envolve todo o processo de desenvolvimento como uma unidade, assistindo na totalidade do ciclo de vida do *software* e fornecendo ferramentas para todas as etapas deste ciclo.

2.1.1 Classificação dos ADS

Diferentes abordagens têm sido utilizadas para a classificação dos ADS. Em [PEN88] é apresentada uma proposta contemplando as diferenças e similaridades existentes nos ambientes, enquanto que em [DAR87] é apresentada uma taxionomia baseada nos direcionamentos tecnológicos que geraram o estado-da-arte em ambientes.

Segundo [PEN88], existem quatro considerações para classificarmos os ambientes:

➢Tecnologia Utilizada

Essa abordagem é a mais simples e está centrada no sistema operacional. Tem sido, usualmente, utilizada como uma forma dos fabricantes tornarem seus ambientes mais acessíveis para os usuários.

Como uma extensão natural aparecem os ambientes orientados por sintaxe ou estruturas e linguagens. Estes ambientes introduzem uma camada a mais entre o usuário e os elementos de programação do sistema operacional, oferecendo uma interface composta de ferramentas como editores orientados por sintaxe, depuradores, analisadores, etc.

Outra classe destes ambientes está baseada na tecnologia dos sistemas de gerência de banco de dados. A base de dados é utilizada como um repositório global das informações do ambiente, mantendo-se controle das sucessivas transformações realizadas pelas ferramentas do ambiente.

A tendência atual é a utilização da tecnologia baseada em objetos. As informações são modeladas como relacionamentos entre os objetos e mantidas de forma persistente numa base de dados. Associado ao conceito de objetos estão usualmente a manipulação de classes definidas pelo usuário que possibilitam o encapsulamento de propriedades e comportamentos, organizadas de tal forma a permitir um relacionamento de herança.

➢Características Desejadas

Ambientes de desenvolvimento de *software* devem possuir um conjunto de características desejadas, como as que seguem :

Utilidade: Deve atender aos diferentes tipos de usuários (programadores, analistas, projetistas, gerentes) e, também, atender as diversas atividades do ciclo de vida do *software* (análise dos requisitos, projeto, programação, testes).

Usabilidade: O ambiente deve ser "amigável" possuindo, em suas ferramentas, uma interface consistente, de fácil adaptação e aprendizado.

Adaptabilidade: O ambiente deve ser capaz de se adaptar às características diferenciadas de projetos e preferências individuais de usuários.

Grau de Automação : Algumas atividades do ambiente devem ser capazes de serem realizadas de forma automática, sem interferência do usuário. Por exemplo, geração de código a partir da especificação ou a geração do banco de dados a partir das definições do modelo de dados.

Grau de Integração : As diferentes ferramentas e dados do ambiente devem estar integrados para permitir consistência, tanto das tarefas, como da base de dados.

Valor : Relação custo (investimento, tempo de desenvolvimento, tempo de aprendizagem de novas metodologias) benefício (produtividade incremental, qualidade).

➢ **Suporte para o Desenvolvimento do *Software* Alvo**

Um ambiente de desenvolvimento de *software* deve suportar as características do *software* a ser desenvolvido. As principais características são:

Domínio da Aplicação : Refere-se ao tipo de aplicação que podem ser desenvolvidas no ambiente.

Propriedades Analisáveis : Refere-se ao tipo de propriedades da aplicação como, por exemplo, funcionalidade, eficácia, manutenibilidade, entre outras.

Usuários : Refere-se ao tipo de usuário (analista, programador, projetista)

Atividades do Ciclo de Vida : Refere-se às diferentes fases do ciclo suportadas pelo ambiente (análise dos requisitos, projeto, modelagem dos dados,...)

Atividades Técnicas : Refere-se às atividades de manipulação e representação da aplicação.

➢ **Estrutura Básica da Arquitetura**

A arquitetura identifica os componentes do ambiente, seus relacionamentos estáticos e dinâmicos. Assim, destaca-se:

Máquina Virtual : Esta arquitetura organiza o ambiente em níveis, cada qual com um objetivo específico dando suporte às tarefas dos níveis superiores, através de um processo de comunicação estabelecido em protocolos.

Rede : Esta arquitetura organiza os componentes como uma rede de processos, estabelecendo a comunicação através de um sistema de mensagens.

Centrada em Dados : A organização do ambiente é centrada num núcleo principal estabelecido num repositório de dados. As ferramentas comunicam-se entre si através de fluxos de dados sobre o núcleo.

Centrada em Controle : O núcleo central do ambiente é o sistema de supervisão. As ferramentas integram-se através de fluxos de controle sobre o núcleo.

Segundo [DAR87] os ambientes podem ser classificados como :

↳Ambientes Centrados em Linguagens

Em geral estão formados por um conjunto de ferramentas especificamente projetadas para serem utilizadas no desenvolvimento de *software* sobre uma linguagem particular. São ambientes adequados à fase de programação e, em geral, não fornecem suporte a programação de grande escala.

São exemplos deste tipo de ambiente : InterLisp e SmallTalk.

↳ Ambientes Orientados a Estruturas

Estes ambientes incorporam técnicas que permitem ao usuário manipular as estruturas diretamente, utilizando editores especificamente construídos (editores dirigidos por sintaxe). O editor é o componente central, sendo que é através dele que o usuário interage com as estruturas definidas. Este tipo de ambiente manipula o código através de representações estruturais do mesmo, independentemente das linguagens de implementação, o que deriva na noção de *geradores de ambientes*.

A capacidade destes ambientes de suportar várias fases do processo de desenvolvimento está restringida pela utilização de uma representação física baseada em árvore de sintaxe. Isto dificulta a integração de diferentes ferramentas e o relacionamento de ambientes com componentes estruturais que não podem ser descritos pelo tipo de gramática utilizada para especificação.

↳Ambientes Caixa de Ferramentas (Toolkits)

Esses ambientes consistem de uma coleção de pequenas ferramentas, primeiramente utilizadas na fase de codificação. Essas ferramentas são fracamente vinculadas e facilmente extensíveis pela incorporação de novas ferramentas.

Esses ambientes começaram como uma extensão do sistema operacional, disponibilizando ferramentas como editores, compiladores ou depuradores, bem como ferramentas para suportar tarefas de desenvolvimento de *software* em larga escala.

Os intercâmbios de informações entre ferramentas estão baseados na interação por compartilhamento de arquivos. Esta forma de interação é geralmente *não estruturada* e necessita de conversões explícitas para que as ferramentas possam compartilhar a informação (por meio de operações de importação e exportação).

Os *toolkits* não impõem nenhum tipo de restrição sobre o processo que o usuário segue ou deve seguir. O usuário interage através de uma interface de propósito geral (interpretador de linguagem de comandos), que lhe permite decidir livremente que ferramenta ativar, isto é, não fornece nenhum controle sobre a forma em que as ferramentas são utilizadas.

↳Ambientes Baseados em Métodos

Esses ambientes são desenvolvidos para suportar métodos (ou metodologias) específicos de desenvolvimento de sistemas podendo estar orientados a determinados métodos que suportem determinadas fases do ciclo de vida, assim como métodos para gerenciar o processo de desenvolvimento.

São características desses ambientes interfaces gráficas e textuais, suporte a diferentes ferramentas, onde cada uma está orientada a suprir uma fase do ciclo de desenvolvimento. A integração destas ferramentas se dá através de um repositório de dados. Este tipo de ambiente ficou conhecido como *Computer-Aided Software Engineering* (CASE).

Da mesma forma que ambientes orientados a estruturas, existem ambientes desta classe que permitem a especificação da metodologia. Estes ambientes são denominados *meta-ambientes* [ORT95]. O componente comum e central a estes ambientes é o editor que fornece a interface básica de interação com o usuário. Um meta-ambiente para geração de ambientes baseados em métodos deve fornecer um meta-editor, o qual gera instâncias de editores com conhecimento específicos de cada técnica envolvida no método que está sendo definido.

2.2 Ambientes Orientados a Objetos

Segundo [MEY94] um ambiente orientado a objetos deve possuir os seguintes princípios :

• Abstração de Dados (*Data Abstraction*): um ambiente orientado a objetos deve definir um conjunto de abstrações para permitir a modelagem dos objetos; para cada abstração do ambiente deve existir uma ferramenta correspondente, em que o usuário aplicará as regras ou comandos apropriados ao tipo do objeto associado.

• Manipulação Direta (*Direct Manipulation)* : Define uma propriedade de relação entre o ambiente e sua interface. Esse princípio define a manipulação direta dos objetos que estão envolvidos na modelagem.

• Consistência Semântica (*Semantic Consistency*) : Este princípio assegura a uniformidade do ambiente. Um ambiente orientado a objetos disponibiliza aos usuários um conjunto de símbolos (textuais ou gráficos) para a representação dos objetos em uma determinada interface. Por exemplo, o usuário seleciona um objeto através do seu símbolo e aplica qualquer operação que seja semanticamente válida, não se preocupando com o contexto do símbolo como ferramenta, localização, formato ou representação.

• Tipagem Forte (*Strong Typing*) : Em um ambiente orientado a objeto um tipo é associado a todos os objetos do desenvolvimento, assegurando que cada tipo seja apresentado na representação visual do objeto. Este princípio determina que operações são possíveis para cada objeto, e se alguma operação inválida for acionada para um determinado objeto o ambiente deverá notificar o erro.

• *Redemption* : O ambiente deve possuir um construtivo *feedback*. Quando o usuário acionar uma operação erroneamente e o erro somente poderá ser detectado após a execução da operação, o ambiente não reportará ao usuário o erro e possibilitará o cancelamento da operação, mas sim apresentar ao usuário, quando semanticamente possível, uma versão anterior a ocorrência do erro gerada por uma operação válida.

2.3 Ferramentas CASE

O projeto, implementação, desenvolvimento e manutenção do *software* são atividades complexas e de alto custo, que requerem controle e melhorias constantes. Uma das tecnologias propostas para alcançar estes objetivos são as ferramentas CASE (*Computer Aided Software Engineering*) [VES95] [ZAR90].

CASE é uma tecnologia que utiliza ferramentas automatizadas de forma integrada, permitindo que o uso de métodos formais de desenvolvimento torne-se prático e econômico [VES95]. Através de ferramentas CASE os analistas e programadores obtêm uma melhor visualização dos requisitos, especificações e projetos do *software*, além de um acompanhamento automático nos refinamentos sofridos pelo *software* desde a fase de especificação de requisitos até a implementação permitindo, com isso, que a completeza e a consistência sejam verificadas automaticamente.

Pode-se definir o termo CASE como qualquer *software* e/ou *hardware* que auxilia na especificação, definição e construção de *software* ou simplifica a comunicação entre as partes envolvidas no processo de desenvolvimento [ZAR90].

Na década de 70, vários métodos formais e semi-formais foram desenvolvidos, na maioria baseados em diagramas, no intuito de tornar o processo de desenvolvimento de sistemas menos anárquico e mais formal. Diagramas auxiliam a clareza de raciocínio das pessoas favorecendo a comunicação entre as equipes de desenvolvimento, como também a documentação e depuração dos programas [MAR82]. Entretanto, são extremamente difíceis de criá-los e mantê-los à mão. As ferramentas CASE foram desenvolvidas justamente para suportar interfaces gráficas com as quais analistas e programadores interagem desenhando diagramas, descrevendo e definindo funções e dados, relacionando e identificando componentes do sistema.

2.3.1 Vantagens das Ferramentas CASE

Segundo [MAR82] as ferramentas CASE possuem as seguintes vantagens :

→ Especificações dos requisitos

As ferramentas CASE proporcionam uma especificação dos requisitos do sistema, pois obrigam os projetistas a utilizarem uma metodologia. Estas metodologias de especificação impõem o envolvimento do usuário final, a fim de que o sistema final esteja de acordo com as especificações desejadas pelo usuário.

→ Especificações minuciosas do projeto

Exige que seja feita toda a documentação do projeto, até mesmo alterações devido a manutenção. Enfatiza as estruturas em módulos desprezando estruturas muito grandes e difíceis de manter.

→ Especificações atuais de projeto

As ferramentas CASE oferecem dispositivos para manterem as especificações do projeto atualizadas em face a manutenção contínua do *software*, aprimoramentos e evolução do código.

→ Redução do Tempo de Desenvolvimento

Com o auxílio à especificação do projeto, as ferramentas CASE devem ser tão atraentes que seus usuários - analistas e projetistas - desejem usá-las antes de iniciarem a implementação.

→ Código altamente flexível e de fácil manutenção

Nenhum projeto de *software* bem sucedido chega ao fim. Os usuários finais vão exigir aperfeiçoamentos funcionais ou identificarão erros na operação do *software*, tornando-se necessária alguma forma de desenvolvimento contínuo ou de serviço de manutenção. As ferramentas CASE proporcionam este tipo de ambiente, auxiliando na manutenção do *software* e tornando-o altamente flexível.

Em resumo, as ferramentas CASE de alta qualidade, pois:

- são atraentes;
- auxiliam no projeto do *software;*
- mantêm a sincronização com o código-fonte;
- reduzem os riscos de falhas dos sistemas;
- reduzem o tempo total de desenvolvimento.

2.3.2 CASE e Hipertexto

Cada vez mais Ambientes CASE estão se tornando maiores e mais flexíveis, pois estão incorporando novos conceitos ao seu funcionamento como, por exemplo, o conceito de *hipertexto* [BIG88].

Hipertexto foi primeiramente conceituado como uma forma de armazenar pequenas informações, podendo acessá-las de forma arbitrária [CYB92]. Também pode ser definido como um mecanismo de criação e representação de vínculos entre peças

discretas de dados, podendo ser conceituada como um modelo entidade-relacionamento que armazena e recupera informações estruturadas.

Ferramentas CASE podem ser estruturadas utilizando sistemas de hipertextos com o conceito de gerenciamento das informações envolvidas no ambiente CASE. A Tabela 2.1 mostra a relação dos hipertextos com as ferramentas CASE.

TABELA 2.1 - Características de Hipertextos e Ferramentas CASE [CYB92]

Hipertexto	**CASE**
Autoria de Documentos	Editor de Diagramas Ferramentas Orientadas a Texto
Navegação (*Browsing*)	Criação
Agregação de Documentos	Bibliotecas Dados Estruturados
Estruturas Virtuais	Geração de Código
Computação Dinâmica	Resultados *RunTime*
Trabalho em Grupo	Desenvolvimento em Grupo
Extensibilidade / Tolerância a Falhas	Múltiplas Metodologias
Verificação de Consistência	Validação Verificação

Em hipertexto, a informação é fragmentada em várias páginas e estas são agrupadas e "ligadas" de forma que o navegador possa recuperá-las sem qualquer ordem pré-estabelecida. Estendendo este conceito para os ambientes CASE podemos acomodar um conjunto de ferramentas CASE integradas garantindo coerência, consistência e flexibilidade durante todo o ciclo de desenvolvimento do *software*.

Ambientes CASE com este tipo de conceito (hipertexto) baseiam-se em um conjunto de ferramentas para, de forma incremental e interativa, construir descrições do *software* que está sendo desenvolvido. Porém os problemas que devem ser considerados são [CYB92] :

- utilizam diferentes classes de documentos em cada fase do processo de desenvolvimento;
- os documentos produzidos por uma ferramenta devem ser utilizados por outras ferramentas em processos posteriores;

- os documentos descrevem faces distintas da mesma realidade, portanto o sistema deve permitir a navegação entre os documentos inter-relacionados permitindo diferentes visões da mesma informação.

Existem vários aspectos que podem ser afetados pela incorporação de um sistema de hipertexto a uma ferramenta CASE. Segundo [ORT95], estes aspectos são:

→ Interface com o usuário

A tarefa principal de um sistema de hipertexto é proporcionar ao usuário o fornecimento da informação de forma clara e interativa podendo ser navegado (*browsing*) através de sua estrutura de elos e nodos (documentos e/ou subdocumentos).

→ Integração de Documentos

A necessidade de integrar e utilizar documentos e arquivos de diversos tipos é característica do desenvolvimento de *software* suportado por ferramentas CASE. Os sistemas que combinam hipertexto e CASE ressaltam a importância de integrar todos os documentos com uma base de dados.

→ Representação de Informação Semi-Estruturada

Uma considerável quantidade de informação no desenvolvimento de sistemas, principalmente nas primeiras fases de desenvolvimento é muito informal e ainda difusa. As propriedades de vinculação do hipertexto permitem que esta informação possa ser semanticamente relacionada em formas arbitrárias devido ao modelo básico de nodos e elos.

→ Suporte ao Trabalho Cooperativo

O desenvolvimento de *software* pode ser visto como um esforço cooperativo de uma organização. Portanto, um ADS deve dar suporte ao trabalho cooperativo para proporcionar o desenvolvimento integrado de sistemas não sendo mais somente um conjunto isolado de ferramentas, mas sim uma ferramenta capaz de suportar a coordenação de grandes grupos de trabalho.

Segundo [ORT95] outro aspecto que deve ser observado é o suporte que os sistemas de hipertexto podem oferecer às atividades de reutilização de *software* e engenharia reversa.

2.3.3 Problemas das Ferramentas CASE

Apesar das vantagens que as ferramentas CASE apresentam no aumento da produtividade, elas apresentam várias limitações que reduzem estas vantagens. Entre algumas desvantagens podem ser citadas as mencionadas em [MAR82] :

- as ferramentas CASE são confeccionadas para suportar um conjunto pré-definido de técnicas diagramáticas; logo, podem não suportar técnicas que o usuário deseje e necessite para determinadas aplicações;

- a maioria das ferramentas CASE atuais não possuem facilidades para configuração automática dos documentos e quando possuem não permitem que o usuário faça ajustes estéticos que desejar;

- a primeira geração de ferramentas CASE somente enfatiza a interação com o usuário e em sua maioria são ineficientes na produção de documentação impressa;

- as ferramentas CASE são difíceis de se integrarem com outros ambientes, pois são construídas de maneira verticalizada.

2.3.4 Ferramenta CASE Orientada a Objetos

Uma ferramenta CASE orientada a objetos suporta o desenvolvimento automatizado das fases de análise, projeto e implementação orientado a objetos [KNU94]. O propósito desta integração é a representatividade dos modelos de análise, projeto e implementação em um modelo totalmente orientado a objetos.

Uma típica ferramenta CASE orientada a objetos disponibiliza editores diagramáticos para serem utilizados nas etapas citadas anteriormente. Muitas ferramentas que existem são projetadas utilizando-se de conceitos variantes da análise estruturada e projeto estruturado. Estas ferramentas utilizam um dicionário de dados como interface entre as ferramentas que compõem a CASE.

O desenvolvimento orientado a objetos pode ser integrado por meio de um modelo orientado a objetos [KNU94]. Isto é possível porque os modelos gerados pela análise, projeto e implementação possuem uma mesma representação lógica, consistente e de completeza.

2.4 Trabalho Cooperativo

Em ambientes de desenvolvimento tradicionais, os sistemas, em sua maioria são desenvolvidos por ferramentas isoladas [SOU96]. A disponibilidade da tecnologia de redes de computadores e a necessidade de compartilhamento de recursos e informação impulsionam as pesquisas na área de atividades cooperativas, isto é, desenvolvimento de suporte ao trabalho em grupo.

Atualmente, os computadores são considerados ferramentas de trabalho essenciais, pois além de apoiar a realização de tarefas individuais, tornam-se importantes meios de interação e cooperação entre pessoas [BOR95] [GRU95].

2.4.1 CSCW e Groupware

Trabalho cooperativo suportado por computador, mais conhecido como CSCW (*Computer Supported Cooperative Work*) tem como objetivo proporcionar o suporte computacional para que as pessoas possam interagir cooperativamente [GRU91].

CSCW é um campo multidisciplinar que estuda sistemas computacionais em uma organização e que integra atividades de processamento e comunicação.

O termo *Groupware* costuma ser utilizado quase como sinônimo de CSCW. Geralmente o termo *groupware* é utilizado para designar ferramentas para CSCW [ELL91] [KHO95]. Esta tecnologia surgiu da união dos computadores, das bases de informações e da tecnologia de comunicação com a possibilidade de suportar ou não a cooperação.

2.4.2 Objetivo do Trabalho Cooperativo

Esquematicamente, podemos colocar o objetivo do trabalho cooperativo suportado por computador como o de assistir grupos em atividades de :

Comunicação: baseada em computador ou mediada por computador, que é apoiada pelas telecomunicações e pela tecnologia de processamento computacional;

Colaboração: que demanda o compartilhamento de informações ;

Coordenação: onde a comunicação e a colaboração são realçadas se as atividades de grupo são coordenadas.

A colaboração e a comunicação podem ser melhoradas se existir coordenação. Sem coordenação, podem ocorrer conflitos proporcionando redundância na execução das tarefas.

2.4.3 Características de CSCW

Prover suporte por computador para cooperação é difícil e requer um grande entendimento de como os grupos e organizações trabalham [GRU91]. Deste modo, Borges em [BOR95] descreve que os principais requisitos de CSCW são :

- o sistema deve facilitar a cooperação entre os indivíduos, ao invés de impor práticas que causem mudanças radicais na forma de trabalho;

- sistemas CSCW devem reconhecer que mudanças são freqüentes neste contexto e que, por isso, devem ser capazes de permitir a redefinição de procedimentos e processos, além de disseminar estas mudanças entre os participantes;

- a construção de aplicações menores e interrelacionadas é preferível ao desenvolvimento de aplicações monolíticas que incluam o conjunto completo de tarefas;

- informações que serão usadas no trabalho cooperativo devem se situar dentro do domínio do grupo de participantes.

2.4.4 Taxonomia de Aplicações CSCW

Basicamente, existem duas abordagens gerais sobre sistemas de CSCW. A primeira e mais utilizada é a abordagem de desenvolvimento de sistemas que suportam a troca de informação entre usuários. A segunda, consiste no desenvolvimento de sistemas que exploram o compartilhamento de informações para permitir cooperação.

Freqüentemente essas técnicas são combinadas formando cinco classes de sistemas de CSCW: sistemas de mensagens, conferência via computador, salas de reunião, sistemas de co-autoria e argumentação e sistemas de *workflow*. A Tabela 2.2 resume a principal abordagem utilizada por cada tipo de sistema.

TABELA 2.2 - Aplicações de CSCW e suas respectivas Abordagens [SOU96]

Sistema	Abordagem
Sistema de Mensagens	Troca de Informação
Sistema de Conferência via Computador	Troca de Informação
Sistema de Reunião	Compartilhamento de Informações
Sistemas de Co-Autoria e Argumentação	Compartilhamento de Informações
Sistema de *Workflow*	Compartilhamento de Informações

Sistema de Mensagens : Corresponde ao popularmente conhecido "correio eletrônico", onde ocorre troca de mensagens de forma assíncrona entre usuários. É, atualmente, a única aplicação *groupware* bem sucedida, ou seja, largamente utilizada nas organizações.

Sistema de Conferência via Computador : Envolve comunicações assíncronas e síncronas (incluindo teleconferência), permitindo a troca de informações acerca de um determinado assunto. Os *bulletin boards systems* são exemplos de sistemas de conferência, tanto síncrona quanto assíncrona (dependendo da utilização). A teleconferência (ou vídeo-conferência) utiliza redes de computadores de alta velocidade para a transmissão da imagem e da voz dos participantes.

Sistema de Reunião : Foram desenvolvidos para apoiar diferentes tipos de trabalho em grupo face-a-face. As reuniões possuem um papel muito importante nas organizações atuais e, com o auxílio de computadores, podem ser bastante facilitadas.

Sistema de Co-Autoria e Argumentação : São, em geral, uma classe de sistemas que suporta e representa a negociação e argumentação envolvida no grupo de trabalho. São ferramentas que permitem elementos de grupos construírem um objeto conjuntamente, seja um gráfico ou um texto. São geralmente assíncronos. Um exemplo deste tipo de aplicação é o processador de textos IBM/Lotus WordPro onde um documento pode ser construído por vários autores.

Sistema de *Workflow* : São sistemas que têm por objetivo automatizar os processos de trabalho. Estes sistemas tentam automatizar e substituir o processamento baseado em papel através de formulários digitalizados circulando através de redes de computadores.

2.4.5 Incorporação de Trabalho Cooperativo nos ADS

A partir do estudo da funcionalidade dos sistemas cooperativos é possível determinar que a melhor forma de incorporar suporte para trabalho cooperativo em um ADS é através da formalização do processo de desenvolvimento de *software*. A partir disto surge o conceito de ADS cooperativo que envolve os conceitos de um ADS tradicional com as facilidades e vantagens de um CSCW.

Segundo Vessey e Sravanapudi [VES95] ADS's cooperativos são compostos por três camadas básicas de ferramentas classificadas de acordo com o seu grau de especialização (Figura 2.2): *taskware, teamware* e *groupware*.

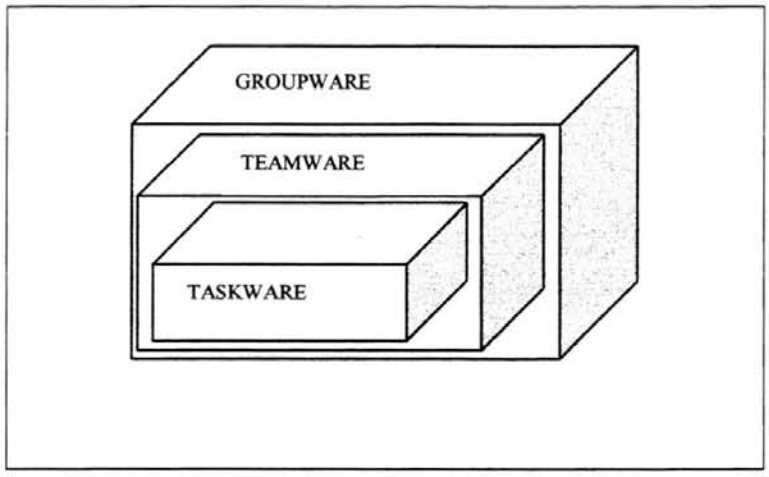

FIGURA 2.2 - Arquitetura de ADS's cooperativos [VES95]

Taskware refere-se ao suporte à realização da tarefa principal de um ADS: permitir o desenvolvimento de *software* através da utilização de diagramas e linguagens de acordo com uma metodologia e com assistência automatizada. A maioria das ferramentas CASE atuais são *taskware*.

Teamware provê suporte específico ao trabalho em equipe para qualquer tarefa que compartilhe os elementos construídos durante o ciclo de desenvolvimento. É encarregado da coordenação das atividades das equipes de desenvolvimento, envolvendo o compartilhamento de dados (diagramas, especificações, código-fonte, etc.) e monitoramento.

Groupware provê suporte à comunicação. Suporta as necessidades de tarefas independentes de uma equipe de trabalho como, por exemplo, a comunicação direta com outros membros da equipe ao invés da comunicação indireta a partir dos elementos compartilháveis. Deste modo, poderia ser provido por pacotes genéricos como, por exemplo, correio eletrônico, *bulletin board system* ou, ainda, ferramentas mais sofisticadas como o Lotus Notes.

3 Processo de Desenvolvimento de Sistemas de Informações na Internet

Um Sistema de Informação na Internet pode ser visto como um sistema de *workflow* de documentos tendo como interface para clientes *browsers* e acessando bases de documentos e de dados em servidores distribuídos [POM98a]. Basicamente, o funcionamento de um SII pode ser visto como o gerenciamento do ciclo de vida de hiperdocumentos, estruturados e dinâmicos, que compartilham uma base de dados comum. Em SII, diversos tipos de documentos são gerados, atualizados e examinados, em parte ou em todo, por diversos tipos de usuários.

Para projetar um SII é necessário modelar os documentos que serão manipulados, a base de dados que conterá os dados compartilhados pelos vários documentos e as transações que serão realizadas pelos diversos usuários do SII. Para desenvolver um SII será necessário projetar a arquitetura do *software*, o que significa projetar os objetos da aplicação (de interface, de controle e de conteúdo), decidir sobre a utilização de bibliotecas de componentes e *frameworks* de diversos domínios, estabelecer os sistemas de gestão de base de dados e de documentos que serão utilizados, especificar a interação destes componentes com outros componentes como *browsers* e sistemas de notificação de eventos através de diversos protocolos de comunicação.

A implementação destes SII pode ser programada em diversas linguagens, como Java [COR96][FLA97] e Smaltalk[PAR94]. A integração destes componentes pode ser feita com diversos mecanismos como *Common Object Request Broker Architecture* (CORBA)[NAU98], *Remote Method Invocation*(RMI)[NEW96], *Java DataBase Connectivity* (JDBC) [SHR96] e outros.

Um ambiente para o desenvolvimento de SII deve ser um gerenciador de documentos distribuídos na *World Wide Web* (WWW). Este ambiente deve suportar um ciclo de desenvolvimento em que diversos documentos são gerados, atualizados, examinados e utilizados por diferentes usuários, trabalhando em diversas localidades. Este tipo de ambiente deve ter a capacidade de suportar a troca de informações entre grupos de usuários formando, assim, um *groupware* de desenvolvimento.

Este capítulo examina o processo de desenvolvimento de sistemas de informações na Internet. Estes sistemas são baseados em navegadores (browsers) Web utilizando sistemas de banco de dados, tecnologia e funcionalidades dos hiperdocumentos. Geralmente, a programação é realizada utilizando linguagens distribuídas e orientadas a objetos como, por exemplo, Java, utiliza também, *frameworks* de aplicações e bibliotecas de componentes. Diversos requisitos de integração como JDBC, CORBA são utilizados.

3.1 Componentes para Modelagem de Sistema de Informação na Internet

Na fase de análise são levantados os requisitos do sistema, definidos os documentos, a base de dados, as transações, a organização do *workflow,* os tipos de usuários e as restrições do sistema (Figura 3.1). Nesta seção são apresentados algumas considerações para a modelagem destes conceitos.

FIGURA 3.1 - Tipos de Requisitos do Desenvolvimento de *Software* para SII

3.1.1 Definição dos Documentos

Documentos são a base dos SII podendo ser visualizados, navegados e manipulados através de *browsers* (navegadores) que oferecem interface padrão, portabilidade e acesso distribuído. Os documentos possuem uma estrutura de composição e de navegação [SCH95].

A definição dos documentos deve representar os aspectos estruturais e comportamentais que serão mapeados, em fases posteriores, para o ambiente de implementação como estruturas de classes, base de dados e sistemas de manipulação de documentos do tipo HTML com seus métodos. Nesta fase não há preocupação com questões de implementação.

O analista deverá definir a estrutura de composição e estrutura de navegação dos documentos do *software* a ser desenvolvido. Quanto a composição deverá levar em consideração as seguintes características :

➔ Composição dos Documentos (HTML ou outras técnicas de representação)

Estruturas Estáticas (apenas dados a serem vistos pelos usuários): imagens, tabelas, gráficos, textos.

Estruturas Dinâmicas (ações que são executadas quando ativadas): applets, acesso a base de dados, Common Gateway Interface (CGI), ferramentas (OLE).

As aplicações feitas para WWW devem ser projetadas para realizar a navegação através do espaço de informações. A navegação é uma característica deste tipo de aplicação em que o usuário pode percorrer as informações que se apresentam na tela do computador [SCH95] [ROS95].

Quanto à estrutura de navegação devem ser identificados:

→ **Nós** : recipientes de informação básica

→ **Elos** : relacionamento entre os nós

Estáticos : relacionamento entre nodos, como links (HTML)

Dinâmicos : executam ações (geração automática de páginas, acesso a base de dados, ativação de programas, ativação remota de métodos – RMI, execução de applets)

→ **Estruturas de Acesso** : Menus, índices, roteiros guiados (caminhos de navegação), etc.

Documentos serão base da interação de qualquer sistema a ser desenvolvido, podendo ser considerado como a visão que o sistema oferece dos dados armazenados. O analista deverá definir qual a estrutura e a composição destes documentos. Para isto deverá levar em consideração o que é necessário, para os usuários finais, visualizarem nas diversas etapas da execução do sistema definindo, desta forma, diversas visões para os documentos.

Os conceitos utilizados na modelagem dos documentos são apresentados na Figura 3.2. Basicamente podemos considerar que um documento é uma agregação de documentos e/ou de vários componentes (elos, textos, imagens, tabelas, applets, e outros).

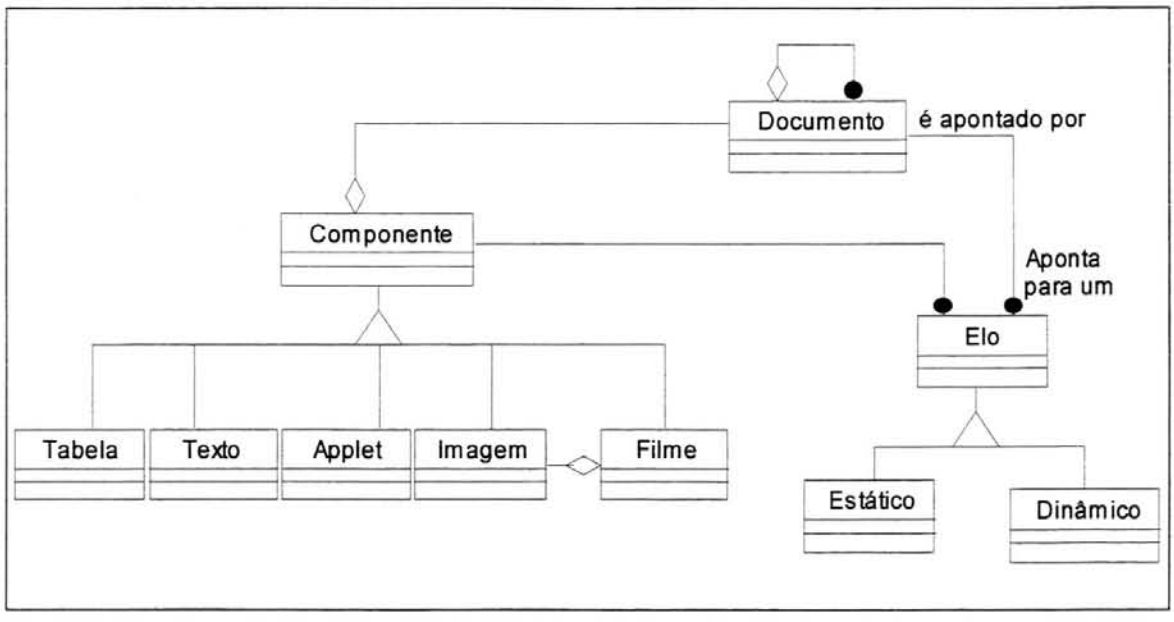

FIGURA 3.2 – Modelo Conceitual para Composição de Documentos (notação OMT[RUM91]), baseada em [GRA96], com modificações

Para a definição dos documentos pode-se fazer uso de metodologias como, por exemplo, OOHDM (Figura 3.3) [SCH95][ROS95], HDM [GAR93], etc. Estas metodologias procuram fornecer um modelo de estruturação mais natural para as

informações através de classes que definem a estrutura e de suas instâncias (nós) que efetivamente mantêm a informação. Desta forma, é possível associar maior semântica às unidades do hiperdocumento e, por conseqüência, dar suporte a mecanismos mais sofisticados de consulta e organização dos dados, além de facilitar a reutilização e manutenção.

A metodologia OOHDM utiliza mecanismos de abstração e composição numa arquitetura orientada a objetos que permite uma descrição concisa das informações complexas e uma especificação de padrões de navegação e interface [SCH95].

No OOHDM uma aplicação é construída em um processo de quatro passos para suportar um modelo incremental ou de protótipo. São eles : projeto conceitual, projeto navegacional, projeto de interface abstrata (Figura 3.3) e implementação. A Tabela 3.1 mostra os quatro passos e as informações relevantes de cada um.

TABELA 3.1 - Metodologia OOHDM

Atividades	Produtos	Formalismo	Mecanismos
Projeto Conceitual	Classes, Sub-sistemas, relacionamentos, perspectivas de atributos	Modelagem orientada a objetos	Classificação, agregação, generalização e especialização
Projeto Navegacional	Nós, Links, estruturas de acesso, contexto navegacional	Views, State charts, classes de contexto	Classificação, agregação, generalização e especialização
Projeto de Interface Abstrata	Objetos de interface abstrata, respostas a eventos externos, transformações de interface	Abstract Data View (ADV), diagramas de configuração (figura 3.3), ADV-charts	Mapeamento entre navegação e Objetos perceptíveis
Implementação	aplicação		

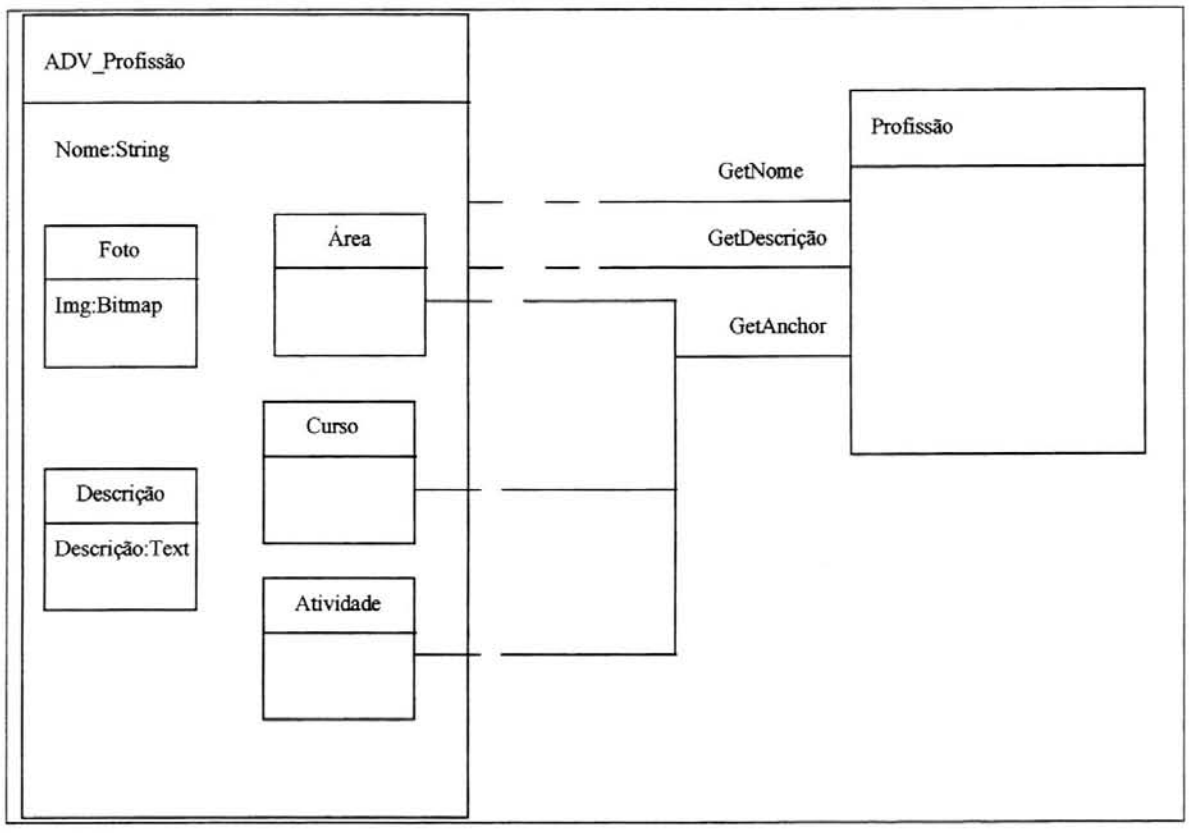

FIGURA 3.3 - Exemplo de Definição de Documentos utilizando OOHDM [SCH95] - Diagrama de Configuração

3.1.2 Modelagem da Base de Dados

Os documentos (em HTML) de uma aplicação podem armazenar todos os dados da aplicação. Isto acarreta, no entanto, problemas de acesso, integridade dos dados, redundância e segurança. Por isto, usualmente, os documentos contêm referências às bases de dados que armazenam as informações. Para a modelagem da base de dados de um SII é necessário definir a base de documentos, a base de dados e a integração destas duas. Para a modelagem dos dados poderá ser utilizado o modelo Entidade-Relacionamento (ER) [MAR82][CHE76], representando as entidades, atributos e relacionamentos que serão mantidos ou suas extensões, inclusive modelagem orientada a objetos.

A Figura 3.4 apresenta um modelo de objetos utilizados para conceituar os componentes da modelagem de base de dados.

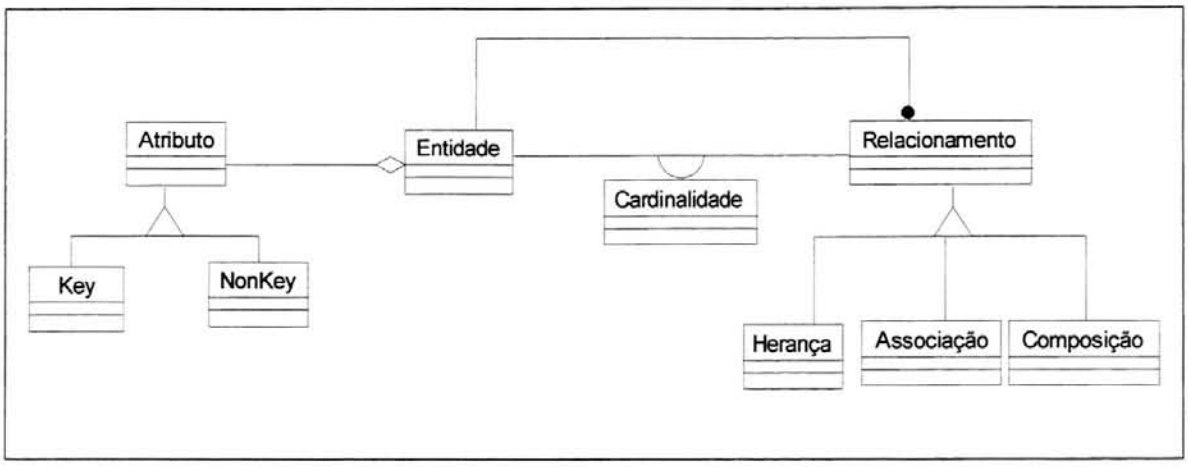

FIGURA 3.4 – Modelo Conceitual da Base de Dados

A modelagem dos dados pode ser realizada utilizando o diagrama de entidade-relacionamento (Figura 3.5) [MAR82] [CHE76] ou modelo de objetos [RUM91]. Estes diagramas possuem componentes como : objetos de dados, atributos, relacionamentos (composição, associação, herança) e vários indicadores como cardinalidade e opcionalidade. O exemplo apresenta instancias da classe Entidade (*usuário, localidade, especialidade e área*), da classe Relacionamento, não possuindo instâncias das outras classes.

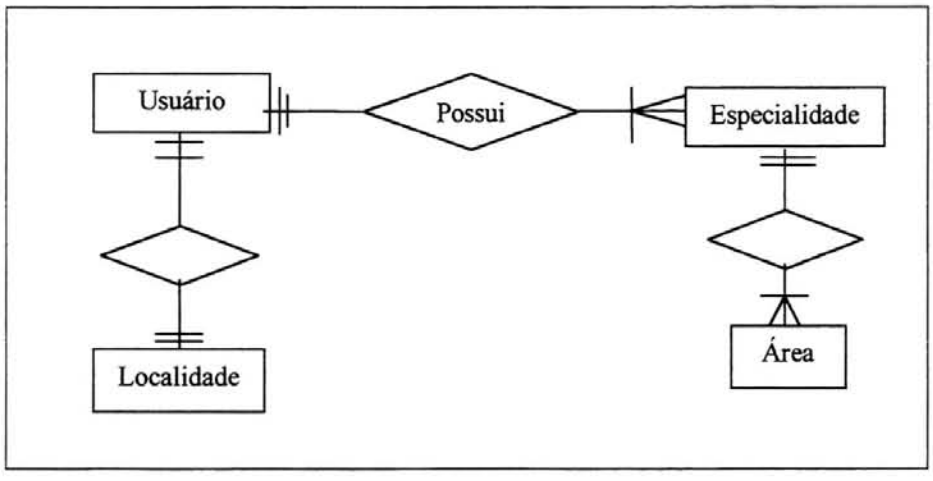

FIGURA 3.5 - Exemplo simples de diagrama ER

3.1.3 Transações

A especificação de um SII deverá definir quais serão os tipos de usuários do sistema e as transações nas quais estes participam. Para realizar esta tarefa poderão ser utilizadas notações para descrição de processos como, por exemplo, as apresentadas em [JAC92], [YOU92] , entre outras. Estas notações devem ser estendidas para suportar o conceito de documentos.

O modelo com os conceitos que devem ser modelados na especificação das transações é apresentado na Figura 3.6. Este modelo de objetos apresenta a relação das transações com o fluxo de documentos e o acesso às bases de dados e quais os principais eventos ativados por uma transação.

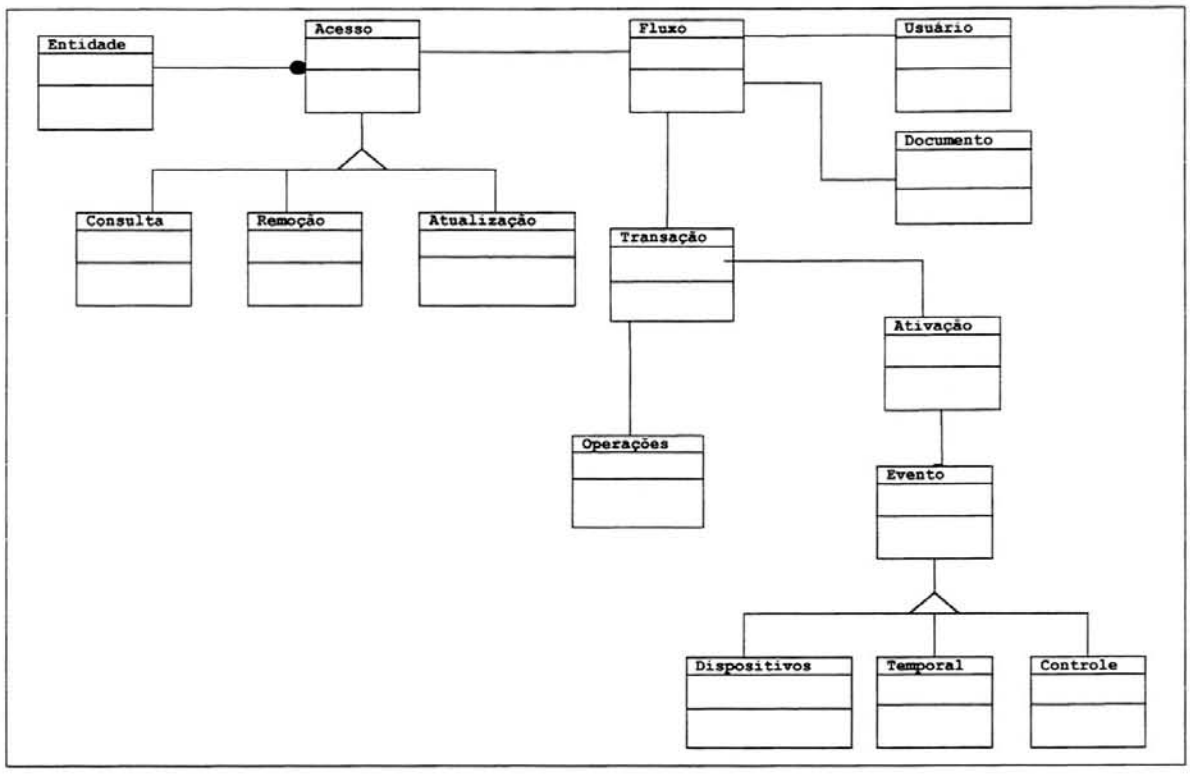

FIGURA 3.6 – Modelo Conceitual para Representação de Transação

A Figura 3.7 mostra a modelagem de uma determinada transação **T**, a qual possui contextos de entrada (**agente**) que pode ser um usuário de tipos definidos (pessoa, *software* ou dispositivo). Esta transação recebe na entrada uma informação **a**, que é manipulada e armazenada na entidade **A.** Por outro lado, uma informação **g** é recuperada da entidade **B.** Após ser manipulada é gerado um documento com esta informação.

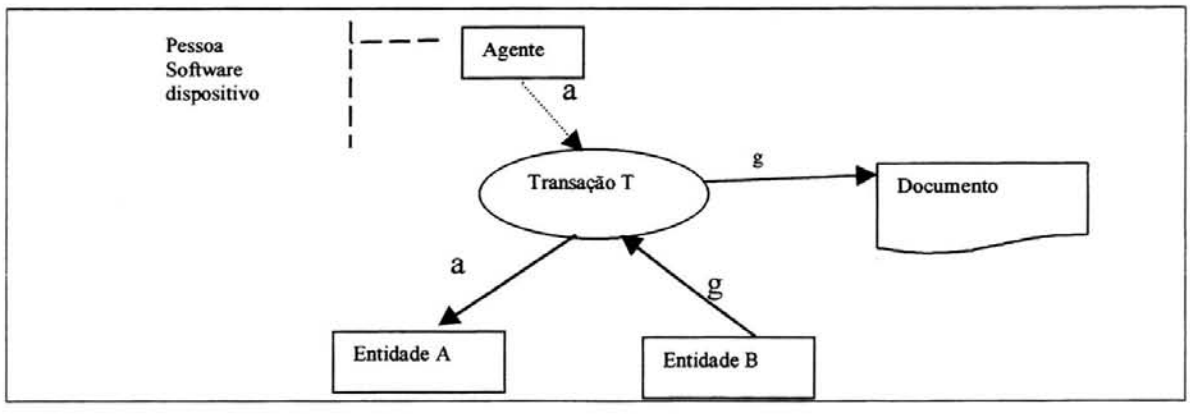

FIGURA 3.7 - Exemplo de modelagem de processos estendido

3.1.4 Modelo de Workflow

Um Workflow é definido como uma coleção de tarefas organizadas para realizar um processo (por exemplo, processar uma ordem de pedido), podendo ser representado através de uma definição de processo [GRA96]. Um processo consiste em uma rede de atividades e seus relacionamentos, critérios de início e término do processo e informações sobre atividades em si como participantes, aplicativos e dados de sistema de informação relacionados.

Alguns conceitos devem ser considerados na modelagem do *workflow* (Figura 3.8) :

- Evento : Algo que ocorre, algo que acontece;
- Ator : Alguém ou algo que atua;
- Atividade : Um conjunto de eventos que ocorre sob a responsabilidade de um ator;
- Gatilho (*trigger*) : Um evento **e** desencadeia a atividade **a** se a ocorrência de **e** causa a realização de **a**;
- Papel (*role*) : Um papel é um conjunto de capacidades que especificam quando um ator é qualificado para ser destinado a uma atividade;
- Processo : É um conjunto de atividades que compartilham um mesmo repositório.

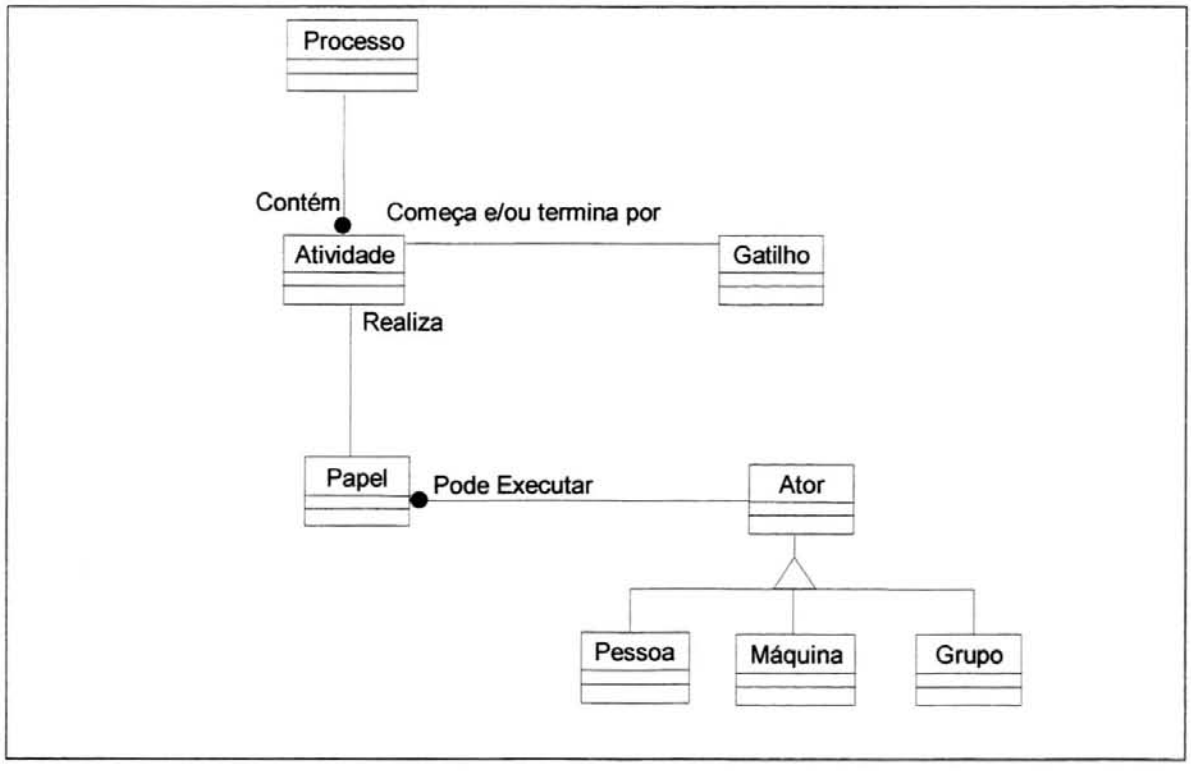

FIGURA 3.8 – Modelo Conceitual de Workflow

Para a definição do workflow para as transações que serão realizadas durante o processo do SII, bem como a definição do fluxo de dados e eventos que podem ocorrer durante o mesmo, pode-se utilizar a notação do sistema de gerenciamento de *workflow* (*workflow management system*) como, por exemplo, um workflow do tipo ad hoc [GEO98](Figura 3.9)

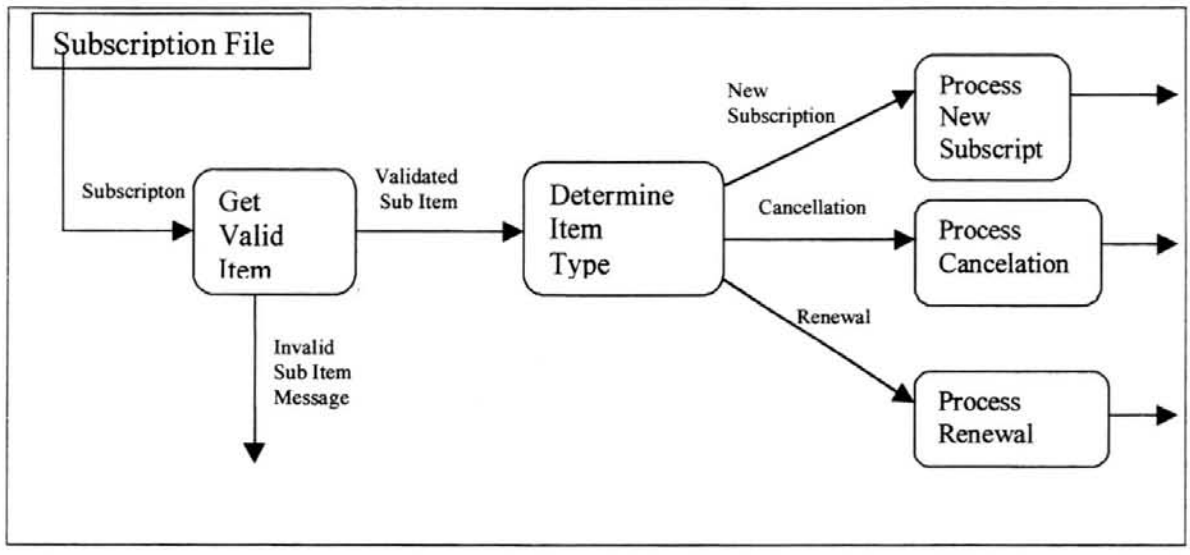

FIGURA 3.9 - Exemplo de um *workflow*

3.1.5 Restrições

A especificação de um SII requer, também, a definição de restrições do SII como, por exemplo, restrições quanto ao desempenho, segurança, custos e tempos de desenvolvimento. Para um sistema na Internet deve-se ter um cuidado especial com relação a estes aspectos, principalmente com a segurança. As restrições são apresentadas em forma de texto, de equações matemáticas ou por tabelas.

3.2 Ambiente de Apoio ao Desenvolvimento de SII

Na especificação de um SII é necessária a modelagem de diversos conceitos utilizando diversas notações. Pode-se realizar esta tarefa utilizando ferramentas específicas para cada conceito, isso no entanto, pode acarretar perda de tempo, de desenvolvimento e de integração dos modelos.

Ambientes integrados que agregam diversas ferramentas de modelagem (abrangendo todos os conceitos analisados anteriormente) são uma solução para o desenvolvimento de SII. Utilizando-se do ambiente WWW para sua execução podendo ser acessado por diversos usuários para a execução de suas tarefas, diminuindo assim o tempo de desenvolvimento e facilitando a integração de todos os modelos.

Ambientes integrados possuem alguns benefícios [PRE95] :

- Transferência harmoniosa de informações (modelos, programas, documentos, dados) de uma ferramenta para outra e de uma etapa da engenharia de *software* para a seguinte;

- Uma redução do esforço exigido para realizar atividades como, por exemplo, gerenciamento de configuração de *software*, garantia de qualidade e produção de documentação;

- Um aumento no controle do projeto, que é obtido por meio de um melhor planejamento, monitoração e comunicação;

- Coordenação melhorada entre os membros de uma equipe que esteja trabalhando num grande projeto de *software*.

Os ambientes integrados combinam mecanismos de integração para dados, ferramentas e interação homem/computador [PRE95]. A integração de dados pode ser conseguida por meio do intercâmbio direto de informações, das estruturas de arquivos comuns, do compartilhamento de dados através de um repositório. A integração das ferramentas pode ser projetada de forma customizada por empresas de desenvolvimento ou pode ser obtida através de um *software* que gerencie a base de dados. A integração homem/computador é conseguida por meios de padrões de interface.

4 Aspectos de Implementação de Ferramentas CASE em Java

Uma ferramenta CASE que possui como ambiente de trabalho a Internet deve suportar algumas características que este ambiente requer. Estas ferramentas devem gerenciar vários tipos de documentos gerados, atualizados, examinados e utilizados por diferentes tipos de usuários, trabalhando em diversas localidades. Além disso, deve permitir a individualização das características de desenvolvimento e visualização das informações por cada usuário. Deverá, também, permitir a troca de experiências (através de ferramentas de *chat* e *mail*) e o compartilhamento de informações (dicionário de dados) entre os usuários.

Este capítulo apresenta o estado da arte sobre ambientes de desenvolvimento de *software*, ferramentas CASE e ferramentas CASE divulgadas na WWW para a utilização na Internet. Este capítulo apresenta, também, algumas características que deverão ser implementados para que a ferramenta CASE desejada possua as características citadas acima como, por exemplo, várias formas de visualização da informação, compartilhamento de informação, armazenamento dos documentos, etc.

4.1 Estado da Arte

Nesta seção são apresentados algumas ferramentas CASE existentes no mercado, ambientes de desenvolvimento para a linguagem de programação Java e ferramentas CASE existentes no ambiente WWW.

4.1.1 Ferramentas CASE

A principal característica das ferramentas CASE para apoio ao desenvolvimento de aplicações orientadas a objetos é a utilização de editores gráficos para produção de diagramas de classe e objetos. Utilizam-se de notações como OMT [RUM91], Booch [BOO91], Coad/Yourdon [COA90], UML[FOW97] além de outras.

As ferramentas CASE orientada a objetos permitem, na maioria das vezes, a edição de um determinado modelo de classes em várias notações, dependendo da escolha do usuário. Muitas dessas ferramentas, permitem a geração do código fonte em uma determinada linguagem de programação (como por exemplo Java, C++, Smalltalk, etc.) dos elementos gráficos especificados nos diagramas. Suportam também engenharia reversa, que consiste na geração de diagramas a partir do código fonte.

Uma das vantagens da utilização dessas ferramentas é a facilidade de modificação gráfica dos modelos (inclusão, exclusão e alteração de classes, relacionamento, atributos, métodos, etc.) e a melhor visualização das características das classes [VES95]. Porém, estas ferramentas apresentam desvantagem com relação ao trabalho cooperativo das equipes de desenvolvimento, pois não dispõem de mecanismos de troca e compartilhamento de informações.

A seguir são descritas algumas ferramentas CASE orientadas a objetos que possuem as características anteriormente citadas, como Rational Rose e With Class.

➔ Rational Rose

O Rational Rose [RAT98] é uma ferramenta CASE que agrega um conjunto de editores diagramáticos que suportam a modelagem gráfica de sistemas administrativos orientados a objetos. Sua interface gráfica consiste de três elementos: *browser*, *diagram window* e *documentation*, conforme mostra a Figura 4.1.

O *browser* é uma ferramenta de navegação hierárquica que permite ao usuário visualizar nomes dos diagramas de objetos, processos e classes, bem como pacotes lógicos, classes, módulos e pacotes de componentes associados ao modelo [RAT98].

O *diagram window* permite ao usuário criar e modificar os diagramas da aplicação. Para construir uma aplicação através da modelagem gráfica, o usuário deve definir as classes, métodos, atributos e seus relacionamentos (herança, associação, agregação, etc.).

Documentation mostra os comentários agregados nas classes, métodos, atributos, relacionamentos, diagramas. Estes comentários são agregados na criação destes componentes, ou acrescentados utilizando atalhos para esta função.

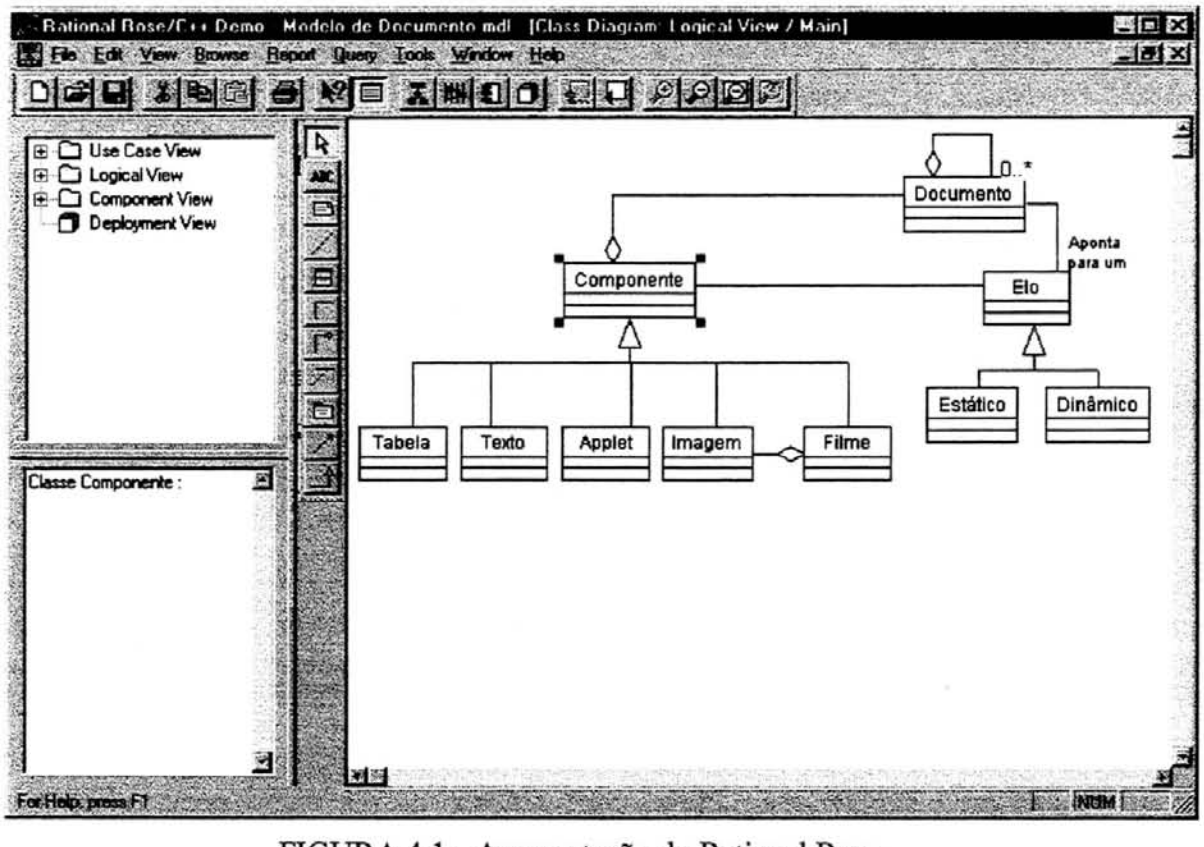

FIGURA 4.1 - Apresentação do Rational Rose

Outras características desta ferramenta são :

- Engenharia reversa
- Suporte a várias notações (UML, OMT, Booch)

→ With Class

With Class [WIT97] é uma ferramenta CASE OO, utilizada para assistir o desenvolvedor ou programador de sistemas principalmente na modelagem gráfica do sistema utilizando metodologias OO. Sua interface consiste de quatro elementos básicos (Figura 4.2) : d*iagram window, browser, class view toolbar* e *components toolbar.*

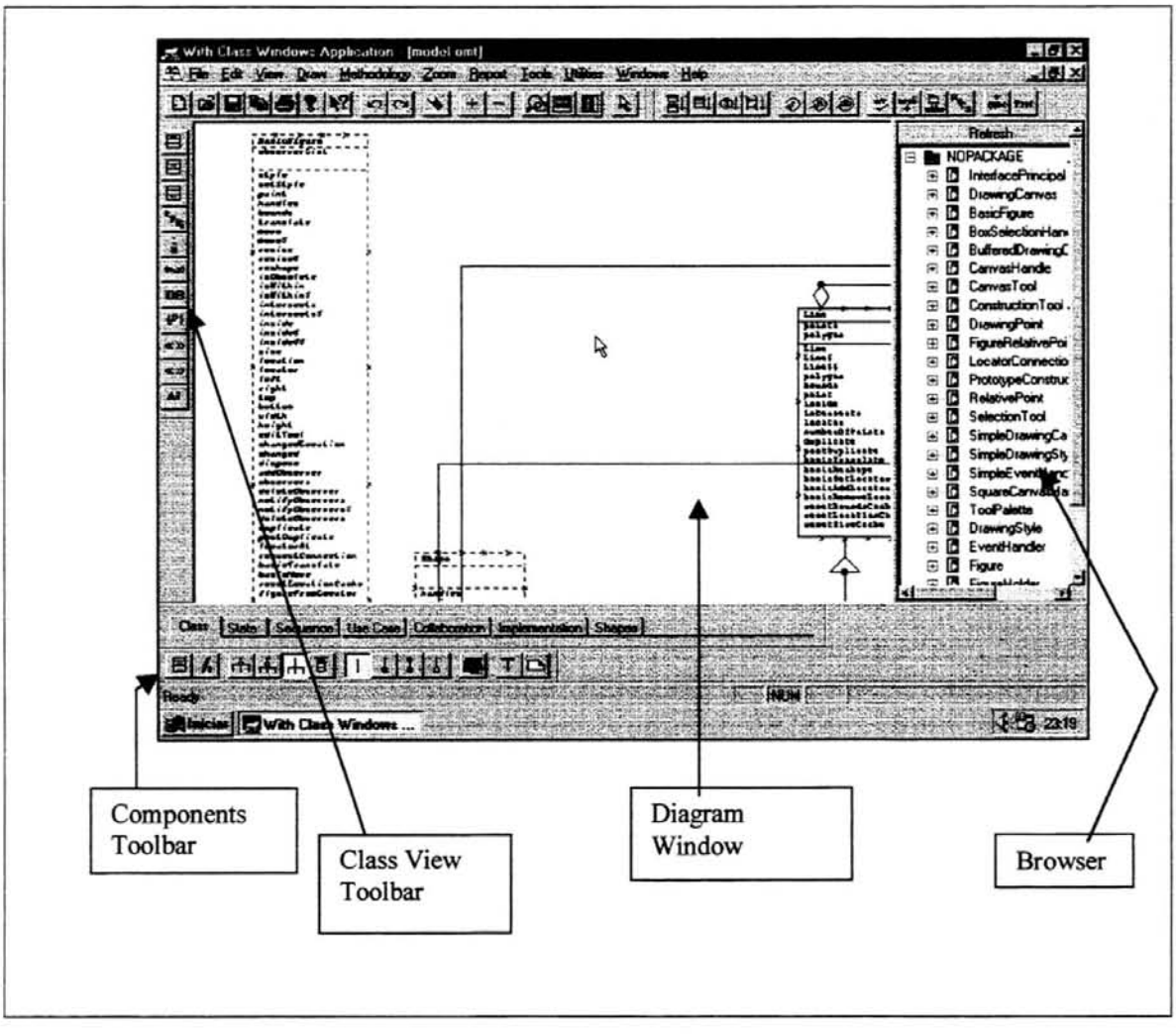

FIGURA 4.2 - Componentes da ferramenta CASE *With Class*

- *Diagram Window :* área onde o usuário cria e manipula seus diagramas;
- *Browser :* permite a navegação das classes definidas, bem como a visualização dos métodos e atributos destas classes;

- *Components Toolbar :* Barra de ferramentas que indica os diagramas e seus respectivos componentes. São eles: diagrama de classes, diagrama de estados, diagrama de colaboração, diagrama de cenários, etc.

- *Class View Toolbar :* apresenta as ferramentas de visualização das classes do diagrama como, por exemplo, visualizar somente a classe sem os atributos e métodos.

Esta ferramenta suporta as seguintes metodologias : OMT, UML, Coad-Yourdon, Booch e Shlaer-Mellor. Suporta também engenharia reversa para as seguintes linguagens de programação : Java, C++, Delphi.

4.1.2 Ambientes de Desenvolvimento

Os ambientes de desenvolvimento de *software* auxiliam os programadores na elaboração de sistemas, pela integração das ferramentas utilizadas e pelo gerenciamento das várias tarefas de administração associadas à programação [WIN93]. Eles simplificam o processo de desenvolvimento de *software* fornecendo convenções uniformes para a codificação, edição e testes dos programas.

Ambientes se destinam a suportar as tarefas de desenvolvimento de *software* [WAS94]. São portanto um meio, que tem como função apoiar a tarefa de desenvolvimento de sistemas. Ambientes são aplicações extremamente sofisticadas e muitas das soluções e ferramentas adotadas na construção destes se mostram úteis para uma vasta gama de aplicações.

Nesta seção serão apresentados alguns ambientes de desenvolvimento em Java, como o Java WorkShop, Symantec Café e Microsoft Visual J++.

➢Java WorkShop (JWS)

JWS criado pela Sun Microsystem, foi implementado como um misto de *browser* com ambiente de desenvolvimento [JWS97]. Desenvolvido totalmente em Java, implementa diversas funções de sua interface com código HTML. JWS foi criado para que o usuário não tenha que utilizar diretamente as ferramentas do *Java Development Kit* (JDK) que são ferramentas executáveis em linhas de comandos.

Com o JWS é possível criar aplicações e *applets* Java de uma forma fácil e rápida, utilizando as ferramentas do JDK com um rápido *click* do *mouse*. Além das ferramentas do JDK este ambiente disponibiliza outras ferramentas e facilidades inexistentes no JDK, como um editor de textos utilizado para a edição de programas Java e um gerenciador de projetos utilizado para a organização dos programas Java (Figura 4.3).

As principais ferramentas do JWS são:

- **Project Manager** : Um projeto é uma coleção de atributos que define como um programa é inserido em um ambiente de desenvolvimento. Estes atributos incluem os nomes de todos os programas Java que o projeto necessita para que seja compilado.

• **Portfolio Manager :** Portfolio é a organização de projetos em grupos lógicos. É possível, por exemplo, ter vários *applets* em um portfolio e jogos em outro. As informações no JWS são armazenadas em Projetos, que por sua vez são agrupadas em Portfolios [JWS97].

O Portfolio Manager possui um conjunto de ícones, e cada um deles representa um projeto no Portfolio. Cada tipo de projeto (como *applets*, aplicações, pacotes, imagens, etc.) terá seu próprio ícone.

• **Build Manager :** Utilizado para compilar os programas Java transformando-o em arquivos *bytecodes*. Pode-se compilar arquivos simples e/ou projetos. No caso de projetos será efetuada a compilação de todos os programas que fazem parte do mesmo.

Esta ferramenta imprime mensagens em uma janela de trabalho mantendo o usuário informado do processo de compilação. Caso ocorra um erro de sintaxe, a mensagem correspondente será apresentada e, selecionando o erro será apresentado o programa fonte na respectiva linha.

Outra ferramenta do JWS é o **Project Tester** utilizado para testar os programas e projetos. Possui ainda um editor de código integrado a um *browser* que permite apresentar a hierarquia de classes com toda a documentação dos métodos associados. É dotado também de um *debugger* que permite acompanhar a execução do código gerado.

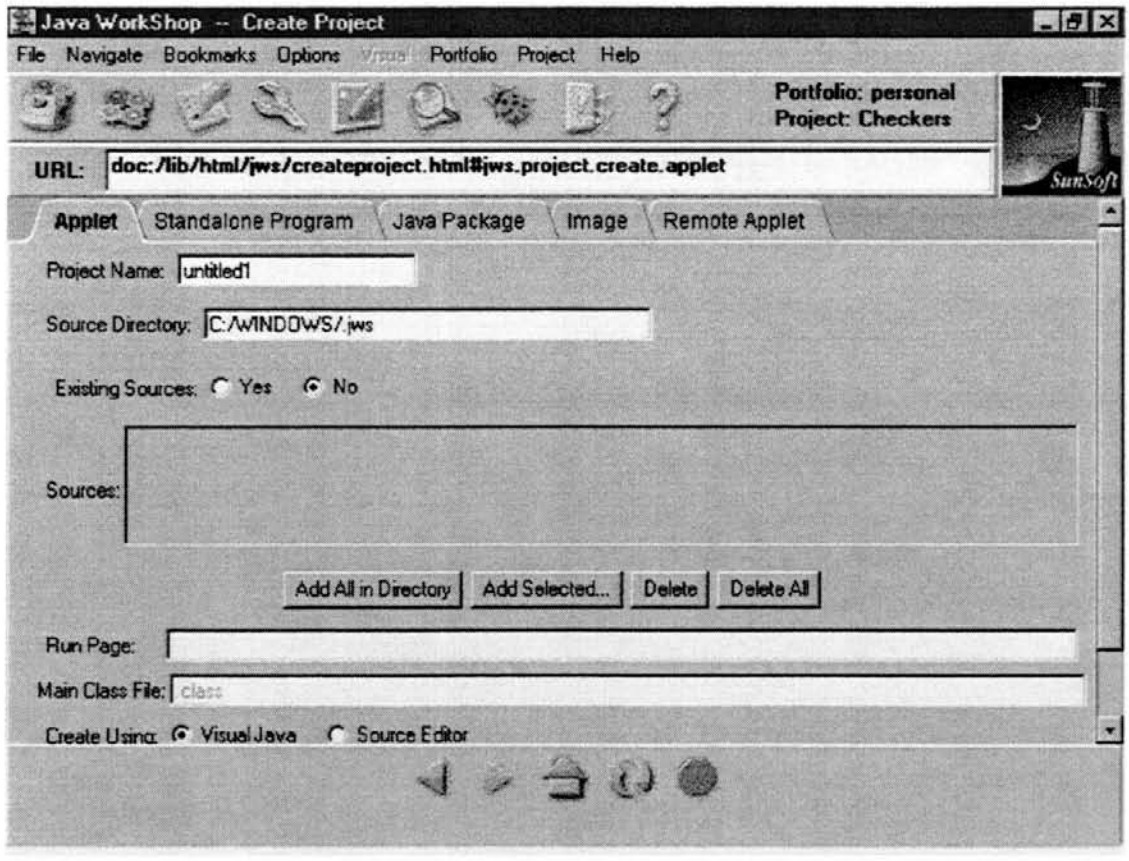

FIGURA 4.3 - Tela do Java WorkShop

➢Symantec Café

O Symantec Café, lançado pela Symantec, é um ambiente integrado dotado de recursos para acelerar a produtividade no desenvolvimento de aplicações Java [MAR96]. Ele utiliza um compilador proprietário para a geração dos *bytecodes* além de um compilador *Just-in-Time* para Java que acelera a execução de *applets*, transformando os *bytecode* em código de máquina. Todo o ambiente e o compilador foram escritos na linguagem C.

O Symantec Café (Figura 4.4) é composto dos módulos Studio que permite a criação visual de aplicações, Project Express, que automatiza a criação de projetos e AppExpress que permite gerar aplicações rapidamente. É dotado também de um editor de classes que permite observar a hierarquia e o relacionamento entre as classes.

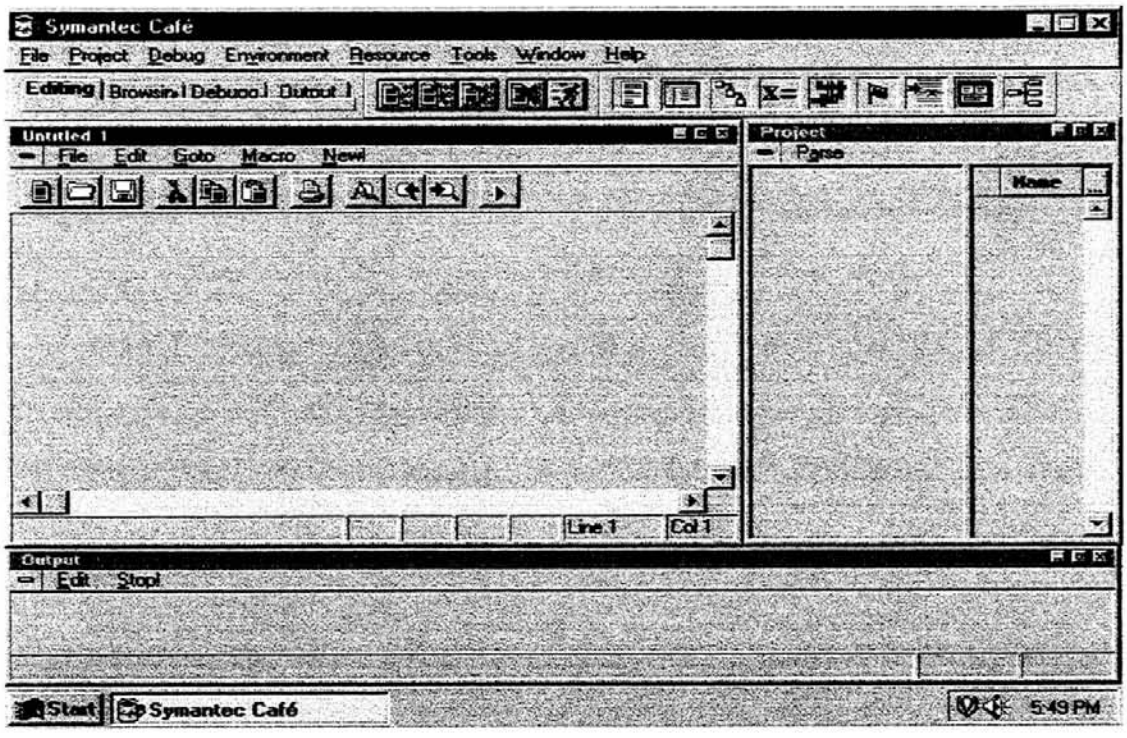

FIGURA 4.4 - VISÃO GERAL DO AMBIENTE SYMANTEC CAFÉ

Para uma melhor visão deste ambiente é apresentada a Tabela 4.1 com as principais ferramentas e suas características.

TABELA 4.1 - Principais componentes do ambiente Symantec Café

Componente	**Descrição**
Toolbar	Controle principal do ambiente. Disponibiliza acesso para todos os outros componentes.

AppExpress	Auxilia na criação rápida de projetos e programas Java.
Café Studio	Utilizado para a criação visual de interfaces gráficas.
Project Manager	Organiza graficamente e gerencia os programas Java associados a um determinado projeto.
Editor de Código	Editor de texto utilizado para a edição de programas Java.
Editor de Classes	Organiza graficamente classes e seus respectivos códigos.
Editor Hierárquico	Mostra e edita as relações entre as classes em forma de árvore.
Compilador	Compilador otimizado para programas Java
Debugger	Permite a visualização da execução passo-a-passo de um programa Java.
Compilador Just-In-Time	Compilador que converte os *bytecodes* gerados na compilação em código de máquina.

➢**Microsoft Visual J++**

Criado pela Microsoft o Visual J++ é, segundo [MIC97], um completo ambiente integrado de desenvolvimento de *software* Java. Além das vantagens dos ambientes de desenvolvimento, que com um simples click do mouse possibilita o usuário editar, compilar, executar e realizar uma verificação no código-fonte, este ambiente disponibiliza uma ferramenta para o desenvolvimento visual das aplicações e um acelerador de compilação Java.

J++ utiliza o padrão *Microsoft* de interface, também utilizado em outras linguagens como o Visual C++ e Visual Basic, conforme pode ser visualizado na Figura 4.5 Por isso, este ambiente tem seu aprendizado facilitado, o que pode ser feito rapidamente, principalmente se o usuário já teve algum contato estas outras linguagens de programação.

As suas principais ferramentas são:

• **Editor de Código :** Como em qualquer outro ambiente de desenvolvimento, o Visual J++ disponibiliza um editor de código fonte.

• **Editor Gráfico :** Disponibilizado para facilitar a criação de imagens tipo *bitmap* utilizados em ícones, cursores ou para qualquer outra porção de um programa Java. Este editor suporta os seguintes formatos de imagens : *Bitmap* (BMP), *Graphics Interface Format* (GIF) e *Joint Photographic Experts Graphic* (JPEG).

• **Ferramenta para Interface Gráfica :** O Visual J++ possui esta ferramenta para auxiliar na criação automática de interfaces gráficas, utilizadas como *layouts* em programas Java.

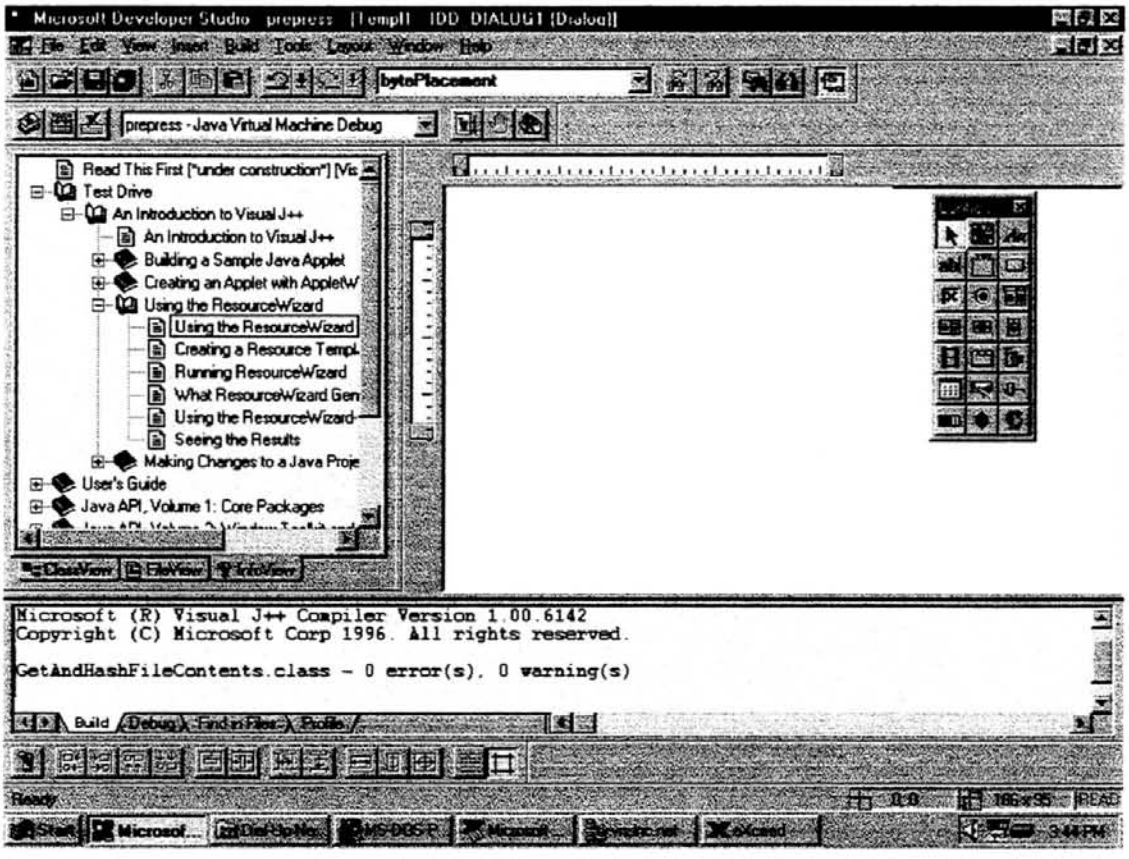

FIGURA 4.5 - VISÃO GERAL DO VISUAL J++

O Microsoft Visual J++ também utiliza o conceito de projetos para a organização dos programas Java. Isto permite agrupar programas que estão envolvidos na solução de um mesmo problema.

4.1.3 Ferramentas CASE na WWW

→ER Model Database Design

Esta ferramenta é um projeto da Universidade da Georgia, EUA. Ela foi idealizada para o projeto gráfico de bases de dados [ZHE97]. Implementada na linguagem Java possui as seguintes características [ZHE97]:

- Modelagem ER para bases de dados;
- Converte a visualização gráfica da base de dados em um esquema SQL (arquivo .sql)
- Provê ferramentas para projetar, criar e manter bases de dados.

A Figura 4.6 apresenta esta ferramenta e seus principais módulos.

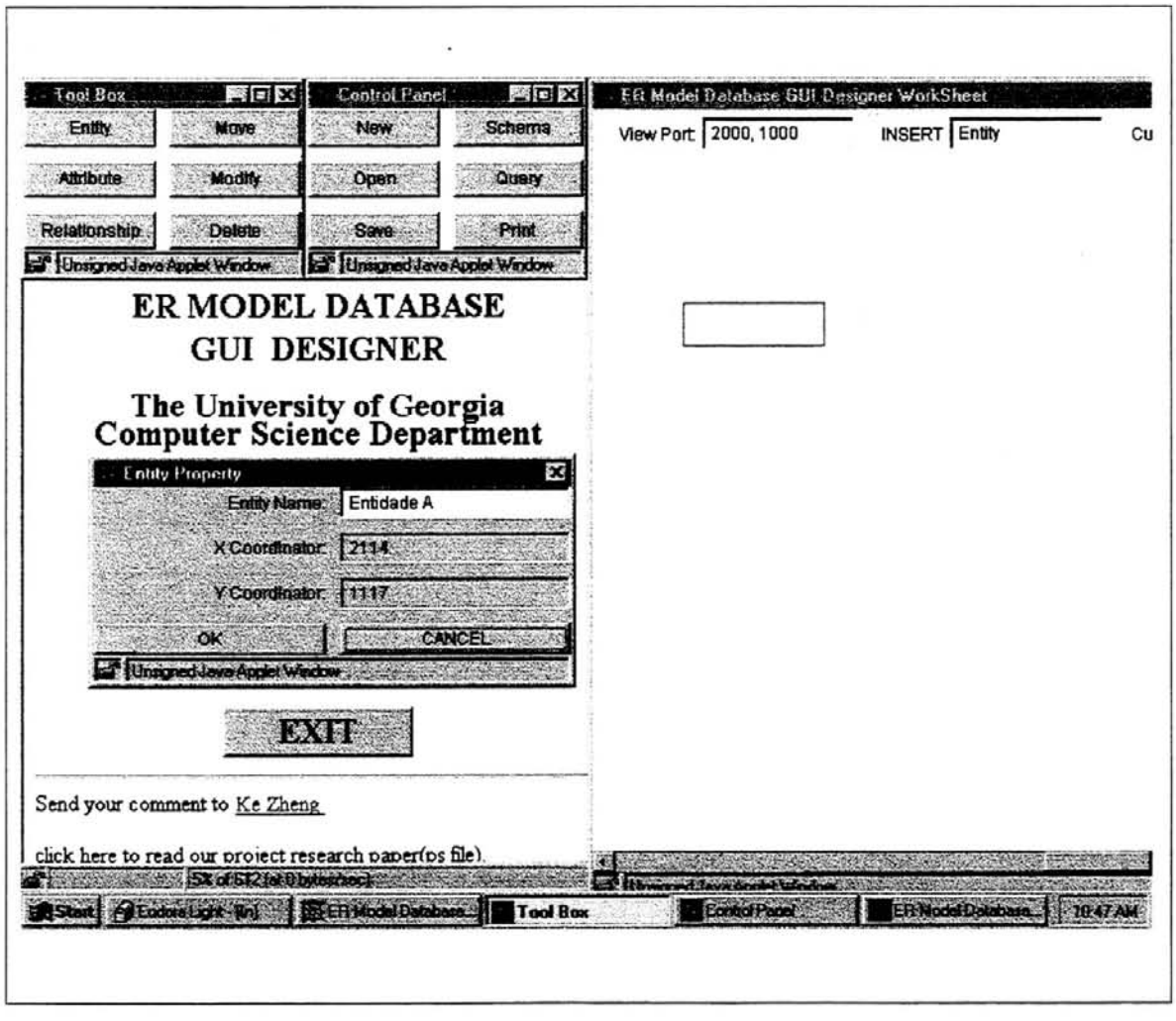

FIGURA 4.6 - ER GUI Database Design - Universidade da Georgia

➔ Editor OMT

Este editor visa a modelagem de sistemas utilizando a notação orientada a objetos OMT (Figura 4.7) [BWY97]. Implementado na linguagem Java oferece, somente, alguns recursos gráficos para a modelagem. Sendo uma ferramenta um pouco rústica, ela não permite ao usuário as seguintes tarefas :

- Armazenamento da modelagem;

- Consistência com uma base de dados;

Apesar da utilização de uma interface gráfica, esta ferramenta apresenta restrições com relação a sua utilização para modelagem de sistemas. Não existe recursos de compartilhamento de informações.

Segundo [BWY97], esta ferramenta foi implementada para um simples exercício de programação visual utilizando a linguagem Java.

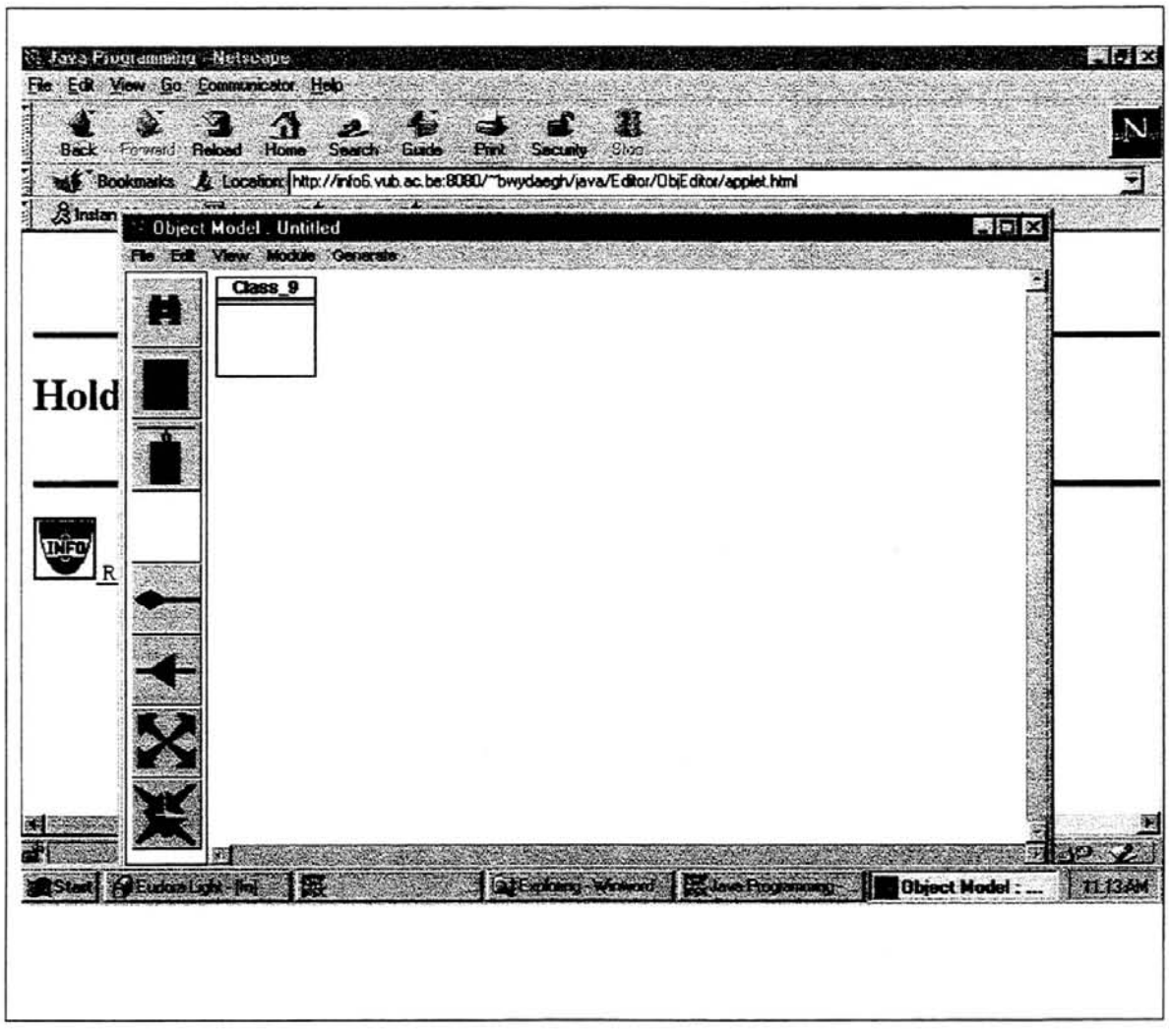

FIGURA 4.7 - Editor OMT - Universidade da Bélgica

4.2 Visualização

4.2.1 Arquitetura Model/View/Controller

A arquitetura Model/View/Controller (MVC) foi introduzida como parte da versão do SmallTalk-80 da linguagem de programação Smalltalk [PAR94]. MVC foi projetada para reduzir o esforço de programação necessário na implementação de sistemas que apresentam vários dados de forma sincronizada. Sua característica principal é que o modelo, as visões e os controladores são tratados como entidades separadas; isto faz com que qualquer alteração realizada no modelo seja refletida automaticamente em cada uma das visões.

O objetivo principal da arquitetura MVC é a separação da interação homem-máquina do domínio do problema, e do modo como isto é gerenciado [BRA95a][BRA95b]. Esta separação (dados e aplicação) gera maior flexibilidade e grande possibilidade de reuso.

Na Figura 4.8 é apresentada a arquitetura MVC em sua forma mais geral. Há um modelo, múltiplos controladores que manipulam este modelo e várias visões dos dados do modelo que se alteram quando é alterado o estado do modelo. Pode-se verificar nesta arquitetura que uma instância de uma visão tem associada uma instância do seu modelo e uma instância de um controlador. O modelo pode ser qualquer instância de qualquer classe que implemente um protocolo qualquer, mas as instâncias do controlador e da visão são geralmente projetadas em conjunto para implementar as manipulações adequadas aos conceitos do modelo que a visão representa.

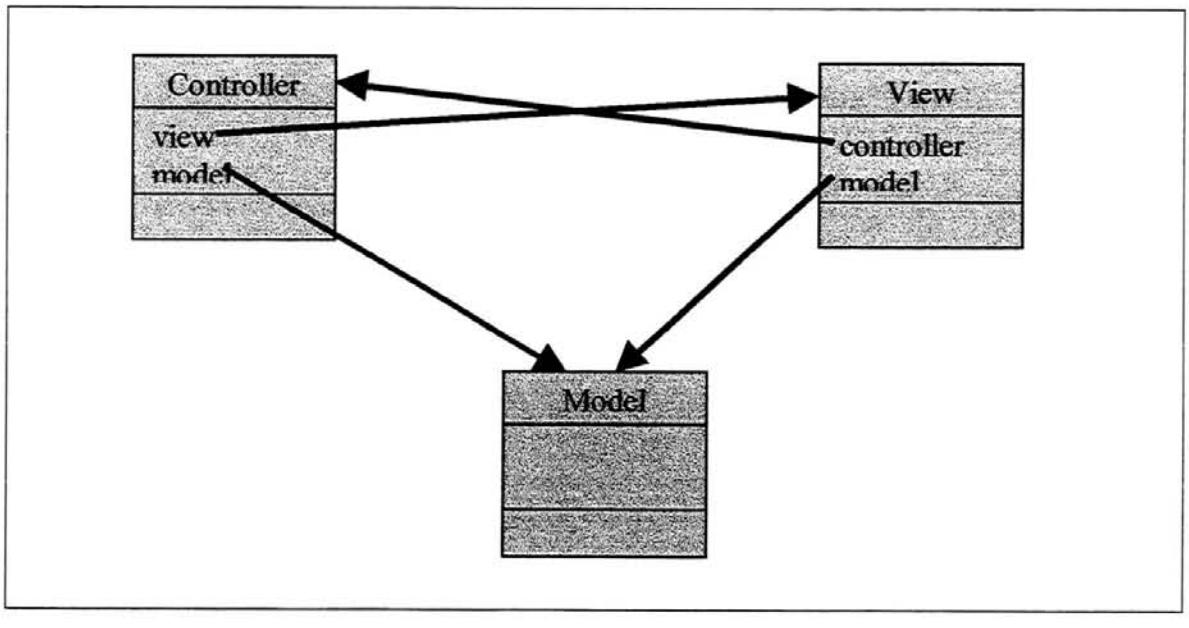

FIGURA 4.8 – Arquitetura Model/View/Controller

Esta arquitetura possibilita a independência dos modelos das suas visões, de maneira que um modelo pode ter várias visões associadas as quais desconhece. As visões recebem os avisos de mudança de seu modelo através de um mecanismo indireto de anúncio de mudanças. Com este mecanismo uma visão se alista como dependente do seu modelo e quando se produz alguma alteração de estado do modelo é gerada uma

mensagem anunciando a mudança. Todas as visões cadastradas como dependentes do modelo recebem a notificação e atualizam seus estados.

Esta arquitetura apresenta as seguintes vantagens :

➔ é definida, claramente, a separação entre componentes de um programa. O problema em cada domínio pode ser resolvido independentemente;

➔ a ligação entre o modelo e a visão é dinâmica, isto é, ocorre em tempo de execução.

A incorporação do MVC em um projeto acarreta que componentes do sistemas são projetados separadamente e são unidos em tempo de execução[BAR95][PAR94]. Se, posteriormente, observa-se que um dos componentes está funcionando de forma errônea, pode-se reestruturá-la sem afetar os outros componentes.

As desvantagens da utilização desta arquitetura são :

➔ requer maior esforço no projeto e desenvolvimento, devido a sua complexidade;

➔ possível ocorrência de *overhead* de métodos chamados entre objetos *Model* e *Views*.

Pode-se definir as partes do MVC como :

Model : é o objeto que representa os dados da aplicação [BRA95a]. Gerencia os dados e conduz todas as transformações destes dados. O modelo não possui o conhecimento especificado do controlador ou das visões, mas possui referências internas para ambos (*controller* e *view*). Preferivelmente, o sistema obtém a responsabilidade de manter a comunicação entre o modelo e as visões, e notificar as visões quando ocorrer mudanças no modelo.

View : é o objeto que gerencia a visualização das informações do modelo [BRA95a]. Produz a representação visual dos objetos do modelo e mostra os dados para o usuário. Interage com o modelo através de uma referência com o modelo.

Controller : é o objeto que provê os meios para a interação do usuário com a representação dos dados do modelo [BRA95a]. Interage com o modelo através de uma referência interna.

A Figura 4.9 apresenta o relacionamento entre as classes da arquitetura MVC.

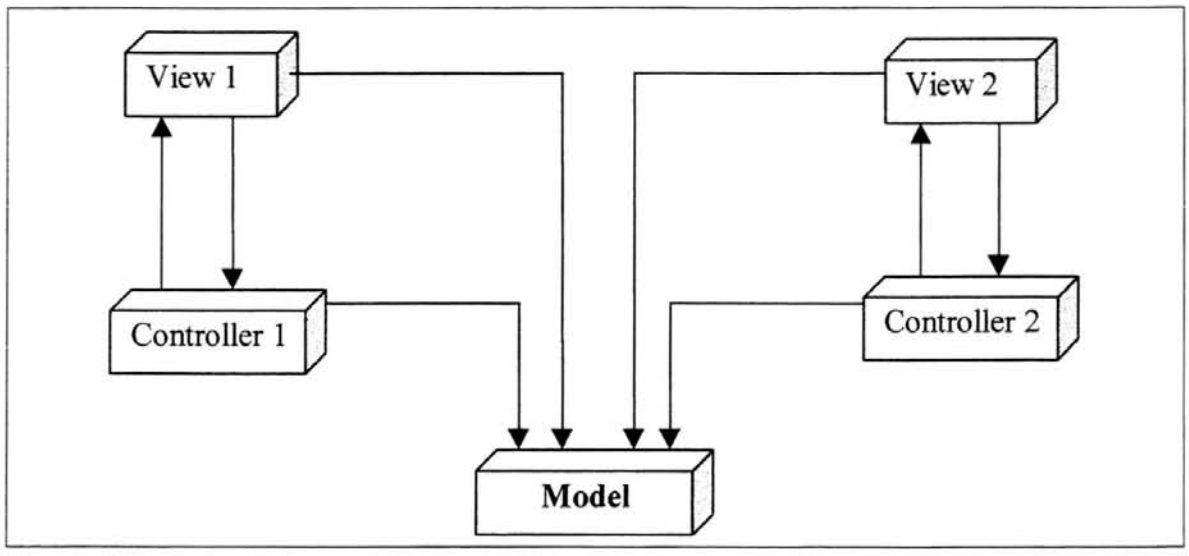

FIGURA 4.9 – Relacionamento entre as Classes da Arquitetura MVC

4.2.2 MVC em Java

A linguagem de programação Java proporciona suporte a arquitetura MVC através das seguintes classes :

Observer : qualquer objeto que quer ser notificado quando o estado de um outro objeto é alterado;

Observable: qualquer objeto cujo estado pode ser de interesse e sobre o qual outro objeto tem demonstrado esse interesse.

Estas duas classes podem ser utilizadas para muitas outras coisas do que somente a implementação da arquitetura MVC. Elas são convenientes para qualquer sistema onde objetos precisam ser notificados das alterações que ocorrem em outros objetos [SUN96a].

Tipicamente, o modelo (*model*) é uma subclasse de **Observable** e as visões (*view*) são implementações de **Observer**, como ilustra a Figura 4.10. Estas duas classes manejam adequadamente a função de notificação das alterações que a arquitetura MVC necessita. Proporcionam o mecanismo pela qual as visões e os controladores possam ser automaticamente notificados das alterações do modelo. Referências ao objeto do modelo, tanto nos controladores como nas visões, permitem a estes o acesso aos dados do modelo [SUN96a].

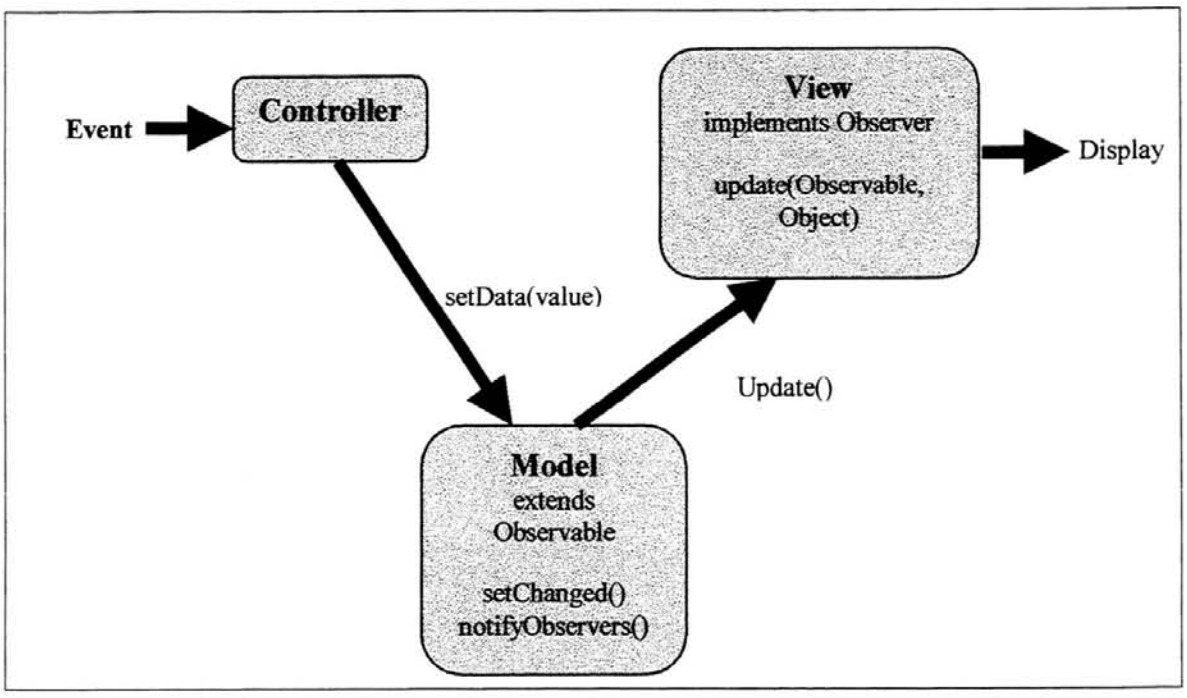

FIGURA 4.10 - MVC em Java

Os principais métodos das classes *Observer* e *Observable* são apresentados na Tabela 4.2.

TABELA 4.2 - Principais Métodos das Classes *Observer* e *Observable*

Observer	
Public void update (Observable obs, Object obj)	Chamado Quando ocorreu uma alteração no estado do modelo (objeto Observable);
Observable	
Public void addObserver(Observer obs)	Adiciona um observador a lista interna de observadores;
Public void deleteObserver(Observer obs)	Remove um observador da lista interna de observadores;
Public void deleteObservers()	Remove todos os observadores da lista interna de observadores;
Public int countObserver()	Retorna o número de observadores que estão na lista interna de observadores;
Protected void setChanged()	Ajusta o *flag* interno que indica a ocorrência de alteração no modelo;

Protected void clearChanged()	Limpa o *flag* que indica a ocorrência de alterações no modelo;
Protected boolean hasChanged()	Retorna um valor lógico se o modelo alterou seu estado;
Public void notifyObservers() *Public void notifyObservers(Object obj)*	Verifica se o *flag* interno está indicando alteração no modelo e notifica todos os observadores;

Para a criação do modelo é necessário a criação de uma subclasse da classe Observable. Como esta classe implementa todos os métodos necessários para proporcionar o funcionamento do tipo Observador/Observado, a classe derivada somente necessita proporcionar alguns mecanismos para ajustar seu funcionamento particular e proporcionar acesso ao estado interno do objeto *Observable*.

As visões são obtidas através da implementação da interface Observer. Esta nova classe observa as modificações de outro objeto. Nesta classe deve ser implementado um método chamado *update*, este será invocado quando o modelo alterar seu estado através do método *notifyObservers*.

Os controladores são os tratadores de evento da linguagem Java. Na versão 1.02 e anteriores é utilizado o modelo de herança (*Inheritance Model*)[BAL97], enquanto que nas versões 1.1.1 e posteriores é utilizado o modelo de delegação(*Delegation Model*)[BAL97][DED97][SUM97][SUN96].

➔Modelo de Herança (*Inheritance Model*)

Este modelo processa os eventos gerados pelo usuário e é baseado na herança. Basicamente recebe e processa os eventos ocorridos em uma *Graphical User Interface* (GUI), sendo que as instâncias dos componentes gráficos devem rescrever os métodos **action()** ou **handleEvent()** para os eventos que deverão ser tratados.

Neste modelo o evento é propagado seqüencialmente na hierarquia da GUI até ser processado ou ter chegado à raiz da hierarquia. O resultado deste modelo é que os programas tem essencialmente duas escolhas para estruturarem seus *event-handling* :

- Cada componente pode ser subclassificado para especificar seus *event-handling*;
- Todos os eventos de uma hierarquia (ou subconjunto) podem ser manipulados por um *container* particular. O resultado disto é que o *container* deve rescrever os métodos **action()** ou **handleEvent()**, devendo conter os procedimentos para processar os eventos.

No modelo de herança os eventos não são filtrados [BAL97]. Eles são enviados para todos os respectivos componentes sem a preocupação de serem ou não utilizados. Isto pode acarretar em um problema geral de performance, particularmente com eventos *high-frequency* como, por exemplo, o evento *MouseMove*.

➔Modelo de Delegação (*Delegation Model*)

Delegação é a habilidade de uma classe delegar parte de sua interface para uma instância específica que implementa esta interface [SUM97][DED97] Suas principais características, segundo [SUM97] são as seguintes :

- Simples e de fácil aprendizado;
- Apresenta uma separação clara entre a aplicação e a GUI;
- Facilita a criação de um código de manipulação de eventos robusto;
- Flexibilidade para vários modelos de aplicação para fluxo e propagação de eventos;
- Para a criação de ferramentas visuais, ativa em *run-time* a descoberta dos eventos que um componente gera, bem como os eventos que ele observa.

Este modelo possui tipos de eventos que são encapsulados em uma hierarquia de classes iniciada por *java.util.EventObject*. Um evento é propagado desde o objeto *Source* até o objeto *Listener* .

Um *Listener* é um objeto que implementa uma interface *EventListener*, estendida a partir da classe genérica *java.util.EventListener*. Uma interface *EventListener* define um ou mais métodos, os quais são invocados pelo *Event Source* responsável por cada tipo de evento manipulado pela interface.

Um *Event Source* é um objeto que origina ou dispara eventos [SUM97]. Este objeto possui a capacidade de detectar eventos (como um botão (Button)) e notificar os objetos *Listener* da ocorrência do evento. Em um programa, o *Event Source* é tipicamente um componente GUI e o *Listener* é normalmente um objeto *Adapter* que implementa o *Listener* apropriado (ou conjunto deles) para uma aplicação para o controle do fluxo e manipulação dos eventos. O objeto *Listener* pode, também, ser outro componente AWT que implementa uma ou mais interfaces *Listener* para a proposta de objetos *hooking* da GUI.

Um objeto *Listener* pode ser registrado em um objeto *Source*. Esta é a forma que o objeto *Listener* requisita a notificação da ocorrência de todos os eventos da classe especificada para qual o objeto *Listener* foi projetado para responder aos eventos de uma classe especificada pela implementação de uma interface *Listener*.

Por exemplo, em um simples programa que permite dois diferentes objetos *Listener* serem instanciadas a partir de duas diferentes classes *Listener*. Cada um destes objetos é registrado para receber todos os eventos envolvidos na manipulação de um objeto *Frame* (abrir, fechar, minimizar, etc.). Os eventos envolvidos no objeto *Frame* serão enviados para os dois objetos *Listener*.

Um desses objetos *Listener* implementa a interface *WindowListener* e, consequentemente, deve implementar todos os seis métodos desta interface. O outro objeto *Listener* é uma subclasse da classe *WindowAdapter* que implementa a interface *WindowListener*.

Foram adicionados métodos nas classes dos componentes AWT como, por exemplo, a classe *Button*, para que estes suportem este novo modelo de eventos. A Tabela 4.3 apresenta os componentes AWT e os métodos adicionados.

TABELA 4.3 - Componentes AWT e Eventos Java gerados [FLA97]

Componente	Evento	Descrição
Button	ActionEvent	Click do Botão
Checkbox	ItemEvent	Selecionar um Item
CheckboxMenuItem	ItemEvent	Selecionar um Item
Choice	ItemEvent	Selecionar um Item
Component	ComponentEvent	Mover, redimensionar, mostrar e esconder um componente
	KeyEvent	Pressionar uma tecla
	MouseEvent	Pressionar um botão do mouse, o mouse entrar ou sair de um componente, mover ou arrastar um componente
Container	ContainerEvent	Componente é adicionado ou removido de um container
List	ActionEvent	Duplo click em um item
	ItemEvent	Selecionar um item
MenuItem	ActionEvent	Selecionar um item do menu
Scrollbar	AdjustmentEvent	Ajustar a barra de rolagem
TextComponent	TextEvent	Alterar o texto
TextField	ActionEvent	Terminar edição do texto
Window	WindowEvent	Abrir, fechar, minimizar, maximizar uma janela

4.2.3 *Framework* HotDraw

HotDraw é um *framework* gráfico bidimensional para editores de desenho estruturados inicialmente escrito na linguagem Smalltalk [BRA95a][BRA95b]. Pode ser utilizado por diferentes aplicações como, por exemplo, um simples programa de desenho ou um editor diagramático. O editor pode ser a aplicação completa ou pode ser uma pequena parte de um grande sistema.

A classe mais importante no *HotDraw* é a classe *Figure* [JOH91]. Esta classe implementa mecanismos que, quando instanciados, permitem criar elementos gráficos como, por exemplo, linhas, elipses, textos entre outros. Uma aplicação do *hotdraw* utiliza esta classe para criar seus elementos gráficos. As figuras são responsáveis pela notificação aos seus dependentes de sua alteração. Outras importantes classes do *hotdraw* são *Drawing, Tool, Handle e DrawingEditor.*

Um editor escrito utilizando o *hotdraw* possui um conjunto de ferramentas (*tools*) que são utilizadas para manipular os desenhos (elementos gráficos). Tem-se ferramentas para criar novas figuras e ferramentas para manipular figuras já existentes.

4.2.4 *Framework HotDraw* para Java

O *framework HotDraw* para a linguagem Java foi desenvolvido pela empresa *Knowledge Systems Corporation* e disponibilizado na Internet para ser utilizado por estudantes, pesquisadores e desenvolvedores de sistemas [KNO97]. Este *framework* apresenta a mesma concepção do *framework* original (versão Smalltalk).

Com este *framework* é possível implementar em Java editores e/ou manipuladores de figuras de forma fácil e rápida, instanciando e subclassificando suas principais classes como, por exemplo, classe *Figure*.

O problema deste *framework* é basicamente a documentação apresentada. Apesar de ser baseada na versão para a linguagem Smalltalk, este *framework* apresenta novas classes que, de certa forma, acrescentam flexibilidade e adaptação à nova linguagem (Java), principalmente por esta apresentar características como independência de plataforma e segurança. Porém com a inclusão destas classes, a documentação genérica do *HotDraw* não é suficiente, e a solução para isto é verificar a documentação apresentada em cada arquivo de classe, resultando perda de tempo no desenvolvimento, fugindo assim do propósito da utilização do *framework*.

4.3 Groupware em Java

Groupware é um conjunto de ferramentas que facilitam a cooperação dentro de uma organização [HIL97]. É definido como um sistema baseado em computador que suporta grupo de pessoas engajadas numa tarefa ou objetivo comum e que provêm uma interface para um ambiente compartilhado [REI95], como foi visto no capítulo 2.

Para a implementação de uma ferramenta CASE na linguagem de programação Java, observando os conceitos analisados no capítulo anterior, tem-se a necessidade de implementação de um sistema de *groupware*, em Java, para suprir as necessidades de colaboração, cooperação e troca de mensagens entre os usuários da ferramenta CASE.

Uma ferramenta CASE deve suportar o trabalho em equipe durante o desenvolvimento de *software* [SER93]. Seu principal objetivo deve ser o suporte aos aspectos de cooperação na modelagem do *software*, focando-se nas atividades de diagramação. Este suporte envolve alguns aspectos que devem ser considerados, tais como a notação de modelagem adotada, controle de concorrência, comunicação e percepção entre os membros da equipe.

A ferramenta CASE deve suportar vários tipos de notações de modelagens para os diferentes tipos de usuários, pois cada usuário envolvido no desenvolvimento possui um diferente grau de conhecimento e experiência com relação às técnicas de modelagem. Isso faz com que cada usuário tenha uma escolha sobre a técnica de

modelagem mais adequada ao seu conhecimento, podendo ele fazer sua escolha a notação desejada.

No ambiente WWW, como em qualquer outro ambiente de rede, deve-se ter um controle de concorrência com as informações armazenadas na base de dados para que não ocorram problemas de integridade dos dados e redundância de informações [BER87]. Isto pode ser conseguido através de concessões de atualização da base de dados, onde somente um usuário pode atualizar um conjunto de informações que a ele foram delegados. A discussão sobre este problema é visto mais profundamente no capítulo 5.

Além da comunicação indireta proporcionada pelo armazenamento dos documentos, estas ferramentas CASE devem proporcionar comunicação direta entre os usuários que estão desenvolvendo uma determinada tarefa. Esta comunicação pode ser feita através de mecanismos conhecidos como, por exemplo, *eletronic-mail (e-mail)* e *chat*.

A percepção deve ser outra característica forte neste tipo de ferramenta CASE. Quando ocorrer uma alteração na base de dados ou quando algum usuário se ausentar e não responder suas mensagens, os outros usuários devem ser notificados imediatamente.

Não existe uma biblioteca de classes específica para a implementação de *groupware*. Porém, esta linguagem disponibiliza recursos que, de certa forma, auxiliam no desenvolvimento de um sistema de *groupware* como, por exemplo, a biblioteca de classes que auxiliam no desenvolvimento de sistemas de comunicação na rede (*java.net*) entre outras.

4.4 Armazenamento de Dados

O armazenamento das informações que uma ferramenta CASE manipula é uma de suas das características mais importantes. Deve-se salientar que além de uma base de dados consistente deve-se ter um sistema de arquivos que possibilite aos usuários armazenar informações temporárias. Nesta seção são apresentados alternativas para estes dois tipos de armazenamento.

4.4.1 Java File System (JFS)

JFS é um simples protocolo de rede para sistema de arquivos para *applets* Java baseado em TCP/IP [CAM97]. Ele permite que *applets* possam carregar e salvar arquivos em um servidor remoto. Por questões de segurança os *applets* não podem acessar (ler e gravar) arquivos na máquina cliente onde estão sendo executados. Para que se possa trabalhar com arquivos é necessário um mecanismo que possibilite e gerencie estas tarefas. O JFS disponibiliza uma forma dos *applets* acessarem arquivos.

Basicamente o JFS é dividido em duas partes : o servidor JFS (*JFS server*) e o cliente JFS (*JFS client*), como mostra a figura 4.11. O JFS *server* é um programa Java que é executado em qualquer máquina como, por exemplo, o servidor Web. O JFS

client são *applets* Java que conectam-se com o JFS *server* através de uma conexão TCP (via porta 9888) [CAM97].

Após estabelecida a conexão e o cliente ter conseguido sua autenticação, este poderá realizar tarefas como, por exemplo :

→ Carregar, salvar e remover arquivos ;

→ Criar diretórios;

→ Conseguir informações sobre outros usuários;

→ Alterar permissões de arquivos e diretórios;

→ Enviar mensagens (e-mail), ...

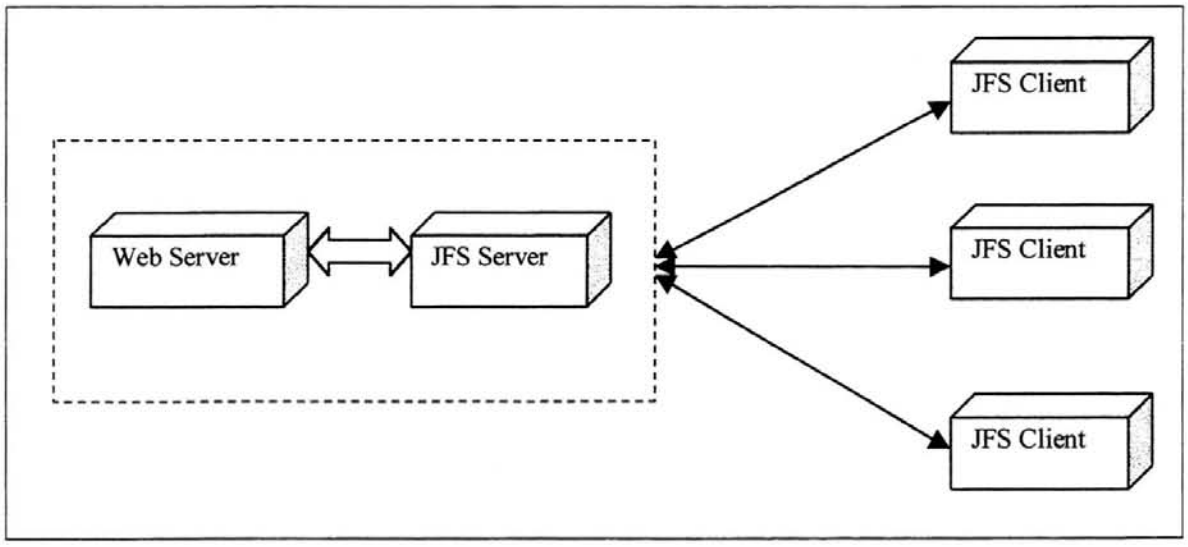

FIGURA 4.11 - Arquitetura JFS

Este sistema de arquivos é similar ao sistema de arquivos do Unix, possui diretórios, arquivos, usuários e grupos [CAM97]. Entretanto, o sistema de arquivos JFS também suporta :

Lista de Controle de Acesso : cada arquivo ou diretório possui uma lista arbitrária de usuários e grupos associados, cada qual com permissões de leitura e escrita. O acesso a estas listas não é permitido aos usuários.

Multiversões de Arquivos : Um arquivo pode ser, ou single-version (como o Unix) ou multi-version (como VMS).Todas as versões do arquivo podem ser copiadas, renomeadas, deletadas juntas [CAM97].

Tipos MIME : Cada arquivo possui um tipo MIME para identificar o seu contexto como, por exemplo, "text/plain" [CAM97]. Isto pode ser utilizado para executar um programa *handler* para visualizar este arquivo.

Os direitos de acesso, detalhes das versões e tipos MIME dos arquivos em cada diretório JFS são armazenados em um arquivo chamado **.jfs**, localizado em cada diretório do sistema local.

O JFS *server* utiliza um diretório no sistema de arquivo local, na máquina onde está sendo executado, como diretório *root*. Propriedades e direitos de acesso para os arquivos deste "sistema de arquivos virtual" são armazenados em um arquivo chamado **.jfs** em cada diretório [CAM97]. O diretório **root** contém os arquivos básicos necessários para o sistema JFS como ,por exemplo, "/etc/users" e "/dev".

Para executar o servidor deve-se digitar o seguinte comando :

"*C/>* Java JFSserver ./root"

Uma vez o cliente estando conectado ao servidor, toda a comunicação entre JFS *client* e JFS *server* é estabelecida na forma de *requests* e *reply* [CAM97]. Cada *request* gerado a partir de um cliente forçará o servidor a retornar através de um *reply*. Por exemplo, para carregar um arquivo a partir do JFS o cliente enviará um *request* (Get). Em resposta, o servidor enviará, ou *Fail* ou *Data reply*, dependendo do sucesso ou falha da requisição.

Como o sistema de arquivos do Unix, o JFS suporta arquivo especiais chamados **devices**. Arquivos device são usualmente encontrados no diretório **/dev/** , e possuem um tipo MIME "jfs/special" [CAM97]. Quando um JFS *client* requisita a leitura ou escrita de um arquivo *device*, o JFS *server* carrega por nome o *device driver class* (subclasse de *DeviceDriver*) e invoca os métodos **read** ou **write**.

Devices provêm um forma para o clientes realizarem tarefas como : imprimir ou enviar mensagens (e-mail). Os seguintes *devices* fazem parte do JFS :

dev/Printer : Dados escritos neste *device* são impressos no servidor de impressão. Este servidor de impressão deve ser definido em **/etc/printers**.

/dev/Web : Quando a requisição *Get* é enviada para este *device*, a URL é utilizada para buscar as informações para o cliente.

/dev/TCP : Quando este *device* recebe a requisição *Get* do cliente, esta conecta-se à máquina e porta fixada na requisição, e através desta porta e device o cliente envia as informações desejadas a máquina remota.

/dev/Server : Esta *device* é uma forma para o cliente buscar e alterar as informações do servidor. Utilizado somente pelo administrador.

4.4.2 Java DataBase Connectivity (JDBC)

Muitos programadores gostariam de codificar seus sistemas independentemente do banco de dados que utilizarão. Para isto foi definido um *Application Programming Interfaces* (API) que suporta os comandos básicos de SQL. Este API chama-se Java DataBase Connectivity (JDBC).

O JDBC é um padrão Java para a conexão com Banco de Dados (BD) [PAT96][NEW96]. Ele tem a pretensão de abastecer o vazio que existe no nível de conexão, já que empresas estão desenvolvendo interfaces proprietárias.

O JDBC cria um nível de programação para comunicação com o banco de dados de uma forma uniforme similar aos conceitos do ODBC que se tornou padrão em computadores pessoais e *Local Area Network* (LAN) [SIG95][NEW96]. O padrão JDBC é baseado no X/*Open* SQL *Call Level Interface* , o mesmo que o ODBC da Microsoft se baseia.

A principal diferença de um programa usuário ODBC e um programa usuário JDBC é o fato de que o primeiro estabelece uma conexão com apenas um BD, enquanto o outro pode, em um mesmo programa, estabelecer conexão com vários BD, localizados em diferentes plataformas, integrando suas informações.

Classes para abrir transações com BD são implementadas em Java para permitir maior interação, o que não iria ocorrer se fossem implementadas como chamadas de funções escritas na linguagem C e agregadas a programas Java, utilizando o padrão ODBC [PAT96][SIG95]. Com JDBC mantém-se a segurança, a robustez e a portabilidade, que são características da linguagem de programação Java.

De qualquer modo, para promover a utilização e manter o mesmo nível de compatibilidade, o JDBC pode ser implementado utilizando o ODBC e também com outras SQL API.

São duas as principais interfaces do JDBC. A primeira é o JDBC API utilizado para escrever aplicações. O segundo é utilizado em um nível mais baixo para a conexão dos drivers dos bancos de dados.

JDBC API é expressado como uma série de interfaces abstratas Java, o que permite a uma aplicação abrir conexões para banco de dados específicos, executar comandos SQL e processar resultados (Figura 4.12).

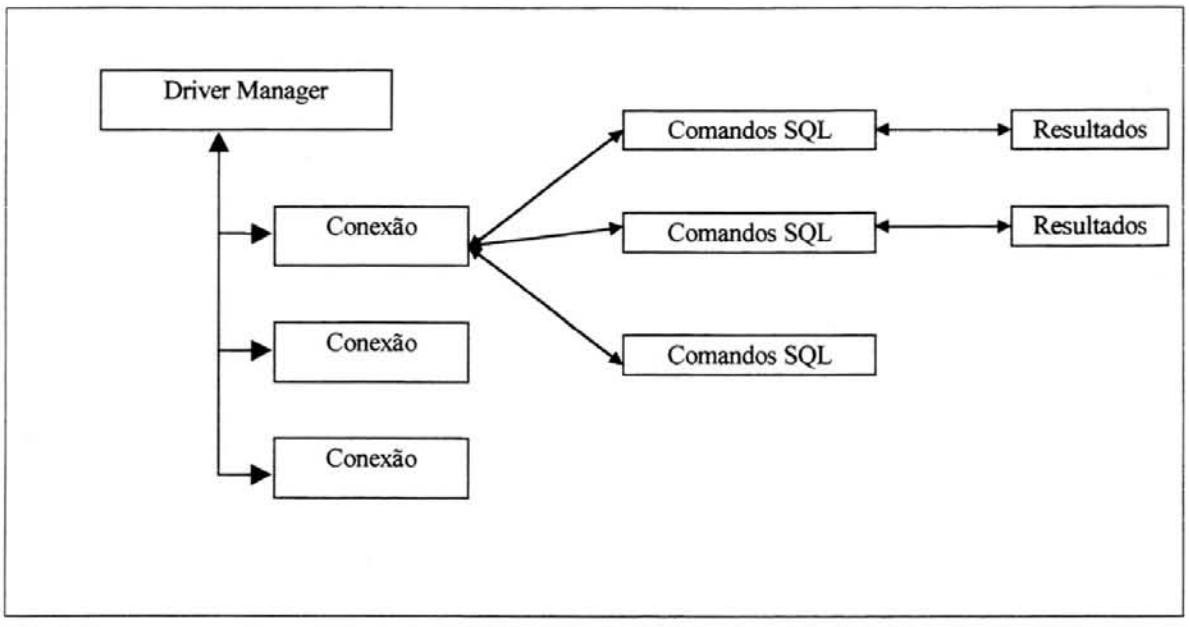

FIGURA 4.12 - Principais funções do JDBC

As interfaces mais importantes são :

• java.sql.DriverManager : Manipula a leitura dos drivers e suporta a criação de novas conexões a bancos de dados.

• java.sql.Connection : Representa as conexões para um banco de dados em particular.

• java.sql.Statement : Atua como um container para a execução de comandos SQL em uma conexão.

• java.sql.ResultSet : Controla o acesso aos resultados de um dado comando SQL.

A interface java.sql.Statement tem dois subtipos : java.sql.PreparedStatement para a execução pré-compilada de um comando SQL e java.sql.CallableStatement para execução de uma chamada a um procedimento armazenado em um banco de dados [HAM96][PAT96].

No JDBC Driver Interface a maior parte dos drivers de banco de dados simplesmente precisa definir implementações das classes abstratas definidas pelo JDBC API. Especificamente, cada driver tem que definir implementações de **java.sql.Connection, java.sql.Statement, java.sql.PreparedStatement, java.sql.CallableStatement e java.sql.ResultSet.**

Além disso, cada driver de banco de dados precisa definir uma classe que implementa a interface java.sql.Driver utilizada pela classe genérica java.sql.DriverManager quando ela precisa alocar um driver para um banco de dados específico. A Figura 4.13 mostra a divisão do JDBC.

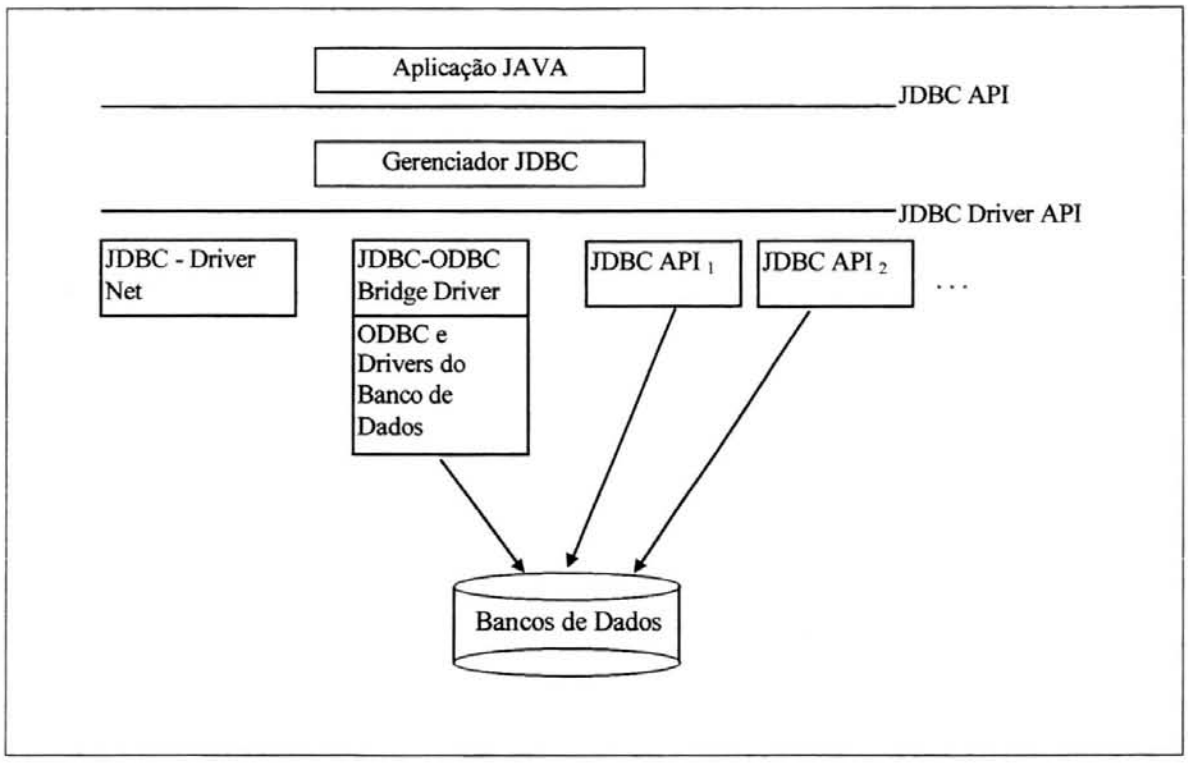

FIGURA 4.13 Níveis de funcionamento do JDBC

➔Cenários para Uso

Java pode ser utilizada em conjunto com a WWW [COR96][NEW96][SHA96a]. Programas em Java são agregados a páginas HTML como *applet*, enquanto que aplicações implementadas em Java são chamadas de *standalone*

Applets : a mais divulgada forma de uso do Java é através da implementação de *applets*, que são enviados na rede Internet juntamente com documentos HTML [NEW96]. Assim, podemos ter acesso a bancos de dados através de *applets* que utilizam JDBC como interface para o mesmo (figura 4.14).

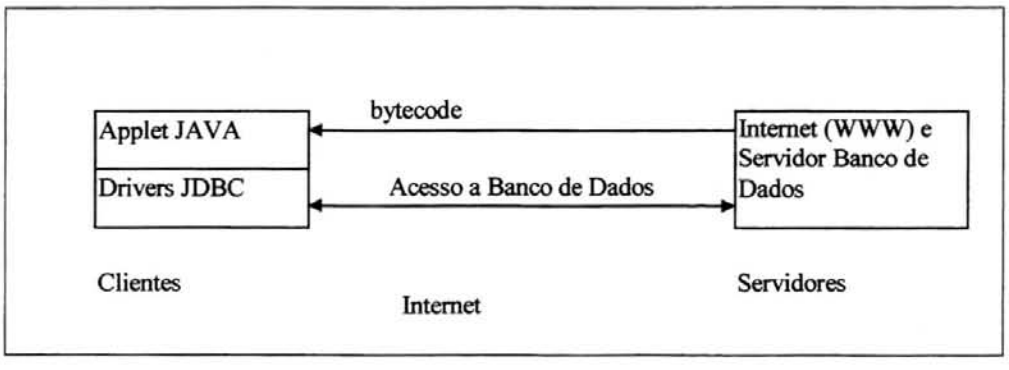

FIGURA 4.14 - Acesso a Banco de Dados utilizando applets

Por exemplo, um usuário deseja verificar, em forma de gráfico, o histórico de preços de um determinado produto. O mecanismo de visualização desta informação (histórico de preços) pode estar disponível em uma página HTML da empresa que vende o produto, o usuário então deverá acessar esta página e através de um *applet* a página acessará um banco de dados relacional obtendo o histórico dos preços e gerando o gráfico.

Estes *applets* diferem de aplicações tradicionais de banco de dados nos seguintes aspectos :

• *Applets* não confiáveis são muitas vezes restritos nas operações que podem efetuar [HAM96]. Elas devem ter permissão para a execução. Em particular, não é permitido qualquer acesso a arquivos locais e não é permitido abrir conexões em redes para máquinas arbitrárias.

• Considerações de performance de acesso para a implementação de conexão a bancos de dados difere quando o banco de dados estiver na metade do caminho ao redor do mundo [HAM96]. Por exemplo, em um programa Java que acessa informações de um banco de dados localizado no Instituto de Informática em Porto Alegre, quando acessado da mesma localidade recuperará as informações de uma forma mais rápida, do que quando acessado de um local extremamente distante.

Aplicações: Java pode também ser utilizada para a construção de aplicações convencionais (Figura 4.15) [NEW96]. Neste cenário, o código Java é confiável e é permitida a leitura e gravação de arquivos, abertura de conexões com outras máquinas, etc.

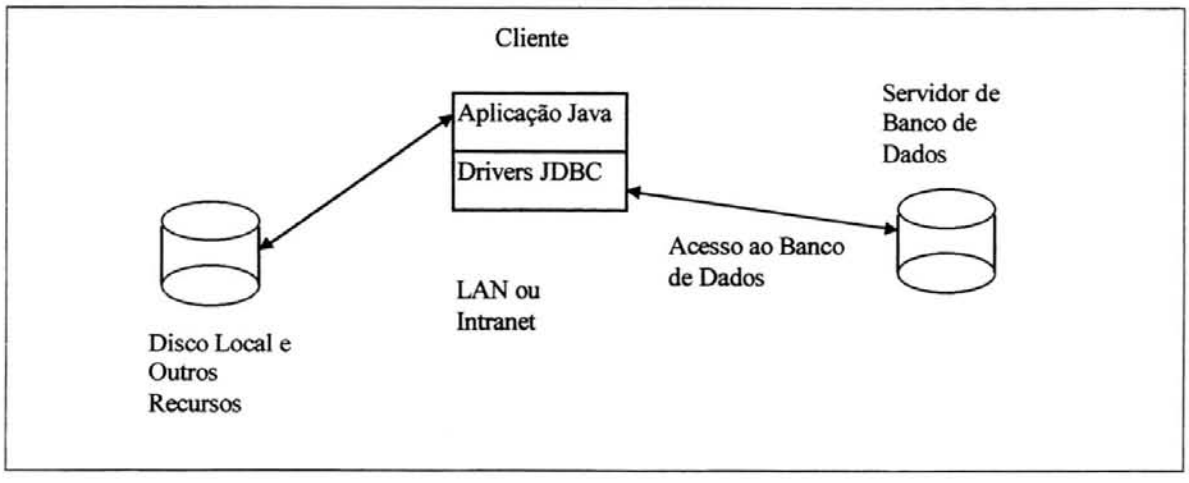

FIGURA 4.15 - Acesso a Banco de Dados utilizando aplicações Java

Outros cenários :

• *Applets* são confiáveis quando convenceram a máquina virtual Java disto. Eles são considerados confiáveis quando são sinalizados com uma chave criptografada

específica ou porque o usuário decidiu que o *applet* é confiável em um código dado [HAM96].

• Acesso Três Camadas (*Three-Tier*) : Neste cenário, aplicações Java criam chamadas para uma "camada intermediária" de serviços na rede, cuja implementações ficam em torno do acesso ao BD. Estas chamadas devem ser criadas completamente por um *Remote Procedure Call* ou por *Object Request Broker* ou em outro caso, a camada intermediária deve ser implementada utilizando o paradigma de objetos [HAM96] .

➔Conexões com Banco de Dados

O poder e as funcionalidades dos SGBD podem ser aplicados diretamente na WWW, permitindo a distribuição e o acesso das informações armazenadas nas bases de dados por um número bastante grande de usuários. Essa integração permite que um usuário, utilizando um *browser*, consulte informações de qualquer banco de dados que esteja conectado à rede Internet, desde que não existam proteções de acesso aos mesmos ou o usuário tenha permissão de acesso aos mesmos.

Com o JDBC é possível abrir conexões com banco de dados, independentemente da localização dos mesmos e também independentemente do banco de dados em questão. Os passos para se conseguir acessar com êxito BD desde aplicações Java são as seguintes :

• Abrir uma conexão;

• Definir a URL do banco de dados;

• Registrar drivers dos bancos de dados;

• Executar comandos SQL;

• Manipular resultados.

↳Abrindo uma Conexão:

Quando se precisar acessar um banco de dados é necessário criar um objeto que execute tal tarefa. Utiliza-se o método java.sql.DriverManager.getConnection (URL). Este método recebe a *Uniform Resource Locators* (URL) do banco de dados como argumento. Por exemplo :

```
Connection con = DriverManager.getConnection("jdbc:odbc:bd_cpgcc"," ", " ");
```

O driver gerenciador do JDBC tentará localizar um driver que possa conectar o banco de dados representado pela URL. O driver gerenciador do JDBC testará cada driver para saber se este poderá conectar com a URL dada. Os drivers examinam a URL para verificar se ela especifica um sub-protocolo que suporte a conexão e se ele tentaria uma conexão com um banco de dados específico.

Se a conexão for estabelecida, então será possível executar comandos SQL e manipular seus resultados.

Escolha entre os Drivers :

Pode, às vezes, ser o caso que muitos drivers JDBC estarem capacitados para a conexão a uma dada URL. Por exemplo, quando a conexão para um dado banco de dados remoto poderia ser possível utilizar um driver JDBC-ODBC *Bridge* ou um driver JDBC para protocolo genérico de rede ou, ainda, utilizar um driver oferecido pelo vendedor do banco de dados.

O JDBC permite usuários especificar uma lista de drivers para "setar" a propriedade "sql.Drivers". Assim, pode-se especificar qual será a seqüência de drivers utilizada pelo JDBC para tentar a conexão com banco de dados.

Definindo a URL :

O WWW padronizou o sistema de nomeação que suporta as seguintes propriedades:

• Drivers diferentes podem utilizar diferentes esquemas para nomear um banco de dados. Por exemplo, um driver JDBC-ODBC *Bridge* pode suportar um estilo ODBC, considerando que um driver de protocolo de rede pode precisar conhecer informações adicionais, mas ele pode descobrir qual máquina e porta conectar.

• Se um usuário "baixar" um *applet* que precise de conexão com um banco de dados, então deveria ser possível a abertura de uma conexão com o banco de dados.

Este mecanismo de nomeação chama-se URL. E é recomendado que o JDBC URL esteja estruturado como :

jdbc:<subprotocolo>:<subname>

Onde o nome de subprotocolo é um gênero particular de mecanismo de conexão a um banco de dados que pode ser suportada por um ou mais drivers. O conteúdo e sintaxe do subnome dependerá do subprotocolo.

Registrando Drivers :

O driver gerenciador do JDBC precisa conhecer quais drivers de banco de dados estão avaliados.

Primeiro a classe do JDBC java.sql.DriverManager é inicializada, ela procurará a propriedade "sql.Drivers" nas propriedades do sistema. Se a propriedade existir e consistir de uma lista de drivers, o *Driver Manager* armazenará cada driver da lista.

O programador por sua vez, pode explicitamente, realizar a leitura do driver utilizando o método padrão *Class.forName*, como por exemplo :

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Em ambos os casos é de responsabilidade de cada novo driver o seu registro junto ao driver gerenciador do JDBC.

➔ Considerações de Segurança

A segurança é um dos principais conceitos que as empresas se preocupam ao compartilhar os dados de seus banco de dados na Internet ou em Intranets. Primeiro consideremos a segurança do servidor de banco de dados. Se for permitida uma conexão direta com o servidor de banco de dados, os programas Java/JDBC necessitarão preparar um bom suporte de segurança[NEW96][SHA96b].

A segurança na requisição do banco de dados e na transmissão de dados na rede, especialmente no caso da Internet deve ser considerada pelos programas Java/JDBC [SHA96b]. Os protocolos de rede usados para acessar banco de dados tem sido geralmente fixados pelo vendedor do SGBD ou pelo vendedor da conectividade. Entretanto, um programador Java/JDBC deve verificar que protocolo de rede fornece segurança adequada para utilizar diante de *drivers* JDBC e servidores de SGBD.

Por causa dos *drivers* JDBC poderem ser utilizados em uma variedade de diferentes situações, é importante que os *drivers* sejam escritos seguindo simples regras para prevenir que *applets* criem conexões ilegais a banco de dados. Estas regras são desnecessárias se os *drivers* são enviados juntos com os *applets*.

Estas regras são aplicadas na abertura da conexão. Este é o ponto quando o *driver* e a máquina virtual verificam que a atual chamada realmente possui permissão para a conexão para o banco de dados. Após a abertura da conexão não são necessárias verificações adicionais [SHR96].

5 Projeto de um Ambiente de Modelagem Distribuída

Este capítulo descreve o projeto de um ambiente de desenvolvimento de sistemas de informação na Internet, apresentando os principais requisitos que este tipo de ambiente deve possuir. Apresenta, ainda, o projeto do EDI, suas funcionalidades e características. Por fim, é apresentado o modelo semântico do dicionário de dados do ambiente.

5.1 Requisitos de um ADS na Internet

O processo de desenvolvimento de *software* envolve uma contínua produção e transformação de notações, partindo-se da especificação de requisitos até a produção de código executável [PEN93][FIN94]. Estas produções e transformações podem ser suportadas por ferramentas que constituem os ambientes de desenvolvimento de *software*. Na produção de *software*, várias ferramentas e notações são utilizadas para atender as fases de seu desenvolvimento[BRO92]. O que ocorre com estas ferramentas é o isolamento ou independência uma das outras, dificultando o relacionamento entre os documentos utilizados e/ou gerados durante este processo.

Para o desenvolvimento de sistemas diversas metodologias foram propostas como, por exemplo OMT [RUM91], UML[FOW97], Booch [BOO91] e OOSE [JAC92]. Estas metodologias fornecem descrições de como o *software* deve ser criado e mantido [ORT95]. Em geral, estas descrições estão baseadas em um conjunto de documentos produzidos durante o ciclo de vida, definindo a estrutura e os relacionamentos desses documentos.

Tendo em vista a popularização da Internet, principalmente com a tecnologia de hiperdocumentos, e o surgimento da linguagem de programação Java, se faz cada vez mais necessária a incorporação de tecnologias que suportem o desenvolvimento de *software* de forma distribuída na rede. Estas tecnologias devem possibilitar aos desenvolvedores atuar nas diversas etapas existentes do ciclo de desenvolvimento do *software* gerando e trocando suas informações (documentos) entre si como se estivessem num mesmo ambiente local.

O desenvolvimento de sistemas de informação na Internet necessita de um suporte à criação colaborativa de documentos para que estes sejam concluídos de forma rápida, eficaz e com qualidade. Segundo [BOR95] o desenvolvimento de sistemas não é uma tarefa trivial, exigindo que grupos de pessoas com diferentes formações cooperem. Em grandes projetos os desenvolvedores passam 70% do tempo trabalhando de forma colaborativa [VES95].

Por isso os ADS's devem possuir ferramentas para permitir a colaboração nas tarefas do desenvolvimento de sistemas. Segundo [REI95] a colaboração expressa o ato de operar simultaneamente, trabalhar em comum, cooperar no desenvolvimento de tarefas. Esta colaboração deve ser, para o usuário, transparente e independente de qualquer plataforma de *hardware* e *software*.

A colaboração em um ADS na Internet possibilita que uma tarefa seja "desenvolvida" por diversos especialistas (colaboradores) habilitados em diferentes localidades. A colaboração na autoria de documentos do ciclo de desenvolvimento deve possuir as seguintes funcionalidades [SOU98] :

- Gerenciamento do depósito de documentos;

- Acesso remoto para diferentes tipos de usuários.

Um ADS que possui a Internet como seu principal ambiente de interação permite aos seus usuários a independência de plataforma e de localidade, uma vez que as ferramentas do ambiente podem ser acessadas de qualquer nodo da rede. Além disso, um ADS na Internet apresenta requisitos importantes que o distingue de outros ADS. São elas:

- *Cooperação :* Um ADS na Internet utiliza recursos de comunicação entre os usuários para que estes possam trocar mensagens, a fim de executar uma determinada tarefa ou trocar experiências [REI95][DAL97]. Os principais meios de comunicação são: *eletronic-mail (e-mail)* e *chat.* Outra forma de cooperação é o compartilhamento de informações através da base de dados ou através de documentos HTML (*HyperText Markup Language*). Com isto podem ser compartilhadas ferramentas, informações, diagramas; Além disso, os usuários poderão anotar documentos criados por outros usuários, podendo comentar ou até mesmo, sugerir a inclusão de novos conceitos nos modelos.

- *Coordenação :* É a capacidade de gerenciar as interações dos usuários controlando o fato de diversas pessoas estarem trabalhando nos mesmos objetos ao mesmo tempo e, também, de integrar as contribuições individuais de cada participante num único produto;

- *Segurança :* A segurança consiste em garantir que apenas usuários autorizados terão acesso à base de dados comum. Diferentes tipos de usuários possuem diferentes visões dos dados armazenados e diferentes níveis de acesso. Além dos mecanismos de segurança implementados pelos sistemas de gerenciamento de banco de dados, a Internet provê segurança através dos *browsers* e dos mecanismos existentes na linguagem de programação Java;

- *Autoria :* A autoria consiste em fornecer mecanismos que possibilitem o planejamento da atividade do grupo de trabalho e a representação da mesma no ambiente compartilhado. A Internet facilita o trabalho de autoria, principalmente, pela visualização de documentos através dos navegadores (*browsers*) e pelo fácil acesso às informações;

- *Gestão do Dicionário de Dados :* O dicionário de dados é o repositório de todas as informações do projeto, modelagem, implementação e testes. O ADS deve gerenciar o acesso ao dicionário de dados garantindo a consistência e segurança dos dados. Os editores manipulam as interfaces gráficas e armazenam no dicionário de dados todas às informações semânticas. Com o dicionário de dados disponível na Internet os usuários não se preocupam com a localização das informações, principalmente se não estiverem em seu ambiente de trabalho;

- *Consistência dos Modelos :* O ADS deve garantir a consistência dos modelos com relação ao dicionário de dados. Um dicionário de dados integra todos

os modelos das notações. A Internet, através dos mecanismos de compartilhamento das informações e coordenação das tarefas dos usuários garante a consistência dos modelos;

- *Homogeneidade dos Editores :* A Internet auxilia na adaptação das interfaces ao usuário através da homogeneidade de interfaces proporcionadas pelos navegadores;

- *Adaptação ao Usuário :* Como diversas visões de um mesmo conjunto de dados são manipuladas pelo ADS é possível a adaptação do ambiente às características do usuário;

- *Interação Global* : A Internet é, sem dúvida, o maior sistema aberto existente atualmente. Com ele diversos usuários podem "navegar" em busca de informações disponibilizadas através das páginas Web. Esta interação global permite um melhor compartilhamento de bibliotecas de componentes, a ampla exposição dos produtos gerados nas fases de desenvolvimento, o acesso a *frameworks* e o compartilhamento de resultados e experiências;

Adicionalmente um ADS na Internet deve possuir características comuns a outros ADS como, por exemplo, geração de código e engenharia reversa.

5.1 Uma Proposta de ADS na Internet

Um ADS na Internet deve possuir as características citadas anteriormente. Além disso deve possuir ferramentas (editores) capazes de auxiliar o desenvolvimento colaborativo de *software*. Um protótipo de editor colaborativo desenvolvido em Java é apresentado nas seções subseqüentes.

O ADS na Internet oferece homogeneidade de interfaces conseguida através das visualizações nos navegadores. Este ambiente possui um dicionário de dados armazenado em uma base de dados no servidor. Esta base de dados implementa o modelo conceitual.

Os editores deste ambiente interagem com os usuários através de visões específicas da base de dados. Por exemplo, editor de cenários visualiza objetos e a troca de mensagens entre eles. Editor de objetos visualiza a estrutura de herança e associação entre classes. Editor de estados visualiza as transformações de estado dos objetos de uma classe.

Uma mesma visão pode ter diferentes apresentações, conforme o desejo do usuário. A visão do modelo de objetos pode ser vista com letras em negrito e fonte *Arial* por um usuário e itálico e *Times New Roman* por outro.

Pode-se especificar que diferentes usuários de documentos podem requerer visões específicas. Algumas visões são sobre sub-conceitos específicos (por exemplo, a modelagem dinâmica de objetos), outras visões referem-se a forma de visualização de conceitos (por exemplo, alguns usuários preferem ver diagramas de objetos com a notação de Rumbaugh [RUM91], outros com a notação do Booch [BOO91]). Detalhes

das visões que são apenas referências de apresentação dos usuários (por exemplo, tipo da fonte, tamanho da fonte e cor) são especificadas nas anotações.

O ADS suporta trabalho individual ou colaborativo. A colaboração pode ser de várias formas como troca de mensagens através de e-mail e chat, compartilhamento de informações através dos modelos visualizados pelos usuários ou na forma de anotações sobre modelos que estão sendo criados. Por exemplo, um usuário cria um modelo na sua visão e o envia a outros usuários que podem criticar e sugerir modificações, utilizando anotações sobre o modelo proposto. Estas anotações podem ser na forma de inclusão, remoção, substituição ou comentário.

Projetos possuem diversos desenvolvedores que podem estar distribuídos em diversas localidades e que podem realizar tarefas distintas sobre os documentos, ou mesmo colaborar na realização de uma mesma tarefa. O ambiente distribuído suporta este tipo de trabalho, controlando o acesso (prioridades de atualizações, autorizações, gestão de visões e outras) e gerenciando os documentos.

O ambiente proposto deve possuir diversas ferramentas de apoio ao desenvolvimento de software. Entre estas ferramentas podemos citar: editor de objetos, editor de cenários, editor de diagramas de ação para métodos, editor textual de dicionário de dados e geradores de código. Uma particular ferramenta é a do gerente de projeto, que deve alocar desenvolvedores para as tarefas.

O ambiente possui uma base de dados comum, onde estão armazenadas as informações dos sistemas. Esta base de dados deverá representar todos os conceitos manipulados pelas ferramentas, isto é, apresentar no modelo conceitual todos os conceitos presentes nas diversas visões que as ferramentas específicas manipulam.

5.2 Funcionalidades e Características do EDI

O editor diagramático para a Internet (EDI) é um editor de diagramas com características próprias, tais como: vários usuários podem realizar diversas tarefas trocando informações e compartilhando uma única base de dados; cada usuário pode visualizar de forma específica um mesmo conjunto de dados; independência de plataforma, uma vez que a ferramenta é executada sobre navegadores Web (Netscape, Internet Explorer, Mosaic, etc.).

Vários usuários acessando a ferramenta ao mesmo tempo e requerendo diversas visualizações sobre a mesma base de dados, requer do sistema capacidade de gerenciar estas visões e a atualização da base de dados. Estas visualizações são conseguidas através da aplicação da arquitetura *Model/View/Controller* (MVC) [PAR94] [BRA95a] descrita na seção 4.2.1.

No protótipo o modelo é representado pela base conceitual. Esta base é gerenciada por um sistema de banco de dados e representa os conceitos utilizados pelas representações das notações. Na seção 5.3 são apresentados detalhes do modelo conceitual (dicionário de dados). As visões são dependentes de como cada usuário visualiza as informações da base de dados. Cada visão é apresentada pelo editor específico da visão. A atualização da base de dados e das diversas visões se dá de forma

consistente, pois, quando o modelo é alterado (base de dados), automaticamente as visões agregadas são notificadas da alteração.

Quando uma determinada tarefa é iniciada (isto é, quando um editor de uma particular visão é ativado), somente um usuário receberá a especificação desta visão (usuário proprietário). Analisando a especificação da tarefa, este usuário poderá requisitar o bloqueio dos conceitos que, possivelmente poderão sofrer alterações. Estes conceitos variam de acordo com o editor utilizado para o desenvolvimento da tarefa como, por exemplo, se o usuário utilizar o editor de objetos, as classes, atributos, métodos, associações, agregações e heranças utilizadas na modelagem da tarefa deverão ser bloqueados para que as alterações realizadas sobre estes conceitos não tornem os modelos extraídos a partir destes conceitos inconsistentes.

Quando o usuário (proprietário ou colaborador) iniciar uma tarefa, isto é, criar ou manipular uma visão, uma cópia dos conceitos existentes no dicionário de dados é gerada formando o dicionário local do usuário. Este dicionário local conterá todos os conceitos manipulados pelo usuário.

Quando o usuário proprietário estiver satisfeito com sua modelagem poderá desbloquear o modelo. No entanto, o usuário proprietário poderá solicitar a colaboração a um grupo de trabalho ou grupo de discussão para que haja troca de experiências e sugestões entre vários usuários. O usuário proprietário compartilhará parte ou todo modelo criado com um ou vários *usuários colaboradores*. Estes usuários receberão autorização para acessar os modelos em questão. Os usuários colaboradores só manipulam visões, não possuindo autorização para atualização do dicionário de dados.

Cada usuário, tanto proprietário como colaborador, possui um perfil (*profile*) contendo características específicas de apresentação e criação dos modelos como, por exemplo, a cor com que os objetos gráficos de sua autoria serão apresentados pelo editor.

Os usuários, tanto proprietário como colaborador, possuem um conjunto de funções que foram previamente estabelecidas. A Tabela 5.1 apresenta estas funções.

TABELA 5.1 – Funções dos Usuários

Tipo de Usuário	**Funções**
Proprietário	Criar visão Bloquear modelo (ou componentes dele) Editar visão Incluir visão no modelo (ou componentes) Editar anotação Desbloquear modelo Enviar visão para análise dos colaboradores
Colaborador	Receber visões Anotar visões Devolver visões Enviar visões para análise de outros colaboradores

Na função de editar visão o usuário proprietário poderá incluir, excluir ou modificar componentes. Estes componentes são visões de conceitos existentes no dicionário de dados, isto é, quando o usuário estiver editando a visão estará editando o dicionário de dados.

Outra função do usuário proprietário é a edição de anotações. Nesta função o usuário proprietário deverá analisar as anotações feitas pelos usuários colaboradores, podendo aceitá-las, rejeitá-las ou discuti-las com o(s) colaborador(es). O dicionário de dados do modelo só será atualizado quando o usuário aceitar a anotação.

A consistência dos modelos é garantida pelo dicionário de dados. Cada usuário possuirá uma cópia do dicionário de dados. Esta cópia é referente aos conceitos manipulados pelo usuário. Quando o usuário, por exemplo, desejar incluir uma nova classe no diagrama, o editor verificará a sua existência na cópia do dicionário de dados. Se esta classe não existir será verificado o dicionário de dados global do modelo. Encontrando o conceito referente a classe a ser incluída, este é incluído na cópia do dicionário do usuário e incluído na visão como sendo um componente criado pelo usuário proprietário. Caso não exista o conceito no dicionário de dados o editor não impõe nenhum tipo de restrições a operação realizada pelo usuário.

5.2.1 Colaboração no EDI

Entre as funcionalidades do editor pode-se salientar o mecanismo de colaboração no desenvolvimento das tarefas. As colaborações podem ser diretas ou indiretas.

A colaboração indireta ocorre através do compartilhamento de documentos, isto é, os diversos usuários do editor poderão, através de um sistema de arquivos, compartilhar documentos (diagramas, textos e imagens).

Colaboração direta ocorre através de mecanismos de troca de mensagens e anotações. Os usuários (desenvolvedores) podem trocar mensagens (informações e experiências) através de protocolos de comunicação (correio eletrônico (e-mail) e *chat*) implementados em Java. Estes programas agregados ao ambiente permitem a formação de *workgroup* entre usuários, ou envolvidos na mesma tarefa ou especialistas em uma determinada área. O mecanismo de *chat* permite uma interação entre todos os usuários do editor ou somente entre os usuários cadastrados no mesmo grupo de trabalho.

As informações podem ser compartilhadas entre o usuário proprietário e os usuários colaboradores. Quando o usuário proprietário inicia a modelagem de um sistema cria sua visão. Esta visão poderá ser armazenada no dicionário de dados ou então compartilhada com os usuários colaboradores. Assim o usuário proprietário poderá solicitar a colaboração a um grupo de trabalho ou grupo de discussão para que haja troca de experiências entre vários usuários.

Quando o usuário colaborador auxilia na criação do documento referente a tarefa de modelagem, ele somente irá inserir anotações no documento. Estas anotações não refletirão alterações no dicionário de dados, sendo esta uma tarefa exclusiva do usuário proprietário.

O usuário proprietário fica aguardando a colaboração dos usuários para poder encerrar a tarefa e atualizar o dicionário de dados. O ciclo de desenvolvimento da tarefa pode ser visualizado no diagrama de estados apresentado na Figura 5.1. Esta ciclo envolve o documento desde a sua elaboração até a sua conclusão.

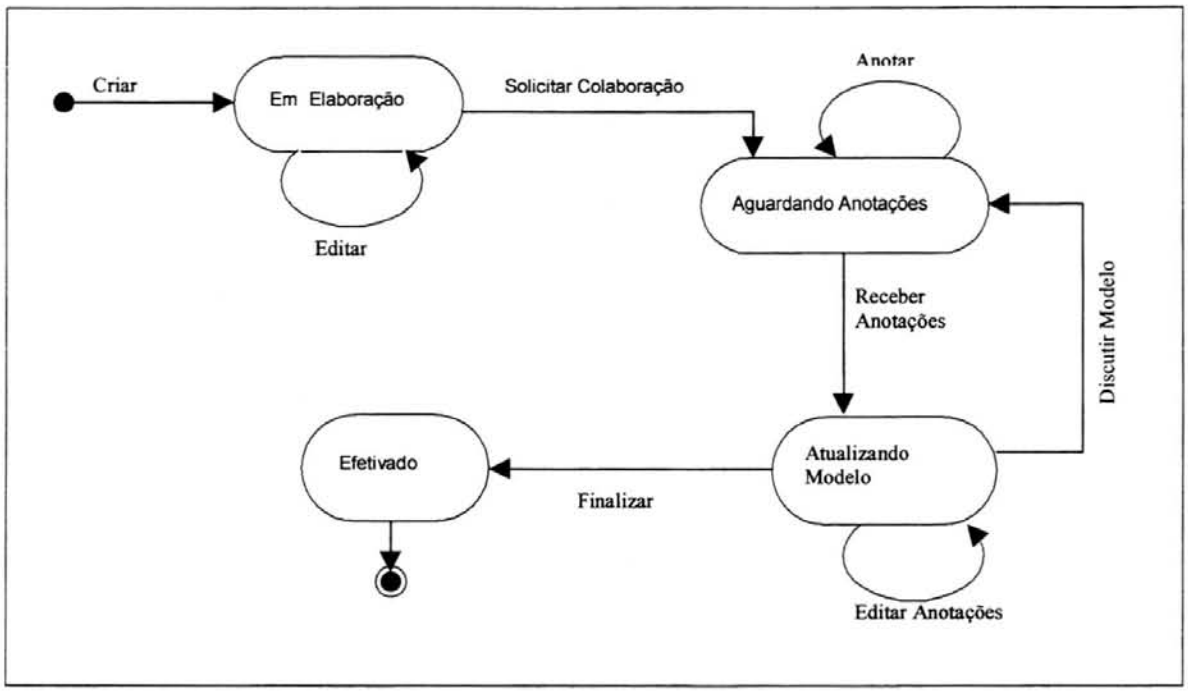

FIGURA 5.1 - Ciclo de vida dos Documentos – Usuário Proprietário

Estes estados podem ser descritos como segue:

- "*Em Elaboração*" : Neste estado o usuário proprietário cria a primeira visão da tarefa que está sendo realizada;

- "*Aguardando Anotações*" : Neste estágio o usuário proprietário aguarda que os usuários colaboradores anotem o documento a eles deferidos. Neste período o usuário proprietário pode visualizar as anotações que foram realizadas;

- "*Atualizando Modelo*": Neste estado o dicionário de dados do modelo será atualizado com as informações das visões;

- "*Efetivado*" : Nesta fase, a tarefa estará concluída e incorporada ao modelo conceitual que é desbloqueado.

Os eventos associados as mudanças de estados são:

- *"Editar"*: Neste evento o usuário proprietário poderá manipular os componentes do documento criado. Podendo incluir novos conceitos, excluir conceitos existentes ou alterá-los;

- "*Solicitar Colaboração*" : Neste ação o usuário solicita a colaboração na execução da tarefa a outros usuários, chamados de usuários especialistas ou usuário colaboradores;

- *"Anotar":* O usuário colaborador poderá, através deste evento, incluir anotações no modelo. Poderá propor a inclusão, exclusão ou substituição de um ou vários componentes do modelo;

- *"Editar Anotações":* Nesta ação o usuário proprietário confirmará (aceitará) as anotações que estiverem de acordo com a sua visão da solução da tarefa e rejeitará as demais. Os aceites passam a fazer parte da sua visão;

- *"Discutir Modelo":* Nesta ação o usuário poderá contestar as anotações realizadas pelos colaboradores, solicitando maiores informações;

- *"Receber Anotações":* Após as inserção das anotações feita pelos usuários colaboradores, o usuário proprietário irá receber estas anotações para que possa analisá-las (incluindo-as ou não no dicionário de dados);

- *"Finalizar":* Neste evento, todos os conceitos utilizados no modelo são desbloqueados e o documento passa para um estado de efetivado.

Quando o usuário colaborador recebe uma solicitação para auxiliar na execução de uma determinada tarefa (construção de um documento), este faz anotações sobre o documento disposto terminando sua colaboração para a elaboração do documento.

5.2.2 Anotações no EDI

A colaboração através de anotações é uma forma de comunicar idéias ou opiniões sobre o conteúdo de um documento criado [CAT89]. Um usuário colaborador analisa o modelo recebido, insere suas anotações junto com o diagrama e "envia" ao usuário proprietário do diagrama. Estas anotações são inseridas com o intuito de aperfeiçoar o documento, e também podem ser utilizadas como registro das argumentações ocorridas durante a criação do diagrama. Além disso as anotações servem como um mecanismo de comunicação entre o usuário proprietário e os usuários colaboradores.

As anotações são utilizadas para: apresentar idéias ou opiniões (inserir comentários) e alterar o modelo (substituição, inclusão ou remoção de um componente); Toda a anotação poderá ter associada um comentário explicativo, conforme é apresentado no modelo de objetos da Figura 5.2.

Uma anotação é uma visão que, pode ou não corresponder a conceitos existentes no dicionário de dados. Porém, como já foi salientado, o usuário colaborador somente poderá trabalhar sobre visões, não sendo permitido atualização do dicionário de dados do ambiente. Somente será atualizado a cópia do modelo que acompanha a visão do usuário colaborador.

Apresentado o modelo de anotações que o EDI utiliza, inicialmente baseado no modelo apresentado em [SOU98] são identificadas as operações que podem ocorrer na inclusão de anotações nos diagramas desenvolvidos.

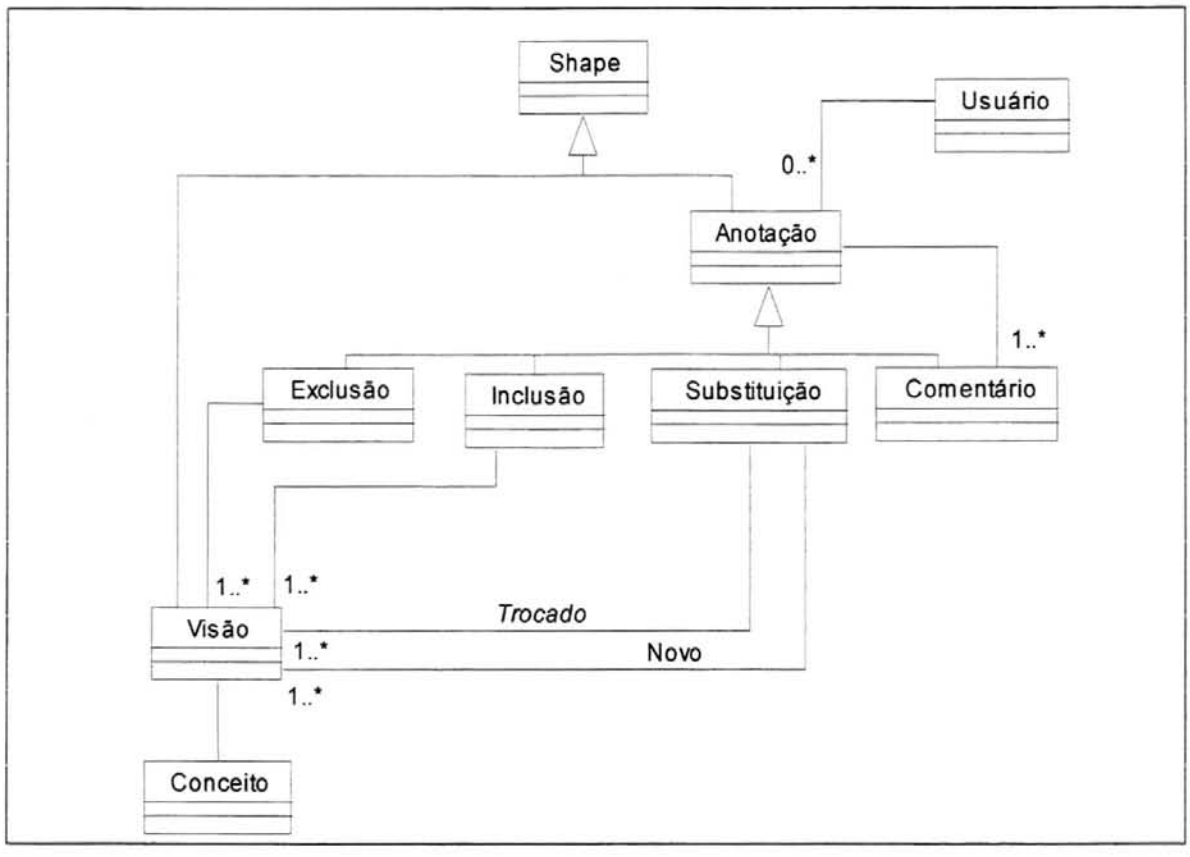

FIGURA 5.2 – Modelo de Objetos para Anotações

O usuário colaborador, além das operações básicas (incluir, excluir e alterar) executadas sobre objetos definidos por ele, poderá anotar componentes já definidos por outros usuários (proprietário ou outros colaboradores). A seguir são apresentas todas as operações semânticas que um usuário colaborador pode realizar sobre um modelo de um usuário proprietário, em um editor de diagrama de classes (visão).

- **Substituição**

Com este tipo de anotação o usuário poderá substituir uma determinada visão (representação gráfica de um ou mais conceitos do dicionário de dados) por outro semanticamente equivalente. Por exemplo, substituir uma classe definida pelo usuário proprietário por outra classe (definida pelo usuário colaborador).

A anotação de substituição de componentes permite que os usuários colaboradores proponham alterações de desenho sem alteração do dicionário de dados. A Figura 5.3 apresenta um exemplo de anotação de substituição realizado por um usuário colaborador, em que este usuário propõem a substituição de um conjunto de componentes por um outro componente. A Figura mostra, ainda, como ficaria o modelo caso o usuário proprietário aceitasse a anotação proposta pelo colaborador.

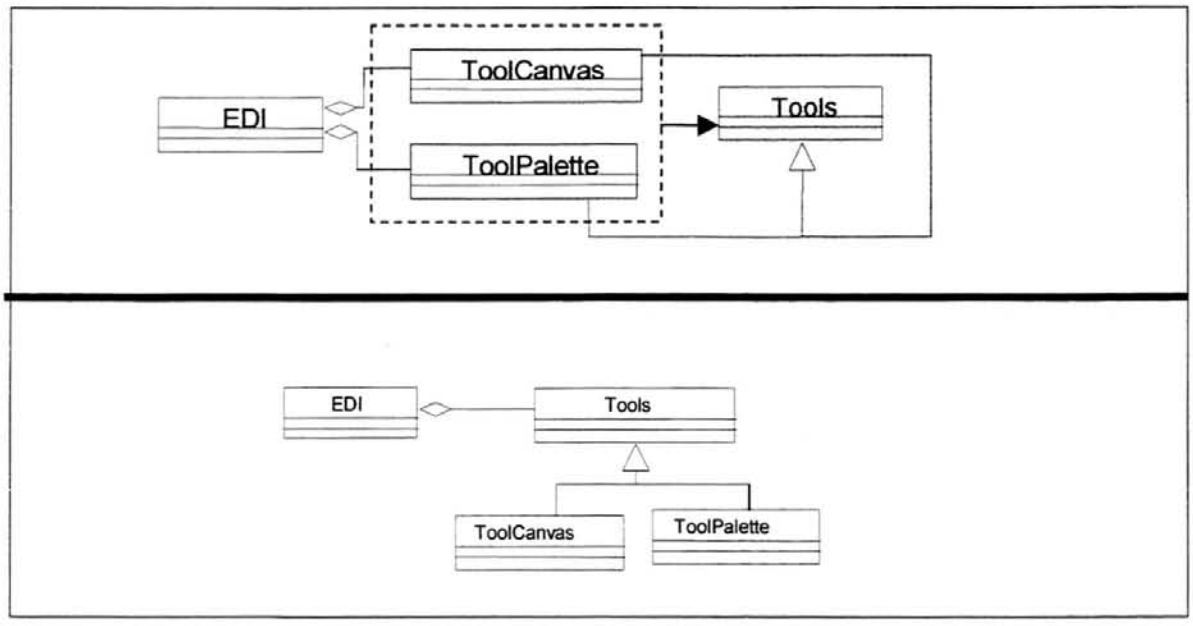

FIGURA 5.3 - Exemplo de Anotação de Substituição

A anotação de substituição pode proporcionar ao usuário colaborador maior flexibilidade na manipulação dos componentes do modelo. Cada componente do modelo pode ser transformado (como anotação) em outro componente existente no contexto da notação utilizada pelo EDI.

Quando este tipo de anotação é realizada sobre o modelo, o EDI verificará quais informações existentes no componente a ser transformado serão (re)utilizadas no novo componente. A Figura 5.4 apresenta um exemplo ilustrando substituições de um atributo em uma classe e de um relacionamento de herança em um relacionamento de agregação.

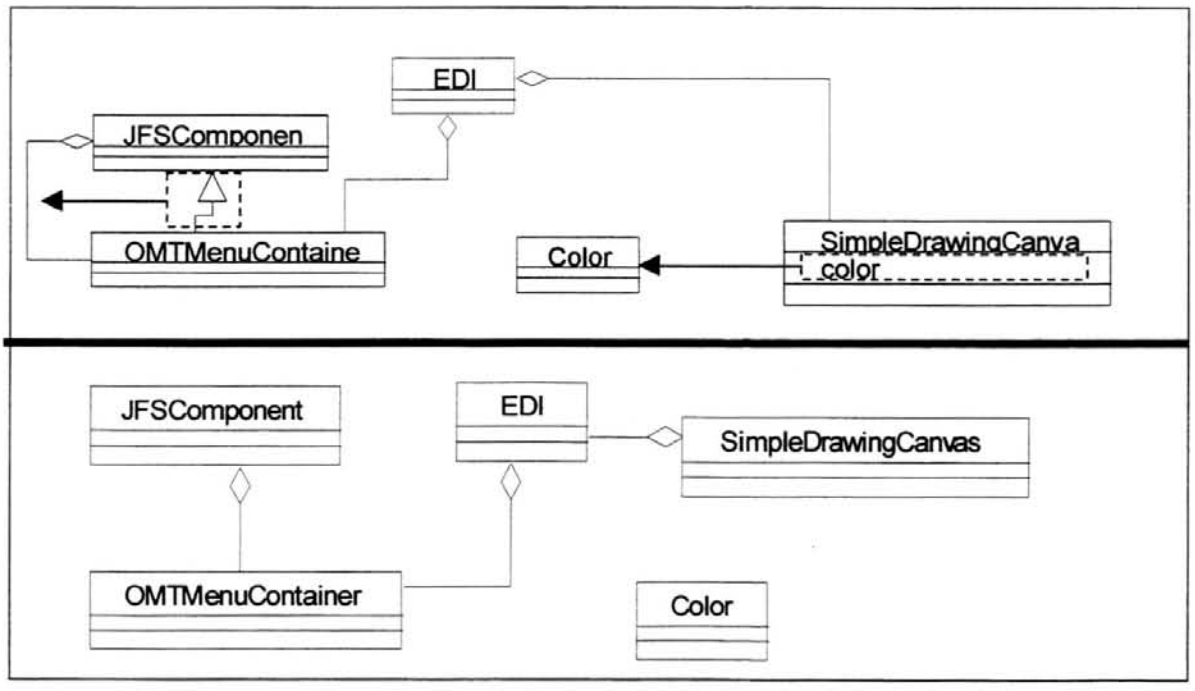

FIGURA 5.4 – Exemplos de Anotações de Substituição

➢ **Inclusão**

O usuário colaborador pode incluir novos componentes no modelo. Estes novos componentes são verificados contra o dicionário de dados. Se forem novos não são incluídos no D.D., apenas na cópia do D.D. do usuário colaborador, isto é, a criação de novos componentes ocorre apenas localmente e irá refletir no D.D. geral quando o usuário proprietário aceitar a anotação. Se o componente já existir no D.D., este é incluído no diagrama (com a identificação do proprietário) e, também, na cópia do D.D. do usuário colaborador, conforme descrito anteriormente.

Pode-se definir a anotação de inclusão como sendo uma simples proposta de inclusão do colaborador ao proprietário do modelo. Para tanto, o usuário colaborador deverá escolher qual o componente que deseja incluir no modelo e inseri-lo. A Figura 5.5 mostra um exemplo de anotação de inclusão. O componente com a cor diferenciada do resto do modelo é a anotação realizada pelo usuário que possui a cor vermelha como sua identificação.

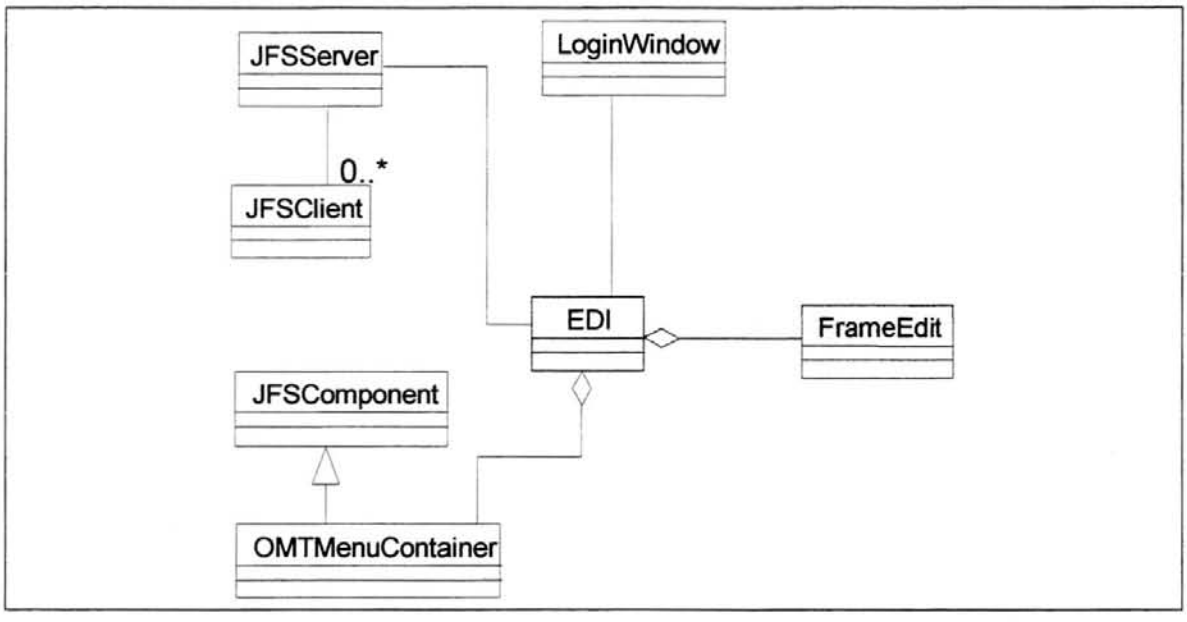

FIGURA 5.5 – Exemplo de Anotação de Inclusão

➢ **Exclusão**

Com esta anotação o usuário colaborador poderá excluir componentes anteriormente definidos. Porém esta exclusão só será efetuada, isto é, o objeto só será realmente excluído se o colaborador estiver excluindo um componente criado por ele. Se o objeto a ser excluído tiver sido definido por outro usuário (proprietário ou outro colaborador) esta exclusão será na cópia do dicionário de dados local, na qual será indicado, graficamente, a exclusão e só será efetivada pelo usuário proprietário na edição das anotações.

A Figura 5.6 apresenta um exemplo de anotação de exclusão realizada por um usuário colaborador. Este usuário propõem a exclusão de um conjunto de componentes inseridas pelo usuário proprietário.

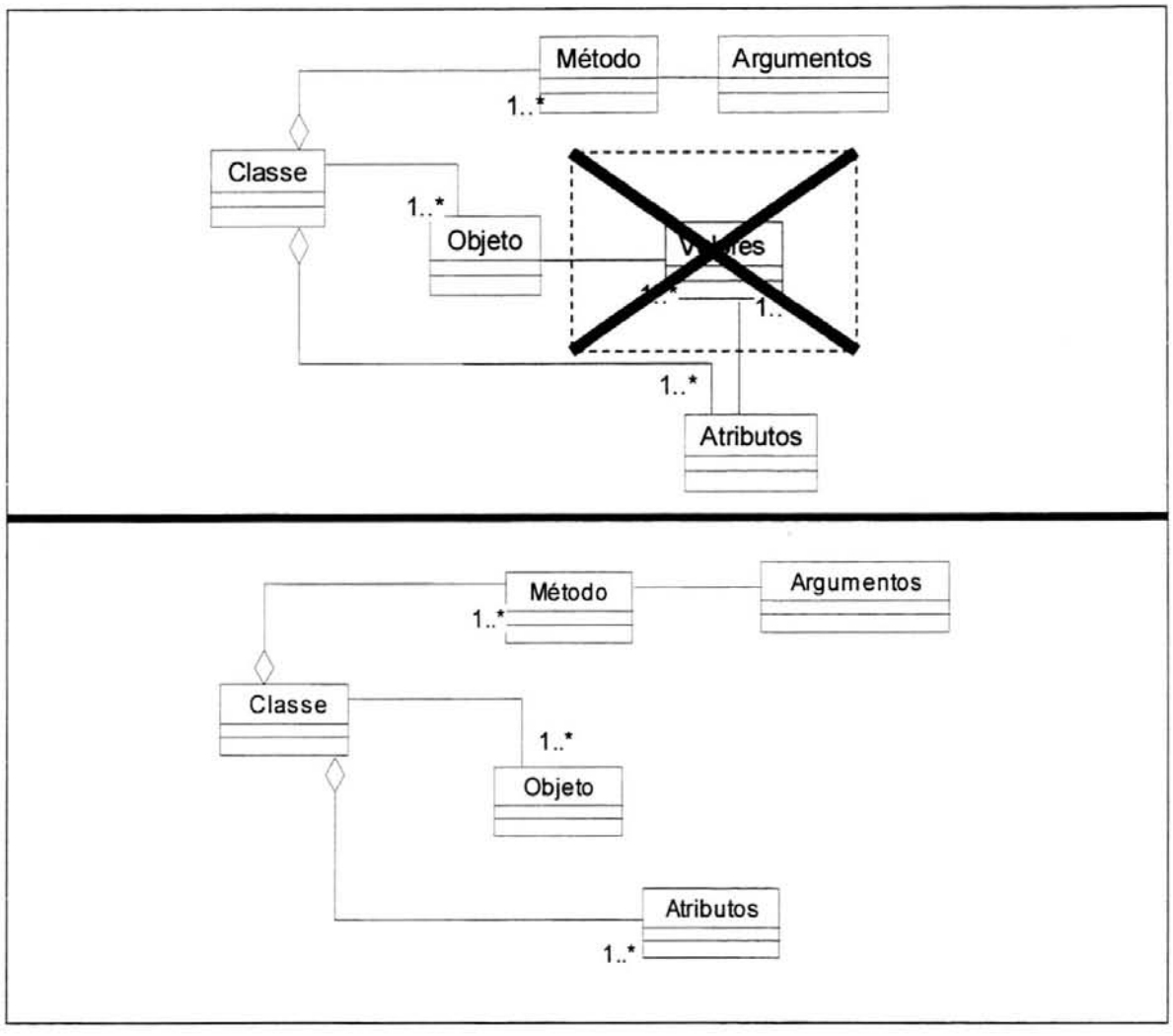

FIGURA 5.6 – Exemplo de Anotação de Exclusão

➢ **Comentário**

Tanto o usuário colaborador quanto o usuário proprietário podem inserir comentários no modelo que está sendo criado. Estes comentário são utilizados para elucidar e/ou explicitar idéias propostas através de anotações.

Estes comentários podem ser gerais ou específicos. *Gerais* quando não estão ligados a componentes do modelo; e *específicos* quando estão diretamente ligados a uma visão (agregados a alguma anotação, ou simplesmente a um componente). A Figura 5.7 mostra estes dois tipo de comentários, ambos, neste exemplo, definidos pelo usuário colaborador.

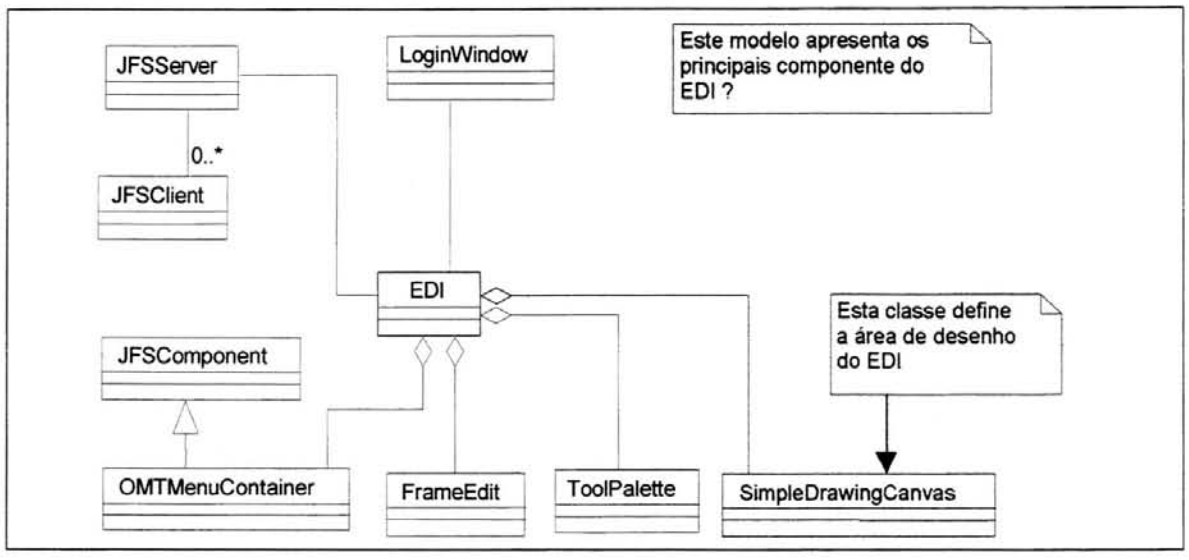

FIGURA 5.7 – Exemplo de Comentários no Modelo

5.3 Modelo Semântico do Dicionário de Dados

Para o modelo de dados, utilizado para descrever o modelo conceitual do domínio da aplicação, foi utilizado a notação UML (*Unifield Modeling Language*) [FOW97]. Este modelo fornece uma estrutura semântica que facilita a descrição do domínio e a representação dos conceitos utilizados pelas notações utilizadas nas ferramentas do ambiente proposto.

O ambiente possui uma base de dados comum (dicionário de dados do ADS) onde estão armazenadas as informações dos sistemas. Esta base de dados deverá representar todos os conceitos manipulados pelas ferramentas, isto é, apresentar em modelo conceitual (Figura 5.8) todos os conceitos presentes nas diversas visões que as ferramentas específicas manipulam.

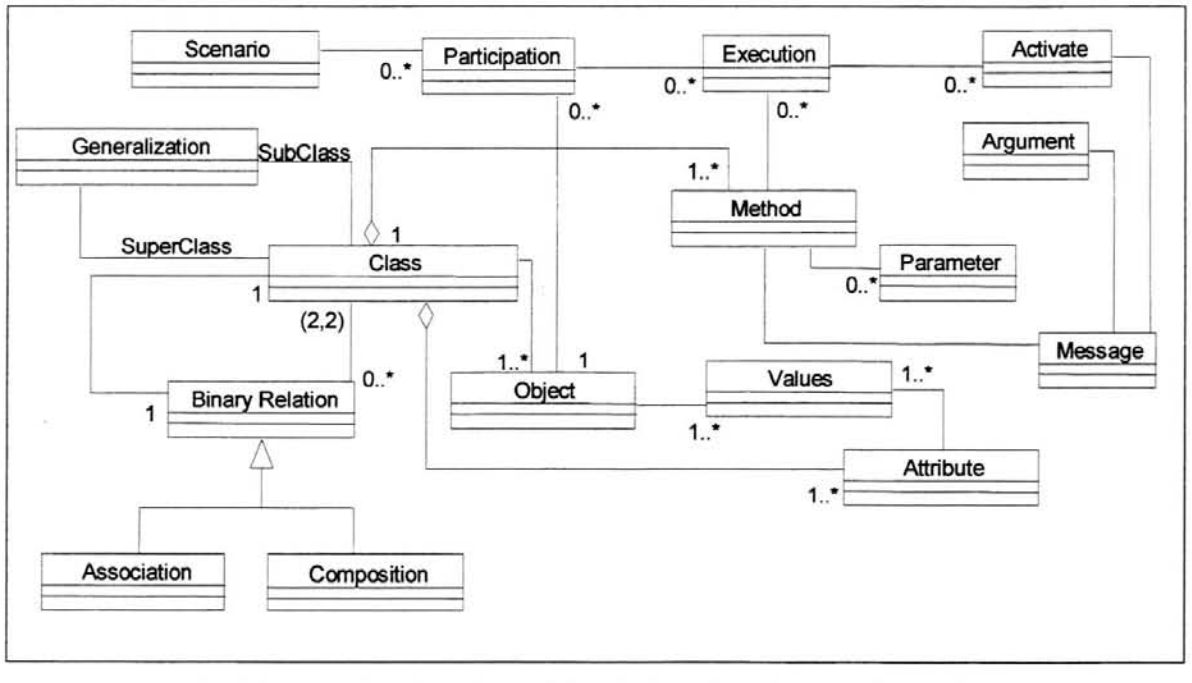

FIGURA 5.8 - Modelo Parcial de Objetos Semânticos do Ambiente

A partir deste modelo conceitual cada editor representará parte destes conceitos de forma gráfica, de acordo com a notação utilizada. Cada notação deverá ter um editor próprio, especializado do *framework* de edição genérico. Cada editor manipula um submodelo conceitual.

Cada conceito existente no modelo conceitual deve possuir a visão correspondente no EDI. No protótipo construído (EDI para modelagem de objetos) cada conceito relativo a notação utilizada (conceito de classe, método, atributo, etc.) possui suas classes de visão correspondente (classShape, methodShape, AttributeShape, etc.). A Figura 5.9 apresenta a relação entre o conceito do dicionário de dados e suas respectivas classes de visão.

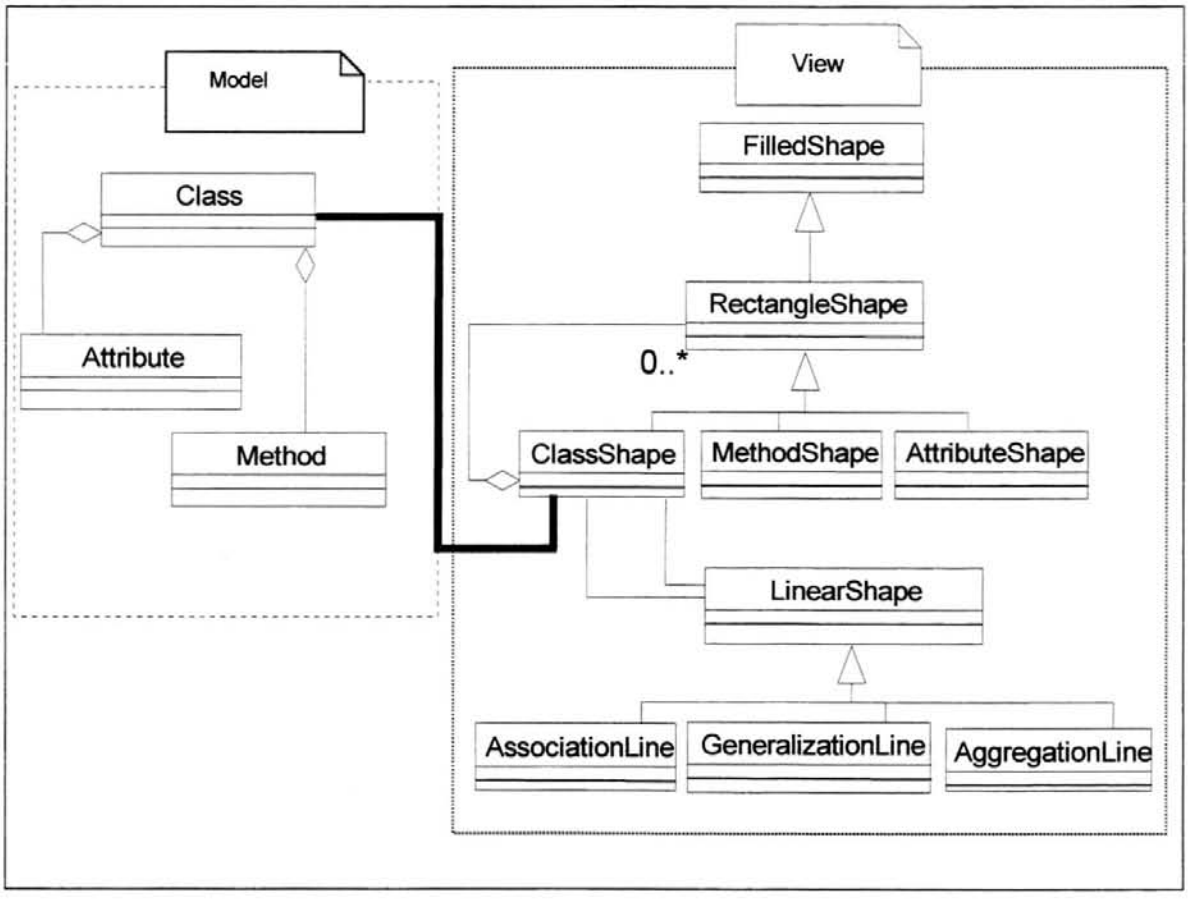

FIGURA 5.9 – Relação entre Modelo Conceitual e Modelo de Visão (Model X View)

Além disso, cada componente de visão deve possuir um controlador associado para controlar a interação do usuário com a visão. A Figura 6.5 mostra o relacionamento existente entre os componentes visuais (visão) e seus respectivos controladores.

Visualizando a Figura 5.9 pode-se observar a extensibilidade do modelo para a construção de novos editores (EDI). Na implementação de um novo editor (com uma nova notação) deverão ser identificados as características visuais dos componentes a serem manipulados. Figuras fechadas (como, por exemplo, classe na modelagem de objetos) serão subclassificadas da classe FilledShape, e figuras lineares (como, por exemplo, associação no modelo de objetos) serão subclassificadas da classe LinearShape.

Com isso, a extensibilidade do ambiente proposto torna fácil e rápido a inclusão de novas ferramentas, uma vez que todo o comportamento genérico dos componentes gráficos já estão implementados, restando somente a especialização dos novos componentes. O Capítulo 6 apresenta maiores detalhes da implementação do EDI.

6 Implementação de um Protótipo Experimental em Java

Neste capítulo são apresentadas as características de implementação do EDI (Editor de Diagramas para a Internet). O EDI é uma ferramenta construída com a capacidade de proporcionar o desenvolvimento cooperativo de sistemas na Internet [POM98b]. Este protótipo foi construído com a finalidade de comprovar a aplicação das características que um ADS deve possuir quando tiver a Internet como sistema de operação.

6.1 Características Gerais

Para se conseguir toda a funcionalidade do EDI foi necessário a utilização de uma linguagem de programação que suporte a implementação das características citadas anteriormente. A utilização da linguagem de programação Java [COR96][FLA97] se deve ao fato dela possuir mecanismos e bibliotecas de classes que auxiliaram na implementação das características da ferramenta.

Assim, a independência de plataforma é obtida através da implementação dos *applets*, programas Java que são carregados e executados juntamente com as páginas HTML. Foi utilizada a versão 1.1.x na implementação do protótipo, e este está disponível na seguinte URL: http://www.inf.ufrgs.br/~pomper/EDI/.

6.2 Aspectos de Implementação do EDI

O EDI possui vários níveis de funcionamento conforme pode ser visualizado na Figura 6.3. Basicamente ele pode ser dividido em um *applet* (que executa no cliente) e uma aplicação (que executa no servidor) que gerencia os usuários. Sendo o EDI um editor colaborativo para o desenvolvimento de sistemas na Internet foram utilizadas diversas técnicas de projeto (padrões de projeto) e bibliotecas de classes (*Java File System, HotDraw*) para a criação e validação do editor.

No EDI vários usuários acessam a ferramenta ao mesmo tempo e requerem visualizações diversas de um mesmo conjunto de dados. Para isso, a base de dados requer do sistema capacidade de gerenciar estas visões e a atualização dos dados. As diversas visões requeridas pelos usuários são obtidas através da implementação da arquitetura MVC [PAR94][BRA95a]. Como foi descrito no capítulo 4, esta arquitetura divide o sistema a ser implementado em três módulos: modelo (*model*), visões (*views*) e controladores (*controllers*).

Para a implementação das visões do modelo conceitual foi utilizado o *framework HotDraw* [JOH92]. Este *framework* possui um conjunto de classes que facilita a criação de editores diagramáticos. Além disso, para solucionar alguns problemas relativos à implementação gráfica de alguns componentes utilizados pelas notações que o EDI suporta foram utilizados os padrões de projeto Observer e Composite [GAM95].

Padrões de projeto são estruturas de colaboração de classes identificados freqüentemente no projeto de *frameworks* orientados a objetos [GAM95]. O padrão de projeto *Composite* compõe objetos dentro de estruturas de árvores para representar hierarquias parte-todo. Um exemplo são as figuras para representar classes (com os métodos e atributos) utilizadas no protótipo implementado. A Figura 6.1 apresenta o modelo de classes da aplicação do padrão de projeto Composite na implementação do EDI.

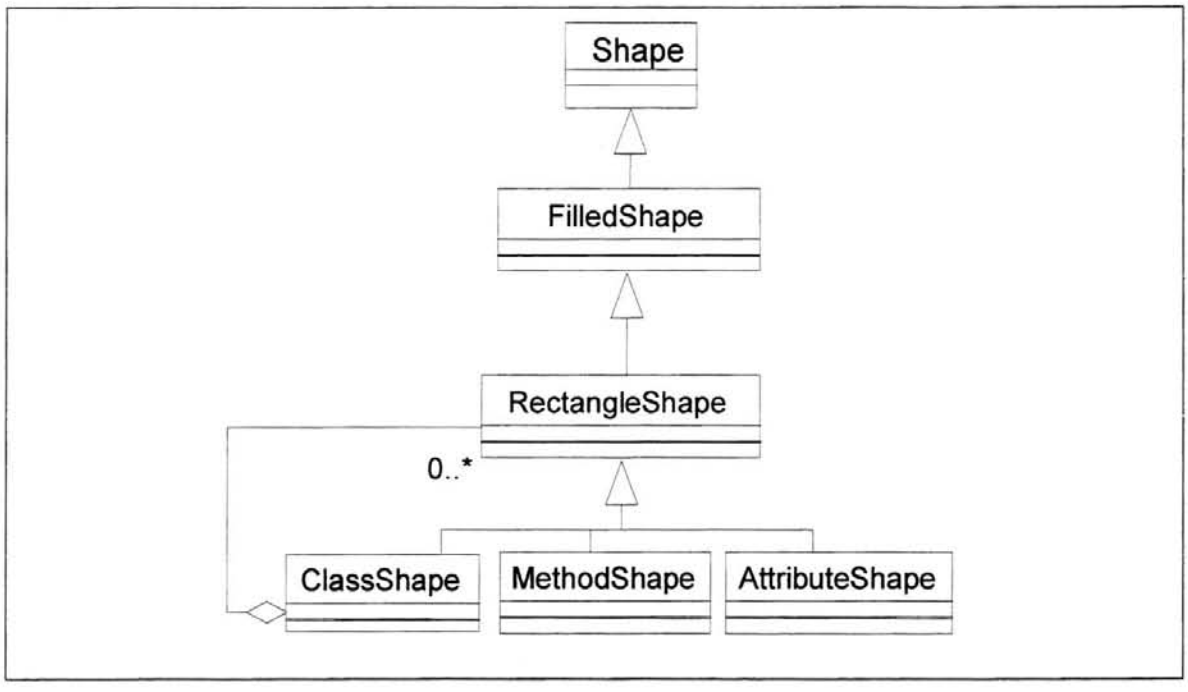

FIGURA 6.1 - Padrão de Projeto Composite [GAM95] Utilizado na Implementação do EDI

O padrão *Observer* define uma relação de dependência de um para muitos objetos, de maneira que quando um objeto muda seu estado, todos seus dependentes são notificados e atualizados automaticamente. Este padrão define, basicamente, o funcionamento da arquitetura MVC [PAR94][BRA95a]. No protótipo este padrão foi utilizado para permitir diversas visões de um mesmo modelo. Cada componente da notação utilizada pelo EDI possui sua visão (shape) e seu controlador (tool).

Foi utilizado, também, o *framework Java File System* (JFS) para a construção e gerenciamento do sistema de arquivos das visões manipuladas pelos clientes. A Figura 6.2 apresenta as principais classes utilizadas e implementadas para a manipulação de arquivos através do *framework* JFS.

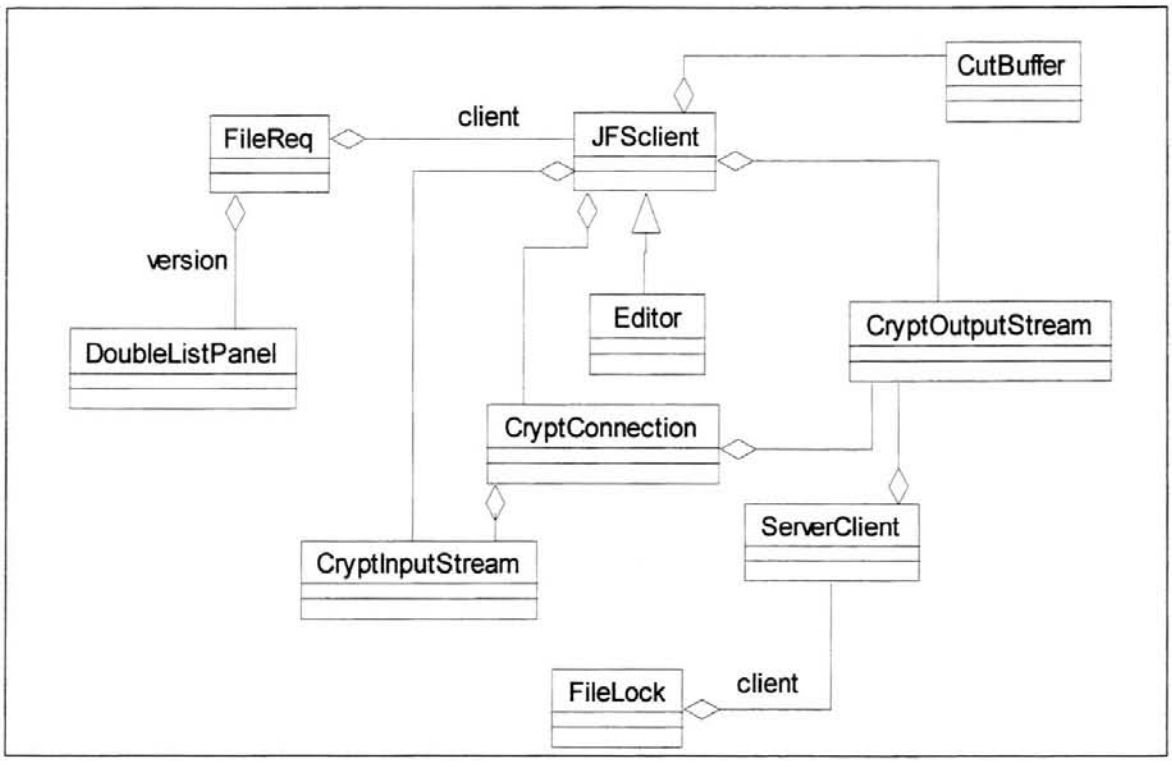

FIGURA 6.2 - Modelo de Classes do *Framework* JFS

Para que os usuários comuns possam manipular as visões a eles atribuídas foi implementado, utilizando-se da biblioteca de classes JFS, um sistema de arquivos. Este sistema permite que cada usuário do ambiente possua um espaço (diretório) onde são armazenados vários tipos de informações como, por exemplo, visões dos modelos, textos, figuras, etc.

Estas informações podem ser compartilhadas com os outros usuários. Quando o usuário (proprietário) inicia a modelagem de um sistema e deseja que outros usuários (colaboradores) o auxilie na execução da tarefa, ele simplesmente, enviará uma mensagem para os usuários desejados e compartilhará o seu diretório e o arquivo (visão) para que os colaboradores anotem o diagrama proposto.

Os usuários (desenvolvedores) podem trocar mensagens (informações e experiências) através de protocolos de comunicação implementados na linguagem Java como o correio eletrônico (e-mail) e *chat*. Estes programas agregados ao ambiente permitem a formação de *workgroup* entre usuários envolvidos na mesma tarefa ou especialistas em uma determinada área. Devido ao fato destas ferramentas estarem agregadas ao ambiente os usuários somente poderão trocar mensagens entre os usuários cadastrados no ambiente.

A Figura 6.3 apresenta uma visão geral da arquitetura do EDI. O ambiente proposto possui três camadas : *Database Server, Web Server* e *JFS Server*. Estas três camadas podem estar presentes em uma única máquina ou distribuídas em várias máquinas na Internet.

O usuário acessa o servidor Web (*Web Server)* na *Universal Resource Locator* (URL) do ambiente. Ao carregar a página HTML irá também carregar o *applet Java*, que consiste no EDI, podendo criar, alterar ou anotar modelos diagramáticos.

O EDI possui a seguinte arquitetura :

➔**DataBase Server :** Servidor de banco de dados onde estão armazenados as informações do sistema baseado no modelo semântico das classes;

➔**Web Server :** Servidor WWW onde estão armazenadas as páginas HTML e os *applets* Java (ferramentas) do ambiente.

➔**JFS Server :** Servidor de sistema de arquivos utilizado para armazenar informações temporárias dos usuários como, por exemplo, um diagrama ou um texto. É utilizado, também, para gerenciar a colaboração entre os usuários.

➔ **Users :** São os desenvolvedores (proprietários e colaboradores) do *software*.

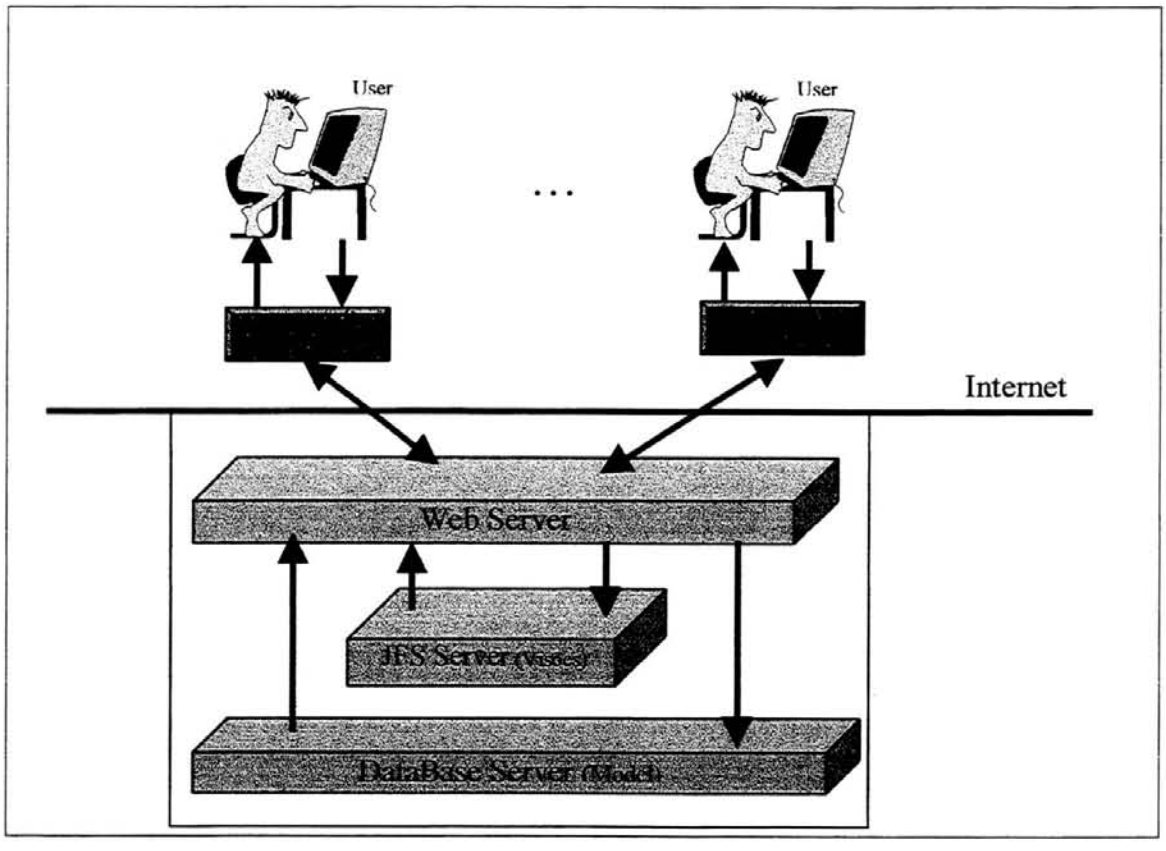

FIGURA 6.3 - Arquitetura do Editor Diagramático para Internet

Cada notação tem um EDI específico. Estão sendo construídos EDI específicos, utilizando as classes definidas em EDI anteriores para chegar a definição de um *framework* para a criação de editores diagramáticos.

6.3 Modelagem do Editor Diagramático

Nesta seção é descrita a modelagem do editor, apresentando o modelo de classes implementadas. Discute-se, também, a utilização de padrões de projeto (*design patterns*) [GAM95] no desenvolvimento do Editor.

O EDI foi implementado como um conjunto de pequenas ferramentas que agregadas formam um ambiente colaborativo para o desenvolvimento de sistemas. A Figura 6.4 apresenta as principais classes que compõem o EDI. Este modelo mostra a conexão que o EDI possui com o servidor (classe JFSServer) e os componentes principais do Editor (classes OMTMenuContainer, FrameEdit, ToolPalette e SimpleDrawingCanvas).

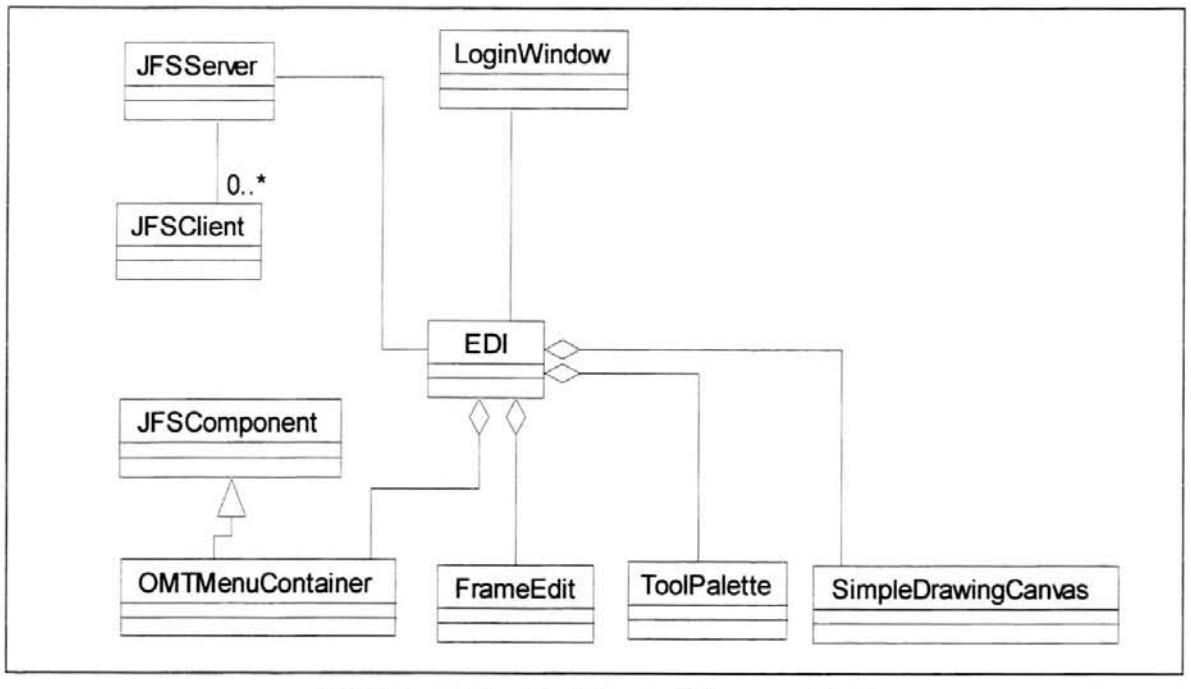

FIGURA 6.4 – Modelo de Objetos do EDI

A Figura 6.5 apresenta as classes implementadas para a utilização na modelagem dos sistemas. Estas classes representam o início da abstração para a criação de um *framework* para Editores Diagramáticos para diversas notações. Analisando as classes implementadas verifica-se que sua implementação ocorre de forma independente da base de dados (modelo conceitual) e dos componentes genéricos do EDI como, por exemplo, área de desenho, ferramentas de colaboração e ferramentas de edição.

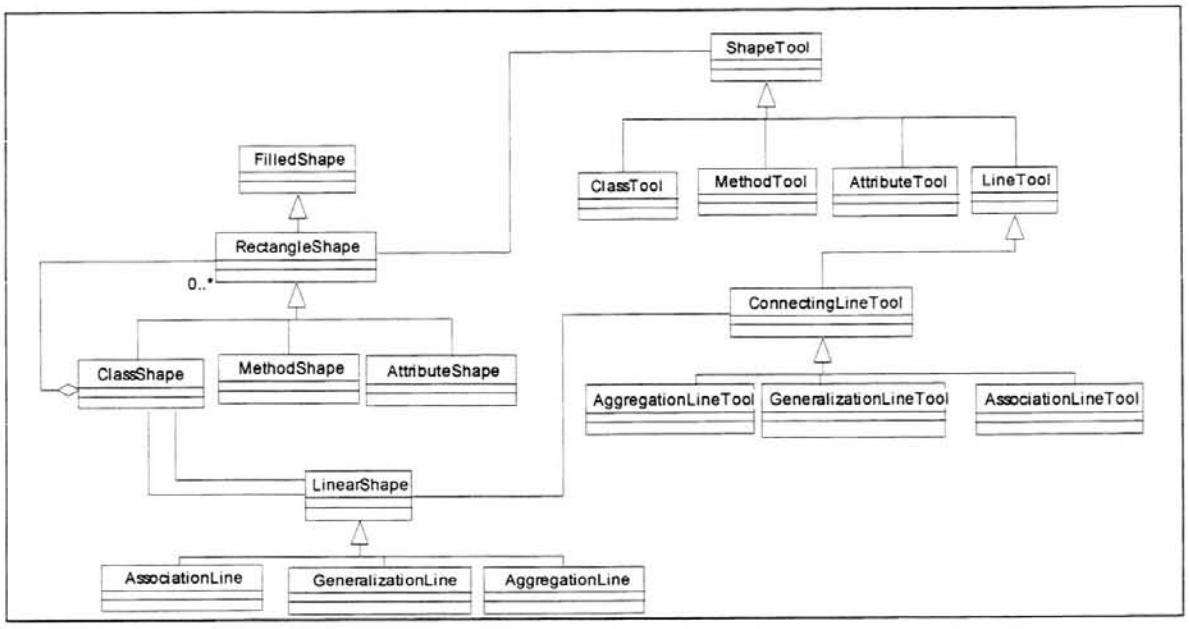

FIGURA 6.5 – Modelagem dos Componentes da Notação do EDI

6.4 Ferramentas do EDI

O protótipo foi implementado na linguagem Java utilizando os conceitos descritos nas seções anteriores como, por exemplo, arquitetura MVC e sistema de arquivos JFS. Para executar a ferramenta é necessário o acesso a URL do ambiente e acessar suas respectivas páginas HTML utilizando um navegador que suporte a linguagem Java versão 1.1.x, isto é, a execução dos *applets*.

Antes do usuário acessar as ferramentas do ambiente, este deve ser cadastrado no sistema pelo usuário *root* (Figura 6.6). Este cadastramento define o grupo de trabalho do usuário, sua área de armazenamento local, *username,* a cor do usuário entre outras características. Além do cadastro dos usuários do ambiente, este *applet* possibilita o cadastro de impressoras e o monitoramento das conexões dos usuários. Cada usuário cadastrado recebe uma senha que lhe permitirá o acesso às ferramentas do ambiente.

Cada usuário possuirá, desde a efetivação do cadastro, uma cor. Esta cor o identificará nas anotações que virá a fazer. Por exemplo, quando o usuário proprietário for realizar a aceitação das anotações dos usuários colaboradores, este poderá identificar através das cores "qual" usuário fez "que" anotação.

Uma vez cadastrado, o usuário poderá acessar o ambiente e utilizar suas ferramentas. O protótipo implementado apresenta um editor de notação UML, agregando recursos para o compartilhamento de informações e, consequentemente, o trabalho em equipe. A Figura 6.7 apresenta uma visão geral do editor.

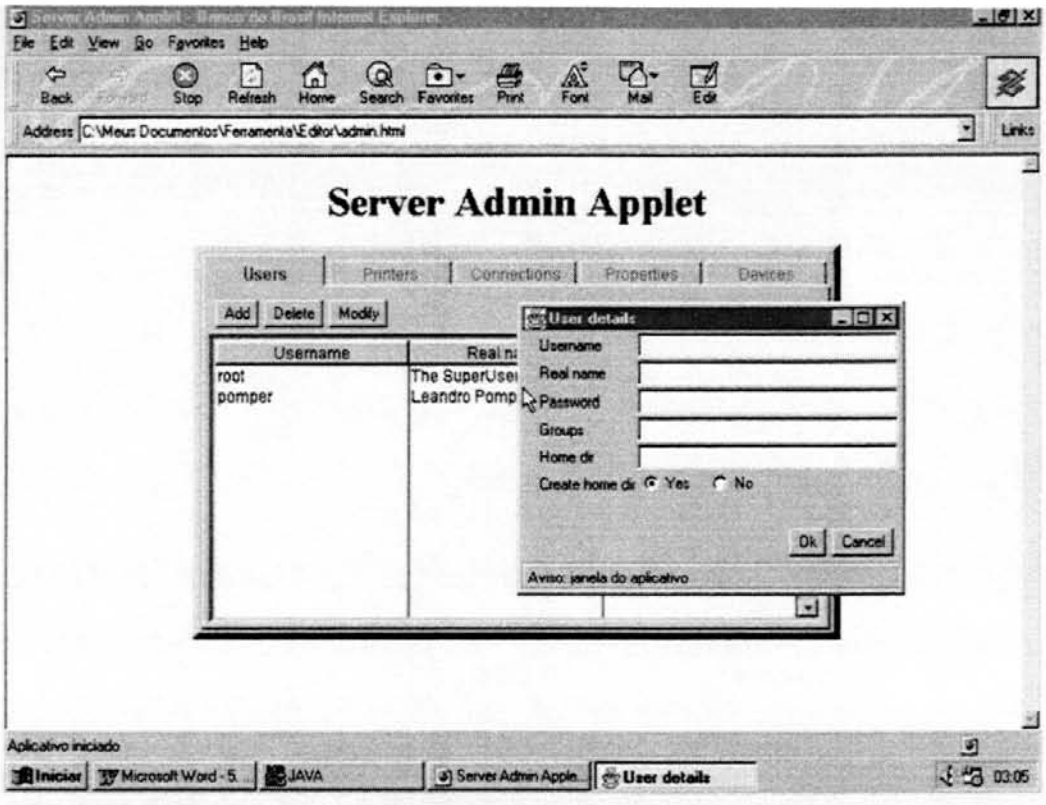

FIGURA 6.6 - *Applet* de Cadastro e Gerenciamento dos Usuários do Ambiente

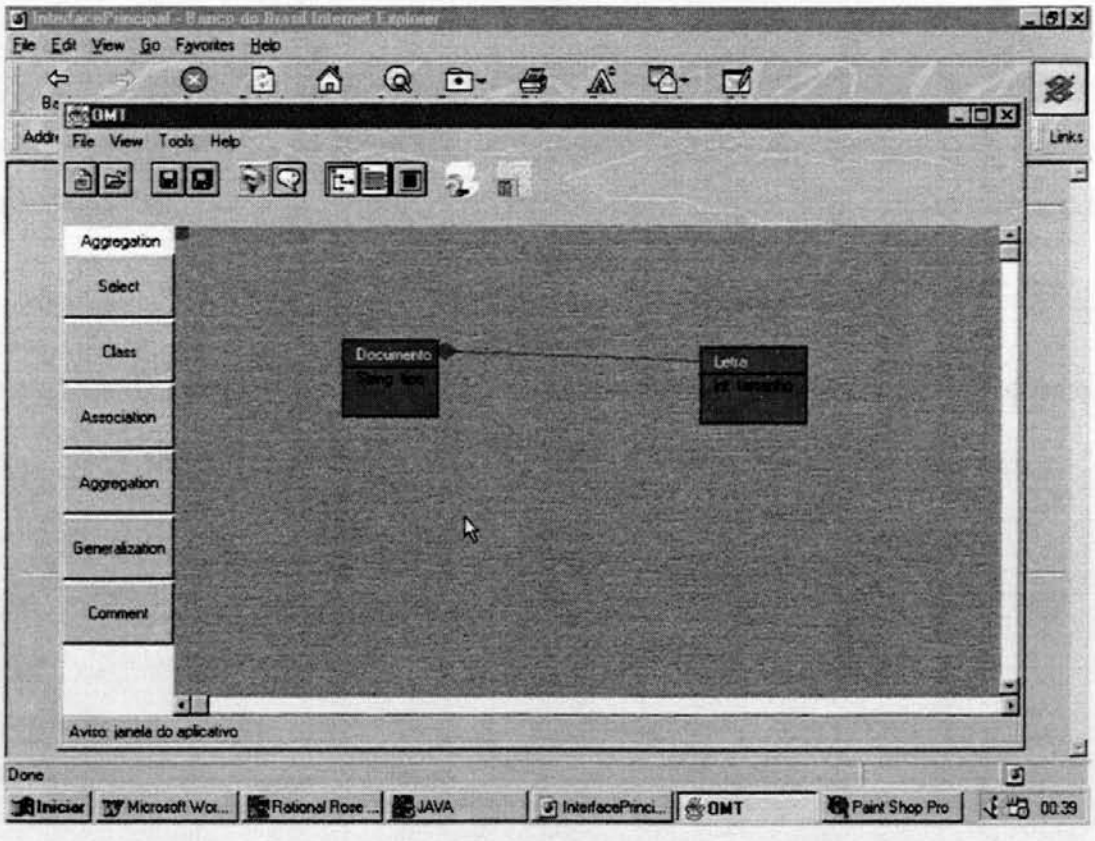

FIGURA 6.7 - Visão geral da Ferramenta de Modelagem

A interface gráfica do editor é dividida em menu principal, barra de ferramentas horizontal, barra de ferramentas vertical e área de desenho. Estas divisões são descritas a seguir.

6.4.1 Menu Principal

Possui as seguintes opções: *File* (manipulação de arquivos), *View* (visualização do modelo, quanto a notação, e visualização das tarefas em andamento e por realizar), *Tools* (gerador código Java), *Help* (ajuda ao usuário com relação aos recursos técnicos do editor);

6.4.2 Barra de Ferramentas Horizontal

As ferramentas dispostas nesta barra são :

 - Criar um novo documento;

 - Abrir uma visão (um diagrama);

 - Salvar um diagrama (visão);

 - Opção "Salvar Como";

 - Ferramenta para enviar mensagens eletrônicas;

 - Ferramenta para abrir canias de conversa entre os usuários (Talk);

 - Ferramenta para compartilhar diretórios e arquivos (visões)

 - Editor de textos

 - Opção que mostra os usuários e suas respectivas cores;

 - Help on-line;

 - Opção para sair do editor.

6.4.3 Barra de Ferramentas Vertical

Ferramentas para edição da notação. No protótipo é disponibilizado ao usuário ferramentas para acrescentar classes, relacionamentos entre as classes (herança, agregação e associação) e, também, acrescentar comentários ao modelo, tanto genéricos como associados a algum componente.

Para alterar alguma característica das classes como, por exemplo, nome, tipo (abstrata, final, publica ou privada), incluir atributos e métodos deve-se selecionar a ferramenta *Select* e posteriormente dar um duplo click sobre a classe desejada. Desta

forma, aparecerá uma janela (Figura 6.8) contendo as informações da respectiva classe, podendo as informações serem incluídas, alteradas ou removidas.

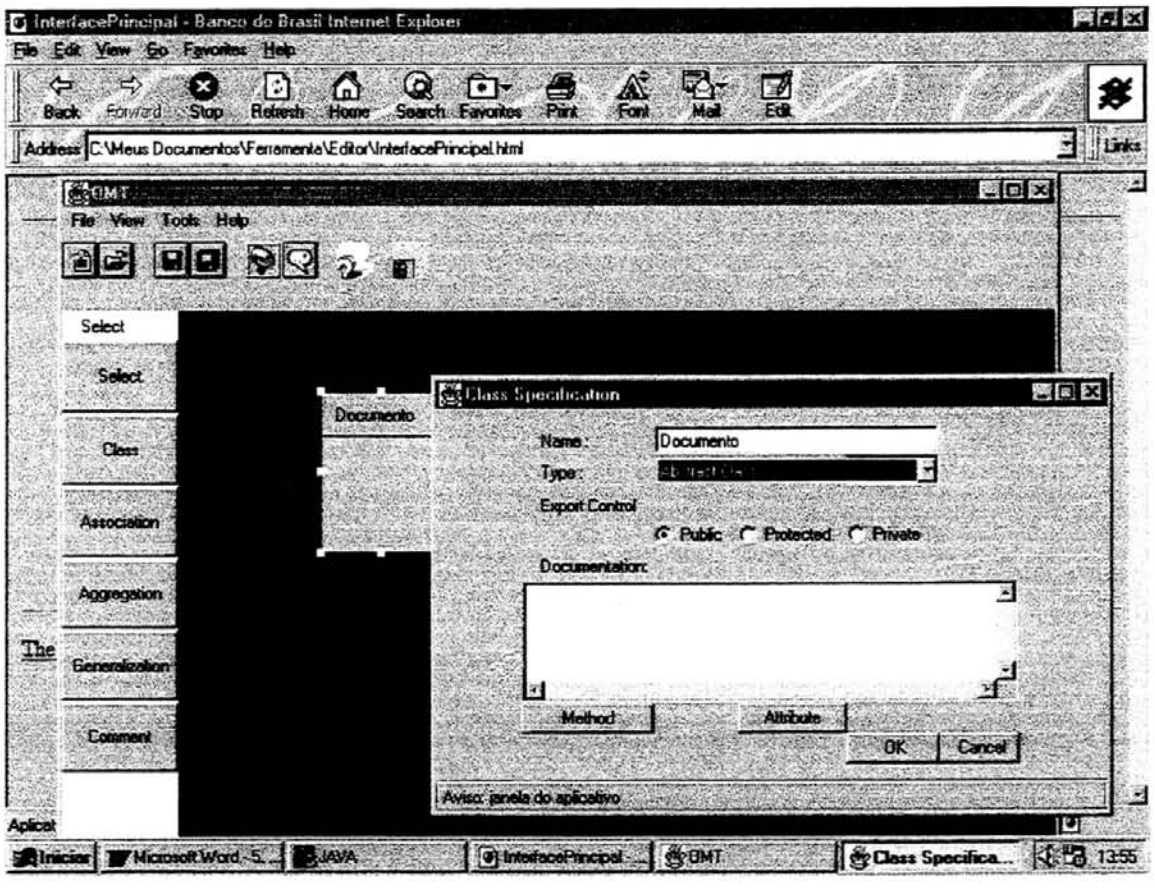

FIGURA 6.8 - Janela de Especificação da Classe

6.4.4 Atributos e Métodos

Para adicionar métodos ou atributos o usuário deverá acionar, na janela de especificação da classe, a opção para tal. Na primeira janela (Figura 6.9) será apresentada uma lista contendo todos os atributos ou métodos da classe.

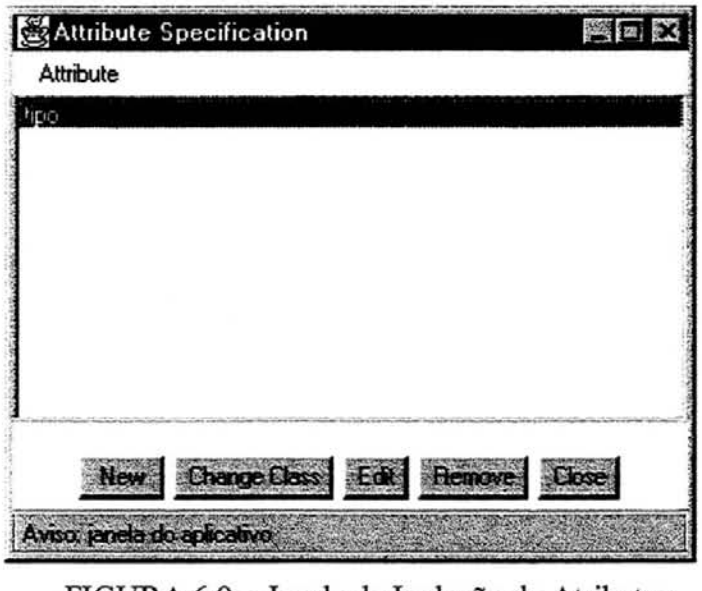

FIGURA 6.9 – Janela de Inclusão de Atributos

As opções nesta janela são :

- "New" : Cria um novo atributo ou método;

- "Change Class" : Permite ao usuário alterar a classe na qual o atributo ou método pertence;

- "Edit" : Permite alterar as características do atributo ou método;

- "Remove" : Remove o(s) atributo(s) ou método(s) selecionado(s);

- "Close" : Fecha a janela e retorna ao frame de especificação da classe.

Na criação ou edição de um atributo ou método é possível determinar suas características, conforme mostra a Figura 6.10, como nome e comentário.

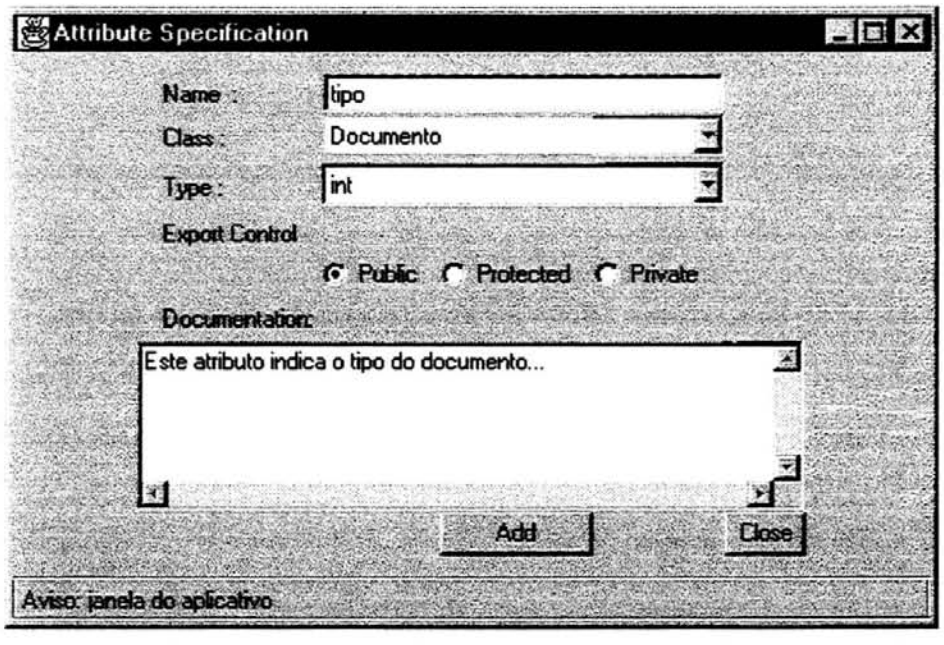

FIGURA 6.10 – Janela de Especificação de Atributos

6.4.5 Editor de Texto

O EDI permite que os usuários armazenem textos relacionados ou não com a modelagem dos sistemas. Para tanto, é disponibilizado um simples editor de texto, apresentado na Figura 6.11, que pode ser acionado através de uma opção da barra de ferramentas horizontal, conforme apresentado anteriormente.

Este editor apresenta as seguintes opções :

"Load" : permite ao usuário carregar, no editor, um texto já existente;

"Save" : permite ao usuário salvar o texto escrito;

"Insert" : permite ao usuário incluir um texto já existente ao texto em elaboração;

"New" : Cria um novo texto;

"Print" : permite ao usuário imprimir o texto ;

"Search" : procura por uma determinada palavra ou expressão;

"Replace" : substitui uma palavra ou expressão por outra palavra ou expressão;

Além destas opções o editor permite as funções de copiar, cortar e colar partes do texto.

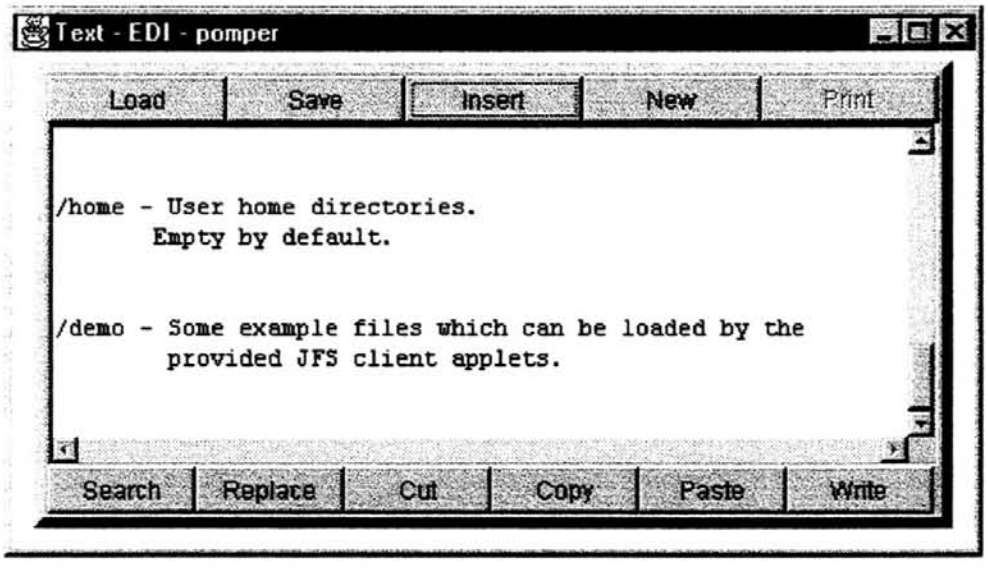

FIGURA 6.11 – Editor de Textos do EDI

6.4.6 Browser de arquivos

Para que haja colaboração, por compartilhamento de documentos (visões de diagramas), entre os usuários é necessário a utilização do *file browser* (Figura 6.12). Esta ferramenta permite ao usuário proprietário escolher os usuários ou grupo de usuário que poderão realizar anotações no devido documento.

Este compartilhamento é feito da seguinte forma: o usuário proprietário cria uma visão da tarefa que está sendo modelada. Para completar a tarefa com o máximo de qualidade, este usuário deverá compartilhar o diagrama com outros usuários capacitados a resolver e/ou complementar a tarefa (usuários colaboradores). Para isso o usuário proprietário armazenará a visão desenvolvida em um diretório comum ao grupo de trabalho e notificará os usuários colaboradores através do mecanismo de mensagem eletrônica (e-mail).

Quando um usuário (proprietário ou colaborador) estiver trabalhando sobre o documento, este ficará bloqueado para os outros usuários, a fim de garantir a consistência das alterações realizadas.

O usuário proprietário ficará responsável pelo término da tarefa. Este deverá ficar constantemente analisando as alterações e sugestões enviadas e adicionadas pelos usuários colaboradores. Para isso, cada usuário (colaborador e proprietário) possuirá uma cor que o distinguirá dos outros usuários. Por exemplo, o usuário proprietário estará relacionado com a cor preta, assim todos os componentes adicionados por este usuário serão da cor preta.

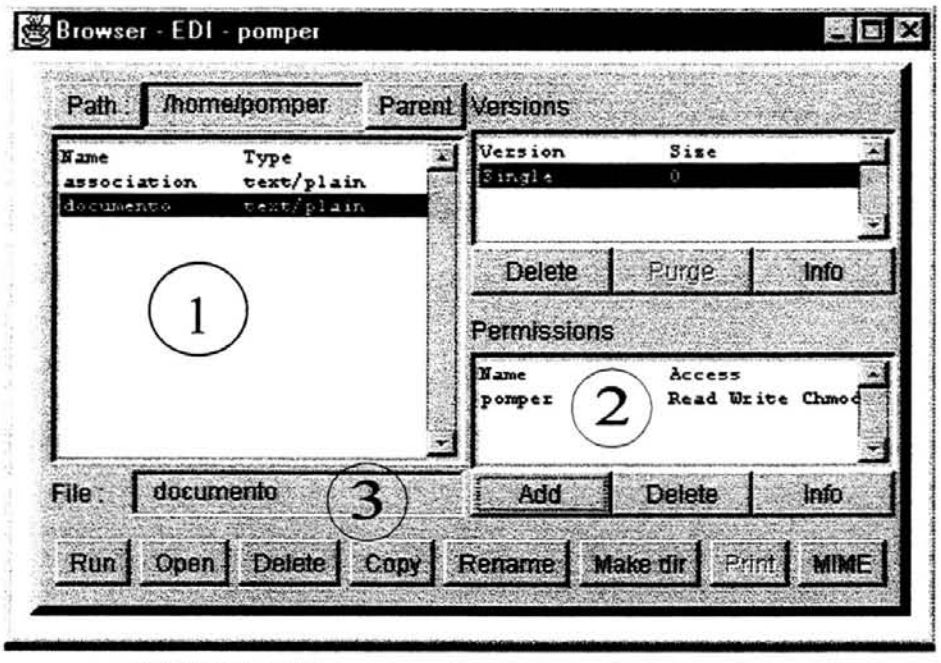

FIGURA 6.12 – Compartilhamento de Documentos

Na Figura 6.12 está indicada as principais partes do *File Browser*. Estas partes são descritas a seguir :

1 - Opção que apresenta o diretório corrente e todos os arquivos (documentos) armazenados.
Apresenta ainda o documento selecionado e opções para mudar de diretório (botão *parent*)

2 - Opção compartilhar o documento com outros usuários;
Opção "Add" : adiciona um usuário ou grupo de usuários na lista
Opção "Delete" : remove um usuário ou grupo de usuários da lista;
Opção "Info" : apresenta informações do documento.

3 - Opções para remover, copiar e renomear documentos e criar subdiretórios.

6.4.7 Visualizador de Cores

Esta opção permite ao usuário identificar, através das cores, quais os usuários realizaram anotações sobre o modelo proposto. Este visualizador apresenta uma tabela contendo em sua primeira coluna o *username* do usuário e na segunda coluna a sua respectiva cor. A Figura 6.13 apresenta este visualizador de cores.

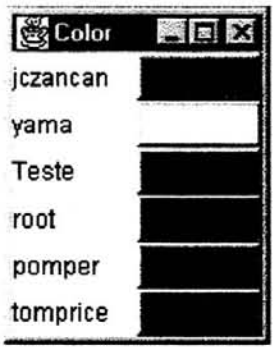

FIGURA 6.13 – Visualizador de Cores

6.4.8 e-Mail

Uma ferramenta agregada ao ambiente, utilizada para a troca de mensagens é o e-mail (*eletronic mail*). Esta ferramenta permite que os usuários cadastrados no EDI possa enviar mensagens eletrônicas a outros usuários cadastrados no ambiente. A Figura 6.14 apresenta a janela de composição da mensagem. Esta ferramenta é uma forma importante no processo de desenvolvimento de sistemas.

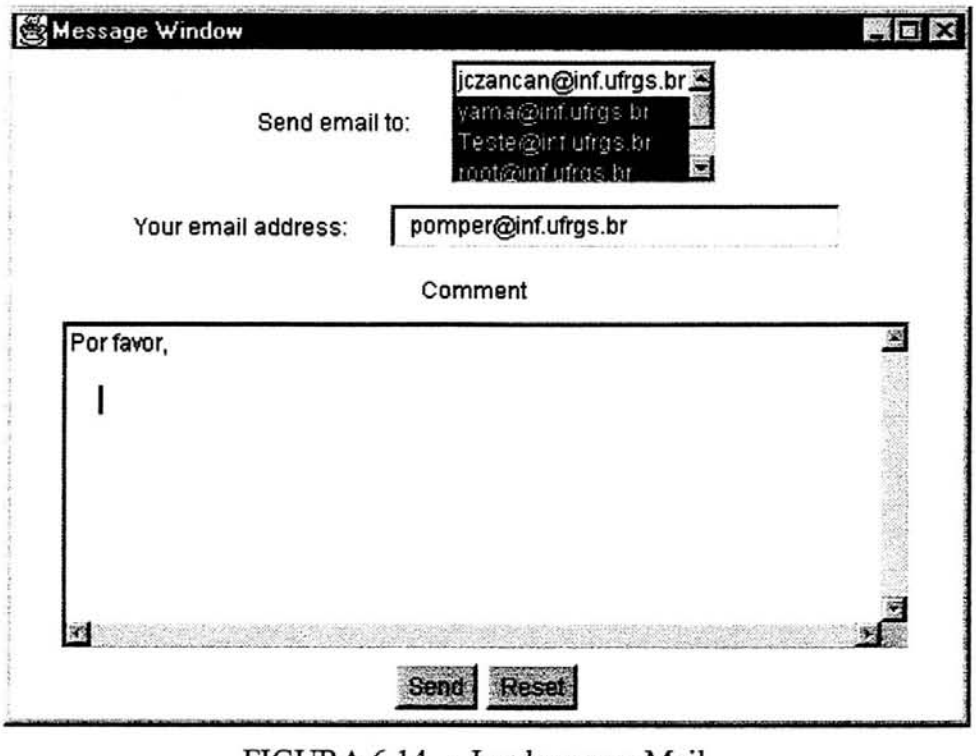

FIGURA 6.14 – Janela para e-Mail

6.4.9 Talk

Para esta opção foram criadas, basicamente, duas classes - *TalkServer* e *TalkClient*. A classe *TalkServer* implementa o gerenciamento da troca de mensagens entre os usuários que queiram fazer de forma *on-line*. Este gerenciamento torna-se essencial para a sincronização das mensagens (envio e recebimento). A classe *TalkClient* implementa a interface para o usuário, enviando as mensagens e configurações do mesmo.

Este método de troca de mensagens pode ser feito de duas formas: aberta ou fechada. Quando a comunicação é aberta, todos os usuários conectados receberão a mensagem enviada. Quando a comunicação é fechada, o usuário "seta" sua senha (*password*) e envia a mensagem. Esta mensagem só será recebida pelos usuários que tiverem "setado" a mesma senha. A figura 6.15 apresenta o *Frame* utilizado pelo usuário para manter uma comunicação *on-line*.

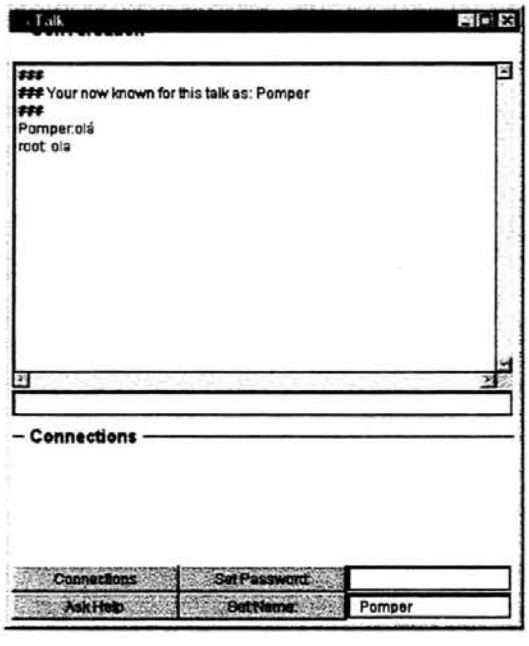

FIGURA 6.15 - Janela da troca de mensagens através de *Talk*

6.5 Exemplo de Utilização do EDI

Nesta seção são apresentados alguns exemplos de utilização do EDI. demonstrando seu funcionamento e características. Na Figura 6.16 é apresentado o EDI com a modelagem de classes e relacionamentos (associação e agregação). As Figuras 6.17 e 6.18 apresentam, respectivamente, anotação de inclusão e a forma de identificar o usuário colaborador através das cores.

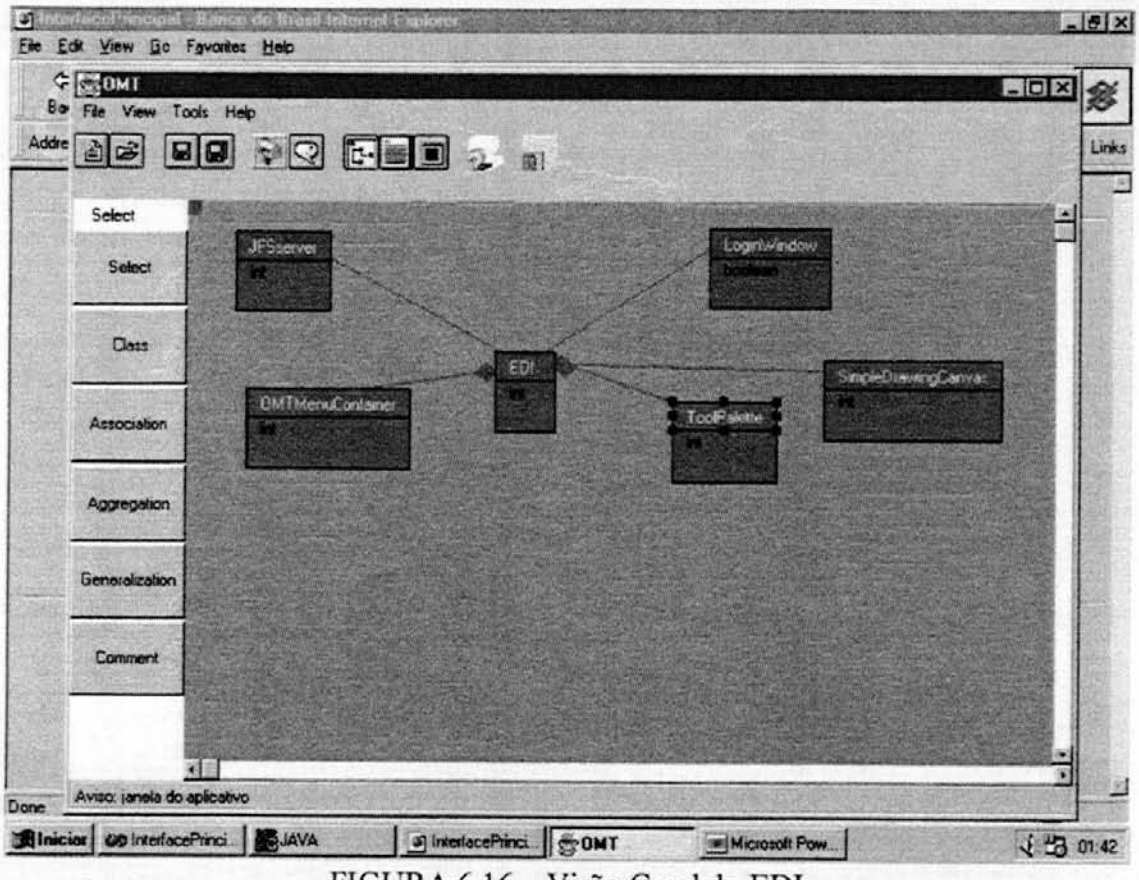

FIGURA 6.16 – Visão Geral do EDI

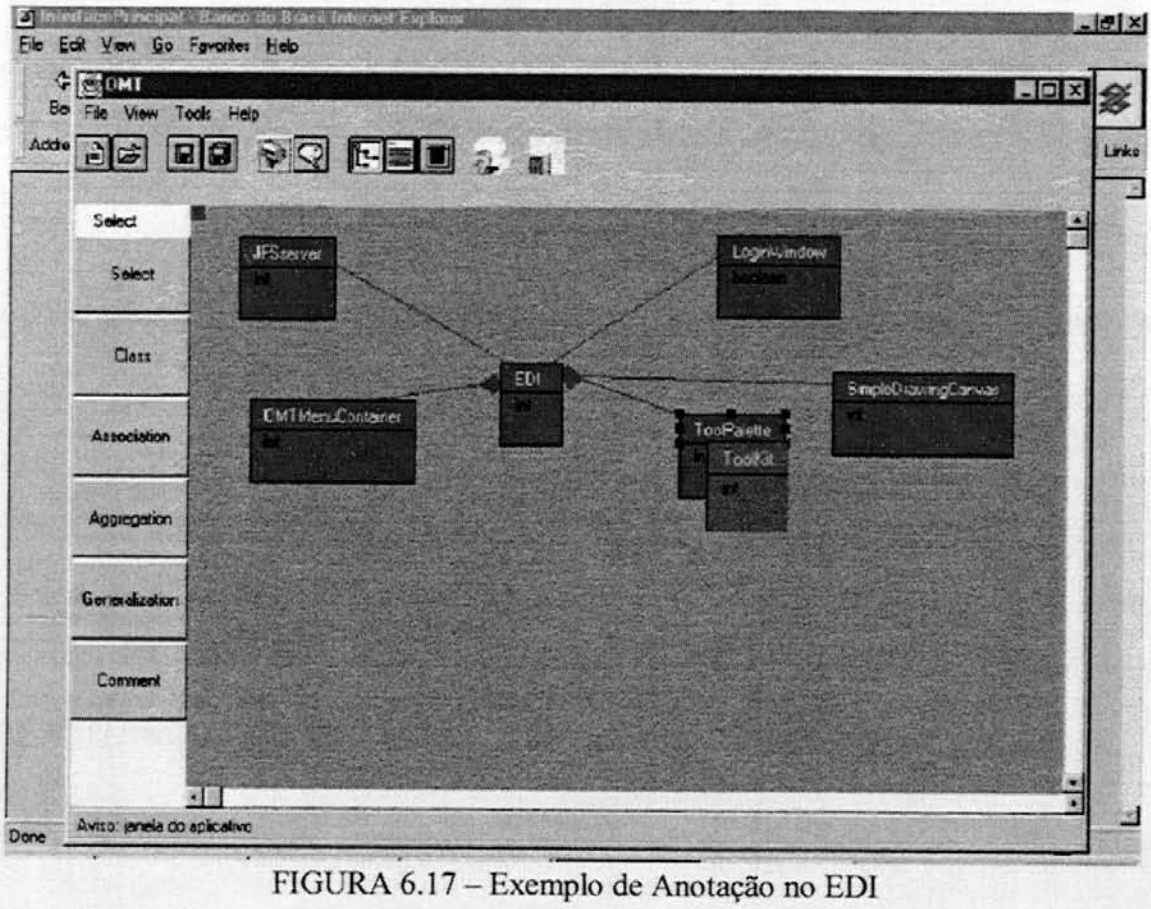

FIGURA 6.17 – Exemplo de Anotação no EDI

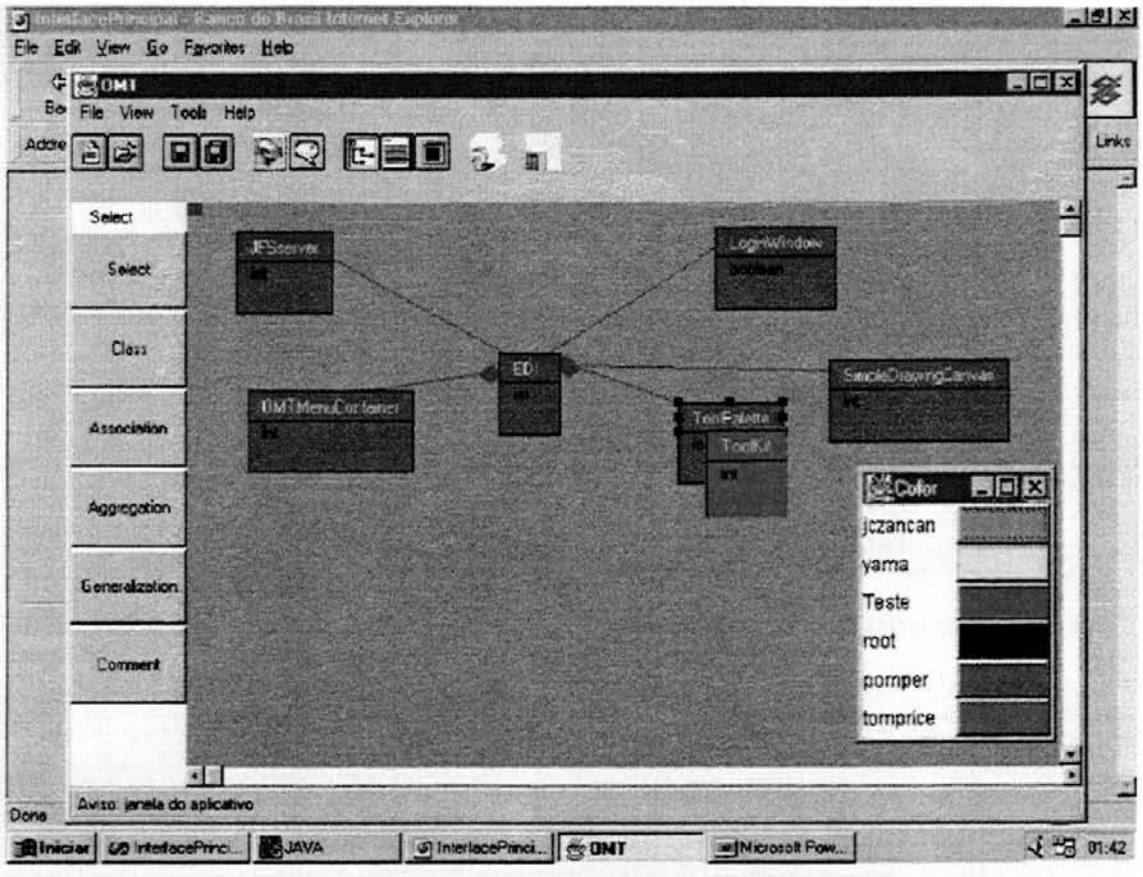

FIGURA 6.18 – Identificação do Usuário Colaborador

7 Conclusão

O desenvolvimento colaborativo de sistemas traz diversos benefícios, tanto para as empresas que empregam este tipo de desenvolvimento quanto para os desenvolvedores. As empresas ganham em produtividade e qualidade dos produtos desenvolvidos. Enquanto que os desenvolvedores ganham com relação ao conhecimento e troca de experiências que ocorre neste tipo de desenvolvimento.

A utilização da Internet como base operacional para este tipo de ambiente possibilita uma maior flexibilidade para os desenvolvedores. Estes desenvolvedores podem, através da Internet, acessar as informações relacionadas ao desenvolvimento sem a preocupação da localização da base de dados (dicionário de dados), da portabilidade de plataforma (arquitetura em camadas) e das ferramentas de *groupware* disponíveis (e-mail, chat).

A partir disso, e também verificando que automatizando o processo de desenvolvimento, através de ferramentas que suportam notações diagramáticas, auxiliam os desenvolvedores através da expressividade gráfica que facilita a comunicação entre os membros das equipes de desenvolvimento, a manutenção, a depuração e a documentação dos sistemas [MEL89], foi implementado um protótipo que valida as considerações apresentadas ao longo deste trabalho. Diversas características de colaboração que um ADS na Internet deve possuir foram experimentadas, como foi descrito no Capítulo 3.

Os resultados obtidos desta pesquisa, demostram a viabilidade de se combinar um ambiente aberto (Internet) com o desenvolvimento colaborativo de *software*. A colaboração na especificação de sistemas de informação é obtida através do compartilhamento das visões e das anotações adicionadas as visões pelos colaboradores.

A abordagem de diferentes visões de um mesmo modelo possibilita o uso integrado de diferentes notações para a especificação de sistemas de informação por diferentes usuários. Esta abordagem de múltiplas visões foi conseguida através da implementação da arquitetura MVC, permitindo o ambiente a trabalhar com um mesmo modelo semântico. O Modelo é implementado como um dicionário de dados existente em um servidor, com serviços de acesso e atualização.

Com a utilização desta arquitetura, o Editor pode ser facilmente estendido. No Capítulo 5 foi apresentado a relação entre modelo, visão e controlador dos componentes da notação utilizada pelo EDI. Para a extensão e/ou criação de novos editores diagramáticos será necessário, somente, a subclassificação das classes de modelo, visões e controladores e implementação do comportamento específico de cada componente da nova notação.

Outra importante contribuição deste trabalho é a utilização de um modelo de anotações durante o desenvolvimento colaborativo de SII. Conforme apresentado no Capítulo 5, estas anotações auxiliam os desenvolvedores na troca de idéias e experiências durante a construção de documentos. A grande diferença do modelo proposto por [SOU98] é a anotação de substituição de componentes. Este tipo de

anotação proporciona aos colaboradores uma maior flexibilidade nas contribuições realizadas no diagrama que está sendo construído.

Como o protótipo foi implementado em *Java*, como *applet*, os documentos do desenvolvimento não podem ser armazenados localmente por questões de segurança impostas pelos navegadores e pela própria linguagem Java. Atualmente existe um sistema de arquivos (implementado utilizando o *framework* JFS) que executa no servidor e permite aos usuários o armazenamento de informações de forma não-estruturada (como textos, visões de diagramas e imagens).

Com o surgimento de mecanismos de certificação de segurança, por parte dos navegadores e por parte da linguagem Java, haverá a possibilidade dos usuários armazenarem informações na máquina cliente (localmente). Com isso, o ambiente ganhará em termos de desempenho e flexibilidade, pois a execução do ambiente será totalmente realizada na máquina cliente.

7.1 Trabalhos Futuros

Durante a execução deste trabalho e implementação do protótipo, foram identificados alguns aspectos que deverão ser melhor analisados ou estendidos em trabalhos futuros. São eles:

⇨Implementação de Mudanças de Domínio de Modelagem

Mudanças de domínio permitem ao usuário colaborador uma flexibilidade maior nas anotações realizadas sobre os documentos. Estas mudanças permitem que um determinado componente do modelo que está sendo criado possa ser transformado (como anotação) em outro componente existente no contexto da notação utilizada pelo Editor.

⇨Controle de Versões

Observa-se que mecanismos de controle de versões são fundamentais em atividades de natureza evolutiva, nas quais é importante o registro, não só de um, mas de vários estágios de desenvolvimento dos objetos do projeto como, por exemplo, o processo de desenvolvimento de sistemas de informação.

O controle de versões permitirá ao ambiente que as contribuições realizadas pelos usuários colaboradores mantenham a consistência com relação ao modelo proposto pelo usuário proprietário. Este controle pode ser obtido através de mecanismos embutidos em cada operação realizada sobre uma determinada versão do documento.

⇨Implementação do Dicionário de Dados

Segundo [FOR89] as funções que um dicionário deve possuir são: prover integridade de dados, compartilhamento de informações (controlar, bloquear, desbloquear e gerenciar o acesso dos usuários), prover a integração entre os dados e ferramentas do ambiente e padronizar os documentos.

⇨Extensibilidade

O trabalho descrito faz parte de um projeto que envolve a especificação de um *framework* para a construção de editores diagramáticos. Esta especificação deverá ser alcançada através do processo evolutivo que está sendo empregado na construção de EDI's, analisando cada risco envolvendo as características de editores diagramáticos colaborativos para Internet.

⇨Validação

O protótipo deve ser validado perante um grupo maior de usuários. Esta validação servirá para avaliar aspectos funcionais, de segurança, de interface e de mecanismos de colaboração.

Em suma, as principais contribuições deste trabalho foram: experimentar a colaboração assíncrona na Internet; trabalhar com conceitos de anotações em documentos de desenvolvimento; e construir de forma genérica um editor de diagramas, facilitando sua extensibilidade.

Bibliografia

[BAL 97] BALDWIN, Richard G. **Event Handling in JDK 1.0 and JDK 1.1 :** A First Look at the Delegation Event Model. Disponível por WWW em http://sunsite.utk.edu/java/javazine/Issue/EventHandling3.html (20/12/97).

[BRA 95a] BRANT, John Michael. **HotDraw.** Urbana-Champaign: University of Illinois, 1995. 48 p. (Dissertação de Mestrado). Disponível por WWW em http://st-www.cs.uiuc.edu/ftp/pub/papers/HotDraw.ps.gz

[BRA 95b] BRANT, John Michel; JOHNSON, Ralph E. **Creating Tools in HotDraw by Composition.** Disponível por WWW em http://st-www.cs.uiuc.edu/ftp/pub/papers/Hotdraw/HotDrawTools.ps.Z

[BER 87] BERNSTEIN, Philip A. et al. **Concurrency Control and Recovery Database Systems.** CA : Addison-Wesley Publishing Company, 1987. 370 p.

[BIG 88] BIGELOW, B. Hypertext and CASE. **IEEE Software**, Los Angeles, CA, v.5, n.3, p.23-27, Mar. 1988.

[BOO 91] BOOCH, G. **Object Oriented Design**: with Applications. Redwood City: The Benjamin/Cumming, 1991.

[BOR 95] BORGES, Marcos Roberto da Silva; CAVALCANTI, Maria Cláudia Reis; CAMPOS, Maria Luiza Machado. Suporte por Computador ao Trabalho Cooperativo. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 15.; JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, 14.; 1995. **Anais...** [S.l.]: SBC, 1995.

[BRO 92] BROWN, A.; FEILER, P.; WALLNAU, K. **Understanding Integration in a Software Development Environment.** Pittsburgh, Pennsylvania: Software Engineering Institue, 1992. 31 p. (Technical Report CMU/SEI-91-TR-31).

[BWY 97] BWYDAEGH, J. **OMT Editor.** Disponível por WWW em http://info6.vub.ac.be:8080/~bwydaegh/java/Editor/objEditor/ (23/05/1997).

[CAM 97] CAMERON, Jamie. **Java File System.** Disponível por WWW em http://www.uni-koeln.de/themen/java/examples/contest/PR8ADPL7/applet.html (18/06/1997).

[CAT 89] CATLIN, Timothy; BUSH, Paulette; YANKELOVICH, Nicole. InterNote: Extending a Hypermedia Framework to Support Annotative Collaboration. In: ACM CONFERENCE ON HYPERTEXT, 2., 1989, Pittsburgh, Pennsylvania. **Proceedings...** New York: ACM, 1989. p. 365-378.

[CHE 76] CHEN, P. The Entity Relationship Model- Toward a Unified View of Data. **ACM Transactions on Data Base Systems,** New York, v. 1, n. 1, p. 6-36, Mar. 1976.

[COA 90] COAD, P.; YOURDON, E. **OOA: Object-Oriented Analysis.** Englewood Cliffs: Prentice Hall, 1990.

[COR 96] CORNELL, Gary; HORSTMANN, Cay S. **Core Java.** California: SunSoft Press, 1996. 621p.

[CYB 92] CYBULSKI L.; REED, Karl. A Hypertext Based Software-Engineering Environment. **IEEE Software,** New York, v.9, n. 2, p. 62-68, Mar. 1992.

[DAR 87] DART, S. et al. Software Development Environment . **Computer**, Los Angeles, CA, v.20, n.11, p. 18-28, Nov. 1987.

[DAL 97] DALMOLIN, Luiz Cláudio. **Diagrama Global de Colaborações:** uma Proposta para a Melhoria do Projeto de Software Orientado a Objeto. Porto Alegre: II PUCRS, 1997. 96 p. Dissertação de Mestrado.

[DED 97] DEDIEU, Oliver. **A Proposition for a Delegation Model in Java.** Disponível por WWW em http://falconet.inria.fr/~dedieu/delegation.html (19/12/97).

[DIL 93] DILLON, Tharam; TAN, Poh Lee. **Object-Oriented Conceptual Modeling.** New York: Prentice Hall, 1993. 315p.

[ELL 91] ELLIS, C. A.; GIBBS, S. J.; REIN, G. L. Groupware: Some Issues and Experiencies. **Communications of the ACM**, New York, v. 34, n. 1, p. 38-58, Jan. 1991.

[FLA 97] FLANAGAN, David. **Java in a Nutshell**: A Desktop Quick Reference. Cambridge: O'Reilly, 1997. 610 p.

[FIN 94] FINKELSTEIN, Anthony et al. **Software Process Modeling and Technology**. New York: John Willey & Sons Inc, 1994. 362 p.

[FOW 97] FOWLER, Martin. **UML Distilled.** Massachusetts: Addison Wesley Longman Inc., 1997. 179 p.

[FOR 89] FORTE, G. In Search of the Integrated Environment. **CASE *Outlook***, v. 3, n.12, p.5-27, Dec.1989.

[GAM 95] GAMMA, Erich et al. **Design Patterns:** Elements of Reusable Object-Oriented Software. Massachusetts: Addison-Wesley, 1995. 395 p.

[GAR 93] GARZOTTO, F. PAOLINI, P.; SCHWABE, D. HDM – A Model-based Approach to Hypertext Application Design. **ACM Transactions on Information Systems**, New York, v.11, n. 1, Jan. 1993.

[GEO 98] GEORGAKOPOULOS, D.; HORNICK, M.; SHETH, A. **An Overview of Workflow Management:** From Process Modeling to Workflow Automation Infrastructure. Disponível por FTP em ftp.gte.com/pub/dom/reports/georg95a.ps . (1998).

[GIM 94] GIMENES, I. **O processo de Engenharia de Software:** Ambientes e Formalismos. Caxambu, MG, Brasil: Sociedade Brasileira de Computação, 1994. 42 p.

[GRA 96] GRALA, A. **Modelagem de Documentos Auxiliada por Técnicas de Modelagem de Workflow**: Trabalho Individual. Porto Alegre: CPGCC-UFRGS, 1996. (TI - 564).

[GRU 91] GRUDIN, Jonathan. CSCW Introduction. **Communications of the ACM,** New York, v.34, n.12, p.30-34, Dec. 1991.

[GRU 95] GRUDIN, Jonathan. Groupware and Social Dynamics: Eight Challenges for the Developers. **Communication of the ACM,** New York, v.37, n.5, p.92-105, May 1995.

[HAM 96] HAMANN, Jerrid. **Analysis of Java Database Connectivity and its Application in a Multi-Platform, Multi-DBMS Environment**. Texas: Texas A&M University, 1996.

[HIL 97] HILLS, Mellanie. **Intranet as Groupware.** New York: John Wiley & Sons, Inc., 1997. 308 p.

[JAC 92] JACOBSON, I. et al. **Object-Oriented Software Engineering**: a Use Case Driven Approach. Addison-Wesley: Wokingham, 1992.

[JOH 91] JOHNSON, R. Documenting Frameworks Using Patterns. **ACM SIGPLAN Notices**, New York, v.26, n.11, p.63-76, Oct.1991. Trabalho apresentado na Conference on Object-Oriented Programming, Systems, and Applications, OOPSLA, 1991, Phoenix, US.

[JWS 97] JAVA WORKSHOP. Disponível por WWW em http://www.sun.com/jws/ (1997).

[KHO 95] KHOSHAFIAN, Setrag; BUCKIEWICZ, Marek. **Introduction to Groupware, Workflow and Workgroup Computing.** New York: John Wiley & Sons, INC, 1995.

[KNO 97] KNOWLEDGE SYSTEMS CORPORATION. **Java Version of HotDraw.** Disponível por WWW em http://www.ksccary.com/ksc/HotDraw/HotDraw (20/06/97).

[KNU 94] KNUDSEN, J. Lindskov et al. **Object-Oriented Enviroments**: The MJ0LNER Approach. New York: Prentice-Hall, 1994. 627 p.

[MAR 82] MARTIN, J. **Computer Data Base Organization**. New York: Prentice-Hall, 1982.

[MAR 96] MARINHO, Alexandre. Symantec Café X Java WorkShop. **PC WORLD**, Rio de Janeiro, n.52, p. 74-75, out. 1996.

[MEL 89] MELO, Walcélio Louzada Martins. **Proposta de um Editor Diagramático Generalizado.** Porto Alegre: CPGCC da UFRGS, 1989. 247 p. Dissertação de Mestrado.

[MEY 94] MEYER, Bertrand. **An Object-Oriented Enviroment**: Principles and Application. New York: Prentice-Hall, 1994. 231 p.

[MIC 97] MICROSOFT CORPORATION. **Microsoft J++** . Help disponível no Software (1997).

[NEW 96] NEWMAN, Alexander; et al. **Using Java**: Special Edition. Indianopolis: QUE, 1996. 869 p.

[NAG 96] NAGL, Manfred (Ed.). **Building Tightly Integrated Software Development Enviroment:** The IPSEN Approach. Berlin: Springer-Verlag, 1996. 709 p. (Lecture Notes in Computer Science, v.1170).

[NAU 98] NAUGHTON, Patrick; SCHILDT, Herbert. **Java 1.1:** The Complete Reference. Osborne McGraw-Hill: Berkeley, 1998. 1028 p.

[ORT 95] ORTIGOSA, Alvaro Manuel. **Proposta de um Ambiente Adaptável de Apoio ao Processo de Desenvolvimento de Software.** Porto Alegre: CPGCC da UFRGS, 1995. 192 p. Dissertação de Mestrado.

[PAR 94] PARCPLACE System Inc. **Visual Works Reference Manual.** CA: ParcPlace System Inc. , 1994. 434 p.

[PAT 96] PATEL, Pratik; MOSS, Karl. **Java Database Programming with JDBC.** Arizona: Coriolis Group Books, 1996. 480 p.

[PEN 93] PENEDO, M. Process-based Software Engineering Environments (PSEE). In: BRAZILIAN SYMPOSIUM OF SOFTWARE ENGINEERING, 7., 1993, Rio de Janeiro. **Anais...**Rio de Janeiro: PUC-RJ, 1993. Palestra convidada.

[PER 91] PERRY, D.; KAISER, G. Models of Software Development Environments. **Transactions on Software Engineering,** New York, v.17, n.3, p.283-295, Mar. 1991.

[PER 92] PERIN, Marcelo Gattermann. **Um Sistema de Gerenciamento de Hiperdocumentos para Ambientes de Desenvolvimento de Software.** Porto Alegre: CPGCC da UFRGS, 1992, 173 p. Dissertação de Mestrado.

[PRE 95] PRESSMAN, Roger S. **Engenharia de Software.** São Paulo: Makron Books, 1995. 1056 p.

[POM 98a] POMPERMAIER, Leandro Bento; PRICE, Roberto Tom. Considerações sobre o Desenvolvimento de Sistemas de Informação na Internet. In: WORKSHOP IBEROAMERICANO DE ENGENHARIA DE REQUISITOS E AMBIENTES DE SOFTWARE – IDEAS, 1., 1998, Torres – RS. **Anais...** Torres: CPGCC da UFRGS, 1998.

[POM 98b] POMPERMAIER, Leandro Bento Pompermaier; PRICE, Roberto Tom. Um Editor de Diagramas para a Internet. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE – SBES. 12., 1998, Maringá – PR. **Anais...** Maringá: DIN/UEM, 1998. p. 265-277.

[RAT 98] RATIONAL. **Rational Extends Reach of Visual Modeling With Rational Rose 98.** Disponível por WWW em http://www.rational.com/press/releases/data/rose98 (10/02/98).

[REI 95] REIS, Rodrigo Quites. **Desenvolvimento Cooperativo de *Software*:** Ferramentas e Metodologias: Trabalho Individual. Porto Alegre: CPGCC-UFRGS. 1995. 96 p. (TI- 497/95)

[ROS 95] ROSSI, G. et al. An Object-Oriented Model for Designing the Human-Computer Interface of Hypermedia Aplications. In: INTERNATIONAL WORKSHOP ON HIPERMEDIA DESIGN, 1995. **Proceedings...** [S.l.: s.n.] , 1995.

[RUM 91] RUMBAUGH, J. et al. **Object-Oriented Modeling and Design.** Englewood Cliffs: Prentice Hall, 1991. 652 p.

[SCH 95] SCHWABE, D.; ROSSI, G. The Object-Oriented Hypermedia Design Model. **Communications of the ACM**, New York, v. 38, n. 8, p. 45-46 , Aug. 1995.

[SER93] SERBANATI, Luca Dan. **Integrating Tools for Software Development.** Englewood Cliffs : Prentice Hall, 1993. 180 p.

[SHA 96a] SHAFFER, Gail. Conectividade Entre Servidores Web e Banco de Dados. **PC Magazine Brasil**, São Paulo, n.10, out. 1996.

[SHA 96b] SHAH, Rawn. **Integranting Databases with Java via JDBC**. Disponível por HTTP em http://www.javaworld.com/javaworld/jw-06-1996/jw-06-jdbc.html (29/06/1996).

[SIG 95] SIGNORE, Robert; CREAMER, John; STEGMAN, Michael O.**The ODBC Solution**: Open Database Connectivity in Distributed Environments.New York : McGraw-Hill, Inc., 1995.

[SOU 96] SOUZA, Adriana Silveira de. **Um Estudo sobre Trabalho Cooperativo suportado por Computador (CSCW)**: Trabalho Individual. Porto Alegre: CPGCC-UFRGS, 1996. (TI - 540).

[SOU 98] SOUZA, Cleidson Ronald Botelho. **Um Framework para Editores de Diagramas Cooperativos Baseados em Anotações.** Campinas: Instituto de Computação- Universidade Estadual de Campinas, 1998. 107 p. Dissertação de Mestrado.

[SUM 97] SUN MICROSYSTEMS. **Java AWT:** Delegation Event Model. Disponível por WWW em http://java.sun.com/jdk1.1.1/docs/guide/awt/designspec/events.html (20/12/97).

[SUN 96] SUNDSTED, Todd. **Introduction to the AWT**: a description of Java's user interface toolkit. Disponível por WWW em http://www.javaworld.com/javaworld/jw-07-1996/jw-07-awt.html (15/08/96).

[SUN 96a] SUNDSTED, Todd. **Observer and Observable :** an Introduction to the Observer interface and Observer class using the Model/View/Controller architecture as a guide. Disponível por WWW em http://www.javaworld.com/javaworld/jw-10-1996/jw-10-howto.html (30/09/96).

[TRE 94] TREVISAN, Alexandre Rubin. ***Framework* para Integração de Ferramentas**: Trabalho Individual. Porto Alegre: CPGCC da UFRGS, 1994. (TI – 391)

[UDE 96] UDELL, J. Seus Negócios na Internet. **Byte Brasil,** São Paulo, v.5, n.8, p. 44-59, ago. 1996.

[VES 95] VESSEY, Iris; SRAVANAPUDI, Ajay Paul. CASE Tools as Collaborative Support Technologies. **Communications of the ACM,** New York, v.38, n.1, p.83-95, Jan. 1995,

[WAS 94] WASSERMAN, A. The Next Generation of Integrated Software Developments Environments. In: BRAZILIAN SYMPOSIUM OF SOFTWARE ENGINEERING , 8., 1994, Curitiba. **Anais...** Curitiba: Centro Internacional de Tecnologia de Software, 1994. p. 60-72. Palestra convidada.

[WIN 93] WINBLAD, Ann L; EDWARDS, Samuel D.; KING, David R. **Software Orientado a Objetos.** São Paulo: Makron Books, 1993.

[WIT 97] WITH CLASS FOR JAVA. **Java Programming using With Class.** Disponível por WWW em http://www.microgold.com/Stage/Tutor-java.html (02/06/97).

[YOU 92] YOURDON, Edward. **Análise Estruturada Moderna.** São Paulo: Campus, 1992.

[ZAR 90] ZARELLA, P. **CASE Tool Integration and Standarization.** Pittsburgh: Software Engineering Institute, 1990.

[ZHE 97] ZHENG, Ke; et al. **E-R GUI Database Design.** Disponível via HTTP em http://www.cs.uga.edu/~zheng/GUIDemo.html (12/09/97)

Informática  *UFRGS*

CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

"Ambiente Integrado de Modelagem Distribuída para Sistemas de Informação na Internet"

por

Leandro Bento Pompermaier

Dissertação apresentada aos Senhores:

Profa. Dra. Itana Maria de Souza Gimenes (UEM)

Prof. Dr. Marcelo Soares Pimenta

Prof. Dr. José Valdeni de Lima

Vista e permitida a impressão.
Porto Alegre, 12/03/99.

Prof. Dr. Roberto Tom Price,
Orientador.

Profa. Carla Maria Dal Sasso Freitas
Coordenadora do Programa de Pós-Graduação
em Computação - PPGC
Instituto de Informática - UFRGS