Um Framework de Ferramenta Interativa para um Ambiente Integrado de Testes

Marcos Salenko Guimarães

Um Framework de Ferramenta Interativa para um Ambiente Integrado de Testes[1]

Marcos Salenko Guimarães[2]

Instituto de Computação – UNICAMP

Banca Examinadora:

- Eliane Martins (orientadora)[3]
- Cecília Mary Fischer Rubira[3]
- Itana Maria de Souza Gimenes[4]
- Luiz Eduardo Buzato (suplente)[3]

[1] Dissertação apresentada ao Instituto de Computação da UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência de Computação.

[2] O autor é Bacharel em Ciência da Computação da UFGo.

[3] Professor(a) do Instituto de Computação - UNICAMP.

[4] Professora do Departamento de Informática - Centro de Ciências Tecnológicas - Universidade Estadual de Maringá.

Um Framework de Ferramenta Interativa para um Ambiente Integrado de Testes

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida pelo Sr. Marcos Salenko Guimarães e aprovada pela Comissão Julgadora.

Campinas, 07 de janeiro de 1997.

Eliane Martins

Profa. Eliane Martins
Orientadora

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

UNICAMP
BIBLIOTECA CENTRAL

Tese de Mestrado defendida e aprovada em 29 de novembro de 1996 pela Banca Examinadora composta pelos Professores Doutores

Profª. Drª. Itana Maria de Souza Gimenes

Profª. Drª. Cecília Mary Fischer Rubira

Profª. Drª. Eliane Martins

AGRADECIMENTOS

Ao Criador, pela permissão.

À orientadora Dra. Eliane Martins pela disponibilidade e atenção.

Ao CNPq e FAPESP, pelo apoio financeiro.

Ao Erasmo G. Munuera, Alexandre de Melo Rezende, Maurício Atanasio, Ana Waleska X. Araújo, Márcio Stefani, Raul C. Requião e Nevton Cereja que apoiaram diretamente ou indiretamente.

Agradeço à Profa. Cecília M. F. Rubira, pela prontidão em solucionar dúvidas.

À Itana Maria de Souza Gimenes, por ser membro da banca examinadora.

UM FRAMEWORK DE FERRAMENTA INTERATIVA PARA UM AMBIENTE INTEGRADO DE TESTES

RESUMO

Este trabalho apresenta o desenvolvimento de um framework para construção de interface com o usuário que atenda aos requisitos de um ambiente integrado de testes considerado (o ATIFS), que são: portabilidade, extensibilidade, uniformidade e a independência entre a interface com o usuário e a aplicação.

Foi utilizado o paradigma orientado a objetos pois este facilita a satisfação de alguns dos requisitos acima, principalmente a extensibilidade e uniformidade. Além destas vantagens, este paradigma também facilita a reutilização, particularmente útil para atender ao requisito de uniformidade.

A reutilização é facilitada com o uso de framework, conjunto de classes e da interação entre elas. Para a reutilização de um framework basta: instanciar as classes existentes ou criar novas subclasses a partir das classes existentes. Frameworks podem ser utilizados em vários domínios de aplicação, mas a maioria dos estudos até então são relativos ao desenvolvimento de interface com o usuário. Foi feito um estudo de alguns modelos (MVC, MVC++, MV, Framework baseado em Modo e Modelo em camadas) a fim de adotar o mais adequado para o ATIFS.

A partir do modelo adotado (o MVC++), foi possível projetar o framework do ATIFS, o qual irá permitir a construção de interfaces com o usuário de ferramentas interativas que compõem este ambiente.

O framework definido para o ATIFS foi implementado de forma que os projetistas poderão instanciar as classes referentes à interface ao construirem suas ferramentas. Desse modo, garante-se que todas as ferramentas interativas seguirão o padrão de comportamento e apresentação do ATIFS.

ABSTRACT

This work presents a framework for developing user interfaces that meets the requirements of an integrated environment such as portability, extensibility, independence between user interface and application, and uniformity.

These requirements are easier to achieve with object-oriented paradigm. Reusability, an important property to obtain the desired uniformity, is one example. One of the advantages of the object-oriented model is the facility of reusability. This characteristics is particularly useful for meeting the uniformity requirement.

Reusability becomes easier with the use of frameworks. A framework is composed by a set of classes and the interactions between them. For using frameworks, the classes are instantiated or new subclasses are created from the existing ones. Frameworks may be used in several applications domains and there are many models for user interfaces. The study of some models (MVC, MV, MVC++, Mode Framework and Layered Model) was done in order to select one to use in this work.

Using the selected model (MVC++), the design of a framework for the target environment has been done for permiting the construction of user interfaces for the interactive tools that integrate this environment.

The defined framework is implemented in a manner that the designers need only to instantiate classes for developing of their tools. In this way, all interactive tools will follow the same standard of the apresentation and behavior.

DEDICAÇÃO

À toda a minha família.
Em especial meus pais. Sem eles este trabalho
não seria possível: *Sancler* e *Eleonora.*
Meu irmão: *Marcelo.*
Minha irmã: *Milena.*
A todos meus sinceros agradecimentos pela compreensão e apoio.

Microsoft Windows™, *Microsoft Word*™ e *Microsoft Access*™ são marcas registradas da Microsoft Corporation.
Motif™ é marca registrada da Open Software Foundation.
X Window System™ é marca registrada da Massachusetts Institute Technology.
MacApp™ *e Macintosh*™ são marcas registradas da Apple Inc.
UNIX ™ é marca registrada da AT&T.

Índice

Capítulo 1 INTRODUÇÃO 1

PARTE I - CONTEXTO DO TRABALHO 4

Capítulo 2 AMBIENTES INTEGRADOS E INTERFACE COM O USUÁRIO 5

2.1 INTEGRAÇÃO DE FERRAMENTAS 5
2.1.1 CONCEITOS GERAIS 5
2.1.2 ASPECTO DE INTEGRAÇÃO DE APRESENTAÇÃO 7
2.2 REQUISITOS DE INTERFACE COM O USUÁRIO PARA UM AMBIENTE INTEGRADO 8
2.2.1 UNIFORMIDADE (OU CONSISTÊNCIA) 9
2.2.2 PORTABILIDADE 9
2.2.3 SEPARAÇÃO DE INTERFACE COM O USUÁRIO E APLICAÇÃO 9
2.2.4 EXTENSIBILIDADE 10
2.2.5 FORMAS DE PROCESSAR AS ENTRADAS DO USUÁRIO 10
2.2.6 MANIPULAÇÃO DIRETA 10
2.3 ESTRUTURA DA INTERFACE COM O USUÁRIO 11
2.4 TRABALHOS CORRELATOS 11
2.5 PROBLEMA 12
2.6 CONCLUSÃO 13

Capítulo 3 DESCRIÇÃO DO ATIFS 14

3.1 INTRODUÇÃO 14
3.1.1 CONCEITOS GERAIS 14
3.1.2 OBJETIVO 15
3.2 ARQUITETURA DO ATIFS 16
3.3 APLICAÇÃO PRÁTICA 18
3.4 CONCLUSÃO 18

Capítulo 4 INTERFACE COM O USUÁRIO E ORIENTAÇÃO A OBJETOS 19

4.1 CONCEITOS BÁSICOS 19
4.1.1 FACILIDADE DE USO DA INTERFACE 20
4.1.2 ESTILOS DE INTERAÇÃO 21
4.1.3 METODOLOGIA PARA DESENVOLVIMENTO DE INTERFACES 22
4.2 DESENVOLVIMENTO DE INTERFACE ORIENTADA A OBJETOS 23
4.2.1 FRAMEWORKS 23
4.2.1.1 CONCEITO 23
4.2.1.2 CONSTRUÇÃO DE FRAMEWORKS 24
4.2.2 MODELOS PARA DESENVOLVIMENTO DE INTERFACES 26
4.2.2.1 MODELO-VISÃO-CONTROLADOR (MVC) 26
4.2.2.2 MODELO BASEADO EM MODO 27
4.2.2.3 MVC++ 31
4.2.2.4 MODELO EM CAMADAS 32
4.2.2.5 MODELO-VISÃO (MV) 33
4.2.3 EXEMPLOS DE FRAMEWORKS PARA CONSTRUÇÃO DE INTERFACES 34
4.2.3.1 MacApp 34
4.2.3.2 ET++ 34
4.3 CONCLUSÃO 36

PARTE II - DESENVOLVIMENTO E APLICAÇÃO DE UM FRAMEWORK DE INTERFACE PARA O ATIFS 38

Capítulo 5 UM PROTÓTIPO DE INTERFACE COM O USUÁRIO 39

5.1 INTRODUÇÃO 39
5.2 ANÁLISE DO SISTEMA 40
5.2.1 OBJETIVO 40
5.2.2 ANÁLISE DO USUÁRIO 40
5.2.3 ANÁLISE DE TAREFAS 40
5.2.3.1 NÍVEL 1 40
5.2.3.2 NÍVEL 2 41
5.2.3.3 NÍVEL 3 41
5.3 PROJETO DA INTERAÇÃO 42
5.3.1 INTRODUÇÃO 42
5.3.2 APRESENTAÇÃO DA INTERFACE DO ATIFS 43
5.3.2.1 JANELA PRINCIPAL 43

5.3.2.2 PROJETO 46
5.3.2.3 EDITOR 48
5.3.2.4 FERRAMENTA 52
5.4 CONCLUSÃO 53

Capítulo 6 PROJETO DE SOFTWARE DA INTERFACE 55
6.1 INTRODUÇÃO 55
6.2 COMPONENTE VISÃO 56
6.2.1 FERRAMENTA DE CONSTRUÇÃO DE INTERFACES (TOOLKIT) 56
6.2.2 PROTOCOLO DE APRESENTAÇÃO 57
6.2.2.1 JANELA 59
6.2.2.2 BARRA DE MENU 60
6.2.2.3 QUADRO DE DIÁLOGO 60
6.3 COMPONENTE MODELO 61
6.3.1 NORMAS PARA CRIAR SUBCLASSES DE MODELO 61
6.3.2 MODELO 62
6.4 COMPONENTE CONTROLADOR 62
6.4.1 FORMAS DE IMPLEMENTAÇÃO DO CONTROLADOR 63
6.4.1.1 HERANÇA MÚLTIPLA 63
6.4.1.2 REFLEXÃO COMPUTACIONAL 64
6.4.1.3 DELEGAÇÃO 67
6.4.2 PROJETO DO CONTROLADOR DO ATIFS 68
6.4.2.1 CLASSE CONTROLADORA 68
6.4.2.2 INTERAÇÃO ENTRE CONTROLADOR, VISÃO E MODELO 69
6.4.3 IMPLEMENTAÇÃO DO CONTROLADOR 72
6.5 CONCLUSÃO 75

Capítulo 7 DESCRIÇÃO DO USO DO FRAMEWORK 77
7.1 INTRODUÇÃO 77
7.2 DOCUMENTAÇÃO DE FRAMEWORKS 78
7.3 COMPONENTE VISÃO 79
7.3.1 PADRÃO 1: PROPÓSITO 79
7.3.2 PADRÃO 2: IMPLEMENTAÇÃO DE MENUS 81
7.3.3 PADRÃO 3: IMPLEMENTAÇÃO DE UM QUADRO DE DIÁLOGO 82
7.3.4 EXEMPLO 85
7.4 COMPONENTE MODELO 86
7.4.1 PADRÃO 1: PROPÓSITO 86
7.4.2 EXEMPLO 86
7.5 COMPONENTE CONTROLADOR 88
7.5.1 PADRÃO 1: PROPÓSITO 88
7.5.2 PADRÃO 2: IMPLEMENTAÇÃO DO CONTROLADOR 88
7.5.3 EXEMPLO 89
7.6 CONCLUSÃO 93

Capítulo 8 CONCLUSÃO E EXTENSÕES FUTURAS 94
8.1 O QUE FOI FEITO? 94
8.2 O QUE DEVE SER FEITO? 96
8.3 EXTENSÕES FUTURAS 96

REFERÊNCIAS BIBLIOGRÁFICAS 98

ANEXOS 103

Apêndice A FRAMEWORK DA INTERFACE DO ATIFS 104
A.1 COMPONENTE VISÃO 105
A.1.1 Janela 105
A.1.2 BarraMenu 105
A.1.3 JanelaPrincipal 107
A.1.4 JanelaEditor 109
A.1.5 JanelaFerramenta 111
A.1.6 ÁreaTexto 111
A.1.7 ÁreaOpções 112
A.1.8 PainelJPrincipal 112
A.1.9 QuadroDialogo 112
A.1.10 Painel 113
A.1.11 Lista 113
A.1.12 CaixaEscolha 113
A.1.13 Botão 113
A.1.14 CaixaTexto 114
A.2 CONTROLADOR 114
A.2.1 VControlador 114
A.2.2 VControladorAmbiente 114
A.2.3 VControladorFerramenta 115
A.2.4 VControladorEditor 116
A.3 COMPONENTE MODELO 116
A.4 CLASSE APLICACÃO 116

Apêndice B SERVIÇOS IMPLEMENTADOS DO FRAMEWORK 118
B.1 SERVIÇOS GENÉRICOS 118
B.2 SERVIÇOS AGRUPADOS POR CLASSES 119
B.2.1 Aplicação 119
B.2.2 JanelaPrincipal, JanelaEditor e JanelaFerramenta 120
B.2.3 ÁreaTexto 121
B.2.4 ÁreaOpções 122
B.2.5 Menu 123
B.2.6 QuadroDialogo 123
B.2.7 Painel 124
B.2.8 Botão 124

B.2.9 CaixaEscolha 125
B.2.10 CaixaTexto 125
B.2.11 Lista 126
B.2.12 Controlador 127

Lista de Figuras

Figura 3.1 Estrutura do ambiente de testes 16

Figura 4.1 Estrutura geral de um sistema interativo 20
Figura 4.2 Estrutura do MVC 27
Figura 4.3 Estrutura de um objeto Mode 28
Figura 4.4 Caixa de diálogo de um modo com dois submodos 29
Figura 4.5 Modelo baseado em Modo 30
Figura 4.6 O modelo MVC++ 31
Figura 4.7 Modelo em camadas 32
Figura 4.8 Modelo do MV. 33
Figura 4.9 Arquitetura do ET++ 35

Figura 5.1 Janela principal do ATIFS 44
Figura 5.2 Quadro de diálogo Palheta 45
Figura 5.3 Quadro de diálogo para editar o projeto 47
Figura 5.4 Janela Principal no contexto de um projeto selecionado pelo usuário 48
Figura 5.5 Janela de editor 49
Figura 5.6 Abertura de arquivos 50
Figura 5.7 Exemplo de um quadro de diálogo de mensagem 51
Figura 5.8 Janela de ferramenta 53

Figura 6.1 Estrutura de classes do componente Visão 58
Figura 6.2 Modelo de objetos do Protocolo de Apresentação. 59
Figura 6.3 O modelo de objetos do componente Modelo 62
Figura 6.4 Derivando o Controlador através de herança múltipla. 63
Figura 6.5 Mecanismo de reflexão computacional 65
Figura 6.6 Implementação da interface baseada em reflexão computacional 66
Figura 6.7 Implementação baseada em delegação 67
Figura 6.8 Controlador contém a Visão e associado ao Modelo. 69
Figura 6.9 Traço de eventos quando o usuário seleciona item "Abrir..." do menu Arquivo 70
Figura 6.10 Traço de eventos em um caso de erro. 71
Figura 6.11 Traço de eventos quando um item de menu opcional é selecionado. 72
Figura 6.12 Modelo do componente Controlador. 73
Figura 6.13 Framework da interface do ATIFS 74

Figura 7.1 Janela da Ferramenta XXX 81
Figura 7.2 Layout de um quadro de diálogo 84
Figura 7.3 Quadro de diálogo Palheta 85
Figura 7.4 Classe pertencente ao Modelo 87
Figura 7.5 Traço de eventos entre Visão, Controlador e Modelo 90

Figura A.1 Modelo de objetos do Protocolo de Apresentação 106
Figura A.2 Modelo dinâmico da classe JanelaPrincipal 108
Figura A.3 Modelo dinâmico da classe JanelaEditor 110
Figura A.4 Modelo de objetos dos controladores virtuais 115
Figura A.5 O Modelo de objetos do componente Modelo 116
Figura A.6 Classe Aplicação 117

Capítulo 1

INTRODUÇÃO

O enfoque deste trabalho é voltado para a interface com o usuário do ambiente integrado de testes chamado ATIFS (ver detalhes no capítulo 2). Vale também lembrar que o enfoque dado nesse trabalho é voltado para aspectos de Engenharia de Software, e não para aspectos referentes à interação humana, à psicologia e nem aos fatores humanos.

É sabido que projetar e implementar uma interface com o usuário adequada para um determinado sistema, e, ao mesmo tempo, satisfazer as exigências do usuário é considerado difícil. Constatou-se que 50% do código e do desenvolvimento de um sistema são voltados para a interface **[Mye94]**.

É crucial estabelecer, nesta interface, uma *independência de diálogo* (a ser explicada na seção 2.2.3), a interface e o sistema devem ser o mais independente possível. Assim, esta independência faz com que se tenha maior modularidade tanto na parte de interface quanto na aplicação e, portanto, a sua manutenção torna-se mais fácil.

Quanto a separação de interface com o usuário e aplicação, segundo **[Mye94]**, na prática, é difícil de separar-se a interface da aplicação. Algumas ferramentas modernas auxiliam esta separação oferecendo procedimentos do tipo "*call-back*", nos quais cada *widget* (menu, botão, barra de rolamento, campo de entrada de cadeia de caracteres) na tela requer, pelo menos, um procedimento da aplicação. O problema é que, em uma interface, existem centenas de *widgets*. Se ocorrer uma alteração na aplicação ou na interface, o trabalho para ajustar o sistema é muito grande. Portanto, a manutenção do sistema deste tipo é um pesadelo.

A *facilidade de extensão (ou extensibilidade)* também é um requisito importante pois à medida que a aplicação evolui, a interface deve também permitir a sua evolução.

Finalmente, a interface deve ser *uniforme*. Os componentes que compõem o ambiente integrado devem ter a mesma forma de interação com a interface com o usuário e, também, por outro lado, o usuário deve interagir de uma forma padronizada com todos os componentes do ambiente.

Para atender a alguns destes requisitos mencionados anteriormente, esta interface será desenvolvida utilizando o paradigma orientado a objetos. Este paradigma é vantajoso pois oferece soluções a várias dificuldades no desenvolvimento de interfaces, dentre as quais **[LVC89]** e **[KZ90]**:

- provê bom mecanismo de abstração de dados;
- oferece facilidade de extensão através do mecanismo de herança;
- facilita a modularização, o que permite uma separação de interface e aplicação;
- facilita a manutenção;
- facilita a reutilização, tanto de projeto quanto de código.

Além disso, o encapsulamento e facilidade de extensão favorecem os dois primeiros requisitos mencionados anteriormente (independência de diálogo e facilidade de extensão). Este paradigma também favorece à uniformidade graças a facilidade de reutilização de objetos.

A facilidade de reutilização foi um outro fator que influenciou na escolha do paradigma de orientação a objetos. Um encapsulamento da interface com o usuário faz com que se obtenha componentes reutilizáveis e adequados para abstrações de dados de uma aplicação particular. Assim, o projetista de ferramentas ou do ambiente simplesmente reutiliza os objetos já implementados reforçando a uniformidade da interface com o usuário do ambiente.

E finalmente, uma outra razão para o uso deste paradigma é que o ambiente integrado de testes considerado neste trabalho também é orientado a objetos, dessa forma, garantindo-se a uniformidade desde o desenvolvimento do ambiente.

O objetivo desse trabalho é propor uma estrutura de projeto (ou "framework") para desenvolver uma interface com o usuário dentro dos padrões definidos (ver seção 5.3) no ambiente integrado de testes. A interface vai permitir que o usuário utilize as funcionalidades do ambiente (inclusive às suas ferramentas que o compõem) utilizando ferramentas modernas disponíveis para o diálogo homem-computador: mecanismos de janelas, menus, uso do mouse, entre outras facilidades.

O trabalho está dividido em duas partes. A primeira parte trata do contexto do trabalho contendo os seguintes capítulos: no capítulo 2, tem-se um estudo sobre ambientes integrados e a sua interface com o usuário; o capítulo 3 menciona um estudo teórico sobre o ambiente integrado ATIFS; no capítulo 4, tem-se um estudo teórico sobre a

interface com o usuário de modo geral e é escolhido um modelo a ser utilizado para desenvolver um framework de interface com o usuário do ATIFS.

Na segunda parte da dissertação, tem-se o desenvolvimento e aplicação de uma framework para o ATIFS. Nesta parte, no capítulo 5, tem-se uma descrição do protótipo de interface com o usuário do ATIFS. Nos capítulos 6 e 7, tem-se o desenvolvimento e aplicação do framework para este ambiente. Finalmente, a dissertação se encerra com a conclusão e os anexos.

PARTE I - CONTEXTO DO TRABALHO

Esta parte trata do contexto do trabalho. Inicialmente, tem-se um estudo sobre ambientes integrados e interface com o usuário no contexto geral.

Em seguida, tem-se um estudo sobre o ATIFS que é um ambiente integrado de testes considerado neste trabalho.

Esta parte finaliza com um estudo de modelos orientados a objetos para desenvolvimento de interface com o usuário considerados neste trabalho.

Capítulo 2

AMBIENTES INTEGRADOS E INTERFACE COM O USUÁRIO

Este capítulo descreve alguns conceitos fundamentais de ambientes integrados e, em seguida, aborda aspectos referentes à interface com o usuário nestes ambientes.

Inicialmente, tem-se uma visão geral sobre integração de ferramentas. Na próxima seção, tem-se uma descrição de interface com o usuário de ambientes integrados. A terceira seção descreve a estrutura da interface com o usuário. Em seguida, tem-se uma descrição de trabalhos correlatos referentes ao contexto deste capítulo. A quinta seção menciona o problema de interface com o usuário para ambientes integrados. Finalmente, este capítulo se encerra com a sua conclusão.

2.1 INTEGRAÇÃO DE FERRAMENTAS

Nesta seção, serão apresentados alguns conceitos sobre a integração de ferramentas e, em seguida, tem-se uma descrição do aspecto de integração de apresentação de um ambiente.

2.1.1 CONCEITOS GERAIS

As ferramentas (ou ferramentas CASE) são sistemas que dão suporte a atividades específicas do desenvolvimento de software **[Som95]**. Por exemplo, o *Microsoft Word* é uma ferramenta de documentação.

Para cobrir uma ou mais atividades, um conjunto de ferramentas é necessário. Portanto, há necessidade de integrar ferramentas em um único sistema. Este único sistema, segundo **[Som95]**, é denominado *CASE Integrado* que é obtido quando as ferramentas CASE trabalham juntos de forma integrada.

Neste trabalho, ao invés de usar o termo "CASE Integrado", serão utilizados os termos "ambiente" e "ambiente integrado" para designar um conjunto de ferramentas integradas entre si.

Um ambiente integrado pode ser **[Som95]**:

- *aberto* - quando permite acréscimo e decréscimo de ferramentas;
- *fechado* - quando não permite acréscimo e nem decréscimo de ferramentas; o conjunto de ferramentas não varia no ambiente.

Para obter um ambiente integrado, é preciso integrar ferramentas provendo as seguintes características **[Pre95a]**:

- oferecer um mecanismo para compartilhar as informações de engenharia de software entre todas as ferramentas contidas no ambiente;
- permitir que a mudança efetuada num item de informação seja rastreada até outros itens de informação relacionados;
- oferecer controle de versão e gerenciamento de configuração global para todas as informações de engenharia de software;
- permitir acesso direto, não-seqüencial, a qualquer ferramenta contida no ambiente;
- estabelecer suporte automatizado para um contexto procedimental de trabalho de engenharia de software que integre as ferramentas e os dados numa estrutura de divisão de trabalho padrão;
- possibilitar que os usuários de cada ferramenta experimentem uma visão e uma percepção consistentes em nível de interface pessoa-computador;
- apoiar a comunicação entre os engenheiros de software;
- coletar tanto métricas administrativas como técnicas que possam ser usadas para melhorar o processo e o produto.

Para que um ambiente incorpore estas características de forma coerente e concisa, o ambiente integrado deve ter uma boa integração de ferramentas. A integração pode ser definida de duas formas**[BFW92]**:

- IPSE - tem-se uma infra-estrutura básica definida durante a sua fase de análise e projeto do ambiente. Esta infra-estrutura possui objetos para serem utilizados pelas as ferramentas que compõem o ambiente;
- Integração de CASE - concentra-se somente nas ferramentas. Para que as ferramentas se comuniquem, estas são adaptadas individualmente e, posteriormente, é feita uma adaptação aos serviços de integração.

O IPSE é uma forma de integração que geralmente é desenvolvida em meio acadêmico, e se tem concentrado principalmente na integração de dados (será vista posteriormente).

A Integração de CASE é uma forma de integração, geralmente desenvolvida na área comercial, que foi introduzida por causa das necessidades de mercado de integrar as ferramentas já existentes.

A integração pode ser feita em cinco aspectos, a saber **[Som95]**:

- *apresentação* - os usuários do ambiente devem interagir com as ferramentas de forma consistente fazendo com que as novas ferramentas acrescentadas no ambiente sejam mais fáceis de aprender. Esta integração é realizada pelos serviços de interface com o usuário;
- *plataforma* - garante às ferramentas a transparência relativa a plataforma utilizada;
- *controle* - prover mecanismos para que as ferramentas possam transferir dados entre si, bem como usar serviços uma das outras;
- *dados* - é a capacidade de compartilhar informações de projeto entre ferramentas. Esta integração é realizada pelo repositório e serviços de integração por dados;
- *processo* - dá suporte a controle de processo de controle de software.

Como este trabalho trata da interface com o usuário, o aspecto que será enfocado aqui é relativo à integração da apresentação que será apresentado na próxima seção. Se o leitor deseja mais detalhes sobre integração de ferramentas, ver em **[BFW92]**, **[Som95]** e **[CN92]**.

2.1.2 ASPECTO DE INTEGRAÇÃO DE APRESENTAÇÃO

O aspecto de integração de apresentação **[Som95]** garante que as ferramentas de um ambiente tenham o mesmo estilo de interação e o mesmo conjunto de padrões comuns de interação com o usuário. As ferramentas devem ter aparências semelhantes. Desta forma, reduz-se o tempo de aprendizado do usuário quando, por exemplo, uma nova ferramenta é integrada em um ambiente.

Para que se tenha estas propriedades de interface com o usuário, tem-se três níveis de integração de apresentação **[Som95]**:

- *integração de sistema de janela* - as ferramentas são integradas neste nível quando se usa o mesmo sistema de janelas e se apresenta uma interface comum dos comandos relativos à manipulação de janelas. As janelas têm a mesma aparência e os mesmos comandos para mover, redimensionar e "iconizar" (transformar uma janela em um simbolo gráfico chamado ícone) janelas;
- *integração de comandos* - as ferramentas são integradas neste nível quando se tem a mesma forma de fornecer comandos. Quando se trata de uma interface textual, as sintaxes das linhas de comandos e parâmetros são as mesmas para todos os comandos. Quando se trata de interface gráfica com menus e botões, os comandos possuem os mesmos nomes. Os itens de menu são posicionados nos mesmos lugares em cada aplicação. As mesmas representações (ícones) são usadas em todos os sub-sistemas para definir botões e menus;

- *integração de interação* - este nível se aplica a interfaces do tipo manipulação direta onde o usuário interage diretamente com entidades textuais ou gráficas da interface. A integração de interação significa que se tem as mesmas operações de manipulação direta (por exemplo, seleção e remoção) são fornecidas em todos os sub-sistemas. Portanto, se um texto é selecionado através de duplo-clique do botão do mouse, esta operação de seleção deve ser aplicável a todas as entidades em um diagrama realizando-a da mesma forma (usando um duplo-clique). Exemplos onde a integração de apresentação é suportada são sistemas de processadores de texto e sistemas de editores gráficos.

Em ambientes abertos, a integração de interface a nível de sistema de janela é mais difícil de realizar. Nestes ambientes, as ferramentas são desenvolvidas em diferentes tempos por diferentes desenvolvedores. Como o ambiente obtem novas ferramentas, se torna progressivamente mais difícil de manter a uniformidade da interface. As diretrizes de integração de interface podem ser fornecidas mas os projetistas das diretrizes não podem prever todos os possíveis usos do ambiente.

Nos ambientes UNIX, o sistema de janelas *de facto* é o *X-Windows* com o toolkit *Motif*. Neste sistema, tem-se somente a integração a nível de janelas. Portanto, as ferramentas e ambientes para UNIX são raramente bem integrados a nível de interação e de comando.

Na plataforma PC, o padrão *de facto* é o Microsoft Windows. Este é melhor que o X/ Motif pois suporta os três níveis de integração de apresentação. As janelas incluem uma caixa de ferramentas que suporta as diretrizes de construção de menus e formas específicas de interação, bem como um sistema de janela comum. A maneira mais fácil de implementar uma interface de ferramentas é seguir estas diretrizes. Como resultado, a maioria das ferramentas integradas no Windows possuem uma aparência e comportamento consistentes.

Em ambientes fechados, geralmente se tem boa integração de apresentação. Como novas ferramentas não podem ser introduzidas, é possível obter uma interface comum em todas as ferramentas integradas em um ambiente. As convenções e padrões podem ser desenvolvidos para diferentes ferramentas de um ambiente.

Com base nestas descrições, note que a interface com o usuário deve ter algumas características especiais para atender às exigências dos ambientes integrados, as quais serão mostradas no próximo item.

2.2 REQUISITOS DE INTERFACE COM O USUÁRIO PARA UM AMBIENTE INTEGRADO

Para atender aos aspectos de integração de apresentação, foram levantados os requisitos mínimos de interface com o usuário para ambientes integrados, que serão vistos a seguir.

2.2.1 UNIFORMIDADE (OU CONSISTÊNCIA)

Esta terminologia tem dois enfoques dependendo do seu contexto. Para o usuário, a uniformidade [YTT88] significa que a interface deve oferecer o mesmo padrão de interação para todos os componentes do ambiente. A uniformidade permite ao usuário maior facilidade de operação com o sistema favorecendo, então, a usabilidade do sistema.

Para os projetistas de ferramentas, os componentes do ambiente interagem da mesma forma com a interface. Há um esquema uniforme (ou padronizado) de interação entre ferramentas e a interface, combinação/comunicação de ferramentas, compartilhamento de dados, gerenciamento de objetos persistentes e adição de novas ferramentas.

Assim sendo, além de ser construída com base em um sistema de gerenciamento e controle de janelas que seja o padrão da plataforma na qual o ambiente está sendo implementado (ex.: Windows ou Motif), a interface com o usuário também deve fornecer, aos projetistas dos sub-sistemas, as diretrizes para garantir um padrão de apresentação e de interação entre a interface e os sub-sistemas.

Segundo [TBC+88], uma interface uniforme faz com que o mesmo conjunto de operações disponível tenha o mesmo sentido em toda a interface do sistema. Assim, reduz a dificuldade de aprendizado do usuário na operação de um sistema.

No contexto de aspecto de integração de apresentação, as ferramentas devem ser bem integradas tanto nos aspectos de aparência e comportamento quanto nos aspectos de paradigma de interação (vistos na seção anterior - seção 2.1.2).

Sobre os níveis de integração de apresentação vistos na seção 2.1.2, quanto maior quantidade de níveis de integração de apresentação são realizadas, melhor é a integração de apresentação de um ambiente.

2.2.2 PORTABILIDADE

A interface com o usuário deve ser portável, é possível executar a interface em diferentes plataformas sem alterar o seu código-fonte. Este requisito é importante pois o ambiente pode rodar em várias plataformas.

2.2.3 SEPARAÇÃO DE INTERFACE COM O USUÁRIO E APLICAÇÃO

Alguns autores chamam esta distinção de *independência de diálogo* [HH89], o domínio da interface deve ser independente do domínio da aplicação. Para reforçar esta independência, a interface deve ser desenvolvida de forma separada e independente da aplicação.

As principais vantagens de se garantir a independência de diálogo são [HH89]:

1. menor fluxo de controle e dados entre a interface e o sistema devido à esta independência. Esta troca de informações é bem organizada e só ocorre quando é realmente necessária;
2. quando ocorre uma modificação da aplicação, esta não implica em grandes alterações na interface e vice-versa.

Este requisito é considerado como um dos mais importantes, principalmente quando se trata de um ambiente integrado (detalhes no [TBC+88]). O ambiente pode mudar de configuração (acréscimo de novas ferramentas, por exemplo). Esta mudança de configuração implica em mudanças no domínio da aplicação. Portanto, a interface com o usuário deve ser o mais independente possível para que não sofra grandes modificações quando tiver de se adaptar a novas configurações.

A interface também pode sofrer alterações, portanto, quanto maior esta independência, menor serão feitas as mudanças no domínio da aplicação para se adaptar a uma nova interface.

2.2.4 EXTENSIBILIDADE

Este requisito se refere à facilidade de estender a interface com o usuário (acréscimo de novos tipos de dados de exibição, por exemplo). A extensibilidade é importante para o ambiente, pois todo o ambiente integrado pode ser estendido, por exemplo, através do acréscimo de novas ferramentas.

2.2.5 FORMAS DE PROCESSAR AS ENTRADAS DO USUÁRIO

É sabido que a interface recebe eventos do usuário. Há três formas de processar as entradas do usuário, que são classificadas da seguinte forma:

1. *Modelo de controle interno (ou prompting)* - se as rotinas de entrada aparecem como sub-rotinas da aplicação. Este tipo de interface é altamente preemptivo, o usuário tem um controle do programa muito restrito pois o controle do sistema é feito pela aplicação.
2. *Modelo de controle externo (ou dispatching)* - a aplicação aparece como uma sub-rotina do processador de entrada. Esta abordagem possui como principal desvantagem: uma distorção da lógica de algumas aplicações forçando os programadores a possuírem uma estrutura "plana" da aplicação, a aplicação é simplesmente considerada como uma camada que oferece um conjunto de serviços.
3. *Modelo concorrente* - as rotinas de entrada da interface e a aplicação são logicamente processos concorrentes e cooperativos.

2.2.6 MANIPULAÇÃO DIRETA

É um paradigma de interface que utiliza representações visuais(objetos gráficos) manipuladas pelo usuário através de um dispositivo (por exemplo, mouse). Este paradigma requer uma alta taxa de comunicação entre interface e aplicação.

2.3 ESTRUTURA DA INTERFACE COM O USUÁRIO

A interface com o usuário de ferramentas integradas, para atender aos requisitos mencionados acima, deve ser composta por duas camadas **[Pre95a]**:

- *Kit de Ferramenta de Interfaces* - neste trabalho, o termo "kit de ferramenta de interfaces" não será utilizado. Ao invés deste termo, será usado o termo "ferramenta de construção de interfaces" ou simplesmente *toolkit*. A *Ferramenta de Construção de Interfaces* contém software para gerenciamento de interfaces pessoa-computador e uma biblioteca de objetos de *display*. Existem várias ferramentas como, por exemplo: WXWindows **[Sma94]**, SUIT **[CPP92]**, Tcl/Tk **[Ous94]**.
- *Protocolo de Apresentação* - é o conjunto de diretrizes que dá a todas as ferramentas a mesma aparência e impressão. Convenções sobre *layout* de tela, nomes e organizações de menus, ícones, nomes de objetos, o uso do mouse e do teclado e o mecanismo de acesso às ferramentas são todos definidos como parte do protocolo de apresentação.

Portanto, é possível obter uma interface uniforme (apresentação e comportamento padronizado) a partir de um *toolkit* existente criando uma camada adicional *Protocolo de Apresentação* que deve definir todo o padrão de interface de um ambiente utilizando serviços de um *toolkit*. Esta abordagem será vista com maiores detalhes posteriormente.

2.4 TRABALHOS CORRELATOS

Este trabalho trata dos aspectos relativos à integração da apresentação de ferramentas interativas que integram um ambiente de desenvolvimento de software. Foram, portanto, estudados alguns trabalhos tanto na área de desenvolvimento de interface com o usuário quanto na área de desenvolvimento de ambientes integrados.

Os trabalhos referentes ao desenvolvimento de interface, de uma maneira geral, são voltados para o desenvolvimento de aplicações genéricas, não levando em conta os aspectos de integração da apresentação mencionados na seção 2.2. Estes trabalhos não serão citados aqui, pois fogem um pouco aos objetivos deste projeto.

Com relação aos estudos referentes a ambientes integrados, foram vistos alguns trabalhos que enfocam os aspectos de integração de apresentação. Estes trabalhos são citados a seguir.

Os trabalhos **[TBC+88]**, **[YTT88]**, **[TNB+94]**, **[TNB+95]** relatam um ambiente integrado chamado *Arcadia-1* atendendo a todos os requisitos mencionados na seção 2.2. Entretanto, esta abordagem utiliza um método de separação entre interface e aplicação bastante complexo e de difícil implementação.

O trabalho [KZ90] menciona alguns modelos de interfaces considerados (MVC e APV). Em seguida, o autor se direciona para o modelo APV preocupando-se principalmente com a uniformidade.

Em [NO93], tem-se um estudo sobre interface com o usuário com base no PCTE (framework para construção de ambientes integrados). Este trabalho utiliza como padrão de interface, o *X Windows System*.

Um trabalho mais recente é apresentado em [OCM95] o qual mostra o desenvolvimento de um framework de interface com o usuário que pode ser utilizado através de uma biblioteca de classes. Esta abordagem é considerada como um ponto importante para o desenvolvimento de interface com o usuário para um ambiente integrado pois usa o processo de reutilização de código para facilitar o atendimento do requisito de uniformidade.

No contexto da interface com o usuário do ATIFS, tem-se o objetivo de desenvolver um framework de interface que permita a reutilização tanto de projeto quanto ao código de interface (este tema será visto em detalhes posteriormente - no capítulo 6). Este framework é implementado como uma biblioteca de classes para que os projetistas de ferramentas possam utilizá-la a fim de implementar uma interface com o usuário da sua ferramenta.

2.5 PROBLEMA

O objetivo desse trabalho é obter um framework (ver o conceito de framework na seção 4.2.1) que permite desenvolver uma interface com o usuário para o ambiente integrado ATIFS (verá visto no próximo capítulo), que garanta o atendimento dos requisitos mencionados na seção 2.2.

Para atender a alguns destes requisitos, será utilizado o paradigma orientado a objetos para o desenvolvimento da interface, pois este oferece, entre outras vantagens, encapsulamento, modularidade e facilidade de extensão. Assim, este paradigma favorece os dois últimos requisitos mencionados (independência de diálogo e facilidade de extensão).

Como o ATIFS também é orientado a objetos, o requisito da uniformidade é favorecido pois definir um padrão de interação entre o sistema e a interface é facilitado, pois tanto o sistema quanto a interface são orientados a objetos.

A facilidade de reutilização foi um outro fator que influenciou na escolha do paradigma de orientação a objetos. Um encapsulamento da interface com o usuário faz com que se obtenha componentes reutilizáveis e adequados para abstrações de dados de uma aplicação particular. Assim, o projetista de ferramentas ou do ambiente simplesmente reutiliza os objetos já implementados reforçando a uniformidade da interface com o usuário do ambiente.

O framework permite reutilizar projeto de interface e também pode ser implementado como bibliotecas de classes para reutilização de códigos. Portanto, a uniformidade e reutilização são temas bastante relevantes neste trabalho.

2.6 CONCLUSÃO

Foram vistos os conceitos básicos sobre ambientes integrados descrevendo as suas características bem como algumas noções de interface com o usuário para este sistema.

Posteriormente, com base nas características de ambientes e da sua interface com o usuário, foram levantados os requisitos mínimos de interface com o usuário para ambientes integrados.

Foi visto também que é importante que se tenha uma camada de Protocolo de Apresentação e Ferramenta de Construção de Interfaces para que seja feita a integração de apresentação do ambiente. Assim, as interfaces com o usuário das ferramentas que compõem o ambiente tornam-se consistentes.

Em seguida, foram vistas as características de estrutura de interface com o usuário tais como: uniformidade, separação de partes da interface com o usuário e aplicação, formas de processar as entradas do usuário e manipulação direta.

Em seguida, foram vistos os trabalhos correlatos no contexto de interface com o usuário de ambientes integrados. Finalmente, este capítulo se encerrou com o problema deste trabalho.

No próximo capítulo, tem-se um estudo teórico sobre o ambiente considerado neste trabalho que é o ATIFS apresentando a sua definição e sua arquitetura.

Capítulo 3

DESCRIÇÃO DO ATIFS

Neste capítulo, tem-se uma descrição detalhada do Ambiente de Teste baseado em Injeção de Falhas por *Software* (ATIFS)[1]. Inicialmente, tem-se uma seção introdutória sobre o referido ambiente de teste e em seguida, uma seção descrevendo a arquitetura deste ambiente. O capítulo se encerra de uma aplicação preliminar deste ambiente.

3.1 INTRODUÇÃO

O ambiente[2] de teste é um ambiente aberto que deve dar suporte a vários tipos de testes de protocolo, especialmente de protocolos tolerantes a falhas, tais como: teste de tolerância a falhas, teste de desempenho, teste de interoperabilidade e teste de conformidade. Estes testes serão descritos posteriormente. A seguir, define-se algumas terminologias importantes usadas no contexto do ATIFS.

3.1.1 CONCEITOS GERAIS

A falha é a causa suposta ou constatada de um erro. O erro é a parte do estado do sistema que pode levar a um defeito[3] (quando o serviço fornecido pelo sistema não está de acordo com a especificação): um erro afetando o serviço indica que um defeito ocorreu ou está para ocorrer.

1. Este ambiente ainda está em desenvolvimento.
2. O termo "Ambiente" serve para designar um conjunto de ferramentas integradas.
3. A terminologia em português está baseada em [LPS+89]

3.1.2 OBJETIVO

O objetivo do ambiente é oferecer suporte aos seguintes testes:

1. *testes de tolerância a falhas* - determinam se os mecanismos que detectam os erros/ falhas, agem de acordo com a especificação do sistema. Caso contrário, seja quando não detecta algum erro/falha ou seja quando não trata um erro/falha conforme à especificação, considera-se que o sistema está com defeito. No ATIFS, os testes de tolerância a falhas serão feitos usando a técnica de injeção de falhas. A **injeção de falhas** pode adquirir diversas formas, segundo o nível de abstração utilizado para representar o sistema, e o nível de aplicação das falhas. Existem diferentes formas de aplicação de falhas **[Mar93]**. No ATIFS, é considerada a injeção de falhas por software, na qual erros são introduzidos com o objetivo de emular a consequência de falhas físicas no sistema, e podem consistir em alterações no conteúdo dos registros ou da memória, em alterações na sequência de instruções (emula falhas de CPU), ou ainda na perda ou duplicação de mensagens (para emular falhas no meio de comunicação);
2. *testes de conformidade* - determinam se a implementação se comporta de acordo com a especificação do protocolo;
3. *testes de interoperabilidade* - verificam se duas implementações conforme à mesma norma podem efetivamente interoperar. O fato de que uma implementação está de acordo com a norma de um protocolo não garante que ela seja capaz de interoperar com outra implementação conforme à mesma norma. Entre outros motivos, dada a variedade de opções oferecidas, é possível que duas implementações conformes à mesma norma não sejam capazes de interoperar devido à escolha de opções incompatíveis. Daí a necessidade de realizar este tipo de teste;
4. *teste de desempenho* - colocam em evidência os problemas relativos ao dimensionamento dos elementos de um sistema de comunicação e à duração de sua utilização. Eles podem ser divididos em:
   - 4.1. *testes de capacidade* - permitem determinar o comportamento do sistema quando se tem um grande número de entidades de teste operando simultaneamente, e determinar a degradação do serviço fornecido em função do aumento do número de entidades;
   - 4.2. *testes de resistência* - permitem determinar, para uma carga constante, a degradação do serviço fornecido ao longo do tempo;
   - 4.3. *testes de carga* - visam observar o comportamento do sistema em presença de uma carga importante;
   - 4.4. *avaliação do desempenho* - visam obter medidas do desempenho do sistema, tais como: tempo de tratamento de uma PDU (unidade de dados do protocolo), vazão máxima, tempo de reação a eventos diversos.

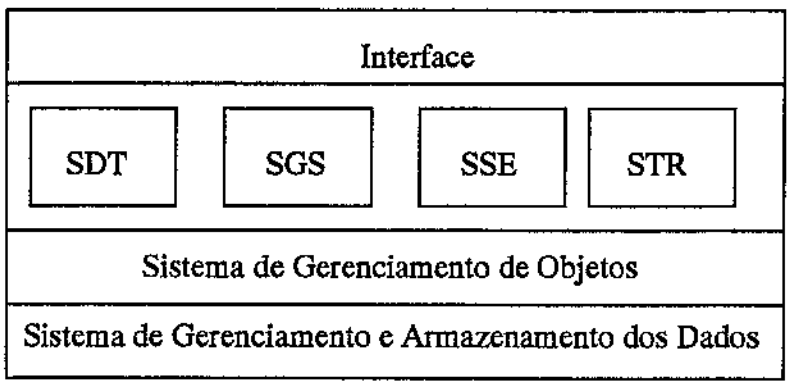
A seguir, tem-se uma descrição da arquitetura do ATIFS.

3.2 ARQUITETURA DO ATIFS

A figura 2.1 mostra o particionamento das funcionalidades que visam cobrir todas as atividades do processo de testes, que são as seguintes:

1. desenvolvimento dos testes;
2. execução dos testes;
3. análise dos resultados.

Figura 2.1 Estrutura do ambiente de testes



Desta figura, vê-se que o ATIFS está dividido nos seguintes sub-sistemas: Desenvolvimento de Testes (SDT), Geração do Script (SGS), Suporte à Execução (SSE) e Tratamento dos Resultados (STR). Além destes sub-sistemas, o ATIFS dispõe ainda de Interface, Sistema de Gerenciamento de Objetos e Sistema de Gerenciamento e Armazenamento dos Dados. Estes componentes estão descritos de forma mais detalhada logo a seguir.

INTERFACE

A interface deverá:

1. fornecer ao usuário o acesso a todas as funcionalidades oferecidas pelo ambiente. Dessa forma, o usuário poderá tanto construir quanto acompanhar a execução das experiências de forma interativa. Deverá também ser possível o acesso às estatísticas realizadas sobre os resultados parciais;
2. permitir que as ferramentas possam exibir ao usuário: os dados (de forma reduzida) para uma análise manual; os resultados parciais das experiências, para acompanhamento; os resultados de análises estatísticas realizadas após o término das experiências, sob forma de histogramas, curvas, ou outras.

Essa interface deverá utilizar todas as ferramentas modernas disponíveis para o diálogo homem-máquina, que oferecem: mecanismos de janelas, menus, uso do mouse, entre outras facilidades. Para isto, serão empregadas as bibliotecas disponíveis no IBM-PC (sistema *Microsoft Windows*).

A interface dá suporte ao aspecto de integração de apresentação do ATIFS (ver seção 2.1.2) garantindo o padrão de apresentação e comportamento da interface do ATIFS.

DESENVOLVIMENTO DE TESTES (SDT)

Este componente inclui basicamente as funções de gerenciamento de casos de testes e das falhas. Os casos de testes podem ser gerados de forma semi-automática, a partir de uma especificação na forma de máquinas de estados finitos. As falhas são obtidas estatisticamente, a partir da definição de um conjunto de falhas consideradas.

GERAÇÃO DO SCRIPT (SGS)

Esta função permite que o usuário defina como os testes vão ser realizados, requerendo para tanto informações tais como: duração dos testes, número de casos de testes a serem aplicados, número de falhas, número de nós do sistema.

SUPORTE À EXECUÇÃO (SSE)

Este componente inclui as funções de controlar a execução dos testes, monitorar o sistema em teste e supervisionar os recursos necessários para a realização dos mesmos.

TRATAMENTO DOS RESULTADOS (STR)

Este componente é responsável pelo armazenamento e análise dos resultados, fornecendo o veredicto de conformidade (o sistema se comporta ou não conforme o esperado) e medidas de desempenho e de confiabilidade.

SISTEMA DE GERENCIAMENTO DE OBJETOS (SGO)

O Sistema de Gerenciamento de Objetos (SGO) é responsável pelos serviços de integração de dados, permitindo que as ferramentas possam ter acesso aos dados armazenados em um repositório comum. Esse repositório sendo implementado por um banco de dados que está sendo utilizado.

O SGO também provê serviços de controle de configuração para os dados gerados e utilizados pelas ferramentas, os quais podem ser: uma especificação de protocolo, casos de teste ou de falhas, scripts ou resultados da execução de testes, entre outros.

SISTEMA DE GERENCIAMENTO E ARMAZENAMENTO DOS DADOS (SGAD)

Este componente complementa os serviços de integração de dados que implementa o repositório do ambiente.

O SGAD não é implementado pelo ATIFS, sendo utilizado um sistema de gerenciamento de banco de dados comercial, como *Microsoft Access*, por exemplo.

3.3 APLICAÇÃO PRÁTICA

O ATIFS será usado preliminarmente no teste de uma implementação do protocolo X.25, que está sendo desenvolvida no âmbito do projeto SCOM-X25, da Divisão de Sistemas de Solo (DSS) do Instituto Nacional de Pesquisas Espaciais (INPE).

O projeto SCOM-X25 [MEC93] tem como objetivo possibilitar que equipamentos do tipo IBM-PC possam ter acesso a redes públicas que operam segundo o padrão X.25 da CCITT. Com o objetivo de liberar o máximo a CPU do microcomputador das atividades de comunicação, possibilitando seu maior aproveitamento para processamento de aplicativos, foi desenvolvida uma interface de comunicação (placa PX25) para se encarregar de abrigar o *software* que irá fornecer o protocolo X.25 (SX25). No INPE, a Missão Espacial Completa Brasileira (MECB) prevê a utilização da placa PX25 com o Conjunto de Medidas de Velocidade (CMV) da Estação Terrena de Cuiabá. O CMV mede a velocidade do satélite através do efeito *Doppler*; os pedidos de medida serão iniciados no Centro de Controle de Satélites, em São José dos Campos, e transmitidos ao CMV via Rede de Comunicação de Dados (RECDAS - é uma rede privativa do INPE que oferece serviços de comutação de pacotes). O CMV estará conectado à RECDAS através do SCOM-X25.

3.4 CONCLUSÃO

Foi visto uma descrição do ambiente integrado ATIFS mencionando alguns conceitos importantes e a sua arquitetura.

No próximo capítulo, tem-se uma descrição teórica de interface com o usuário no contexto geral e a orientação a objetos.

Capítulo 4

INTERFACE COM O USUÁRIO E ORIENTAÇÃO A OBJETOS

Este capítulo descreve os conceitos básicos de interface com o usuário, em particular, descreve vários modelos voltados para interface orientada a objetos.

Este capítulo contém três seções: conceitos básicos sobre a interface com o usuário; seguido da apresentação do desenvolvimento de interfaces orientadas a objetos e; finalmente, a conclusão com uma avaliação sobre este desenvolvimento.

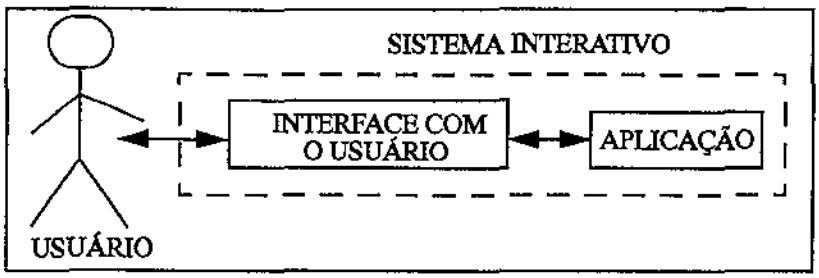
4.1 CONCEITOS BÁSICOS

Um sistema interativo **[Luc93]** é um tipo de sistema de informação que interage com o usuário (humano) possuindo basicamente duas partes: a interface com o usuário e a aplicação como mostra a figura 4.1.

A partir deste ponto, no decorrer desta dissertação, a terminologia "interface com o usuário" será simplesmente mencionada pelo termo "interface" a fim de facilitar a sua leitura.

A interface oferece o meio de comunicação entre o usuário e a aplicação. A aplicação é uma porção do sistema interativo que não interage com o usuário sendo responsável pelas tarefas funcionais do sistema tais como: cálculos matemáticos, interpretação de um formato de arquivos e geração de resultados que foram requisitados pelo usuário.

Figura 4.1 Estrutura geral de um sistema interativo



Uma interface é considerada bem projetada quando esta oferece recursos eficientes e intuitivos facilitando ao usuário na operação do sistema e conseqüentemente, reduzindo o tempo de aprendizado e, ao mesmo tempo, aumentando a sua produtividade.

4.1.1 FACILIDADE DE USO DA INTERFACE

É importante que um sistema ofereça facilidades ao usuário na sua operação. O usuário deve entender o que o sistema está solicitando e, também, com o mínimo de esforço, fazer requisições ao sistema. O sistema, por sua vez, deve responder a estas requisições de forma compreensível, clara e satisfatória. Além disso, o usuário deve saber o que está fazendo no sistema. Assim, o sistema é considerado "amigável" quando se tem esta facilidade de diálogo entre o sistema e o usuário.

Atualmente, segundo **[Mye94]**, na área comercial, a qualidade de interface está se tornando crítica no sucesso dos produtos. Um dos motivos do seu sucesso é permitir que as pessoas não gastem muito tempo lendo manuais.

Isto significa que *facilidade de uso* (ou *usabilidade*) também é um fator importante da interface pois permite que o usuário, nos primeiros contatos com o *software*, sinta nenhuma ou, pelo menos, poucas frustrações.

As interfaces mal projetadas contribuem para muitas frustrações ou até mesmo, para desastres inclusive com perda de vida humana. Por exemplo, segundo **[Mye94]**, um certo número de pessoas morreram por excesso de radiação na Therac-25, sendo que um dos motivos do acidente foi um erro na manipulação de cursor do sistema.

A interface é considerada "muito importante" ou até mesmo essencial para algumas aplicações tais como: pesquisas em geral, biologia computacional, computação na área comercial e bibliotecas eletrônicas. Apesar da sua importância, o custo de desenvolvimento de interface é elevado e também consome bastante tempo. Segundo **[Mye94]**, constatou-se que a interface pode corresponder a mais de 50% do esforço de desenvolvimento de um sistema.

Em resumo, a interface tornou-se um dos fatores predominantes para o sucesso de um sistema. Em muitos casos, um software é adquirido mais pela facilidade de ser entendido e utilizado do que por outras qualidades.

4.1.2 ESTILOS DE INTERAÇÃO

Segundo **[Luc93]** e **[HH89]**, os estilos de interação são as formas pelas quais as pessoas interagem com os computadores. Os principais estilos de interação são:

1. **WYSIWYG** - uma interface do tipo WYSIWYG (*What You See Is What You Get*) garante ao usuário que o resultado que ele vê é realmente o resultado do processamento (saída). Por exemplo, editor de texto WYSIWYG mostra na tela um texto que será impresso; as palavras em itálico aparecem em itálico; vários tamanhos e fontes de letras são diferenciados na tela resultando uma visão real do que será realmente o resultado de uma impressão. Os editores não-WYSIWYG não oferecem esta qualidade, exibindo caracteres de controle.
2. **Linguagem de comandos** - interação convencional em que o usuário simplesmente digita comandos (seqüência de caracteres) como entrada do sistema.
3. **Linguagem natural** - semelhante ao caso anterior. O usuário não está preso a uma sintaxe rígida e vocabulário exíguo. Este estilo oferece uma interação voltada para o diálogo do dia-a-dia. São aplicáveis somente em áreas de conhecimento muito específicas.
4. **Manipulação Direta** - permite que o usuário manipule, geralmente com o uso do mouse, representações da realidade. Por exemplo, o usuário deseja redimensionar a janela, ele simplesmente arrasta a borda da janela com o mouse (clique sobre a borda da janela, com o botão pressionado, movimenta o mouse para o local desejado e libera o botão) sem precisar lembrar de um comando ou um item de menu.
5. **Demonstracional** - estilo no qual o usuário pode usar exemplos, fornecidos através de manipulação direta, para especificar operações abstratas. Para isto, o sistema usa inferências para "sugerir" generalizações. Por exemplo, o usuário novamente arrastou um arquivo com extensão .bak até a uma lata de lixo, então a interface pode perguntar se o usuário deseja excluir todos os arquivos com a extensão .bak.
6. **Icônico** - estilo baseado no uso de ícones. Os ícones são pequenos símbolos gráficos que representam uma analogia a um objeto, ação ou propriedade. Geralmente, o usuário simplesmente seleciona estes símbolos com o mouse.
7. **Menus** - são um conjunto de opções que aparece em coluna ou em círculo (menu *pie*) facilitando o diálogo com o usuário pois reduz a necessidade de memorização e limita o conjunto de opções.
8. ***Form fill-in*** ou ***fill-in-the-blanks*** - este estilo é empregado na entrada de dados que é composta por vários campos de preenchimento permitindo que o usuário navegue para qualquer campo desejado e preencha o campo. A vantagem é que elimina a necessidade de lembrar de todos os argumentos de um determinado comando.

Aqui se tem todos os estilos de interação reconhecidos por **[Luc93]**. Vale lembrar que estes estilos de interação não são exclusivos. Por exemplo, existem interfaces que são do tipo manipulação direta e também do tipo menu.

Aqui se encerra a seção sobre estilos de interação. A seguir, tem-se a seção sobre metodologia para desenvolvimento de interfaces.

4.1.3 METODOLOGIA PARA DESENVOLVIMENTO DE INTERFACES

Como todo software em geral, o desenvolvimento de uma interface pode ser dividido em diferentes etapas, como a seguir **[HH93]**:

1. **Análise do Sistema** - deve-se especificar "o quê" a interface deve fazer, contendo os seguintes itens:

   1.1. objetivo - o objetivo da interface;

   1.2. análise do usuário - descrição da(s) característica(s), da habilidade do(s) usuário(s) e conclusões levantando os requisitos de interface do sistema.

   1.3. análise de tarefas - identifica as tarefas do usuário, define objetos e as ações que manipulam estes objetos.
2. **Projeto da Interação** - define a interface com o qual usuário irá interagir. Esta fase registra todos os aspectos da interface (cores, *layout* e outros) exigindo o mínimo de *background* em áreas como a psicologia e afins, aplicáveis neste contexto.
3. **Projeto de Software da Interface** - esta fase beneficia-se de técnicas e métodos de Engenharia de Software a fim de especificar interface, utilizando metodologia de especificação de sistemas.
4. **Implementação do Software da Interface** - o programador utiliza os recursos oferecidos por ferramentas de construção de interfaces (*toolkits*), bibliotecas gráficas, sistemas de ajuda (*help*), modelos de estrutura de controle de diálogo, UIMS (*User Interface Management Systems*) e outros para implementar o projeto do software da interface.

Estas etapas podem ser realizadas de acordo com as diferentes abordagens existentes para o desenvolvimento de software, dentre elas, a orientação a objetos, que é o enfoque deste estudo.

Vale mencionar que, neste trabalho, o desenvolvimento da interface do ATIFS será baseado nas etapas citadas nesta seção.

Nos itens a seguir, serão apresentados os principais aspectos do desenvolvimento de interface usando o paradigma de orientação a objetos.

4.2 DESENVOLVIMENTO DE INTERFACE ORIENTADA A OBJETOS

A orientação a objetos é vantajosa pois oferece soluções a várias dificuldades no desenvolvimento de interfaces, dentre as quais [LVC89]e [KZ90]:

- provê bom mecanismo de abstração de dados;
- oferece facilidade de extensão através do mecanismo de herança;
- facilita a modularização, o que permite uma separação de interface e aplicação;
- facilita a manutenção;
- facilita a reutilização, tanto de projeto quanto de código.

A construção de interfaces orientadas a objetos já vem sendo usada desde no início da década de 80, existindo um número relativamente grande de trabalhos a respeito. Uma característica da maioria desses trabalhos é que eles se baseiam na noção de "framework", o qual é importante na reutilização tanto de projeto quanto de código. A definição de framework, bem como a apresentação de modelos e alguns frameworks para construção de interfaces são o tema dos itens a seguir.

Apesar destas vantagens, a orientação a objetos pode apresentar a sua limitação. O desempenho das interfaces orientadas a objetos pode ser inferior em relação às interfaces procedurais devido a sua complexidade por causa da herança, hierarquia de classes, polimorfismo e outros recursos que podem comprometer o desempenho.

4.2.1 FRAMEWORKS

Existem várias metodologias usadas no desenvolvimento de interfaces. Algumas se baseiam em frameworks e outras não. O [RPB+91], por exemplo, cita alguns passos do desenvolvimento da interface interativa sem utilizar nenhum framework. Outras metodologias simplemente definem um framework de interface para definir uma interface de um sistema (por exemplo, [Sha91], [Jaa95a], [Jaa95b]).

Este trabalho utilizará um modelo para construção de interface como base na definição de um framework de interface do ATIFS a fim de que este framework possa ser reutilizado pelos projetistas na construção de interface das suas ferramentas interativas.

Uma das vantagens da orientação a objetos, conforme já foi mencionada, é a facilidade de reutilização, tanto de componentes de um programa quanto do projeto. Os frameworks são a base para a construção de projetos reutilizáveis.

4.2.1.1 CONCEITO

Segundo [WJ90], um "framework" pode ser considerado como uma arquitetura de alto nível de abstração, consistindo em classes abstratas, classes concretas e de relacionamentos entre elas.

As classes abstratas são classes genéricas que não podem ser instanciadas. Estas possuem métodos que ainda devem ser definidos. Aplicando-se a um caso específico, as classes abstratas podem ser derivadas obtendo suas subclasses concretas com seus métodos definidos.

As classes concretas são classes que podem ser instanciadas possuindo todos os métodos e atributos definidos.

Johnson e Foote **[JF88]** definem o framework como sendo um projeto abstrato orientado a objetos que consiste em uma classe abstrata para cada componente (parte de um programa) principal. As interfaces entre estes componentes do projeto são definidas em termos de um conjunto de mensagens. Geralmente, há uma biblioteca de subclasses que podem ser usadas como componentes do projeto.

A contribuição de um framework para a reutilização é portanto evidente: mesmo que se necessite criar novas subclasses, estas são fáceis de produzir pois as suas superclasses abstratas definem o seu projeto e também uma grande parte de seu código.

Em geral, os frameworks são dependentes do domínio do problema, pois eles podem ser vistos como uma forma de descrição da solução para uma determinada classe de problemas. Os frameworks podem ser usados em diferentes áreas.

Há uma grande parte dos trabalhos que enfoca sistemas interativos. Alguns destes frameworks foram estudados para a realização deste trabalho, os quais serão apresentados posteriormente. No item a seguir, tem-se uma descrição de construção de frameworks.

4.2.1.2 CONSTRUÇÃO DE FRAMEWORKS

Tem-se três aspectos de desenvolvimento de frameworks **[WJ90]**: projeto, uso e descrição.

Projetar um framework para um sub-sistema é desenvolver um modelo que represente uma generalização de um sub-sistema. Em outras palavras, o framework não somente serve para extrair classes abstratas, mas também deve ser utilizável por diferentes aplicações. Portanto, é difícil projetar um framework pois o projetista deve ter muito conhecimento do domínio do problema e muita experiência de projeto de framework para obter esta generalização de um sub-sistema que permite desenvolver aplicações a partir desta.

A segunda etapa é quanto ao uso de frameworks. Para usar o framework, é preciso criar uma aplicação a partir do modelo que foi projetado. Segundo **[WJ90]**, o uso pode se dar de uma das duas formas:

1. definição de novas classes/subclasses que se façam necessárias, ou;
2. configuração de um conjunto de objetos instanciados a partir das classes existentes, através do fornecimento de parâmetros a cada objeto e conectando-os entre si.

No primeiro caso acima, o comportamento específico da interface da ferramenta seria definido acrescentando-se novos métodos às subclasses derivadas a partir das classes do framework. Cada método acrescentado deve respeitar as convenções da(s) respectiva(s) superclasse(s). Os frameworks definidos para este tipo foram denominados como *caixa-branca* **[JF88]**, pois quem vai utilizá-los é obrigado a conhecer a sua estrutura interna.

Na segunda forma de especialização, o framework recebe um conjunto de parâmetros que representam o comportamento específico de cada ferramenta. Estes parâmetros são fornecidos através da interface de cada classe, esta forma de uso requer que seja conhecida apenas esta interface. Desse modo, este tipo de framework, assim especializado, passou a ser conhecido como framework *caixa-preta*.

A maioria das aplicações necessita da criação de novas subclasses dentro do framework. Após a criação de novas subclasses, o próximo passo do uso do framework é conectar ou configurar os seus objetos.

Há aplicações que são desenvolvidas simplesmente configurando-se um conjunto de objetos passando parâmetros bem definidos. Este conjunto de objetos é criado e cada um deles é inicializado. Finalmente, as operações são executadas pelos objetos a fim de conectá-los.

Estas aplicações são muito similares entre si, de forma que estas possam ser escritas automaticamente. Isto ocorre comumente nos frameworks de interface com o usuário que possuem ferramentas que escrevem automaticamente um código para configurar a interface.

A terceira etapa de desenvolvimento de frameworks é a descrição. Esta descrição deve permitir não somente representar classes que ele contém e a interação entre elas, mas também as instruções relativas à criação de novas subclasses e à configuração de aplicações.

As dificuldades da descrição de framework levantadas podem ser resumidas nos seguintes pontos:

1. existência de poucas metodologias de análise e projeto que fornecem representações para frameworks, e;
2. existência de poucas linguagens que ofereçam comandos para configuração de sub-sistemas.

Apesar destas dificuldades, foram desenvolvidas diferentes técnicas para documentar frameworks, denominadas de *padrões de projeto* [1] **[Pre95b]**. O capítulo 7 utiliza uma técnica definida em **[Pre95b]**, para a descrição do framework da interface do ATIFS.

1. do inglês "*design patterns*"

A área de interface é aquela para a qual foram dirigidos grande parte dos estudos sobre frameworks. Há diversos frameworks que são baseados em modelos orientados a objetos para desenvolvimento de interfaces. Alguns deles são apresentados nos itens que se seguem.

Dentre os modelos estudados, foi escolhido aquele que melhor se adeqüe às necessidades do ATIFS. Com base no modelo escolhido, foi projetado um framework específico para a interface do ATIFS.

4.2.2 MODELOS PARA DESENVOLVIMENTO DE INTERFACES

Dentre os vários modelos de interface existentes, foram considerados, neste estudo, os seguintes: MVC, Modelo baseado em Modo, MVC++, Modelo em Camadas e MV. Eles são apresentados resumidamente a seguir.

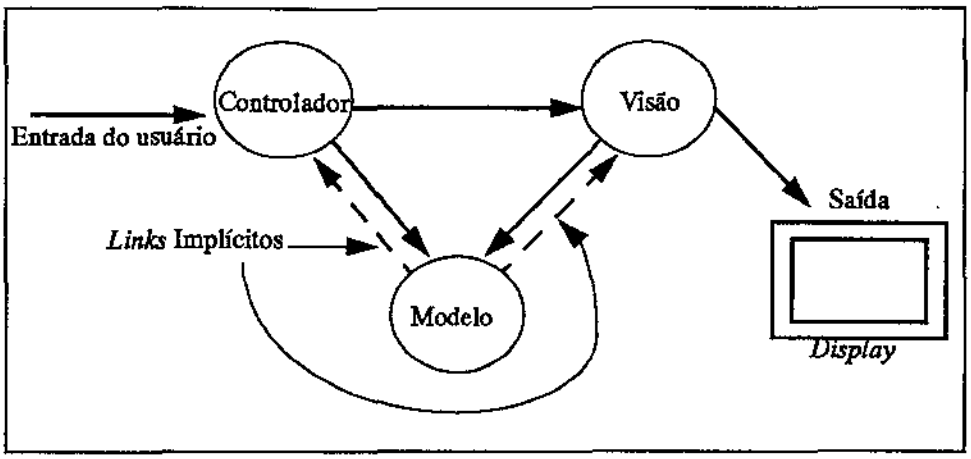
4.2.2.1 MODELO-VISÃO-CONTROLADOR (MVC)

Este modelo compõe-se de três componentes (figura 4.2): Modelo, Visão e Controlador **[BC91],[Sha91],[Jaa95a]**. Cada componente possui sua função específica:

1. *Controlador* - recebe entradas do usuário e realiza as ações correspondentes, seja acionando a Visão ou Modelo. O controlador também escalona interações com outros controladores.
2. *Visão* - controla os aspectos visuais, exibindo na tela os dados do Modelo;
3. *Modelo* - contém e/ou gera as informações a serem exibidas para o usuário; também consome as informações que este gera. Representa uma aplicação específica.

O funcionamento padrão é: o usuário gera ações de entrada e o Controlador ativo responde a estas ações invocando ações no Modelo. O Modelo, por sua vez, executa as operações requisitadas, possivelmente mudando o seu estado e retornando um "sinal" para todas as visões dependentes (através de *links* implícitos ilustrados na figura 4.2) informando que o estado mudou. Cada Visão pode consultar o novo estado do Modelo e atualizar a tela, se necessário.

Figura 4.2 Estrutura do MVC



Se o Controlador recebe uma entrada do usuário cujo contexto é totalmente voltado para aspectos de interface então este invocará diretamente o componente Visão (sem invocar o componente Modelo). Dependendo da situação, a Visão pode invocar o Modelo para atualizar algum dado relevante.

4.2.2.2 MODELO BASEADO EM MODO

Este modelo **[Sha91]** é baseado no MVC e projetado para interface de manipulação direta.

Em aplicações convencionais, *Modo* é um estado do sistema. Por exemplo, em um editor de texto, o usuário pode mudar para o modo de inserção para que ele possa inserir caracteres no texto. O usuário pode chavear de um modo para outro (geralmente isto é feito usando uma combinação de teclas) como, por exemplo, no caso do editor de texto, o usuário pode chavear do modo de inserção para o modo de deleção quando ele deseja excluir caracteres no texto.

Foi constatado que esta abordagem traz dificuldades para os usuários sejam eles principiantes ou experientes. Os principiantes têm dificuldades em determinar como chavear de um modo para outro; também fica difícil para o usuário lembrar do modo que se encontra no momento.

Por outro lado os experientes sempre procuravam estratégias de operação de chaveamento de modos, a fim de facilitar o seu trabalho.

Portanto, no conceito convencional de modos explicado anteriormente não é considerado adequado para uma interface amigável, então **[Sha91]** introduziu um novo conceito de modo.

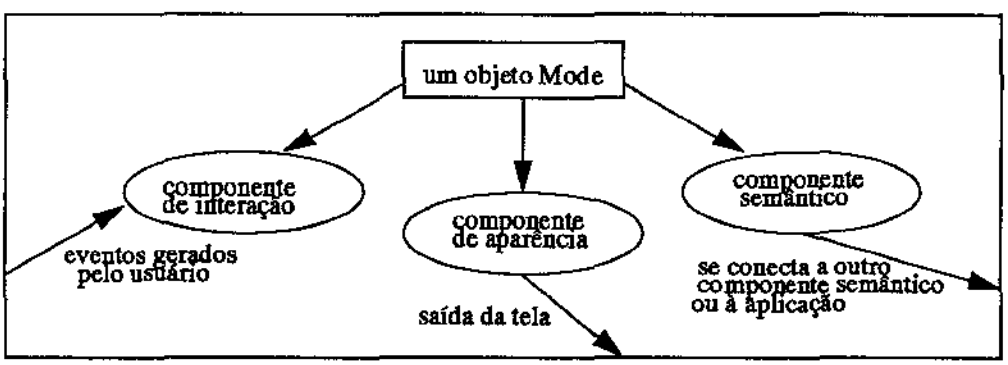
Os *modos* são áreas na tela que interagem com o usuário, são sempre bem visíveis e se entra e sai de um modo simplesmente movendo-se o cursor para dentro e fora da área, respectivamente. Logo, a operação de chaveamento de modos torna-se mais fácil em relação aos modos das aplicações convencionais.

As operações permitidas de um determinado *modo* são apresentadas (de forma não-ambígua) por menus e quadros de diálogos que podem ser invocados por ações genéricas e simples (por exemplo, um clique no botão exibido na tela).

Seguindo esta idéia, um *modelo baseado em modos* pode ser definido para descrever e construir uma interface **[Sha91]**. O bloco de construção básico do modelo baseado em modos é chamado de *Mode*. Um *Mode* é composto por três objetos como ilustra a figura 4.3:

1. *componente de interação* - objeto que recebe eventos do usuário;
2. *componente de aparência* - objeto responsável pela exibição de dados na tela;
3. *componente semântico* - é o objeto conectado à aplicação ou a um outro objeto semântico.

Figura 4.3 Estrutura de um objeto Mode



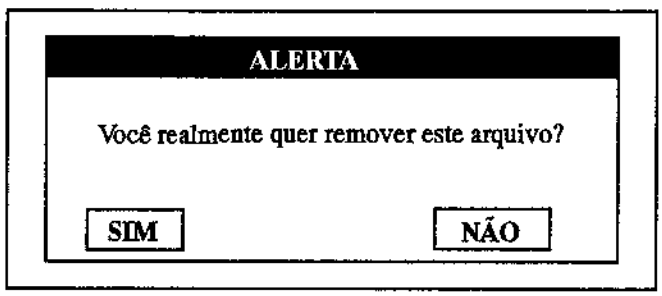
A classe *Mode* implementa uma estrutura de modos de forma hierárquica. Um modo pode conter outros modos chamados submodos. Qualquer modo é um submodo de um único modo denominado de "supermodo".

Para exemplificar, a figura 4.4 ilustra um quadro de diálogo como sendo um modo com dois submodos: um submodo "SIM" e outro "NÃO". Estes dois submodos representam os botões SIM e NÃO que se destacam quando o botão esquerdo do mouse é pressionado sobre eles, e voltam ao seu estado normal, quando o botão do mouse é liberado ou quando o cursor sai da sua área.

Este comportamento dos botões é diferente do seu supermodo (o quadro de diálogo ALERTA) que não responde da mesma forma aos seus botões SIM e NÃO.

Observe que o texto: "Você realmente quer remover este arquivo?" na figura 4.4, não é considerado como modo. Apesar deste afetar a aparência do quadro de diálogo, este não interage com o usuário.

Figura 4.4 Caixa de diálogo de um modo com dois submodos



O botão "SIM" tem a seguinte definição:

1. *Aparência* - fundo branco com borda preta de largura 1 e com uma string "SIM" centralizada. Aparência destacada é o inverso da aparência normal.
2. *Semântica* - confirma a remoção do arquivo.
3. *Interação* - destaca-se quando o botão esquerdo do mouse é pressionado dentro da sua área; e volta ao seu estado original quando o cursor sai ou quando o botão do mouse é liberado.

Observe que o submodo "NÃO" tem a mesma interação. A diferença está na *semântica* e *aparência*.

O modelo tem cinco classes que são:

1. *Mode* - coordena as atividades voltadas para a interação tais como: despacho de eventos, gerenciamento de janelas e exibição de objetos.
2. *MController* - determina o comportamento da interface executando os *componentes de interação* do *Mode*, determina qual é a resposta que deve ser feita após o recebimento de um evento do *componente de interação* do *Mode*. Determinada a resposta, o *MController* solicita ao *componente de interação* que execute as ações necessárias para fornecer essa resposta.
3. *SemanticObject* - representa a interação entre interface com o usuário e aplicação. Portanto, há informações semânticas referentes tanto ao domínio da interface quanto ao da aplicação são mantidas nesta classe.
4. *MDisplayObject* - controla em "background" os objetos de exibição, controla as cores, bordas e outras propriedades destes tipos de objetos.
5. *Application* - a aplicação propriamente dita.

O objeto *Mode* recebe eventos através do seu *componente de interação* e este requisita o processamento de um evento ao *MController*. O *MController*, por sua vez, consulta uma tabela chamada *eventResponses* para determinar a resposta ao evento recebido. Se a resposta deve ser realizada somente pela aplicação, então se envia uma mensagem a um *SemanticObject* que este, por sua vez, envia mensagens à aplicação.

A aplicação pode solicitar atualização de tela enviando uma mensagem ao componente semântico do *Mode* (somente via *SemanticObject*).

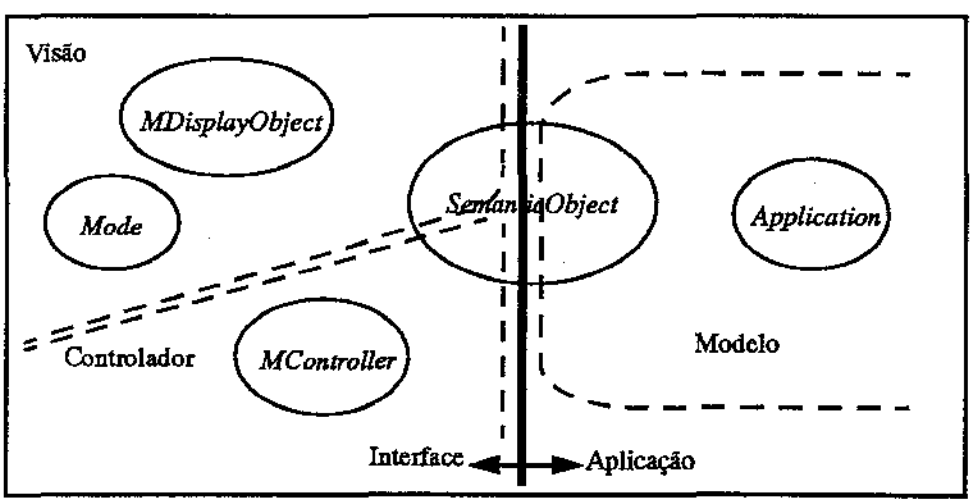
Se a resposta ao evento gerado pelo usuário deve ser feita pela própria interface, então o *MController* invoca método(s) público(s) do seu *componente de aparência* da classe *Mode*.

A classe *MDisplayObject* somente recebe mensagens referentes a cores, dados sobre a borda da janela e outras propriedades dos objetos de exibição.

O modelo baseado em Modo é uma extensão do MVC; a relação dos seus componentes com os do MVC é mostrada na figura 4.5.

Uma outra diferença com relação ao MVC é que neste último o Controlador é quem recebe os eventos do usuário, enquanto, no modelo baseado em Modos, é o *componente de interação* do modo quem recebe estes eventos.

Figura 4.5 Modelo baseado em Modo

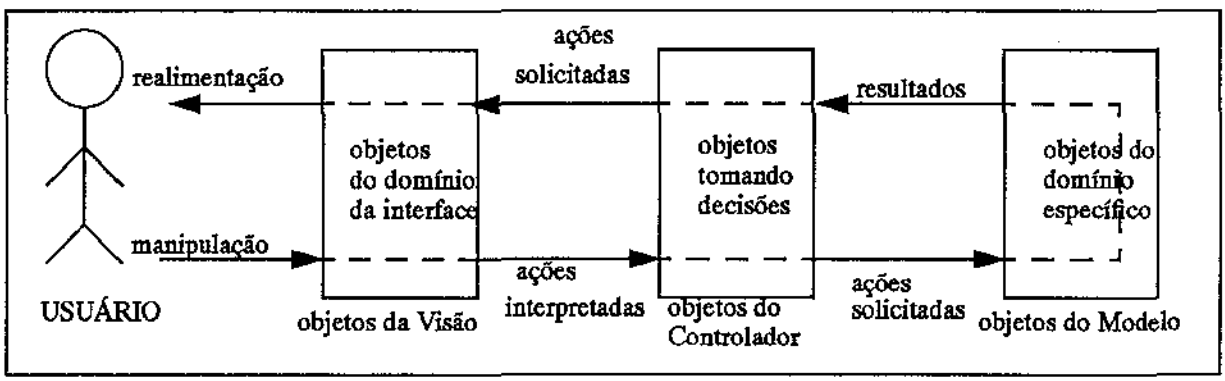


A aparência de um *Mode* é mudada substituindo seu *componente de aparência* por um outro diferente. Isto também ocorre na mudança de semântica e interação de um modo. Por exemplo, um modo que se destaca, tem dois objetos de aparência diferentes: um representa o estado normal e outro, estado destacado. Quando um modo deve ser destacado, substitui-se o objeto de característica normal pelo destacado. Quando o modo deve voltar ao estado normal, o objeto de característica normal é restaurado.

4.2.2.3 *MVC++*

O MVC++**[Jaa95a],[Jaa95b]** é uma variante do MVC original contendo três componentes: objetos do Modelo, objetos do Controlador e objetos da Visão como ilustra a figura 4.6.

Figura 4.6 O modelo MVC++



Os objetos do Modelo são gerenciados pelos objetos do Controlador e não "acionam" os objetos da Visão, ao contrário do MVC.

Os objetos da Visão contêm somente componentes de interface (estes objetos são quem interagem com o usuário) e funções de call-back. Estes objetos têm conhecimento de como exibir informações ao usuário e como receber as ações do usuário, mas não decidem como responder a estas ações. Para isso é acionado o Controlador.

Os objetos do Controlador tomam todas as decisões necessárias nas quais são específicas da aplicação contendo todo o conhecimento de como a aplicação deve trabalhar. Portanto, os objetos do Controlador integram os objetos da Visão e Modelo.

O Modelo realiza as ações referentes à aplicação, ignorando totalmente a existência da Visão.

Na figura 4.6, tem-se as entradas e as saídas chamadas de *manipulação* e *realimentação*, respectivamente. A manipulação representa *métodos de manipulação* sendo invocados, os quais são responsáveis pela recepção das ações do usuário.

A realimentação representa os *métodos de realimentação* sendo invocados, estes são responsáveis pela exibição de certos dados para o usuário final.

Existem também os *métodos de consulta* que servem para que os objetos do Controlador investiguem o estado da Visão.

4.2.2.4 *MODELO EM CAMADAS*

Este modelo foi introduzido pelo **[LB91]** que se compõe de 4 (quatro) camadas. A figura 4.7 representa camadas como um conjunto de objetos e as setas representam as mensagens.

A camada 1, de *eventos físicos* (de baixo nível), recebe eventos primitivos do sistema de janelas tais como: botão esquerdo do mouse pressionado e tecla "k" liberada.

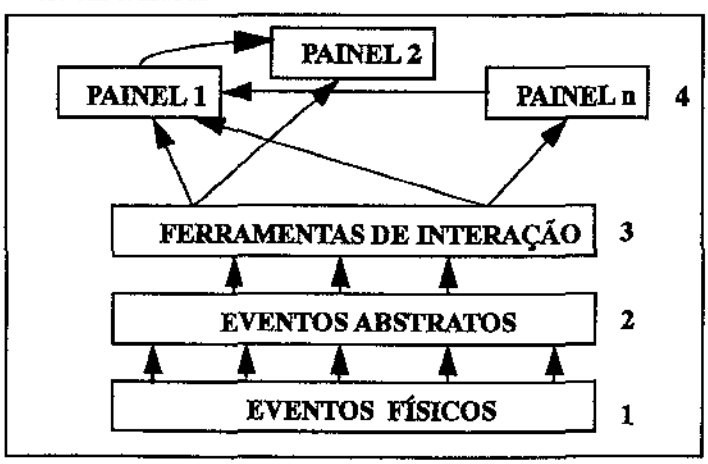
A segunda camada (*eventos abstratos*) trata-se da composição de eventos físicos da camada 1, formando um evento abstrato. Por exemplo, pressionar e liberar o botão esquerdo do mouse (que são dois eventos físicos distintos) são compostos formando um evento abstrato chamado clique.

A terceira camada é composta por *objetos de interação* tais como: menus, barra de rolagem, botões; e, cada objeto possui, dentre outras partes, uma parte chamada *estilo de interação* contendo gerenciadores de eventos (abstratos) relevantes a este objeto. Por exemplo, o objeto menu mantém os gerenciadores de eventos: clique, arraste de mouse e outros eventos relevantes. Conclui-se que esta camada faz um papel de *toolkit* dispondo de serviços fornecidos pelos objetos básicos de interação à camada superior.

A quarta camada representa *Painéis de Interação*. Estes painéis compõem-se de ferramentas de interação e painéis. A figura mostra que os painéis recebem mensagens não só da camada 3 mas também da própria camada 4 (de *painéis abstratos*)

Os painéis podem ser organizados hierarquicamente para representar uma interface alvo. Esta hierarquia é representada por uma árvore cuja raíz é um painel representando uma janela inteira ou tela da estação de trabalho. As folhas são as ferramentas básicas de interação. Os painéis contêm (sub-)painéis e estes podem ser adicionados por ferramentas de interação para reutilização.

Figura 4.7 Modelo em camadas



A figura 4.7 não representa a aplicação. Os autores mencionam que estes painéis interagem com a aplicação mas não explicitam como é feita esta interação.

Em seguida, tem-se uma descrição do último modelo considerado neste trabalho que é o MV (Modelo-Visão).

4.2.2.5 MODELO-VISÃO (MV)

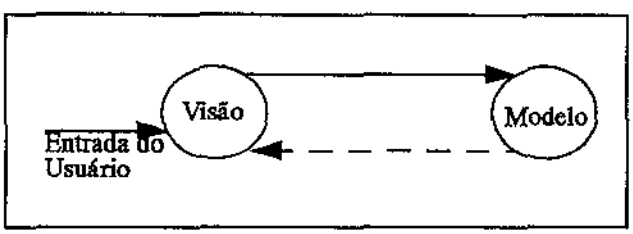
O MV **[Col95]** é uma variante do MVC, voltado para interfaces de manipulação direta. Dado que, neste tipo de interface, o comportamento dos seus objetos é mais complexo, fica difícil centralizar todo o comportamento dinâmico em um único Controlador. Assim sendo, cada objeto da Visão já tem embutido o seu próprio Controlador.

Portanto, no componente Visão, tem-se tanto a aparência quanto o comportamento definido em cada objeto de interface. Já o componente Modelo é idêntico ao do MVC, este representa a aplicação.

A Visão interage com o Modelo através de um fluxo de mensagens representado pela seta e retorno de dados, pela seta tracejada na figura 4.8.

Note que o fluxo de mensagens é unidirecional (sempre da Visão para o Modelo). Isto significa que o componente Visão é quem deve invocar o componente Modelo e nunca no sentido inverso.

Figura 4.8 Modelo do MV.



A entrada do usuário é obtida pelo componente Visão; este, por sua vez, avalia o evento e determina a sua resposta. Se a resposta deve ser feita pelo Modelo, então lhe envia uma mensagem. Se a resposta deve ser realizada pela própria Visão, então ela mesma responde ao usuário.

Após o estudo de modelos para construção de interfaces, tem-se alguns exemplos de frameworks para desenvolvimento de interfaces considerados neste trabalho.

4.2.3 EXEMPLOS DE FRAMEWORKS PARA CONSTRUÇÃO DE INTERFACES

Nesta seção, serão apresentados dois frameworks para construção de interface com o usuário para suas aplicações específicas, os frameworks são: MacApp e ET++.

4.2.3.1 *MacApp*

O MacApp **[Sch86]** é um framework (baseado no modelo MV, segundo **[Col95]**) para implementar interfaces conforme o padrão *Macintosh*. Contém menus, janelas que rolam e que podem ser movidas na tela. Neste framework, não foram definidos os conteúdos da janela e itens de menu.

Quanto a sua estrutura, ele é formado por mais de 30 (trinta) classes contendo mais de 450 (quatrocentos e cinqüenta) métodos. Devido a grande quantidade de classes e métodos, tem-se uma descrição sucinta somente das classes mais importantes que são: *TApplication, TDocument* e *TView*.

A classe *TApplication* representa uma aplicação como um todo incluindo disparo da aplicação, configuração de barra de menu e mantém tipos de documentos que a aplicação suporta.

A aplicação deve ser representada como uma subclasse do *TApplication* sobrepondo quaisquer métodos desta superclasse para definir o contexto específico da aplicação.

Vale lembrar que, após a inicialização da aplicação, a classe *TApplication* não processa e nem responde a eventos do usuário pois este possui somente atributos e serviços referentes à aplicação como um todo, informações suficientes a fim de inicializar a aplicação.

A classe *TDocument* processa comandos para salvar arquivos e fechar uma aplicação as quais são específicas para cada documento.

A classe *TView* se encarrega dos aspectos referentes à apresentação tais como: desenhar imagens, destacar uma seleção, manipulação de imagens através do mouse e outras operações disponíveis. O *TView* tem conhecimento de quando há necessidade de redesenhar e também de quando a seleção deve ser destacada. Esta classe não tem conhecimento de como executar estas operações. Portanto, o projetista deve sobrepor métodos definindo estes comportamentos na sua subclasse do *TView*.

4.2.3.2 *ET++*

A arquitetura do ET++ **[WGM88]** é baseada no MacApp e integra uma rica coleção de blocos de construção de interfaces bem como uma estrutura básica de dados em um sistema homogêneo e extensível.

Este framework foi projetado para UNIX seguindo as rigorosas funcionalidades do MacApp e dos elementos de interface com o usuário na caixa de ferramentas do Macintosh.

O objetivo do ET++ é integrar uma estrutura híbrida do MacApp e uma caixa de ferramenta do Macintosh (não orientada a objetos) em um único framework orientado a objetos. Como uma das desvantagens do MacApp, ao desenvolver uma aplicação complexa, o **[WGM88]** constata a necessidade do conhecimento da estrutura interna do MacApp e também da sua caixa de ferramenta.

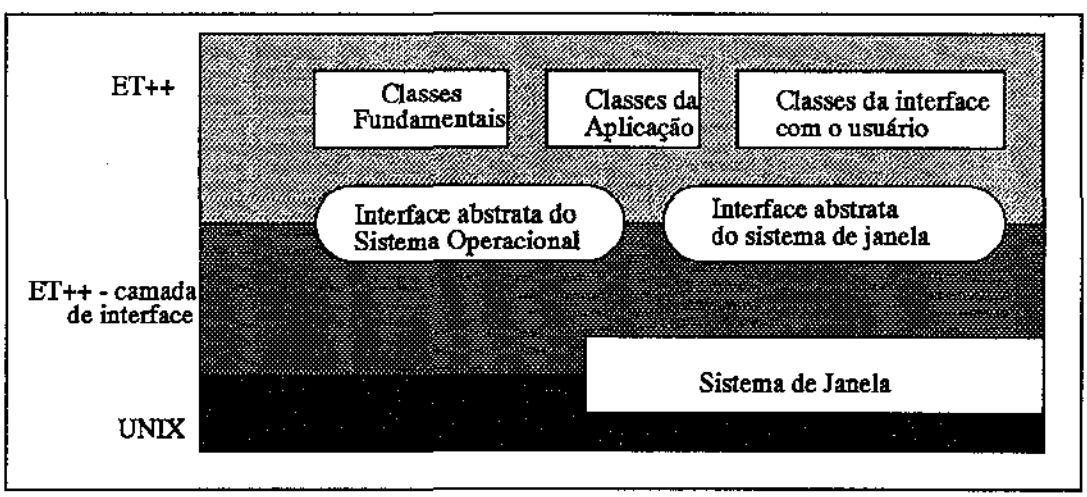
O ET++ foi projetado para resolver alguns destes problemas do MacApp inclusive a portabilidade pois o MacApp não oferece esta facilidade.

Quanto a sua arquitetura, tem-se basicamente três camadas: *UNIX, camada de interface* e *ET++* como mostra a figura 4.9.

As *classes fundamentais* representam as estruturas básicas (vetores, listas, conjuntos, etc.). As *classes de interface com o usuário* implementam todos os elementos gráficos encontrados em quase todas as ferramentas que são: menus, quadros de diálogos ou barras de rolamento[1]. As *classes da aplicação* são classes de alto nível que representam uma estrutura de controle comum das aplicações em um ambiente gráfico altamente sofisticado.

A *camada de interface* oferece um pequeno conjunto de funções para o gerenciamento de janelas, manipulação de entradas e desenhos em vários dispositivos.

Figura 4.9 Arquitetura do ET++



As classes básicas da aplicação são derivadas do MacApp e, portanto, têm um comportamento e interfaces similares. Estas classes são: *Application, Document, View, ViewFrame, Window* e *Command*. Estes fornecem métodos para gerenciamento de documentos, janelas e comandos do usuário.

1. do inglês *scrollbars*

Após o estudo sobre estes modelos e levantamento de trabalhos de frameworks para construção de interfaces, em seguida, tem-se uma avaliação destes modelos estudados, com intuito de determinar aquele a ser usado neste trabalho. Posteriormente, com base no modelo escolhido, será projetado um framework para desenvolver a interface do ATIFS que será visto a partir do capítulo 6 desta dissertação.

4.3 CONCLUSÃO

Inicialmente, foi visto um estudo teórico sobre interface, seguida de uma apresentação dos aspectos de construção de frameworks de interface.

Posteriormente, foi apresentado um estudo sobre o desenvolvimento de interfaces orientadas a objetos mostrando a importância do uso deste paradigma, permitindo a reutilização de projeto e de código, através do uso de frameworks.

Os frameworks podem ser projetados com base em um modelo para construção de interfaces. Foram vistos alguns modelos de interface (MVC, Framework baseado em Modo, MVC++, Modelo em Camadas e MV).

O leitor que deseja estender o seu estudo sobre modelos de interface, há alguns que não foram apresentados neste trabalho, tais como: APV **[KZ90]**, PAC **[BC91]**.

Há também alguns frameworks para desenvolvimento de interface que não foram mostrados neste trabalho, por exemplo, o *Interviews* para *X11 Window System* **[Ege91]**,**[LVC89]**.

Com base nos modelos apresentados, decidiu-se que o framework a ser seguido é o MVC++, pelas seguintes razões: (1) o alto grau de reutilização e (2) separação de componentes Visão e Modelo.

O MVC++ permite que o Modelo não tenha conhecimento da Visão e vice-versa pois não há uma interação direta entre estes componentes. Isto não ocorre no MVC original e MV. Assim, se o Modelo é alterado, somente o Controlador precisa ser alterado, evitando se assim uma modificação do componente Visão.

A outra vantagem do MVC++ é a maior facilidade de reutilização pois somente o Controlador mantém instruções de decisão (o que é considerado de difícil reutilização) permitindo que a Visão e o Modelo sejam bastante reutilizáveis.

Isto não ocorre, por exemplo, no MV, pois as instruções de decisão (são instruções que geralmente são predominadas por uma série de condições tratando-se de gerenciar ou controlar algum objeto) permanecem no componente Visão dificultando a sua reutilização.

O Modelo em Camadas apresenta uma estrutura propícia de atender o aspecto de integração de apresentação, conforme foi mostrado na seção 2.1.2. No entanto, este modelo não pode ser adotado pelo ATIFS pois foi visto que não se tem uma descrição detalhada de como é feita a interação entre a aplicação e a interface.

Já o Modelo baseado em Modos possui o mesmo problema do MVC original. Há um forte acomplamento da Visão e o Controlador, o último é responsável pelo comportamento da apresentação da interface. Isto não ocorre no MV e nem MVC++ pois o comportamento da apresentação é definido pelo componente Visão.

A seguir, tem-se a segunda parte desta dissertação que trata de desenvolvimento e aplicação de um framework de interface para o ATIFS.

PARTE II - DESENVOLVIMENTO E APLICAÇÃO DE UM FRAMEWORK DE INTERFACE PARA O ATIFS

Esta parte apresenta o desenvolvimento de um framework específico para a interface do ATIFS com base no MVC++ descrito na seção 4.2.2.3.

Inicialmente, para desenvolver um framework para este ambiente, tem-se uma descrição do protótipo de interface com o usuário do ATIFS. Este protótipo foi desenvolvido no contexto genérico e não para uma ferramenta específica que compõe o ATIFS.

Em seguida, tem-se uma descrição de projeto de framework de interface do ATIFS.

Para mostrar que o framework é adequado para o referido ambiente, tem-se uma descrição de como usar o framework juntamente com exemplos.

Finalmente, esta parte se encerra com a conclusão da dissertação.

Capítulo 5

UM PROTÓTIPO DE INTERFACE COM O USUÁRIO

Este capítulo apresenta o protótipo da interface do ATIFS.

Este capítulo está estruturado da seguinte forma: na primeira seção, tem-se uma introdutória. Na segunda seção, tem-se a especificação do protótipo da interface do ATIFS na fase de análise de sistemas. Na terceira seção, tem-se o projeto da interação (projeto voltado para a apresentação da interface). Finalmente, tem-se a conclusão deste capítulo.

5.1 INTRODUÇÃO

Vale lembrar que, neste capítulo, o termo **apresentação** é considerado como sinônimo de *layout* e também de *aparência* de interface.

Inicialmente, tem-se um estudo voltado para a interface do ATIFS. É preciso seguir a metodologia de desenvolvimento de interfaces que foi mencionada na seção 4.1.3, para especificar a interface durante as fases de análise do sistema e projeto da interação (esta fase é praticamente baseada em protótipos a fim de obter uma interface adequada para o usuário final).

Portanto, neste capítulo, tem-se esta especificação de interface nas fases de análise e projeto da interação.

5.2 ANÁLISE DO SISTEMA

Nesta fase, tem-se uma descrição da interface do ATIFS realizada na fase de análise de sistemas. Primeiramente, é preciso especificar o objetivo, fazer a análise do usuário (descrição do usuário) e, finalmente, desenvolver uma análise de tarefas da interface do ATIFS (ver detalhes na seção 3.2.1).

5.2.1 OBJETIVO

A interface do ATIFS deve fornecer um único ponto de entrada para que o usuário realize as funcionalidades do ATIFS, a interface é a única alternativa do usuário para interagir com o sistema. Assim, o usuário deve operar os objetos de interação contidos na interface a fim de que seja exibido exatamente o que o usuário solicitou.

5.2.2 ANÁLISE DO USUÁRIO

Todos os usuários do ATIFS têm conhecimento de computação pois operam máquinas utilizando ambientes gráficos como, por exemplo, Microsoft Windows na plataforma IBM-PC.

Foi constatado que existem dois tipos de usuários: os esporádicos (usuários que usam esporadicamente o sistema) e permanentes (usuários que utilizam permanentemente o sistema). Portanto, a interface deve fornecer facilidades para ambos os tipos de usuários.

Foi constatado que a *usabilidade* (ou facilidade do uso) é um item muito importante para a interface do ATIFS. Isto significa que a interface deve ser amigável e que tenha uma facilidade de aprendizado para que os usuários não percam muito tempo lendo manuais de instruções.

5.2.3 ANÁLISE DE TAREFAS

Aqui, nesta seção, tem-se um levantamento de tarefas que a interface deve conter. Inicialmente, tem-se o primeiro nível de abstração de tarefas (são tarefas que estão no contexto genérico não especificando detalhes) e logo, em seguida, tem-se dois níveis adicionais de maior detalhamento destas tarefas.

5.2.3.1 NÍVEL 1

As tarefas estão ainda no alto nível de abstração mas estas serão úteis para desenvolver os níveis mais detalhados. Portanto, a seguir, tem-se uma lista de tarefas de nível 1, que são:

1 - Fornecer o projeto

2 - Fornecer especificações

3 - Solicitar geração de testes

4 - Solicitar execução dos testes

5 - Solicitar resultados

5.2.3.2 NÍVEL 2

As tarefas no nível 1 são "explodidas" para visualizar os seus detalhes, como, por exemplo, a tarefa "Fornecer especificação" do nível 1 gerou 4(quatro) novas sub-tarefas: "Fornecer especificação de controle", "Fornecer especificação de dados", "Fornecer especificação de falhas" e "Fornecer aplicação".

As tarefas do nível 2 da interface do ATIFS são:

1- Fornecer o projeto

2.1 - Fornecer especificação de controle

2.2 - Fornecer especificação de dados

2.3 - Fornecer especificação de falhas

2.4 - Fornecer a aplicação

3.1 - Solicitar a geração de testes abstratos

3.2 - Solicitar a geração de dados de teste

3.3 - Solicitar a geração de lista de falhas

3.4 - Solicitar a geração de uma aplicação instrumentada

3.5 - Solicitar a geração de script

4 - Solicitar a execução de script

5.1 - Solicitar dados que foram coletados após o teste

5.2 - Solicitar o histórico de testes

5.3 - Solicitar as medidas desejadas

Note que alguns itens não foram "explodidos", não foi necessário apresentar maior nível de detalhe.

5.2.3.3 NÍVEL 3

Seguindo o mesmo raciocínio do nível 2, tem-se as tarefas do nível 3 que já estão descritos de forma bastante detalhada. Estas tarefas são:

1.1 - Criar novo projeto

1.2 - Abrir projeto existente

1.3 - Excluir projeto existente

2.1.1 - Criar uma nova especificação de controle

2.1.2 - Editar especificação de controle existente

2.1.3 - Imprimir especificação de controle

2.1.4 - Excluir especificação de controle

2.2.1 - Criar uma nova especificação de dados

2.2.2 - Editar especificação de dados existente

2.2.3 - Imprimir especificação de dados

2.2.4 - Excluir especificação de dados
2.3.1 - Criar uma nova especificação de falhas
2.3.2 - Editar especificação de falhas existente
2.3.3 - Imprimir especificação de falhas
2.3.4 - Excluir especificação de falhas
2.4.1 - Fornecer aplicação
2.4.2 - Fornecer dados sobre a aplicação (p.ex. versão da aplicação)
3.1.1 - Fornecer dados sobre geração de testes abstratos
3.1.2 - Solicitar geração de testes abstratos
3.2.1 - Fornecer dados sobre geração de dados de teste
3.2.2 - Solicitar geração de dados de teste
3.3.1 - Fornecer dados sobre geração de lista de falhas
3.3.2 - Solicitar geração de lista de falhas
3.4.1 - Fornecer dados sobre geração de uma aplicação instrumentada
3.4.2 - Solicitar geração de aplicação instrumentada
3.5 - Solicitar a geração de script
4 - Solicitar a execução de script
5.1.1 - Solicitar dados que foram coletados após o teste
5.1.2 - Solicitar veredito
5.2 - Solicitar o histórico de testes
5.3 - Solicitar as medidas desejadas (p.ex. histograma)

Não é preciso detalhar ainda mais estas tarefas pois note que, se estas tarefas forem ainda mais detalhadas, o seu contexto será voltado para o nível de objetos de interação da interface.

Portanto, aqui se encerra a fase de análise da interface. A próxima fase diz respeito ao projeto da interação que será descrito na próxima seção.

5.3 PROJETO DA INTERAÇÃO

Nesta seção, tem-se dois itens. O primeiro é a parte introdutória e o segundo se trata da descrição do protótipo de interface do ATIFS.

5.3.1 INTRODUÇÃO

Nesta seção, é apresentado o projeto da interface do ATIFS. Nesta fase, é desenvolvido um protótipo, o qual é importante para definir os aspectos de apresentação e comportamento da interface ("look and feel").

Cumpre notar que nesta fase, de acordo com o modelo de ciclo de vida para interface apresentado na seção 4.1.3., não é mostrado nenhum aspecto de como implementar a interface, tal sendo tratado na fase de Projeto de Software da Interface, mostrado no próximo capítulo.

No caso do ATIFS, foi constatado que o requisito de interface de manipulação direta (visto na seção 2.2.6) deve ser descartado, pelas seguinte razões:

- este ambiente exige um alto grau de processamento por parte da aplicação. No entanto, uma interface de manipulação direta afetaria negativamente o desempenho de processamento do ATIFS.
- o ATIFS não requer uma apresentação gráfica sofisticada com alto grau de *feedback* (resposta da interface). Portanto, o requisito de manipulação direta não é necessário.
- para atender às necessidades de interface do ATIFS, vistas na seção 3.2.1, não é necessário que esta interface seja do tipo manipulação direta, apesar de que, atualmente, este tipo de interface é considerado bastante aceitável pela maioria dos usuários.

Com base na análise de tarefas e em alguns dados levantados durante a fase de análise, é possível desenvolver várias versões de protótipos até que obtenha uma versão que deve permitir que o usuário do ATIFS realize todas as tarefas levantadas na fase de análise de sistemas com facilidade. Portanto, o requisito de *usabilidade* (ou facilidade de uso) é de maior relevância nesta fase de projeto.

Vale lembrar que o protótipo de interface oferece uma outra utilidade. Este auxilia o levantamento de padrões de apresentação e comportamento de interface (*look and feel*). Assim, é possível obter uma uniformidade da interface do ambiente. Esta abordagem será vista posteriormente.

Na próxima seção, será apresentada a descrição da última versão do prototipo desenvolvido neste trabalho.

5.3.2 APRESENTAÇÃO DA INTERFACE DO ATIFS

Nesta seção, tem-se uma descrição do protótipo da interface do ATIFS no ponto de vista do usuário.

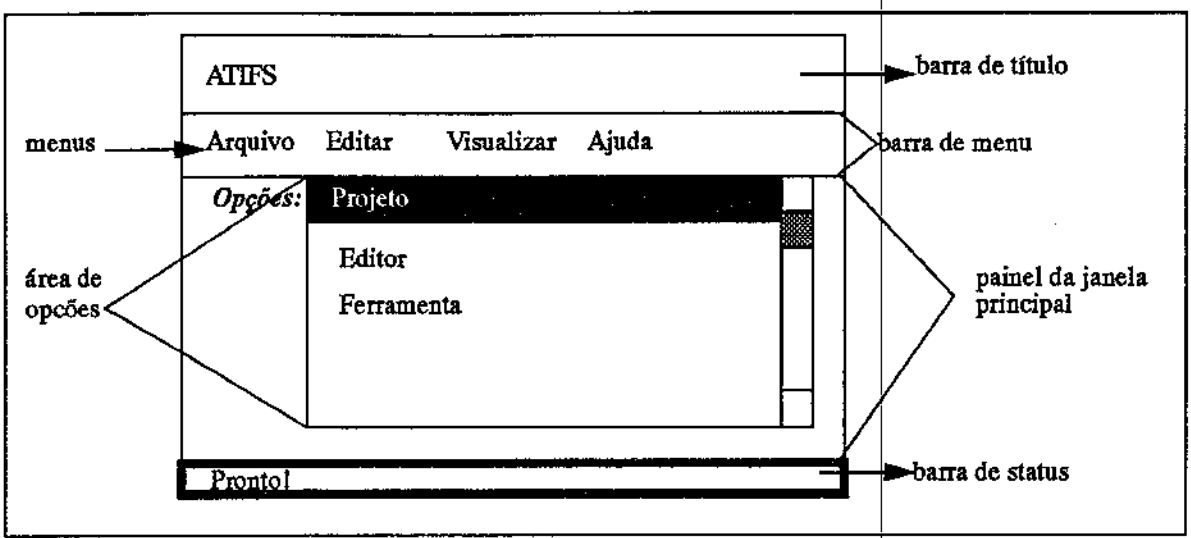
Vale lembrar que o protótipo será descrito de forma genérica e, não específica a cada ferramenta que compõe o ambiente.

Primeiramente, ao executar o ATIFS, o usuário visualiza a sua janela principal que será descrita logo em seguida.

5.3.2.1 JANELA PRINCIPAL

A *Janela Principal* (na figura 5.1) é uma janela do ambiente ATIFS contendo, na sua barra de menu, os menus e, no seu painel, uma área de opções.

Figura 5.1 Janela principal do ATIFS



ÁREA DE OPÇÕES

A área de opções contém uma lista de opções permitindo que o usuário selecione o tipo de dado sobre o qual ele deseja operar.

BARRA DE STATUS

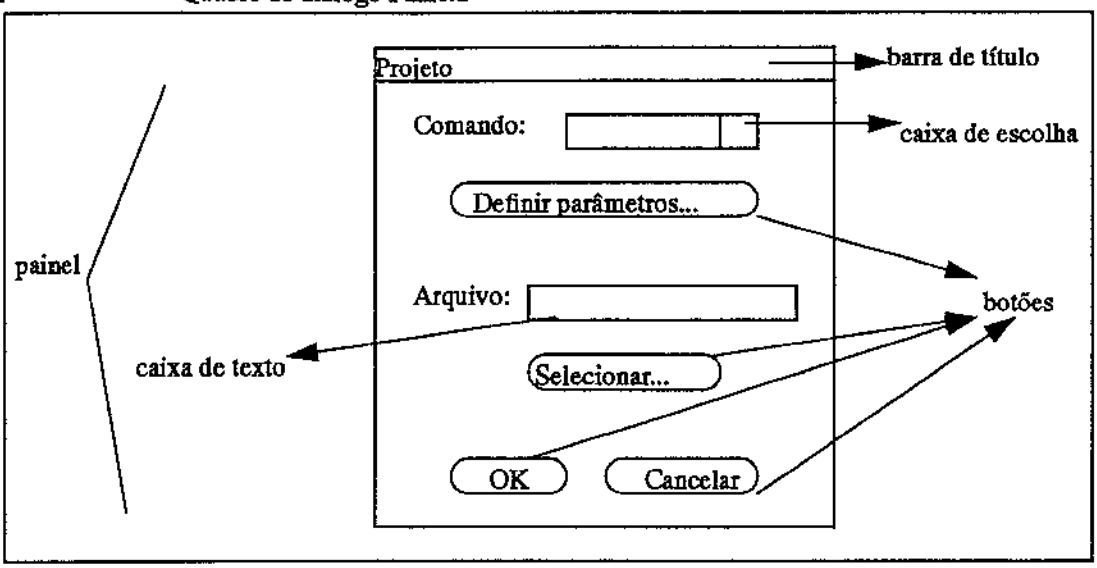
A barra de status indica ao usuário se o ambiente está pronto para receber comandos do usuário ou, em caso contrário, o ambiente está em execução (ocupado).

BARRA DE MENUS

Na barra de menu, tem-se o menu *Visualizar* que possui três itens:

1. *"Iniciar editor..."* - executa um editor apropriado (ou default) para um tipo de dado selecionado na área de opções, será visto na seção 5.3.2.3;
2. *"Executar ferramenta..."* - executa uma ferramenta apropriada (ou default) para um tipo de dado selecionado na área de opções, será visto na seção 5.3.2.4;
3. *"Palheta..."* - abre-se um quadro de diálogo de configuração chamado *Palheta*. O seu layout é ilustrado pela figura 5.2.

Figura 5.2 Quadro de diálogo Palheta



BARRA DE TÍTULO

Note que este quadro de diálogo (na figura 5.2) possui como titulo "Projeto" na barra de título. Isto significa que a Palheta está no contexto de projeto (portanto, o usuário pode fornecer comandos para abrir, fechar, incluir, excluir projetos).

PAINEL

No seu painel, a palheta possui os seguintes objetos de interação:

1. Botão - são quatro botões: "Ok" (para executar o comando escolhido), "Cancelar" (para cancelar a visualização deste quadro de diálogo), "Definir Parâmetros..." (para abrir um novo quadro de diálogo que solicitará ao usuário os parâmetros exigidos pelo comando selecionado na caixa de escolha na figura 5.2) e "Selecionar..." que, se clicado, abre um seletor de arquivos (ver seção 5.3.2.3) para que o usuário selecione um arquivo. O arquivo selecionado é preenchido na caixa de texto que é mostrada pela figura 5.2.
2. Caixa de Texto - permite que o usuário digite o nome do arquivo.
3. Caixa de Escolha - contém todos os comandos disponíveis e somente um deles pode ser selecionado. O usuário, então, seleciona um comando desejado e este será exibido na caixa de escolha.

No contexto da janela principal, a Palheta pode permanecer somente em três contextos: Editor, Projeto ou Ferramenta.

Na janela principal, a Palheta é útil para permitir que o usuário forneça comandos referentes à integração ou remoção de editores e ferramentas e comandos de inserção, remoção, alteração, abertura e fechamento de projetos no ambiente ATIFS.

Quanto aos comandos referentes ao projeto, tem-se uma descrição logo no próximo item.

5.3.2.2 *PROJETO*

O projeto representa um teste específico como, por exemplo, Projeto SX25 - nível 3. O aquivo de projeto mantém dados de projeto tais como: nome, descrição, diretório de trabalho e arquivo de projeto. Pode-se ter dados adicionais como, por exemplo, dados de configuração de projeto como, por exemplo, um conjunto de ferramentas e editores disponíveis a um projeto.

Este trabalho se trata de um protótipo de interface (a fim de levantar os componentes de interface), portanto, aqui, será apresentado de forma simplificada, o projeto mantém os seguintes dados: nome, descrição, diretório de trabalho e arquivo de projeto.

No contexto de projetos, além dos comandos de inserção e remoção de projetos, o usuário também pode criar novos projetos, alterar alguns dados de projeto, abrir um projeto.

INVOCAÇÃO A PARTIR DA JANELA PRINCIPAL

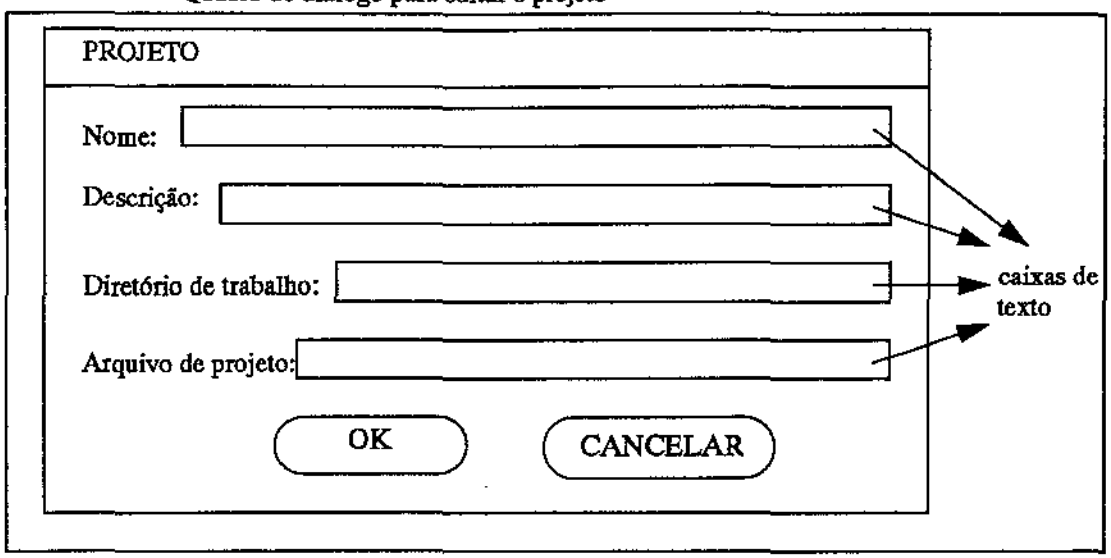
Na Janela Principal, o usuário seleciona a opção "Projeto" na sua área de opções. Em seguida, o usuário deve selecionar o item de menu *Palheta* do menu *Visualizar*.

No quadro de diálogo Palheta (ilustrado na figura 5.2), o usuário deve selecionar o comando "Criar novo" na sua caixa de escolha e clicar no botão OK.

CRIAÇÃO DE UM NOVO PROJETO

Quando o usuário seleciona um comando para criar um novo projeto, então o quadro de diálogo ilustrado na figura 5.3 é exibido.

Figura 5.3 Quadro de diálogo para editar o projeto



O usuário, portanto, deve preencher os dados do projeto e então clicar no botão "Ok". O sistema, então, verifica os dados fornecidos pelo usuário e, se os dados checados estiverem corretos, um novo projeto é criado e armazenado em sua base de dados.

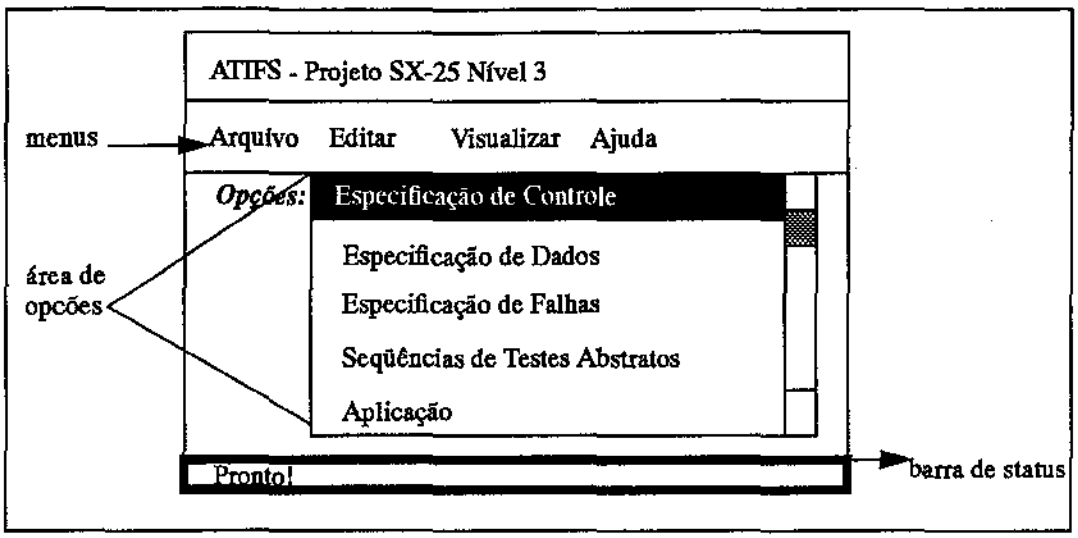
EDIÇÃO DE DADOS DE UM PROJETO

Na Palheta, se o usuário seleciona o comando *editar* (a Palheta deve estar no contexto de projeto), o quadro de diálogo ilustrado na figura 5.3 é exibido contendo dados preenchidos nas caixas de texto permitindo que o usuário edite estes dados.

ABERTURA DE UM PROJETO

Na Palheta, se o usuário seleciona o comando "abrir" na sua caixa de escolha, um projeto é selecionado na sua caixa de texto e, finalmente, clica no OK; abre-se uma nova janela principal no contexto de projeto como, por exemplo, a janela ilustrada na figura 5.4.

Figura 5.4 Janela Principal no contexto de um projeto selecionado pelo usuário



Note que a janela ilustrada na figura 5.4 possui o seu layout idêntico ao de janela principal. Assim, o usuário não terá dificuldade de aprendizado pois todo o seu conhecimento de operação da janela principal na figura 5.1 também se aplica a esta janela.

A partir desta janela, quando o usuário abre o quadro de diálogo Palheta, ele notará que os comandos disponíveis estão no nível mais específico, que são operações para excluir, criar um novo arquivo, inserir, editar e outros. Isto ocorre pois o usuário está no contexto específico de um projeto.

Alguns comandos (como, por exemplo, excluir) acarreta simplesmente uma simples remoção de um arquivo.

Outros comandos (p.ex. editar, criar um novo) acarreta uma chamada de um editor apropriado para um determinado formato de arquivo no estado pronto para criar um novo arquivo ou para editar um arquivo (p.ex. um editor exibindo um conteúdo do arquivo selecionado quando este é carregado).

Há comandos que acarretam uma execução de uma ferramenta apropriada para executar este comando. Por exemplo, o comando "gerar" no contexto de "Seqüências de Testes Abstratos". Para executar este comando, é preciso que o sistema invoque uma ferramenta adequada.

Nesta janela principal, fechar esta janela (ou sair desta aplicação) significa que o usuário deseja fechar o projeto.

5.3.2.3 EDITOR

Nesta seção, será abordado de forma genérica como o usuário edita um arquivo.

INVOCAÇÕES A PARTIR DA JANELA PRINCIPAL

Há duas formas de abrir um editor a partir da janela principal: via menu *Visualizar* ou via Palheta.

A primeira é realizada da seguinte forma: o usuário seleciona uma opção (que representa dados) na área de opções na janela principal. Em seguida, o usuário deve selecionar o item *"Iniciar Editor..."* do menu *Visualizar.*

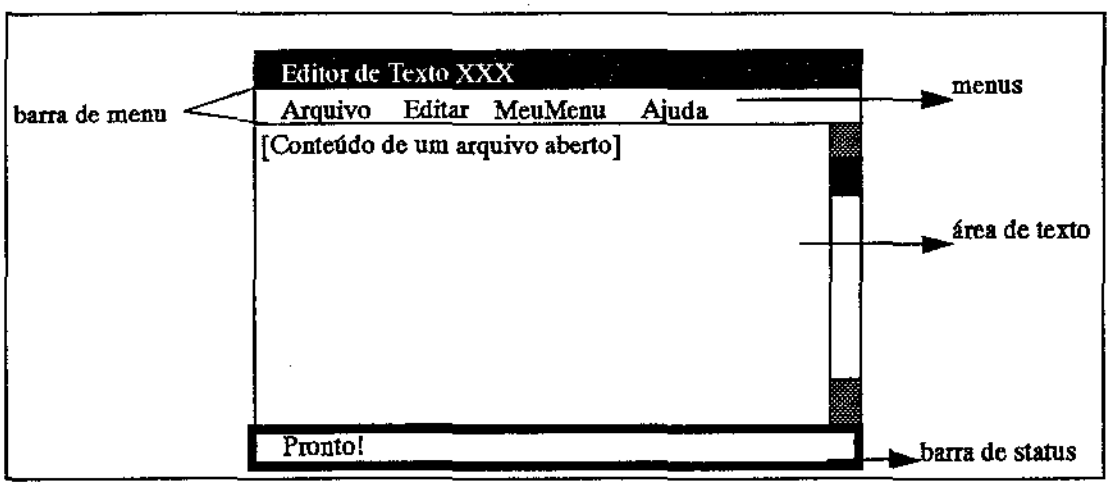
A segunda forma é semelhante à primeira. Ao invés de selecionar o item "Iniciar Editor...", o usuário deve selecionar o item *Palheta* no mesmo menu.

Na Palheta, o usuário deve selecionar o comando *editar* e digitar o nome do arquivo na sua caixa de texto. Ao clicar no botão OK, um editor é executado.

USO DA JANELA DE EDITOR

Todo editor tem a sua *Janela de Editor* (ilustrada na figura 5.5). Esta janela contém uma área de texto e os seus menus.

Figura 5.5 Janela de editor



BARRA DE MENU

Na barra de menu, tem-se o menu Arquivo com os seguintes itens:

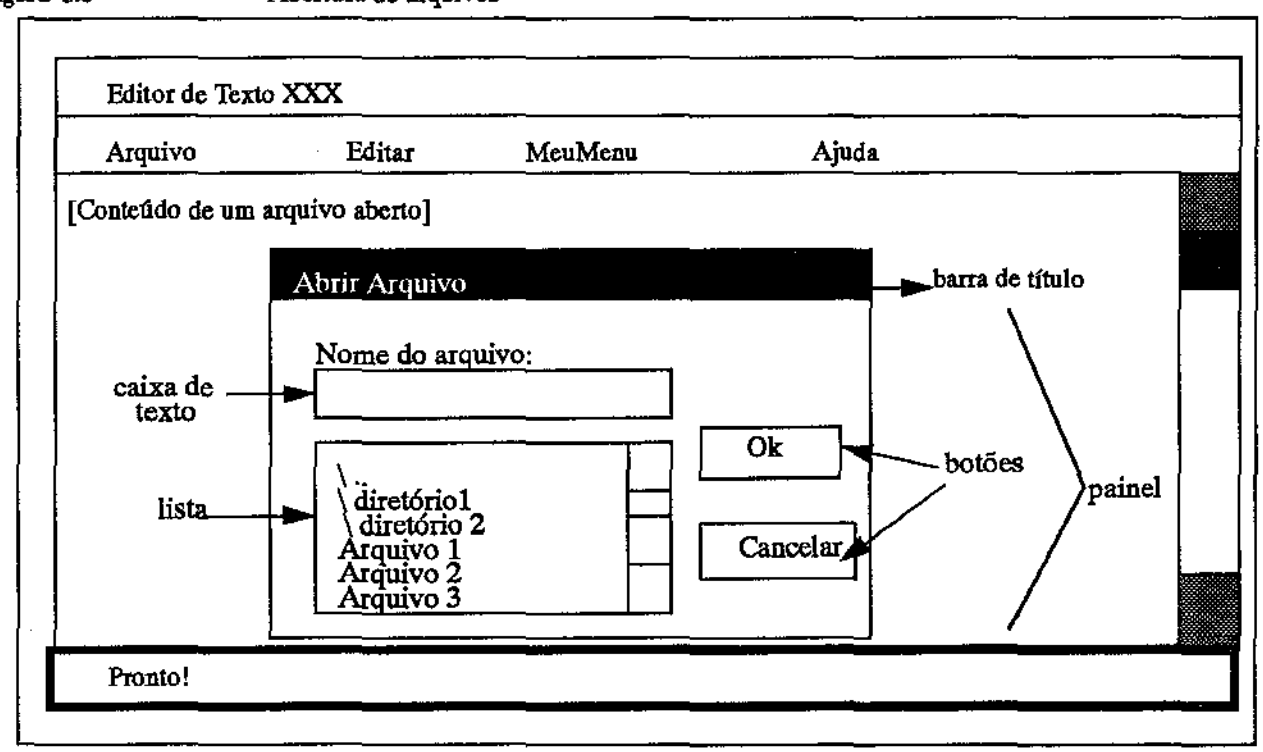
1. "Novo" - quando o usuário deseja criar um novo arquivo. Quando este item é selecionado, a aplicação passa para um estado na qual esta esteja pronta para criar um novo arquivo;
2. "Abrir..." - quando o usuário deseja abrir um arquivo já existente. Vale lembrar que as reticências neste item significa que, quando estes itens forem selecionados,

aparecerá um quadro de diálogo ou uma janela. Portanto, quando este item é selecionado, o quadro de diálogo do tipo seletor de arquivo (ilustrada na figura 5.6) é exibido para que o usuário selecione um arquivo na qual deve ser aberto.

O quadro de diálogo é bastante simples pois contém no seu painel, somente dois botões: "Ok" e "Cancelar". Se o botão "Ok" for clicado, então abre ou salva (depende do contexto deste quadro de diálogo que é mostrado na barra de título) o arquivo selecionado. Já o botão "Cancelar", se for clicado faz com que o quadro de diálogo desapareça sendo ignorada a operação do usuário.

O quadro de diálogo ilustrado pela figura 5.6, também possui uma lista de arquivos e diretórios para facilitar o usuário na seleção do arquivo desejado. Se o usuário simplesmente deseja digitar o nome do arquivo, então ele o digita na caixa de texto.

Figura 5.6 Abertura de arquivos

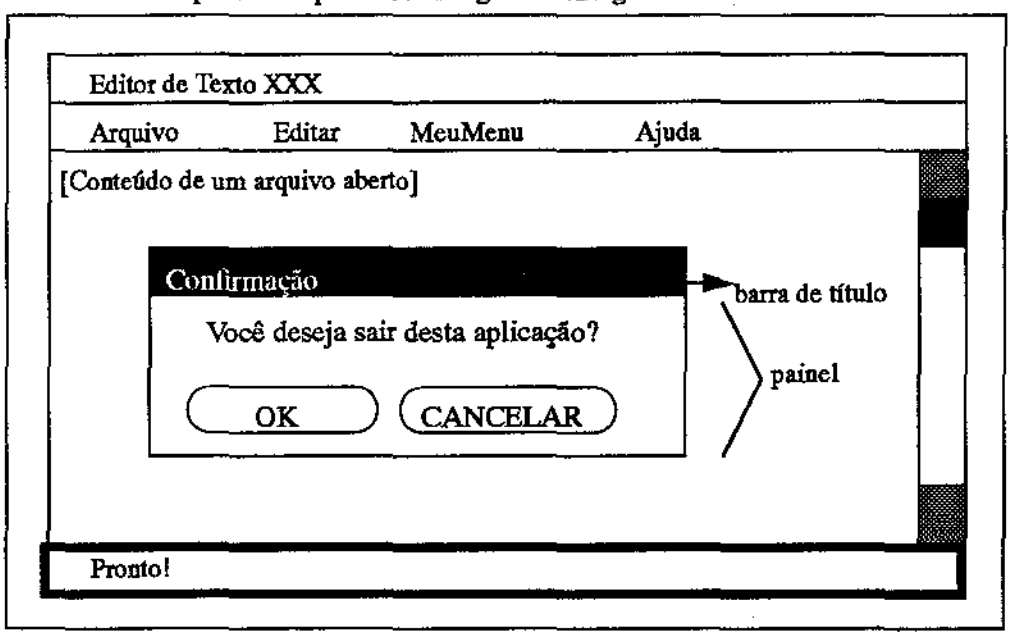


3. "Salvar" - atualiza o arquivo atualmente aberto. A interface não apresenta nenhuma resposta a este evento.
4. "Salvar como..." - salva como um novo arquivo. Quando este item é selecionado, o quadro de diálogo do tipo seletor de arquivo (ilustrado na figura 5.6) é exibido a fim de obter o arquivo que se deseja salvar.
5. "Imprimir..." - imprime o arquivo aberto. Não há nenhuma resposta definida par este evento. Há ambientes de suporte (Windows, p.ex.) que oferece um quadro de

diálogo referente à impressão de arquivos solicitando ao usuário que se deseja imprimir todo o texto ou parte dele, número de cópias e outras informações.

6. "Sair" - quando o usuário deseja sair da aplicação. Quando este item é selecionado, um quadro de diálogo de mensagem (ilustrado na figura 5.7) é exibido, solicitando uma confirmação do usuário a saída da aplicação. Se o usuário confirmar, esta janela é fechada.

Figura 5.7 Exemplo de um quadro de diálogo de mensagem



Na figura 5.7, tem-se um quadro de diálogo solicitando uma confirmação se o usuário realmente deseja sair da aplicação. Caso o usuário realmente deseja sair desta, ele deve clicar no botão "OK"; caso contrário, o usuário clica no botão "CANCELAR".

Voltando no contexto dos menus da janela principal, o menu *Editar* possui os seguintes itens:

1. "Colar" - move o conteúdo selecionado pelo usuário, para área de transferência ou *Clipboard*. O usuário notará que o conteúdo selecionado desaparecerá na sua área de trabalho.
2. "Copiar" - copia o conteúdo selecionado pelo usuário, para área de transferência ou *Clipboard*. O usuário não visualiza a resposta deste evento.
3. "Cortar" - copia o conteúdo do *Clipboard* para a sua área de trabalho. O usuário, então, visualiza os novos dados exibidos na área de trabalho.

O menu *Ajuda* contém os seguintes itens:

1. "Que aplicação é esta?" - se esta opção é selecionada, executa-se um sistema de ajuda informando ao usuário o que se trata da aplicação que ele está operando.
2. "Usando esta aplicação" - abre-se um sistema de ajuda informando ao usuário sobre a operação da aplicação sob consideração.
3. "Sobre..." - quando este item é clicado, abre-se um simples quadro de diálogo de mensagem exibindo dados referentes à aplicação, tais como: nome da aplicação, nome(s) do(s) desenvolvedor(es), versão da aplicação, data de desenvolvimento e outros.

O editor também possui um menu chamado "MeuMenu". Este menu foi inserido nesta janela para representar um conjunto de menus que é específico a cada editor do ambiente, é menu reservado para que os projetistas de ferramentas implementem menus específicos à sua aplicação. Este tema será visto no próximo capítulo.

ÁREA DE TEXTO

A área de texto permite que o usuário edite e visualize um conteúdo de um arquivo.

BARRA DE STATUS

A barra de status, da mesma forma que na janela principal, indica o estado atual do editor.

5.3.2.4 FERRAMENTA

Nesta seção, se trata de ferramentas interativas do ATIFS.

INVOCAÇÕES A PARTIR DA JANELA PRINCIPAL

Há duas formas de abrir uma ferramenta a partir da janela principal: via menu Visualizar ou via Palheta.

A primeira é realizada da seguinte forma: o usuário seleciona uma opção (que representa uma ação) na área de opções. Em seguida, o usuário deve selecionar o item *"Iniciar Ferramenta..."* do menu *Visualizar*.

A segunda forma é semelhante à primeira. Ao invés de selecionar o item "Iniciar Ferramenta...", o usuário deve selecionar o item *Palheta* no mesmo menu.

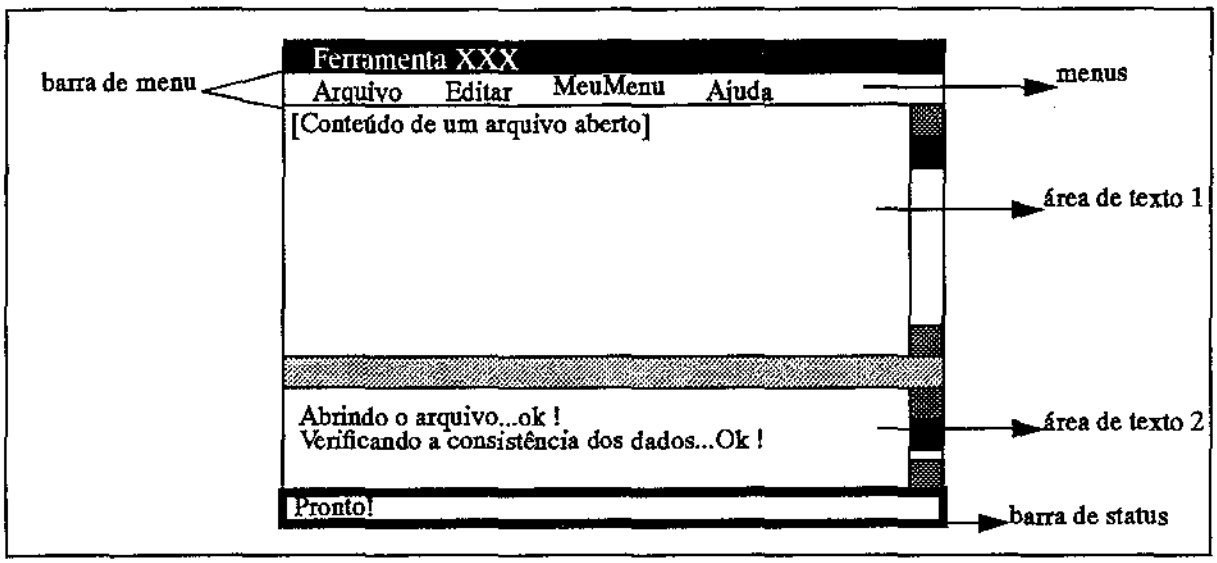
Na Palheta, o usuário deve selecionar o comando *gerar* (ou *executar*) e digitar o nome do arquivo que representa uma ferramenta selecionada na sua caixa de texto. Ao clicar no botão OK, uma ferramenta é executada.

USO DA JANELA DE FERRAMENTA

Toda ferramenta interativa tem a sua *Janela de Ferramenta* (ilustrada na figura 5.8), esta janela é apropriada para ferramentas funcionais que compõem o ambiente. Todas

as ferramentas possuem a mesma apresentação, com estrutura de menus bastante semelhantes e duas áreas de texto.

Figura 5.8 Janela de ferramenta



BARRA DE MENU

A figura 5.8 mostra um exemplo de uma janela de ferramenta cujos menus já foram descritos na seção 5.3.2.3.

ÁREAS DE TEXTO

Esta janela contém duas áreas de texto: área de texto 1 serve para exibir e editar um conteúdo de um arquivo aberto e a área de texto 2 exibe um progresso de execução da ferramenta.

BARRA DE STATUS

A barra de status da janela de ferramenta serve para indicar ao usuário que a ferramenta está pronta para receber comandos do usuário ou, em caso contrário, a ferramenta está em execução, por exemplo.

5.4 CONCLUSÃO

Neste capítulo, foram mostradas as fases iniciais do ciclo de desenvolvimento de interfaces descrito na seção 4.1.3. Inicialmente, foi feito um levantamento das tarefas do usuário do ATIFS, atividade correspondente à fase de Análise. Em seguida, foi

construído um protótipo da interface, com base nas tarefas levantadas, como parte da fase de Projeto da Interação.

O próximo capítulo é relativo à fase de Projeto de Software da Interface, onde é definido como realizar a interface. Nesta fase, o uso de framework é então requerido, sendo mostrado como foi projetado o framework para construção de interface para as ferramentas interativas que irão compor o ATIFS.

Capítulo 6

PROJETO DE SOFTWARE DA INTERFACE

Este capítulo apresenta um projeto de um framework para desenvolver uma interface do ATIFS com base no modelo MVC++.

Este capítulo está estruturado da seguinte forma: a primeira seção é a introdução do capítulo. Na segunda, terceira e quarta seções, tem-se o projeto de software da interface para o componente Visão, para o Modelo (de forma superficial) e para Controlador, respectivamente. Finalmente, tem-se uma seção conclusiva deste capítulo.

6.1 INTRODUÇÃO

No capítulo anterior, foi mostrado o protótipo da interface o qual serve para mostrar o aspecto visual e do comportamento das interfaces de ferramentas que irão constituir o ATIFS e do próprio ambiente ATIFS.

Nesse capítulo, são mostrados os aspectos relativos ao Projeto de Software da interface (seguindo a terminologia apresentada na seção 4.1.3). Nesta fase, é definido o framework para ATIFS com base no MVC++ (visto na seção 4.2.2.3); o enfoque será dado na definição do componente Visão, que trata justamente dos aspectos referentes ao aspecto de apresentação e comportamento da interface de uma aplicação (visto o protótipo no capítulo 5). Os outros componentes serão definidos de forma menos detalhada, somente os aspectos relativos à interação entre os três componentes são apresentados.

Este framework irá permitir aos projetistas de ferramentas interativas que irão compor o ATIFS, definir uma interface uniforme para estas ferramentas.

6.2 COMPONENTE VISÃO

O componente Visão é a parte de um sistema interativo que interage diretamente com o usuário, conforme foi apresentado em 4.2.2.3. Portanto, esse componente é o responsável por garantir o padrão de *look and feel* (padrão de apresentação e comportamento) das diversas ferramentas do ATIFS.

Conforme foi visto na seção 2.3, o componente Visão é constituído por um *toolkit* e um Protocolo de Apresentação.

No caso do ATIFS, o *toolkit* utilizado é o *WXWindows*, o qual é apresentado no primeiro item.

No segundo item, tem-se todo o desenvolvimento da camada (grupo de objetos) do Protocolo de Apresentação do ATIFS levantando novas classes derivadas a partir das classes oferecidas pela biblioteca do *WXWindows*.

6.2.1 FERRAMENTA DE CONSTRUÇÃO DE INTERFACES (TOOLKIT)

A ferramenta de construção de interfaces ou simplesmente *toolkit* tem por objetivo auxiliar o desenvolvimento de interfaces de propósito geral. A ferramenta escolhida é o *WXWindows* [Sma94] por várias razões:

1. o WXWindows é uma ferramenta de domínio público reduzindo, assim, o seu custo (esta está disponível na Internet via ftp: *ftp.aiai.ed.ac.uk* no diretório */pub/packages/wxwin*; ou via www: *http://www.aiai.ed.ac.uk/aiai/software.html*);
2. este *toolkit* permite desenvolver os objetos de interação com facilidade pois oferece uma biblioteca de classes de objetos de interação.
3. o WXWindows permite desenvolver interfaces que podem ser usadas em várias plataformas tais como: Microsoft Windows, OSF Motif, X-Windows. Com isso, o requisito de portabilidade (ver na seção 2.2.2) é atendido.

O WXWindows oferece uma série de classes voltada para a apresentação e inclui também alguns serviços de rede de computadores, de operações de cadeias de caracteres e de arquivos. Todas estas classes podem ser derivadas e utilizadas oferecendo ao projetista a total liberdade de desenvolver uma interface de propósito geral.

Para o ATIFS, estas classes da biblioteca do WXWindows não são suficientes para implementar uma interface adequada pois, somente com as classes do WXWindows, os projetistas podem desenvolver interfaces sem nenhum padrão de apresentação e comportamento (ou padrão de *look and feel*). Assim, não se tem uma uniformidade da interface do ATIFS dificultando o aprendizado do usuário na operação do ambiente.

Para evitar este problema, tem-se uma outra camada (conjunto de objetos) denominada Protocolo de Apresentação. Esta camada (será vista posteriormente) deve garantir o padrão de *look and feel* para que se tenha a uniformidade da interface.

Os padrões de *look and feel* da interface do ATIFS foram levantados na descrição do protótipo de interface vista no capítulo 5. Em seguida, tem-se o desenvolvimento do Protocolo de Apresentação do ATIFS que implementará estes padrões.

Portanto, na próxima seção, tem-se a seção sobre Protocolo de Apresentação.

6.2.2 PROTOCOLO DE APRESENTAÇÃO

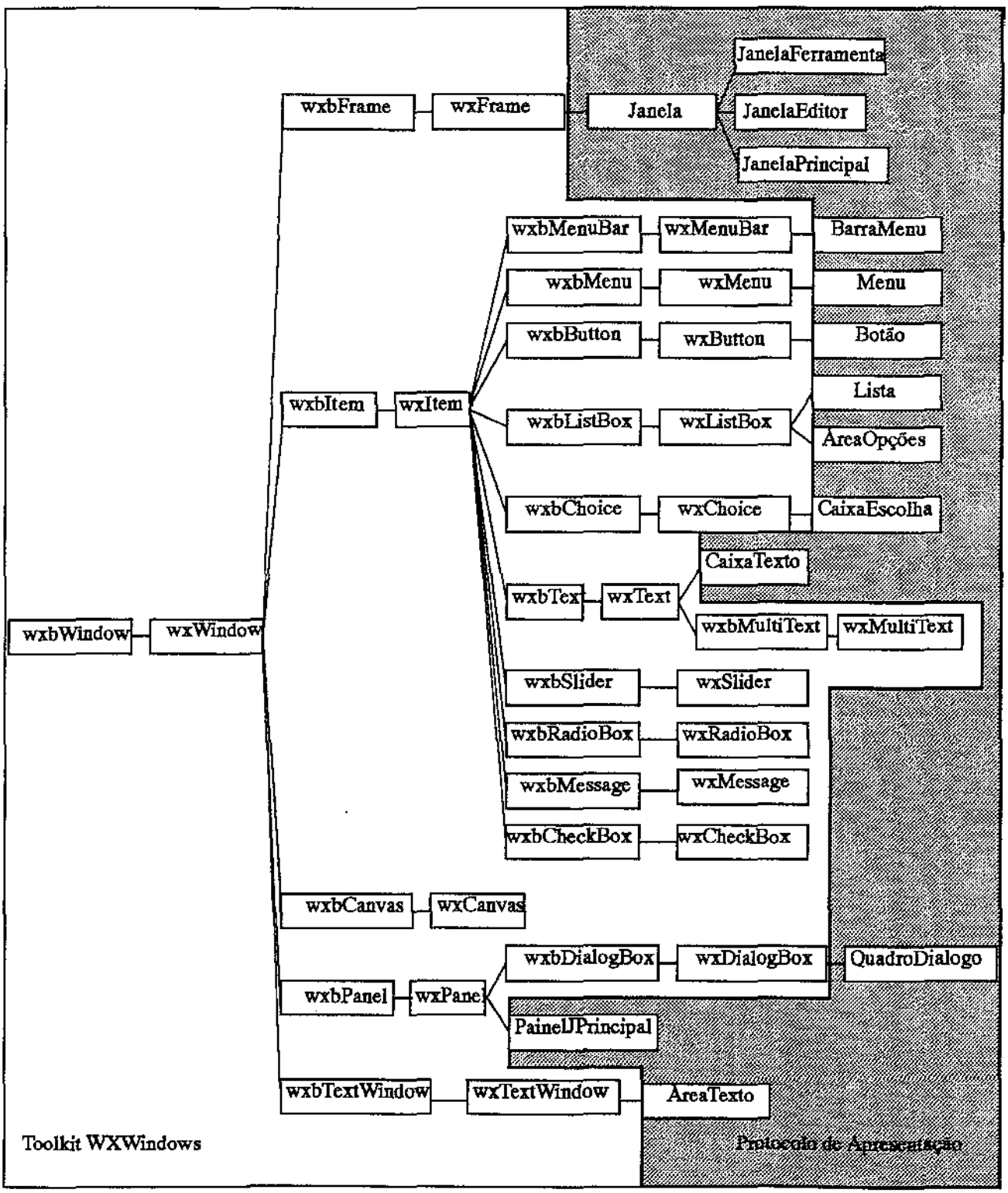
Foi visto que o WXWindows oferece um conjunto de classes genéricas que permitem desenvolver interfaces para qualquer sistema interativo. Neste item, será mostrado como usar estas classes para obter a interface padronizada do ATIFS, como foi desenvolvido o Protocolo de Apresentação do ATIFS.

A partir das classes genéricas oferecidas pelo WXWindows, derivam-se as classes específicas que vão constituir o Protocolo de Apresentação do ATIFS.

A figura 6.1 mostra a estrutura de classes do componente Visão, contendo as classes genéricas[1] oferecidas pelo *toolkit* e as subclasses derivadas a partir destas e que constituem o Protocolo de Apresentação (na área hachurada).

1. Para melhor visualização, não estão apresentadas todas as classes do WXWindows, mas somente aquelas mais relevantes para o modelo apresentado. Também não são mostradas aqui as relações entre as subclasses do Protocolo de Apresentação.

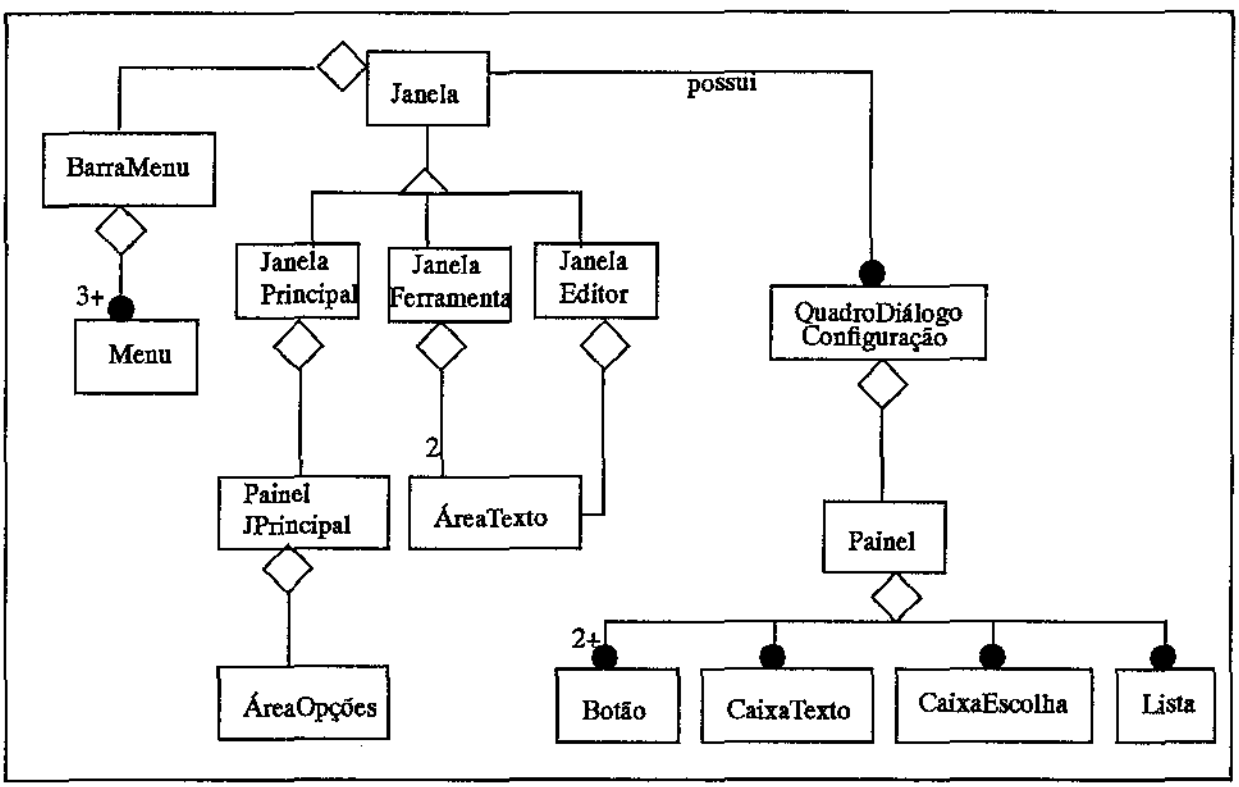
Figura 6.1 Estrutura de classes do componente Visão



Vale lembrar que este Protocolo de Apresentação é dependente do WXWindows. Se o toolkit é substituído por um outro, o processo de levantamento de classes do Protocolo de Apresentação deverá ser realizado novamente lenvantando novas subclasses a partir de um novo toolkit.

Portanto, com base na figura 6.1, é possível obter um modelo de objetos do Protocolo de Apresentação, ilustrado na figura 6.2.

Figura 6.2 Modelo de objetos do Protocolo de Apresentação.



A figura 6.2 ilustra um modelo de objetos. O modelo de objetos do protocolo de apresentação completo contendo atributos e métodos está no apêndice A. Os sub-itens a seguir descrevem os principais componentes desse modelo.

6.2.2.1 JANELA

O modelo mostra três tipos de janelas: de ferramenta, de editor e janela principal. A janela de editor é adequada para editores (ou ferramenta de documentação) contendo um único objeto de área de texto. A figura 5.5 mostra um exemplo de uma janela de editor.

A janela de ferramenta é uma janela adequada para ferramentas interativas funcionais do ATIFS, são ferramentas interativas que, a partir de um tipo de dado, gera um novo

tipo como saída. Esta janela contém dois objetos do tipo área de texto cujo layout está ilustrado na figura 5.8.

A janela principal é a janela adequada para implementar a janela principal do ATIFS ilustrada na figura 5.1. A janela principal possui um painel contendo uma área de opções. Este painel serve para permitir a inserção de uma área de opções em uma janela principal.

Todas as janelas do ATIFS possuem um objeto de barra de menu que será visto no próximo item.

6.2.2.2 BARRA DE MENU

A barra de menu é composta por vários menus. Uma barra tem, no mínimo, 3 (três) menus obrigatórios que são: Arquivo, Editar e Ajuda, vistos em 5.3.2.3.

Estes menus são obrigatoriamente instanciados para cada janela criada pelos projetistas de ferramentas, não podendo ser excluídos nem alterados, mas podendo ser eventualmente desabilitados.

Além destes, o projetista pode acrescentar objetos de menus mais específicos para a sua ferramenta (menus opcionais).

6.2.2.3 QUADRO DE DIÁLOGO

Os quadros de diálogo servem para solicitar informações complementares ao usuário, ou para exibir mensagem de alerta, ou outros.

Neste estudo, foi considerado que quadros de diálogo podem ser de três tipos: seletor de arquivos, mensagem e configuração. Estes tipos são explicados nos sub-itens a seguir.

SELETOR DE ARQUIVOS

Este tipo de quadro serve para solicitar ao usuário a seleção de arquivos.

Este quadro é implementado pelo WXWindows, não sendo por isto mostrado no modelo. O projetista de ferramenta pode usar este tipo de quadro através de um serviço do Protocolo de Apresentação passando os parâmetros necessários para a sua instanciação. Um exemplo desse tipo de quadro é mostrado na figura 5.6.

QUADRO DE DIÁLOGO DE MENSAGEM

Este tipo de quadro serve para exibir mensagens de alerta ou de confirmação. Um exemplo deste tipo de quadro está na figura 5.7.

Esse tipo de quadro também é implementado pelo WXWindows, por isto também não é mostrado no modelo. Seu uso também se faz através de um serviço do Protocolo de

Apresentação, o qual restringe o tipo de quadro que pode ser utilizado pelos projetistas de ferramentas. Desta forma, tem-se os seguintes tipos de quadros de mensagem:

- *de um botão* - pode ser usado para exibição de um alerta ("Disco não inserido!", p.ex.). Para esta situação de alerta, é obrigatório que o quadro de diálogo tenha um único botão: "OK".
- *de dois botões* - pode ser usado para exibir um alerta para uma operação destrutiva solicitada pelo usuário (p.ex.:"Você perderá todos os dados do disco!") oferecendo uma chance de cancelar a operação. Os botões, geralmente, possuem rótulos: "Ok" e "Cancelar".
- *de três botões* - pode ser usado para a confirmação da última operação solicitada. Os rótulos dos três botões são padronizados da seguinte forma: "Sim", "Não", "Cancelar".

QUADRO DE DIÁLOGO DE CONFIGURAÇÃO

Este tipo de quadro serve para solicitar ao usuário informações adicionais e dados que, geralmente, são específicos à aplicação.

Este quadro pode ser instanciado pelos projetistas de ferramentas, os quais vão definir o seu layout. Este layout é obtido através da combinação dos componentes: painel, botão, lista, caixa de texto e caixa de escolha.

Um quadro de diálogo de configuração especial é a Palheta, usada na janela principal do ATIFS, servindo para o usuário fornecer comandos para o ambiente. Um exemplo da mesma é mostrado na figura 5.2.

O projetista de ferramenta pode definir diferentes layouts de quadro de diálogo de configuração. No entanto, o Protocolo de Apresentação estabelece como restrição que, pelo menos, dois botões são obrigatórios: OK e Cancelar.

6.3 COMPONENTE MODELO

Nesta seção, tem-se apenas diretrizes para desenvolvimento do componente Modelo pois este está fora do escopo deste trabalho. Portanto, o objetivo desta seção é mostrar algumas normas que não devem ser violadas a fim de manter a independência dos componentes Visão e Modelo.

6.3.1 NORMAS PARA CRIAR SUBCLASSES DE MODELO

Esta seção menciona algumas diretrizes do Modelo a fim de não violar alguns requisitos do ATIFS, que são:

1. o Modelo só pode interagir com a Visão através do Controlador. A forma como se dá essa interação, será mostrada na seção 6.4;

2. o modelo de objetos do componente Modelo contém somente classes abstratas. Portanto, a partir deste modelo, o projetista pode criar subclasses concretas obtendo serviços primitivos através do mecanismo de herança;
3. o projetista de ferramentas deve retirar das classes que compõem o Modelo tudo que se refere à interface com o usuário, que passarão para as classes do componente Visão e Controlador.

Se o componente Modelo segue estas diretrizes, pode se afirmar que o Modelo está favorecendo o requisito da independência da interface e aplicação.

6.3.2 MODELO

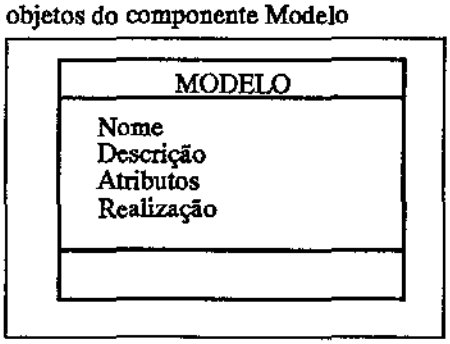
A princípio, o Modelo possui somente uma classe contendo os seguintes atributos mencionados pelo **[Pre95a]**:

- *Nome* - contém o nome do objeto de configuração;
- *Descrição* - contém a descrição do objeto;
- *Atributos* - aqui se tem os atributos que serão levantados para um objeto de configuração específico.
- *Realização* - um ponteiro que aponta para onde o dado está armazenado.

Este modelo possui alguns métodos gerais de todas as classes pertencentes ao componente Modelo.

Note que o componente Modelo não possui um ponteiro que aponta para o seu Controlador pois o Modelo é autônomo não invocando o Controlador e nem a Visão. Esta abordagem reforça a independência entre o Modelo e Controlador/Visão.

Figura 6.3 O modelo de objetos do componente Modelo



6.4 COMPONENTE CONTROLADOR

Esta seção descreve a implementação do componente Controlador o qual será abordado conforme a seguinte estrutura. O primeiro item descreve algumas propostas

levantadas para implementação do Controlador. No segundo item, tem-se o projeto do Controlador do ATIFS propriamente dito. No terceiro (e último) item, tem-se a implementação do Controlador.

6.4.1 FORMAS DE IMPLEMENTAÇÃO DO CONTROLADOR

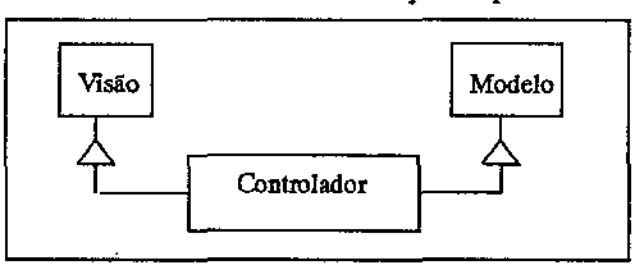
Conforme foi visto na seção 4.2.2.3, o Controlador gerencia a interação entre os componentes Modelo e Visão, integrando-as de acordo com as características específicas de cada aplicação. Neste item, serão mostrados três mecanismos (oferecidos pelo paradigma orientado a objetos) que foram utilizados para implementação da interação entre esses três componentes, que são: herança múltipla, reflexão computacional e delegação.

6.4.1.1 HERANÇA MÚLTIPLA

A herança múltipla é um mecanismo de implementação que permite que uma classe tenha mais de uma superclasse e que herde características de todos os seus ancestrais. A classe com mais de uma superclasse chama-se *classe de junção* (no capítulo 4 em **[RBP+91]**).

Utilizando este mecanismo o *Controlador* seria implementado como classe de junção, herdando a estrutura e o comportamento tanto da Visão quanto do Modelo. Esta implementação do Controlador é ilustrada pela figura 6.4.

Figura 6.4 Derivando o Controlador através de herança múltipla.



Como o Controlador é o componente que deve ser implementado pelo projetista de ferramenta, este poderia sobrepor os métodos do componente Visão e também do Modelo. Permitindo esta sobreposição, os padrões de interface poderiam ser redefinidos e violados pelo projetista.

Para evitar o problema mencionado anteriormente, poder-se-ia utilizar uma forma especial de herança, denominado de *anotação*, o qual foi empregado em *Chiron*, o sistema de gerenciamento de interface do ambiente Arcadia-1 **[TBC+88],[YTT88], [TNB+94], [TNB+95]**.

Neste sistema, a cada tipo abstrato de dados de uma aplicação (Modelo) é associado um **artista**, para fazer a correspondência com o objeto da Visão. O artista é uma subclasse tanto do tipo abstrato de dados (Modelo) quanto da Visão. A anotação faz

com que um artista possa acrescentar novas operações às já existentes, mas tanto as novas operações quanto as extensões só podem fazer modificações aos novos estados.

Um artista só acrescenta novos estados para levar em conta aspectos relativos à exibição de um objeto, e só estende operações existentes para permitir que a visualização do objeto seja modificada quando o objeto o for.

Dessa forma, ao construir um artista, nem os aspectos sintáticos (referentes à assinatura das operações) e nem os aspectos semânticos (que especificam as relações das operações) de um tipo abstrato de dados devem ser modificados.

Dessa forma, não é permitido que o projetista de ferramenta viole a especificação de tipos abstratos de dados do Modelo e nem da Visão.

Este mecanismo de herança com anotação é oferecido pela linguagem Eiffel, C++ (através de métodos virtuais) e também é conhecido como *Hotspot* por **[Pre95b]**.

Segundo **[Wol92]**, a herança múltipla (vista na seção 6.4.1.1) sobrecarrega o sistema pois a classe de junção herda todos os atributos e métodos das suas superclasses (inclusive aqueles que serão desnecessários).

Conclui-se que, tendo em vista o problema de herança múltipla, é preciso obter outras alternativas de implementação do Controlador, que serão vistas a seguir.

6.4.1.2 REFLEXÃO COMPUTACIONAL

O outro mecansimo de implementação estudado é baseado na reflexão computacional **[Mae87]**,**[Fer89]** que oferece soluções bastante elegantes e transparentes à abordagem orientada a objetos.

Nas arquiteturas reflexivas, um sistema computacional é composto por dois níveis: um *nível base* e um *meta-nível*. A tarefa do nível base é resolver os seus problemas e retornar as informações ao domínio externo. Enquanto a tarefa do meta-nível é resolver problemas (mudar um comportamento de um objeto, por exemplo) e retornar informações ao nível de objetos.

Nesta arquitetura, todos os objetos possuem o seu *meta-objeto*. As estruturas contidas em um objeto representam exclusivamente as entidades do domínio. As estruturas do meta-objeto mantêm todas as informações reflexivas tais como: informações sobre a implementação e a interpretação do objeto.

No nível básico, os objetos incorporam métodos reflexivos e não-reflexivos. Os **métodos reflexivos** são métodos que, ao serem invocados, o seu meta-objeto é notificado através de uma chamada a um método de um objeto associado. A execução de um método de um meta-objeto pode resultar em mudanças de comportamento do sistema como um todo.

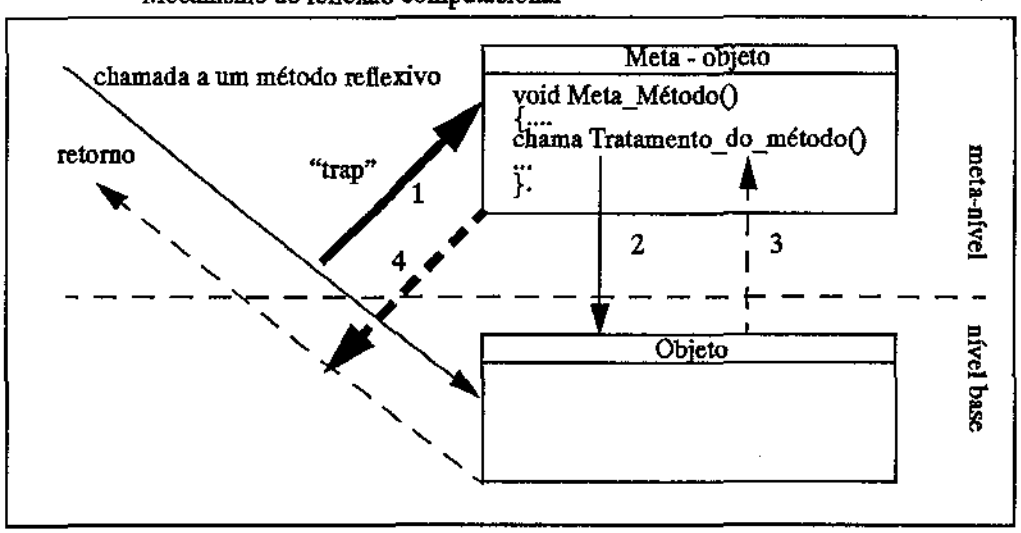
Os **métodos não-reflexivos** são métodos normais no paradigma orientado a objetos, se este tipo de método é invocado, ele é executado normalmente e o seu meta-objeto não detectará esta chamada.

Na figura 6.5, tem-se um esquema baseado no Open C++ que descreve o mecanismo de reflexão computacional. As setas representam os fluxos de mensagem e as tracejadas representam o retorno de dados.

Com base na figura 6.5, quando ocorre uma chamada ao objeto, ocorre um "trap" (representado pela seta 1) que é uma chamada a um método do seu meta-objeto. Então, um método do meta-objeto é executado e este, por sua vez, invoca um método do seu objeto (representado pela seta 2).

Este método, por sua vez, retorna um dado para o meta-método associado (representado pela seta 3). O meta-objeto, portanto, retornará um valor para um agente que invocou o objeto (representado pela seta 4).

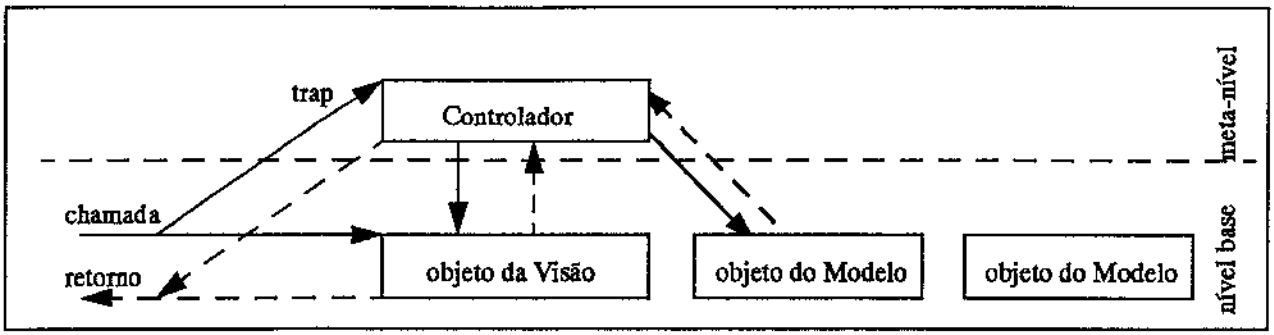
Figura 6.5 Mecanismo de reflexão computacional



Aplicando esta abordagem ao caso do MVC++, o Controlador pode ser um meta-objeto associado aos objetos do Modelo e da Visão. O seu papel é fazer uma interação entre os componentes Visão e Modelo, ilustrado na figura 6.6 (as setas representam os fluxos de mensagem e as tracejadas representam retorno de dados).

Note que há objetos do Modelo que não têm um meta-objeto associado, significando que estes objetos não têm a mínima relevância para a interface, são exclusivamente relevantes para a aplicação.

Figura 6.6 Implementação da interface baseada em reflexão computacional



Cada objeto da Visão possui métodos (ou operações) reflexivos e não-reflexivos, definidos durante a fase de projeto. Se um método deve afetar um objeto do Modelo então se define como método reflexivo; caso contrário, ele é definido como não-reflexivo.

Quando ocorre uma chamada a um método reflexivo de um objeto da Visão, o procedimento é o mesmo mencionado anteriormente. A única diferença é que o meta-objeto interage tanto com o objeto da Visão quanto com o objeto do Modelo.

A reflexão computacional é considerada como um estudo bastante recente, foi constatado que não há algum trabalho concluído relativo à implementação de interface com o usuário usando mecanismo de reflexão computacional.

Esta proposta de implementação da interface descrita nesta seção é simplesmente um passo inicial de como implementar uma interface usando o mecanismo de reflexão fornecendo uma transparência relativa à interação dos componentes Visão e Modelo. É preciso realizar novas pesquisas a fim de assegurar que esta proposta realmente garante a realização da interface. Estas pesquisas serão consideradas como uma extensão futura deste trabalho.

Este mecanismo é bastante elegante mas requer uma linguagem de suporte à programação reflexiva. Não se dispunha de tal ferramenta no início do trabalho, sendo essa uma das razões pela qual a reflexão não foi adotada aqui. Mas a principal razão é que um dos requisitos da interface é a portabilidade (ver seção 2.2.2), pois as ferramentas serão implementadas tanto em plataforma UNIX quanto Windows. Ora, até o momento da conclusão desse trabalho, a única linguagem disponível, Open C++, não tinha versão para Windows.

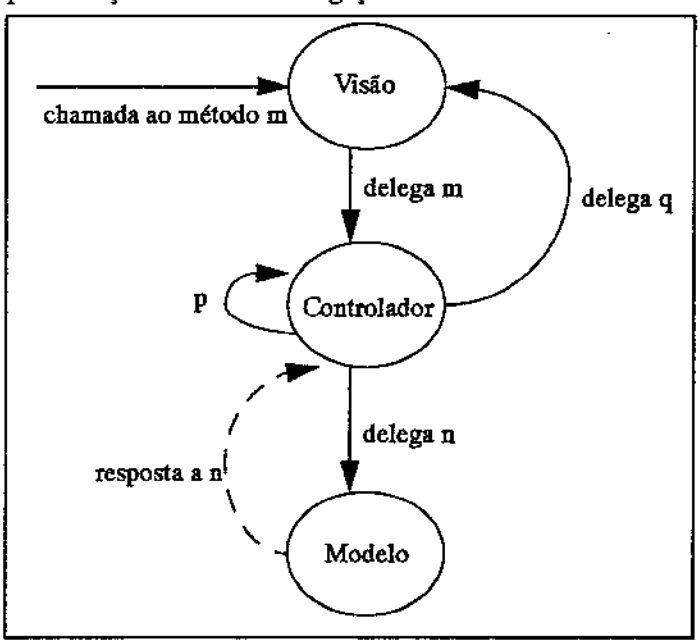
Portanto, por motivo citado acima, será apresentada uma outra alternativa de mecanismo de implementação: a delegação, que será vista a seguir.

6.4.1.3 DELEGAÇÃO

A delegação **[Wol92], [Rub94]** é um mecanismo de implementação no qual um objeto delega uma operação para um outro objeto a fim de executá-la.

A figura 6.7 mostra o comportamento dinâmico do Controlador, cujo papel é realizar a delegação. O Controlador mantém todas as instruções de decisões, quando recebe alguma chamada proveniente da Visão, o Controlador decide quais medidas devem ser tomadas invocando alguns métodos dos objetos relacionados.

Figura 6.7 Implementação baseada em delegação



Com base na figura 6.7 (as setas representam chamadas e a tracejada, o retorno), o usuário gera um evento que é uma chamada a um método *m* de um objeto da Visão. Se o evento recebido por um objeto da Visão deve afetar o componente Modelo, então a Visão delega uma mensagem para o seu Controlador.

Este Controlador pode executar algum método interno *p*, como parte da ação e aciona o método *n* do Modelo. Este, por sua vez, executa a ação correspondente e eventualmente gera uma resposta para o Controlador.

Quando o Controlador recebe uma resposta para a mensagem *n* do Modelo, o Controlador pode invocar o método *q* do objeto da Visão[1] como resposta à delegação da mensagem *m*.

Aqui se encerra as formas de implementação do componente Controlador. Na próxima seção, tem-se a avaliação destas formas a fim de escolher a mais adequada para o caso do ATIFS e também a aplicação desta implementação ao caso deste ambiente.

6.4.2 PROJETO DO CONTROLADOR DO ATIFS

No item anterior, foram mostradas as diferentes formas de implementação do Controlador do framework MVC++ que o paradigma orientado a objetos oferece. Os mecanismos considerados foram: herança múltipla, reflexão computacional e delegação.

A herança múltipla é o mecanismo de implementação que sobrecarrega o sistema obtendo atributos e métodos desnecessários.

Foi visto que a reflexão computacional oferece uma solução bastante elegante no paradigma orientada a objetos mas requer recurso especial (um pré-processador) para implementar esta arquitetura.

Este mecanismo de implementação não será utilizado neste trabalho pois este ainda está em fase inicial da sua evolução e, também, no decorrer deste trabalho, não havia uma ferramenta disponível que desse suporte à arquitetura reflexiva.

A delegação foi, portanto, a implementação escolhida para ser aplicada ao caso do ATIFS. A delegação é uma implementação que contorna o problema da herança múltipla de uma forma simples. Além desta vantagem, a delegação não exige nenhum recurso especial de linguagem para implementá-la que será a linguagem ANSI C++.

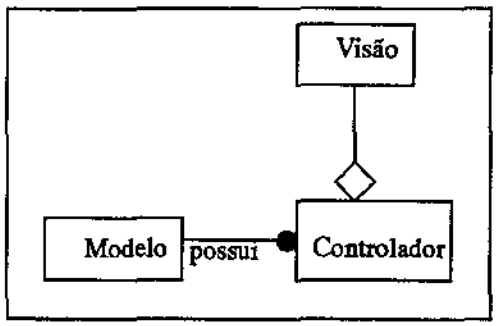
Os sub-itens a seguir, mostram o projeto da classe Controlador, bem como a interação entre esta e os componentes Modelo e Visão, com base no mecanismo de delegação.

6.4.2.1 CLASSE CONTROLADORA

Existe um único Controlador para cada Visão; as classes de Controlador, Visão e Modelo estão da forma mostrada na figura 6.8. Assim, se um determinado objeto do Modelo possui várias visões, há um objeto Controlador para cada uma destas visões, controlando a interação das mesmas com o objeto de Modelo.

1. Vale lembrar que não é permitido ao Modelo responder diretamente ao objeto da Visão.

Figura 6.8 Controlador contém a Visão e associado ao Modelo.



Um objeto Controlador gerencia a interação de um objeto do Modelo e o objeto de Visão associado através de delegação, conforme já foi dito. Para isso, o objeto Controlador deve conhecer os serviços oferecidos pelos objetos do Modelo e de Visão, mas não tem conhecimento de nenhum detalhe interno de como esses objetos prestam esses serviços.

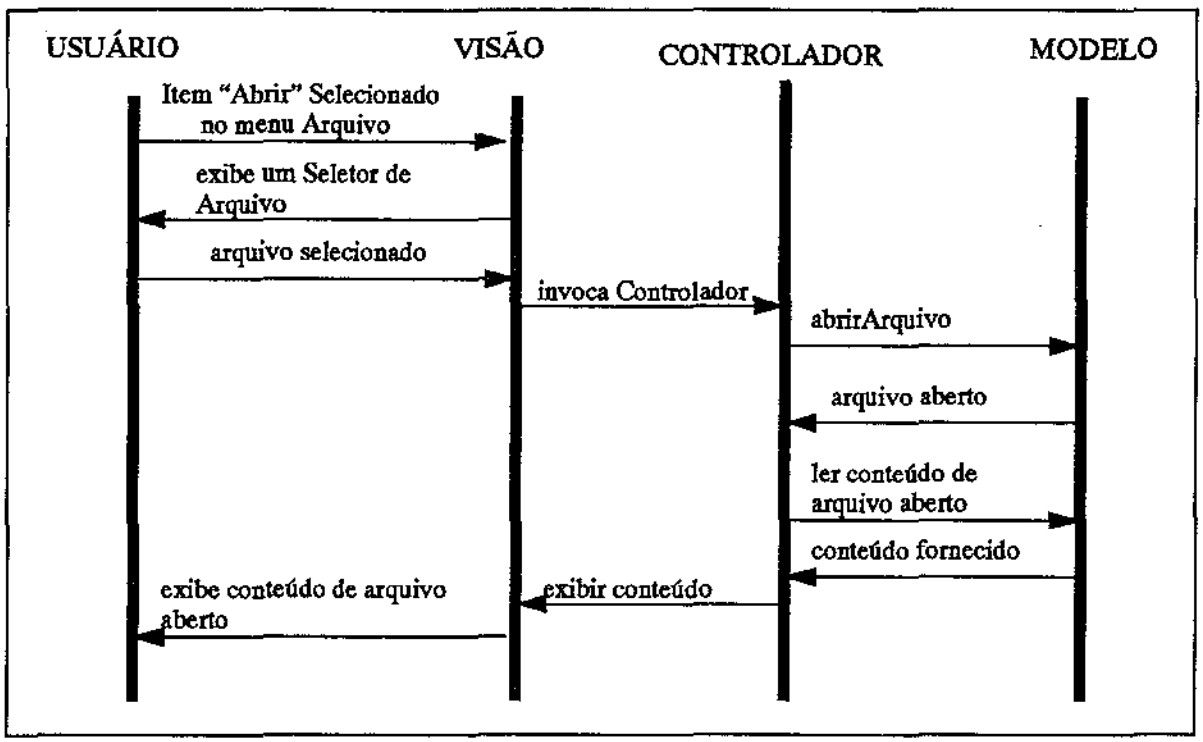
Como o componente Controlador é, portanto, altamente dependente da aplicação deve, por isso, ser desenvolvido pelo projetista de cada ferramenta. No entanto, devem ser respeitadas algumas diretrizes quanto ao seu comportamento, especialmente no que diz a respeito à interação entre os três componentes, o que será tratado no item a seguir.

6.4.2.2 INTERAÇÃO ENTRE CONTROLADOR, VISÃO E MODELO

A interação entre os três componentes será ilustrada através de exemplos, os quais são dados na forma de traço de eventos (ver **[RBP+91]**, item 8.5.3), que é uma lista ordenada de eventos trocados entre os diferentes objetos do sistema e também entre estes e entidades externas, como o usuário ou outro sistema.

O primeiro traço de eventos (figura 6.9) mostra as interações que podem ocorrer quando o usuário seleciona o item "Abrir..." do menu obrigatório Arquivo (ver detalhes na seção 5.3.2.3).

Figura 6.9 Traço de eventos quando o usuário seleciona item "Abrir..." do menu Arquivo



Quando o usuário seleciona o item "Abrir..." do menu Arquivo, o componente Visão solicita ao usuário uma seleção de arquivo exibindo um tipo de quadro de diálogo Seletor de Arquivos (visto na seção 5.3.2.3).

Selecionado o arquivo, a Visão invoca o Controlador chamando um método específico que conterá códigos referentes à abertura de arquivos. Neste método, o Controlador solicita ao Modelo a abertura de um arquivo selecionado.

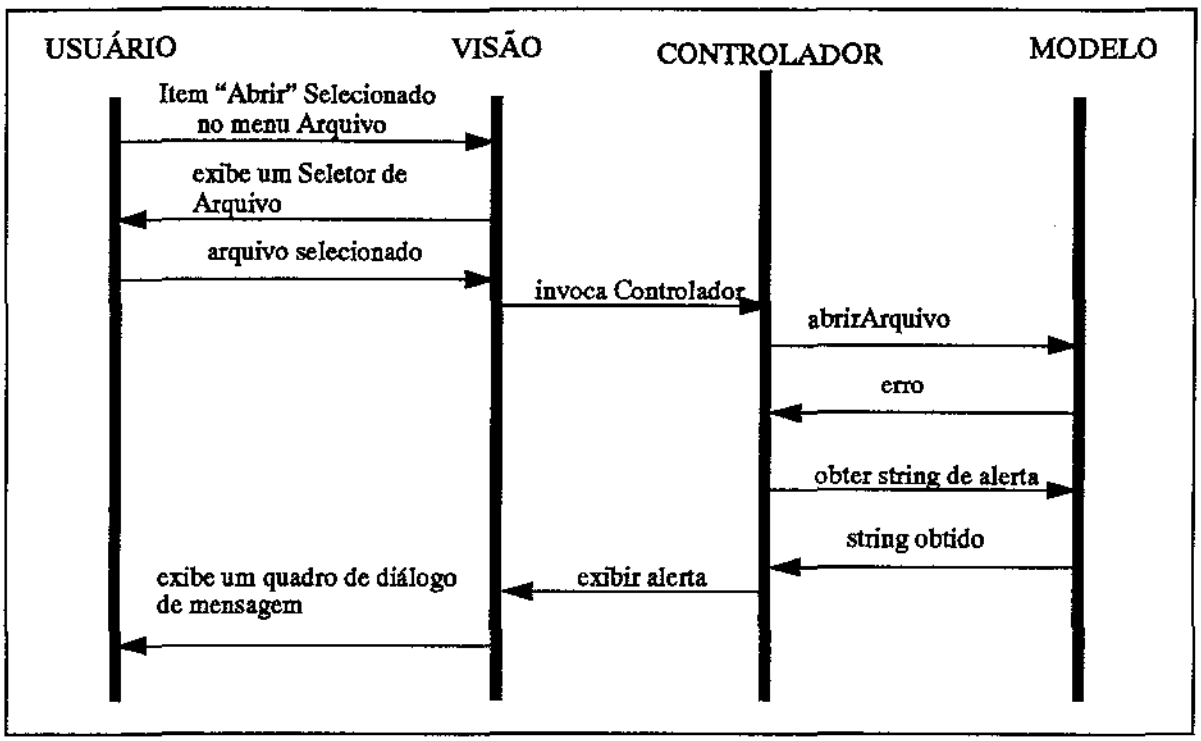
O componente Modelo, checa se este arquivo existe, e confirma ao Controlador que o arquivo está aberto.

Aberto o arquivo, o Controlador lê o conteúdo do arquivo e solicita a Visão exibir o conteúdo através dos serviços do Protocolo de Apresentação descritos no apêndice B.

A Visão, por sua vez, exibe o conteúdo do arquivo respondendo ao usuário a operação de abertura de arquivo.

O que foi mostrado acima ilustra o que ocorre em situação normal. Para exemplificar um caso de erro, supõe-se que o arquivo selecionado pelo usuário não existe. Neste caso, tem-se o traço de evento na figura 6.10.

Figura 6.10 Traço de eventos em um caso de erro.

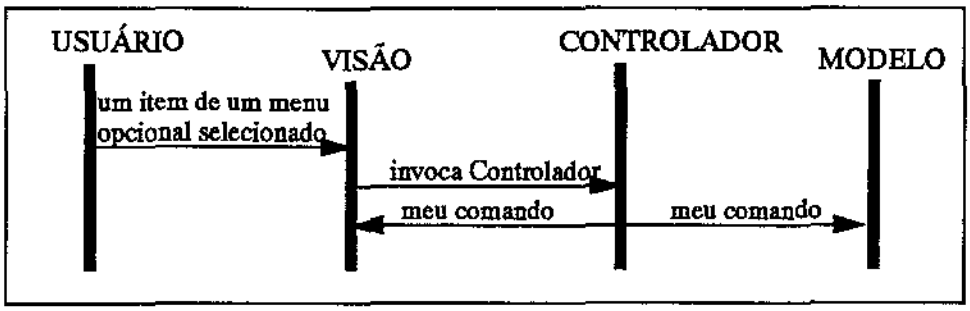


A diferença em relação ao primeiro traço de eventos é quando o componente Modelo retorna um erro. O Controlador, por sua vez, solicita ao Modelo um string (cadeia de caracteres) de erro através do evento "obter string de alerta".

Obtido este string, o Controlador solicita à Visão a exibição de um alerta passando como parâmetro o string de alerta. Assim, a Visão exibe um quadro de diálogo de mensagem com este alerta.

O outro cenário considerado neste trabalho é quando o usuário seleciona um item de menu opcional que é específico a uma ferramenta. Neste caso, tem-se o traço de eventos genérico ilustrado na figura 6.11.

Figura 6.11 Traço de eventos quando um item de menu opcional é selecionado.



Quando o usuário seleciona um item de menu opcional, a Visão invoca o Controlador e este, por sua vez, deve solicitar ao Modelo utilizando um serviço do Modelo.

Neste caso, tem-se um traço de eventos genérico pois se trata de um item de menu opcional de uma aplicação do ATIFS. Portanto, o Controlador deve ser definido pelo projetista a fim de que possa responder a eventos específicos (representados pelas setas "meu comando" na figura 6.11) a uma aplicação interagindo com o Modelo e Visão. A implementação do Controlador será vista de forma mais detalhada na próxima seção.

O Controlador, por exemplo, pode solicitar ao componente Visão uma exibição de um quadro de diálogo de configuração (ver detalhes na seção 6.2.2.3) específica a uma aplicação.

Foi visto que os quadros de diálogo de configuração possuem obrigatoriamente dois botões: OK e Cancelar. Se o botão OK é clicado, o componente Visão invoca o Controlador para que o Controlador possa obter os dados fornecidos pelo usuário.

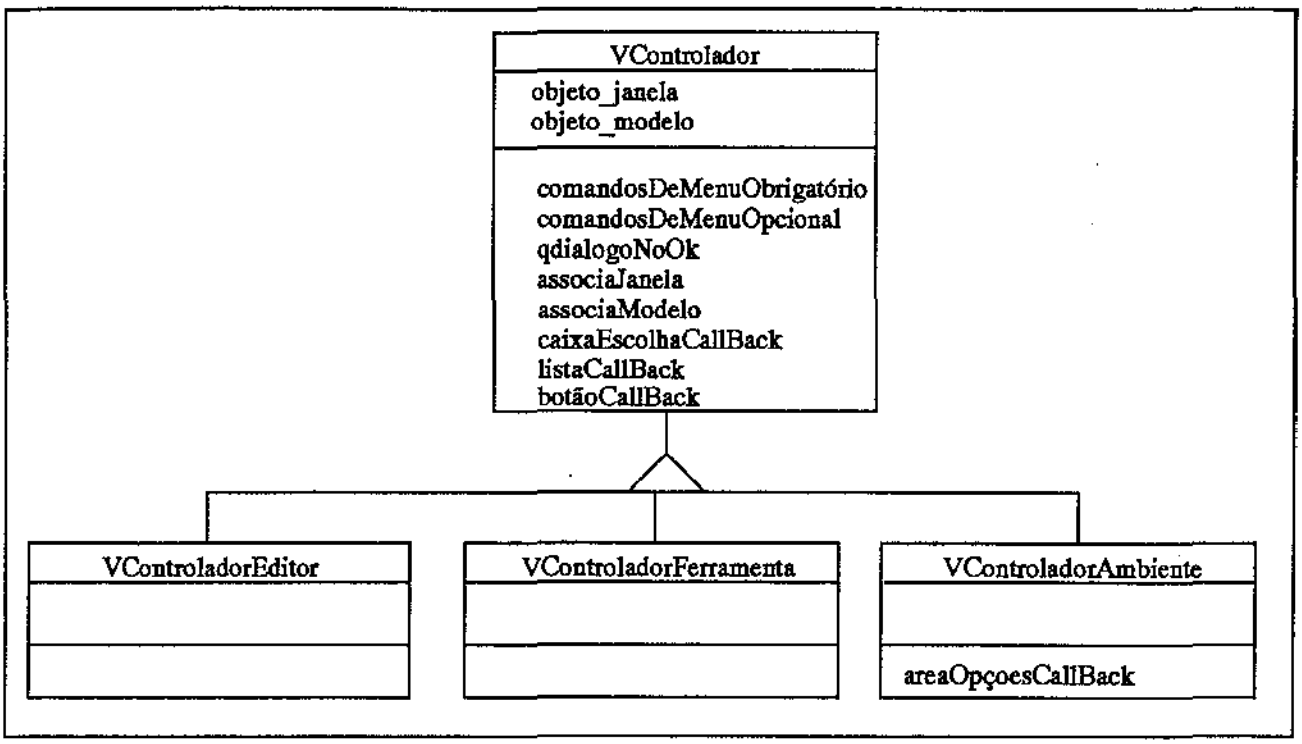
Se o usuário clica no botão Cancelar, a Visão destrói o objeto de quadro de diálogo sem gerar nenhum evento para o Controlador e aguarda um novo evento do usuário.

6.4.3 *IMPLEMENTAÇÃO DO CONTROLADOR*

Para ilustrar o modelo de objetos do componente Controlador, tem-se a figura 6.12 que ilustra as classes abstratas com seus atributos e métodos.

Na figura 6.12, são mostradas as classes *VControladorAmbiente* (controla a interface do ambiente), *VControladorFerramenta* (controla a interface de uma ferramenta funcional) e *VControladorEditor* (controla a interface de um editor). Estas classes são derivadas de uma classe genérica, *VControlador*, na qual são definidas todas as características comuns aos três tipos (ver detalhes no apêndice A).

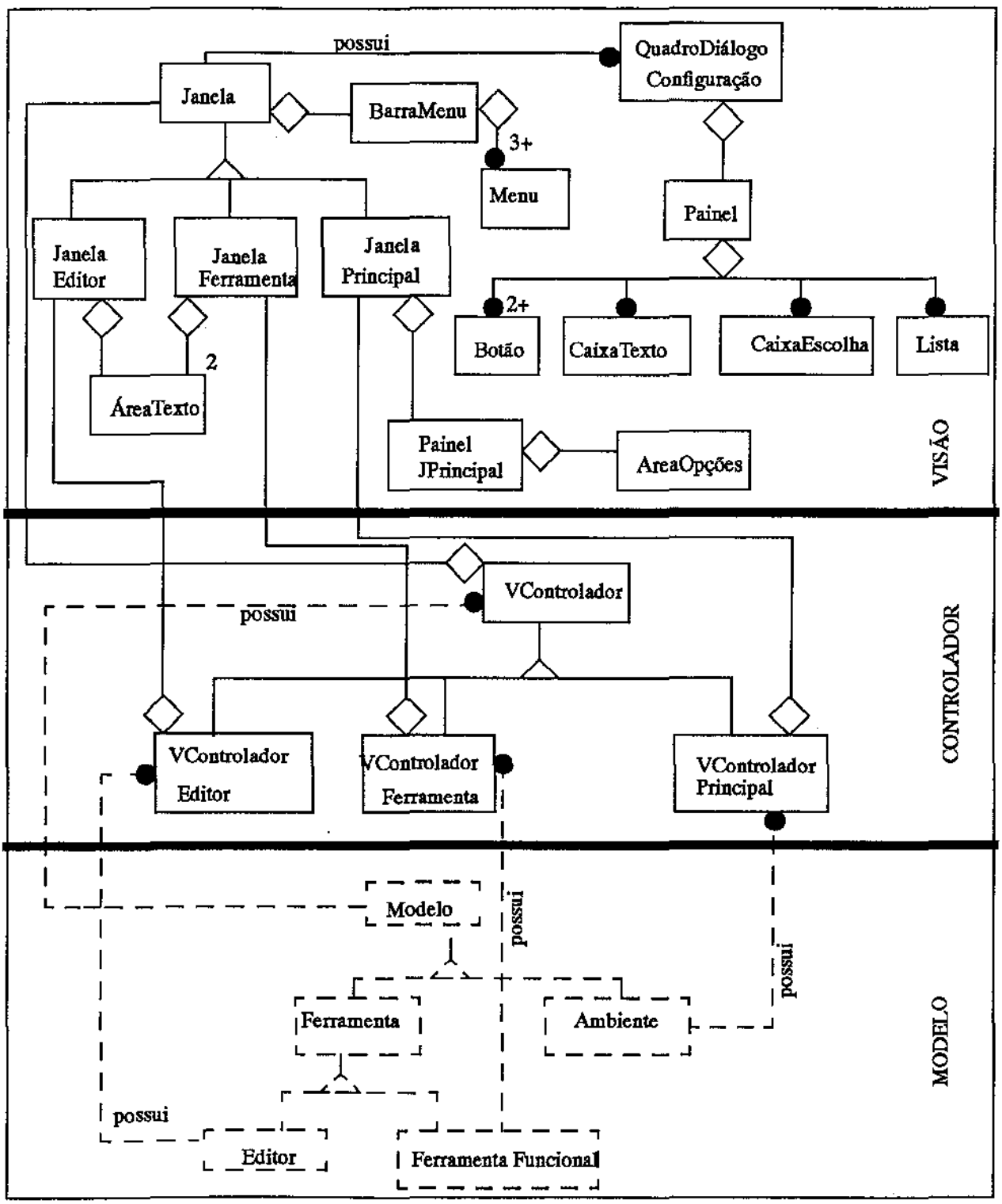
Figura 6.12 Modelo do componente Controlador.



É importante notar que as classes descritas acima são abstratas. Elas servem para definir o comportamento de cada Controlador conforme foi descrito no item anterior, visando garantir dessa forma a uniformidade na interação entre os componentes. Portanto, ao construir as subclasses concretas que tratam de aspectos específicos da sua aplicação, o projetista de ferramentas não deve criar novos métodos que tratam da comunicação entre os componentes, pois perde-se a uniformidade na interação. O apêndice B mostra quais métodos que o projetista deve alterar para poder definir aspectos específicos da sua ferramenta.

Note que o VControlador possui dois atributos que são os ponteiros que associam os componentes Visão e Modelo.

Figura 6.13 Framework da interface do ATIFS



Cada método das classes do componente Controlador são invocados somente quando um determinado evento é gerado. Por exemplo, o método *comandosDeMenuOpcional* é invocado somente quando um item de um menu opcional é selecionado pelo usuário. Portanto, para responder a este evento, o projetista define este método e pode ser implementado usando linguagem ANSI C++.

Já o método *comandosDeMenuObrigatório* é invocado somente quando um item de menu obrigatório é invocado. A descrição de cada método e atributo destas classes está no apêndice A. Será visto com maiores detalhes sobre como implementar um Controlador na seção 7.5.

Definido o Controlador, este é inicializado instanciando automaticamente um objeto de janela (Visão); e também, durante a inicialização da aplicação, são feitas as associações dos componentes Controlador e Modelo como mostra a figura 6.13.

O Modelo possuiria duas subclasses Ambiente e Ferramenta. Na classe Ambiente, tem-se as funcionalidades do ambiente ATIFS e a classe Ferramenta, as funcionalidades das ferramentas interativas que integram o ambiente.

Na classe Ferramenta, ter-se-ia duas subclasses. Uma representa um editor e outra representa ferramenta funcional.

A figura 6.13 que mostra modelo de objetos que representa o framework da interface do ATIFS[1]. Esta figura utiliza a notação da metodologia OMT. As classes e relacionamentos tracejados indicam as partes a serem definidas pelos projetistas.

Após a inicialização da aplicação, esta permanece no estado de espera aguardando eventos do usuário.

6.5 CONCLUSÃO

Neste capítulo, foram mostrados aspectos de projeto de um sistema interativo do ATIFS (ambiente ou ferramenta), com base no MVC++. Foram descritos os três componentes, com ênfase na Visão e no Controlador, que são os mais relevantes no contexto desse trabalho.

Para o componente Visão, tem-se descrições do toolkit *WXWindows*, dos padrões de *look and feel* da interface e também se tem o desenvolvimento do modelo de objetos do Protocolo de Apresentação. As classes do WXWindows e do protocolo de apresentação levantadas, portanto, compõem o sub-framework do componente Visão.

1. Para melhor visualização, esta figura ilustra o componente Visão somente a parte do Protocolo de Apresentação.

O Protocolo de Apresentação é desenvolvido a partir do *WXWindows* a fim de oferecer serviços aos projetistas para que eles desenvolvam uma interface para seus sistemas com o *look and feel* padronizado por esta camada.

Contrariamente o componente Visão, cujo sub-framework contém classes abstratas e concretas, estas últimas podendo ser instanciadas quando da criação de aplicação, para o componente Modelo, foram dadas unicamente as diretrizes para que os projetistas de ferramentas possam construir modelos que sigam o padrão de interação com a interface que não viole os requisitos definidos para o ATIFS.

Finalmente, foi também apresentada a definição abstrata do componente Controlador. Este componente é importante pois ele é que torna possível que o requisito *independência de diálogo* seja satisfeito. Por ser altamente dependente da aplicação, este componente deve ser contruído pelos projetistas de ferramentas. Para tanto, estes criarão seus controladores a partir das classes abstratas que definem o padrão de interação entre Modelo e Visão usado no ATIFS. Esta interação é do tipo controle externo, no qual os objetos de Modelo só são invocados a partir dos objetos da Visão.

Este trabalho apresentou uma forma de projetar um framework de interface de um ambiente com base no modelo escolhido MVC++.

Neste capítulo, foram mostradas algumas formas de mecanismos de implementação que podem ser aplicados para o MVC++. Os três mecanismos estudados foram herança múltipla, reflexão e delegação, sendo este último o mecanismo escolhido, pois não apresenta a sobrecarga do primeiro e nem requer ferramenta especial, como o segundo.

Para mostrar uma aplicação da delegação, este trabalho apresenta a implementação baseada em delegação para realizar o MVC++ no caso do ATIFS.

O próximo capítulo contém uma descrição do uso do framework da interface do ATIFS.

Capítulo 7

DESCRIÇÃO DO USO DO FRAMEWORK

Neste capítulo, tem-se uma descrição do uso do framework da interface do ATIFS. Assim, esta descrição orienta o projetista como desenvolver a interface da sua aplicação.

Este capítulo possui a parte introdutória na primeira seção. A seguir, tem-se uma seção que descreve como usar este framework nos três componentes básicos do MVC++: Modelo, Visão e Controlador; e, finalmente, este capítulo se encerra com a sua conclusão.

7.1 INTRODUÇÃO

Este capítulo aborda o uso do framework, descrevendo como o projetista de ferramentas utiliza o framework descrito no capítulo 6.

Vale lembrar que foi visto que o framework possui três componentes. O componente Visão é do tipo caixa-preta. O componente Controlador é do tipo caixa-branca. Finalmente, o componente Modelo não foi definido no âmbito deste trabalho pois este foge do escopo do trabalho.

Apesar dos frameworks caixa-branca (ver detalhes na seção 4.2.1.2) serem mais flexíveis, pois permitem a especialização de classes que podem vir a ser utilizadas por outras aplicações além daquelas consideradas no domínio do problema, o framework caixa-preta é mais adequado no contexto de interface com o usuário do ATIFS pelas seguintes razões:

- segundo **[JF88]**, este tipo de framework facilita a reutilização: basta apenas que sua interface seja compreendida pelos projetistas das ferramentas;
- o framework é reutilizado sem alterações e acréscimos, o que evita a construção de ferramentas com interfaces não uniformes com o padrão definido para o ATIFS.

É importante que o framework seja descrito de forma clara e concisa permitindo que seus usuários possam usá-lo de forma correta. Existem técnicas para descrever frameworks que são chamadas de *padrões de projeto*[1] **[Pre95b]** que serão descritas na próxima seção.

Vale lembrar que este tema será abordado principalmente nos aspectos referentes aos componentes Visão e Controlador. O Modelo será descrito de forma superficial pois este foge ao escopo do trabalho.

O capítulo se divide em cinco seções: a primeira apresenta uma parte teórica sobre documentação de frameworks; a segunda apresenta como usar o framework no contexto do componente Visão do ATIFS; a terceira seção descreve o uso deste no contexto do componente Modelo; a quarta seção o menciona no contexto do Controlador. E, finalmente, este capítulo se encerra com a conclusão.

7.2 DOCUMENTAÇÃO DE FRAMEWORKS

Existem várias técnicas para descrever frameworks que podem ser vistas em **[Joh92]**, **[Pre95b]**, **[Coa92]**, **[Cop92]**, **[HRG90]**, **[GHJ+93]**.

No caso do ATIFS, o objetivo do padrão de projeto não é apenas documentar o framework, mas também descrever como usar o framework. Portanto, o padrão de projeto escolhido é o do **[Joh92]** e **[Pre95b]** do tipo *guia de programação*[2]. Segundo **[Pre95b]**, este tipo de padrão de projeto é o mais adequado para este fim.

O guia de programação deve conter um conjunto de padrões. Os padrões devem ser orientados a problemas e não a soluções. Cada padrão descreve um problema como uma parte do problema de projeto e como resolvê-lo.

O guia de programação deve descrever **[Joh92]**:

- o propósito do framework - domínio do problema para qual ele foi projetado, o qual deve ser descrito logo no primeiro padrão pois, no início da leitura de uma documentação de framework, o leitor geralmente deseja saber o propósito do framework;

1. do inglês *design patterns.*
2. do inglês *cookbook.*

- como usar o framework - descrição informal de como usar o framework para construir uma aplicação;
- projeto detalhado do framework - as informações do projeto detalhado devem ficar o mais escondido possível do leitor pois a maioria deles não está interessada nestas informações.

Recomenda-se também que, na documentação do projeto, tenha-se um pequeno conjunto de exemplos. Estes exemplos não servem para mostrar como usar e nem explicar o projeto do framework, mas sim mostrar que o framework é adequado para o seu próposito.

Vale lembrar que não será descrito o projeto do framework para a interface do ATIFS pois já foi descrito no capítulo 6 e nos apêndices A e B. Portanto, neste capítulo, tem-se o propósito e o como usar o framework da interface do ATIFS.

Nas próximas seções, usam-se as diretrizes dadasna referência supra-citada para descrever o framework da interface do ATIFS seguindo o padrão designado como guia de programação.

7.3 COMPONENTE VISÃO

Foi visto o projeto do framework para o componente Visão na seção 6.2. Nesta seção, será visto como usar o framework para implementar uma interface com o usuário do ATIFS.

7.3.1 PADRÃO 1: PROPÓSITO

O framework serve para desenvolver uma interface com o usuário do ATIFS. Assim, o usuário obtém uma interface dentro dos padrões definidos (ver seção 5.3) neste ambiente.

Inicialmente, o projetista (usuário deste framework) deve definir a inicialização da sua aplicação usando a linguagem ANSI C++ (para permitir portabilidade de diversas plataformas) e também se pode invocar serviços do protocolo de apresentação e do Modelo.

Nesta inicialização, tem-se a implementação de menus, instanciação de um objeto Controlador (será visto na seção 7.5), instanciação de um objeto de uma classe do componente Modelo e associação do Controlador e Modelo.

Vale lembrar que, ao criar um objeto Controlador, automaticamente o seu objeto de janela é criado.

Para criar menu(s), ver "Padrão 2: Implementação de menus" na seção 7.3.2.

Para criar Controlador, ver seção 7.5.

A inicialização é feita através da definição do método *inicialização* da classe *Aplicação*. Este método deve sempre retornar um objeto do tipo janela.

A seguir, tem-se um exemplo (em C++) fornecendo uma estrutura inicial de definição do método *inicialização* da classe Aplicação:

```
Janela *Aplicacao::inicializacao(void)
{
// criar um objeto de menu - ver seção "Padrão 2: Implementação de menus"
Menu* menu = new Menu;
menu->inserirItens (ITEM1, "Item 1", "O item 1");
menu->inserirItens(ITEM2, "Item 2", "O item 2");
Menu[0] = menu;
rotulos[0] = copiaString("MeuMenu");

// Criar um objeto Controlador - será visto na seção 7.5.
obj_controlador = new Controlador("Ferramenta XXX", "*.xxx", "help1.hlp", "help2.hlp",
"Ferramenta XXX - Protótipo versão 1.0 \n Marcos Salenko Guimarães \n Março de 1995", 1,
menu, rotulo);

// Instanciando a classe do Modelo criado - ver na seção 7.4
objeto_modelo = new Ambiente;

// Associa o Controlador ao Modelo.
obj_contolador->associaModelo(objeto_modelo);
objeto_modelo->associaControlador(obj_controlador);

// retorno obrigatorio de um objeto de janela criado
return obj_controlador->objeto_janela; // objeto de janela criado pelo Controlador
}
```

Os comentários indicam as instruções de criação de menus, criação de um objeto Controlador, instanciação de um objeto do Modelo que, por enquanto, são irrelevantes no contexto deste item.

Note também que sempre um objeto do tipo Controlador e um objeto do tipo Modelo devem ser associados. O objeto Controlador deve utilizar o método *associaModelo* (objeto do Modelo como parâmetro) para ser associado ao Modelo e o método *associarControlador* (objeto do Controlador como parâmetro) do Modelo deve ser invocado para que o Modelo seja associado ao seu objeto Controlador.

7.3.2 PADRÃO 2: IMPLEMENTAÇÃO DE MENUS

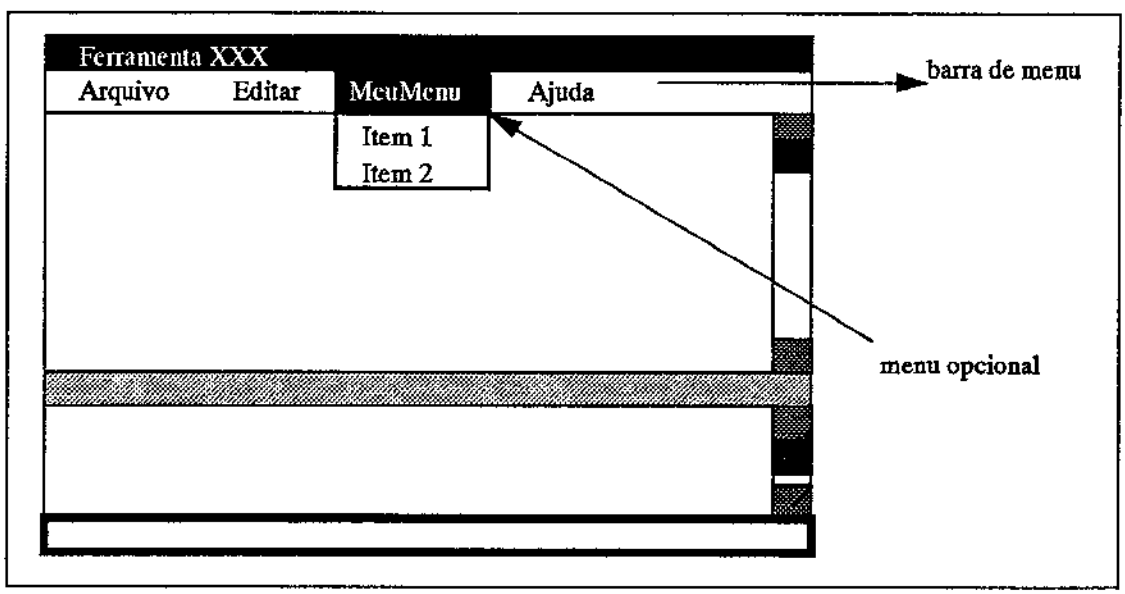
Nesta seção, tem-se a descrição de como o projetista insere menus opcionais e seus itens em uma janela de ferramenta ou de editor.

Para implementar menus, o projetista deve criar um vetor de objetos de menu e um vetor de rótulos de menu. Estes vetores representam um conjunto de menus opcionais. Os rótulos são cadeias de caracteres que aparecem na barra de menu da janela como mostra a figura 7.1. O tamanho destes vetores corresponde à quantidade de menus opcionais que devem ser inseridos na janela.

Por exemplo, para implementar o menu cujo rótulo é "MeuMenu" como mostra a figura 7.1, é preciso criar um vetor de menus de tamanho 1 (pois esta janela tem somente um menu opcional) chamado *menus* e um vetor de rótulos (também de tamanho 1) chamado *rotulos*. Para isso, tem-se a seguinte declaração:

```
Menu *menus[1];
char *rotulos[1];
```

Figura 7.1 Janela da Ferramenta XXX



A seguir, instancia-se um objeto da classe *Menu*. Para criar um objeto de menu chamado *menu_especifico*, insere-se o seguinte código:

```
Menu *menu_especifico = new Menu;
```

Em seguida, o projetista deve inserir itens de menu deste objeto recém-criado. Por exemplo, deseja-se inserir dois itens: "Item 1" e "Item 2". Para implementá-los, tem-se o código abaixo:

menu_especifico->inserirItens(METODOX, "Item 1", "Exemplo de um item"); // o primeiro item

menu_especifico->inserirItens(METODOY, "Item 2", "Abre um quadro de diálogo específico"); // o segundo item

Vale lembrar que o projetista deve utilizar os serviços do protocolo de apresentação para desenvolver a sua interface. No apêndice B, tem-se a descrição de todos os serviços do protocolo de apresentação do ATIFS.

Finalizando, atribui-se o objeto de menu criado ao vetor *menus* e se atribui um rótulo ao vetor *rotulos:*

menu[1] = menu_especifico;
rotulos[1] = copiaString("MeuMenu");

Vale lembrar que, para definir mais um menu, basta repetir os passos descritos nesta seção criando novos objetos de menu, inserindo novos itens a estes objetos, atribuindo valores aos vetores *menu* e *rotulos* (com mais de um menu, estes vetores devem ser declarados com o tamanho correspondente à quantidade de menus definidos).

Em suma, para construir um menu opcional da aplicação, basta instanciar a classe Menu, inserir itens deste objeto instanciado e atribuí-los como sendo um elemento de um vetor.

Estes vetores são passados como parâmetros na instanciação de um objeto do Controlador. Para desenvolver um Controlador, ver seção 7.5.

Em seguida, a interface da aplicação pode ter seu(s) quadro(s) de diálogo para obter dados fornecidos pelo usuário a fim de, por exemplo, configurar a ferramenta. Para desenvolver estes quadros de diálogos, ver "Padrão 3: Implementação de um quadro de diálogo".

7.3.3 PADRÃO 3: IMPLEMENTAÇÃO DE UM QUADRO DE DIÁLOGO

Nesta seção, tem-se uma descrição de como implementar um quadro de diálogo específico à aplicação. Para atingir este objetivo, tem-se quatro etapas:

1. Instanciar um objeto da classe *QuadroDialogo*;
2. Instanciar um objeto da classe *Painel*;
3. Inserir objetos de interação deste quadro de diálogo;
4. Exibir o quadro de diálogo.

No passo 1, é preciso instanciar um objeto da classe *QuadroDialogo*, a classe que representa o quadro de diálogo. Para criá-lo, tem-se, por exemplo, um objeto de quadro de diálogo chamado *obj_qdialog* usando a seguinte linha de código (estes parâmetros estão descritos no apêndice B):

```
obj_qdialog = new QuadroDialogo(objeto_janela, "Quadro de Dialogo de Configuracao", 100, 100, 600, 400); // Instancia um objeto tipo QuadroDialogo
```

É preciso instanciar o objeto do tipo *Painel* para possibilitar a inserção de quaisquer objetos de interação disponíveis (botão, caixa de escolha, lista e caixa de texto) para construir o seu layout. Por exemplo, um objeto *meu_painel*:

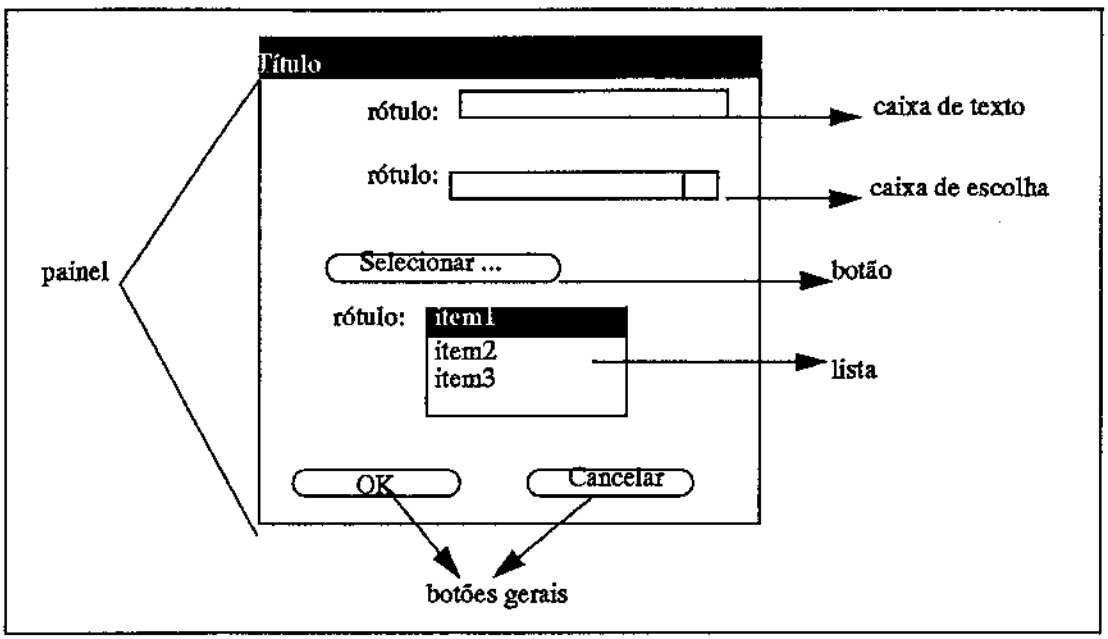
```
meu_painel = new Painel(obj_qdialog);
```

Onde *obj_qdialog* é o objeto de quadro de diálogo criado anteriormente ao qual deve ser associado ao seu painel.

A partir deste código, instancia-se objetos das classes que representam os objetos de interação que são: *Botão, Lista, CaixaEscolha* e *CaixaTexto*. Os detalhes de cada uma destas classes estão nos apêndices A e B. Por exemplo, para obter o layout mostrado na figura 7.2, basta instanciar objetos em uma classe adequada como mostra o seguinte código:

```
obj_txt = new CaixaTexto(obj, "rótulo:", 5, 10, 40, -1);
char *escolhas[4];
escolhas[0] = copiaString("Opcao1");
escolhas[1] = copiaString("Opcao2");
escolhas[2] = copiaString("Opcao3");
escolhas[3] = copiaString("Opcao4");
obj_choice = new CaixaEscolha(obj, "rótulo:", 5, 60, -1, -1, 4, escolhas);
botao = new Botao(obj, "Selecionar...", 5, 110);
lista = new Lista(obj, "rótulo:", 5, 160, -1, -1);
lista->inserirOpcao("item1");
lista->inserirOpcao("item2");
lista->inserirOpcao("item3");
```

Figura 7.2 Layout de um quadro de diálogo



Vale lembrar que todos os quadros de diálogo devem seguir o padrão de layout do ATIFS. Todos estes quadros criados a partir deste framework já possuem dois botões obrigatórios implementados: "Ok" e "Cancelar" (ver detalhes na seção 6.2.2.3). Portanto, não é preciso definir estes botões.

Para permitir que este quadro de diálogo se mantenha no padrão do ambiente ATIFS, o método da classe *Painel* chamado *fimEdição* deve ser invocado após a definição dos objetos de interação do quadro de diálogo. No caso do exemplo, tem-se a seguinte sentença:

```
meu_painel->fimEdicao(obj); // fim de edição de layout
```

Finalmente, o quadro de diálogo está implementado. Para exibí-lo, utilize o método *exibir* do objeto de quadro de diálogo, como, por exemplo:

```
obj_dialog->exibir(TRUE); // exibe o quadro de diálogo
```

Vale lembrar que, para definir mais um quadro de diálogo, basta repetir os passos descritos nesta seção criando novos objetos.

Portanto, na próxima seção, tem-se um exemplo voltado para o contexto do ATIFS a fim de mostrar como um projetista implementa a apresentação do quadro de dialogo *Palheta*.

7.3.4 EXEMPLO

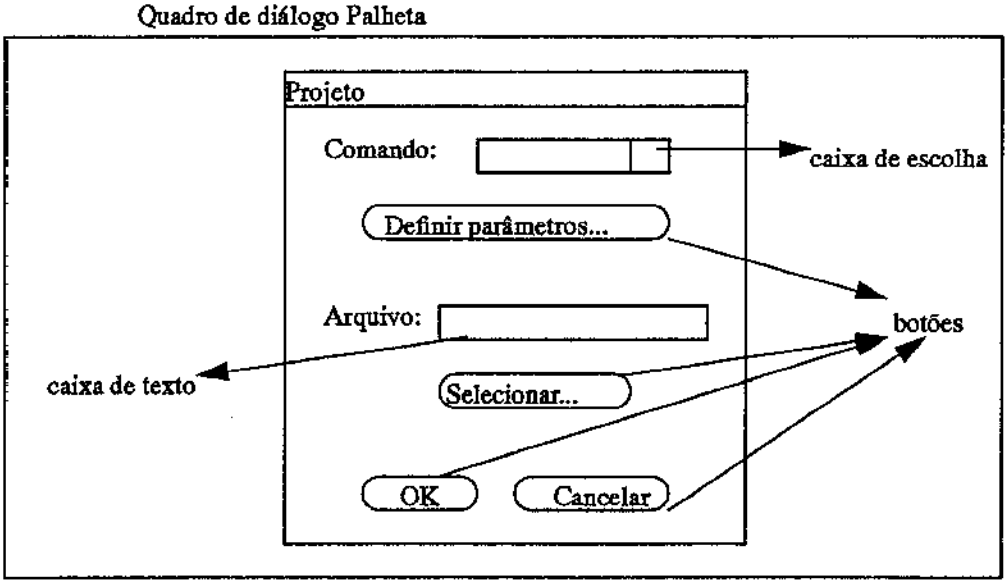
Foi visto anteriormente que, a partir da janela principal do ATIFS, é possível abrir um quadro de diálogo do tipo Palheta. Portanto, o exemplo escolhido é desenvolver a aparência da Palheta (a descrição do layout da Palheta é dada na seção 5.3.2.1).

Para implementar o layout da Palheta, o projetista deve utilizar os serviços oferecidos pelo protocolo de apresentação do ATIFS. A lista completa dos serviços está descrita no apêndice B.

Inicialmente, o projetista deve instanciar objetos de quadro de diálogo e um painel (área onde se tem um conjunto de objetos de interação) da Palheta como mostra a figura 7.3 cujos objetos de interação são:

- caixa de escolha "comando:" é um objeto da classe CaixaEscolha;
- botão "Selecionar..." - objeto da classe Botão;
- botão "Definir parâmetros..." é um objeto da classe Botão;
- caixa de texto "Arquivo:" é um objeto da classe CaixaTexto.

Figura 7.3 Quadro de diálogo Palheta



O código abaixo mostra como implementar esta Palheta:

```
palheta = new QuadroDialogo(objeto_janela, "Quadro de Dialogo de Configuracao", 100, 100, 600, 400); // instancia um objeto tipo QuadroDialogo
painel_palheta = new Painel(palheta);
char *opcoes[3];
opcoes[0] = copiaString("Editar");
opcoes[1] = copiaString("Criar novo");
opcoes[2] = copiaString("Excluir");
```

```
obj_choice = new CaixaEscolha(obj, "Comando:", 5, 10, 3, opcoes);
// aqui o projetista cria o quadro de diálogo que será exibida se o botão "Definir
Parâmetros..." for clicado - supoe-se que o objeto deste quadro de diálogo chama-se
obj_dialogo.
obj_botao1 = new Botao(obj, "Definir Parâmetros...", 20, 60);
obj_caixatexto = new CaixaTexto (obj, "Arquivo:", 5, 110, -1, -1);
obj_botao2 = new Botao(obj, "Selecionar...", 5, 170);
painel_palheta->fimEdicao(obj);
palheta->exibir(TRUE);
```

Este código é suficiente para implementar a apresentação da Palheta do ATIFS.

7.4 COMPONENTE MODELO

Neste caso, tem-se somente um único padrão que será visto no próximo item. Em seguida, tem-se o segundo item mostrando um exemplo do uso do framework para o componente Modelo no caso da Palheta.

7.4.1 PADRÃO 1: PROPÓSITO

Este framework foi projetado para implementar o componente Modelo do ATIFS. Neste componente, tem-se somente uma classe abstrata a fim de oferecer primitivas (serviços genéricos) para as classes que comporão este componente.

Para implementar o componente Modelo do ATIFS, basta criar subclasse(s) da classe Modelo. Esta(s) subclasse(s) obterão automaticamente todas as primitivas de serviços graças ao mecanismo de herança. Esta classe, portanto, deve ser instanciada a um objeto no método *inicialização* na classe Aplicação.

A classe Modelo pode ter mais de uma subclasse (não há limite máximo quanto ao número de subclasses) e também, é permitido que alguns métodos herdados sejam sobrepostos.

É bastante simples implementar o componente Modelo mas vale lembrar que este componente deve seguir rigorosamente as normas descritas na seção 6.3.1.

7.4.2 EXEMPLO

Com base no exemplo do quadro de diálogo Palheta, inicialmente, deve-se determinar qual classe do Modelo que a Palheta deve usar.

A classe escolhida é *Ambiente*. Esta classe (ilustrada na figura 7.4) representa o ambiente ATIFS como um todo. Esta classe possui os seguintes atributos:

- *lista_projetos* - contém uma lista de projetos que compõem o ambiente;

- *lista_editores* - contém uma lista de editores integrados no ambiente;
- *lista_ferramentas* - lista de ferramentas integradas no ATIFS.

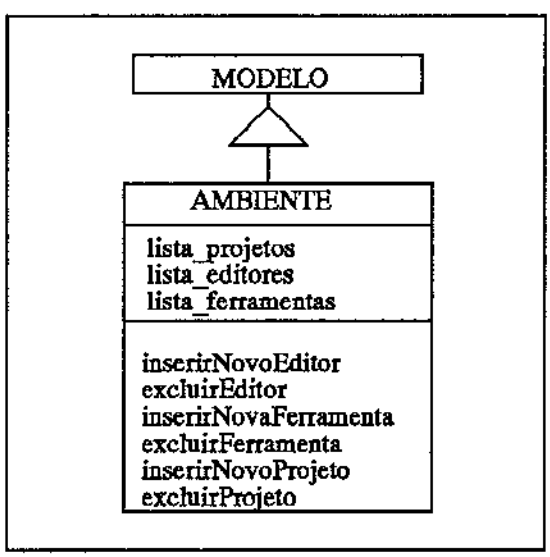
Esta classe contém uma série de serviços que são:

- *inserirNovoProjeto* - insere (ou integra) um novo projeto no ambiente;
- *excluirProjeto* - exclui o projeto do ambiente;
- *inserirNovoEditor* - insere um novo editor;
- *excluirEditor* - exclui um editor;
- *inserirNovaFerramenta* - insere nova ferramenta;
- *excluirFerramenta* - exclui uma ferramenta;

Vale mencionar que esta classe está sendo apresentada de forma simplificada pois esta está fora do escopo deste trabalho.

Esta classe deve ser implementada pelo projetista definindo seus atributos e métodos citados anteriormente. Aqui não será apresentado o código fonte pois a intenção é mostrar o uso desta classe sem detalhes de implementação.

Figura 7.4 Classe pertencente ao Modelo



Finalmente, no método *inicialização* da classe Aplicação, tem-se uma instancialização de um objeto do Modelo. No caso do exemplo, tem-se:

objeto_modelo = new Ambiente;

7.5 COMPONENTE CONTROLADOR

Neste caso, tem-se somente dois padrões que serão vistos no próximo item. Em seguida, tem-se o segundo item mostrando um exemplo do uso do framework para o componente Controlador no caso da Palheta.

7.5.1 PADRÃO 1: PROPÓSITO

Este framework foi projetado para implementar o Controlador que gerenciará a interação entre componentes Visão e Modelo do ATIFS. Para implementar o Controlador, tem-se a descrição na próxima seção.

7.5.2 PADRÃO 2: IMPLEMENTAÇÃO DO CONTROLADOR

Para implementar um Controlador, deve-se criar uma subclasse da classe VControladorFerramenta (quando a aplicação é uma ferramenta funcional) ou VControladorEditor (quando a aplicação é um editor) ou VControladorJPrincipal (quando a aplicação é um ambiente).

No Controlador, os métodos devem ser definidos sobrepondo os métodos virtuais da sua superclasse a fim de definir o seu comportamento. Os métodos que podem ser definidos são: *comandosDeMenuEspecifico*, *comandosDeMenuObrigatório*, *qDialogoNoOk*, *choiceBoxCallBack*, *listaCallBack*, *áreaOpcoesCallBack* e *botãoCallBack*.

A diferença de cada método é simplemente temporal, o momento de chamada, os métodos são invocados somente após um determinado evento ocorrido. Por exemplo, o método *comandosDeMenuEspecifico* será invocado somente quando o usuário seleciona uma opção de um menu opcional da aplicação. A identificação do item de menu selecionado é obtida como parâmetro do método permitindo que o Controlador determine qual item de menu foi selecionado. Assim, é possível implementar um comportamento para cada item de menu.

Já o método *comandosDeMenuObrigatório* é invocado somente quando um item de menu obrigatório é selecionado pelo usuário juntamente com a identificação do item de menu selecionado.

O método *qdialogoNoOk*, por exemplo, é invocado somente quando ocorre um clique em um botão "Ok" do quadro de diálogo implementado pelo projetista. O projetista, portanto, pode obter a identificação do quadro de diálogo e implementar a resposta adequada para cada quadro de diálogo implementado.

As formas de implementar estes métodos são idênticas, estes podem ser preenchidos com códigos da linguagem de programação ANSI C++, chamadas de serviços do protocolo de apresentação (ver apêndice B), do componente Modelo ou até mesmo do próprio Controlador.

Após a definição de métodos, vale lembrar que esta classe controladora deve ser instanciada passando alguns parâmetros importantes (ver apêndice B) durante a inicialização da aplicação (foi visto na seção 7.3.1).

7.5.3 EXEMPLO

No contexto da Palheta, tem-se, então, uma nova classe que faz o papel do Controlador que será chamada simplesmente de "Controlador".

Vale lembrar que a implementação do Controlador para a palheta é uma parte da implementação do Controlador do ambiente pois foi visto que a Palheta é considerada como um quadro de diálogo específico ao ambiente.

Para implementar o Controlador, o projetista deve obter a classe "Controlador" derivando a classe *VControladorAmbiente* existente no protocolo de apresentação.

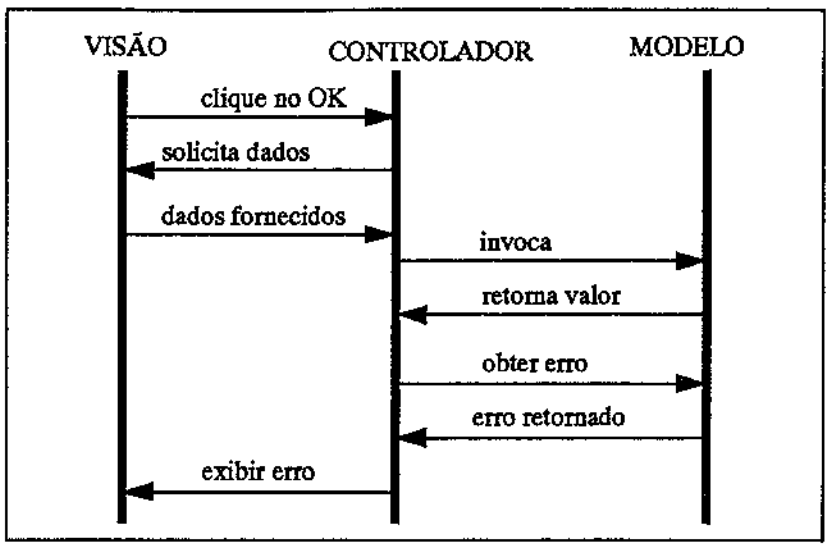
Criada a classe controladora, o projetista deve definir o método *qdialogoNoOk*. Já que este método será invocado somente quando o usuário clica no botão Ok da palheta. Este método deve obter dados mantidos (fornecidos pelo usuário) pelos objetos de interação que compõem este quadro de diálogo invocando os serviços do próprio protocolo de apresentação.

Após a obtenção dos dados dos objetos de interação, estes valores são passados para um objeto da classe *Ambiente* do componente Modelo usando o protocolo definido pelo componente Modelo.

A classe *Ambiente*, por sua vez, pode retornar algum valor inteiro. O Controlador pode determinar se o valor retornado indica que a execução foi normal ou ocorreu algum erro.

Se ocorreu algum tipo de erro, o Controlador solicita a identificação do erro ocorrido invocando um serviço do Modelo. Obtida esta identificação, então o Controlador solicita ao componente Visão para exibição de erro. Para ilustrar este comportamento, a figura 7.5 mostra o traço de eventos (notação da metodologia OMT) dos três componentes: Visão, Controlador e Modelo.

Figura 7.5 Traço de eventos entre Visão, Controlador e Modelo



Para exemplificar, supõe-se que o nome do objeto instanciado da classe do Modelo (classe *Modelo*) é *objeto_do_modelo*.

Supõe-se também que a variável *contexto* é uma cadeia de caracteres que representa o contexto da Palheta (ver seção sobre janela principal na seção 5.3.2.1). Os códigos para cada contexto da Palheta são análogos. A seguir, tem-se o código somente no contexto de projeto, o usuário pode fornecer comandos, por exemplo, para excluir ou incluir um projeto mantido pelo ATIFS.

```
void Controlador::qdialogoNoOk (QuadroDialogo *obj)
{
char *comando;
char *arquivo;
char *erro;
int status;
comando = copiaString(meu_painel->obj_choice->obterOpcaoSelecionada());
arquivo = copiaString(meu_painel->obj_caixatexto->obterConteudo());
if (comparaString(contexto, "Projeto")) // verifica se o contexto é Projeto
{
if (comparaString(comando, "excluir"))
{
 status = objeto_do_modelo->excluirProjeto(arquivo);
 if (status == erro)
    {
    erro = objeto_do_modelo->obterErro();
```

```
        exibirAlerta(erro, 1);
      }
}
if (comparaString(comando, "inserir"))
{
 status=objeto_do_modelo->inserirNovoProjeto(arquivo);
 if (status == erro) // se erro...
 {
      erro = objeto_do_modelo->obterErro(); // obter o erro
      exibirAlerta(erro, 1); // exibe uma alerta
 }
}
...
}
...
}
```

É importante lembrar que o Controlador pode definir os *call-backs* dos objetos de interação. No caso do ATIFS, é possível definir os *call-backs* dos objetos da classe *Lista*, *CaixaEscolha* e *Botão*.

No exemplo, é preciso definir um *call-back* do objeto do tipo *CaixaEscolha* que obtém o comando selecionado pelo usuário pois se o usuário selecionar o comando "excluir", o botão "Definir Parâmetros..." da Palheta deve ser desabilitado impossibilitando que o usuário especifique os parâmetros; caso contrário, este botão deve ser habilitado.

Para isto, o projetista deve definir o método do Controlador chamado *caixaEscolhaCallBack* pois este é invocado pela Visão somente quando o usuário seleciona alguma opção em uma caixa de escolha. A seguir, tem-se o código que descreve este comportamento:

```
void Controlador::caixaEscolhaCallBack(CaixaEscolha *obj)
{
char *comando;
comando = meu_painel->obj_choicebox->obterOpcaoSelecionada(); // obtém a opção selecionada pelo usuário na caixa de escolha.
if (comparaString(comando, "excluir")) // se comando é "excluir"...
      meu_painel->botao1->habilita(FALSE); // desabilita o botão
else
      meu_painel->botao1->habilita(TRUE); // habilita o botão
}
```

Além disso, o projetista deve definir os call-backs dos botões da Palheta. Se o botão "Definir Parâmetros..." for clicado, exibe-se um quadro de diálogo, por exemplo, *obj_dialog2* com o painel *meu_painel2*.

Se o botão "Selecionar..." for clicado, exibe-se um seletor de arquivo (ver detalhes na seção 5.3.2.3). Dado o arquivo selecionado (pelo usuário), este arquivo deve ser exibido pela caixa de texto.

Para implementar este comportamento, o método de call-back de botão deve ser definido como mostra a seguir:

```
Controlador::callBackBotao (Botao *obj)
{
if (obj == obj_botao1)
  {
    obj_dialog2 = new QuadroDialogo(objeto_janela, "Parametros", 250, 250, 600, 400);
    meu_painel2 = new Painel(obj_dialog2);
    desenharqdialogo2(obj_dialog2);
    meu_painel2->fimEdicao(obj_dialog2);
    obj_dialog2->exibir(TRUE);
  }
 if (obj == botao2)
 {
  char *auxiliar = "                                    ";
  auxiliar = copystring("                                    ");
  auxiliar = selecionarArquivo(1, NULL,"*.*");
  obj_txt->exibirConteudo(auxiliar);
 }
}
```

Finalmente, no método *inicialização* da classe Aplicação, tem-se uma instanciação de um objeto do Controlador. No caso do exemplo, tem-se:

```
obj_controlador = new Controlador("ATIFS", "*.*", "help1.hlp", "help2.hlp", "ATIFS -
Protótipo versão 1.0 \n Marcos Salenko Guimarães \n Março de 1995", 1, menu, rotulo);
```

Os detalhes dos parâmetros de configuração do Controlador estão no apêndice B.

Vale lembrar que se deve criar dois arquivos de texto em ANSI puro (sem sinais de controle): um arquivo trata da aplicação e o segundo trata de como operar o sistema. Isto permite que a aplicação ofereça todas as informações de ajuda ao usuário.

No caso do exemplo, ter-se-ia um arquivo "help1.hlp" descrevendo a aplicação e o segundo "help2.hlp" descrevendo como operar esta aplicação.

Isto é tudo para obter uma noção básica de implementação de um Controlador.

7.6 CONCLUSÃO

No caso do framework do ATIFS, o componente Visão é do tipo caixa-preta e os componentes Modelo e Controlador são do tipo caixa-branca.

Foi visto também que existem várias técnicas para descrever um framework que são chamadas de *padrões de projeto*. No contexto deste capítulo, foi utilizado o padrão de projeto do tipo *guia de programação*.

Este tipo de padrão de projeto permite documentar o framework usando um conjunto de padrões que descrevem o propósito, os problemas e como resolvê-los usando o framework. Este padrão de projeto foi utilizado para descrever o framework do ATIFS.

Foi visto também o exemplo de uso deste framework no contexto do ATIFS no que se refere ao quadro de diálogo Palheta. Portanto, este capítulo mostrou a implementação do quadro de diálogo Palheta e como os três componentes interagem neste caso.

Descrever um framework é uma área que ainda está em evolução. Não há um estudo que comprove qual é o padrão de projeto que descreve melhor um framework.

Neste trabalho, foi usado o padrão *Guia de Programação* por ser mais conhecido para descrever como usar um framework.

Após a descrição do uso do framework, tem-se encerramento final deste trabalho no próximo capítulo.

Capítulo 8

CONCLUSÃO E EXTENSÕES FUTURAS

Este capítulo encerra este trabalho com os seguintes tópicos: na primeira seção, tem-se o que foi feito neste trabalho. Na próxima seção, tem-se a descrição da sua situação atual. Finalmente, este capítulo encerra com as extensões futuras para a continuidade deste trabalho.

8.1 O QUE FOI FEITO?

Esta dissertação teve por objetivo auxiliar os projetistas de ferramentas no desenvolvimento de uma interface com o usuário de um ambiente integrado de testes.

No capítulo 2, foi feito um estudo de forma genérica sobre ambientes integrados e interface com o usuário a fim de obter uma base teórica e principalmente, os requisitos da interface a fim de atender os aspectos de integração de apresentação de ambiente.

O capítulo 3 descreve o ambiente considerado neste trabalho denominado ATIFS mostrando alguns conceitos e a sua arquitetura.

Foi escolhido o paradigma orientado a objetos pois atende melhor aos requisitos apresentados na seção 2.2, quais sejam: uniformidade, separação de interface e aplicação, portabilidade e extensibilidade.

No capítulo 4, foi visto que os frameworks são modelos que representam uma estrutura de projeto contendo classes abstratas e concretas juntamente com os seus relacionamentos. Estes frameworks possuem os seguintes aspectos de desenvolvimento:

- *projeto* - projeta-se um modelo que deve ser reutilizável cobrindo um conjunto de domínios do problema;
- *uso* - o framework deve ser usado para desenvolver uma aplicação. Foi visto que existem duas formas de uso: o de caixa-preta (uso baseado em instanciações de classes) e de caixa-branca (uso baseado na criação de novas subclasses a partir de uma classe).
- *descrição* - o framework pode ser documentado usando *padrões de projeto* a fim de especificá-lo, descrever como usá-lo e outros objetivos. Existem vários padrões. Foi visto que, no caso do ATIFS, foi utilizado o padrão do tipo *guia de programação* para descrever como usar o framework do ATIFS.

No contexto de interface, foram estudados vários modelos para construção de interface com o usuário: MVC, Modelo baseado em Modos, MVC++, MV, Modelo de Camadas; com o intuito de escolher o mais adequado para o ATIFS. Dentre os modelos estudados, o MV e o MVC++ foram considerados como os que melhor poderiam atender aos requisitos de interface para o ATIFS.

Inicialmente, foi projetado um framework da interface com o usuário do ATIFS com base no MV (visto na seção 4.2.2.5). Foi constatado que o MV demonstrou a sua deficiência que é a *forte dependência do componente Visão ao Modelo.*

Quando o Modelo é alterado, o componente Visão pode sofrer alterações (dependendo do tipo de alterações do Modelo). Isto não ocorre no MVC++, uma vez que o Modelo sofre alterações, o componente Controlador é quem sofre esta influência evitando, assim, alterações no componente Visão.

Este problema do MV está no componente Visão, o qual não possui uma separação explícita do contexto genérico (que representa o padrão de *look and feel* da interface do ATIFS) e o contexto específico de uma ferramenta (interface no contexto específico de uma ferramenta). No MVC++, o componente Visão representa a interface nos aspectos independente de uma ferramenta e o Controlador representa a interface no contexto específico de uma ferramenta.

Portanto, o MVC++ (*Modelo-Visão-Controlador++*) foi escolhido por fornecer maior grau de reutilização e de independência de diálogo dentre os modelos estudados.

Com base no MVC++, foi projetado um framework específico da interface com o usuário do referido ambiente integrado. Desta forma, os projetistas de ferramentas podem desenvolver uma interface para suas ferramentas partindo desta estrutura pré-definida.

Projetado o framework, é preciso documentar como usá-lo para que os projetistas de ferramentas possam ter conhecimento de como definir a sua interface da sua ferramenta. Este processo foi documentado no capítulo 7 usando *padrão de projeto* **[Pre95b]** do tipo *guia de programação* juntamente com alguns exemplos.

O framework definido para o ATIFS foi implementado em ANSI C++ de forma que os projetistas poderão instanciar as classes referentes à interface das suas ferramentas. Desse modo, garante-se que todas as ferramentas integradas seguirão o padrão de comportamento e apresentação do ATIFS.

8.2 O QUE DEVE SER FEITO?

Este framework deve sofrer o processo de "maturação" que é obtido através de aplicações práticas realizadas pelos próprios projetistas de ferramentas. Desta forma, podem surgir problemas no framework, seja de projeto, seja de implementação. Portanto, estes problemas farão com que o framework seja modificado a fim de otimizá-lo.

8.3 EXTENSÕES FUTURAS

A possibilidade de melhoria quanto a implementação do framework é a utilização do mecanismo de reflexão computacional oferecendo assim uma separação mais nítida da interação entre componentes Visão e Controlador. O Controlador pode ser um meta-objeto da Visão e Modelo.

Foi constatado que o Protocolo de Apresentação do ATIFS é totalmente dependente do toolkit utilizado. Portanto, a outra possibilidade de melhoria é permitir que a camada do Protocolo de Apresentação seja independente do toolkit. Assim, não haveria necessidade de re-projetar esta camada ao substituir um toolkit por um outro.

Como extensão de trabalho, seria possível tornar o framework do ATIFS totalmente do tipo caixa-preta, no qual os componentes Visão, Modelo e Controlador seriam obtidos simplesmente instanciando classes e configurando os objetos.

Isto não foi possível pois o componente Modelo do ambiente como um todo não estava totalmente especificado, o ATIFS ainda está em desenvolvimento. Quando o componente Modelo estiver totalmente definido, o framework para o componente Modelo pode ser projetado e pode ser tornar um framework do tipo caixa-preta.

O desafio é transformar o framework do componente Controlador em caixa-preta. Este é considerado como uma parte bastante difícil de ser reutilizável pois este implementa o diálogo entre Modelo e Visão, sendo portanto muito dependente da ferramenta que está sendo implementada.

Se este desafio for superado, seguindo o raciocínio do [JF88], como o framework do ATIFS é totalmente do tipo caixa-preta, o que falta é somente obter dados de configuração de objetos. Desta forma, seria possível desenvolver quaisquer sistemas interativos que compõem o ambiente integrado de forma automática. O projetista de ferramenta, por exemplo, utilizaria um software que, obtidos alguns dados fornecidos

pelo projetista, geraria um código da sua ferramenta automaticamente seguindo todos os padrões estabelecidos pelo ambiente.

Após o processo de "otimização" do framework, como extensão futura deste trabalho, deve-se desenvolver uma ferramenta interativa de construção de interfaces. Uma das utilidades do framework é permitir o desenvolvimento de um gerador de aplicações. Portanto, é possível desenvolver uma ferramenta interativa de construção de interface que automatiza o trabalho dos projetistas de ferramentas no desenvolvimento de interface. Assim, não é necessário que o projetista codifique manualmente a sua ferramenta pois a ferramenta interativa de construção de interface é quem geraria este código automaticamente.

REFERÊNCIAS BIBLIOGRÁFICAS

[BC91] Len Bass and Joelle Coutaz, "Developing Software for User Interfaces", Addison-Wesley Publishing Company, Inc., 1991.

[BFW92] Alan W. Brown, Peter H. Feiler, Kurt C. Wallnau, "Understanding Integration in a Software Development Environment", Technical Report CMU/SEI-91-TR-31, January 1992.

[CN92] Minder Chen, Ronald J. Norman, "A Framework for Integrated CASE", IEEE Software, March 1992.

[Coa92] Peter Coad, "Object-Oriented Patterns", Communications of the ACM, 35(9).

[Col95] Dave Collins, "Designing Object-Oriented User Interfaces", Benjamin-Cummings, 1995.

[Cop92] J. O. Coplien, "Advanced C++ Programming Styles and Idioms". Reading, Massachusetts: Addison-Wesley.

[CPP92] Matthew Conway, Randy Paush, Kimberly Passarella, "A Tutorial for SUIT - The Simple User Interface Toolkit", Tutorial, Computer Science Department, University of Virginia, 1992.

[Ege91] R.K. Ege, "Improving Object-Oriented User Interfaces with Constraints", Information and Software Technology, Vol. 33, nº 2, March, 1991.

[Fer89] Jacques Ferber, "Computational Reflection in Class based Object Oriented Languages", OOPSLA'89 Proceedings, October 1-6, 1989.

[GHJ+93] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns: Abstraction and Reuse of Object-Oriented Design". Proceedings of the ECOOP'93 Conference, Springer Verlag.

[Har89] Rex Hartson, "User-Interface Management Control and Communication", IEEE Software, January 89, pg. 62 - 70.

[HH89] H.Rex Hartson and Deborah Hix, "Human-Computer Interface Development: Concepts and Systems for its Management", ACM Computing Surveys, 21(1):5-92, March 1989.

[HH93] H.Rex Hartson and Deborah Hix, "Developing User Interfaces - Ensuring Usability Through Product & Process", John Wiley & Sons, Inc., 1993.

[HRG90] R. Helm, I. M. Rolland, D. Gangopadhyay, "Contracts: Specifying Behavioral Compositions in Object-Oriented Systems", Proceedings of OOPSLA'90, Ottawa, Canada.

[Jaa95a] Ari Jaaksi, "Implementing Interactive Applications in C++", Software Practice & Experience, March 1995.

[Jaa95b] Ari Jaaksi, "Object-Oriented Specification for User Interfaces", Software Practice & Experience, November 1995.

[JF88] Ralph E. Johnson and Brian Foote, "Designing Reusable Classes", JOOP Vol.1, no. 2, June/July 1988.

[Joh92] Ralph E. Johnson, "Documenting Frameworks using Patterns", OOPSLA'92, pp. 63-76.

[KZ90] S.Konecni, I. de Zegher, "Object-Oriented Programming & the Design of an Open System for User Interfaces", Proceedings of the Second International Conference TOOLS (Tecnology of Object-Oriented Languages and Systems), Paris, 1990, pp. 553-563.

[LB91] Chris Lafra and Jan Van den Bos, "A Layered Object-Oriented Model for Interaction". No capítulo 4 do livro editado por E.H.Blake, P. Wisskirchen, "Advances in Object-Oriented Graphics I", Springer-Verlag 1991.

[LPS+89] Julius Leite, Alderico R. de Paula, Jorge M. de Souza, Marta Martini, Orlando G. Loques Filho, "Fundamentos da Computação Tolerante a Falhas", mini-curso do III Simpósio de Computadores Tolerantes a Falhas - 20 a 22 de setembro de 1989 na PUC/RJ.

[Luc93] Fábio Nogueira de Lucena, "Construção de Interfaces Homem-Computador" - Tese de Mestrado - Março de 1993 - DCC/IMECC/UNICAMP

[LVC89] M.A. Linton, J.M. Vlissides, P.R.Calder, "Composing User Interface with InterViews", IEEE Computer, Fevereiro/89, pp 8-22.

[Mae87] Pattie Maes, "Concepts and Experiments in Computational Reflection", OOPSLA'87 Proceedings, October 4-8, 1987.

[Mar93] Eliane Martins, "Validação experimental da tolerância a falhas: a técnica de injeção de falhas", Minicursos do V. SCTF, ITA, 1993.

[MEC93] MECB/INPE, "Documento de Concepção do SCOM-X.25", 1993 (a ser publicado)

[MS92] Peiwei Mi and Walt Scacchi, "Process Integration in CASE Environments", IEEE Software, March 1992.

[Mye94] Brad A. Myers, "Challenges of HCI Design and Implementation", Interactions, January 94, pg. 73 - 83

[NO93] F. Neelamkavil and S. O'Shea, "Using PCTE with X-based user interface development tools", Information and Software Technology, January 1993, Butterworth-Heinemann, pp 17 - 22.

[OCM95] Juliano Lopes de Oliveira, Cleida Queiroz Cunha e Geovane Cayres Magalhães, "Modelo de Objetos para Construção de Interfaces Visuais Dinâmicas", SBES'95, outubro de 1995.

[Ous94] John K. Ousterhout, "Tcl and the Tk Toolkit", Addison-Wesley Professional Computing Series, ISBN 0-201-63337-X, 1994.

[Pre95a] Roger S. Pressman, "Engenharia de Software", Mc Graw Hill, 1995.

[Pre95b] Wolgang Pree, "Design Patterns for Object-Oriented Software Development", Addison Wesley, 1995.

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy and William Lorensen, "Object-Oriented Modeling and Design", Prentice-Hall International Editions, 1991.

[Rub94] Cecília Mary Fisher Rubira, "Structuring Fault-Tolerant Object-Oriented Systems Using Inheritance and Delegation", PhD Thesis, Dept. of Computing Science, University of Newcastle upon Tyne, October 1994.

[Sch86] "MacApp: An Application Framework", Byte, August 1986, pp. 189-193.

[Sha91] Yen-Ping Shan, "An Object-Oriented Framework for Direct-Manipulation User Interfaces. No capítulo 1 do livro editado por E.H.Blake, P. Wisskirchen, "Advances in Object-Oriented Graphics I", Springer-Verlag 1991.

[Sma94] Julian Smart, "User Manual for WXWindows: a Portable C++ GUI Toolkit", October, Artificial Intelligence Institute, University of Edinburg, 1994.

[Som95] Ian Sommerville, "Software Engineering", Fifth Edition, no capítulo 25, Addison-Wesley, 1995.

[TBC+88] Richard N. Taylor, Frank C. Belz, Lori A. Clarke, Leon Osterweil, Richard W. Jelby, Jack C. Wileden, Alexander L. Wolf, Michael Young, "Foundations for the Arcadia Environment Architecture", Proceedings of the ACM Software Engineering Symposium on Practical Software Development Environments, Boston-Massachussets, November, 1988.

[TNB+94] Richard N. Taylor, Kari A. Nies, Gregory A. Bolcer, Craig A. MacFarlane, Gregory F. Johnson, Kenneth M. Anderson, "Supporting Separations of Concerns and Concurrency in the Chiron-1 User Interface System", UCI Technical Report 94-12, March 1994.

[TNB+95] Richard N. Taylor, Kari A. Nies, Gregory A. Bolcer, Craig A. MacFarlane, Gregory F. Johnson, Kenneth M. Anderson, "Chiron-1: A Software Architecture for User Interface Development, Maintenance, and Run-Time Support", ACM Transaction on Computer-Human Interaction, Vol.2, No.2, June 1995, pp. 105-144.

[TR94] Guilherme Horta Travassos e Ana Regina Cavalcanti da Rocha, "Um Modelo para Construção e Integração de Ferramentas", 8º Simpósio Brasileiro de Engenharia de Software, Curitiba-PR, 25 a 28 do outubro de 1994.

[Vuo93] S.T.Vuong, "Protocol conformance testing" em Tutoriais e Mini-cursos do SBRC-11, Campinas, SP, 1993.

[WGM88] André Weinand, Erich Gamma, Rudolf Marty, "ET++ - An Object-Oriented Application Framework in C++", OOPSLA'88 Proceedings, September 25-30, 1988.

[WJ90] Rebecca J. Wirfs-Brock and Ralph E. Johnson, "Surveying Current Research in Object-Oriented Design", Communication of the ACM, Setembro de 1990.

[Wol92] Mario Wolczko, "Encapsulation, delegation, and inheritance in Object-oriented languages", Software Engineering Journal, March 1992.

[YTT88] Michael Young, Richard N. Taylor and Dennis B. Troup, "Software Environment Architectures and User Interface Facilities", IEEE Transactions on Software Engineering, vol. 14, no. 6, June 1988.

ANEXOS

Esta parte contém dois apêndices, são:

- Apêndice A - Framework da Interface do ATIFS

- Apêndice B - Serviços Implementados do Framework..

Apêndice A

FRAMEWORK DA INTERFACE DO ATIFS

Este apêndice apresenta o modelo de objetos do Protocolo de Apresentação do ATIFS contendo basicamente quatro seções: uma menciona o modelo de objetos do componente Visão do ATIFS; a segunda descreve o modelo de objetos do Controlador; na terceira seção, tem-se um modelo de objetos do componente Modelo e, finalmente, tem-se uma descrição da classe *Aplicação* (classe requerida pelo *WXWindows*, representa uma ferramenta ou um ambiente ou um editor como um todo sendo útil para inicializar qualquer uma destas aplicações).

Há duas ressalvas importantes sobre este apêndice, que são:

1. os métodos das classes referentes aos componentes Visão e Controlador, não serão descritos pois, estes se encontram no apêndice B (sobre os serviços do protocolo de apresentação).
2. algumas classes implementadas possuem o seu comportamento definido no protocolo de apresentação. Portanto, tem-se o modelo dinâmico (usando a notação da metodologia OMT) destas classes a fim de descrever o seu comportamento.

Algumas classes possuem o seu modelo dinâmico e outras não. Isto não significa que a especificação do comportamento dinâmico do protocolo de apresentação do ATIFS esteja incompleta pois as classes do Protocolo de Apresentação são derivadas a partir de uma classe da biblioteca do *toolkit* WXWindows (ver a seção 6.2.1). Graças ao mecanismo de herança, o comportamento destas classes estão encapsulados nas classes do *toolkit*. Portanto, as classes que não possuem o seu modelo dinâmico somente implementam o protocolo e as demais, além de implementar o protocolo, possuem o seu comportamento específico (ou adicional).

A seguir, tem-se então, a descrição das classes do protocolo de apresentação do ATIFS.

A.1 COMPONENTE VISÃO

O modelo de objetos do componente Visão do ATIFS compõe-se de várias classes que estão ilustradas na figura A.1 (usando a notação da metodologia OMT) e do *toolkit* WXWindows. A seguir, tem-se itens referentes a classes do Protocolo de Apresentação.

A.1.1 *Janela*

A classe Janela representa uma janela de forma genérica contendo somente uma barra de menu. Esta classe é subclasse de uma classe oferecida pela biblioteca do toolkit: o wxFrame, contendo vários atributos adicionais:

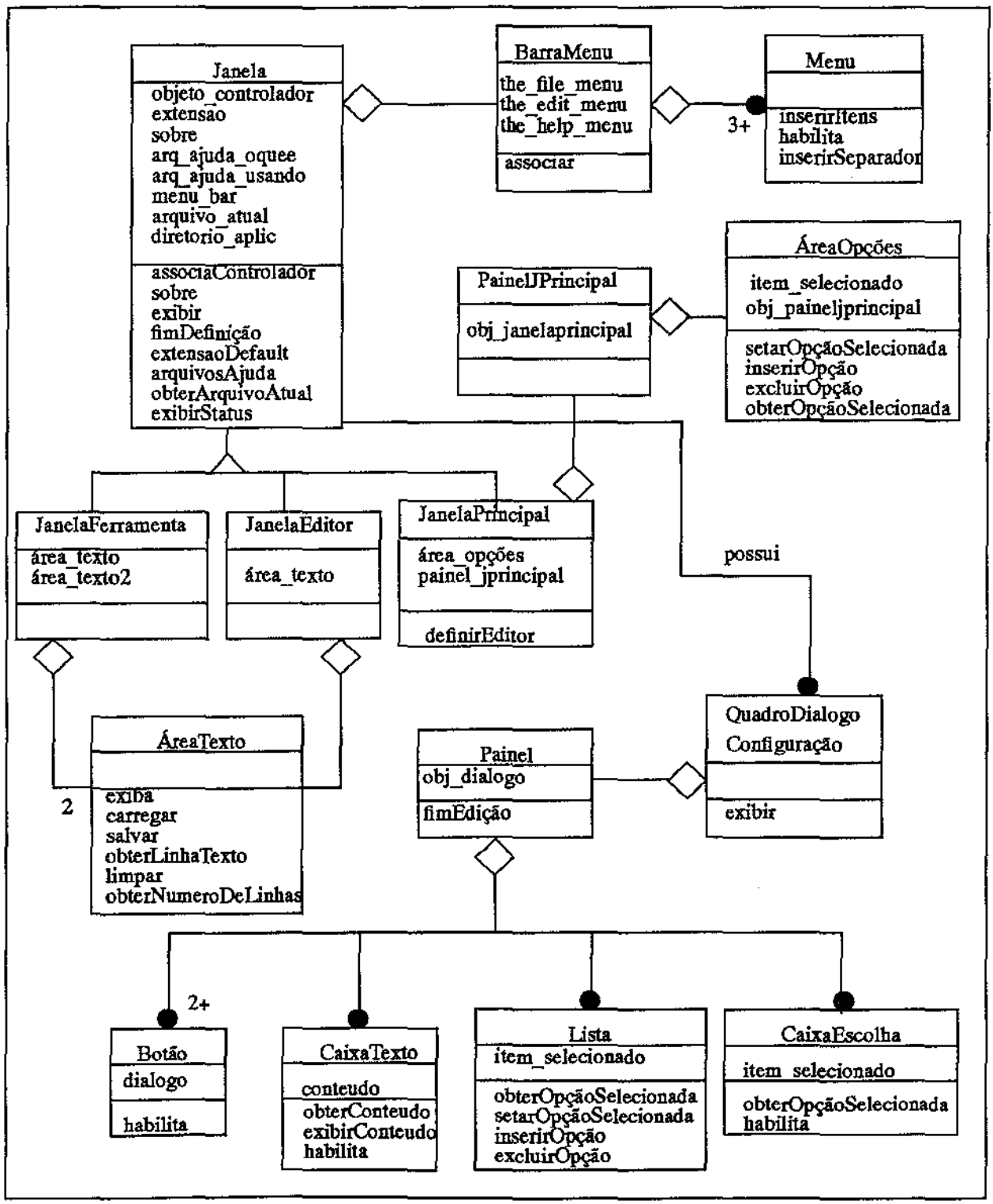
1. *sobre_aplic* - mantém uma cadeia de caracteres mantendo dados da aplicação fornecidos pelo projetista. Esta cadeia de caracteres será exibida somente quando o usuário seleciona o item de menu "Sobre" no menu "Ajuda". Selecionada esta opção, abre-se um quadro de diálogo contendo esta cadeia de caracteres.
2. *nome_aplic* - nome da aplicação, atributo do tipo cadeia de caracteres (ou string).
3. arq_ajuda_oquee - atributo do tipo cadeia de caracteres representando nome do arquivo (fornecido pelo projetista) de ajuda que mantém informação sobre a aplicação.
4. *arq_ajuda_usando* - atributo do tipo cadeia de caracteres representando o nome do arquivo (fornecido pelo projetista) de ajuda que referencia o como usar a aplicação.
5. *objeto_controlador* - ponteiro para o seu objeto Controlador instanciado a partir da classe controladora definida pelo projetista.
6. *diretorio_aplic* - diretório da aplicação do tipo cadeia de caracteres.
7. *arquivo_atual* - arquivo atual aberto do tipo cadeia de caracteres.
8. *menu_bar* - ponteiro para o seu objeto de barra de menu da classe BarraMenu (será vista na próxima seção).

A.1.2 *BarraMenu*

Esta classe representa um conjunto de menus denominado *barra de menu.* Esta classe é derivada a partir da classe *wxMenuBar* da biblioteca do *WxWindows* contendo três atributos específicos, que são:

1. *the_file_menu* - ponteiro para um menu obrigatório *Arquivo* da classe *Menu.* Este ponteiro é importante para uso interno do protocolo de apresentação.
2. *the_edit_menu* - ponteiro para menu *Editar* da classe *Menu,* importante para uso interno do protocolo de apresentação.
3. *the_help_menu* - ponteiro para o menu *Ajuda* da classe *Menu.*

Figura A.1 Modelo de objetos do Protocolo de Apresentação



A.1.3 JanelaPrincipal

Esta classe representa a janela principal do ATIFS ilustrada na figura 5.1. Note que esta classe é uma especialização da classe Janela. Portanto, todos os atributos e métodos são automaticamente herdados pela classe Janela. A seguir, tem-se os seus atributos específicos, que são:

1. *painel_jprincipal* - ponteiro que aponta para um objeto da classe PainelJPrincipal (esta classe será descrita posteriormente).
2. *área_opções* - ponteiro para objeto da classe ÁreaOpções (esta classe também será descrita posteriormente).

Esta janela foi implementada também toda a sua parte dinâmica (comportamental). Para especificar este comportamento, tem-se o seu modelo dinâmico na figura A.2.

Inicialmente, como todas as janelas do ATIFS, tem-se o estado de "inicialização", durante a sua inicialização, cria-se os seus objetos componentes da janela, acrescenta-se os seus menus e, finalmente, a janela é exibida.

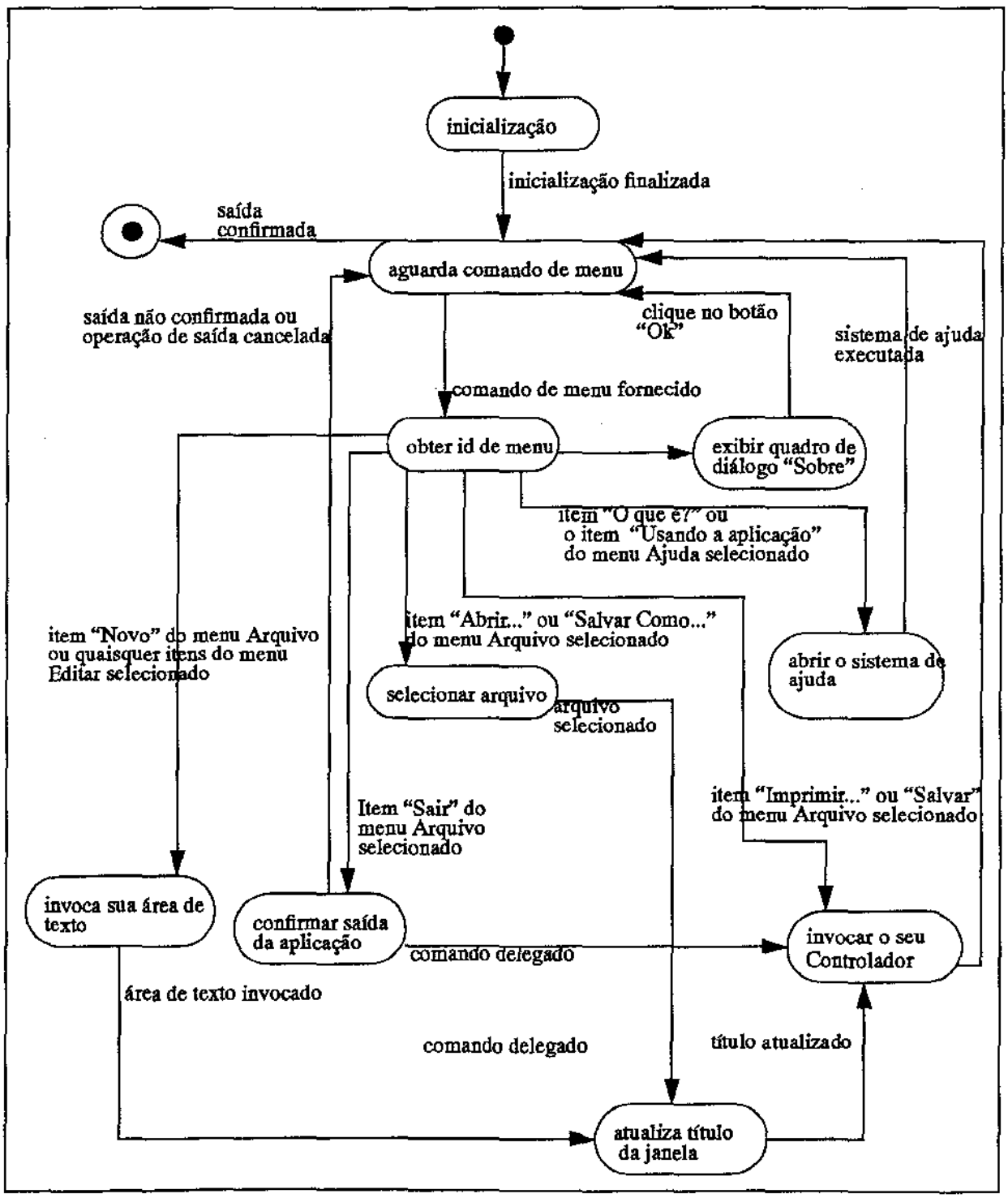
Exibida a janela, esta permanece em um estado de espera de eventos no estado "aguarda comando de menu". Gerado um evento para esta janela, esta passa para o estado de avaliação do evento (no estado "obter id de menu") determinando o evento e a sua resposta.

No caso do protocolo de apresentação do ATIFS, o evento esperado é o comando de menu selecionado pelo usuário. A resposta da janela dependerá do item de menu selecionado. Portanto, a partir do estado "obter o id de menu", tem-se, na figura A.2, os estados que representam as respostas de cada item de menu selecionado.

Por exemplo, se o usuário seleciona a opção "Novo" do menu Arquivo, o próximo estado é o estado "invocar um editor". Isto significa que um editor será carregado a fim de criar um novo arquivo. Em seguida, tem-se o estado "invocar o seu Controlador", um método do Controlador é invocado.

Se o usuário seleciona a opção "Abrir..." do menu Arquivo, tem-se o estado "selecionar arquivo". Isto significa que o sistema solicitará ao usuário uma seleção de arquivo através de um quadro de diálogo apropriado. Feita a seleção, tem-se o estado "invocar o seu Controlador" que significa que o Controlador, por exemplo, deve solicitar ao sistema para que este invoque um editor abrindo o arquivo selecionado a fim de visualizar, editar e outros recursos oferecidos pelo próprio editor.

Figura A.2 Modelo dinâmico da classe JanelaPrincipal



Após a invocação do seu Controlador, a Visão passa para o estado "aguardar comando e menu", aguardando um novo comando de menu.

Se o item "O que é ?" ou "Usando" do menu Ajuda for selecionado, invoca-se um sistema de ajuda abrindo um determinado arquivo (definido pelo projetista) informando ao usuário dados sobre o sistema ou como operar o sistema.

Quando o item "Sobre" do menu Ajuda é selecionado, exibe-se um quadro de diálogo informando ao usuário dados sobre a aplicação contendo: nome, versão, ano e nomes dos desenvolvedores.

Finalmente, se o usuário seleciona a opção "Sair" do menu Arquivo, tem-se um quadro de diálogo solicitando a confirmação da saída da aplicação no estado "confirmar saída da aplicação".

Vale lembrar que as demais opções de menu (que não forão mencionadas nesta seção) não estão especificadas pois estas estão desabilitadas impossibilitando que o usuário as selecione.

A.1.4 JanelaEditor

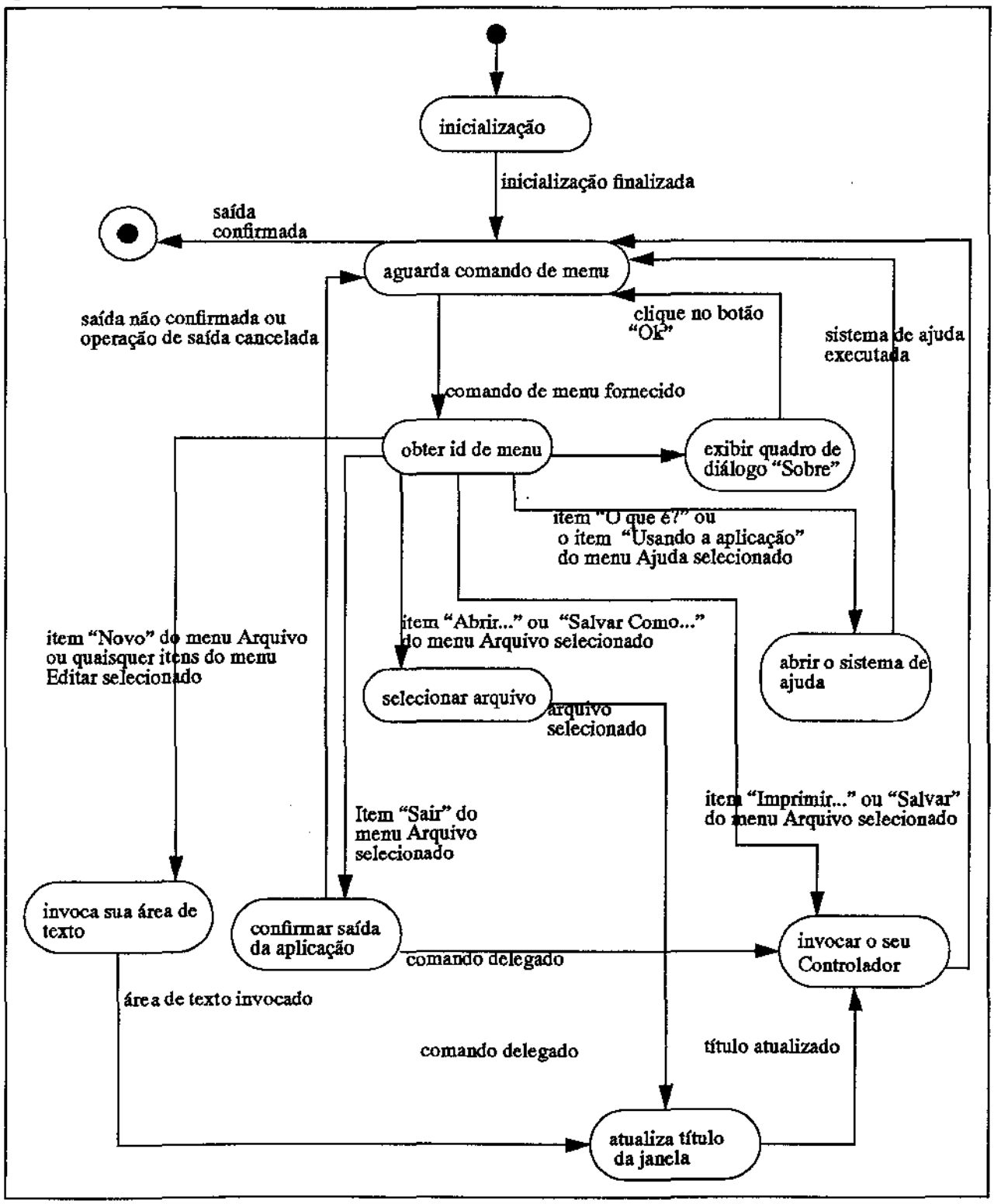
Esta classe representa a janela de editor descrita na seção 5.3.2.3. Esta classe é subclasse da Janela, portanto, além dos atributos e métodos herdados, ela possui somente um atributo específico: *área_texto* que é um ponteiro para a sua área de texto (da classe ÁreaTexto) onde o usuário edita um texto.

O comportamento desta janela foi implementado. Portanto, tem-se o seu modelo dinâmico na figura A.3.

Este modelo é ligeiramente mais complexo em relação ao da janela principal pois a janela trata de um conjunto maior de eventos (comandos de menus) em relação a janela principal. Portanto, tem-se maior quantidade de estados a partir do estado "obter o id de menu".

Se o usuário seleciona a opção "Novo" do menu Arquivo, a classe JanelaEditor obtém o id de menu no estado "obter o id de menu". Com esta identificação da opção selecionada, o estado atual passa para "invoca sua área de texto", a janela invocará algum serviço oferecido pelo seu objeto componente do tipo ÁreaTexto.

Figura A.3 Modelo dinâmico da classe JanelaEditor



Neste caso, o conteúdo desta área deve ser envaziado (limpo). Quando esta operação é realizada, tem-se o novo estado que é "atualiza o título da janela".

Neste estado, o título da janela deve ser atualizado para um título sem o nome do arquivo. Isto significa que não há arquivo aberto. E, finalmente, após a atualização da janela, tem-se o estado "invocar o seu Controlador" passando o seu controle para o Controlador.

Quando o usuário seleciona o item "Abrir..." ou o item "Salvar como..." do menu Arquivo, um quadro de diálogo de seleção de arquivo é exibido no estado "selecionar arquivo".

Selecionado o arquivo, o título da janela deve ser atualizado no estado "atualizar o título da janela" a fim de informar ao usuário que o contexto da janela foi "chaveado" para o arquivo selecionado.

Quando o item "Salvar" ou o item "Imprimir..." no Arquivo é selecionado, o controle é passado para o seu controlador no estado "invocar o seu controlador" para que ele, o Controlador, possa invocar serviços do Modelo.

Note que os comportamentos do menu Ajuda e do item "Sair" do menu Arquivo desta janela são idênticos os da janela principal. Portanto, o seu comportamento foi descrito na seção A.1.3.

A.1.5 JanelaFerramenta

Esta classe representa a janela de ferramenta (descrita na seção 5.3.2.4). Foi visto que esta janela possui duas áreas de texto (como ilustra a figura 5.8), portanto, esta classe possui dois atributos adicionais:

1. *área_texto* - ponteiro para área de texto 1 (classe ÁreaTexto).
2. *área_texto2* - ponteiro para área de texto 2 (classe ÁreaTexto).

O comportamento dinâmico da janela de ferramenta é idêntico ao da janela de editor pois ambas as janelas se têm o mesmo conjunto de itens de menu. A única diferença entre estes dois tipos de janelas é simplesmente o contexto, um para editores e outro para ferramentas. Portanto, a descrição do comportamento da janela está na seção A.1.4 visto que o comportamento da janela de editor é também válido para a de ferramenta.

A.1.6 ÁreaTexto

Esta classe representa uma área de texto onde o usuário edita um texto e visualiza o conteúdo de arquivos (de texto). Os objetos desta classe são instanciados automaticamente (o projetista não instancia esta classe diretamente) durante a criação de um objeto das classes JanelaEditor e JanelaFerramenta.

Esta classe é uma subclasse da classe *wxTextWindow* pertencente à biblioteca do toolkit WXWindows. Foi preciso inserir somente um atributo adicional: *conteudo* do tipo booleano, se o valor é FALSE, significa que esta área de texto está vazia; caso contrário, a área de texto contém dados.

A.1.7 ÁreaOpções

Esta classe representa uma área onde o usuário seleciona alguma opção, o seu objeto é criado de forma automática durante a criação de um objeto da classe JanelaPrincipal. Esta área existem exclusivamente nas janelas principais.

Esta classe é uma subclasse da classe *wxListBox* da biblioteca WXWindows contendo somente dois atributos específicos:

1. *obj_paineljprincipal* - ponteiro para um objeto da classe PainelJPrincipal
2. *item_selecionado* - uma cadeia de caracteres que representa a opção selecionada pelo usuário.

A.1.8 PainelJPrincipal

Esta classe representa uma área de trabalho da janela principal. Esta foi inserida devido à restrição do próprio *toolkit*, isto é, sem esta classe, não seria possível inserir um objeto do tipo *ÁreaOpções* na janela principal.

Esta classe é derivada da classe *wxPanel* da biblioteca do WXWindows cujo atributo específico é *obj_janelaprincipal* (um ponteiro para a sua janela principal). Esta classe não está disponível para projetistas (de ferramenta e nem de interface do ambiente) pois o objeto desta classe é criado e utilizado de forma transparente a fim de facilitar a tarefa dos projetistas.

Esta classe serve para manter consistência de apresentação da área de opções em uma janela principal. Por exemplo, quando a janela principal é redimensionada, um método desta classe é invocada nesta circunstância a fim de redimensionar a sua área de opções de forma adequada.

A.1.9 QuadroDialogo

Esta classe representa os quadros de diálogo. Os quadros de diálogo são quadros que servem para solicitar e obter dados fornecidos pelo usuário.

No caso do protocolo de apresentação do ATIFS, esta classe é subclasse da classe do WXWindows chamada *wxDialogBox* contendo somente um atributo adicional: *obj_janela* (ponteiro para a sua janela da classe Janela), cujo objetivo é associar ao seu objeto de janela.

A.1.10 Painel

Esta classe representa um conjunto de objetos de interação que pode ser composto por objetos do tipo Lista, Botão, CaixaEscolha e CaixaTexto, que serão vistos posteriormente. No caso do ATIFS, todos os painéis já possuem de forma pré-definida dois botões: "Ok" e "Cancelar". O projetista, então, somente define quais objetos de interação devem pertencer ao seu painel exceto estes dois botões.

Esta classe é subclasse da classe *wxForm* do WXWindows que possui, além dos atributos e métodos herdados de sua superclasse, somente um atributo adicional: *obj_dialog* que é um ponteiro para o seu objeto da classe *QuadroDialogo*.

A.1.11 Lista

Esta classe apresenta uma lista de opções (a sua aparência é idêntica a da área de opções) disponíveis para o usuário. Para selecionar uma opção, o usuário deve clicar na opção desejada.

A classe Lista é subclasse da classe *wxListBox* da biblioteca do *WXWindows* contendo somente um atributo específico: *item_selecionado* cujo dado armazenado é uma cadeia de caracteres representando a opção que o usuário selecionou.

A.1.12 CaixaEscolha

A classe CaixaEscolha representa a *caixa de escolha* (exemplo na figura 5.2) contendo uma lista de opções somente quando o seu botão é clicado. Apresentada uma lista de opções, o usuário, então, deve clicar sobre uma opção desejada. Quando uma determinada opção é selecionada, então a lista de opções é eliminada; e somente a opção selecionada será exibida.

O seu papel é semelhante ao da Lista mas existem certas diferenças. A vantagem da lista é que a sua lista de opções pode ser pré-definida e também se pode alterar dinamicamente. Por outro lado, no caso da caixa de escolha, a sua lista de opções deve ser pré-definida.

A vantagem da caixa de escolha sobre a lista é que esta ocupa menos espaço no quadro de diálogo, visto que ele exibe a sua lista somente quando o seu botão é clicado.

A CaixaEscolha é uma subclasse da classe *wxChoice* pertencente à biblioteca do WXWindows. Foi preciso inserir somente um atributo (*item_selecionado*) que mantém a opção selecionada pelo usuário em uma cadeia de caracteres.

A.1.13 Botão

Esta classe representa um botão (exemplo na figura 5.2). Se clicado, o método do seu Controlador *botãoCallBack* é invocado na qual pode ser definido pelo projetista.

Esta classe é definida como subclasse da *wxButton* do WXWindows contendo somente um novo atributo (*dialogo*) que é um ponteiro que associa a um objeto do tipo *QuadroDialogo* (criado pelo projetista)

A.1.14 CaixaTexto

Esta classe é representada como um retângulo onde o usuário preenche com uma cadeia de caracteres. Tem-se um exemplo deste tipo de objeto na figura 5.2.

Esta classe é subclasse da *wxText* da biblioteca do WXWindows contendo um único atributo específico que é *conteúdo* que mantém uma cadeia de caracteres fornecido pelo usuário.

A.2 CONTROLADOR

Tem-se basicamente quatro tipos de Controladores virtuais (como mostra a figura A.4): um para editores (*VControladorEditor*), outro para ferramentas (*VControladorFerramenta*) e, finalmente, um para desenvolver a interface principal do ATIFS (*VControladorAmbiente*). Finalmente, estes controladores virtuais possuem a mesma superclasse que é definido por um Controlador virtual de forma mais abstrata: o *VControlador*.

Quando o projetista define o seu Controlador, dependendo do que se trata do sistema que está sendo implementado, derivado a partir de um dos controladores virtuais mencionados acima, exceto *VControlador* (nunca deve ser derivado e nem instanciado).

A seguir, tem-se quatro seções, cada seção menciona estes controladores virtuais.

A.2.1 VControlador

Esta classe é a mais abstrata quando se trata de controladores contendo somente dois atributos e vários métodos virtuais. Os atributos são: *objeto_janela* (um ponteiro para um objeto da classe Janela e *objeto_modelo* (um ponteiro para um objeto da classe Modelo).

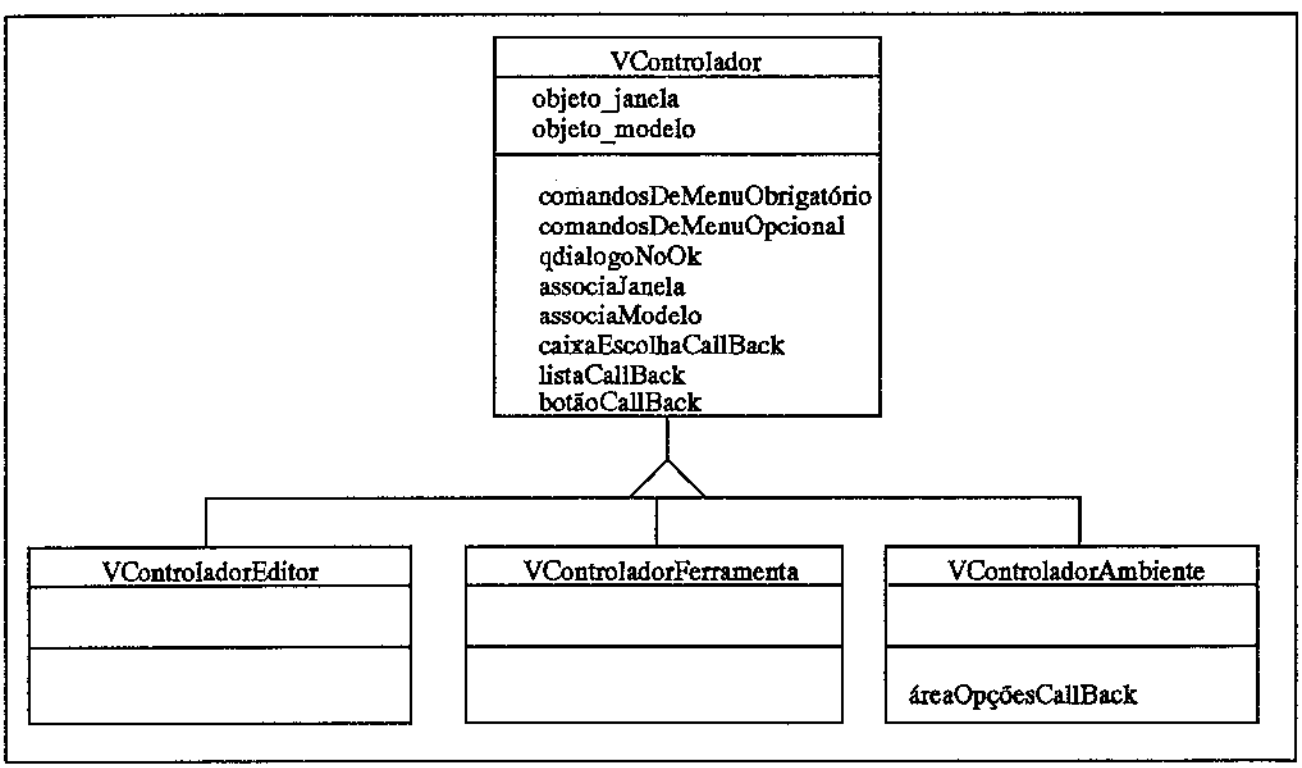
Os seus serviços (ou métodos) são todos virtuais para que possa definir controladores virtuais específicos que serão vistos a seguir.

A.2.2 VControladorAmbiente

Esta classe é uma subclasse do VControlador. O VControladorAmbiente possui um contexto mais especializado em relação ao VControlador pois se trata de interface principal do ATIFS.

O VControladorAmbiente possui, além dos mesmos conjuntos de atributos e métodos virtuais da sua superclasse através de sobreposição de métodos como mostra a figura A.4. Note que a classe VControladorAmbiente possui um atributo *(obj_janela)* que aponta para um objeto da classe *JanelaPrincipal.* Além disso, esta classe também possui um método virtual adicional que é *áreaOpçõesCallBack* cuja função é definir um *call-back* de um área de opções, visto que somente a janela principal possui este tipo de área de trabalho.

Figura A.4 Modelo de objetos dos controladores virtuais



A.2.3 VControladorFerramenta

Aplica-se o mesmo raciocínio do VControladorAmbiente ao caso do VControladorFerramenta, o VControladorFerramenta é o VControlador voltado para o contexto de ferramenta.

Note que a classe VControladorFerramenta possui um atributo *(obj_janela)* que aponta para um objeto da classe *JanelaFerrramenta.* Esta classe possui os mesmos conjuntos de métodos virtuais do VControlador permitindo que o projetista defina o Controlador específico de uma ferramenta.

A.2.4 *VControladorEditor*

Este é o VControlador voltado para editores contendo o atributo *obj_janela* que é um ponteiro para um objeto do tipo *JanelaEditor.*

Note que esta classe contém os mesmos conjuntos de métodos virtuais do VControlador, prontos para serem definidos pelo projetista.

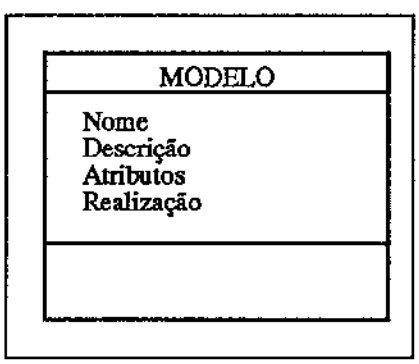
A.3 COMPONENTE MODELO

No componente Modelo (ver figura A.5), tem-se a classe Modelo. As classes do componente Modelo devem ser subclasses da classe Modelo para que obtenha as suas propriedades básicas herdadas.

Na classe Modelo, a princípio, tem-se os seguintes atributos:

- *Nome* - contém o nome do objeto de configuração;
- *Descrição* - contém a descrição do objeto;
- *Atributos* - aqui se tem os atributos que serão levantados para um objeto de configuração específico.
- *Realização* - um ponteiro que aponta para onde o dado está armazenado;

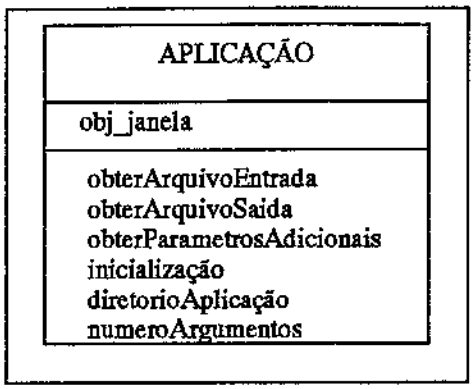
Figura A.5 O Modelo de objetos do componente Modelo



A.4 CLASSE APLICACÃO

Todos os tipos de sistemas (ferramentas, ambiente, editores) devem possuir de forma obrigatória (devido às regras de implementação) a classe *Aplicação.* Esta classe (ilustrada na figura A.6) representa um programa principal do sistema como a função *main()* na linguagem de programação C. Portanto, é importante descrever esta classe que será feita nesta seção.

Figura A.6 Classe Aplicação



Esta classe é útil principalmente para definir a inicialização da aplicação, obter argumentos na chamada de um sistema e obter algumas informações sobre a aplicação. Portanto, o projetista deve definir, pelo menos, o método *inicialização* desta classe (ver um exemplo na seção 7.3.1).

No contexto estrutural, esta classe foi implementada como subclasse da classe *wxApp* da biblioteca do WXWindows contendo somente um atributo adicional que é *obj_janela* (ponteiro para um objeto do tipo Janela). Os seus métodos estão descritos no apêndice B.

Apêndice B

SERVIÇOS IMPLEMENTADOS DO FRAMEWORK

Este apêndice apresenta todos os serviços oferecidos pelo protocolo de apresentação do ATIFS e do componente Controlador. Cada serviço tem uma função específica juntamente com os seus parâmetros e alguns retornam algum valor que pode ser relevante ao objeto que o invocou.

Inicialmente, tem-se os serviços que não pertencem a nenhuma classe pois estes não armazenam nenhuma informação simplesmente invocam funções do *toolkit WXWindows*. Posteriormente, tem-se uma lista de serviços que foram agrupados por classes pertencentes aos modelos de objetos do protocolo de apresentação e do componente Controlador (ver detalhes destes modelos de objetos no apêndice A).

B.1 SERVIÇOS GENÉRICOS

No caso do ATIFS, os serviços genéricos, aqueles que não pertencem a nenhuma classe, são compostos por três serviços, que são:

1. Para exibir um alerta.

int resposta = exibirAlerta(char *mensagem, int nr_botoes)

onde: resposta - resposta do usuário que podem ser OK ou Cancelar
nr_botões - número de botões (1 - Ok e 2 - Ok e Cancelar)
mensagem - mensagem de alerta

2. Para exibir uma confirmação

int resposta = exibirConfirmação(char *mensagem, int tipo)

onde: resposta - resposta do usuário que podem ser: Sim, Nao e Cancelar.

tipo - se 1 o quadro de diálogo terá 2 botões: "Sim" e "Não" e se o tipo for 2 quadro terá três botões: "Sim", "Não" e "Cancelar".

mensagem - mensagem de confirmação.

3. Para solicitar ao usuário uma seleção de arquivo

char *arquivo = selecionarArquivo(int estado, char *diretorio, char *extensão)

onde: arquivo - arquivo selecionado pelo usuário (tem-se um path completo com nome de arquivo selecionado).

estado - se 1 o estado é abrir um arquivo e se 2 o estado é salvar um arquivo.

diretorio - o diretorio default

extensão - a extensão default

B.2 SERVIÇOS AGRUPADOS POR CLASSES

Nesta seção, tem-se uma lista completa de serviços agrupados por classes as quais eles pertencem. A seguir, tem-se várias seções onde cada seção representa uma classe do modelo de objetos do protocolo de apresentação do ATIFS.

B.2.1 Aplicação

1. Inicialização de uma aplicação

```
Janela *Aplicação::inicialização()
{
// Aqui o projetista deve instanciar o(s) objeto(s) da classe Menu e instanciar um Controlador.
// insere instanciação de uma classe do componente Modelo.
// insere aqui as associações entre Controlador e Modelo
}
```

2. obter o diretório de uma aplicação

char *diretorio = Aplicação.diretorioAtual ()

onde: diretorio - diretório da aplicação obtido

3. obter um número de parâmetros

int numero_argumentos = Aplicação.numeroArgumentos()

onde:numero_argumentos - número de argumentos obtido

4. obter uma arquivo de entrada

char *arquivo_entrada = Aplicação.obterArquivoEntrada()

onde: arquivo_entrada - arquivo de entrada obtido

5. obter um arquivo de saída

char *arquivo_saida = Aplicação.obterArquivoSaida()

onde: arquivo_saida - arquivo de saída obtido

6. obter parâmetros adicionais

char *parametros_adicionais = Aplicação.obterParametrosAdicionais()

onde: parametros_adicionais - parâmetros adicionais obtidos

B.2.2 JanelaPrincipal, JanelaEditor e JanelaFerramenta

7. Exibir e esconder uma janela

objeto->exibir(bool estado)

Se estado for TRUE a janela é exibida e se FALSE, a janela é escondida.

8. Definir um editor que será invocado quando o usuário seleciona o item "Novo" ou "Abrir..." do menu obrigatório Arquivo (serviço exclusivo da JanelaPrincipal)

objeto->definirEditor(char *arquivo_editor)

onde: arquivo_editor - nome do arquivo de editor

9. Obter um arquivo atual (disponível somente na *JanelaEditor e JanelaFerramenta*)

arquivo_atual = objeto->obterArquivoAtual()

onde: arquivo_atual - é uma cadeia de caracteres representando o arquivo atual retornado.

10. Exibir uma cadeia de caracteres na barra de status da janela

objeto->exibirStatus(string)

onde: string - cadeia de caracteres que deve ser exibida.

B.2.3 ÁreaTexto

A sua instanciação é feita automaticamente. Quando se trata de janela de editor, tem-se o ponteiro de objeto para a área de trabalho chamado: *área_texto*. Quando se trata de ferramenta, tem-se dois duas áreas de trabalho, os ponteiros são: *área_texto* e *área_texto2*.

11. Limpar o seu conteúdo

objeto->limpar()

12. Exibir uma string em uma área de texto

objeto->exiba(char *string)

onde: string - uma string que deve ser exibida na área de texto

13. Carregar arquivo exibindo o seu conteúdo na área de texto

objeto->carregar(char *arquivo)

onde: arquivo - nome do arquivo a ser carregado (pode-se incluir o seu caminho)

14. Salvar o conteúdo da área de texto em um arquivo

objeto->salvar(char *arquivo)

onde: arquivo - nome do arquivo(inclui o seu caminho)

15. Obter número de linhas da área de texto

numerodelinhas = objeto->obterNumeroDeLinhas()

onde: numerodelinhas - número inteiro que representa o número de linhas que a área de texto contém.

16. Obter uma linha de texto na área de texto

char *buffer = objeto->obterLinhaTexto(int linha)

onde: linha - linha do texto (começando do zero)
buffer - linha de texto desejada

B.2.4 ÁreaOpções

A sua instanciação é automática. Quando se trata de janela principal, tem-se o ponteiro de objeto para a área de área de opções chamado: *área_opções*.

17. Setar opção selecionada inicial (ou default)

objeto->setarOpçãoSelecionada(int n)

onde: n - posição da lista (iniciando no 0)

18. Inserir opções na lista

objeto->inserirOpção(char *rotulo)

onde: rotulo - rótulo da nova opção

19. Excluir a n-ésima opção na lista

objeto->excluirOpção(int n)

onde: n - posição na lista (iniciando em 0)

20. Obter item selecionado na lista retornando uma string contendo a opção selecionada.

char *opção_selecionada = objeto->obterOpçãoSelecionada()

B.2.5 Menu

21. Instanciação

objeto = new Menu

22. Inserir um item a um menu

objeto->inserirItens(int ID, char *rotulo, char *comentario)

onde: ID - ID de menu
rotulo - rótulo do item de menu
comentario - um pequeno comentário que aparece na barra de status da janela

PS: Os ID's de menu devem ser definidos como um número inteiro maior ou igual a 500 pois os ID's abaixo deste número, são reservados pelo Padrão-ATIFS.

23. Habilitar e desabilitar um item de menu

objeto->habilita(int ID, bool estado)

onde: ID - ID de menu
estado - se TRUE habilita o item de menu e se FALSE, desabilita o item

24. Inserir um separador de itens de menu a fim de agrupar itens de menu melhorando a sua estrutura.

objeto->inserirSeparador();

B.2.6 QuadroDialogo

25. Instanciação

objeto = new QuadroDialogo (Janela *obj, char *titulo, int x, int y, int largura, int altura)

onde: obj - objeto janela proprietária do quadro de diálogo
titulo - título do quadro de diálogo
x e y - coordenadas da borda esquerda e superior do quadro
largura - largura do quadro

altura - altura do quadro

26. Exibir quadro de diálogo

objeto->exibir(bool estado)

onde: estado - se TRUE exibe o quadro de diálogo

B.2.7 Painel

Vale lembrar que todo o painel de quadro de diálogo do ATIFS já possui dois botões pré-definidos, que são: Ok e Cancelar.

27. Instanciação

objeto = new Painel(QuadroDialogo *obj)

onde: obj - objeto de quadro de dialogo

28. Ajustes finais do quadro de diálogo (sempre após a instanciação dos objetos de interação, componentes do quadro de diálogo) associando o painel a um objeto de quadro de diálogo.

objeto->fimEdição(QuadroDialogo *obj);

onde: obj - objeto de quadro de diálogo que deve ser associado.

B.2.8 Botão

29. Instanciação

objeto = new Botão (QuadroDialogo *obj, char *rotulo, int x, int y)

onde: obj - objeto quadro de diálogo proprietário
rotulo - rótulo do botão.
x e y - coordenadas da borda esquerda e superior do botão

30. Habilita e desabilita um botão

objeto->habilita(int estado)

onde: estado - se TRUE - habilita o objeto; se FALSE - desabilita o objeto

B.2.9 CaixaEscolha

31. Instanciação

objeto = CaixaEscolha (QuadroDialogo *obj, char *rotulo, int x, int y, int largura, int altura, int nr_opções, char *opções[])

onde: obj - objeto quadro de diálogo proprietário
rotulo - rótulo da caixa de escolha
x e y - coordenadas da borda esquerda e superior da caixa de escolha
largura - largura da caixa de escolha
altura - altura da caixa de escolha
nr_opções - número de opções disponíveis
opções - é um vetor de string contendo todas as opções disponíveis

32. Obter item selecionado (como uma cadeia de caracteres) de uma caixa de escolha.

char *opção_selecionada = objeto->obterOpçãoSelecionada()

33. Habilita e desabilita uma caixa de escolha

objeto->habilita(int estado)

onde: estado - se TRUE - habilita o objeto; se FALSE - desabilita o objeto

B.2.10 CaixaTexto

34. Instanciação

objeto = new CaixaTexto (QuadroDialogo *obj, char *rótulo, int x, int y, int largura, int altura)

onde: obj - objeto quadro de diálogo proprietário
rotulo - rótulo da caixa de texto
x e y - coordenadas da borda esquerda e superior da caixa de texto
largura - largura da caixa de texto
altura - altura da caixa de texto

35. Obter o conteúdo de uma caixa de texto

char *conteudo = objeto->obterConteudo()

36. Exibir valor em uma caixa de texto

objeto->exibirConteudo(char *conteudo)

onde: conteudo - cadeia de caracteres que deve ser exibida.

37. Habilita e desabilita uma caixa de texto

objeto->habilita(int estado)

onde: estado - se TRUE - habilita o objeto; se FALSE - desabilita o objeto

B.2.11 Lista

38. Instanciação

objeto = Lista (QuadroDiálogo *obj, char *rotulo, int x, int y, int largura, int altura)

onde: obj - objeto quadro de diálogo proprietário
rotulo - rótulo da lista
x e y - coordenadas da borda esquerda e superior da lista
largura - largura da lista
altura - altura da lista

39. Setar uma opção como selecionada por default

objeto->setarOpçãoSelecionada(int n)

onde: n - posição da lista (iniciando no 0)

40. Inserir uma opção em uma lista

objeto->inserirOpção(char *rotulo)

onde: rotulo - rótulo da nova opção.

41. Exclui a n-ésima opção na lista

objeto->excluirOpção(int n)

onde: n - posição na lista (iniciando em 0)

42. Obter item selecionado (como uma string) em uma lista

char *opção_selecionada = objeto->obterOpçãoSelecionada()

B.2.12 Controlador

43. Instanciação (supõe-se que a classe denominada de *Controlador* seja a classe controladora definida pelo projetista)

objeto_controlador = new Controlador(char *titulo_janela, char *extensao_default, char *arquivo_ajuda1, char *arquivo_ajuda2, char *sobre, int num_menu, char *menu[], char *rotulo[])

onde: titulo_janela - titulo da janela

extensao_default - extensao de arquivo default da aplicação (válido somente no caso de ferramenta e editor. No caso da janela principal, este parâmetro não deve ser especificado).

arquivo_ajuda1 - nome do arquivo de ajuda (arquivo de texto em ANSI) contendo informação referentes à aplicação.

arquivo_ajuda2 - nome do arquivo de ajuda em ANSI que se refere à operação desta aplicação.

sobre - cadeia de caracteres que se refere a aplicação, tais como: nome, versão, nome dos desenvolvedores, data de desenvolvimento, etc.). É permitido o uso de caracteres de formatação existentes na linguagem C++.

num_menu - número de menus opcionais.

menu[] - vetor de objetos de menu.

rotulos[] - vetor de rotulos dos respectivos menus no vetor menu[].

No caso de uma ferramenta funcional, tem-se a seguinte definição da função construtora:

*Controlador::Controlador(char *titulo_janela, char *extensao_default, char *arquivo_ajuda1,char *arquivo_ajuda2, char *sobre, int num_menu, Menu *menu[], char *rotulo[]): VControladorFerramenta(titulo_janela, extensao_default, arquivo_ajuda1, arquivo_ajuda2, sobre, num_menu, menu, rotulo)*

{

}

No caso de um editor, tem-se a seguinte definição da função construtora:

*Controlador::Controlador(char *titulo_janela, char *extensao_default, char *arquivo_ajuda1,char *arquivo_ajuda2, char *sobre, int num_menu, Menu*

**menu[], char *rotulo[]): VControladorEditor(titulo_janela, extensao_default, arquivo_ajuda1, arquivo_ajuda2, sobre, num_menu, menu, rotulo)*

{

}

No caso do ambiente, tem-se a seguinte definição da função construtora:

*Controlador::Controlador(char *titulo_janela, char *arquivo_ajuda1,char *arquivo_ajuda2, char *sobre, int num_menu, Menu *menu[], char *rotulo[]): VControladorAmbiente(titulo_janela, arquivo_ajuda1, arquivo_ajuda2, sobre, num_menu, menu, rotulo)*

{

}

44. Associar a uma janela

objeto->associaJanela(Janela *obj)

onde: obj - objeto de janela

45. Associar a um objeto do Modelo

objeto->associaModelo (Modelo *obj);

onde: obj - objeto do Modelo

46. Respondendo aos comandos de menu(s) obrigatório(s)

Deve definir o método (é executada quando um item de menu obrigatório é selecionado):

Controlador :: comandosDeMenuObrigatorio (int ID)

{

switch (ID):

case XXX

{

...

}

case YYY

{

...

}

....

}

onde: ID - id de item selecionado de um menu obrigatório

47. Respondendo aos comandos de menu(s) opcional(is)

Deve definir o método (é executada quando um item de menu opcional é selecionado):

Controlador :: comandosDeMenuOpcional (int ID)
{
switch (ID):
case XXX
{
...
}
case YYY
{
...
}
....
}

onde: ID - id de item selecionado de um menu opcional

48. Respondendo a um clique no botão obrigatório Ok de um quadro de diálogo

Deve definir o método (é executado quando ocorre um clique no botão Ok de um quadro de diálogo):

*Controlador :: qdialogoNoOk (QuadroDiálogo *obj)*
{
switch (obj):
case QD1
{
// Invoca-se métodos de objetos de interação a fim de obter dados importantes.
// Invoca serviços do Modelo a fim de configurá-lo, por exemplo.
...
}
case QD2
{
...

```
}
...
}
```

onde: obj - objeto de quadro de diálogo

49. Faz o papel de call-back dos objetos do tipo CaixaEscolha

Para responder aos eventos da caixa de escolha, o projetista pode definir o método da classe Controlador:

```
Controlador::caixaEscolhaCallBack(caixaEscolha *obj)
{
// aqui insere o código de resposta a este call-back.
}
```

O parâmetro *obj* é um ponteiro para o objeto que recebeu o evento, assim, é possível determinar a resposta do evento a cada objeto.

50. Faz o papel de call-back dos objetos do tipo Botão

Para responder aos eventos da caixa de escolha, o projetista pode definir o método da classe Controlador:

```
Controlador::botãoCallBack(Botão *obj)
{
// aqui insere o código de resposta a este call-back.
}
```

O parâmetro *obj* é um ponteiro para o objeto que recebeu o evento, assim, é possível determinar a resposta do evento a cada objeto.

51. Faz o papel de call-back dos objetos do tipo Lista

Para responder aos eventos da lista, o projetista pode definir o método da classe Controlador:

```
Controlador::listaCallBack(Lista *obj)
{
// aqui insere o código de resposta a este call-back.
}
```

O parâmetro *obj* é um ponteiro para o objeto que recebeu o evento, assim, é possível determinar a resposta do evento a cada objeto.

52. Faz o papel de call-back dos objetos do tipo ÁreaOpções

Para responder aos eventos da área de opções, o projetista pode definir o método da classe Controlador:

```
Controlador::areaOpcoesCallBack(void)
{
// aqui insere o código de resposta a este call-back.
}
```

OBS: Este serviço está disponível somente quando este Controlador for subclasse do VControladorAmbiente, quando se trata da janela principal.