FRAME_BD - Uma Infra-estrutura para Construção de Ambientes Integrados para Projeto de Software

João Gualberto Rizzo Araujo (jgra@ufba.br)
Décio Fonseca (df@di.ufpe.br)

Universidade Federal de Pernambuco
CCEN - Departamento de Informática

19 de maio de 1995

Abstract

In this work, we present FRAME_BD, an infra-structure to support the creation of integrated environments for database design and software development. FRAME_BD offers a set of classes, organized into modules, that allows the construction of tools to support the software development process. FRAME_BD allows fast development of new tools by the reuse of existent classes in the infra-structure. FRAME_BD offers object persistence, graphical interfaces, hypertext links and error handling services. This speeds up the development of new tools and makes their integration smooth.

Resumo

Neste trabalho apresentamos FRAME_BD, uma infra-estrutura para suportar a criação de ambientes integrados para projeto de bases de dados e desenvolvimento de software. FRAME_BD oferece um conjunto de classes, organizadas em módulos, que permite a construção de ferramentas para suportar o processo de desenvolvimento de software. FRAME_BD oferece persistência de objetos, interfaces gráficas, ligações de hipertexto e serviços de manipulação de erros. Isto acelera o desenvolvimento de novas ferramentas e torna sua integração mais suave.

1 - Ferramentas CASE - Um Breve Histórico

As primeiras ferramentas para produção de software existentes foram os editores de texto e os compiladores, sem os quais é praticamente impossível criar algum sistema de software. Nos anos 70 começaram a surgir as primeiras ferramentas para documentação estruturada e análise dos sistemas de informação. Estas ferramentas iniciais eram geralmente baseadas em mainframes e possuiam interface apenas textual.

Com o avanço da análise estruturada, ficou evidente a necessidade da utilização de ferramentas automatizadas para lidar com o grande volume de informações gerado no processo de desenvolvimento. Muitas ferramentas foram ainda desenvolvidas no ambiente dos mainframes. Com o surgimento dos computadores pessoais e das interfaces gráficas vieram as ferramentas gráficas para suporte aos métodos estruturados. Apesar deste avanço, todas as informações capturadas no processo de desenvolvimento ficavam confinadas à própria ferramenta, dificultando a troca de informações entre ferramentas diferentes.

As chamadas ferramentas de segunda geração [Mar88] desenvolvidas nos anos 80, foram projetadas principalmente para suportar ainda os métodos estruturados, usando as notações gráficas correspondentes, mas incorporando uma maior aquisição semântica das

aplicações e reforçando as restrições e os caminhos das metodologias. A informação captada por estas ferramentas é armazenada em um dicionário de projeto, de forma que possa ser compartilhada por outras ferramentas do mesmo ambiente. Apesar de estarem limitadas a ferramentas do mesmo fabricante, as informações do dicionário podem ser exportadas em um padrão conhecido(em geral ASCII) para que possam ser aproveitadas por outras ferramentas.

No final dos anos 80, seguindo a mesma linha, apareceram ferramentas mais poderosas (ferrramentas CASE Integradas ou plataformas CASE), baseadas em repositórios globais, que servem para uma organização como um todo, assim como para projetos específicos, e que cobrem completamente o ciclo de desenvolvimeto das aplicações. Estas ferramentas, apesar de todo o seu potencial, são voltadas para métodos e tipos de aplicações específicos, com poucas facilidades de extensão. Nos anos 90, existe uma tendência com sistemas abertos e extensíveis, que facilitem a integração de ferramentas e a troca de informações sobre os projetos, que sejam adaptáveis a novos métodos e que sejam portáveis para diferentes plataformas.

Entre os trabalhos mais recentes nesta área estão os Ambientes Integrados para Desenvolvimento de Software (IPSE) [BM92] [NC92] [TN92]. Um ambiente integrado deve prover suporte a todas as atividades do processo de desenvolvimento de uma aplicação, desde a análise de viabilidade e levantamento de requisitos do sistema até a manutenção. Em um ambiente integrado os dados produzidos por uma ferramenta são armazenados em uma base de dados comum, de forma que possam ser utilizadas por outras. Os IPSE's devem possuir uma arquitetura aberta que possibilite a integração controlada de novas ferramentas, encaixando-as no processo global de desenvolvimento.

2 - Ambientes IPSE

Em termos simplificados um IPSE é composto de uma infraestrutura (ambiente), que provê serviços, e um conjunto de ferramentas que provê funcionalidades para desenvolvimento de software. A **interface pública** define a interação entre o ambiente e as ferramentas que ele se propõe a assistir. Os IPSEs começaram a ser desenvolvidos no final dos anos 80 e início dos anos 90. São ambientes relativamente novos onde ainda existe muito a ser definido.

O objetivo maior de um IPSE é oferecer um ambiente produtivo e eficiente para o desenvolvimento de aplicações com um alto nível de qualidade. Os IPSE's devem buscar um nível maior de gerenciamento de metodologias e controle de qualidade sobre os produtos gerados. Para atingir estes objetivos os IPSE's em geral devem satisfazer os seguintes pre-requisitos, como aponta [BM92]:

- Generalidade. O IPSE deve ser capaz de suportar uma gama variada de aplicações e métodos de desenvolvimento.
- Flexibilidade. Capacidade de se adequar a diferentes necessidades de uma variedade de usuários.
- Homogeneidade. Interface consistente para acesso aos serviços.
- Portabilidade. Independência de uma configuração de hardware especifica.
- Compatibilidade. Caminho suave para migração dos métodos e ferramentas existentes para um ambiente IPSE.

O IPSE permite a integração de diversas ferramentas em um mesmo ambiente, compartilhando uma base de dados comum, apresentando uma interface consistente ao usuário, sincronizando a utilização das ferramentas e estabelecendo mecanismos de comunicação entre as mesmas. O conjunto de ferramentas de um IPSE pode ser estendido de duas formas: através

da construção de novas ferramentas utilizando a infraestrutura oferecida ou através da incorporação de ferramentas externas já existentes.

Os termos CASE e I-CASE (Integrated CASE) são originados nas ferramentas de suporte aos métodos estruturados desenvolvidos nas duas últimas décadas. Com a evolução das ferramentas CASE as diferenças entre elas e os IPSE estão ficando menos claras. Mas algumas diferenças fundamentais ainda existem [BM92]:

- IPSE's tendem a suportar diversos métodos, enquanto as ferramentas CASE concentram-se em um linha única e provêm poucas facilidades de extensão.
- IPSE's têm sido mais utilizados em aplicações científicas e de engenharia, enquanto as ferramentas CASE são usadas em sistemas comerciais que lidam com grandes quantidades de dados (Data Intensive).
- As pesquisas em ambientes do tipo IPSE têm se centrado mais nos mecanismos de integração de ferramentas e na infra-estrutura propriamente dita do que nas funcionalidades e produtividade das ferramentas de projeto.

A arquitetura dos ambientes IPSE também é um tanto diferenciada da arquitetura das plataformas CASE convencionais. As plataformas CASE estão estruturadas como um conjunto de ferramentas específicas que compartilham um repositório de dados comum. Esta arquitetura não reforça nenhum tipo de padrão entre as diversas ferramentas e nem abre a possibilidade para integração de novas ferramentas que não sejam do próprio fabricante da plataforma.

Os ambientes IPSE possuem uma arquitetura interna bem mais elaborada, de forma a oferecer uma maior quantidade de serviços as ferramentas finais. Isto facilita a padronização, aumentando o controle da execução e composição de ferramentas e abre a oportunidade para integração de novas ferramentas ao ambiente. A figura 1 representa um modelo genérico de arquitetura para um IPSE.

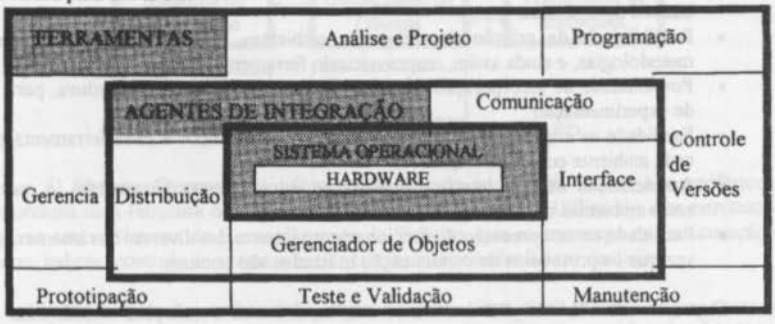

Figura 1 - Arquitetura de um IPSE

O gerenciador de objetos é responsável pela manutenção dos dados do ambiente. Em cooperação com o gerenciador de objetos estão os outros agentes de integração do sistema, que são os elementos responsáveis por oferecer serviços às ferramentas. Os agentes de integração reduzem o esforço de construção de novas ferramentas pois permitem a reutilização dos blocos básicos que compõem o ambiente. As ferramentas específicas para gerenciamento e desenvolvimento de projetos de software podem ser construídas utilizando-se os serviços do ambiente ou apenas integradas no ambiente através dos serviços de controle de processos. O conjunto total de ferramentas do ambiente deve ser integrado também no que diz respeito ao

acompanhamento da metodologia escolhida.Uma maior discussão sobre os aspectos de integração de ferramentas e integração do processo de desenvolvimento (metodologias) pode ser encontrado nos trabalhos de [NC92][BM92][TN92][HBP92][MS92] e [Jar92].

3 - A Infra-estrututra FRAME_BD

Nesta seção apresentaremos a infra-estrutura para construção de ambientes de desenvolvimento de aplicações denominado FRAME_BD. Cada ambiente criado a partir do FRAME_BD é composto por um conjunto de ferramentas **integrado** e **extensível**. As ferramentas são ditas integradas pois todas compartilham uma base de dados comum, uma forma unificada de interação com o usuário e possuem padrões de comunicação com as outras ferramentas e com o restante do ambiente. O conjunto é dito extensível pois é possível a criação de novas ferramentas ou extensão das já existentes.

3.1 - Introdução

O FRAME_BD foi desenvolvido a partir da necessidade de criação de ferramentas automatizadas para projeto de Banco de Dados que dessem suporte à metodologia orientada a objetos especificada em [Ara92].

Durante a fase de levantamento de informações sobre ferramentas CASE e ambientes integrados para construção de software notou-se uma clara tendência ao desenvolvimento de infra-estruturas de suporte à integração de ferramentas. Existem diversas vantagens no desenvolvimento de uma infra-estrutura de apoio, ao invés de ferramentas específicas, principalmente em um ambiente acadêmico onde novas idéias estão sendo constantemente avaliadas. Algumas destas vantagens podem ser enumeradas:

- Facilidade na construção de novas ferramentas, através da reutilização dos blocos básicos do ambiente.
- Possibilidade de criação de diferentes ambientes, dando suporte a diferentes metodologias, e ainda assim, reaproveitando ferramentas entre os ambientes.
- Possibilidade de incorporação de novos blocos à própria infra-estrutura, para fins de experimentação.
- Facilidade na criação de novos tipos de objetos específicos a cada ferramenta ou a cada ambiente como um todo.
- Apresentação de uma interface consistente entre diversas ferramentas, e mesmo, entre ambientes distintos.
- Facilidade na compreensão do funcionamento interno das diversas ferramentas, uma vez que os protocolos de comunicação utilizados são comuns.

O projeto do FRAME_BD baseou-se nas experiências e soluções encontradas nos Ambientes Integrados para Projeto de Software (IPSE) [Pre87] [Tho89] [BCN92] [Wyb91] [TN92]. Este tipo de ambiente possui uma estrutura básica definida, onde as ferramentas reais serão incorporadas.

O FRAME_BD oferece serviços de **persistência de objetos** e **criação uniformizada de interfaces**, e um sistema de **hipertexto**, com capacidade para armazenamento de qualquer tipo de objeto(textos, gráficos e sons); um módulo especializado em **tratamento de erros**; um protocolo padronizado de **troca de mensagens** entre as ferramentas, facilidades para criação e manipulação de **diagramas** através de **ferramentas de desenho** e **objetos gráficos;** e ainda o **coordenador**, que é o responsável pela consistência de apresentação do ambiente de acordo

acompanhamento da metodologia escolhida.Uma maior discussão sobre os aspectos de integração de ferramentas e integração do processo de desenvolvimento (metodologias) pode ser encontrado nos trabalhos de [NC92][BM92][TN92][HBP92][MS92] e [Jar92].

3 - A Infra-estrututra FRAME_BD

Nesta seção apresentaremos a infra-estrutura para construção de ambientes de desenvolvimento de aplicações denominado FRAME_BD. Cada ambiente criado a partir do FRAME_BD é composto por um conjunto de ferramentas **integrado** e **extensível**. As ferramentas são ditas integradas pois todas compartilham uma base de dados comum, uma forma unificada de interação com o usuário e possuem padrões de comunicação com as outras ferramentas e com o restante do ambiente. O conjunto é dito extensível pois é possível a criação de novas ferramentas ou extensão das já existentes.

3.1 - Introdução

O FRAME_BD foi desenvolvido a partir da necessidade de criação de ferramentas automatizadas para projeto de Banco de Dados que dessem suporte à metodologia orientada a objetos especificada em [Ara92].

Durante a fase de levantamento de informações sobre ferramentas CASE e ambientes integrados para construção de software notou-se uma clara tendência ao desenvolvimento de infra-estruturas de suporte à integração de ferramentas. Existem diversas vantagens no desenvolvimento de uma infra-estrutura de apoio, ao invés de ferramentas específicas, principalmente em um ambiente acadêmico onde novas idéias estão sendo constantemente avaliadas. Algumas destas vantagens podem ser enumeradas:

- Facilidade na construção de novas ferramentas, através da reutilização dos blocos básicos do ambiente.
- Possibilidade de criação de diferentes ambientes, dando suporte a diferentes metodologias, e ainda assim, reaproveitando ferramentas entre os ambientes.
- Possibilidade de incorporação de novos blocos à própria infra-estrutura, para fins de experimentação.
- Facilidade na criação de novos tipos de objetos específicos a cada ferramenta ou a cada ambiente como um todo.
- Apresentação de uma interface consistente entre diversas ferramentas, e mesmo, entre ambientes distintos.
- Facilidade na compreensão do funcionamento interno das diversas ferramentas, uma vez que os protocolos de comunicação utilizados são comuns.

O projeto do FRAME_BD baseou-se nas experiências e soluções encontradas nos Ambientes Integrados para Projeto de Software (IPSE) [Pre87] [Tho89] [BCN92] [Wyb91] [TN92]. Este tipo de ambiente possui uma estrutura básica definida, onde as ferramentas reais serão incorporadas.

O FRAME_BD oferece serviços de **persistência de objetos** e **criação uniformizada de interfaces**, e um sistema de **hipertexto**, com capacidade para armazenamento de qualquer tipo de objeto(textos, gráficos e sons); um módulo especializado em **tratamento de erros**; um protocolo padronizado de **troca de mensagens** entre as ferramentas, facilidades para criação e manipulação de **diagramas** através de **ferramentas de desenho** e **objetos gráficos**; e ainda o **coordenador**, que é o responsável pela consistência de apresentação do ambiente de acordo

com a metodologia escolhida. Estes serviços (módulos) permitem que haja uma maior harmonia entre a execução das ferramentas, facilitam a construção de interfaces consistentes e permitem um encadeamento metodológico das ferramentas. Desta forma as saídas geradas por umas podem ser utilizadas por outras, em uma sequência lógica e bem coordenada.

O FRAME_BD constitui-se, então, de um conjunto de módulos que permitem a criação de diversos ambientes, cada qual adaptado à sua metodologia de trabalho. Os módulos do FRAME_BD são formados por classes que fornecem as funcionalidades básicas para a construção das ferramentas.

3.2 - Organização Lógica dos Módulos

Os módulos existentes no FRAME_BD podem ser vistos logicamente como na figura 2. Os módulos com borda espessa pertencem ao FRAME_BD. A parte hachurada representa os elementos que são construídos em cada ambiente.

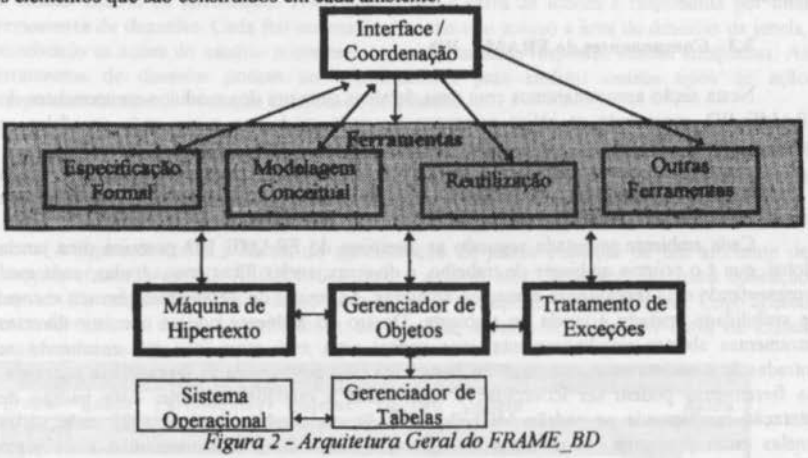

Figura 2 - Arquitetura Geral do FRAME_BD

O **Sistema Operacional**, fornece os serviços básicos de acesso aos periféricos e proporciona uma estrutura de arquivos para utilização dos discos. Utilizando esta estrutura de arquivos está o **Sistema de Gerenciamento de Tabelas**, que se encarrega da estruturação de tabelas, índices, controle de acesso e outras tarefas relacionadas à base de dados.

O **Módulo de Persistência de Objetos** fornece uma interface entre os programas de aplicação e o sistema relacional. Ele se encarrega de encapsular as funcionalidades do gerenciador relacional em métodos de mais alto nível, além de prover uma interface orientada a objetos para acesso à base de dados.

A **Máquina de Hipertexto** fornece às ferramentas a capacidade de tornar os seus objetos parte da malha geral de hipertexto.

O **Tratamento de Erros** foi unificado para evitar a propagações indesejáveis dentro do ambiente. Quando uma exceção é detectada, um objeto de erro é criado e assume o controle do sistema, geralmente dando um alerta ao usuário, podendo encerrar a execução

(**Interromper**), tentar novamente executar o procedimento causador da falha (**Repetir**) ou simplesmente continuar a execução (**Continuar**).

As **Ferramentas** serão responsáveis pela condução do usuário dentro da metodologia e pela aquisição do conhecimento sobre a aplicação através dos métodos e técnicas especificados na mesma. Este nível não é parte integrante da infra-estrutura básica do FRAME_BD e um conjunto de ferramentas específico deve ser desenvolvido para cada metodologia.

O nível de **Interface** é o responsável pela manutenção dos objetos de interação com o usuário, tais como janelas, diálogos e menus. Cada ferramenta criada fará uso destes objetos dentro do padrão de interface definido pelo ambiente.

O **Sistema de Coordenação** age em conjunto com a interface para garantir a consistência do ambiente apresentado ao usuário, seguindo as diretrizes da metodologia adotada.

3.3 - Componentes do FRAME_BD

Nesta seção apresentaremos com mais detalhes cada um dos módulos componentes do FRAME_BD, mostrando as idéias presentes em cada módulo e como cada um deles se posiciona em relação ao ambiente e às ferramentas que deverão ser construídas.

3.3.1 - Interface com o Usuário

Cada ambiente projetado segundo as diretrizes do FRAME_BD possuirá uma janela global, que é o próprio ambiente de trabalho, e diversas janelas filhas enquadradas, cada qual representando uma **ferramenta** dentro do ambiente. As janelas de ferramentas têm seu escopo de visibilidade limitado à janela do ambiente. Dentro do ambiente podem coexistir diversas ferramentas abertas simultaneamente, mas apenas uma está ativa, ou seja, recebendo as entradas do usuário a cada instante. Esta ferramenta ativa é chamada de **ferramenta corrente**. As ferramentas podem ser iconizadas e restauradas a qualquer instante. Este padrão de interação corresponde ao padrão MDI (Multiple Document Interface) [Pet92] onde várias janelas estão presentes em uma mesma aplicação, tratando com documentos ou objetos diferentes. Cada janela pode também apresentar diferentes visões de um mesmo documento ou objeto. O FRAME_BD leva mais adiante esta idéia propondo um padrão de utilização destas janelas dentro de um ambiente integrado e, além disto, manipulando uma grande diversidade de objetos, tudo sob o controle de um gerenciador central.

A **janela principal** possui um **menu** inicial que é oferecido assim que se entra no ambiente. Este menu deve permitir ao usuário o acesso aos seus projetos e as ferramentas disponíveis no ambiente, sendo manipulado pelas diversas **ferramentas do ambiente**. Cada ferramenta que é ativada, passando a ser a **ferramenta corrente**, pode modificar o menu do ambiente de forma que este reflita as suas necessidades. Mesmo com esta flexibilidade, o **coordenador** continua a ter controle sobre os itens de menu que podem ou não estar disponíveis em cada fase do projeto.

O **coordenador** é sempre notificado de mudanças no estado do sistema para poder habilitar ou desabilitar os botões da barra de ferramentas e os itens de menu para o usuário.

As **janelas de ferramentas** não possuem um menu próprio, fazendo uso do menu principal do ambiente. Todas podem possuir uma **barra de ícones** para agilizar a interação com o usuário. Esta barra de ícones ocupa um espaço horizontal no topo da ferramenta com largura igual à da própria janela. A barra de ícones pode ser definida e utilizada livremente pelas ferramentas. Cada ferramenta tem total controle sobre os botões da sua barra de ícones que podem ou não estar disponíveis aos usuários a cada momento.

Cada ferramenta possui um espaço que é chamado de **área de desenho**. Na área de desenho as ferramentas recebem as entradas do usuário e apresentam visualmente os seus resultados. Esta área ocupa o espaço da janela que vai da barra de ferramentas até o final da mesma. Ela permite a apresentação tanto de elementos textuais como gráficos ou uma combinação de ambos.

Cada ícone de uma barra pode estar associado a uma ação imediata ou a uma mudança no estado interno da ferramenta. Toda ação sobre a barra de ícones é respondida por uma **ferramenta de desenho**. Cada ferramenta de desenho tem acesso à área de desenho da janela, percebendo as ações do usuário sobre esta área e fornecendo respostas visuais adequadas. As ferramentas de desenho podem ser especializadas para realizar outros tipos de ação independentemente da interação com a área de desenho.

As ferramentas desenvolvidas para o ambiente recebem toda esta infra-estrutura já montada, realizando apenas as adaptações que lhe sejam específicas. Isto reforça os padrões de interface com o usuário e agiliza o processo de criação de novas ferramentas.

A figura **3** ilustra a forma de apresentação da janela principal de um ambiente de exemplo, chamado de BD_RECICLAR, onde o usuário está trabalhando em uma aplicação chamada Sistema de Matrículas, como pode ser visto no topo da janela principal. Logo abaixo do título temos o menu do ambiente e, em seguida, a barra de ícones. A barra de ícones contém somente dois botões.

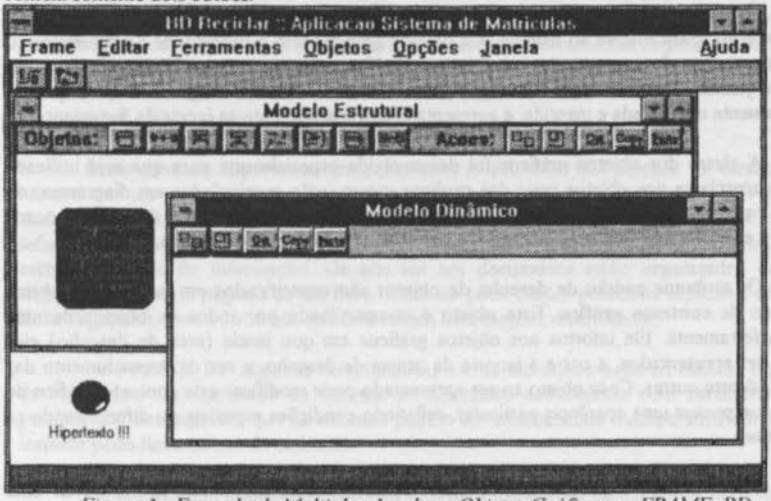

Figura 3 - Exemplo de Multiplas Janelas e Objetos Gráficos no FRAME_BD

3.3.2 - Classes para Criação de Ferramentas

As classes básicas para criação de novas ferramentas no ambiente são **TToolWindow**, que é a classe básica para a **janela da ferramenta**, **TToolBar**, que é a classe básica para a **barra de ícones**, e **TCanvas** que é a classe básica para criação da **área de desenho**. A criação de novas ferramentas pode ser feita de forma incremental. Para que se ter uma nova janela basta criar uma subclasse de **TToolWindow**, sem necessidade de se adicionar nenhuma funcionalidade à mesma. Deve-se informar ao **coordenador** em que circunstâncias esta ferramenta pode ser disponibilizada ou não.

A partir deste esboço inicial o projetista deverá partir para a definição de um menu apropriado à sua ferramenta, de uma barra de ícones específica, dos objetos que serão manipulados e das ferramentas de desenho apropriadas para lidar com estes objetos e responder às ações do usuário.

A barra de ícones de cada ferramenta no ambiente deve ser mantida por uma subclasse de **TToolBar**. A barra de ferramentas também tem condição de determinar a disponibilização dos ícones ao usuário a partir de mudanças no estado interno da ferramenta. A criação de subclasses de **TCanvas** também é justificada quando desejamos modificar o comportamento padrão de apresentação dos objetos.

3.3.3 - Objetos Gráficos e Ferramentas de Desenho

Tendo em vista que os ambientes a serem construídos em cima desta infra-estrutura serão ambientes com uma grande demanda por aplicações gráficas, com necessidade de criar diagramas e manipular objetos razoavelmente complexos, o FRAME_BD oferece um conjunto de classes, integradas com o sistema de interface, que torna a tarefa de construção destas ferramentas bastante simples.

Os **objetos gráficos** são aqueles que possuem características especiais que permitem apresentá-los graficamente ao usuário. Os objetos gráficos têm a capacidade de determinar o seu formato quando apresentados em uma janela. Eles podem ser apresentados em qualquer tipo de janela. Cada área de desenho possui uma lista de objetos gráficos que, quando propriamente inicializada e mantida, é apresentada automaticamente na janela da ferramenta.

A classe dos objetos gráficos foi desenvolvida especialmente para que seja utilizada como superclasse dos objetos reais dos projetos e que serão manipulados em diagramas, ou outros tipos de especificações, pelos usuários. Cada tipo de objeto pode determinar o seu formato e redefinir os seus atributos de apresentação (cor, fonte, tipo de linha etc.).

Os atributos padrão de desenho de objetos são especificados em um tipo de objeto chamado de **contexto gráfico**. Este objeto é compartilhado por todos os objetos de uma mesma ferramenta. Ele informa aos objetos gráficos em que janela (área de desenho) eles devem ser apresentados, a cor e a largura da caneta de desenho, a cor de preenchimento das figuras, dentre outras. Cada objeto ao ser apresentado pode modificar este contexto gráfico de forma que possua uma aparência particular, refletindo condições especiais ou diferenciando-se dos demais.

Para cada tipo de objeto gráfico específico a ser criado, devemos também criar uma **ferramenta de desenho** que cuide das interações entre o usuário e a área de desenho. A classe

das ferramentas de desenho é capaz de perceber as entradas do usuário e fornecer as respostas adequadas. A ferramenta de desenho é capaz de perceber os toques nos botões do mouse e operações de arrastar e soltar (drag-drop). Cada tipo específico de ferramenta deve prover as respostas adequadas ao tipo de desenho ou diagrama que está sendo criado.

As ferramentas de desenho podem verificar o correto posicionamento do mouse a cada ação do usuário dando-lhes uma resposta visual adequada. Uma ação, por exemplo, de arrastar um objeto corresponde a diversas ações contínuas de desenhá-lo e apagá-lo dando a impressão de movimento. Um toque no botão do mouse em uma posição não permitida para o tipo do objeto pode ser imediatamente informada ao usuário.

As classes para manutenção dos objetos gráficos são **TGObject**, que é a classe básica para os objetos gráficos, **TDrawingTool**, que é a classe básica das ferramentas de desenho que interagem com a área de desenho (**TCanvas**) e **TPaintContext**, que é o contexto gráfico compartilhado por todos os objetos gráficos e ferramentas de desenho de uma mesma ferramenta.

Estas classes, notadamente **TDrawingTool** e **TGObject**, não forçam a utilização de uma técnica particular para criação e apresentação de objetos. Elas apenas provêm uma forma de interação comum com o restante da ferramenta, no caso, a área de desenho e a barra de ícones. No momento em que o usuário pressiona um ícone em uma barra, a ferramenta de desenho associada é selecionada e imediatamente comunicada. Esta assume o controle, podendo tomar uma ação imediata ou esperar por novas entradas do usuário. À medida que novas entradas vão sendo feitas na área de desenho, a ferramenta de desenho vai sendo comunicada, permitindo assim que tarefas completamente diversas possam ser realizadas por diferentes ferramentas de desenho.

Na figura **3** é possível visualizar um objeto gráfico sendo mostrado na área de desenho de uma ferramenta parcialmente escondida. O objeto em questão é um dos objetos da base de dados geral do ambiente, mas herda características dos objetos gráficos que permitem a sua apresentação. Este objeto pode ser manipulado através das ferramentas disponíveis na barra de ícones da ferramenta.

3.3.4 - Hipertexto

O **sistema de hipertexto** fornecido com o FRAME_BD oferece as características básicas de um sistema convencional [Con87]. O sistema está organizado em torno de **documentos** que possuem dados relativos à sua criação, assunto, autores e outras informações. Cada documento é composto por uma sequência de **nós**. Cada nó contém uma certa quantidade de informação. Os nós em um documento estão organizados de forma sequencial, como as páginas de um livro. Cada nó pode conter ponteiros especiais, chamados **links** (ligações), para outros nós que contenham informações relacionadas.

No FRAME_BD algumas características extras são adicionadas ao modelo tradicional de hipertexto. Os elementos do sistema de hipertexto são objetos com características de persistência. Isto significa que os mesmos podem ser armazenados recuperados em qualquer instante pelas ferramentas do ambiente.

Os nós também podem ser especializados para tratamento de diferentes tipos de dados, estruturados ou não. A classe básica **Nó** contém apenas formas de gravação e recuperação de

informação, mas não tem conhecimento do significado ou de alguma possível estruturação desta informação. Desta forma conseguimos o armazenameto de diferentes tipos de dados como textos, gráficos, sons e até mesmo objetos do próprio FRAME_BD.

Os nós do sistema de hipertexto podem ser utilizados independentemente da estrutura organizada em documentos. Eles podem aparecer de forma isolada para conter informações adicionais sobre elementos do projeto ou para servir como elo de ligação entre objetos da aplicação que sejam intimamente relacionados. As ligações, definidas na classe **Link**, podem relacionar nós de diferentes documentos, bem como nós isolados.

Diversos trabalhos apresentam experiências de integração de sistemas de hipertexto e ambientes para produção de software [GS90] [CR92] [Lan91] [Big88].

3.3.5 - Persistência de Objetos

O ambiente FRAME_BD oferece às suas aplicações uma forma unificada de armazenamento de objetos de modo que qualquer aplicação pode acessar dados gerados por outras aplicações do ambiente. Para isto, basta que a aplicação seja capaz de manipular os tipos de objetos que lhe interessam, não necessitando ter conhecimento sobre como ou por quem foram gerados.

O módulo de persistência oferece uma interface uniforme para tratamento de objetos persistentes. Independentemente do tipo específico de objeto manipulado pela aplicação, a sua interface em relação às suas características de persistência é idêntica a qualquer outro tipo de objeto persistente. Apenas quatro métodos são utilizados na manipulação dos objetos persistentes: **Insert**, **Update**, **Delete** e **Retrieve**.

A classe de interface entre os objetos persistentes e a máquina de gerenciamento relacional é a classe **Persistente**. Esta classe fornece os mecanismos básicos de gravação, atualização, busca e remoção de objetos.

Para que uma classe possa tornar os seus objetos persistentes, ela deve ser subclasse, direta ou indiretamente, da classe **Persistente**. As subclasses de **Persistente** são responsáveis pela determinação das suas características específicas como o nome da tabela, onde seus valores devem ser armazenados, quantidade, nome e tipo dos seus atributos e campos de identificação dos objetos (chaves). As subclasses de **Persistente** devem apenas redefinir alguns métodos que tratem destas informações específicas e façam a movimentação de valores das variáveis de instância para o buffer de leitura e gravação e vice-versa.

Não é obrigatório que toda subclasse de **Persistente** defina uma tabela para que seus objetos sejam armazenados. Neste caso o sistema busca alguma tabela nas suas superclasses. Se alguma é encontrada, esta é utilizada para os objetos da classe. Se nenhuma tabela é encontrada, isto significa que os objetos desta classe não podem ser gravados.

Diferentes subclasses de uma mesma classe podem aproveitar uma tabela em comum, ou podem definir tabelas específicas para cada uma. As tabelas definidas nas subclasses têm precedência de utilização sobre as tabelas das superclasses. Uma subclasse pode optar por salvar parte de seus atributos na tabela da sua superclasse e outra parte na sua própria tabela.

O sistema de persistência permite também que, em uma mesma linha de herança, objetos de classes diferentes sejam armazenados em tabelas diferentes e que objetos das subclasses armazenem as características herdadas em suas próprias tabelas ou na tabela da superclasse. Além disto, atributos complexos podem ser armazenados em tabelas próprias separadas.

A classe **Persistente** utiliza os serviços definidos por um outro conjunto de classes que fazem a interação real com o sistema de gerenciamento relacional. Este esquema de classes é uma modelagem orientada a objetos de um sistema relacional genérico. Neste esquema temos as classes **Database**, que representa o banco de dados como um todo, **Table**, que representa as tabelas de um banco de dados relacional, **Record**, que representa os registros das tabelas, e **Field**, que representa os campos dos registros.Uma discussão sobre o mapeamento do modelo orientado a objetos para o modelo relacional [Cod70] pode ser encontrada em [Ara94].

3.3.6 - Tratamento de Erros

Em um ambiente de trabalho gráfico há uma clara distinção entre os elementos de interface e os objetos que efetivamente realizam as operações. Normalmente, os erros ocorrem nestas classes internas da aplicação devido a inconsistência de informações ou problemas relativos ao tratamento com a base de dados relacional. O sistema de tratamento de erros foi então modelado de forma a prover um mecanismo eficiente de comunicação entre as classes internas da aplicação e as classes de interface.

Na eventualidade de um erro qualquer, um objeto da classe **CSError** é criado a partir de um determinado código de exceção. Os objetos da classe **CSError** são também objetos persistentes e, baseados no código de exceção informado, têm condições de buscar informações sobre o erro, tal como mensagens para o usuário, tipos de erros (**Fatais** ou **Avisos**) e códigos de ação (**Continuar**, **Repetir** ou **Interromper**).

Uma vez detectado o erro e criado o objeto correspondente, este objeto é imediatamente retornado ao objeto que chamou o método. Caso este objeto seja capaz de tratar a exceção, a ação indicada pelo objeto de erro deve ser realizada e o objeto pode ser descartado. Caso o objeto chamador não seja capaz de tratar o erro, o objeto de erro deve ser passado ao objeto subsequente e assim por diante até chegarmos nas classes de interface.

Os objetos da classe **CSError** têm condições de apresentar uma mensagem ao usuário indicando o tipo de erro que ocorreu na aplicação e de cancelar a execução do sistema caso a ação indicada seja de interrupção. As ações de continuação ou de repetição devem ser realizadas pelos objetos que recebem o objeto de erro.

3.3.7 - Classes da Aplicação

As classes da aplicação são referentes aos objetos tratados em uma determinada metodologia. Cada ambiente criado a partir do FRAME_BD deverá manipular um conjunto diferenciado de objetos a depender da metodologia adotada.

O FRAME_BD oferece uma classe básica de apoio à definição de classes persistentes. A classe **Base** define dois atributos, nome e descrição. Ela define também um método que recupera uma lista de nomes de objetos, independente da tabela em que estejam armazenados. A tabela a ser utilizada neste método depende exclusivamente do tipo do objeto em particular.

Esta classe é subclasse direta de **Persistente**. Desta forma toda subclasse de **Base** herdará também características de persistência. Os detalhes para redefinição de métodos e atributos de **Persistente** podem ser encontrados [Ara94].

O novo ambiente pode optar por utilizar ou não a classe **Base**. A vantagem em utilizá-la está basicamente no tratamento de um objeto de hipertexto associado a cada objeto da classe. Todo objeto de uma subclasse de **Base** posui acesso a um nó do hipertexto. Este nó de hipertexto pode ser utilizado para documentação do objeto e para incluí-lo na rede global de hipertexto do sistema.

A classe **Base** serve para exemplificar como uma classe deve ser definida para que herde as características de persistência e redefina apenas as informações necessárias. Esta classe não possui uma tabela associada, ficando a gravação de suas variáveis de instância condicionada às definições de suas subclasses.

3.3.8 - Coordenador do Ambiente

O **coordenador** do FRAME_BD é o elemento que inicializa e mantém o ambiente. Ele é responsável pela ativação e desativação de ferramentas, pelo controle da integridade e coerência do ambiente para o usuário e pela comunicação entre as diversas ferramentas e por garantir que os passos especificados na metodologia estão sendo corretamente seguidos. Ele possui acesso a todas as informações existentes na base de dados e pode determinar se é possível ao usuário prosseguir ou retroceder nas fases da metodologia.

O FRAME_BD oferece, assim como nos outros módulos, uma classe que implementa as funções básicas de inicialização e manutenção do ambiente e fornece pontos específicos para que novas funcionalidades sejam adicionadas. O coordenador é definido de forma que cada ambiente específico herde suas características mantendo um padrão de interface consistente com o usuário.

O coordenador está associado a uma janela que funciona como limitador visual do ambiente. Cada ferramenta existente é uma janela independente, mas está sempre enquadrada na moldura da janela principal. A janela principal é o instrumento através do qual o coordenador exerce a sua função. Através da interface o coordenador pode inibir ou permitir determinadas ações do usuário.

Ele responde também pela coerência da interface do ambiente em relação à metodologia utilizada. Em cada fase do processo, nem todas as opções existententes no ambiente podem estar disponíveis ao usuário. Cabe ao coordenador selecionar quais operações podem ser realizadas em cada etapa.

Como a interação do usuário com as ferramentas é feita de forma direta, sem a intervenção do coordenador, é necessário que as ferramentas solicitem ao coordenador que verifiquem o estado do ambiente sempre que estiverem alterando-o de alguma forma. Esta verificação de consistência pode disponibilizar novos ícones e ferramentas e fazer com que outros sejam desabilitados. Este procedimento pode inclusive fechar ferramentas que estejam correntemente abertas caso a sua presença não seja adequada às novas condições do ambiente.

Outro aspecto interessante do coordenador é que ele mantém um controle rígido sobre o menu do ambiente. Cada ferramenta que é ativada pode modificar o menu do ambiente de

acordo com as suas necessidades. Sempre que um novo menu é associado à janela do ambiente, o coordenador deve verificar quais as opções deste menu que devem ou não ser disponibilizadas aos usuários. Além disto, toda ação do usuário sobre o menu deve ser passada diretamente ao coordenador, que por sua vez, decide se executa alguma ação ou envia uma mensagem à ferramenta ativa indicando a seleção do usuário.

Este protocolo pode ser melhor entendido através da figura **4**. Aqui podemos ver o coordenador como um controlador da interface do ambiente para o usuário. Ele funciona como um filtro que só permite a utilização uma parte do ambiente total. Na figura, as ferramentas 1 e 3 podem ser utilizadas pelo usuário nesta fase da metodologia. O mesmo não acontece com as ferramentas 2, 4 e 5. Algumas ações do usuário sobre as ferramentas ou sobre o próprio ambiente fazem com que uma mensagem de verificação de consistência seja enviada ao coordenador. Durante esta verificação o coordenador tem acesso à base de dados e total controle sobre os elementos da interface.

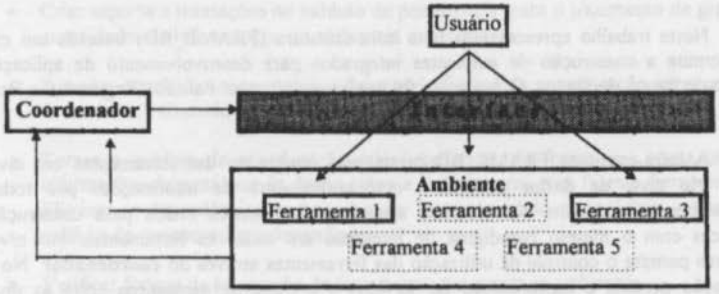

Figura 4 - O coordenador metodológico

A classe básica para a criação de uma ambiente de trabalho e um coordenador é **TMainWindow**. **TMainWindow** oferece suporte a navegação entre as diversas ferramentas existentes no ambiente, além de "iconização", maximização e arrumação das mesmas. Ela é responsável pela inicialização e registro da aplicação, pela criação e manutenção da barra de ferramentas da janela principal e manutenção de um menu padrão.

Cada ambiente a ser construído utilizando o FRAME_BD deve definir uma subclasse de **TMainWindow** para ser o seu coordenador específico.

3.3.9 - Comunicação entre as Aplicações no FRAME_BD

Toda a comunicação entre o **coordenador** e as ferramentas e entre as próprias ferramentas é realizado através de mensagens. Cada mensagem é composta de um código de identificação, um identificador da aplicação (ferramenta, janela) que deve receber a mensagem, e dois parâmetros adicionais que contêm dados específicos da mensagem. Estes dados podem conter valores simples ou ponteiros para estruturas de objetos conhecidos de ambas as partes.

No modelo de comunicação do FRAME_BD, a comunicação entre o coordenador e as ferramentas é realizado de forma direta, uma vez que cada aplicação (ferramenta) conhece o identifcador do coordenador e vice-versa. Para a troca de mensagens entre as próprias

ferramentas, o coordenador deve agir como um intermediador das mensagens. Esta intermediação é importante, pois possibilita ao coordenador um controle mais apurado sobre o que está ocorrendo no ambiente, além de evitar possíveis problemas devido, por exemplo, a mensagens enviadas a ferramentas que não estão ativas no momento.

Desta forma, as diversas ferramentas do ambiente, têm necessidade apenas de conhecer quais as outras ferramentas do ambiente e quais as mensagens que elas são capazes de responder. Isto elimina a necessidade de controle, por parte de cada ferramenta, de quais as ferramentas ativas no momento e quais os seus identificadores durante aquela execução em particular. Este controle é mantido pelo coordenador que pode tomar as decisões apropriadas frente a uma situação de impossibilidade da chegada da mensagem no seu destino e devolve um código de *status* à aplicação que enviou a mensagem indicando o sucesso da operação ou possíveis erros ocorridos durante o envio da mensagem.

4 - Considerações Finais

Neste trabalho apresentamos uma infra-estrutura (FRAME_BD) baseada em classes que permite a construção de ambientes integrados para desenvolvimento de aplicações e projeto de Banco de Dados. O Ambiente foi implementado com Paradox Engine 3.0 e Borland C++ 3.1.

A infra-estrutura FRAME_BD prima pela integração das ferramentas em diversos níveis. No nível de **dados** permite o compartilhamento de informações por todas as ferramentas do ambiente. No nível de **apresentação** fornece meios para construção de interfaces com o mesmo paradigma de interação em todas as ferramentas. No nível de **processo** permite o controle da utilização das ferramentas através do **coordenador**. No nível de **função** permite a especificação de serviços e a troca de mensagens entre as diversas ferramentas. No nível de **gerência** permite o desenvolvimento de ferramentas que monitorem o conteúdo de toda a base de dados relatando as reais atividades desenvolvidas em um projeto.

Alguns elementos criados no FRAME_BD são essenciais para a criação de ambientes realmente coesos, integrados e produtivos. O **coordenador** metodológico é fundamental para guiar o projetista dentro da metodologia e garantir uma qualidade mínima no trabalho realizado. O processo de **comunicação entre ferramentas**, com a especificação de serviços e a troca de informações em tempo de execução, contribui bastante para aumentar o nível de integração e a produtividade na construção de novas ferramentas e no projeto de aplicações.

O sistema de **hipertexto** é um elemento importante em ambientes para desenvolvimento de aplicações, pois possui inúmeras possibilidades de aproveitamento. Os sistemas de **persistência** e construção de **interface** permitem que o projetista das ferramentas trabalhe mais nos objetivos da sua ferramenta do que nos elementos de suporte. O sistema de **tratamento de erros** padroniza o tratamento dos mesmos evitando propagações indesejadas pelo ambiente.

A utilização dos conceitos de orientação a objetos permitiu o uso de técnicas de projeto que minimizam o esforço inicial de construção de uma ferramenta e permite o seu desenvolvimento incremental. A orientação a objetos permitiu também a propagação de funcionalidades aos objetos através da herança e herança múltipla. O nível de reutilização de código também é elevado, pois todos os protocolos especificados no ambiente são parcialmente implementados e totalmente reutilizados.

O FRAME_BD é uma experiência na construção de ambientes integrados para desenvolvimento de aplicações. Muito trabalho ainda pode ainda realizado, tanto a nível de especificação como a nivel de implementação para incrementar as capacidades do ambiente.

Alguns destes aspectos podem ser notados desde já com a utilização do BD_RECICLAR e com a própria convivência com a infra-estrutura:

- Criar uma maior independência do módulo de interface da biblioteca de classes utilizada, para aumentar a portabilidade do sistema e permitir uma migração menos traumática para um outro tipo de ambiente.

- Desenvolver um gerenciador de objetos, também genérico, mas inteligente que fosse capaz de perceber diferentes tipos de ligação entre objetos da base de dados e manter a consistência entre eles.

- Criar suporte a transações no módulo de persistência, para o tratamento de grandes quantidades de informações em ambientes multiusuário.

- Suportar o trabalho cooperativo, adicionando formas de comunicação entre grupos de usuários e controle de atividades.

- Tornar o módulo de coordenação mais inteligente, de forma que possa utilizar informações da metabase de dados sobre a metodologia, suas fases e ferramentas e Eliminar as dependências do coordenador com o módulo de interface e criar um módulo de coordenação independente.

- Verificar formas de integração de ferramentas externas aos ambientes.

- Buscar novas soluções de persistência dos objetos para eliminar a necessidade de mapeamentos do modelo orientado a objetos para o modelo relacional e permitir a distribuição dos dados, mantendo, se possível, o mesmo protocolo de interação.

6 - Bibliografia

[Ara92] J. B. Araujo Jr. Uma Metodologia para desenvolvimento de Sistemas Orientados a Objetos. Tese de Mestrado, Universidade Federal de Pernambuco, Outubro 1992.

[Ara94] J. G. R. Araujo. FRAME_BD - Uma Infra-estrutura para Construção de Ambientes Integrados para Projetos de Software. Tese de Mestrado, Universidade Federal de Pernambuco, Agosto 1994.

[Ban89] M. Bandeira. Modelo E/D de Dados. Tese de Mestrado, Uniersidade Federal de Pernambuco, Dezembro 1989.

[BCN92] C. Batini, S. Ceri e S. B. Navathe. *Conceptual Database Design: An Entity-Relationship Approach*. The Benjamin/Cummings Publishing Company, 1992.

[Big88] J. Bigelow. Hypertext and CASE. *IEEE Software*, Março 1988.

[BM92] A. W. Brown e J. A. McDermid. Learning from IPSE's Mistakes. *IEEE Software*, Março 1992.

[Car90] R. M. M. Carrano. Estudo Formal de um Modelo de Dados Orientado a Objetos. Tese de Mestrado, UFPE, Novembro 1990.

[Cod70] E. F. Codd. A Relational Model for Large Shared Data Banks. *Communications of the ACM*, 13(6), Junho 1970.

[Con87] J. Conklin. A Survey of Hypertext. Relatório Técnico STP-356-86, MCC, Austin, Texas, fevereiro 1987.

[CR92] J. L. Cybulski e K. Reed. A Hypertext Based Software-Engineering Environment. *IEEE Software*, Março 1992.

[GP90] M. R. Girardi e T. Price. Um Sistema para Recuperação de Classes Reutilizáveis no Desenvolvimento Orientado a Objetos. *RBC Revista Brasileira de Computação*, 6(2), 1990.

[GS90] P. K. Garg e W. Scacchi. A Hypertext system to Manage Software Life-Cycle Documents. *IEEE Software*, Maio 1990.

[HBP92] A. R. Hevner, S. A. Becker e L. B. Pedowitz. Integrated CASE for Cleanroom development. *IEEE Software*, Março 1992.

[Huf92] C. C. Huff. Elements of a Realistic CASE Tool Adoption Budget. *Communications of the ACM*, 35(4), Abril 1992.

[Jar92] M. Jarke. Strategies for Integrating CASE Environments. *IEEE Software*, Março 1992.

[Kem92] C. F. Kemerer. How the Learning Curve Affects CASE Tool Adoption. *IEEE Software*, Março 1992.

[Lan91] D. B. Lange. Constructing a Hypertext Based Program Development Environment Using a Comercial OODBMS. Relatório técnico, Department of Computer Science - Technical University of Denmark, 1991.

[Mar88] C. F. Martin. Second Generation CASE Tools: A Challenge to Vendors. *IEEE Software*, 5(2), 1988.

[MS92] P. Mi e W Scacchi. Process Integration in CASE Environments. *IEEE Software*, Março 1992.

[NC92] R. J. Norman e M. Chen. A Framework for Integrated CASE. *IEEE Software*, Março 1992.

[Pet92] C. Petzold. *Programando para Windows 3*. Makron Books do Brasil, 1992.

Pre87] R. S. Pressman. *Software Engineering: A Practioer's Approach*. McGraw Hill Int. Editions,1987.

[Som89] I. Sommerville. *Software Engineering*. Addison Wesley Publishing Company, 1989.

[Tho89] I. Thomas. PCTE Interfaces: Supporting Tools in Software-Engineering Environments. *IEEE Software*, Novembro 1989.

[TN92] I. Thomas e B. A. Nejmeh. Definitions of Tool Integration for Environments. *IEEE Software*, Março 1992.

[Wyb91] N. Wybolt. Perspectives on CASE Tool Integration. *Software Engineering Notes*, 16(3), Julho 1991.