UNIVERSIDADE SALVADOR – UNIFACS
PROGRAMA DE PÓS-GRADUAÇÃO EM REDES DE COMPUTADORES
MESTRADO PROFISSIONAL EM REDES DE COMPUTADORES

DEMIAN LESSA

INTERFACES GRÁFICAS COM O USUÁRIO: UMA ABORDAGEM BASEADA EM PADRÕES

Salvador
2005

DEMIAN LESSA

INTERFACES GRÁFICAS COM O USUÁRIO: UMA ABORDAGEM BASEADA EM PADRÕES

Dissertação apresentada ao Mestrado Profissional em Redes de Computadores da Universidade Salvador – UNIFACS, como requisito parcial para obtenção do grau de Mestre.

Orientador: Prof. Dr. Manoel Gomes de Mendonça.

Salvador

2005

Ficha Catalográfica elaborada pelo Sistema de Bibliotecas da Universidade Salvador - UNIFACS

Lessa, Demian

Interfaces gráficas com o usuário: uma abordagem baseada em padrões / Demian Lessa. – Salvador, 2005.

202 f.: il.

Dissertação apresentada ao Mestrado Profissional em Redes de Computadores da Universidade Salvador – UNIFACS, como requisito parcial para a obtenção do grau de Mestre.

Orientador: Prof. Dr. Manoel Gomes de Mendonça.

1. Interfaces gráficas para usuário - Sistema de computador. I. Mendonça, Manoel Gomes de, orient. II. Título.

CDD: 004

TERMO DE APROVAÇÃO

DEMIAN LESSA

INTERFACES GRÁFICAS COM O USUÁRIO: UMA ABORDAGEM BASEADA EM PADRÕES

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre em em Redes de Computadores da Universidade Salvador – UNIFACS, pela seguinte banca examinadora:

Manoel Gomes de Mendonça – Orientador _____
Doutor em Ciência da Computação pela Universidade de Maryland em College Park, Estados Unidos
Universidade Salvador - UNIFACS

Celso Alberto Saibel Santos _____
Doutor em Informatique Fondamentalle et Parallelisme pelo Université Paul Sabatier de Toulouse III, França
Universidade Federal da Bahia – UFBA

Flávio Morais de Assis Silva _____
Doutor em Informática pelo Technische Universität Berlin, Alemanha
Universidade Federal da Bahia – UFBA

Salvador de de 2005

A meus pais, Luiz e Ines, pelo constante incentivo intelectual e, muito especialmente, por todo amor e carinho repetidamente demonstrados.

A minha irmã, Daniela, sobrinhas, Milena e Gabriela, e cunhado, Erik, que, mesmo distantes, são presença contínua em meus pensamentos.

AGRADECIMENTOS

A meus pais, pelos apoios afetivo e intelectual. Também, pelas orientações práticas, revisões de forma e estilo.

Ao Prof. Manoel Gomes de Mendonça Neto, meu orientador, por suas contribuições sempre objetivas e relevantes. Por aceitar meu ritmo peculiar de trabalho e ajudar-me no cumprimento dos prazos. Por fim, pelo apoio, através de suas idéias e experiências, que incentivaram-me a definir a etapa seguinte de minha vida acadêmica.

Aos demais professores dos cursos de pós-graduação da UNIFACS e UFBA, por suas contribuições individuais e coletivas na minha formação acadêmica. Em especial, ao Prof. Joberto Martins, cujas orientações na dissertação de conclusão do curso de Pós-Graduação em Redes de Computadores, na UNIFACS, continuam sendo úteis.

RESUMO

O desenvolvimento de interfaces com o usuário é uma atividade complexa que ocupa parte significativa do esforço de desenvolvimento de um sistema de informação. A busca por novos métodos de apoio ao desenvolvimento dessas interfaces é de extrema importância tanto na redução dos custos de um projeto, quanto na melhoria da qualidade de suas interfaces. Este trabalho parte do princípio que interfaces gráficas podem ser desenvolvidas de forma eficaz através do uso de padrões. A abordagem proposta sugere o uso de uma coleção de padrões de interfaces gráficas em conjunto com uma biblioteca de classes que a implementa. O catálogo que descreve a coleção é utilizado na fase de análise do sistema e a biblioteca de classes em sua implementação. Para demonstrar a viabilidade dessa abordagem, uma coleção de padrões foi elaborada e posteriormente implementada através de uma biblioteca de classes em uma linguagem de programação orientada a objetos. Os resultados desse trabalho sugerem a viabilidade da separação entre a descrição e a implementação de um modelo de interfaces baseado em padrões. Ainda como parte do trabalho, empregou-se a biblioteca de padrões na implementação das interfaces gráficas de um sistema de informações real, de médio porte. Os resultados observados indicam ser adequada a abordagem baseada em padrões para o desenvolvimento das interfaces gráficas de sistemas de informação.

Palavras-chave: Padrão. Interface Gráfica. Manipulação Direta. IHC.

ABSTRACT

User interface development is a complex activity that demands considerable effort during the development of an information system. The search for new methods to support the development of such interfaces is an extremely important task for both the reduction of project costs and the improvement of its interfaces. This work assumes that graphical user interfaces can be developed effectively through the use of patterns. On such a patter-based approach, a graphical interface pattern collection would be used together with a class library that implements it. The collection document would be used during the system analysis phase, while the class library would be used in its implementation. To demonstrate the usability of this approach, a pattern collection was composed and later implemented as a class library on an object-oriented programming language. The results of this work suggest that the separation between the description and the implementation of a pattern-basedinterface model is clearly possible. Further, this pattern class library was used in the development of the graphical interfaces of a real, mid-sized information system. The results indicate that the pattern-based approach is adequate for the development of the graphical interfaces of information systems

Keywords: Design Pattern. User Interface. Direct Manipulation. IHC.

LISTA DE FIGURAS

Figura 1 – Memex 25
Figura 2 – Perfuradora de cartões IBM 026 26
Figura 3 – IBM 407 Accounting Machine.....27
Figura 4 – O teletipo IBM ASR-33 32
Figura 5 – Terminal VT100.....33
Figura 6 – O Sketchap, com parte de uma ponte em destaque na tela. 34
Figura 7 – O Sistema NLS em execução 35
Figura 8 – Dispositivos de Entrada do NLS: mouse, teclado e keyset 36
Figura 9 – O Xerox Alto: monitor, teclado, *mouse* e *keyset*.....37
Figura 10 – Neptune, o gerenciador de arquivo do Xerox Alto.....38
Figura 11 – Bravo, o editor de texto Wysiwyg do Xerox Alto.....38
Figura 12 – Draw, editor gráfico do Xerox Alto.....39
Figura 13 – O ambiente Smalltalk do Xerox Alto.....40
Figura 14 – Estação de Trabalho Sun-1.....41
Figura 15 – Xerox Star 8010, 1981 42
Figura 16 – Apple MacIntosh, 1984.....42
Figura 17 – O ambiente X Windows com o Window Maker 43
Figura 18 – Microsoft Windows 1.0 45
Figura 19 – Microsoft Windows 2.0 46
Figura 20 – Microsoft Windows 3.x 46
Figura 21 – Microsoft Windows 95 47
Figura 22 – Microsoft Bob 47
Figura 23 – Pine, rodando em um terminal emulado.....60
Figura 24 – Gerenciamento de pacote no SUSE Linux com o Yast 61
Figura 25 – Componentes da Arquitetura de Software 71
Figura 26 – MFC Toolkit.....75
Figura 27 – Java AWT Toolkit 75
Figura 28 – wxWidgets Toolkit 75
Figura 29 – Qt Toolkit.....76
Figura 30 – Java SWING com o tema Motif 76
Figura 31 – Java SWING com o tema Metal 76
Figura 32 – Ambiente NextStep 80

Figura 33 – Interface gerada pelo Naked Objects 85
Figura 34 – Sistema Hypercard em execução 88
Figura 35 – Exemplo: Identificador, Obrigatoriedade e Somente-Leitura 105
Figura 36 – Exemplo: Foco e Lookup Local 108
Figura 37 – Exemplo: Alternância de Cores 109
Figura 38 – Exemplo: Barra de Controles 110
Figura 39 – Exemplo: Menu Contextual 111
Figura 40 – Exemplo: Lookup por Delegação-Acionamento da Pesquisa 114
Figura 41 – Exemplo: Lookup por Delegação-Pesquisa e Seleção 114
Figura 42 – Exemplo: Lookup por Delegação, Pesquisa Concluída 114
Figura 43 – Exemplo: Grade 116
Figura 44 – Exemplo: Grade Contextualizada 117
Figura 45 – Exemplo: Grade de Pesquisa Simples (1 de 2) 119
Figura 46 – Exemplo: Grade de Pesquisa Simples (2 de 2) 119
Figura 47– Exemplo: Grade Mestre-Detalhe 121
Figura 48 – Exemplo: Grades Alternativas 123
Figura 49 – Exemplo: Grade de Pesquisa Avançada 125
Figura 50 – Exemplo: Formulário 126
Figura 51 – Exemplo: Formulário Segmentado (1 de 3) 127
Figura 52 – Exemplo: Formulário Segmentado (2 de 3) 128
Figura 53 – Exemplo: Formulário Segmentado (3 de 3) 128
Figura 54 – Exemplo: Formulário Mestre-Detalhe 129
Figura 55 – Exemplo: Assistente (parte 1) 132
Figura 56 – Exemplo: Assistente (parte 2) 132
Figura 57 – Exemplo: Assistente (parte 3) 133

LISTA DE TABELAS

Tabela 1– Quadro Comparativo dos Paradigmas de Interação 67
Tabela 2 – Ambientes de Desenvolvimento (parte 1).....94
Tabela 3 – Ambientes de Desenvolvimento (parte 2).....97
Tabela 4 – Agrupamento dos Padrões de Interfaces Gráficas.....104
Tabela 5 – Módulos e Classes da Biblioteca.....137
Tabela 6 – Padrões Implementados na Biblioteca 140

LISTA DE DIAGRAMAS UML

Diagrama 1 – Principais Classes de Arquitetura de Padrões 141
Diagrama 2 – Classes Concretas para Programação de Interfaces 142
Diagrama 3 – Classes de Apoio à Implementação de Assistentes 143
Diagrama 4 – Classes de Exibição de Informações Contextuais de Editores 143
Diagrama 5 – A Classe TfmAutoGUI 143
Diagrama 6 – A Classe TclDAConfig 144

SUMÁRIO

1 INTRODUÇÃO 17
1.1 JUSTIFICATIVA 18
1.2 OBJETIVO GERAL 19
1.3 OBJETIVOS ESPECÍFICOS 20
1.4 ESCOPO 20
1.5 RESULTADOS 21
1.6 ORGANIZAÇÃO DO TRABALHO 21
2 EVOLUÇÃO DA INTERAÇÃO HUMANO-COMPUTADOR 23
2.1 DESTAQUES HISTÓRICOS 23
2.1.1 Memex 24
2.1.2 Perfuradoras de cartão 25
2.1.3 Time-Sharing 28
2.1.4 Sistemas Operacionais 29
2.1.5 Tecnologia de Vídeo 30
2.1.6 Teletipos e Terminais de Vídeo 31
2.1.7 Sketchpad 34
2.1.8 The Mother of All Demos 34
2.1.9 O Xerox Alto 36
2.1.10 As Estações de Trabalho da Sun Microsystems 40
2.1.11 O Apple MacIntosh 41
2.1.12 O Sistema X Windows 42
2.1.13 A Revolução de 32 Bits 44
2.1.14 O Microsoft Windows 45
2.1.15 O Java 47
2.1.16 O Linux e o Movimento Open Source 48
2.2 OS PARADIGMAS DA INTERAÇÃO HUMANO-COMPUTADOR 49
2.2.1 Sistemas Batch 50
2.2.1.1 Período 50
2.2.1.2 Primeira Geração de Computadores 50
2.2.1.3 Segunda Geração de Computadores 51
2.2.1.4 Interação Humano-Computador 53
2.2.2 Sistemas de Linha de Comando e Sistemas de Tela Cheia 54

2.2.2.1 Período 54
2.2.2.2 Terceira Geração de Computadores 54
2.2.2.3 Quarta Geração de Computadores 57
2.2.2.4 Interação Humano-Computador 58
2.2.3 Sistemas de Manipulação Direta 62
2.2.3.1 Período 62
2.2.3.3 Interação Humano-Computador 64
2.2.4 Análise Comparativa dos Paradigmas de Interação 65
3 INTERFACES DE MANIPULAÇÃO DIRETA 69
3.1 MODELAGEM DE INTERFACES HUMANO-COMPUTADOR 69
3.2 ARQUITETURA DE SOFTWARE GRÁFICO BASEADO EM GUIS 70
3.3 SISTEMAS DE JANELA 71
3.4 TOOLKITS 73
3.5 USER INTERFACE MANAGEMENT SYSTEMS (UIMSS) 78
3.6 CONSTRUTORES DE INTERFACE 81
3.8 ABORDAGEM BASEADA EM PADRÕES 86
3.9 OUTRAS TÉCNICAS E FERRAMENTAS 90
3.9.1 Sistemas Baseados em Modelos 90
3.9.2 Geração Automática de Interfaces 90
3.9.3 Hypercard 91
3.9.4 Outras Ferramentas 91
3.10 LEVANTAMENTO DE AMBIENTES DE DESENVOLVIMENTO 92
3.10.1 Objetivo 92
3.10.2 Metodologia 92
3.10.3 Resultados 95
4 METODOLOGIA 98
4.1 ESTRUTURA 98
4.2 ELABORAÇÃO DA COLEÇÃO DE PADRÕES 99
4.2.1 Escopo 99
4.2.2 Definição do Idioma de Padrões 99
4.3 IMPLEMENTAÇÃO DA BIBLIOTECA PARA GERAÇÃO DE INTERFACES 101
4.3.1 Escopo 101
4.3.2 Modelagem e Implementação 101
5 TRABALHO DESENVOLVIDO 103

5.1 COLEÇÃO DE PADRÕES DE INTERFACES GRÁFICAS..... 103
5.1.1 Identificador 104
5.1.2 Obrigatoriedade 105
5.1.3 Somente-Leitura 106
5.1.4 Foco 107
5.1.5 Alternância de Cores 108
5.1.6 Barra de Controles 109
5.1.7 Menu Contextual 110
5.1.8 Lookup Local 112
5.1.9 Lookup por Delegação 113
5.1.10 Grade 115
5.1.11 Grade Contextualizada 116
5.1.12 Grade de Pesquisa Livre 118
5.1.13 Grade Mestre-Detalhe 120
5.1.14 Grades Alternativas 121
5.1.15 Grade de Pesquisa Avançada 123
5.1.16 Formulário 125
5.1.17 Formulário Segmentado 127
5.1.18 Formulário Mestre-Detalhe 128
5.1.19 Assistente 130
5.1.20 Listagem Simples 133
5.2 IMPLEMENTAÇÃO DE REFERÊNCIA DA COLEÇÃO DE PADRÕES 134
5.3 USANDO OS PADRÕES 145
6 DISCUSSÃO DO TRABALHO 147
6.1 A COLEÇÃO DE PADRÕES 147
6. 2 A IMPLEMENTAÇÃO DE REFERÊNCIA 150
6. 3 VALIDAÇÃO DO TRABALHO 151
7 CONCLUSÃO 153
7.2 LIMITAÇÕES DO TRABALHO 154
7.3 TRABALHOS FUTUROS 155
REFERÊNCIAS 157
APÊNDICE A - Coleção de Padrões 172

1 INTRODUÇÃO

A porção da interface com o usuário tem papel fundamental no sucesso de um aplicativo. Usuários esperam que interfaces sejam práticas e fáceis de utilizar (MYERS; HOLLAN; CRUZ et al., 1996). Estudos revelam que o desenvolvimento de interfaces ocupa cerca de 50% do tempo de um projeto nos esforços de planejamento e implementação (MYERS; ROSSON, 1992). Através do emprego de processos, técnicas e ferramentas adequadas, custos incorridos no desenvolvimento de interfaces tendem a cair (MYERS; ROSSON, 1992), enquanto a produtividade do usuário com o sistema tende a crescer (MYERS, 1994).

Do mesmo modo que ocorre com o desenvolvimento de *software* em geral, não há solução milagrosa (*no silver bullet*) que facilite ou agilize o desenvolvimento das interfaces com o usuário (BROOKS JR, 1995; MYERS, 1994). Além disso, *software* para essa porção específica dos aplicativos é reconhecidamente complexo e difícil de implementar (MYERS, 1995). Portanto, quando disponíveis, são inúmeras as vantagens do emprego de ferramentas de *software* para o desenvolvimento da interface com o usuário. Em geral, evidenciam-se através da maior qualidade das interfaces, da maior facilidade e do menor custo em criar e manter o código associado a essa parte do *software* (MYERS, 1995). Tais ferramentas permitem a elaboração rápida de protótipos e facilitam a implementação de interfaces mais consistentes entre si; reduzem a quantidade de código a ser escrito; promovem a modularidade do *software;* isolam as complexidades do sistema gráfico subjacente; aumentam a confiabilidade da interface e facilitam o desenvolvimento de código para múltiplas plataformas (MYERS, 1995).

Uma abordagem específica para apoio ao desenvolvimento das interfaces com o usuário tem recebido crescente atenção por parte da comunidade acadêmica: o desenvolvimento baseado em padrões (WESSON; COWLEY, 2003). Visível à comunidade de engenharia de *software* desde 1998, quando da publicação do artigo *Common Ground* de Jennifer Tidwell (TODD; KEMP; PHILLIPS, 2004), o modelo promove o uso de coleções de padrões como uma ferramenta de apoio na implementação de interfaces. Um padrão descreve uma possível solução a um problema comum de interface, num contexto específico, através da identificação das qualidades invariantes da solução proposta (TIDWELL, 1998). Uma vantagem marcante dessa abordagem está no seu potencial em aproximar os requisitos de alto

nível de um sistema do projeto de sua interface: usuários são apresentados a um conjunto de padrões e discutem alternativas para a implementação das interfaces através dos exemplos disponíveis na coleção (WESSON; COWLEY, 2003).

O desenvolvimento de coleções de padrões é uma etapa fundamental em direção a uma linguagem de padrões que, por sua vez, é o maior objetivo na pesquisa de padrões (VAN WELIE; VAN DER VEER; ELIËNS, [ca. 2002]). A importância dessa atividade foi decisiva para a definição do tema deste trabalho: o desenvolvimento de uma coleção de padrões de interfaces especificamente voltados para sistemas de informação.

1.1 JUSTIFICATIVA

O crescimento do interesse pela abordagem orientada a padrões para o desenvolvimento de interfaces com o usuário, a escassez de coleções de padrões em geral e, em particular, de padrões orientados a aplicativos comerciais, tipicamente sistemas de informação, motivou a definição dos objetivos e do escopo deste trabalho.

O desenvolvimento de uma coleção de padrões viabilizará uma melhor comunicação entre a equipe de modelagem das interfaces, analistas, programadores e futuros usuários. Durante a análise, permitirá que os usuários discutam, de modo informado, as diversas alternativas de implementação das interfaces para cada tela do sistema. Na implementação, os programadores utilizarão os padrões como modelos para implementar as interfaces propriamente ditas. E, ao longo das sucessivas versões do sistema, e também em sua fase de manutenção, os resultados observados durante o uso das interfaces pelos usuários permitirão ajustar essas interfaces para que fiquem consistentes com os padrões, refinar as descrições dos padrões nos aspectos não considerados originalmente mas observados na prática e identificar novos padrões a partir das variações no uso das interfaces.

Outra vantagem relevante que o emprego de padrões oferecerá está na forma como os responsáveis pela modelagem das interfaces e os programadores do sistema realizarão suas tarefas. O profissional de interfaces passará a concentrar

seus esforços mais no trabalho com os padrões- sua identificação, criação e avaliação. A especificação das interfaces individuais dos aplicativos será realizada com o auxílio da coleção de padrões. Aquelas interfaces não cobertas pela coleção serão especificadas individualmente e, caso sejam identificados novos padrões nessas interfaces, serão incorporados à coleção. O programador ficará responsável pelo desenvolvimento dos meios para o uso desses padrões nos aplicativos. Possivelmente, implementando as propriedades e os comportamentos gerais dos padrões em uma biblioteca de classes, deixando os detalhes da implementação de cada interface para código específico no aplicativo. O modelo de desenvolvimento baseado em padrões deverá promover, portanto, um alto grau de independência entre as diversas atividades e partes envolvidas. Num primeiro instante, os profissionais de interfaces elaboram o catálogo de padrões, que definem unidades de interação entre o usuário e o sistema, e os programadores desenvolvem as bibliotecas de classes que implementam esses padrões. Posteriormente, a coleção de padrões é utilizada para auxiliar na especificação das interfaces concretas de aplicativos, que são implementadas através do uso da biblioteca de classes.

Por fim, o compartilhamento de coleções de padrão entre os profissionais de interfaces, programadores e usuário permitirá a comparação entre coleções, promovendo o refinamento de alguns padrões, o surgimento de novos padrões e o abandono de outros. Essas ações promoverão a evolução da qualidade das coleções e resultarão em vantagens imediatas para novos sistemas.

1.2 OBJETIVO GERAL

Elaborar uma coleção de padrões de interfaces gráficas especificamente voltada para sistemas de informação e aplicações baseadas em acesso a dados através de fontes de dados relacionais.

1.3 OBJETIVOS ESPECÍFICOS

a) Examinar as formas de apoio ao desenvolvimento de interfaces gráficas oferecidas por ambientes de desenvolvimento integrados (IDEs).
b) Definir o idioma, ou conjunto de dimensões, a empregar na descrição dos padrões.
c) Identificar padrões através de coleções disponíveis, da literatura e da experiência do próprio autor ao longo do exercício de suas atividades profissionais.
d) Classificar e descrever os padrões identificados.
e) Implementar uma biblioteca de classes para apoio ao desenvolvimento de interfaces baseadas nos padrões identificados, na linguagem de programação Object Pascal.
f) Implementar um sistema de informações baseado na biblioteca de classes de padrões.

1.4 ESCOPO

A coleção elaborada concentra-se em padrões de interfaces gráficas de sistemas de informação e aplicações voltadas para o acesso a dados através de sistemas gerenciadores de bancos de dados (SGBDs). Nesse tipo de aplicação, a interface visual é gerada em tempo de execução pelo próprio sistema. O ambiente típico de execução dessas aplicações é o *desktop* e por isso a coleção não contempla padrões voltados para a *web*.

A implementação de referência foi desenvolvida na ferramenta Borland Delphi 5.0 e, por isso, a biblioteca resultante é suportada nesse ambiente apenas. No entanto, não deve haver dificuldade em portá-la para outras versões do Borland Delphi ou Borland Kylix. Sistemas implementados com essa biblioteca funcionam apenas nos ambientes Windows de 32 bits.

1.5 RESULTADOS

a) Foi realizado um levantamento com 49 ambientes de desenvolvimento e identificados aqueles que oferecem apoio ao desenvolvimento de interfaces gráficas através de construtores de interfaces, assistentes e outras ferramentas.
b) Foi elaborada uma coleção de padrões de interfaces gráficas com 20 padrões: 5 para identificação visual; 2 para controle; 2 para acesso a dados; 10 para edição de dados e 1 para saída de dados. Cada padrão foi identificado, classificado e descrito ao longo de 7 dimensões distintas.
c) Foi implementada uma biblioteca de classes no ambiente Borland Delphi 5.0 para o apoio ao desenvolvimento de interfaces gráficas baseadas nos padrões elaborados. 11 padrões foram implementados completamente na biblioteca, 7 parcialmente, 1 estava disponível de forma nativa no ambiente e apenas 1 não foi implementado.
d) Foi implementado um sistema de informações para automação de uma indústria. O ambiente de desenvolvimento foi o Borland Delphi 5.0. O sistema foi implementado com o apoio da biblioteca de classes desenvolvida no trabalho.

1.6 ORGANIZAÇÃO DO TRABALHO

A revisão bibliográfica, que compreende os Capítulos 2 e 3, estabelece o referencial teórico para o restante do trabalho. O Capítulo 2 apresenta os eventos históricos de maior influência na evolução da interação humano-computador e, em seguida, analisa essa evolução à luz dos eventos anteriormente discutidos. São destacados três paradigmas principais de interação (sistemas *batch*, linha de comando e manipulação direta) e dois menores (sistemas pré-*batch* e sistemas de tela cheia). O Capítulo 3 discute o desenvolvimento de interfaces gráficas do paradigma de manipulação direta a partir de suas tecnologias, processos e ferramentas. Conclui com a apresentação de um levantamento sobre o apoio ao desenvolvimento de interfaces gráficas nos ambientes integrados de desenvolvimento (IDEs). O Capítulo 4 introduz a metodologia empregada para a

elaboração da coleção de padrões de interfaces e para o desenvolvimento de uma implementação dessa coleção. O Capítulo 5 apresenta o trabalho desenvolvido: a coleção de padrões elaborada e a documentação de sua implementação. Discute, também, um modelo de desenvolvimento adequado para o uso da coleção de padrões de interface. O Capítulo 6 examina os resultados do trabalho. O Capítulo 7 encerra com uma análise de tudo que foi realizado, as contribuições e limitações do trabalho, e sugestões para trabalhos futuros. O Apêndice A apresenta a íntegra da coleção de padrões, no formato adequado para sua utilização na prática.

2 EVOLUÇÃO DA INTERAÇÃO HUMANO-COMPUTADOR

O objetivo deste capítulo é compor uma visão ampla da evolução da interação entre homens e computadores. Esse processo construtivo, fundamentado em uma série de marcos históricos do desenvolvimento das tecnologias de *hardware* e *software*, identifica as principais formas de interação humano-computador (IHC) surgidas no curso da história, relacionando-as aos recursos computacionais de *hardware* e *software* disponíveis ao homem em cada momento.

Para tal, são analisados diversos destaques históricos: idéias originais que viriam inspirar desenvolvimentos em trabalhos posteriores, como nos casos do Memex de Bush, do Sketchpad de Sutherland e do NLS de Engelbart; projetos de importância histórica, por introduzirem novas formas de interação entre o homem e o computador, como nos casos da manipulação direta do Sketchpad, do *mouse* do NLS e da área de trabalho do Xerox Alto; contextos históricos, para ilustrar o *hardware* e/ou *software* tipicamente utilizados durante um período ou segmento específico de mercado, como nos casos dos teletipos, dos terminais de vídeo e do sistema X Windows. Esses destaques históricos estabelecem o ponto de partida para a discussão que se segue sobre as sucessivas gerações de interação humano-computador e seus respectivos paradigmas de interação. Nessa discussão, os destaques históricos anteriormente apresentados são novamente recuperados para contextualizar cada forma de IHC e para identificar a motivação para o desenvolvimento de novas formas de interação em substituição a formas até então disponíveis.

Uma vez entendidos os elementos motivadores e a natureza da evolução histórica da interação humano-computador, será possível classificar as ferramentas de software da IHC de forma bem mais precisa.

2.1 DESTAQUES HISTÓRICOS

A evolução da interação humano-computador é complexa, permeada de eventos que aparentam, muitas vezes, não estar relacionados. Quando observados

de forma mais criteriosa, no entanto, pode-se perceber que tais eventos incluem-se no contexto evolutivo da IHC e colaboram de forma decisiva para a formação de uma visão sistêmica e integrada dessa disciplina.

Este capítulo apresenta destaques históricos que servem ao intuito de contextualizar suas diversas partes. São apresentadas ideias, pesquisas e desenvolvimentos que marcaram significativamente a evolução da interação humano-computador. São recuperados alguns contextos históricos sobre certas tecnologias e produtos que permitem o melhor entendimento das discussões sobre a evolução da interação humano-computador que seguem no restante do capítulo.

2.1.1 Memex

A história das interfaces gráficas tem origem no início da década de 30, antes mesmo da invenção do computador digital. Nesse período, o cientista Vannevar Bush elaborou as idéias de um sistema gráfico hipotético chamado Memex (REIMER, 2005). Bush imaginou o Memex como uma mesa de trabalho equipada com dois visores gráficos sensíveis a toque, um teclado e um *scanner* (Figura 1). O Memex deveria arquivar em microfilme todos os livros, registros e comunicações de seu proprietário, permitindo sua posterior consulta de forma rápida e flexível. Para controlar os dados arquivados, o Memex ofereceria percursos através dos dados armazenados, cada um representando uma linha de pensamento. Com o uso desse sistema, que pretendia auxiliar o cientista no seu trabalho de pesquisa, seu proprietário teria acesso contextualizado a uma vasta base de conhecimentos através de conexões semelhantes a *hiperlinks*. Como não havia como colocar tal dispositivo em funcionamento no período em que foram concebidas, as ideias de Bush pouco foram discutidas à época.

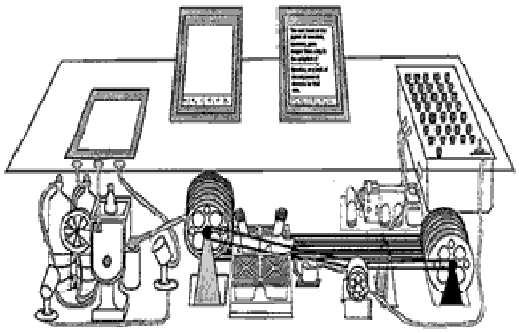


Figura 1 – Memex
Fonte: Ilustração na *LIFE Magazine* (1945).

Aproximadamente em 1937, vários grupos de pesquisa começaram a construir computadores digitais. A Segunda Guerra Mundial ofereceu motivação e financiamento suficientes para produzir tais dispositivos, capazes de realizar desde cálculos balísticos até decodificação de mensagens inimigas. O aperfeiçoamento e a produção em escala comercial de tubos à vácuo ofereceram o mecanismo de rápida alternância necessário para que esses computadores tivessem utilidade prática. A década de 40 marcou o início da operação do primeiro computador totalmente digital, o ENIAC. Oficialmente operacional a partir de fevereiro de 1946, mas em construção desde meados de 1944, o ENIAC não correspondia, em muitos aspectos, ao estágio de desenvolvimento da tecnologia de sua época, precisando ter seus cabos reconectados para rodar novos programas e realizando aritmética decimal ao invés de binária (COMPUTER, 2005). Em 1945, já em meio à revolução do computador digital, Vannevar Bush publicou, no *Atlantic Monthly*, seu artigo seminal, *As We May Think*, reapresentando suas idéias sobre o Memex (BUSH, 1945).

2.1.2 Perfuradoras de cartão

Os computadores dos anos 40 e 50 executavam programas através da reconexão parcial de seus cabos (ENIAC, por exemplo) ou através da leitura de suas instruções a partir de cartões ou fitas de papel perfuradas (Harvard Mark I, por

exemplo) (COMPUTER, 2005). Ainda que a reconexão parcial dos cabos fosse uma prática comum em computadores eletromecânicos anteriores, computadores digitais mais modernos davam preferência ao uso dos cartões e fitas de papel perfuradas. O uso desses cartões perfurados de papelão era uma forma barata e eficiente de armazenamento de dados, já que os cartões ofereciam um registro permanente das transações realizadas.

Os cartões eram perfurados em uma máquina especial chamada perfuradora (*keypunch*), semelhante a uma máquina de datilografia. Em modelos mais avançados, como a IBM 026 (Figura 2), o texto era impresso no topo do cartão durante a digitação, permitindo a conferência dos dados entrados pelo operador humano. Equipamentos eletromecânicos conhecidos como *unit record equipment*, *electric accounting machines* (EMA), ou máquinas de tabulação, eram utilizados para perfurar, ordenar, tabular e imprimir cartões. Essas máquinas já realizavam tarefas de processamento de dados sofisticadas, bem antes do advento dos computadores. Sua programação era realizada através do uso de cabos em paineis de controle apropriados chamados *plugboards* (COMPUTER, 2005). A IBM 407 Accounting Machine (Figura 3) foi a última e mais popular dessas máquinas. Máquinas de tabulação foram tão populares na primeira metade do século XX quanto os computadores se tornariam na segunda.

Figura 2 – Perfuradora de cartões IBM 026
Fonte: Da Cruz (2001).

Figura 3 – IBM 407 Accounting Machine
Fonte: Da Cruz (2001).

Sistemas baseados em cartões seriam abandonados definitivamente em favor de sistemas baseados em discos e terminais interativos anos mais tarde, na década de 70.

Em 1957, num artigo da *Automatic Control Magazine*, Bob Bemer descreveu pela primeira vez o conceito de compartilhamento de tempo (*time-sharing*), a idEia de que um único computador oferece acesso interativo, aparentemente simultâneo, a múltiplos usuários através do compartilhamento do tempo de processamento da máquina (BROOKSHEAR, 2000). Esse artigo influenciou uma série de pesquisas que culminaram com o desenvolvimento de diversos protótipos e sistemas comerciais de *time-sharing* nas décadas de 50 e 60. Em 1961, um protótipo do *Computer Time Sharing System* (CTSS), sistema pioneiro na implementação do conceito de *time-sharing*, havia sido desenvolvido e testado (COMPUTER, 2005; VLECK, 2000, 2005). Logo, ficou claro que as tecnologias eletromecânicas e de tubos a vácuo da época eram inadequadas à demanda computacional de um sistema de *time-sharing:* computadores rápidos, baseados em transistores eram necessários (COMPUTER, 2005; VLECK, 2000, 2005). O CTTS permaneceria em uso até 1973. Implementado em 1964, o *Dartmouth Time-Sharing System* (DTSS) foi outro sistema de *time-sharing* pioneiro de certo sucesso comercial; foi utilizado por um curto período, da segunda metade da década de 60 ao início da década de 70 (TANENBAUM, 2001; VLECK, 2000, 2005). O ano de 1964 também marcou o início do desenvolvimento do MULTICS, um dos mais influentes sistemas operacionais de

seu tempo que, além de *time-sharing*, implementava uma série de outros conceitos avançados: sistema de arquivos integrado ao sistema de memória; memória segmentada; memória virtual hierárquica transparente, em três níveis; implementação em linguagem de alto nível, PL/1, com apenas umas poucas partes implementadas em *assembler*; suporte a diversas linguagens de programação (TANENBAUM, 2001; VLECK, 2000, 2005). Alguns sistemas influenciados pelo MULTICS incluem: UNIX; IBM TSS/360; GCOS 6, 7 e 64, da Honeywell; Primos; VOS, da Stratus; Domain, da Apollo; outros como Amber, GEMSOS, TENEX e TOPS (VLECK, 2000, 2005).

2.1.3 Time-Sharing

Em 1957, num artigo da *Automatic Control Magazine*, Bob Bemer descreveu pela primeira vez o conceito de compartilhamento de tempo (*time-sharing*), a idéia de que um único computador oferece acesso interativo, aparentemente simultâneo, a múltiplos usuários através do compartilhamento do tempo de processamento da máquina (BROOKSHEAR, 2000). Esse artigo influenciou uma série de pesquisas que culminaram com o desenvolvimento de diversos protótipos e sistemas comerciais de *time-sharing* nas décadas de 50 e 60. Em 1961, um protótipo do *Computer Time Sharing System* (CTSS), sistema pioneiro na implementação do conceito de *time-sharing*, havia sido desenvolvido e testado (COMPUTER; VLECK, 2000, 2005). Logo, ficou claro que as tecnologias eletromecânicas e de tubos à vácuo da época eram inadequadas à demanda computacional de um sistema de *time-sharing:* computadores rápidos, baseados em transistores eram necessários (COMPUTER; VLECK, 2000, 2005). O CTTS permaneceria em uso até 1973. Implementado em 1964, o *Dartmouth Time-Sharing System* (DTSS) foi outro sistema de *time-sharing* pioneiro de certo sucesso comercial; foi utilizado por um curto período, da segunda metade da década de 60 ao início da década de 70 (TANENBAUM, 2001, p. 672; VLECK, 2000, 2005). O ano de 1964 também marcou o início do desenvolvimento do MULTICS, um dos mais influentes sistemas operacionais de seu tempo que, além de *time-sharing*, implementava uma série de outros conceitos avançados: sistema de arquivos integrado ao sistema de memória; memória segmentada; memória virtual hierárquica transparente, em três níveis; implementação em linguagem de alto nível, PL/1, com apenas umas poucas partes

implementadas em *assembler*; suporte a diversas linguagens de programação (TANENBAUM, 2001, p. 12-13, 672; VLECK, 2000, 2005). Alguns sistemas influenciados pelo MULTICS incluem: UNIX; IBM TSS/360; GCOS 6, 7 e 64, da Honeywell; Primos; VOS, da Stratus; Domain, da Apollo; outros como Amber, GEMSOS, TENEX e TOPS (VLECK, 2000, 2005).

2.1.4 Sistemas Operacionais

Os primeiros computadores não possuíam qualquer tipo de sistema operacional. Programas eram carregados e sua execução iniciava até que chegasse ao fim ou, mais comumente, que um erro ocorresse. Cada processo tinha total controle sobre a máquina, inclusive dos dispositivos de entrada e saída e, portanto, tinha que incluir o código para manipular esses dispositivos. Com o tempo, foram sendo disponibilizadas bibliotecas de código que se ligavam ao programa do usuário para auxiliar na manipulação dos dispositivos de entrada e saída. Incidentalmente, essas bibliotecas deram lugar a um programa que era carregado antes do primeiro processo ser executado, lia o processo seguinte, controlava sua execução, fazia as tarefas de limpeza do ambiente, registrava as métricas de uso (páginas impressas, cartões perfurados, cartões lidos, espaço de armazenamento utilizado, etc) e seguia então para um novo processo. Monitores, como eram conhecidos esses programas, foram introduzidos experimentalmente em 1957 e representaram o primeiro passo em direção a um sistema operacional.

Até os anos 60, cada vez que um fabricante lançava um novo sistema, introduzia um sistema operacional próprio, totalmente novo. Isso mudou, no entanto, quando a IBM lançou, em 1965, os computadores *mainframe* da série System/360. Todos os computadores da família System/360 rodavam o sistema operacional OS/360, a despeito das enormes diferenças em performance observadas entre as máquinas (TANENBAUM, 2001, p. 10-11).

Nos anos 60, o Massachusetts Institute of Technology, o AT&T Bell Labs e a General Electric trabalharam em conjunto num sistema operacional experimental chamado MULTICS (TANENBAUM, 2001, p. 672-674). Posteriormente, a AT&T Bell Labs desligou-se do projeto. Mais tarde, Dennis Ritchie e Ken Thompson, desenvolvedores do AT&T Bell Labs, iniciaram um projeto para o desenvolvimento

de um sistema operacional para o DEC PDP-7 (TANENBAUM, 2001). O novo sistema, chamado de UNICS (Uniplexed Information and Computing System), deveria ser multitarefa e suportar um usuário. O nome era uma alusão ao projeto MULTICS (de onde se originaram várias ideias) e, ao mesmo tempo, uma referência fonética ao termo *eunuchs* (eunucos, em inglês), sugerindo que o UNICS fosse uma versão "castrada" do MULTICS (TANENBAUM, 2001). Para fugir à homofonia, o nome foi então alterado para UNIX. O sistema operacional UNIX foi oficialmente batizado e rodou pela primeira vez num PDP-11/20, em 1970 (TANENBAUM, 2001, p. 672-673; UNIX, 2005).

2.1.5 Tecnologia de Vídeo

Os primeiros dispositivos de vídeo usados para interação gráfica com computadores empregavam monitores vetoriais, uma espécie de CRT (do inglês *cathod ray tube*, tubo de raios catódicos, em português) semelhante a um osciloscópio. Nesse tipo de dispositivo, o raio traçava linhas retas entre pontos arbitrários, atualizando o dispositivo na mais alta taxa de repetição possível. Tal dispositivo de vídeo foi utilizado em muitos computadores, assim como em jogos de fliperama do final dos anos 70 a meados dos anos 80.

Dispositivos de vídeo vetoriais para computadores não sofriam de forma perceptível os efeitos de *aliasing* (efeito da conversão analógica-digital de um sinal de vídeo e que pode gerar contornos serrilhados ou padrões *moiré* de interferência na imagem) e *pixelation* (efeito em que *pixels*, ou pontos de cores, individuais de uma imagem ficam visíveis ao olho), mas eram limitados em funcionalidade, pois, só podiam exibir o contorno de figuras e uma quantidade relativamente limitada de texto em formato grande (FOLEY, 1990; CATHOD RAY TUBE, 2005).

Em 1973, no Centro de Pesquisa da Xerox em Palo Alto (PARC), foi construída uma máquina que utilizava um dispositivo de vídeo gráfico baseado em mapas de bits ao invés de vetores. Dispositivos vetoriais desenhavam explicitamente pontos, linhas, arcos e formavam caracteres no dispositivo de vídeo, um processo relativamente lento, mas econômico, pois precisava de pouca memória. A abordagem do computador da Xerox, chamado de Alto, era a de determinar cada ponto na tela a partir de um endereço único na memória. Nesse modelo, quase todas as operações gráficas poderiam ser realizadas pela simples cópia de blocos

de dados entre posições de memória, uma técnica batizada de BitBlt (*Bit Block Transfer*) por seus inventores. Essa técnica simplificou a programação das interfaces gráficas enormemente e teve papel vital na viabilização da interface gráfica do Alto (RAYMOND, 2004).

2.1.6 Teletipos e Terminais de Vídeo

Teletipo, ou TTY, é uma espécie de máquina de escrever eletromecânica utilizada para comunicar mensagens escritas entre dois pontos através de um canal elétrico de comunicação. Utilizado desde 1902 na transmissão de mensagens de telégrafo, já era amplamente empregado em inúmeras corporações nos anos 20. A adaptação do teletipo, originalmente desenvolvido para transmissão e recepção de mensagens de telégrafo, para servir como dispositivo de entrada e saída para computadores, foi uma consideração tanto lógica quanto econômica: tratava-se de uma tecnologia madura e reconhecidamente eficiente para a transferência de informações através de fios elétricos; sua fabricação em escala comercial traduzia-se em baixo custo; além disso, teletipos ofereciam uma interface familiar para a maioria dos engenheiros e operadores da época interagirem com os computadores (RAYMOND, 2004).

Alguns dos primeiros computadores (o LGP-30, de meados da década de 50, por exemplo) utilizaram teletipos (TTY) como dispositivos de entrada e/ou saída. Teletipos tornaram-se ainda mais importantes com o advento das interfaces baseadas em linhas de comando. Em tais sistemas, os teletipos eram utilizados para emitir comandos após o surgimento de um *prompt* de comando (caractere ou seqüência de caracteres indicativas de que o sistema está pronto para aceitar o próximo comando). A fita de papel podia ser usada tanto para preparar entradas para sessões *off-line* (quando o operador não se encontra à frente do terminal) quanto para capturar as saídas do processamento. Um dos mais populares teletipos foi o ASR-33 (figura 4), de 1963, com centenas de milhares de unidades produzidas (COMPUTER MUSEUM, 2005; TELETYPE MACHINES, 2005).

Figura 4 – O teletipo IBM ASR-33
Fonte: Raymond e Landley (2004).

Terminais são dispositivos de hardware eletrônicos ou eletromecânicos, utilizados para entrada e exibição de dados de um computador ou sistema computacional. Os primeiros terminais eram teletipos eletromecânicos, como o ASR-33. No início da década de 70, a indústria percebeu a viabilidade econômica dos terminais de vídeo. Esses dispositivos passaram a substituir os então dominantes sistemas baseados teletipos e tornaram mais interativo o uso dos computadores pois gerar caracteres num terminal de vídeo era muito mais rápido do que movimentar a cabeça de impressão de um teletipo. Além disso, os custos com papel e tinta associados ao teletipo foram cortados drasticamente.

Os primeiros terminais de vídeo, chamados de *glass TTY* (teletipos de vidro), não possuiam capacidades gráficas e conectavam-se a *mainframes*. Tipicamente, a comunicação com o *mainframe* se dava através de uma linha serial, geralmente através de uma interface RS-232. Mais tarde, terminais inteligentes como o VT52 e o

VT100 (Figura 5) foram introduzidos. Eles eram chamados de inteligentes por interpretarem seqüências de escape e caracteres de controle para posicionar o cursor e controlar a exibição na tela e por responderem a teclas de função (COMPUTER TERMINAL, 2005). Com essas inovações, os terminais tornaram-se mais capazes, permitindo os usuários explorassem toda a área da tela para a interação com o terminal.

Figura 5 – Terminal VT100
Fonte: <http://vt100.net/vt_history>(jul. 2005).

Concomitante com o desenvolvimento dos terminais de vídeo, o avanço das tecnologias de vídeos gráficos de varredura e os avanços alcançados nas pesquisas da Xerox em Palo Alto permitiram a criação de terminais que exibiam não apenas texto mas também gráficos de varredura (do inglês *raster graphics*), uma imagem digital ou mapa de bits representando uma matriz de *pixels* num dispositivo de vídeo como um monitor, em papel ou em algum outro dispositivo). O computador mandava comandos de desenho para o terminal e este encaminhava as entradas do usuário (do teclado e outros dispositivos) para o computador (RAYMOND, 2004).

2.1.7 Sketchpad

Desenvolvido no Lincoln Laboratory do MIT por Ivan Sutherland, o projeto Sketchpad é um importante marco na história das interfaces gráficas (FOLEY, 1990, p. 8; TUCK, 2001; RAYMOND, 2004). Na sua apresentação em 1963, Sutherland demonstrou diversas inovações, das quais a mais importante era a interface gráfica de manipulação direta (também conhecida como GUI), onde os objetos eram manipulados em tempo real, diretamente na tela, com o auxílio de uma caneta ótica e uma mesa de controle (figura 6). O sistema, visionário em muitas formas, foi de grande influência para diversas pesquisas nos anos que seguiram.

Figura 6 – O Sketchap, com parte de uma ponte em destaque na tela.
Nota: O operador segura uma caneta ótica. O painel atrás do operador e os Botões, a sua rente, também fazem parte da interface com o sistema
Fonte: Sutherland (1963).

2.1.8 The Mother of All Demos

Em 1962, Douglas Engelbart publicou sua filosofia e agenda de pesquisa num relatório de pesquisa chamado *Augmenting Human Intellect: A Conceptual Framework*. Nos anos que se seguiram, trabalhando no laboratório que ele próprio

fundara, o Augmentation Research Center (ARC) no Standford Research Institute (SRI), ele coordenou o desenvolvimento do NLS, ou *oNLine System*, um revolucionário sistema de colaboração (STIM, 1997-1998; TUCK, 2001; RAYMOND, 2004; REIMER, 2005). Em 1968, Engelbart fez a demonstração do NLS no Centro de Convenções de São Francisco, aquela que viria ser conhecida como *the mother of all demos* (a mãe de todas as demonstrações). Nessa demonstração, Engelbart apresentou para os mais de 1.000 profissionais da platéia uma série de inovações, dentre as quais se destacavam: paradigma do papel (texto preto sobre fundo branco, arquivos dentro de pastas e uma área de trabalho), *mouse*, monitor de vídeo baseado em gráficos por varredura, *e-mail*, mensagens instantâneas, videoconferência, hipertexto, múltiplas janelas lado a lado, *groupware*, organização hierárquica de informações, edição de documentos em tela cheia, sistema de ajuda sensível a contexto, além de um sistema de apresentação (STIM, 1997,1998; TUCK, 2001; RAYMOND, 2004; REIMER, 2005).

Figura 7 – O Sistema NLS em execução
Fonte: NLS (Wikipedia).

Figura 8 – Dispositivos de Entrada do NLS: mouse, teclado e keyset
Fonte: NLS (Wikipedia).

2.1.9 O Xerox Alto

Apesar da demonstração do NLS, em 1968, ter mostrado o que era possível em termos de interfaces gráficas, a tradição de engenharia das interfaces gráficas atuais nasceu nas pesquisas desenvolvidas no Palo Alto Research Center (PARC) da Xerox, no início da década de 70 (RAYMOND, 2004). Uma das primeiras invenções do PARC foi a impressora a laser (REIMER, 2005). Necessitando de uma forma gráfica de preparar documentos para impressão a laser, e ainda inspirados pela demonstração de Engelbart em 1968, os pesquisadores do PARC começaram, em 1973, a desenvolver uma máquina chamada Alto. O Alto apresentava um dispositivo de vídeo gráfico (Figura 9), baseado em mapas de bits, com as dimensões e a orientação de uma folha impressa (REIMER, 2005). Além disso, possuia um teclado, uma versão modernizada do *mouse* de Engelbart (ainda com três botões) e uma conexão Ethernet embutida (o protocolo Ethernet também havia sido desenvolvido no PARC). O Xerox Alto havia sido projetado para ser utilizado de forma dedicada, por um usuário apenas (RAYMOND, 2004). Não chegou a ser chamado de estação de trabalho ou computador pessoal, mas, certamente, é o antecessor de ambos.

A despeito dos inúmeros avanços, a interface gráfica do Xerox Alto era monocromática. Já apresentava a maioria dos elementos presentes nas interfaces modernas: ícones, janelas sobrepostas, barras de rolagem, botões de radio- o único elemento ausente era o menu *pull-down*, posteriormente introduzido pelo Apple Lisa

em 1979 (RAYMOND, 2004).

Figura 9 – O Xerox Alto: monitor, teclado, *mouse* e *keyset*
Fonte: Raymond (2004).

Os primeiros aplicativos escritos para o Alto possuíam interfaces cruas e apenas levemente gráficas. O Neptune, por exemplo, era utilizado para gerenciar o conteúdo dos discos; exibia diretórios e arquivos em regiões identificadas na tela por contornos (Figura 10). O Bravo, primeiro processador de textos gráfico WYSIWYG (do inglês *What You See Is What You Get*, alusão ao fato de que o que é exibido na tela representa fielmente o que será impresso), podia exibir diferentes fontes e tamanhos de texto na mesma tela, ao mesmo tempo (Figura 11), e era controlado pelo uso combinado de *mouse* e teclado (REIMER, 2005). O Draw, finalmente, era um sistema de editoração gráfica para elaboração de imagens através de primitivas como curvas, linhas e textos; essas primitivas podiam ser manipuladas na tela de forma semelhante àquela em que o Sketchpad operava (Figura 12).

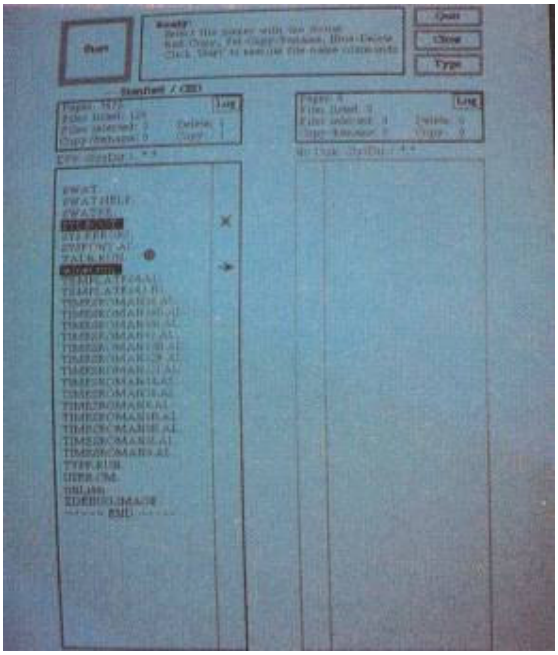


Figura 10 – Neptune, o gerenciador de arquivo do Xerox Alto
Fonte: Wadlow (1981).

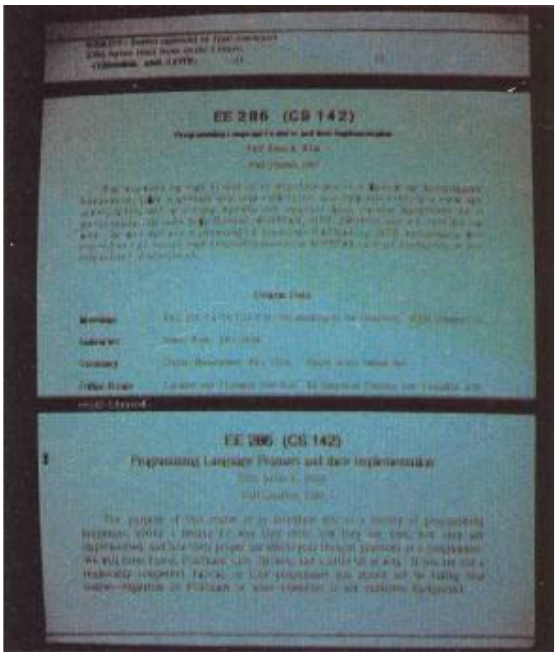


Figura 11 – Bravo, o editor de texto Wysiwyg do Xerox Alto
Fonte: Wadlow (1981).

A disparidade nas abordagens de implementação das interfaces gráficas dos primeiros aplicativos incentivou os pesquisadores do Xerox PARC a buscarem uma forma de aumentar a consistência da interface para novos aplicativos. Para isso, e utilizando as idéias de passagem de mensagens da linguagem Simula, desenvolveram o Smalltalk, uma das primeiras linguagens orientada a objetos da história (TUCK, 2001; REIMER, 2005). O Smalltalk era disponibilizado ao usuário do Alto através de um ambiente gráfico e de desenvolvimento numa só ferramenta- os aplicativos eram desenvolvidos e rodavam no mesmo ambiente (figura 13). Em 1981, oito anos após o lançamento original do Alto, a Xerox lançou o Xerox Star 8010 Document Processor, uma versão comercial reduzida e de pouco sucesso do Alto (RAYMOND, 2004; TUCK, 2001; REIMER, 2005). O preço de mercado era US$17.000,00.

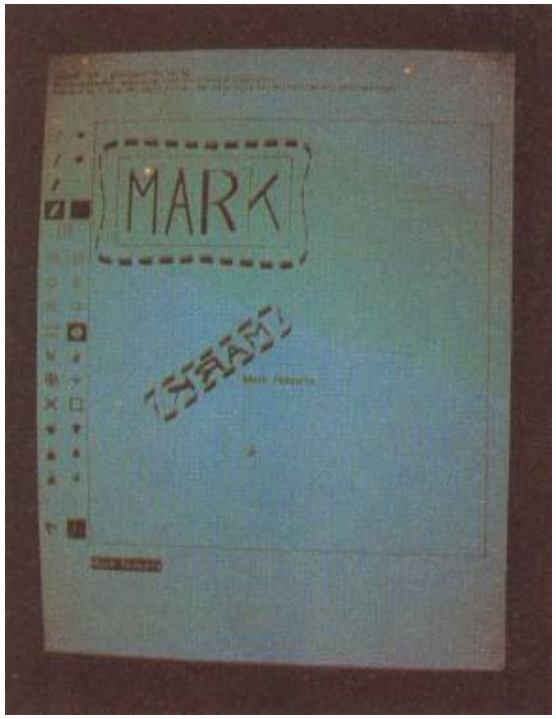


Figura 12 – Draw, editor gráfico do Xerox Alto
Fonte: Wadlow (1981).

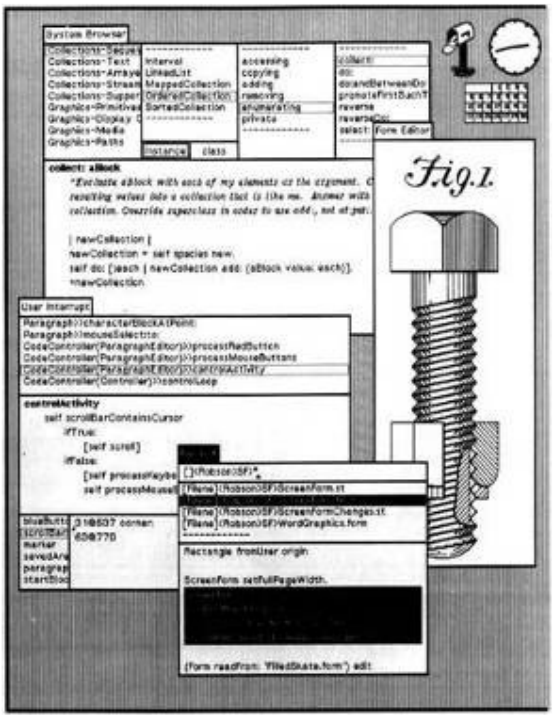


Figura 13 – O ambiente Smalltalk do Xerox Alto
Fonte: Reimer (2005).

O Xerox Alto marcou o início dos sistemas com interfaces gráficas modernas baseadas em janelas, ícones, menus e apontador (em geral, o *mouse*), também conhecidas como WIMP (do inglês *windows, icons, menus and pointer*). Desde então, a maioria dos desenvolvimentos em interfaces gráficas em sistemas comerciais concentrou-se, em grande parte, em melhorias na qualidade dos gráficos e não na introdução de novos modelos de interação. Nesses mais de 30 anos, evoluíram os paradigmas de modelagem de sistemas, as arquiteturas de software, os padrões e ferramentas utilizadas no desenvolvimento das interfaces gráficas, dentre outros, mas a maioria dos aplicativos hoje em dia ainda utiliza interfaces WIMP.

2.1.10 As Estações de Trabalho da Sun Microsystems

Em 1982, a Sun Microsystems lançou um computador híbrido baseado no Unix, com um projeto de *hardware* semelhante ao do Xerox Alto (Figura 14). Muito bem aceito no mercado, esse computador viria a definir um dos mais bem sucedidos padrões de sistemas na história da indústria dos computadores- o padrão das estações de trabalho. Estações de trabalho eram máquinas Unix equipadas com dispositivos de vídeo gráficos de alta resolução, baseados em mapas de bits, e

conexão Ethernet embutida. As primeiras estações foram equipadas com processadores Motorola 68000; mais tarde, foram equipadas com processadores de 32 e 64 bits, quando os computadores pessoais ainda faziam a transição dos processadores de 8 para os de 16 bits (RAYMOND, 2004). Os maiores mercados das estações de trabalho foram empresas de projetos com CAD/CAM, organizações de desenvolvimento de software e prestadoras de serviços na área financeira. Essa plataforma foi a responsável, também, pela popularização do sistema Unix fora do ambiente científico.

Figura 14 – Estação de Trabalho Sun-1
Fonte: Computer History Museum.

2.1.11 O Apple MacIntosh

O MacIntosh foi lançado pela Apple em 1984, tornando o estilo de interface do Xerox Alto pela primeira vez acessível ao usuário doméstico. Lançado por apenas US$2.495,00, uma fração do valor do Xerox Star 8010, o MacIntosh teve um impacto bastante profundo no estilo das interfaces gráficas, já que não foi apenas um experimento de laboratório, mas um bem de consumo de grande sucesso comercial. O Mac, como era conhecido, provou que interfaces gráficas tinham o apelo comercial necessário para vender computadores às massas. E assim, acabou criando um mercado altamente competitivo que viria a crescer e se estabelecer nos anos seguintes (RAYMOND, 2004; TUCK, 2001; REIMER, 2005). As Figuras 15 e 16 a seguir ilustram a diferença entre as interfaces do Xerox Star e do Apple Macintosh.

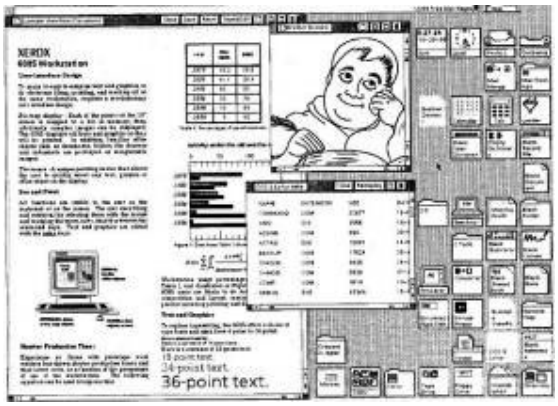


Figura 15 – Xerox Star 8010, 1981
Fonte: HISTORY of the graphical user Interface (WIKIPEDIA, 2005).

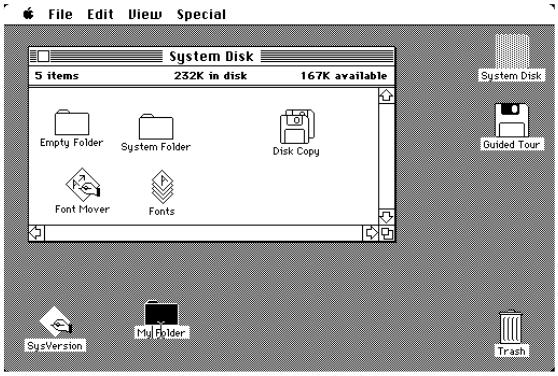


Figura 16 – Apple MacIntosh, 1984
Fonte: HISTORY of the graphical user Interface (WIKIPEDIA, 2005).

2.1.12 O SISTEMA X WINDOWS

O X Windows, ou simplesmente X, é um sistema de janelas para dispositivos de vídeo gráficos baseados em mapas de bits. Seu desenvolvimento foi iniciado em 1984, no MIT, inspirado pelo sistema de janelas W da Universidade de Standford e pelo ambiente SunView da Sun Microsystems (TUCK, 2001). Plataforma independente de *hardware* e de fornecedor, o X Windows oferece o *framework* básico de um ambiente gráfico, suportando o desenho e a movimentação de janelas na tela, a interação com o *mouse* e com o teclado (MYERS, 2004; FOLEY, 1990).

Deliberadamente, no entanto, o X não possui especificações de *widgets* para interação com o usuário, como botões, menus, caixas de texto, etc. Essa separação entre mecanismo e política permite que as decisões estéticas e políticas do desenvolvimento das interfaces gráficas no X sejam tomadas no lado da aplicação, possivelmente através do uso de *toolkits* (bibliotecas de código que suportam o desenvolvimento de interfaces gráficas através de chamadas de APIs). As primeiras versões amplamente utilizadas foram a X10, de 1985-86, e a X11, de 1987 (RAYMOND, 2004). A figura a seguir mostra o X Windows em ação com o gerenciador de janelas Window Maker.

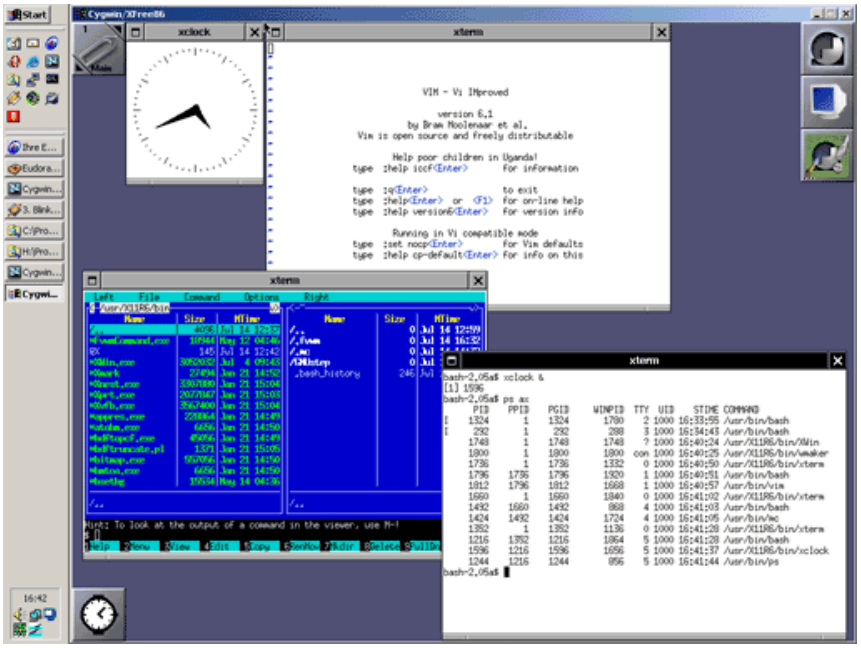


Figura 17 – O ambiente X Windows com o Window Maker
Fonte: HISTORY of the graphical user interface (WIKIPEDIA, 2005).

A partir de 1987-88, o X Windows estabeleceu-se como o ambiente gráfico padrão para o Unix. Sua maior vantagem frente aos demais sistemas competidores era sua natureza aberta (*open source*). Na condição de *open source*, o X era suportado por um consórcio de empresas, dentre as quais se incluíam os maiores vendedores do Unix da época. Outras características marcantes do X Windows também foram essenciais para seu sucesso. Além das mencionadas independência

de fornecedor e *hardware*, e a separação entre mecanismo e política, o X utilizava uma arquitetura cliente/servidor para sua comunicação e, dessa forma, permitia a execução distribuída de aplicações através de um ambiente de rede (MYERS, 2004; FOLEY, 1990). No X, cliente era a aplicação em execução na máquina local ou remota e servidor era o terminal gráfico que disponibiliza os serviços de exibição às aplicações clientes. No caso do X, os termos cliente e servidor são atribuídos na perspectiva do aplicativo em execução. Assim, é possível interagir com diversos aplicativos numa estação X de forma transparente- alguns rodando localmente e outros remotamente.

O X é considerado o primeiro avanço fundamental em infra-estrutura de interfaces gráficas a alcançar ampla aceitação no mercado desde os tempos do Xerox Alto. Ao contrário dos sistemas monolíticos da época, o X tinha uma infra-estrutura cliente/servidor. Além disso, na melhor tradição Unix, o protocolo X promovia a portabilidade e a clara separação de funções (RAYMOND, 2004).

O X é hoje um dos mais populares ambientes gráficos existentes: é padrão *de facto* para as plataformas baseadas em UNIX e Linux; a partir da versão 10.0, é também o ambiente gráfico dos sistemas Mac OS; inúmeras outras plataformas adotaram o X como seu ambiente gráfico principal ou dispõem de *software* para integrar o X Windows em seus ambientes (X.ORG, [ca. 2004]). A comunidade de usuários do X já ultrapassa a marca de 30 milhões (X.ORG, [ca. 2004]).

2.1.13 A Revolução de 32 Bits

A Intel lançou seu primeiro processador de 32 bits para o PC, o 386, em 1987. A ampla disponibilidade desse processador permitiu que as máquinas Unix fossem, pela primeira vez na história, construídas com componentes de *hardware* baratos. Em consequência, o *hardware* de 32 bits do PC acabou sendo responsável pelo encolhimento do mercado de estações de trabalho, de forma cada vez mais rápida a partir de 1990. As estações de trabalho foram então gradualmente substituídas por PCs Unix equipados com monitores e placas de rede. A comunicação dessas máquinas com *mainframes*, ou servidores, passou a acontecer através da emulação em software dos antigos terminais como o VT100.

2.1.14 O Microsoft Windows

O sistema operacional Windows tornou-se, assim como o Macintosh antes, um grande sucesso de mercado. Mas isso não aconteceu antes do lançamento de duas versões de pouco impacto do Windows: a 1.0 de 1985 (figura 18), e a 2.0 de 1987 (Figura 19).

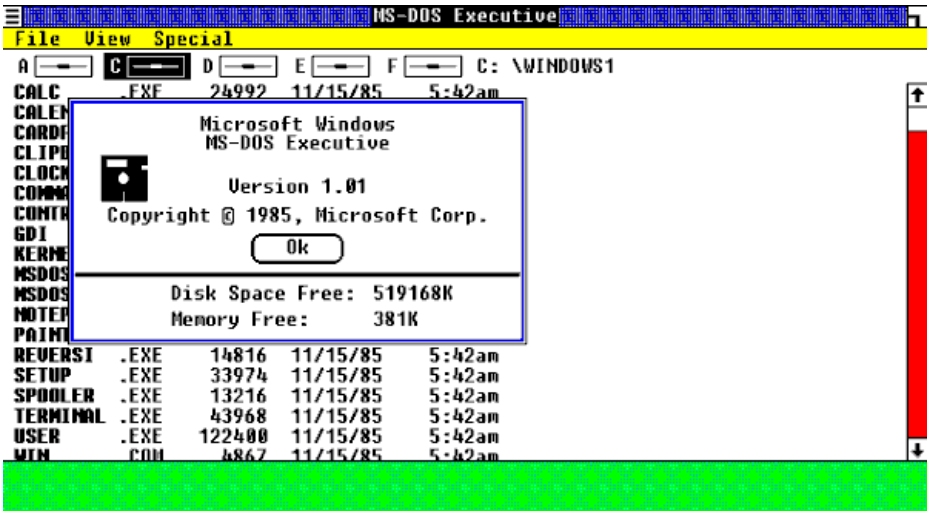


Figura 18 – Microsoft Windows 1.0
Fonte: Windows 1.0 (WIKIPEDIA, 2005).

A despeito do fracasso de seus antecessores, os sistemas Windows 3.0, 3.1 e 3.11 (Figura 20), lançados em 1990, 1992 e 1993, respectivamente, atingiram níveis sem precedentes de popularidade no mercado de ambientes gráficos para o PC (REIMER, 2005; RAYMOND, 2004; TUCK, 2001). Em 1995, o lançamento do Windows 95, primeiro sistema operacional a tirar vantagem dos processadores de 32 bits da Intel, deu início a um sucesso ainda maior. Sua interface gráfica (Figura 21) era muito semelhante à do Mac, com apenas algumas adições originais da Microsoft.

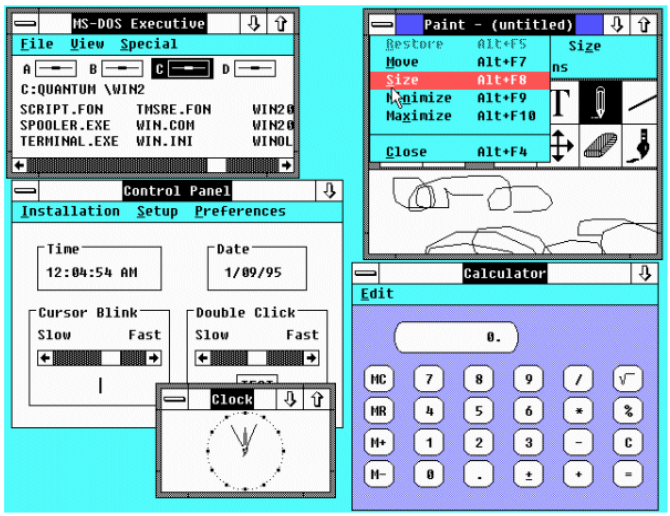


Figura 19 – Microsoft Windows 2.0
Fonte: Windows 2.0 (WIKIPEDIA, 2005).

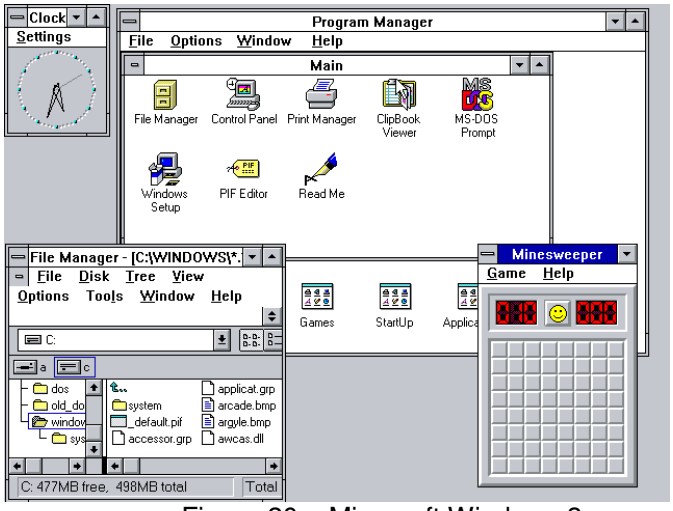


Figura 20 – Microsoft Windows 3.x
Fonte: Windows 3.x (WIKIPEDIA, 2005).

A maior inovação promovida pela Microsoft com os Windows 3.1 e 95 foi o Microsoft Bob, uma coleção de programas que tomavam controle do ambiente do Windows e modulavam sua interface gráfica em algo presumivelmente mais amigável ao usuário do que a clássica área de trabalho introduzida pelo Xerox Alto. A idéia do Bob (Figura 22) era substituir a metáfora do ambiente de trabalho herdada do Xerox Alto por um ambiente virtual animado no qual programas e documentos eram representados por objetos familiares do dia-a-dia (RAYMOND, 2004). O fracasso do Bob foi devastador – tanto que o único legado do Bob nos atuais *softwares* da Microsoft limita-se aos assistentes animados nos aplicativos do

Microsoft Office (RAYMOND, 2004).

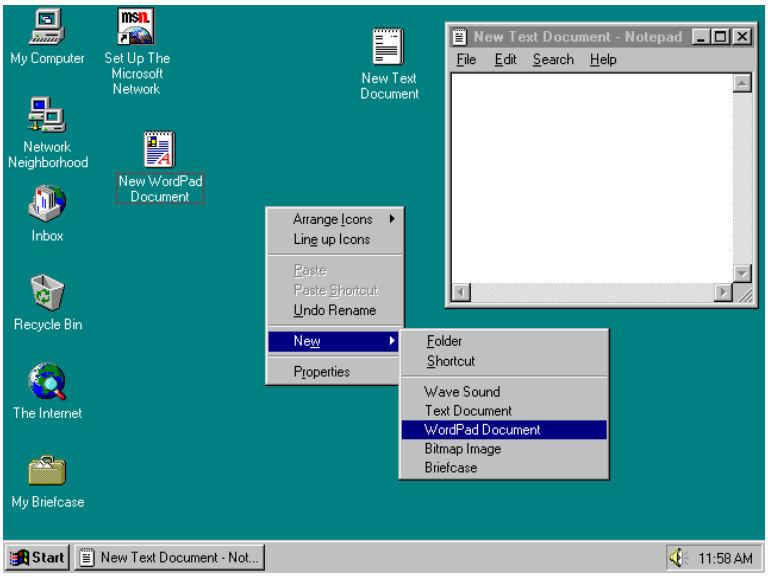


Figura 21 – Microsoft Windows 95
Fonte: Lineback (2001, 2005).

Figura 22 – Microsoft Bob
Fonte: Lineback (2001, 2006).

2.1.15 O Java

O Java, linguagem de programação orientada a objetos, foi desenvolvido em 1991 por James Gosling e outros engenheiros como parte do projeto Green da Sun Microsystems (BYOUS, 1998). No início de 1995, apenas umas poucas cópias binárias alfa do Java rodavam fora da Sun, mas a equipe de engenheiros começava

a preparar o lançamento da plataforma Java- a linguagem e demais ferramentas como a máquina virtual Java, a biblioteca de classes e outros utilitários. Em março desse mesmo ano, a Netscape anunciou oficialmente suporte ao Java no seu Netscape Navigator, o navegador mais amplamente utilizado àquela época (BYOUS, 1998). O Java foi oficialmente introduzido no mercado em maio de 1995 e, rapidamente, tornou-se uma das plataformas de tecnologia de maior sucesso da história (BYOUS, 1998). Gosling havia definido o sucesso da primeira versão da plataforma Java através da quantidade de *downloads* anuais esperados: 10.000; no seu terceiro aniversário, em 1998, 10.000 era a quantidade de *downloads* diários regularmente registrados pela plataforma (BYOUS, 1998). O conceito por trás do Java era uma linguagem de programação em que o código era escrito uma só vez e poderia ser executado em qualquer plataforma- quer fosse Windows, Linux, Unix, Macintosh ou qualquer outro sistema, bastando que tivesse uma máquina virtual Java para interpretar o código intermediário gerado pelo compilador Java (RAYMOND, 2004; FLANAGAN, 2002). Ficou evidente que um dos problemas de tal ambição era fazer as interfaces gráficas se comportarem da mesma forma nas diversas plataformas suportadas (RAYMOND, 2004). Para resolver essa questão, o Java implementou seu próprio ambiente gráfico- não apenas um gerenciador de janelas como no caso do X Windows, mas um *toolkit* completo, juntamente com alguns serviços de mais alto nível (RAYMOND, 2004). Os desenvolvedores do Java acabaram seguindo a mais pura tradição Unix e desenvolveram duas camadas gráficas independentes: os *toolkits* Swing (SUN, [ca. 1998]) e AWT (SUN, 2004).

2.1.16 O Linux e o Movimento Open Source

Com a explosão da Internet em 1993-1994, as deficiências dos sistemas baseados no PC em relação à computação em rede tornavam-se cada vez mais flagrantes. Sistemas como o Mac e o Windows não possuíam provisões para multitarefa preemptiva ou suporte real a múltiplos usuários – eles eram basicamente sistemas pessoais para um único usuário. Ao mesmo tempo, os fornecedores de Unix, sistema equipado com tais recursos, não estavam interessados em entrar no mercado de varejo de baixas margens de lucro (RAYMOND, 2004). E, enquanto eles

hesitavam em meio a essa revolução, o movimento *open source* ganhava momento entre os desenvolvedores Unix e os desbravadores da Internet. O produto chave dessa revolução foi o sistema operacional Linux (RAYMOND, 2004).

Nos anos de 1996-1998, a emergência do movimento *open source* e sua visibilidade ao público em geral abalou a indústria de informática como nenhum outro movimento na década anterior. O surgimento e/ou crescimento de vários projetos *open source* permitiu a criação de versões abertas para quase todas as classes de software proprietário existentes. Com isso, muitos produtos comerciais acabaram não se adaptando à nova situação de mercado e foram definitivamente descontinuados. Um exemplo ilustrativo está diretamente relacionado com o X Windows (RAYMOND, 2004). Devido a sua arquitetura, o X permitia a seleção do *toolkit* a ser utilizados pelos aplicativos e, por isso, jamais se conseguiu uma padronização das interfaces gráficas desenvolvidas através do X. Com a criação dos projetos dos *toolkits* Qt (TROLLTECH, 1994) e GTK (GNU, 1995), ambos originalmente desenvolvidos para o X, houve uma rápida fuga dos *toolkits* proprietários no desenvolvimento de interfaces gráficas no X.

Outros marcos de grande expressão relacionados com o movimento *open source* incluem: o lançamento dos projetos KDE (KDE, 1996) e GNOME (GNOME, 1997), ambientes de trabalho para o Linux (RAYMOND, 2004; REIMER, 2005); a adoção, por parte da Apple, do Unix como base para seu sistema operacional, o MacOS X, lançado em 2001 (RAYMOND, 2004; TUCK, 2001); mais recentemente, as aquisições da Ximian (desenvolvimento de aplicações Linux) e SUSE (desenvolvedora de uma das distribuições mais populares de Linux) pela Novell em 2003.

2.2 OS PARADIGMAS DA INTERAÇÃO HUMANO-COMPUTADOR

Desde a criação do primeiro computador, muito antes dos computadores pessoais e das interfaces gráficas, já havia uma preocupação com a forma de interação homem-máquina. Ao longo da história, houve uma constante e intensa busca por meios cada vez mais eficientes e naturais de se utilizar a máquina. Na medida em que os avanços tecnológicos permitiram a utilização dos computadores

em ambientes mais diversos, para viabilizar aplicações também mais complexas, novas técnicas de interação surgiam e outras eram abandonadas. Olhando a recente história dos computadores, particularmente a partir do surgimento do primeiro computador digital em 1945, é possível identificar três formas amplamente usadas na interação entre homem e computador.

O restante deste capítulo irá apresentar e discutir cada um dos três paradigmas da interação humano-computador identificados. Serão destacadas algumas dimensões de especial interesse, de modo a permitir a comparação entre os paradigmas de interação. Primeiro, o período aproximado durante o qual o modelo de interação foi a forma predominante de utilização da máquina, identificando os eventos que determinam o início e o término do período, assim como sua relevância. Segundo, as tecnologias de *hardware* que viabilizavam o modelo de interação. Terceiro, as pessoas envolvidas na interação humano-computador, os papéis que lhes eram atribuídos e as interações típicas que essas pessoas realizavam com o computador. Quarto, as tecnologias de *software* que viabilizavam o modelo de interação. Por fim, os modelos de desenvolvimento das interfaces e as ferramentas resultantes desses modelos.

2.2.1 Sistemas Batch

2.2.1.1 Período

O paradigma dos sistemas *batch* teve início com o desenvolvimento dos primeiros computadores digitais baseados em transistores, em meados da década de 50, e estendeu-se até a segunda metade da década de 60, com o lançamento dos primeiros sistemas de *time-sharing*, como o CTSS, o DTSS e o IBM System/360. Esse período incluiu duas gerações de computadores: a primeira, dos últimos computadores analógicos e dos primeiros inteiramente digitais (1945-1955), e a segunda, de máquinas baseadas em transistores (1955-1965).

2.2.1.2 Primeira Geração de Computadores

Os computadores da primeira geração eram construídos a partir de circuitos de relés mecânicos, que mais tarde foram substituídos por tubos a vácuo. Os

equipamentos eram enormes, compostos de dezenas de milhares de tubos à vácuo, ocupavam salas inteiras e seu poder de processamento era várias ordens de grandeza inferior ao dos computadores pessoais mais baratos dos dias de hoje. Cada máquina era projetada, construída, programada, operada e mantida por um mesmo grupo de pessoas. A programação se dava em linguagem de máquina, geralmente através da conexão de um *plugboard* ao computador para controlar suas funções básicas. Linguagens de programação e sistemas operacionais simplesmente não existiam (TANENBAUM, 2001, p. 6-7).

A operação desses computadores se dava através da reserva de um horário. O programador entrava na sala no seu horário reservado, conectava seu *plugboard* ao computador e iniciava a operação da máquina esperando que, no decorrer de seu trabalho, nenhum tubo à vácuo queimasse. As tarefas, na sua maioria, limitavam-se a cálculos numéricos como tabelas trigonométricas e logarítmicas. No início dos anos 50 essa rotina seria melhorada de forma significativa com o uso de cartões perfurados (TANENBAUM, 2001, p.7-8).

2.2.1.3 Segunda Geração de Computadores

A invenção do transistor em 1947 permitiu o desenvolvimento de computadores confiáveis o suficiente para que pudessem ser fabricados em escala comercial pela primeira vez. O uso de transistores e circuitos impressos permitiu, além do ganho em confiabilidade, uma redução no tamanho dos computadores da segunda geração. Os primeiros computadores comerciais baseados em transistores datam do final da década de 50 e incluem equipamentos como os IBM 7090, IBM 1620 e IBM 1401, todos de 1959. Os computadores, nessa época, passaram a ser chamados de *mainframes*. Cartões perfurados ou mídia equivalente (como fitas de papel perfurado) eram utilizados como dispositivos de entrada e como meio de armazenamento permanente. Impressoras de linha eram utilizadas como dispositivos de saída (RAYMOND, 2004; TANENBAUM, p. 6-7, 2001).

Computadores da segunda geração ainda eram muito caros e seu uso ficou restrito a ambientes universitários, governamentais e grandes corporações. Os usuários típicos dos computadores dessa época eram cientistas trabalhando em projetos do governo, em pesquisas acadêmicas ou em pesquisa e desenvolvimento para grandes corporações. A operação dos computadores se dava, como na

geração anterior, em seções agendadas antecipadamente. O usuário carregava seus programas repetidamente no computador através de uma leitora de cartões, observava os resultados e decidia o que fazer em seguida. Posteriormente, a operação desses computadores passou a ser realizada por equipes de operadores profissionais.

O uso do computador era precedido da criação, em papel, de um ou mais programas em linguagens de programação como FORTRAN ou *assembler*. Máquinas especiais (*keypunches*) eram então utilizadas para perfurar os programas em um conjunto de cartões. Esse processo era notadamente problemático e sujeito a erros. Perfurados, os cartões eram entregues em uma sala de entrada onde um operador os recebia e eventualmente os alimentava ao computador, onde o trabalho (programa ou conjunto de programas) era então executado, gerando uma impressão com os resultados finais ou, mais freqüentemente, uma notificação de cancelamento do processo acompanhada de um registro de erros. A impressão era levada à sala de saída para que o programador coletasse os resultados de seu trabalho. Essa é uma das primeiras e talvez mais rudimentares formas de operação dos sistemas *batch* (TANENBAUM, p. 7-8, 2001; RAYMOND, 2004).

Na sala de entrada, antes de executar um novo trabalho, o operador aguardava que o trabalho em execução fosse concluído para somente então alimentar os cartões entregues pelo programador. Se preciso, os cartões contendo o compilador FORTRAN eram recuperados de um arquivo físico e carregados. O processo de leitura dos cartões pelo computador era, em si mesmo, lento. Muito tempo era também desperdiçado pelos operadores na realização de tarefas complementares necessárias à execução dos trabalhos. Dados os altos custos dos computadores, a otimização de sua operação a fim de eliminar o tempo desperdiçado era uma questão econômica relevante que precisava ser tratada.

Os sistemas *batch* que seguiram utilizavam idéias avançadas para reduzir o tempo ocioso do computador. Um conjunto de trabalhos era coletado na sala de entrada e carregado em fita magnética através de um computador pequeno e relativamente barato, como o IBM 1401, que era eficiente na leitura de cartões, cópia de fitas e impressão de saídas. A fita era levada à sala do computador principal e montada num *drive* de fita conectado ao computador. O operador carregava então um programa especial chamado monitor, antecessor direto dos sistemas operacionais de hoje, que então lia o primeiro trabalho da fita e o executava,

gerando os resultados em uma segunda fita, a fita de saída, ao invés de uma impressora. Os trabalhos eram executados um após o outro, automaticamente, até o final da fita de entrada. Quando todos os trabalhos da fita de entrada haviam sido processados, ambas as fitas eram removidas, uma nova fita de entrada era carregada e a fita de saída era levada para impressão *off-line* (sem conexão com o computador principal). Esse tipo de sistema ficou conhecido como *load-and-go*, alusão ao fato de que uma vez carregada a fita, o sistema a processava continuamente até seu final. Típicos sistemas monitores da época incluem o FMS (Fortran Monitor System) e o IBSYS, sistema operacional baseado em fita dos IBMs 7090 e 7094 (TANENBAUM, 2001, p. 8).

Para efeito de comparação entre os computadores das duas primeiras gerações, o IBM 650 de 1954, baseado em tubos à vácuo, custava cerca de US$500.000,00, pesava 900kg e sua fonte de alimentação pesava outros 1350kg (ambos eram armazenados em armários de aproximadamente 1,5m x 0,9m x 1,8m). O IBM 1620, baseado em transistores e substituto do IBM 650, era do tamanho de uma mesa de escritório (HISTORY OF COMPUTING HARDWARE, 2005).

2.2.1.4 Interação Humano-Computador

No paradigma dos sistemas *batch*, o uso do computador pelo usuário comum não era previsto. Nos primeiros anos desse período, os computadores eram utilizados pelo grupo de cientistas responsáveis pelo seu desenvolvimento e construção. A interação desses usuários com a máquina se dava, em mais baixo nível, através da interação direta com o *hardware*, e em mais alto nível, através de linguagem de máquina. Esse período pode ser considerado *pré-batch*.

Nos anos seguintes, o advento dos computadores baseados em transistores permitiu a elevação do nível de abstração da interação humano-computador. A programação e a operação dos computadores passaram a ser realizadas por dois grupos distintos: programadores e operadores. Os programadores, normalmente cientistas ou engenheiros, criavam programas em linguagens de baixo nível como *assembler* ou em linguagens de mais alto nível como FORTRAN. Esses programas eram então codificados num formato aceitável pelo computador- cartões perfurados. Impressões eram a forma padrão na qual os resultados desses programas eram apresentados. A interação dos programadores com a máquina, portanto, limitava-se

ao uso de perfuradoras de cartão e à leitura das saídas impressas em papel. Operadores de computador, por outro lado, eram profissionais treinados especialmente para interagir de forma direta com os computadores. Entre suas inúmeras atividades, geralmente ligadas à manipulação dos dispositivos de entrada e saída, destacam-se: a alimentação de cartões perfurados e fitas de papel para o computador principal; a preparação e a montagem de fitas magnéticas para a execução de um bloco de trabalhos (*batch of jobs*); a carga de fitas magnéticas para a impressão de resultados; a carga explícita do software monitor.

Ao custo de vários milhares e até mesmo milhões de dólares por máquina, os esforços de pesquisa nesse tempo concentraram-se no desenvolvimento de tecnologias de *hardware* e *software* visando a maximização de uso e o aumento de *performance* dos computadores. A incorporação de facilidades para aproximar o computador do usuário comum era um aspecto desconsiderado. Assim, na geração dos sistema *batch*, a típica interação humano-computador deu-se através de programas em linguagens como FORTRAN, COBOL e *assembler*, alimentados ao computador através de cartões perfurados e fitas magnéticas. Em suma, a IHC concentrava-se essencialmente no *hardware* e nas linguagens de programação.

2.2.2 Sistemas de Linha de Comando e Sistemas de Tela Cheia

2.2.2.1 Período

A geração dos sistemas de linha de comando teve início com o surgimento dos primeiros sistemas de *time-sharing* e, particularmente, com o lançamento do MULTICS em 1964, e do IBM System/360 em 1965. Estendeu-se até o lançamento do primeiro sistema comercial de interface gráfica de manipulação direta de sucesso, o Apple MacIntosh, em janeiro de 1984. Esse período incluiu duas gerações de computadores: a terceira, de máquinas baseadas em circuitos integrados (1963-1971), e a quarta, de máquinas baseadas em microprocessadores (1971-presente).

2.2.2.2 Terceira Geração de Computadores

A popularização do uso dos computadores teve início a partir da chamada

terceira geração. Computadores dessa geração foram construídos a partir de circuitos integrados (*microchip* ou CI), invenção patenteada por Jack Kilby e Robert Noyce em 1959 (COMPUTER, 2005). Circuitos integrados viabilizaram o desenvolvimento de *mainframes* mais compactos e com maior capacidade de processamento; permitiram também que computadores menores e mais baratos fossem construídos, dando origem ao mercado de minicomputadores e levando o computador, pela primeira vez na história, ao alcance das pequenas empresas.

Dos principais sistemas desse período, destacam-se os computadores da família IBM System/360, lançados em 1965. Esses estão entre os primeiros computadores comerciais a utilizar circuitos integrados e, devido a isso, sua relação custo/desempenho era muito superior aos equipamentos baseados em transistores da segunda geração. Todos os computadores da família possuiam o mesmo conjunto de instruções e a mesma arquitetura e, portanto, *software* escrito para uma máquina poderia rodar em todas as outras máquinas da família. O maior exemplo disso era que todos os sistemas da família eram equipados com uma versão do mesmo sistema operacional, o OS/360: Basic Operating System (BOS/360), Tape Operating System (TOS/360), Card Operating System (COS/360), Disk Operating System (DOS/360) ou OS/360 MVT. A arquitetura do System/360 marcou a divisão entre *hardware* e *software* e a relativização da importância de cada um; sistema pós-360 passaram a ser definidos pelo sistema operacional que executavam (COMPUTER, 2005). Projetados para realizar processamentos tanto científicos quanto comerciais, a diferença básica entre os computadores da família System/360 resumia-se a preço e performance (TANENBAUM, 2001).

O OS/360 foi responsável pela popularização dos sistemas multiprogramados. Nesse tipo de sistema, a memória é segmentada em partições e diversos trabalhos (*jobs*) são carregados em partições distintas de memória. Assim, quando um trabalho inicia um acesso a algum dispositivo de entrada ou saída (E/S), o processador, ao invés de aguardar ociosamente o término desse acesso, passa a ser utilizado para executar um outro trabalho. Outra técnica introduzida nos sistemas dessa geração foi a capacidade de ler novos trabalhos dos cartões para o disco assim que esses estivessem disponíveis. Assim, quando um trabalho encerrava sua execução, o sistema podia carregar um novo trabalho diretamente do disco para a partição de memória livre. Essa técnica ficou conhecida como SPOOLing (Simultaneous Peripheral Operation On Line). A despeito de todos esses avanços,

esses sistemas eram essencialmente versões avançadas de sistemas *batch*.

O desejo para respostas mais rápidas por parte dos computadores levou aos sistemas de *time-sharing*. uma evolução dos sistemas multiprogramados baseados em monitores em que cada usuário tem seu próprio terminal interativo. Desenvolvido em 1961, o CTSS foi pioneiro em muitos aspectos: demonstrou a viabilidade da tecnologia de *time-sharing*; apresentou o primeiro utilitário para formatação de texto; foi o primeiro sistema a utilizar um sistema de correio eletrônico entre usuários. Além de operar em modo *batch*, o CTSS também suportava terminais teletipo locais e remotos, conectados através da rede pública de Telex. Seu fracasso deveu-se, em parte, à ausência de *hardware* de proteção adequado para suportar seu funcionamento. Diferentemente do CTSS, o DTSS é o primeiro sistema de *time-sharing* de larga escala a ser implementado com sucesso. Lançado em 1964, o DTSS utilizava teletipos como terminais e um sistema baseado em comandos como parte do sistema de *time-sharing*. Esse sistema de comandos é considerado o primeiro ambiente integrado de desenvolvimento (IDE) da história. O DTSS foi um sucesso comercial, amplamente utilizado do final da década de 60 ao início da década de 70. Sucessor do CTSS, o MULTICS, lançado em 1965, deu aos sistemas de *time-sharing* a visibilidade necessária para sua ampla adoção, além de introduzir uma série de inovações de grande impacto, como o sistema de arquivos hierárquicos. De relativo sucesso comercial, o MULTICS é hoje considerado o antecessor intelectual dos sistemas operacionais modernos. Partindo das idéias do MULTICS o UNIX começou a ser desenvolvido no final da década de 60 e foi lançado oficialmente no início da década de 70. Ele acompanhava um interpretador de linha de comando e alguns utilitários. Em 1973, o UNIX começou a ser reescrito em C, o que o tornou facilmente portável entre plataformas e o popularizou no meio acadêmico, na comunidade científica, em agências governamentais americanas e em outras corporações (TANENBAUM, 2001, p. 672-674; UNIX, 2005).

Entre os minicomputadores, os DEC PDP e DEC VAX estão entre os mais emblemáticos representantes. O modelo PDP-1 chegou ao mercado 1961; versões subseqüentes do PDP tornaram-se populares nos anos 60 e 70; a partir do final da década de 70, e ao longo da década de 80, os mais populares minicomputadores DEC foram os VAX. Em 1969, a Data General vendeu 50.000 unidades do Nova, um dos primeiros minicomputadores de 16 bits produzidos, e o primeiro a utilizar integração de circuitos em média escala (MSI); modelos posteriores do Nova foram

lançados com integração de circuitos em larga escala (LSI).

2.2.2.3 Quarta Geração de Computadores

O lançamento do primeiro microprocessador comercial pela Intel, em 1971, marca o início da quarta geração de computadores. Diferentemente dos computadores da terceira geração, que eram versões em tamanho reduzido dos *mainframes* da segunda geração, os computadores da quarta geração eram fundamentalmente diferentes. Computadores dessa geração utilizavam um microprocessador para concentrar a maior parte de sua capacidade de processamento num único *chip*. Combinado com o *chip* de RAM, inventado na IBM, os computadores dessa geração tornaram-se mais velozes e menores que antes.

Ao mesmo tempo em que a tecnologia do microprocessador difundia-se, outras mudanças ocorriam. As tecnologias de vídeo tornavam-se suficientemente maduras e economicamente viáveis de modo a permitir o uso do vídeo como um dispositivo de interação com o computador. Terminais de vídeo baseado em caractere foram os primeiros a surgir; mais tarde, apareceram monitores de vídeo com suporte a gráficos de varredura. Um resultado imediato da viabilização da tecnologia de vídeo foi a substituição gradual dos teletipos por terminais de vídeo, que acabaram tornando-se a forma padrão de interação com computadores a partir da segunda metade da década de 70. Outro resultado relevante foi a incorporação dos monitores de vídeo como partes dos computadores pessoais e microcomputadores. Assim como a evolução das tecnologias de vídeo, os desenvolvimentos em torno das tecnologias de rede nesse período tiveram grande impacto na forma de interação humano-computador. Primeiramente, o início da ARPANET militar, em 1969, e sua abertura logo em seguida, em 1970; depois, o desenvolvimento da pilha de protocolos TCP/IP, em 1972, e a criação do protocolo de redes locais Ethernet, no Xerox PARC, em 1973. Esses desenvolvimentos permitiram representar o computador não apenas como um máquina para processamentos numéricos e de informações, mas também como um sistema de comunicação e colaboração.

A introdução do microprocessador em 1971 permitiu o desenvolvimento de computadores de todos os tipos- de supercomputadores a microcomputadores e computadores pessoais. O MITS Altair, de 1975, foi o primeiro computador pessoal

produzido em massa e também o primeiro a utilizar o processador 8080 da Intel. O Intel 8080 também deu origem a uma série de microcomputadores voltados para pequenas empresas que, na maior parte, rodavam o sistema operacional CP/M-80 da Digital Research. A década de 70 viu o lançamento de diversos computadores pessoais como os Apple I e II, o TRS-80 e o Commodore PET. A maioria desses computadores vinha equipada com teclado, *drive* de disco flexível, permitia o armazenamento externo através de fitas cassete e suportava alguma forma de interface de vídeo através de monitores embutidos ou externos (geralmente monocromáticos), televisores especialmente modificados ou televisores comuns.

A era dos computadores pessoais ganharia momento a partir da década de 80 com a entrada da IBM no mercado que a Apple havia provado rentável com seu Apple II. O IBM PC foi lançado em 1981 e, nos anos seguintes, foi sucedido pelos IBM XT (1983) e IBM PC/AT (1984). A perda do controle do mercado de computadores IBM PC compatíveis da IBM ficou clara entre 1986-1987, com o lançamento do Compaq Deskpro 386, baseado no processador 80386 da Intel. Os anos seguintes marcaram o contínuo desenvolvimento do *hardware* dos computadores pessoais através de tecnologias mais eficientes, processadores cada vez mais rápidos, monitores maiores e capazes de exibir mais pontos e mais cores, e preços cada vez mais baixos.

2.2.2.4 Interação Humano-Computador

O paradigma dos sistemas de linha de comando pode ser separado em duas etapas distintas, de acordo com o principal tipo de dispositivo utilizado para interagir com o computador: de 1965 à primeira metade da década de 70, quando predominava o uso do teletipo e, da segunda metade da década de 70 a 1983, quando predominava o uso do terminal de vídeo.

A interação através da linha de comando foi uma profunda mudança quando comparada aos sistemas *batch* do paradigma de IHC anterior. Antes, os programas e dados necessários para realizar uma computação eram previamente arranjados de modo que estivessem disponíveis antes do sistema iniciar sua execução. Simplesmente, não havia interação direta do usuário com o sistema. Com o advento das interfaces de linha de comando, os usuários passaram a interagir com os computadores através do envio de comandos textuais para a máquina. Um

programa especial, chamado interpretador de linha de comando, era utilizado para processar e executar os comandos enviados. Os resultados do processamento eram então enviados de volta ao usuário que podia analisá-lo de imediato. Nesse modelo de interação, portanto, o usuário estabelece a comunicação com o sistema através de uma série de transações do tipo solicitação-resposta, onde as solicitações eram expressas de forma textual através de um vocabulário devidamente estabelecido, num modelo análogo ao do diálogo entre pessoas. No caso das interfaces de linha de comando, esse diálogo era sequencial, já que os usuários percorriam as diversas partes do diálogo numa seqüência previsível, e síncrono, onde uma tarefa por vez era apresentada ao usuário e sua conclusão era necessária antes do prosseguimento para a tarefa seguinte (HARTSON; HIX, 1989). A interatividade dos sistemas era alta, já que passou a ser possível mudar de idéia sobre como proceder durante estágios avançados de uma operação, a partir dos resultados obtidos em etapas anteriores dessa operação. A latência dos sistemas caiu consideravelmente, passando de dias ou horas, como era comum nos sistemas *batch*, para alguns segundos. O *software* adquiriu um caráter exploratório e de interatividade dificilmente imaginável nos sistemas *batch*. Ao invés de processadores numéricos, os computadores passaram a exercer o papel de processadores de informação.

No entanto, a interatividade em si não foi suficiente para popularizar o uso dos sistemas de linha de comando de imediato. A familiaridade com os sistemas *batch* e necessidade de uso do teletipo como meio de interação com os sistemas de linha de comando era motivo de resistência e desconfiança por parte de muitos grupos de usuários. Somente com o advento do terminal de vídeo é que os sistemas interativos de linha de comando começaram a vencer a resistência dos usuários mais conservadores e popularizaram-se. Terminais de vídeo eram mais interativos que os teletipos devido às baixas latências de seu *hardware*: gerar caracteres pela num monitor de vídeo era tarefa ordens de grandeza mais rápida do que imprimir esse mesmo caractere em um teletipo.

A interatividade no lado do *software* também teve seus problemas. O principal delas dizia respeito ao vocabulário. Para interagir com as interfaces de linhas de comando, era preciso memorizar extensos conjuntos de comandos e sua sintaxe. O aspecto que mais dificultava essa memorização era a forma de codificação desses comandos através de mneumônicos. Além disso, muitas vezes os comandos eram implementados por grupos de programadores distintos e acabavam empregando

sintaxes incompatíveis, dificultando ainda mais o aprendizado por parte do usuário.

Com a evolução dos terminais de vídeo, passou a ser possível trabalhar com o computador também em tela cheia. Ao contrário da interface de linha de comando, a interface de tela cheia, também conhecida como modo texto ou interface textual do usuário (TUI), utilizava toda a área da tela mas não oferecia, necessariamente, respostas linha-a-linha. TUIs exibiam suas interfaces utilizando caracteres e símbolos disponíveis nos terminais. Entre os aplicativos que empregaram essas técnicas incluem-se jogos, editores de texto e sistemas baseados em menus e formulários. Diversos sistemas em uso, como o cliente de e-mail *pine* (Figura 23) e o gerenciador de pacotes YaST da SUSE (Figura 24), ilustram as características gerais das interfaces TUI. A despeito de sua relação de proximidade e dependência com os ambientes de linha de comando, justifica-se classificar as interfaces de tela cheia como uma forma de interação à parte, dada sua forma particular de interação com o usuário.

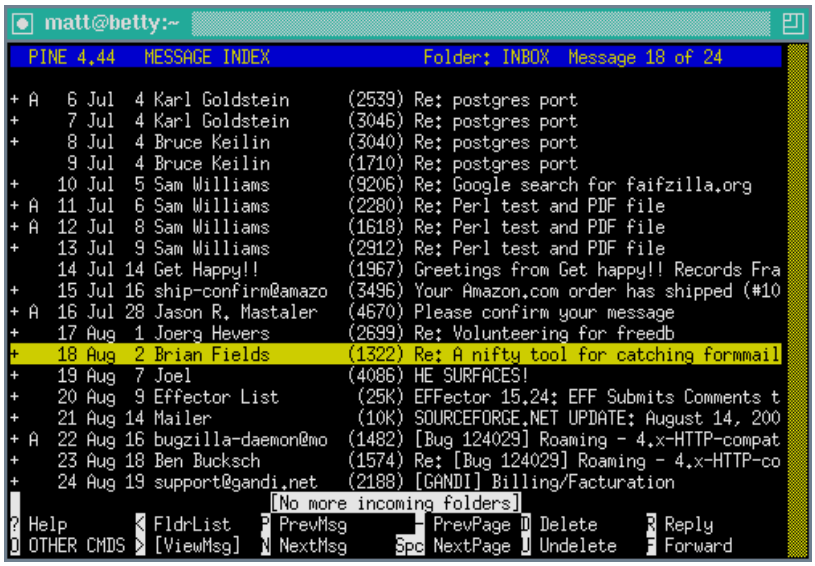


Figura 23 – Pine, rodando em um terminal emulado
Fonte: <http://osx.freshmeat.net/projects/pine/>).

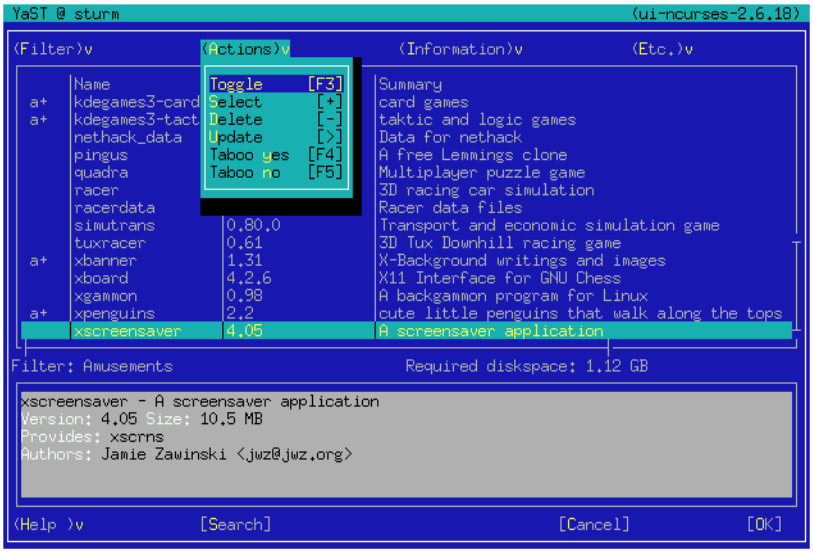


Figura 24 – Gerenciamento de pacote no SUSE Linux com o Yast
Fonte: <http:/www.suse.de/~sh/YaST2-Package-Manager/>.

Ao longo de todo o período dos sistemas de linha de comando, os usuários do computador passaram do seleto grupo de cientistas, engenheiros, agências governamentais e grupos seletos dentro de grandes e médias corporações utilizando *mainframes* e minicomputadores através de teletipos, à pequena empresa utilizando microcomputadores e ao usuário comum utilizando computadores pessoais. A interação humano-computador passou do modelo *batch*, não interativo, baseado em programação e cartões perfurados, para um modelo interativo baseado em uma linguagem de comandos e terminais de vídeo. Nesse novo paradigma, a figura do programador ainda era central, mas agora, mais bem equipado e produtivo, e com acesso a linguagens de programação mais diversas e de mais alto nível, como LISP, Algol, Pascal, C e BASIC. O programador também passou a trabalhar num ambiente de maior colaboração, comunicando-se com outras pessoas e compartilhando recursos através da rede local e da ARPANET. Do outro lado, a figura do usuário comum entrava em cena pela primeira vez. Seu papel ainda era bastante limitado e sua interação com o computador deu-se, basicamente, através dos primeiros computadores pessoais e seus sistemas de linha de comando e aplicativos TUI. O perfil desse usuário ainda era bem diferente do usuário comum dos sistemas de hoje e o uso que faziam da máquina separava-se em uso pessoal e empresarial. Para usuários pessoais, os computadores eram uma plataforma de lazer, para programação não profissional e jogos; para usuários empresariais, eram ferramentas

de edição de texto, planilha eletrônica e processamento de dados (COMPUTER, 2005). Exemplos de *software* disponível à época para tais fins incluem: WordStar (PETRIE, 1999; WORDSTAR, 2005), VisiCalc (BRICKLIN, 1999; COMPUTER, 2005) e dBase (DATABASE INTELLIGENCE, 2005; DBASE, 2005).

2.2.3 Sistemas de Manipulação Direta

2.2.3.1 Período

A geração dos sistemas de manipulação direta teve início com o lançamento do primeiro sistema comercial de interface gráfica de sucesso, o Apple MacIntosh, em janeiro de 1984 e prolonga-se aos dias atuais. Esse período inclui computadores de quarta geração baseados em microprocessadores, com integração de circuitos em larga escala (LSI), muito larga escala (VLSI) e além. A partir dos anos 80, diversas plataformas de hardware passaram a coexistir, desde mainframes, minicomputadores e supercomputadores, a estações de trabalho e computadores pessoais baseados no IBM PC. Sistemas de manipulação direta tornaram-se a forma predominante de interação com as estações de trabalho dos anos 80 e 90 e são, ainda hoje, a principal forma de interação com os computadores pessoais

As origens dos sistemas de manipulação direta podem ser traçadas às idéias de Vannevar Bush sobre um sistema hipotético chamado Memex, no início dos anos 30. O Memex foi, conceitualmente, o primeiro sistema gráfico a incluir uma facilidade de hipertexto para a navegação entre as informações. Somente dezessete anos após a publicação do artigo de Bush, *As We May Think*, em 1945, voltaram a surgir avanços significativos em direção às interfaces de manipulação direta.

Em 1962, Ivan Sutherland apresentou, como parte de seu trabalho de doutorado, o Sketchpad, considerado a primeira implementação de um sistema gráfico de manipulação direta. O Sketchpad operava numa tela de vídeo vetorial com o auxílio de uma caneta ótica e um painel de controle composto por diversos botões. Objetos eram desenhados e manipulados diretamente na tela, através do manuseio da caneta ótica e com o auxílio do painel de controle. Comandos específicos eram enviados ao sistema combinando a operação da caneta com o pressionamento dos botões do painel: “draw”, para a construção de um segmento de

linha; “lock on” para fechar uma figura aberta, “move”, para mover um elemento na tela; “copy”, para copiar e colar um objeto; “circle center” seguido de “draw”, para desenhar um círculo, etc. (SUTHERLAND, 1963). O Sketchpad considerava cada desenho a partir de sua definição topológica; além disso, permitia o arquivamento e a recuperação dos desenhos em arquivos, viabilizando sua reutilização. Suportava a composição de desenhos a partir de outros desenhos (partes), estabelecendo uma relação de dependência entre essas partes e garantindo que a alteração de partes dependentes refletisse automaticamente no desenho principal.

Nessa época, o cientista Douglas Engelbart começava a trabalhar num projeto de pesquisa para a ampliação do intelecto humano. Os resultados desse projeto foram apresentados em 1968, numa conferência em São Francisco, naquela que viria a ser conhecida como “a mãe de todas as demonstrações”. Nessa apresentação, Engelbart apresentou o NLS, oNLine System, um sistema equipado com teclado, *mouse,* um painel de controle e um conjunto de *softwares* revolucionário à época. Além da utilização do *mouse* como um dispositivo de interação substituto das canetas óticas, o NLS introduziu uma série de inovações: um sistema gráfico baseado em janelas (lado-a-lado, sem barras de títulos ou bordas), um sistema hipermídia que permitia o endereçamento e a ligação entre objetos, uma versão primitiva de conferência em vídeo, colaboração e comunicação em rede, dentre outros. A demonstração do NLS abriu definitivamente o caminho para o desenvolvimento das interfaces gráficas modernas.

Motivada pelas possibilidades do modelo gráfico de interação, a Xerox criou um moderno laboratório de pesquisas em 1970 e iniciou o desenvolvimento de diversas tecnologias que culminariam, em 1973, com o desenvolvimento do primeiro sistema gráfico moderno de manipulação direta, o Xerox Alto. O sistema contava com um teclado, um *mouse* de três botões, uma conexão Ethernet e um monitor monocromático baseado na tecnologia de gráficos de varredura. Em termos de *software*, o Alto introduziu a operação BitBlt, para a programação das interfaces gráficas; o Smalltalk, como ambiente gráfico de desenvolvimento e execução da primeira linguagem orientada a objetos; Bravo, o primeiro editor de texto WYSIWYG, dentre outros. A interface do sistema era baseada no conceito de manipulação direta dos objetos gráficos na tela. Utilizava a idéia de uma área de trabalho, metáfora da mesa de trabalho do escritório, e era composta de janelas sobrepostas, ícones, barras de rolagem, botões, caixas de diálogo e menus (curiosamente, o menu *pull-*

down foi o único elemento da interface gráfica ausente, mais tarde introduzido pelo Apple Lisa em 1979-1983). A maioria dos sistemas gráficos modernos é baseada na tradição da área de trabalho e das interfaces WIMP introduzidas originalmente em 1973 pelo Alto.

Os sistemas gráficos após o Xerox Alto inovaram relativamente pouco do ponto de vista da interação com a interface gráfica: o paradigma da manipulação direta permaneceu o elemento central dessas interfaces. No lado da arquitetura e engenharia de *software* utilizados na construção dessas interfaces, entretanto, a evolução foi acentuada. Uma discussão detalhada sobre essas ferramentas será apresentada no Capítulo 3.

2.2.3.3 Interação Humano-Computador

A interação visual com objetos na tela, chamada de manipulação direta, eleva em muito o grau de abstração da interação humano-computador, quando comparada à interação através da linha de comando. Sistemas baseados em transações do tipo solicitação-resposta através de comandos textuais mneumônicos foram substituídos por sistemas ainda mais interativos, onde o *mouse* era utilizado para mover um ponteiro sobre objetos gráficos na tela e executar operações sobre esses objetos. No modelo de manipulação direta, o usuário espera que o sistema ofereça representações de objetos que se comportem como se fossem os objetos propriamente ditos. O sistema, em geral, permite um amplo leque de alternativas de interação- modelo no qual o modo de operação é, basicamente, livre, dependendo apenas das escolhas do usuário. Em relação aos paradigmas de IHC anteriores, a manipulação direta oferece vantagens marcantes: rápido aprendizado da operação do sistema; facilidade de retenção de conceitos operacionais; maior controle sobre as atividades, já que a representação imediata dos efeitos da manipulação sobre os objetos permite a (quase) imediata correção do curso de ação. Entre algumas de suas desvantagens, destacam-se: a dificuldade de programação visual- ainda hoje, à exceção de interfaces baseadas em demonstração e macros, não se tem formas eficientes de especificar diversas das estruturas lógicas de programação (especialmente laços de repetição) através da manipulação direta das interfaces; e as imposições restritivas do ambiente gráfico sobre usuários avançados que, de

outra forma, realizariam certas tarefas de forma muito mais rápida através da linha de comando, como apagar um conjunto de arquivos de um mesmo tipo ou com a mesma extensão – ao invés de selecionar os arquivos um a um através da interface gráfica, o usuário pode simplesmente emitir um comando como “rm *.jpg”.

Na era das interfaces de manipulação direta, os computadores são utilizados por usuários dos mais variados graus de conhecimento, atuando nas mais diferentes áreas e advindos das mais diversas culturas. Dessa forma, impõe-se sobre as interfaces gráficas uma série de requisitos difíceis de alcançar: espera-se que um mesmo sistema seja igualmente fácil de aprender e simples de usar por usuários iniciantes, intermediários e avançados; supõe-se que esses sistemas sejam igualmente eficientes em ambientes tão diversos como empresas da área de saúde e indústria química, por exemplo; por fim, os sistemas devem funcionar levando em conta as diversidades culturais, sociais, econômicas e linguísticas – aspectos tão simples como a internacionalização dos sistemas pode, isoladamente, ser complexa o suficiente para impedir um sistema de funcionar simultaneamente num país de língua portuguesa e em outro de língua hebraica, por exemplo.

Graças à universalização do computador pessoal (a partir do final da década de 80) e das interfaces gráficas de manipulação direta (a partir do início da década de 80), os sistemas de hoje lidam com aspectos e ordens de grandeza mais complexos do que aqueles enfrentados por sistemas dos paradigmas anteriores. Os mais importantes desses aspectos serão tratados no capítulo seguinte, juntamente com a discussão sobre as ferramentas de desenvolvimento de interfaces gráficas.

2.2.4 Análise Comparativa dos Paradigmas de Interação

Os paradigmas de IHC discutidos nas seções anteriores podem ser analisados através de uma série de características observadas anteriormente. Para isso, várias dimensões foram selecionadas com essa finalidade, sendo descritas a seguir.

O computador, a entrada de dados e a saída de dados identificam o *hardware* típico de cada paradigma. O modelo de interação descreve, em termos abstratos, a forma da interação humano-computador. A atuação do usuário junto ao sistema se

dá através de um meio de expressão específico, ou interface; para viabilizar ou facilitar a atuação do usuário, o sistema pode oferecer alguma forma de apoio ao modelo de interação; o tempo de interação é o tempo médio que o usuário leva para realizar uma interação completa. O usuário típico do sistema refere-se ao usuário ou grupo para o qual o sistema foi projetado; assim, o papel do usuário comum no sistema precisa ser identificado à parte. Os sistemas apresentam graus de dificuldade variáveis e, por isso, é preciso analisar o tempo de aprendizado , isto é, quanto tempo leva, em termos relativos, para um usuário novato começar a usar o sistema. Além disso, é preciso saber qual o grau de retenção do aprendizado, ou seja, quanto do aprendizado permanece após uma interrupção no uso do sistema.

A maioria dessas dimensões foi discutida anteriormente neste capítulo. Algumas puderam ser inferidas a partir das características de cada paradigma e das relações entre homem e computador que se estabeleceram em cada período-aspectos também analisados neste capítulo. As dimensões restantes, relacionadas ao uso do sistema, foram analisadas nos seus respectivos contextos, considerando os conceitos de distância e engajamento como medidas de naturalidade. Segundo Hutchins, Hollan e Norman (1985), a naturalidade de um sistema não é uma propriedade isolada de sua interface, mas envolve a relação das tarefas que o usuário executa com a forma que ele realiza essas tarefas através da interface. Quanto mais um sistema aproxima os objetivos e conhecimentos do usuário a seu modelo descritivo, tanto mais ele reduz o esforço cognitivo desse usuário na operação da interface. Essa aproximação pode ser promovida de duas formas: através de comandos e mecanismos que representem bem o modelo mental e os objetivos do usuário; pela apresentação de um modelo conceitual do sistema que possa ser rapidamente percebido, interpretado e avaliado pelo usuário (HUTCHINS; HOLLAN; NORMAN, 1985). O outro aspecto considerado refere-se à qualidade do engajamento da interface. O objetivo é permitir ao usuário interagir com as representações na interface como se essas fossem os objetos reais propriamente ditos. Modelos baseados em conversação, por exemplo, onde o usuário interage com estruturas lingüísticas para referir-se aos objetos de interesse, são menos naturais que aquelas em que, ao invés de descrever as ações, os usuários as realizam (HUTCHINS; HOLLAN; NORMAN, 1985).

A evolução dos paradigmas da IHC destaca duas tendências intimamente relacionadas: o deslocamento do foco central dos paradigmas de interação para o

homem e a busca pela maior naturalidade das interfaces. Na medida em que o homem (usuário final) torna-se o foco do desenvolvimento das interfaces, essas devem tornar-se cada vez mais naturais e fáceis de usar; as interações devem ocorrer em tempo real parar garantir a percepção de naturalidade da interface por parte do usuário; além disso, a interação com a interface deve ocorrer através de meios ótimos – como na manipulação direta de interfaces gráficas através do *mouse*, que se movimenta ao longo de duas dimensões em analogia ao ponteiro associado na interface, que se desloca sobre um ambiente gráfico em duas dimensões. Em termos de prognósticos, algumas tendências para o futuro começam a tomar forma. Raj Reddy, da Carnegie Mellon, por exemplo, descreveu as interfaces SILK (*speech*, *image*, *language understanding*, *knowledge bases*), definidas pela integração do reconhecimento de voz, imagens e compreensão de linguagem através de bases de conhecimento (REDDY, 1996, apud VAN DAM, 1997). Pesquisas desse tipo reforçam que o foco dos sistemas do futuro estará fundamentalmente ligado ao homem através de interfaces que lhe são cada vez mais naturais.

Tabela 1– Quadro Comparativo dos Paradigmas de Interação

Dimensão	Paradigma de Interação Humano-Computador				
	pré-batch	**batch**	**linha de comando**	**TUI**	**manipulação direta**
Computa-dor	mainframe	mainframe	mainframe, minicomp.	minicomp., microcomp.	estação de trabalho, PC
entrada de dados	reconexão de cabos	cartões perfurados	teletipo	terminal de vídeo	mouse, teclado
saída de dados	impressora de linha	cartões perfurados, impressora de linha	teletipo	terminal de vídeo	monitor
modelo de interação	interação com hardware	interação programática	diálogo seqüencial	interação em tela cheia	diálogo assíncrono
atuação	programação	programação	emissão de comando	seleção de menu	manipulação direta
interface	plugboard	trabalho (job)	linha de comando	menus, formulários	objetos gráficos

Dimensão	Paradigma de Interação Humano-Computador				
apoio ao modelo de interação	hardware	compilador	interpretador de comandos	seqüências de escape, teclas de função	gerenciador de janelas
tempo de interação	horas ou dias	horas ou dias	segundos	fração de segundos	fração de segundos
usuário típico	cientista	cientista, engenheiro	técnico, usuário comum	técnico, usuário comum	usuário comum
papel do usuário comum	nenhum	nenhum	secundário	secundário	Primário
tempo de aprendiza-do	muito longo	longo	longo	médio	Curto
retenção do aprendiza-do	baixa	média	média	média	Alta

Este capítulo destacou relevantes influências históricas na evolução dos paradigmas da IHC. Identificou e caracterizou cada um desses paradigmas, discutindo aspectos históricos e os modelos de interação de cada um deles. Por fim, um quadro comparativo foi apresentado, ilustrando as semelhanças e diferenças entre os paradigmas ao longo de uma série de dimensões. O próximo capítulo estenderá a discussão sobre o paradigma de manipulação direta através da análise dos principais tipos de ferramentas utilizadas no desenvolvimento das interfaces gráficas dos aplicativos desse paradigma.

3 INTERFACES DE MANIPULAÇÃO DIRETA

Na medida em que as interfaces tornam-se mais fáceis de usar, ficam também mais difíceis de implementar. Como afirmou Myers (1995), o *software* destinado à criação das interfaces gráficas de manipulação direta é geralmente extenso, complexo, difícil de implementar, depurar e modificar. Myers e Rosson (1992) sugerem ainda que cerca de 48% do código de um aplicativo está diretamente ligado à interface com o usuário e que aproximadamente 50% do tempo de implementação está relacionado a essa interface. Nessas circunstâncias, a busca por técnicas e ferramentas de *software* que auxiliem no desenvolvimento das interfaces gráficas é, não apenas uma tarefa desejável, mas, acima de tudo, de vital importância na economia de tempo e recursos das equipes de desenvolvimento.

Este capítulo apresenta e discute as principais arquiteturas, ferramentas e técnicas de implementação de interfaces gráficas de manipulação direta. Apresenta o modelo de programação empregado por cada ferramenta; as vantagens e desvantagens de seu uso são analisadas do ponto de vista do programador; por fim, as abordagens da ferramenta são comparadas a abordagens de outras ferramentas de modo a ilustrar aspectos positivos e negativos de cada alternativa.

3.1 MODELAGEM DE INTERFACES HUMANO-COMPUTADOR

Antes de discutir questões relativas ao desenvolvimento de interfaces humano-computador, é importante analisar os diferentes modelos lógicos em que essas interfaces são estruturadas: conceitual, funcional (ou semântico), sintático e léxico. No nível conceitual, são estabelecidas as definições relacionadas ao domínio do aplicativo que devem ser compreendidas pelo usuário- objetos, propriedades, relacionamentos, etc. Os detalhes da funcionalidade, como informações necessárias e resultados de cada operação, são especificados através do modelo funcional. O modelo sintático determina as regras através das quais unidades de significado são formadas e agrupadas; por exemplo, os movimentos e cliques do *mouse* necessários para realizar a seleção de um item de menu definem uma unidade

sintática, pois, isoladamente, não oferecem significado ao aplicativo. Por fim, o modelo léxico determina como as unidades de informação são geradas a partir das primitivas dos dispositivos de interação (o *hardware*).

Definidos os elementos do modelo lógico, a interação do usuário com o sistema pode ser explorada na sua forma, ou seja, nos seus modelos sintático e léxico. Primeiramente, são definidos os estilos de interação da aplicação: comandos textuais (linguagens de comando), sistemas de menus, formulários, linguagem natural, manipulação direta, realidade virtual, gesticulação, etc. Observando a forma de cada estilo, o usuário realiza suas tarefas no sistema- entrada de unidades de informação como textos, seleção de valores, cliques de botão, etc. Para uma mesma tarefa, várias técnicas de interação podem ser possíveis, como, por exemplo, o clique de um botão do *mouse* ou o pressionamento de uma tecla de atalho. Cada técnica está associada a um ou mais dispositivos de interação- o *hardware* utilizado para a entrada de dados. Portanto, os estilos ditam a forma de interação, as tarefas estabelecem o que o usuário realiza e as técnicas e dispositivos definem como o usuário realiza a tarefa: um sistema hipotético disponibiliza interfaces de manipulação direta e linha de comandos (os estilos); para excluir um objeto do sistema (a tarefa), um usuário pode arrastar o objeto para a lixeira (a técnica) com o *mouse* (o dispositivo) ou digitar o comando apropriado (a técnica) no teclado (o dispositivo).

3.2 ARQUITETURA DE SOFTWARE GRÁFICO BASEADO EM GUIS

Os sistemas de manipulação direta geralmente empregam uma arquitetura de *software* baseada na seguinte pilha de componentes (MYERS, 1995):

Figura 25 – Componentes da Arquitetura de Software

Tipicamente, um aplicativo utiliza uma ou mais ferramentas de alto nível para auxiliá-lo no desenvolvimento de sua interface visual. Tais ferramentas incluem UIMSs, construtores de interfaces, *frameworks* orientados a objetos, padrões de interfaces, etc. Em geral, as ferramentas de alto nível recorrem aos serviços de mais baixo nível de um *toolkit*, na forma de uma biblioteca procedural (API) ou *framework*, para implementar as interfaces. Os *toolkits*, por sua vez, são implementados a partir de um outro *toolkit* de mais baixo nível, ou diretamente, através de chamadas ao sistema de janelas. Esse, por sua vez, fornece apenas as primitivas gráficas e de comunicação a partir das quais os *toolkits* implementam seus *widgets*. Os sistemas de janela funcionam integrados ao sistema operacional ou como uma camada independente, acima dele. Exceto pelo sistema operacional, as demais camadas são discutidas nas seções que seguem.

3.3 SISTEMAS DE JANELA

Sistema de janela é uma camada de software que auxilia no controle e gerenciamento de diferentes contextos através de sua separação física em partes distintas de uma ou mais telas (MYERS, 1988). Os sistemas de janela correspondem à camada mais baixa do software da interface gráfica. Normalmente, o programador não interage diretamente com o sistema de janelas. Quando precisa desenhar e

manipular diretamente partes específicas de seu aplicativo, no entanto, ele deverá utilizar as APIs do sistema de janelas.

Os sistemas de janela são separados logicamente em duas camadas- o sistema de janelas propriamente dito (ou camada básica) e o gerenciador de janelas. A camada básica gerencia a exibição de gráficos em janelas (modelo de saída) e o acesso aos vários dispositivos de entrada como teclado e *mouse* (modelo de entrada). O gerenciador de janelas é responsável pela exibição da interface (modelo de apresentação) e pela manipulação das janelas e seu conteúdo (modelo de comandos).

O modelo de saída é simples: consiste de uma série de rotinas que o aplicativo chama para desenhar a saída na tela. O modelo de entrada, por outro lado, é relativamente complexo. Funciona através de um fluxo assíncrono de eventos constantemente enviados à janela que possui o foco, isto é, aceita entrada de dados. O sistema de janelas coloca as mensagens numa fila de eventos associados à janela e os aplicativos, então, retiram as mensagens da fila e as processam.

Um dos aspectos mais importantes da apresentação refere-se à forma na qual as janelas são arranjadas- lado a lado ou sobrepostas umas às outras; outros aspectos que definem a forma de interação com o sistema de janelas, incluem: a utilização de ícones para representar janelas; a utilização de barras de título nas janelas; a seleção de pano de fundo para a área de trabalho; o acesso a áreas de controle em regiões das janelas como bordas e barra de título. O modelo de comandos, por fim, determina a forma de controle do ambiente de janelas- por exemplo, como o usuário alterna entre janelas e define a janela que recebe a entrada do teclado (chamada de janela ativa ou janela com o foco). Outros comandos incluem: trazer janelas para a frente e enviar janelas para trás, redimensionar, mover, fechar, reduzir a ícone, restaurar e maximizar.

Sistemas de janela precisam de um canal de comunicação com os aplicativos. A forma mais simples de realizar essa comunicação é através de algum mecanismo de comunicação interprocessos disponível no sistema operacional. Alguns sistemas, no entanto, comunicam-se com os aplicativos através de um protocolo de rede, permitindo que tais aplicativos estejam transparentemente distribuídos pela rede. O X Windows é um exemplo de sistema de janelas que utiliza um protocolo de rede para sua comunicação com os aplicativos (MYERS, 2004).

O sistema de janelas da maioria dos sistemas baseados em UNIX e Linux é o X Windows. A partir do MacOS X, os MacIntoshes também utilizam o X Windows como seu sistema de janelas nativo. O sistema de janelas do Microsoft Windows é parte de seu sistema operacional e o acesso a seus serviços é obtido através do uso da Windows API.

Resumo das Características dos Sistemas de Janela	
Utiliza Serviços das Camadas	Sistema operacional, através de APIs
Serviços Oferecidos	Serviços gráficos de baixo nível, controle de entradas e saídas
Serviços Oferecidos às Camadas	Toolkits
Serviços Oferecidos Através de	APIs procedurais
Profissionais que Utilizam	Programadores apenas
Papel na Análise	Nenhum

3.4 TOOLKITS

A forma mais rudimentar de programar interfaces gráficas é através do emprego de *widget toolkits*, GUI *toolkits* ou, simplesmente, *toolkits*. *Toolkits* são coleções de *widgets*, os elementos fundamentais da interface gráfica que implementam os mecanismos de interação humano-computador. São exemplos de *widgets*: botões, barras de rolagem, menus, caixas de texto, caixas de seleção e janelas. *Toolkits* são geralmente implementados no próprio sistema operacional ou em alguma camada de *software* próxima a ele. São disponibilizados ao programador sob a forma de bibliotecas procedurais de código (APIs) ou *frameworks* orientados a objeto. *Toolkits* orientados a objeto são mais sofisticados e apresentam diversas vantagens em relação aos *toolkits* procedurais: o modelo conceitual de *widgets* como objetos é mais natural (menus e botões na tela parecem objetos); objetos podem realizar as tarefas mecânicas necessárias ao uso dos *toolkits*, como a atualização de sua exibição na tela e o registro automático de *callbacks*; finalmente,

criar um *widget* especializado fica facilitado através do uso de herança. Em contrapartida, *toolkits* procedurais são mais fáceis de implementar, não precisam de sistemas orientados a objetos e são mais portáteis entre linguagens de programação.

Grande parte dos *toolkits* é implementada na linguagem C, sob a forma de uma ou mais bibliotecas dinâmicas. Como resultado, a interface programática se dá através do conjunto de arquivos de cabeçalho do *toolkit* ou através dos arquivos traduzidos equivalentes na linguagem de programação utilizada. Entre os *toolkits* de baixo nível encontram-se aqueles integrados ao sistema operacional, como o MAC OS Toolbox e a Windows API, e os que operam uma camada acima do sistema operacional, como o xlib do X Windows. *Toolkits* de mais alto nível ocorrem sob duas formas: *toolkits* nativos e *toolkits* virtuais (ou multi-plataforma). *Toolkits* nativos são implementados como bibliotecas ou *frameworks* nativos a uma plataforma escolhida e rodam exclusivamente nesse ambiente. Exemplos de *toolkits* nativos incluem: Cocoa (APPLE, [ca. 1996]), no Macintosh; Microsoft Foundation Classes (Microsoft, 1992) e Visual Component Library (BORLAND, 1995), da Borland, no Windows; Motif (OPEN GROUP, [ca. 1989]) e Lesstif (LESSTIF, 1992), no Unix com X Windows. *Toolkits* virtuais empregam duas abordagens distintas em suas implementações. Em uma delas, o *toolkit* virtual se liga a um *toolkit* nativo disponível no sistema. Na outra, o *toolkit* reimplementa os *widgets* em estilos preestabelecidos. Na primeira abordagem, as interfaces apresentam um aspecto visual nativo; no entanto, apenas serviços comuns a todos os *toolkits* suportados tendem a ser disponibilizados pelo *toolkit* virtual. Na segunda abordagem, as interfaces são visualmente consistentes ao longo de todas as plataformas suportadas, mas requerem uma biblioteca em tempo de execução. MFC (MICROSOFT, 1992), Java AWT (SUN, 1995), VCL (BORLAND, 1995) e Eclipse SWT (ECLIPSE, [ca. 2004]) são exemplos de *toolkits* virtuais da primeira abordagem. Já wxWidgets (WXWIDGETS, 1992), Qt (TROLLTECH, 1994), GTK (GNU, 1995) e Java SWING (SUN, [ca. 1998]) são exemplos de *toolkits* virtuais do segundo tipo. Tschater (2004) comparou a programação em vários *toolkits* modernos utilizando uma interface de referência (figuras 26-31).

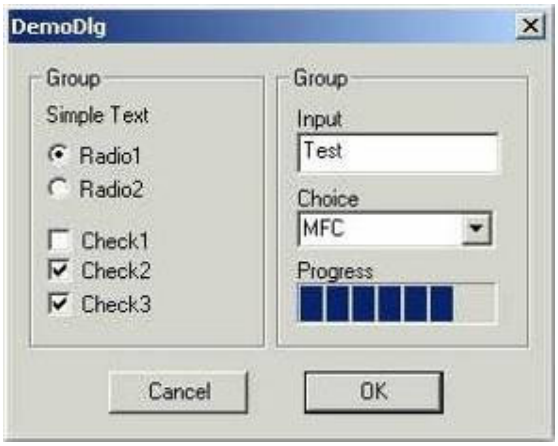


Figura 26 – MFC Toolkit
Fonte: Tschater (2004).

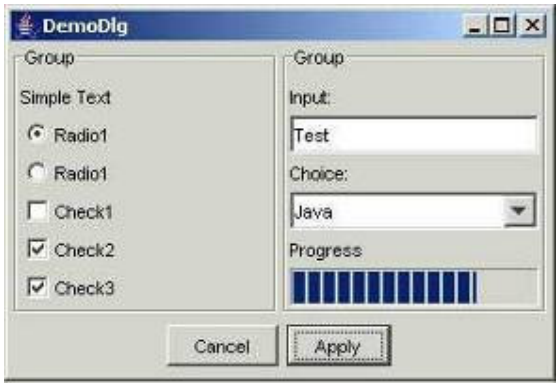


Figura 27 – Java AWT Toolkit
Fonte: Tschater (2004).

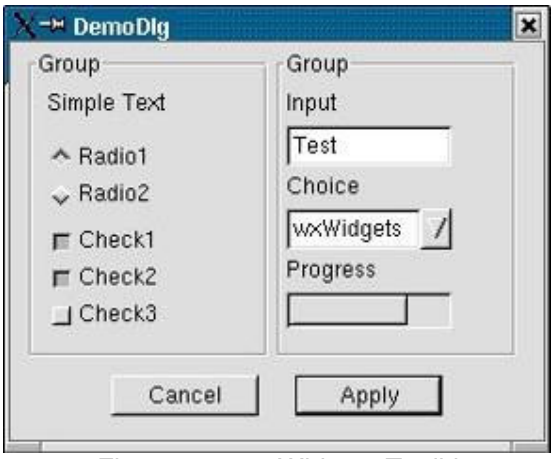


Figura 28 – wxWidgets Toolkit
Fonte: Tschater (2004).

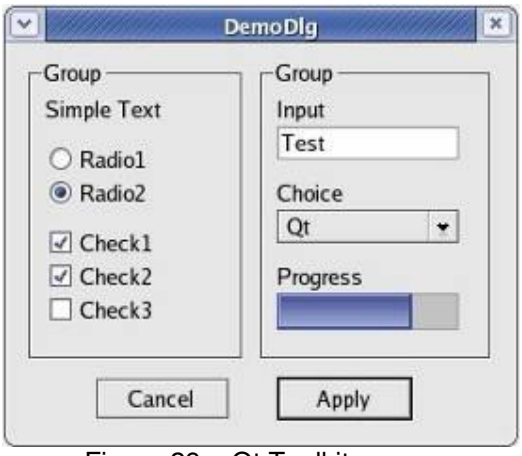


Figura 29 – Qt Toolkit
Fonte: Tschater (2004).

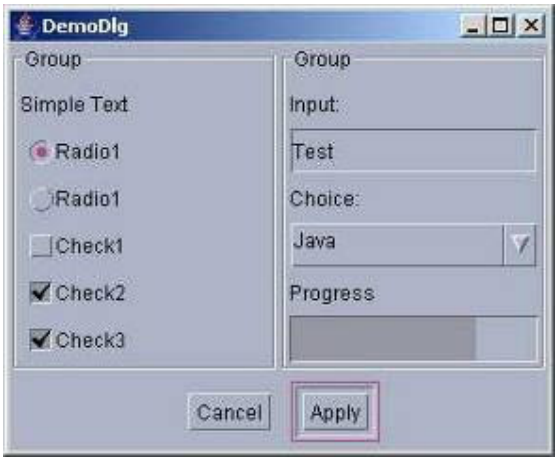


Figura 30 – Java SWING com o tema Motif
Fonte: Tschater (2004).

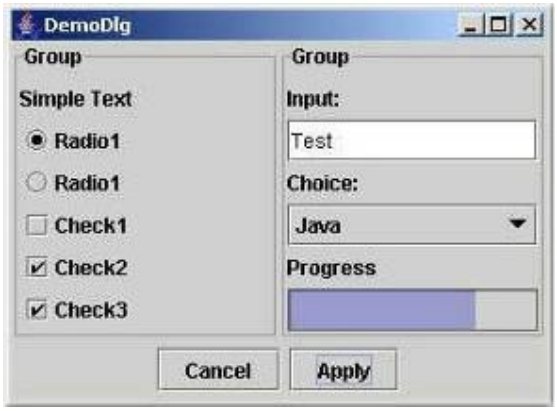


Figura 31 – Java SWING com o tema Metal
Fonte: Tschater (2004).

Toolkits em geral apresentam diversos problemas. O uso de um *toolkit* limita as formas de interação de um aplicativo às formas de interação disponíveis nesse *toolkit*. Outro problema, comum à maioria dos *toolkits* procedurais, é que sua utilização é complicada pela existência de centenas ou até mesmo milhares de rotinas: nem sempre é evidente quais rotinas chamar, e em que ordem, para criar uma certa interface, efeito ou comportamento. Outra dificuldade, também associada aos *toolkits* procedurais, decorre do modelo de entrada, baseado em eventos implementado pelos sistemas de janela: o programador deve registrar uma rotina de *callback* junto ao sistema de janelas para cada evento de *widget* que deseje tratar. O resultado é uma explosão na quantidade de rotinas de tratamento de eventos e a conseqüente dificuldade na leitura e manutenção do código-fonte dos aplicativos, que passam a misturar códigos para interação com o ambiente gráfico e para a implementação da lógica do aplicativo. Esse tipo de problema também ocorre em *toolkits* orientados a objetos, mas em escala reduzida. A maior vantagem do uso de *toolkits* está na reduzida sobrecarga imposta aos sistemas na geração e execução das interfaces gráficas. Tal sobrecarga é, por exemplo, praticamente nula com o uso dos *toolkits* de baixo nível; no caso dos demais *toolkits*, a sobrecarga depende do desempenho individual de diversos elementos como: o modelo de objetos (no caso de *toolkits* orientados a objeto); o mapeamento entre *toolkit* virtual e nativo (quando se tratar de toolkit virtual); a reimplementação dos *widgets* e das primitivas gráficas.

Resumo das Características dos Toolkits	
Utiliza Serviços das Camadas	sistema de janelas, através de APIs
Serviços Oferecidos	widgets
Serviços Oferecidos às Camadas	outros toolkits, ferramentas de alto nível e aplicações
Serviços Oferecidos Através de	APIs procedurais, *framework* de classes
Profissionais que Utilizam	programadores apenas
Papel na Análise	nenhum

3.5 USER INTERFACE MANAGEMENT SYSTEMS (UIMSS)

UIMSs são sistemas que auxiliam nos processos de desenvolvimento e execução da interface humano-computador de aplicativos interativos, oferecendo apoio em *software* para suas diversas etapas: especificação, projeto, definição de protótipos, implementação, execução, alteração e manutenção (HIX, 1990). Os objetivos dos UIMSs eram claros: aumentar a produtividade no desenvolvimento de interfaces interativas (na maioria dos casos TUIs e, principalmente, GUIs); oferecer apoio específico a tarefas ligadas à produção dessas interfaces; apoiar formas alternativas de trabalho em substituição à programação, promovendo a separação entre o desenvolvedor da interface e o programador do aplicativo; viabilizar o desenvolvimento de interfaces melhores e mais avançadas (HIX, 1990). UIMSs deveriam oferecer inúmeras outras facilidades, dentre as quais: consistência, tanto nas interfaces de um mesmo aplicativo quanto nas interfaces dos vários aplicativos desenvolvidos com o mesmo UIMS; apoio a usuários novatos, intermediários e avançados através de múltiplos níveis de ajuda; suporte à manipulação e recuperação de erros; possibilidade de ajuste das interfaces por parte dos usuários (LÖWGREN, 1988). Nos anos 80, UIMSs passaram a ser comparados, em analogia aos papéis que desempenhavam no desenvolvimento de interfaces interativas, a sistemas gerenciadores de bancos de dados (SGBDs) e ferramentas CASE e o papel que esses desempenhavam no desenvolvimento de aplicativos orientados a dados. Ao longo do tempo, os UIMSs foram utilizados no desenvolvimento de interfaces para sistemas baseados nos mais diversos paradigmas de interação: terminais, interfaces textuais (TUIs) e manipulação direta (GUIs).

O primeiro UIMS foi o Reaction Handler, de William Newman, desenvolvido em 1967-68; o termo, entretanto, foi utilizado para descrever essa classe específica de ferramenta somente anos mais, em 1982, por David Kasik (LÖWGREN, 1988; HIX, 1990; MYERS, 1996). Em 1983, em Seeheim, então Alemanha Ocidental, um modelo lógico da arquitetura dos sistemas UIMSs foi proposto. O modelo Seeheim, como ficou conhecido, identificou três componentes principais da arquitetura dos UIMSs e definiu suas atribuições: a camada de apresentação era a responsável pelos aspectos externos da apresentação da interface do usuário; o controle de diálogo definia a estrutura do diálogo entre o usuário e a aplicação; o modelo de interface da aplicação representava a aplicação do ponto de vista do usuário

(LÖWGREN, 1988). Hartson e Hix (1989) observaram a diferença entre os conceitos de diálogo e interface: “[...] diálogo é a troca bilateral, observável, de símbolos e ações entre homem e computador, enquanto interface é o software de apoio e o hardware através do qual essa troca acontece”. Assim, UIMSs que implementassem o modelo Seeheim deveriam ser responsáveis tanto pelo gerenciamento do diálogo quanto pela porção de *software* da interface com o usuário.

Em 1990, Hix propôs a classificação dos UIMSs a partir de suas características comuns, identificando quatro gerações distintas de tais sistemas. A primeira geração, chamada de pré-UIMS, era voltada para programadores apenas. Promovia o uso de fachadas visuais (*façades*), através do uso de protótipos e gerenciadores de exibição. As interfaces eram especificadas através de linguagens formais, tipicamente gramáticas BNF, e complementadas pelo uso de linguagens de programação convencionais. Os protótipos gerados demonstravam a interface do sistema, mas, em geral, as interfaces de produção eram reprogramadas no aplicativo. Sistemas desse tipo ajudavam, basicamente, no gerenciamento da seqüência de diálogos e no desenho das telas e menus. Exemplos de sistemas da primeira geração incluem o Act/1 da Mason and Carey, o Interactive System Productivity Facility da IBM e o Form Management System da DEC (HIX, 1990).

Sistemas da segunda geração passaram a suportar a execução das interfaces. Ainda davam pouca ênfase à etapa de projeto e ao usuário da interface e, ao invés de gramáticas BNF, utilizavam diagramas de estado para representar a interface. Separavam formalmente as interfaces do restante da aplicação e admitiam uma maior quantidade de estilos de interação. Na sua maioria, esses sistemas eram de uso restrito do programador e cobriam apenas uma quantidade limitada de estilos, técnicas e dispositivos de interação; alguns desses sistemas passaram ainda a permitir o desenvolvimento de interfaces gráficas. Na medida em que os UIMSs da segunda geração começaram a abordar uma maior quantidade de etapas do processo de desenvolvimento, iniciou-se a transição para a terceira geração. São exemplos de sistemas da segunda geração: o Menulay de Buxton; o UIDE de Granor e Badler; o Toolkit UIMS de Kasik (HIX, 1990).

A terceira geração de UIMSs passou a integrar, além da execução das interfaces, outras atividades do processo de desenvolvimento das interfaces. Sistemas dessa geração passaram a oferecer ferramentas interativas de manipulação direta para a construção visual das interfaces, indicando uma clara

tendência desses sistemas em concentrar mais esforços para facilitar seu uso. Começaram a sustentar o desenvolvimento de interfaces de manipulação direta (WIMP), raramente suportadas por UIMSs de gerações anteriores. Com isso, o modelo de diálogos passou de síncrono e sequencial para assíncrono, forma mais natural às novas interfaces de manipulação direta. A terceira geração marca o definitivo amadurecimento dos UIMSs e o reconhecimento de sua importância como uma abordagem viável para o desenvolvimento das interfaces de aplicativos. Exemplos de sistemas da terceira geração incluem: os UIMSs das universidades de Alberta, no Canadá, e George Washington, nos Estados Unidos; Peridot, de Myer; Open Dialog, da Apollo; DMS, de Hartson e colegas (HIX, 1990).

A quarta e última geração dos UIMSs continuou a expandir em funcionalidade e facilidade de uso. As ferramentas passaram a dar melhor suporte ao modelo de desenvolvimento interativo das interfaces de manipulação direta. Aspectos complexos, como interfaces dinâmicas, eram tratados por alguns sistemas. Técnicas de inteligência artificial, como sistemas especialistas, começaram a ser utilizadas para melhorar as interfaces, do ponto de vista da engenharia de uso (*usability engineering*). A prototipação rápida passou a ser um elemento central das ferramentas dessa geração. Exemplos de sistemas dessa geração incluem: Serpent, do Software Engineering Institute; Transportable Applications Environment Plus, da NASA; e NextStep (Figura 32), da Next (HIX, 1990).

Figura 32 – Ambiente NextStep
Fonte: HISTORY of the graphical user interface (WIKIPEDIA, 2005).

A despeito do empenho da comunidade científica nas pesquisas em torno das tecnologias associadas aos UIMSs, estes não foram amplamente utilizados na prática. Dentre algumas de suas principais limitações estavam: sua falha na proposta de separação entre interface e aplicação, já que as respostas semânticas adequadas (camada de aplicação) tinham que ser implementadas na camada de interface para garantir a interatividade do aplicativo; sua limitação em relação aos tipos de interfaces suportadas, particularmente, interfaces gráficas de características dinâmicas; a ausência de portabilidade dos UIMSs entre máquinas, sistemas operacionais e sistemas gráficos (LÖWGREN, 1988).

esumo das Características dos UIMSs	
Utiliza Serviços das Camadas	toolkits e outras ferramentas de alto nível
Serviços Oferecidos	interfaces completas para aplicativos
Serviços Oferecidos às Camadas	outros toolkits, ferramentas de alto nível e aplicativos
Serviços Oferecidos Através de	ferramentas e linguagens especializadas, geração de código, interfaces executáveis
Profissionais que Utilizam	tipicamente, programadores; analistas e profissionais de interfaces, através de ferramentas especializadas
Papel na Análise	indireto, através do uso de ferramentas de modelagem e da construção de protótipos

3.6 CONSTRUTORES DE INTERFACE

Sistemas modernos, na maioria, são baseados numa plataforma padronizada, constituída de gerenciadores de janelas, *toolkits* e construtores de interfaces (MYERS; HUDSON; PAUSH, 2000). Construtores de interface (CIs) são ferramentas interativas que permite a criação de interfaces gráficas a partir da manipulação de componentes na tela (MYERS; HUDSON; PAUSH, 2004). Essas ferramentas utilizam uma área de desenho onde os componentes são dispostos para especificar visualmente a interface que, posteriormente, pode ser ligada ao aplicativo (SZEKELEY, 1994). Uma linguagem de programação é utilizada para especificar os

comportamentos da interface; geralmente, essa mesma linguagem é utilizada para implementar a funcionalidade da aplicação (SZEKELEY, 1994).

Um dos aspectos marcantes que contribui para o sucesso dos construtores de interface diz respeito às questões cognitivas mencionadas anteriormente. Ao utilizar um ambiente gráfico para expressar conceitos gráficos, os construtores reduzem o esforço cognitivo exigido do programador para o desenvolvimento da interface, facilitando tanto o aprendizado quanto o uso da ferramenta (MYERS; HUDSON; PAUSH, 2000).

Muitas vezes utilizados como ferramentas de prototipação, os construtores de interface não se encaixam nessa classificação por uma série de motivos: dão suporte à criação apenas da parte estática da interface, como menus e caixas de diálogo; não isolam os códigos da interface e da aplicação; exigem a seleção de componentes concretos (*widgets*) para especificar a interface, fixando certos aspectos da interface numa etapa ainda inicial do ciclo de desenvolvimento da aplicação (SZEKELEY, 1994).

Construtores de interfaces independentes geram alguma forma de saída adequada para a utilização por outras linguagens de programação ou ambientes de desenvolvimento; essa saída pode ser, por exemplo, XML, código fonte em uma linguagem de programação específica ou código compilado. São exemplos de construtores independentes: Glade User Interface Designer (GNOME, 1998) e Qt Designer (TROLLTECH, 2000). Muitos ambientes integrados de desenvolvimento modernos utilizam construtores de interface, dentre outras ferramentas; exemplos incluem: Borland Delphi (BORLAND, 1995) e MS Visual Studio (MICROSOFT, 1992).

Resumo das Características dos Construtores de Interface	
Utiliza Serviços das Camadas	Toolkits
Serviços Oferecidos	Interfaces completas para aplicativos
Serviços Oferecidos às Camadas	Ferramentas de alto nível e aplicativos
Serviços Oferecidos Através de	Construção das interfaces combinada à programação convencional

Resumo das Características dos Construtores de Interface	
Profissionais que Utilizam	Tipicamente programadores
Papel na Análise	Indireto, através do uso de ferramentas de modelagem para gerar interfaces e da construção de protótipos

3.7 INTERFACES ORIENTADAS A OBJETOS

No seu livro Naked Objects, Pawson e Matthews (2002) observam que uma série de fatores promovem a separação entre procedimentos e dados durante o processo de desenvolvimento, abordagem contrária à metodologia orientada a objetos, que sugere que as unidades funcionais dos sistemas, os objetos, devem controlar seus próprios dados através da abstração de tipos e da disponibilização de interfaces funcionais bem definidas (MEYERS, 1997). As abordagens criticadas por Pawson e Matthews (2002) foram: a análise do sistema do ponto de vista do processo; a otimização das interfaces em torno de tarefas; a utilização de metodologias baseadas em exemplos de uso; o modelo MVC, que promove a separação entre o modelo, a interface visual e o controle dos aplicativos; a utilização de arquiteturas baseadas em componentes para o desenvolvimento do software. Para evitar essa separação entre dados e procedimentos sem perder os benefícios das abordagens citadas e não introduzir novos problemas, os autores sugerem uma prática baseada nos princípios da orientação a objetos.

Uma solução orientada a objetos para geração de interfaces deve isolar, através das práticas de encapsulamento e *information hiding* (ocultação de informação), os aspectos do sistema que tendem a modificar-se com o tempo; com isso, a alteração de qualquer desses aspectos não causa impacto significativo na aplicação (HOLUB, 1999). Modelos de implementação como o MVC falham nesse respeito. No MVC, tipicamente, o controlador realiza duas tarefas: obtém as informações dos objetos do modelo através de funções *get* e as exibe através dos *widgets* da interface visual; extrai as informações dos *widgets* da interface visual, passando-as aos objetos do modelo através de funções *set* (HOLUB, 1999). O problema dessa abordagem está na passagem das informações sobre as entidades entre as diferentes camadas do modelo; a alteração de uma entidade do modelo implicaria na necessidade de recodificação de todas as camadas (HOLUB, 1999).

Ferramentas baseadas em construtores de interface não oferecem solução para esse tipo de problema tampouco. Nelas, a alteração de uma entidade implica na necessidade de alteração de todas as telas construídas onde a entidade era exibida; além disso, os códigos implementando os comportamentos dos *widgets* teriam que ser alterados para lidar com as modificações (HOLUB, 1999).

De acordo com Holub (1999), as regras gerais para identificar um modelos de objetos bem estruturado são: todos os dados de um objeto devem ser privados; métodos acessores, representados, em geral, por rotinas tipo *get* e *set*, são inadequados, pois, violam a primeira regra; objetos não devem fornecer informações para a realização de tarefas, mas, ao contrário, objetos que possuem a informação relevante devem realizar tais tarefas; o modelo de objetos deve permitir a alteração na implementação de um objeto através da modificação desse objeto apenas; por fim, as interfaces de interação com o usuário devem ser oferecidas pelos próprios objetos (HOLUB, 1999, 2003).

Um bom exemplo de sistema orientado a objetos é o Naked Objects, de Pawson e Matthews (2002a). Seu funcionamento é baseado na premissa de que sistemas orientados a objeto devem ser comportamentalmente completos. Segundo esse princípio, cada objeto deve conhecer as propriedades (de interesse) da entidade real representada, além de modelar seus comportamentos- aqueles necessários ao aplicativo. Assim, os comportamentos associados a cada objeto são sempre propriedades desse objeto, ao invés de serem implementados em uma outra parte do sistema (PAWSON; MATTHEWS, 2002a). A maior vantagem do emprego desse princípio fica clara quando há necessidade de alterar o sistema, pois, basta modificar a implementação dos comportamentos dos objetos modificados. Naked Objects incentiva o desenvolvimento de objetos comportamentalmente completos de duas formas: primeiro, pela representação visual direta dos objetos do modelo e, segundo, pela arquitetura do sistema, que não oferece uma alternativa à forma de exibição dos objetos. Dessa forma, a plataforma Naked Objects utiliza os objetos do próprio modelo conceitual da aplicação para gerar automaticamente a interface do usuário. Na interface, a interação do usuário com os objetos se dá através da manipulação direta dos diversos objetos (Figura 33).

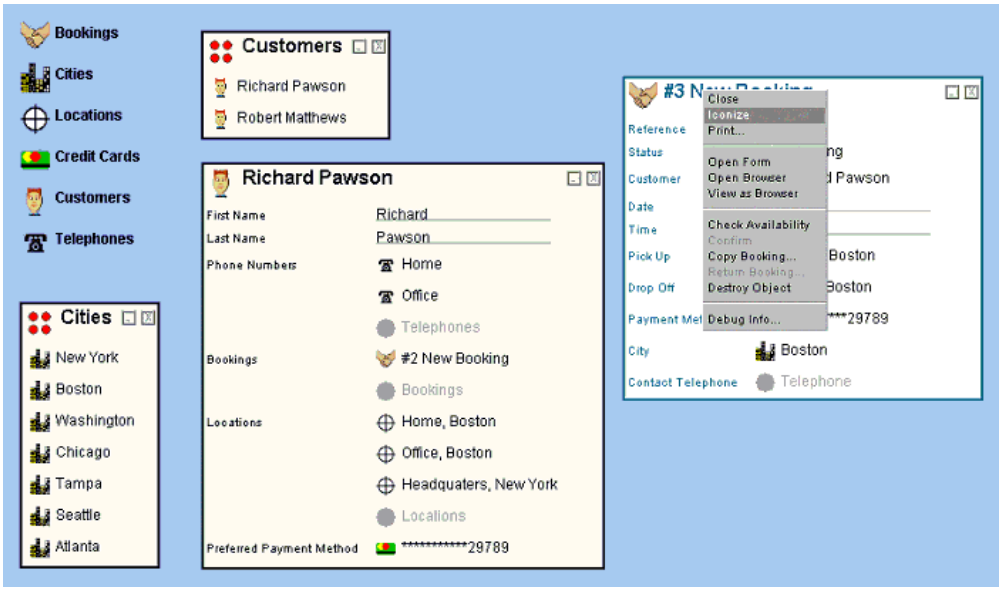


Figura 33 – Interface gerada pelo Naked Objects
Fonte: Pawson e Matthews (2002).

O sistema Naked Objects representa de forma adequada o tipo de benefícios e problemas que um típico sistema orientado a objetos apresenta. Alterações no sistema são possíveis e de fácil realização, através da alteração dos objetos do negócio. Como os comportamentos dos objetos estão encapsulados, a alteração da implementação não interfere com a interface do sistema. Sistemas baseados em interfaces orientadas a objeto utilizam um estilo nome-verbo de interação, ao contrário dos sistemas convencionais, que empregam uma abordagem verbo-nome. No estilo verbo-nome, o usuário seleciona uma tarefa (verbo) de um menu, por exemplo, para, em seguida, oferecer os dados necessários (nome) para completar a tarefa; no estilo nome-verbo, ao contrário, o usuário seleciona um objeto (nome) e então seleciona uma das operações disponíveis (verbo) para realizar sobre esse objeto. Os ambientes de trabalho das interfaces gráficas de manipulação direta utilizam o modelo nome-verbo onde se seleciona um ícone, por exemplo, tendo-se, com o clique do botão direito do *mouse*, acesso às operações disponíveis para realizar sobre aquele objeto. Outras vantagens dos sistemas orientados a objetos incluem ainda: comunicação melhorada entre equipe de desenvolvimento e usuários, já que o negócio (domínio da aplicação) passa a ser a linguagem comum entre as partes; o aumento na velocidade do desenvolvimento, já que os sistemas precisam implementar apenas o modelo de negócios, já que a interface gráfica deve ser fornecida automaticamente pelos próprios objetos (PAWSON; MATTHEWS, 2002a).

Resumo das Características das Interfaces Orientadas a Objeto	
Utiliza Serviços das Camadas	Toolkits
Serviços Oferecidos	Interfaces completas para aplicativos
Serviços Oferecidos às Camadas	Aplicativos
Serviços Oferecidos Através de	Programação das classes de negócio
Profissionais que Utilizam	Programadores
Papel na Análise	Indireto, através da geração automática das interfaces a partir das classes do modelo de negócios

3.8 ABORDAGEM BASEADA EM PADRÕES

Os conceitos de padrão e linguagem de padrões foram derivados da arquitetura, propostos inicialmente por Alexander, Ishikawa, Silverstein et al. (1977) (TODD; KEMP; PHILLIPS, 2004). De acordo com a definição original, padrão é "[...] uma regra em três partes que expressa a relação entre um certo contexto, um problema e uma solução. [...] Cada padrão descreve um problema que ocorre repetidamente em nosso meio e então descreve o elemento fundamental da solução, de forma tal que seja possível utilizar essa solução milhões de vezes. [...]" (1977 apud VAN WELIE; VAN DER VEER; ELIËNS, [ca. 2002]). Portanto, padrões apresentam uma solução provada através de um contexto e permitem determinar quando, como e porque tal solução pode ser aplicada (VAN WELIE; VAN DER VEER; ELIËNS, [ca. 2002]).

Padrões de interfaces visuais são geralmente comparados a guias de estilo (VAN WELIE; VAN DER VEER; ELIËNS, [ca. 2002]). No entanto, guias de estilo são utilizados para capturar conhecimento acerca do desenvolvimento de interfaces através de regras e, ao contrário dos padrões, não são provadas e nem comunicadas através de um contexto (VAN WELIE; VAN DER VEER; ELIËNS, [ca. 2000]).

Van Welie e Trætteberg ([ca. 2000]) sugerem que identificar padrões em

interfaces não é difícil mas, destacar aqueles que trazem benefícios ao usuário e descrever os aspectos relacionados a seu uso, pode ser. Consideram também que contra-exemplos podem ser úteis para motivar a adoção de padrões como uma ferramenta de projeto.

Tidwell (1998) considera que o uso de uma linguagem de padrões, conjunto inter-relacionado de padrões baseados nas mesmas premissas, terminologias e contextos, levaria ao desenvolvimento de interfaces melhores por diversas razões: ao capturar o conhecimento de uma coletividade de modo que possa ser utilizado de imediato; oferecendo uma linguagem comum que pode compreendida e utilizada por desenvolvedores, usuários e programadores da interface; ao permitir que padrões conhecidos possam ser reinterpretados e utilizados em novos contextos; enfatizando valores essenciais da interface como sua facilidade de uso, consistência, estética e conforto; expressando invariantes da interface de modo que sejam facilmente codificadas no *software* (TIDWELL, 1998). Granlund, Lafrenière e Carr (2001) complementam essa lista indicando que linguagens de padrão são uma forma adequada para a documentação dos conhecimentos de desenvolvimento de interfaces pois são apresentados de forma consistente, fácil de ler e oferecem um contexto adequado para o entendimento.

No artigo *Interaction Patterns in User Interfaces*, Van Welie e Trætteberg apresentam uma coleção de padrões orientados à resolução de problemas que usuários enfrentam no uso de sistemas (VAN WELIE; TRÆTTEBERG, [ca. 2000]). Os padrões apresentados adotam uma perspectiva centrada no usuário, onde a principal qualidade observada na identificação dos padrões foi sua facilidade de uso. Os padrões apresentados foram classificados ao longo das seguintes dimensões: problema, princípio de uso, contexto, motivação, solução, explicação, exemplos e contra-exemplos, usos conhecidos e padrões relacionados (VAN WELIE; TRÆTTEBERG, 2000).

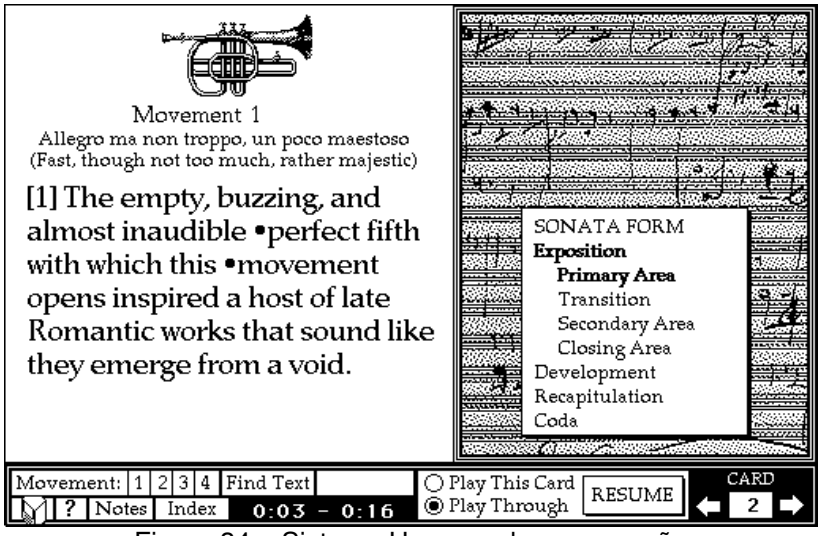


Figura 34 – Sistema Hypercard em execução
Fonte: Hypercard (WIKIPEDIA, 2005).

Padrões de interfaces visuais são geralmente comparados a guias de estilo (VAN WELIE; VAN DER VEER; ELIËNS, [ca. 2002]). No entanto, guias de estilo são utilizados para capturar conhecimento acerca do desenvolvimento de interfaces através de regras e, ao contrário dos padrões, não são provadas e nem comunicadas através de um contexto (VAN WELIE; VAN DER VEER; ELIËNS, [ca. 2000]).

Van Welie e Trætteberg ([ca. 2000]) sugerem que identificar padrões em interfaces não é difícil mas, destacar aqueles que trazem benefícios ao usuário e descrever os aspectos relacionados a seu uso, pode ser. Consideram também que contra-exemplos podem ser úteis para motivar a adoção de padrões como uma ferramenta de projeto.

Tidwell (1998) considera que o uso de uma linguagem de padrões, conjunto inter-relacionado de padrões baseados nas mesmas premissas, terminologias e contextos, levaria ao desenvolvimento de interfaces melhores por diversas razões: ao capturar o conhecimento de uma coletividade de modo que possa ser utilizado de imediato; oferecendo uma linguagem comum que pode compreendida e utilizada por desenvolvedores, usuários e programadores da interface; ao permitir que padrões conhecidos possam ser reinterpretados e utilizados em novos contextos; enfatizando valores essenciais da interface como sua facilidade de uso, consistência, estética e conforto; expressando invariantes da interface de modo que sejam facilmente

codificadas no *software* (TIDWELL, 1998). Granlund, Lafrenière e Carr (2001) complementam essa lista indicando que linguagens de padrão são uma forma adequada para a documentação dos conhecimentos de desenvolvimento de interfaces pois são apresentados de forma consistente, fácil de ler e oferecem um contexto adequado para o entendimento.

No artigo *Interaction Patterns in User Interfaces*, Van Welie e Trætteberg apresentam uma coleção de padrões orientados à resolução de problemas que usuários enfrentam no uso de sistemas (VAN WELIE; TRÆTTEBERG, [ca. 2000]). Os padrões apresentados adotam uma perspectiva centrada no usuário, onde a principal qualidade observada na identificação dos padrões foi sua facilidade de uso. Os padrões apresentados foram classificados ao longo das seguintes dimensões: problema, princípio de uso, contexto, motivação, solução, explicação, exemplos e contra-exemplos, usos conhecidos e padrões relacionados (VAN WELIE; TRÆTTEBERG, 2000).

Mais recentemente, Todd, Kemp e Phillips (2004) propuseram um modelo para determinar a validade de uma linguagem de padrões, estabelecendo, através desse modelo, a diferença entre coleções e linguagens de padrão: linguagens são coleções de padrões relacionados, organizados em uma ou mais hierarquias interligadas, que devem oferecer orientação na forma de utilização das combinações de padrões dessa coleção.

No meio comercial, a SAP, grande fornecedora mundial de *software* corporativo, está entre as pioneiras na aplicação de padrões no desenvolvimento das interfaces de seus produtos (LATZINA, 2003; AREND, 2003; WALOSZEK; EBERLEH, 2003; VERING, 2004). Em um artigo, Arend (2003) explora diversos padrões e aspectos de uso empregados no desenvolvimento de uma interface de faturamento.

Resumo das Características da Abordagem Baseada em Padrões	
Utiliza Serviços das Camadas	Toolkits e outras ferramentas
Serviços Oferecidos	Modelos de interfaces em todas as escalas através de coleções e implementações
Serviços Oferecidos às Camadas	Aplicativos
Serviços Oferecidos Através de	Coleções e frameworks

Resumo das Características da Abordagem Baseada em Padrões	
Profissionais que Utilizam	Programadores, analistas e usuários
PAPEL NA ANÁLISE	Especificação das Uis; comunicação entre usuários, analistas e programadores

3.9 OUTRAS TÉCNICAS E FERRAMENTAS

A seguir, são apresentadas outras classes de ferramentas de menor expressão, aceitação ou que não se enquadrem em alguma das classificações anteriores.

3.9.1 Sistemas Baseados em Modelos

Segundo Puerta e Eisenstein (1999), sistemas de desenvolvimento de interfaces baseados em modelo (MB-UIDEs) baseavam-se na representação declarativa do modelo da interface. Tal representação deveria conter todos os aspectos relevantes da interface, numa linguagem específica para sua modelagem. O maior problema desse tipo de sistema era a necessidade de mapear as informações entre os diversos componentes do modelo da interface, um problema de difícil solução dada a quantidade de variáveis que afetavam esse mapeamento (PUERTA; EISENSTEIN, 1999). Em tempo de desenvolvimento, as interfaces dos sistemas MB-UIDE são construídas e modificadas através de uma ferramenta específica, onde o usuário especifica o que a interface deve apresentar, ao invés de como exibir essa interface (SZEKELEY, 1994).

3.9.2 Geração Automática de Interfaces

Ferramentas de geração automática de interfaces surgiram para resolver um problema associado ao uso de ferramentas baseadas em linguagens: a necessidade de especificar uma quantidade de informações a respeito da interface, como posição, dimensões, formatos e estilos (MYERS, 1995). Nesse tipo de ferramenta, tais escolhas eram realizadas através de uma especificação de alto nível, como um

modelo entidade-relacionamento (ER), no caso de aplicações orientadas a dados (MYERS, 1995; PIZANO, 1993).

Myers (1995) identifica dois problemas com ferramentas de geração automática: primeiro, as interfaces geradas não apresentam qualidade satisfatória; segundo, quando uma linguagem de especificação é utilizada, ao invés de um modelo existente, dificuldades relacionadas com o tempo de aprendizado e o uso da ferramenta tornam-se problemas consideráveis. Pizano (1993), de Baar (1992) e Vanderdonckt (1994) concordam que ferramentas de geração automática são adequadas quando aplicadas a domínios específicos, como na geração de interfaces para aplicações orientadas a dados.

3.9.3 Hypercard

O Apple Hypercard é um sistema original, de difícil classificação. Foi utilizado para a criação e execução de interfaces gráficas baseadas num paradigma visual de cartas e pilhas de cartas. Dentre inúmeras aplicações, foi amplamente utilizado para a prototipação rápida de aplicativos e o desenvolvimento de sistemas hipertexto de baixa complexidade. Possuía uma linguagem de *scripts* integrada, o Hypertalk, que permitia a definição do comportamento dos elementos da interface visual. Apresentava três modos de operação- navegação, autoria e programação. No modo de navegação, o usuário interagia com as interfaces do sistema, mas não as modificava; no modo de autoria, o usuário podia criar ligações entre cartões, alterar a disposição de elementos na tela e editar *scripts*; por fim, o modo de programação dava acesso a todas as funcionalidades do sistema, inclusive a criação de novos cartões e *scripts* (SZEKELEY, 1994). Uma típica tela do HyperCard em execução é apresentada na Figura 34.

3.9.4 Outras Ferramentas

Diversas outras ferramentas foram omitidas. Em geral, são abordagens de interesse apenas histórico ou acadêmico que contribuíram em maior ou menor grau para o desenvolvimento das demais ferramentas descritas anteriormente neste capítulo. Exemplos incluem: redes de mudança de estado, gramáticas de contexto livre, linguagens baseadas em eventos, linguagens declarativas, linguagens de restrição, *screen scrapers*, programação visual, programação baseada em

demonstração, ferramentas para o desenvolvimento de protótipos e ferramentas de visualização (MYERS, 1995).

3.10 LEVANTAMENTO DE AMBIENTES DE DESENVOLVIMENTO

Esta seção apresenta um levantamento realizado a fim de estabelecer o atual grau de apoio ao desenvolvimento de interfaces gráficas de manipulação direta tipicamente oferecido pelos ambientes integrados de desenvolvimento (IDEs). Os resultados desse levantamento serão utilizados para subsidiar a estratégia de desenvolvimento da implementação de referência da coleção de padrões elaborada no trabalho.

3.10.1 Objetivo

O objetivo deste levantamento foi avaliar o tipo de apoio oferecido por ambientes integrados de desenvolvimento (IDEs) à elaboração de interfaces gráficas de manipulação direta.

3.10.2 Metodologia

O levantamento realizado foi de natureza não-probabilística e a seleção da amostra intencional; tal abordagem permitiu a seleção dos elementos mais representativos da população (OLIVEIRA, 2001). As ferramentas foram selecionadas a partir dos seguintes critérios: sua popularidade – isto é, o aparente prestígio que gozam nas comunidades de desenvolvimento; as plataformas suportadas, sendo aquelas utilizadas nos âmbitos corporativo e comercial do mercado brasileiro preferidas em relação às demais; característica da licença – ferramentas com licenças *open source* foram preferidas em relação às demais, devido à facilidade que as primeiras possuem em proliferar-se rapidamente e, portanto, influenciar o modelo de desenvolvimento de um número maior de programadores. Considera-se ainda, sobre licenças, que as de código aberto representam tecnologias intermediárias, amadurecidas e amplamente disponíveis, enquanto as comerciais representam tecnologias avançadas (“estado da arte”) disponíveis a uma população mais restrita.

Selecionadas as ferramentas, as versões mais atuais de cada uma delas, à

época do estudo, foram obtidas através dos respectivos *web sites*, na Internet. Em seguida, foram instaladas e testadas. Três dimensões foram avaliadas:

1. Se a ferramenta possuía um construtor de interfaces integrado, através do qual as interfaces do aplicativo podiam ser construídas de forma visual, na tela, através da seleção e manipulação de *widgets*, em tempo de desenvolvimento.
2. Se a ferramenta disponibilizava, nativamente ou através de módulos dinâmicos carregados no ambiente (*plug-ins*), alguma forma de assistente ou guia interativo para a geração de interfaces a partir de informações sobre como a interface deveria comportar-se (como guias para criação de formulários de entrada de dados, listagens e relatórios).

Tabela 2 – Ambientes de Desenvolvimento (parte 1)

Ferramenta	Licença	Plataforma	URL
#develop	Open Source	C#, VB.NET	http://www.icsharpcode.net/OpenSource/SD/
Amaya	Open Source	Web	http://www.w3.org/Amaya/
Anjuta C/C++ IDE	Open Source	C/C++	http://anjuta.sourceforge.net/
Bloodshed Dev-C++	Open Source	C++	http://www.bloodshed.net/devcpp.html
Bloodshed Dev-Pascal	Open Source	Pascal	http://www.bloodshed.net/devpascal.html
Blue Fish	Open Source	Web	http://bluefish.openoffice.nl/
BlueJ	Freeware	Java	http://www.bluej.org/
Boa Constructor	Open Source	Python	http://boa-constructor.sourceforge.net/
Conglomerate	Open Source	XML	http://www.conglomerate.org/
Delphi	Comercial	Pascal, C++	http://borland.com/delphi
Dev-PHP	Open Source	PHP	http://devphp.sourceforge.net/
DrJava	Open Source	Java	http://sourceforge.net/projects/drjava/
DrPython	Open Source	Python	http://drpython.sourceforge.net/
DrScheme	Open Source	Scheme	http://www.drscheme.org/
Eclipse	Open Source	Java	http://www.eclipse.org/
Eiffel Studio	Freeware	Eiffel	http://www.eiffel.com/
eric3	Open Source	Python	http://www.die-offenbachs.de/detlev/eric3.html
FreeRIDE	Open Source	Ruby	http://freeride.rubyforge.org/
Gambas	Open Source	Basic	http://gambas.sourceforge.net/
Glade	Open Source	Interfaces GTK	http://glade.gnome.org/
GNAVI	Open Source	ADA	http://www.gnavi.org/
HBasic	Open Source	Basic	http://hbasic.sourceforge.net/
JBuilder	Comercial	Java	http://borland.com/jbuilder
JCreator	Freeware	Java	http://www.jcreator.com/
KBasic	Freeware	Basic	http://www.kbasic.de/
KDevelop	Open Source	Vários	http://www.kdevelop.org/
Komodo	Comercial	Perl, Python, PHP, Outros	http://www.activestate.com/Products/Komodo/
Kylix	Comercial	Pascal, C++	http://www.borland.com/kylix/
Lazarus	Open Source	Pascal	http://www.lazarus.freepascal.org/
Mindscript	Open Source	Mindscript	http://mindscript.sourceforge.net/
MonoDevelop	Open Source	C#, VB.NET, Outros	http://www.monodevelop.com/
NetBeans	Freeware	Java	http://www.netbeans.org/
NVU	Open Source	Web	http://www.nvu.com/
Phoenix	Freeware	Basic	http://www.janus-software.com/phoenix_features.html
Pow!	Open Source	Oberon-2, Java, C/C++	http://www.fim.uni-linz.ac.at/pow/Pow.htm
PythonCard	Open Source	Python	http://pythoncard.sourceforge.net/
Qt Designer	Freeware	Interfaces Qt	http://www.trolltech.com/products/qt/designer.html
Rapid	Open Source	ADA	http://unicoi.kennesaw.edu/ase/ase02_02/tools/usafa/rapid/
RHIDE	Open Source	Vários	http://www.rhide.com/
Screem	Open Source	Web	http://www.screem.org/
Ultimate++	Open Source	Ultimate++, C++	http://upp.sourceforge.net/
VIDE	Open Source	g++, Java	http://www.objectcentral.com/vide.htm
Virtual Pascal	Freeware	Pascal	http://www.vpascal.com/
Visual Eiffel	Open Source	Eiffel	http://visual-eiffel.org/
Visual Prolog	Freeware	Prolog	http://www.visual-prolog.com/vip6/
Visual Studio .NET	Comercial	C#, VB.NET, Outros	http://msdn.microsoft.com/vstudio/
Visual Tcl	Open Source	Tcl/Tk	http://vtcl.sourceforge.net/
WideStudio	Freeware	Vários	http://www.widestudio.org/EE/index.html
XBasic	Open Source	Basic	http://xbasic.sourceforge.net/

3. Se a ferramenta disponibilizava outras formas de auxílio ao desenvolvimento das interfaces gráficas como, por exemplo, editores de ícones, menus e outros recursos gráficos do aplicativo; editores WYSIWYG para a plataforma *web*; suporte à programação visual; *plug-ins* para extensão do ambiente com outras ferramentas gráficas; capacidades avançadas no construtor de interfaces como repositórios de modelos e programação baseada em modelos.

As ferramentas foram marcadas com “sim” ou “não” para cada dimensão, representando a disponibilidade ou ausência dessas características na ferramenta. Considerou-se ferramenta de “apoio limitado” aquelas que ofereciam uma ou nenhuma das formas de apoio destacadas.

3.10.3 Resultados

Foram examinados 49 ambientes integrados de desenvolvimento (IDEs) em uso no mercado para identificar o tipo de apoio ao desenvolvimento de interfaces gráficas (GUIs) oferecido. Os resultados desse levantamento são apresentados nas Tabelas 2 e 3. A tabela 2 lista os nomes das ferramentas, o tipo de licença empregado pela versão testada, a principal plataforma suportada e a URL a partir de onde a versão foi baixada. Os tipos de licença foram: *open source*, para licenças aprovadas pelo Open Source Initiative (OPENSOURCE, 2005); *freeware*, para os diversos tipos de licenças gratuitas- avaliação, uso não comercial, uso pessoal, reduzidas ou livres; comerciais. As plataformas identificaram a linguagem utilizada ou a tecnologia empregada; “vários” indica que diversas linguagens estão disponíveis; “outros” indica que mais linguagens são suportadas além das listadas; “web” indica suporte a HTML e, possivelmente, linguagens de *script* do lado do cliente. A Tabela 3 apresenta a versão testada da ferramenta, a data dessa versão e a indicação, com “sim” ou “não”, da disponibilidade de construtores de interface, assistentes e outras ferramentas auxiliares para o desenvolvimento das interfaces gráficas.

Dos ambientes analisados, 5 (10,2%) eram versões comerciais, 10 (20,4%) *freeware* e 34 (69,4%) *open source*. Em termos suporte exclusivo de plataforma, pelo menos 8 (16,3%) suportavam Java, 8 (16,3%) linguagens de *script* (PYTHON,

RUBY; PERL, 2005), 5 (10,2%) C/C++, 5 (10,2%) Pascal, 5 (10,2%) Basic, 4 (8,2%) a plataforma *web*, 3 (6,1%) a plataforma .NET e 2 (4,1%) eram construtores de interface. Não foi possível determinar a data da versão de 10 (20,4%) ferramentas; 27 (55,1%) eram versões 2.0 ou superiores e 11 (22,4%) versões anteriores à 1.0; 30 (61,2%) haviam sido lançadas nos últimos 12 meses, 3 (6,1%) no ano de 2003 e 4 (8,2%) antes de 2003. Dos IDEs, 30 (61,2%) integravam construtores de interfaces, 11 (22,4%) ofereciam assistentes, 11 (22,4%) disponibilizavam outra forma de auxílio, 15 (30,6%) não suportavam forma alguma de auxílio. Somente 6 (12,2%) IDEs ofereciam os três tipos de auxílio enquanto 37 (75,5%) ofereciam apoio limitado à construção de interfaces (um tipo de apoio ou nenhum apoio).

Tabela 3 – Ambientes de Desenvolvimento (parte 2)

Ferramenta	Versão Testada	Data da Versão	Construtor	Assistente	Outros
#develop	1.1.0	04/05	**sim**	**sim**	não
Amaya	0.9.0	02/05	**sim**	não	não
Anjuta C/C++ IDE	1.2.2	04/04	não	não	não
Bloodshed Dev-C++	4.9.9	02/05	não	não	não
Bloodshed Dev-Pascal	1.9.1	ND	não	não	não
Blue Fish	1.0.1	05/05	**sim**	não	não
BlueJ	2.0.4	ND	não	não	**sim**
Boa Constructor	0.40	03/05	**sim**	não	não
Conglomerate	0.9.0	02/05	não	**sim**	não
Delphi	2005	ND	**sim**	**sim**	**sim**
Dev-PHP	2.0.12	03/05	não	não	não
DrJava	20050510-0024	05/05	não	não	não
DrPython	3.10.13	05/05	não	não	não
DrScheme	299.100	ND	não	não	não
Eclipse	3.0.2	ND	**sim**	**sim**	**sim**
Eiffel Studio	5.5	09/04	**sim**	não	não
eric3	3.6.2	02/05	**sim**	**sim**	não
FreeRIDE	0.9.4	04/05	não	não	não
Gambas	1.9.8	05/05	**sim**	não	**sim**
Glade	2.10.0	03/05	**sim**	não	não
GNAVI	2004	12/04	**sim**	não	não
HBasic	0.9.9od	08/04	**sim**	não	não
JBuilder	2005	ND	**sim**	**sim**	**sim**
JCreator	3.50	04/05	não	não	não
KBasic	preview	04/05	**sim**	não	não
KDevelop	3.2.0	03/05	**sim**	não	não
Komodo	3.1	02/05	**sim**	não	não
Kylix	3.0	ND	**sim**	**sim**	**sim**
Lazarus	0.9.6	02/05	**sim**	não	não
Mindscript	0.87	04/05	não	não	**sim**
MonoDevelop	0.6	03/05	não	não	não
NetBeans	4.1	05/05	**sim**	**sim**	**sim**
NVU	1.0PR	04/05	**sim**	não	**sim**
Phoenix	1.5.6	1996	**sim**	não	**sim**
Pow!	3.0b	10/00	não	não	não
PythonCard	0.8.1	10/04	**sim**	não	não
Qt Designer	3.3.4	ND	**sim**	não	não
Rapid	3.0	06/00	**sim**	não	não
RHIDE	1.5	02/03	não	não	não
Screem	0.14.1	05/05	não	**sim**	não
Ultimate++	0.98.5	04/05	**sim**	não	não
VIDE	2.0	04/03	não	não	não
Virtual Pascal	2.1	05/04	não	não	não
Visual Eiffel	5.0	03/05	**sim**	não	não
Visual Prolog	6.2	ND	**sim**	não	não
Visual Studio .NET	2005	ND	**sim**	**sim**	**sim**
Visual Tcl	1.6.0	08/03	**sim**	não	não
WideStudio	3.90.5	05/05	**sim**	**sim**	não
XBasic	6.2.3	10/02	não	não	não

4 METODOLOGIA

Este capítulo identifica e descreve cada um dos procedimentos realizados no desenvolvimento do trabalho. Primeiro, o trabalho é caracterizado e sua estrutura descrita; em seguida, os procedimentos específicos realizados em cada etapa são detalhados.

4.1 ESTRUTURA

Essencialmente, este é um trabalho de engenharia de software que combina a elaboração de uma coleção de padrões de interfaces gráficas com a implementação de referência de uma biblioteca de código, numa linguagem de programação específica, para a geração dessas interfaces.

A elaboração da coleção de padrões de interfaces gráficas, primeira etapa do trabalho, foi um processo dividido em três partes: a definição do idioma para a descrição dos padrões; a identificação de um conjunto de padrões; a documentação desses padrões.

Uma vez construída a coleção de padrões, a etapa seguinte foi o desenvolvimento da biblioteca de padrões de interfaces. Essa parte do trabalho consistiu na modelagem e implementação de um conjunto de classes para oferecer apoio ao desenvolvimento de interfaces baseadas nos padrões descritos na coleção.

Tanto a coleção de padrões quanto a implementação de referência limitaram seus alcances a uma plataforma específica de desenvolvimento, o Windows com o Borland Delphi, e uma classe bem definida de aplicações suportadas- tipicamente, sistemas de informação.

4.2 ELABORAÇÃO DA COLEÇÃO DE PADRÕES

4.2.1 Escopo

A elaboração da coleção de padrões limitou seu escopo a interfaces de aplicativos baseados no modelo relacional, tipicamente sistemas comerciais e sistemas de informação, rodando localmente no computador. Com essa limitação, os usuários e ambientes de uso da coleção passam a ser bem restritos.

4.2.2 Definição do Idioma de Padrões

A primeira etapa do processo de elaboração da coleção de padrões foi a criação de um idioma para descrever cada padrão (VAN WELIE; TRÆTTEBERG, [ca. 2000]). Originalmente, padrões foram definidos através de um contexto, um problema e uma solução (ALEXANDER et al., 1977 apud VAN WELIE; VAN DER VEER; ELIËNS, [ca. 2002]). No idioma proposto neste trabalho, assim como naqueles propostos por Van Welie e Trætteberg ([ca. 2000]) e por Tidwell (1998), além das propriedades originalmente identificadas por Alexander, outras foram introduzidas na especificação dos padrões, na tentativa de facilitar o seu emprego na prática. No total, foram sete as dimensões utilizadas no idioma:

1. ***Nome:*** identifica o padrão de forma única na coleção; é também uma forma de antecipar o tipo de interação que o padrão descreve.
2. ***Grupo:*** classifica o padrão de acordo com o tipo de interação implementada.
3. ***Problema:*** especifica o problema de interação que o padrão procura resolver.
4. ***Contexto:*** identifica as circunstâncias nas quais o padrão pode ser usado; os usuários e ambientes estão naturalmente definidos pela limitação no escopo da coleção.
5. ***Solução:*** descreve objetivamente uma forma de resolver a parte central do problema, sem introduzir novos problema, deixando

possíveis detalhes de lado.

6. ***Benefícios:*** apresenta os ganhos que a aplicação do padrão oferece; alguns exemplos são: performance, satisfação, facilidade de aprendizado, retenção de aprendizado e redução de erros.
7. ***Exemplo:*** ilustra implementações bem sucedidas do padrão, possivelmente com texto adicional explicando os detalhes da solução empregada.

E cinco os grupos utilizados para classificar os padrões:

1. **Indicador Visual:** auxilia o usuário na interação com a interface através de avisos visuais na tela como efeitos de cores.
2. **Controle:** pode referir-se a duas formas de controle distintas- controle da forma de exibição da interface visual ou controle das operações disponíveis através da interface visual.
3. **Acesso a Dados:** permite que dados sejam acessados dinamicamente, mas não alterados. Tipicamente, o acesso é empregado como tarefa auxiliar na edição de dados.
4. **Editor de Dados:** implementa a manipulação de entidades de dados.
5. **Saída de Dados:** permite a apresentação de dados através de um dispositivo adequado de saída (arquivo, folha impressa, etc.).

4.2.3 Identificação dos Padrões

Definido o idioma, a etapa seguinte foi a identificação dos padrões. Para isso, foram utilizadas coleções de padrões já publicadas (TIDWELL, 2005; VAN WELIE, 2003-2005; VAN WELIE, 2000), alguns padrões discutidos em artigos (AREND, 2003; WALOSZEK; EBERLEH, 2003) e outros padrões empregados repetidamente pelo próprio autor em suas atividades profissionais (LESSA, 2004). A identificação foi realizada através da análise e comparação entre os padrões das diversas fontes mencionadas, levando ainda em conta o escopo dos padrões que se pretendia identificar.

4.2.4 Descrição dos Padrões

A etapa final da elaboração da coleção de padrões foi descrever, para cada um dos padrões identificados, as dimensões do idioma proposto. As fontes anteriormente citadas ofereceram a orientação necessária que, combinada à experiência anterior do autor com os diversos padrões identificados, permitiram a descrição de cada um deles.

A coleção construída independe da implementação específica apresentada, que representa apenas uma possível implementação para os padrões da coleção.

4.3 IMPLEMENTAÇÃO DA BIBLIOTECA PARA GERAÇÃO DE INTERFACES

4.3.1 Escopo

O desenvolvimento da biblioteca teve seu escopo limitado da mesma forma que a coleção de padrões; adicionalmente, limitou a plataforma suportada: na sua codificação, foi adotada a plataforma Windows, com Object Pascal e VCL, o *toolkit* do Borland Delphi. Como conseqüência, a biblioteca só pode ser utilizada para programação nessa mesma plataforma.

4.3.2 Modelagem e Implementação

A modelagem da biblioteca definiu as classes do modelo e estabeleceu, com isso, a forma através da qual essas classes relacionavam-se entre si, assim como a forma através da qual o programador poderia utilizá-las. Foi norteada pelos princípios destacados a seguir:

1. *Separação entre o modelo da interface visual e o controle dos dados.*
   Isso foi alcançado através da criação de duas hierarquias de classes

distintas – uma representando as interfaces visuais e outra representando os editores de dados.

2. *A abstração de componentes do modelo da interface visual, de modo a permitir mais de uma implementação concreta.* Nos pontos da biblioteca onde *widgets* concretos seriam necessários, optou-se por versões abstratas de classes *proxy* associadas (o padrão Proxy é descrito em Gamma et al., 1995).
3. *A possibilidade de gerar interfaces automaticamente, sem utilizar o construtor de interfaces.* Alguns dos padrões podem ser gerados de forma automática pela biblioteca. Outros precisam que parte da interface visual seja codificada e implementada através do construtor de interfaces da ferramenta.
4. *A definição de uma interface mínima através da qual o programador usuário da biblioteca pudesse utilizá-la*. Para o uso da biblioteca no modo de geração de interfaces, uma única classe do modelo concentra as configurações e outra oferece as interfaces de execução. No modo programático, duas classes foram criadas para explorar as facilidades de herança visual disponíveis no ambiente.
5. *A criação de uma implementação concreta, pronta para utilização, das classes do modelo*. Tal implementação foi realizada. A biblioteca foi testada em ambiente de desenvolvimento por mais de seis meses; outros testes continuam, já em ambientes de produção.

Definidas as classes e os comportamentos da biblioteca, sua implementação ocorreu sem maiores problemas.

5 TRABALHO DESENVOLVIDO

Os resultados do trabalho desenvolvido serão apresentados em três etapas: primeiramente, a coleção de padrões de interfaces gráficas elaborada especificamente para aplicações orientadas a dados; em seguida, a implementação de uma biblioteca de referência, em linguagem de programação orientada a objetos, para a geração de interfaces gráficas baseadas na coleção de padrões elaborada; finalmente, o desenvolvimento de uma aplicação real com a utilização da biblioteca implementada.

5.1 COLEÇÃO DE PADRÕES DE INTERFACES GRÁFICAS

Uma coleção de padrões de interfaces gráficas foi elaborada, com vinte padrões distribuídos em cinco grupos distintos conforme a tabela 4. Os padrões são apresentados nesta seção de forma descritiva, privilegiando a discussão em torno das motivações para o emprego dos padrões. No apêndice A, utilizou-se outra forma de apresentação, padronizada, de modo a facilitar o uso da coleção nas atividades práticas do profissional de interfaces, assim como permitir a rápida comparação entre os padrões a partir de suas propriedades.

Nas seções que seguem, cada um dos padrões da coleção é descrito e discutido. No apêndice A deste trabalho os padrões são apresentados na sua íntegra, numa forma mais adequada à consulta pelos profissionais de interface, programadores e usuários em geral.

Tabela 4 – Agrupamento dos Padrões de Interfaces Gráficas

Grupo	**Padrão**
Indicador Visual	Identificador
Indicador Visual	Obrigatoriedade
Indicador Visual	Somente-Leitura
Indicador Visual	Foco
Indicador Visual	Zebrado
Controle	Barra de Controles
Controle	Menu Contextual
Acesso a Dados	Lookup Local
Acesso a Dados	Lookup por Delegação
Editor de Dados	Grade
Editor de Dados	Grade Contextualizada
Editor de Dados	Grade de Pesquisa Livre
Editor de Dados	Grade Mestre-Detalhe
Editor de Dados	Grades Alternativas
Editor de Dados	Grade de Pesquisa Avançada
Editor de Dados	Formulário
Editor de Dados	Formulário Segmentado
Editor de Dados	Formulário Mestre-Detalhe
Editor de Dados	Assistente
Saída de Dados	Listagem Simples

5.1.1 Identificador

O primeiro padrão da coleção, Identificador, é um dos mais simples. Quando uma interface está em fase de criação, os diversos elementos na tela precisam, invariavelmente, de alguma forma mínima de explicação acerca de sua função na interface. O objetivo do padrão Identificador é apresentar ao usuário o significado, no modelo conceitual do sistema, de uma porção da interface, através do emprego de um título ou texto explicativo. Em geral, cada identificador refere-se a um único *widget*, mas pode ser utilizado para esclarecer o significado de um conjunto de *widgets* (como uma seção da tela ou grupo de radio) ou parte de um *widget* (como colunas de uma planilha ou visualizador de lista). Por seu papel na interface, identificadores são classificados como Indicadores Visuais.

Identificadores agilizam o uso da interface por parte dos usuários, que passam a reconhecer rapidamente o papel de cada porção da interface; facilitam o aprendizado do uso da interface e sua retenção; a qualidade e consistência das informações preenchidas pelos usuários na interface tendem a aumentar com a

compreensão do significado e do papel de seus diversos elementos no modelo conceitual do sistema. Esse padrão é repetidamente empregado em sistemas de manipulação direta em geral, não sendo específico a sistemas de informação. No entanto, devido ao intenso uso de interfaces baseadas na analogia do formulário, o padrão Somente-Leitura é essencial em sistemas de informação.

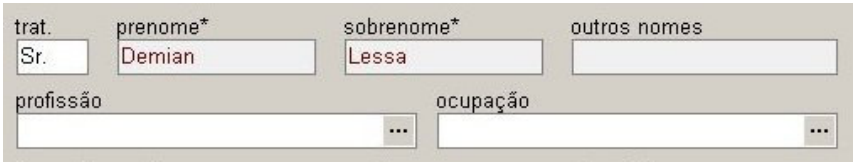


Figura 35 – Exemplo: Identificador, Obrigatoriedade e Somente-Leitura

O fragmento de interface da Figura 35 apresenta parte de um cadastro de pessoa com seis identificadores alinhados acima à esquerda de seis *widgets*. Cada identificador esclarece o significado das partes da informação: “trat.” determina a forma de tratamento empregada na comunicação com a pessoa; “prenome”, “sobrenome” e “outros nomes” identificam as partes que compõem o nome da pessoa; “profissão” e “ocupação” esclarecem a formação da pessoa e a atividade com a qual ela está atualmente envolvida. O padrão Identificador é suportado nativamente pela maioria dos ambientes de desenvolvimento.

5.1.2 Obrigatoriedade

O padrão Obrigatoriedade é outro bem simples. Durante a análise, as diversas informações do modelo são classificadas como obrigatórias ou opcionais. Na interface do sistema, o usuário precisa identificar as partes da informação de preenchimento obrigatório daquelas opcionais. Através do emprego de alguma forma de indicação visual, nos próprios *widgets* que apresentam a informação ou nos identificadores associados, essa obrigatoriedade pode ser destacada. Uma forma conhecida de destaque emprega um caractere '*' próximo ao identificador de cada *widget* cujo preenchimento seja obrigatório.

O padrão Obrigatoriedade agiliza a interação do usuário com a interface, pois permite a rápida distinção entre informações obrigatórias e opcionais, resultando numa menor quantidade de erros pela omissão de informação essencial. Esse padrão é repetidamente empregado na maioria das interfaces dos sistemas que empregam a analogia do formulário para a coleta de informações. Além de sistemas de informação, sistemas *web* empregam esse padrão repetidamente.

No fragmento de interface da Figura 35, os dois únicos conteúdos obrigatórios são rapidamente identificados pois seus títulos estão decorados com um asterisco: "prenome*" e "sobrenome*". O padrão Obrigatoriedade é suportado geralmente através da programação direta da interface visual.

5.1.3 Somente-Leitura

O padrão Somente-Leitura também é simples. Durante o uso de um sistema, em certas ocasiões surge a necessidade de proteger informações de alterações por parte dos usuários. Na interface do sistema, o usuário precisa identificar as partes da informação que podem ser alteradas. Através do emprego de alguma forma de indicação visual, nos próprios *widgets* que apresentam a informação, os controles cuja informação não pode ser alterada podem ser destacados. Uma forma conhecida de destaque emprega a combinação de uma cor de fundo e uma cor de texto características para um *widget* somente-leitura. Em geral, o efeito visual desejado é que o *widget* pareça desabilitado, estático.

O padrão Somente-Leitura agiliza a interação do usuário com a interface, pois permite a rápida distinção entre informações ativas e somente-leitura. Assim, os usuários não tentarão interagir com os *widgets* somente-leitura. Esse padrão é repetidamente empregado em sistemas de manipulação direta em geral, não sendo específico a sistemas de informação. No entanto, devido ao intenso uso de interfaces baseadas na analogia do formulário, o padrão Somente-Leitura é essencial em sistemas de informação.

No fragmento de interface da Figura 35, os *widgets* associados aos identificadores "prenome*", "sobrenome*" e "outros nomes" são somente-leitura. Percebe-se esse fato rapidamente pelas cores de fundo (cinza claro) e cor do texto

(marrom) desses controles, em contraste aos demais *widgets*, que empregam branco e preto como cores de fundo e de texto, respectivamente. A implementação de referência oferece serviços para facilitar o emprego do padrão Somente-Leitura.

5.1.4 Foco

Durante o uso de um sistema, o usuário precisa identificar prontamente qual o controle ativo, isto é, recebe as entradas do teclado. Através do emprego de alguma forma de indicação visual, nos próprios *widgets* que apresentam a informação, o controle ativo pode ser destacado dos demais. Uma forma adequada de destaque emprega ou utiliza cores de fundo e de texto diferenciadas para identificar o *widget* com o foco. Quando o controle recebe o Foco, ele altera suas cores para as cores características do Foco. Quando perde o Foco, as cores são novamente redefinidas para os valores originais. Com o uso sistemático do padrão Foco numa interface, sempre haverá um controle visualmente identificador como o Foco. *Widgets* que empregam o padrão Somente-Leitura jamais devem receber o foco do teclado.

O padrão Foco agiliza a interação do usuário com a interface, pois permite a rápida identificação do controle ativo, evitando o preenchimento equivocado de informações. Adicionalmente, quando o usuário interromper o uso de um sistema por alguns segundos, no seu retorno, ele não precisará recorrer à memória ou procurar pelo cursor do texto nos vários controles da interface, pois o controle ativo estará indicado na tela. Esse padrão é empregado em algumas interfaces de sistemas de manipulação direta, mas seu uso em sistemas de informação não é tão comum como os demais padrões apresentados.

No fragmento de interface da Figura 36, o *widget* ativo destaca-se visualmente dos demais pelo emprego de uma cor de fundo diferenciada. A implementação de referência suporta automaticamente o padrão Foco.

Figura 36 – Exemplo: Foco e Lookup Local

5.1.5 Alternância de Cores

Durante o uso de um sistema, quando o usuário é apresentado a um conjunto tabular de informações, acompanhar os valores de cada uma das linhas de informação, sem equívocos na leitura, requer algum esforço. Para facilitar esse acompanhamento e evitar os equívocos em sua leitura, um auxílio visual baseado na Alternância de Cores pode ser empregado. O princípio básico desse padrão é alternar as cores de fundo em cada linha de informação. Utilizando cores contrastantes, uma linha pode ser facilmente distinguida das linhas anterior e posterior e, assim, as partes da informação podem ser lidas ao longo da linha da planilha mais facilmente. Esse padrão não deve ser utilizado em conjunto com o padrão Somente-Leitura, pois a combinação visual dos dois padrões resultaria na maior dificuldade no uso da interface do que em benefícios adicionais.

O emprego da Alternância de Cores confere maior agilidade à consulta de informações dispostas de forma tabular. Facilita a distinção entre as linhas e, em cada linha, o acompanhamento das partes da informação. Esse padrão é empregado em diversas interfaces de sistemas de manipulação direta. Em sistemas de informação, é comum na apresentação de informações sob a forma tabular, para edição de dados ou na forma de listagens.

A Figura 37 ilustra bem as características de uma interface com alternância de cores. Cada linha da tabela possui cor de fundo diferente da cor de fundo das linhas vizinhas. Ao mesmo tempo, a leitura das informações de cada linha é facilitada pelo contraste de cores, e os dados de uma mesma linha são facilmente lidos desde a primeira até a última coluna. A implementação de referência suporta

automaticamente o padrão Alternância de Cores através de outros padrões- a Listagem Simples e todos os padrões baseados na Grade.

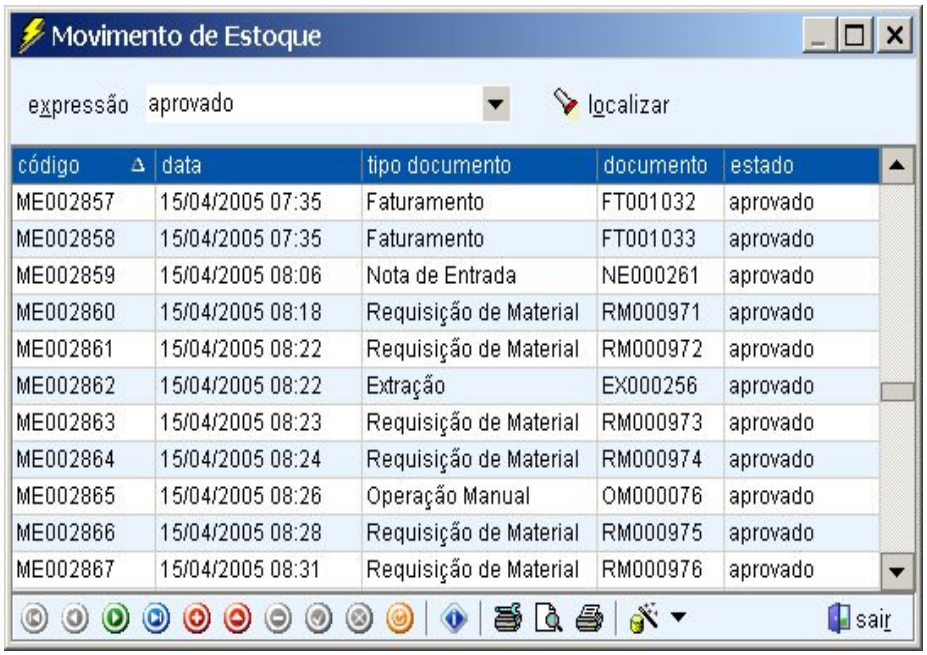


Figura 6 – Exemplo: Alternância de Cores

5.1.6 Barra de Controles

Num sistema de informações, os usuários estão tipicamente envolvidos em tarefas que redundam em um conjunto bem estabelecidos de operações sobre os dados: além das operações CRUD (do acrônimo inglês *Create*, *Read*, *Update and Delete*, refere-se a criação, leitura, atualização e exclusão de registros no banco de dados), a navegação, exportação, impressão e consulta de meta-dados. Para oferecer rápido acesso a esse tipo de operação, é comum empregar uma Barra de Controles. A Barra é composta de uma série de *widgets*, geralmente botões com imagens dispostos lado a lado, que representam as diversas operações suportadas pelo sistema. Além do acesso através da interação visual, a Barra deve permitir o uso de teclas de atalho para acionar as operações, de modo que usuários mais adaptados ao teclado possam manejar o sistema através da forma com a qual estão mais adaptados. O uso de dicas contextuais para informar a operação implementada por cada *widget* é altamente recomendável, pois permite a rápida identificação de sua operação. Finalmente, sugere-se que a Barra diferencie as operações

disponíveis e indisponíveis em cada contexto e estado do sistema, através do bloqueio aos botões da Barra.

O uso da Barra de Controles de forma sistemática nas telas do sistema de informações permite um alto grau de consistência no acesso às operações comuns do sistema. O emprego de imagens associadas às operações facilita o aprendizado e a retenção por parte dos usuários. Finalmente, o rápido acesso às operações através da Barra contribui para a maior agilidade na interação do usuário com o sistema. Esse padrão é repetidamente empregado em sistemas de manipulação direta em geral, não sendo específico a sistemas de informação. Em sistemas de informação, é amplamente empregado como ferramenta para controle e navegação dos dados.

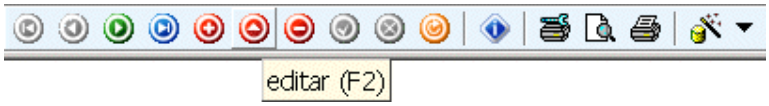


Figura 38 – Exemplo: Barra de Controles

A Figura 38 apresenta uma barra de controles com diversos *widgets*. Os quatro primeiros botões a partir da esquerda representam as operações de navegação; os demais representam, respectivamente, as operações: incluir, editar, excluir, salvar, cancelar (uma edição ou inclusão), recarregar, exibir informações (meta-dados), configurar impressão, visualizar impressão, imprimir e exportar. A Barra apresenta alguns botões em branco e tons de cinza, indicando sua indisponibilidade momentânea, em função do contexto e estado do sistema. A dica contextual é bem identificada através do retângulo colorido “editar (F2)”. Ao passar o *mouse* sobre um determinado *widget* da barra, o usuário é apresentado à operação implementada pelo *widget* e sua tecla de atalho associada. A dica é automaticamente exibida e ocultada, sem a necessidade de intervenção do usuário. A implementação de referência suporta automaticamente o padrão Barra de Controles nos editores da dos, exceto no Assistente.

5.1.7 Menu Contextual

Uma interface visual deve oferecer acesso o mais direto e simples quanto

possível às operações disponíveis- aquelas que serão repetidamente realizadas pela maioria dos usuários. Usuários avançados, no entanto, precisam ser considerados. Em geral, esse tipo de usuário requer acesso a diversas operações avançadas como, por exemplo, o ajuste da forma como as interfaces são arranjadas e exibidas na tela. Para não interferir com a operação padrão de uma interface, o acesso a essas operações avançadas é disponibilizado através de um Menu Contextual. Esse tipo de menu é acionado tanto pelo teclado quanto pelo *mouse* e suas entradas são definidas dinamicamente, através do estado da interface no momento em que o menu é acionado. Como medida preventiva de erros, os Menus Contextuais devem apresentar aos usuários apenas as opções que possam ser realmente executadas no contexto e no estado em que o sistema se encontra.

O uso do Menu Contextual confere maior flexibilidade à interface, permitindo que usuários realizem operações avançados de forma rápida sem, no entanto, interferir na forma padrão da operação disponibilizada aos usuários menos avançados. Esse padrão é amplamente empregado em interfaces de sistemas de manipulação direta, mas seu uso em sistemas de informação não é tão comum como os demais padrões apresentados.

Figura 39 – Exemplo: Menu Contextual

A Figura 39 apresenta um menu contextual que permite selecionar as colunas de uma planilha que devem ficar visíveis numa interface. O menu é exibido quando o botão direito do *mouse* é pressionado sobre a planilha. A lista de colunas exibida

como entradas no menu é montada dinamicamente, a partir da própria planilha-colunas visíveis são exibidas com uma marca e colunas não visíveis são exibidas sem a marca. Além da seleção individual das colunas visíveis, o usuário pode selecionar que todas as colunas fiquem visíveis ao mesmo tempo. Finalmente, a operação “ajustar grade” permite que a planilha mantenha as larguras das colunas de forma automática, de modo que todas as colunas fiquem visíveis sem a necessidade do emprego de uma barra de rolagem. A implementação de referência suporta automaticamente o padrão Menu Contextual através dos padrões baseados na Grade.

5.1.8 Lookup Local

Quando um usuário precisa expressar um valor na interface através de uma lista de possíveis valores, se emprega um padrão baseado em *lookup* (pesquisa). O Lookup Local é utilizado quando a lista dos possíveis valores é pequena o suficiente para que o usuário possa navegar sequencialmente pelos vários elementos e encontrar rapidamente aquele procurado. Em geral, esse tipo de lista é gerado a partir de valores previamente conhecidos ou de uma fonte externa de dados, como uma tabela de um banco de dados relacional. Os valores são apresentados na interface através de um *widget* especializado capaz de permitir a rápida pesquisa por elementos na lista. Normalmente, esses widgets são variações de *combo box* (caixa de seleção) ou *list box* (lista de seleção).

O uso do Lookup Local permite a rápida localização de uma informação desejada em conjuntos de dados limitados, conferindo maior agilidade no preenchimento de dados na interface. Esse padrão é repetidamente empregado em sistemas de manipulação direta em geral, não sendo específico a sistemas de informação. Em sistemas de informação, é amplamente empregado para o preenchimento de valores de domínios limitados e chaves estrangeiras.

A Figura 36 apresenta uma implementação de Lookup Local através da caixa de seleção. A lista de possíveis valores para o campo “sexo” é visível na caixa de seleção: “masculino”, “feminino” e “desconhecido”. O padrão Lookup Local é suportado nativamente pela maioria dos ambientes de desenvolvimento.

5.1.9 Lookup por Delegação

O Lookup por Delegação é uma variação de *lookup* utilizado quando a lista dos possíveis valores é sabidamente longa ou cresce significativa com o tempo. O tamanho do conjunto de possíveis valores inviabiliza a navegação seqüencial pelos valores, e outra estratégia de pesquisa, diferente daquela empregada pelo Lookup Local, é utilizada. Em geral, os possíveis valores desse tipo de lista estão armazenados em uma fonte externa de dados, como uma tabela de um banco de dados relacional. Somente o valor atual do *lookup* é exibido na interface, através de um *widget* adequado. Esse *widget* também oferece um mecanismo para acionar a pesquisa pelo valor procurado, que é delegada a uma interface externa, tipicamente uma Grade de Pesquisa Simples ou Avançada.

O uso do Lookup por Delegação permite a rápida localização de uma informação desejada em conjuntos de dados extensos, conferindo maior agilidade no preenchimento de dados na interface. Com o emprego da delegação, evita-se a carga de uma grande quantidade de dados num controle de Lookup Local, que resultaria numa interface de lenta inicialização e operação. Esse padrão é específico a sistemas de informação, sendo empregado para o preenchimento de valores de chaves estrangeiras cuja tabela de origem pode conter uma quantidade grande de registros.

As Figuras 40, 41 e 42, a seguir, apresentam as sucessivas etapas do uso de um Lookup por Delegação: primeiramente, é acionada a pesquisa através de um botão no *widget* de exibição da cidade do bairro “Pernambués”; a tela de pesquisa é chamada e são procuradas as cidades começadas com “Sa”; “Salvador” é selecionada, através do botão “Selecionar” na parte inferior da tela, e passada ao controle de *lookup*, que então exibe o resultado da pesquisa. A implementação de referência suporta automaticamente o padrão Lookup por Delegação através de uma classe específica de *lookup* que coordena a exibição dos dados e sua pesquisa externa, através do emprego do padrão Grade de Pesquisa Simples.

Figura 40 – Exemplo: Lookup por Delegação-Acionamento da Pesquisa

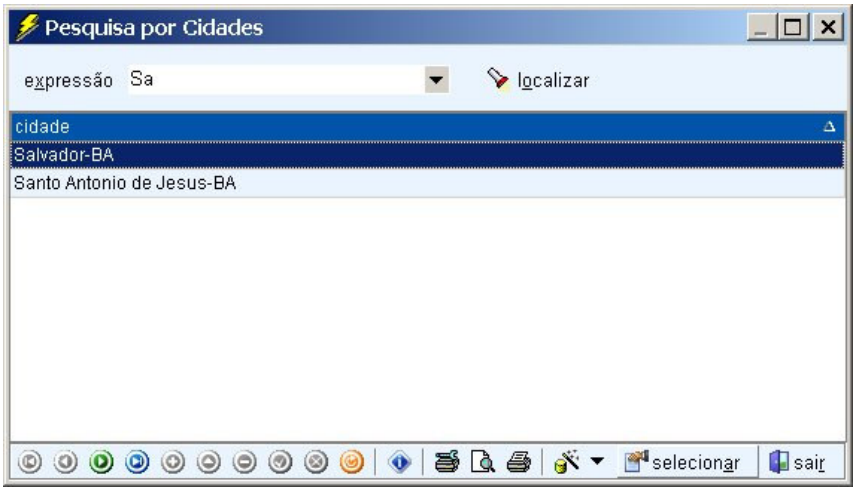


Figura 41 – Exemplo: Lookup por Delegação-Pesquisa e Seleção

Figura 42 – Exemplo: Lookup por Delegação, Pesquisa Concluída

5.1.10 Grade

Em sistemas de informação, o usuário precisa acessar conjuntos de informação de estrutura simples e homogênea, tipicamente representados por tabelas de domínios. O acesso à informação se dá através de suas partes, isto é, seus campos. A quantidade de campos é pequena e o volume de informações é reduzido. O usuário deve ter uma visão geral de cada informação, assim como do conjunto total de informações. Para isso, é utilizado o padrão Grade.

A Grade apresenta dados de forma tabular, onde informações individuais são dispostas em linhas e colunas representam cada parte da informação exibida nas células da planilha. No uso da Grade, partes mais importantes da informação devem ter maior destaque (precedência ou outro destaque visual), enquanto partes conceitualmente associadas devem ser agrupadas (sucessão ou outro agrupamento visual). Geralmente, a Grade é utilizada em conjunto com outros padrões: com Identificadores, para permitir a identificação dos dados de cada coluna da Grade; com uma Barra de Controles, para realizar as diversas operações de manipulação de seus dados; com um Menu Contextual, para definir o estilo visual da Grade e as colunas visíveis; com Lookups Local e por Delegação, para manipular dados de pesquisa; com Alternância de Cores, para facilitar a leitura das informações das Grades não editáveis (cujos dados são somente para leitura). A Grade também é utilizada em conjunto com outras Grades, para compor padrões mais complexos. Outros padrões são definidos a partir da Grade e gozam da maioria de suas características.

O uso da Grade permite a manipulação rápida e prática de conjuntos de dados de tamanho limitado, tanto em quantidade quanto em partes. Esse padrão é amplamente empregado em sistemas de informação para o preenchimento de domínios e tabelas para as quais se espera uma quantidade limitada de registros.

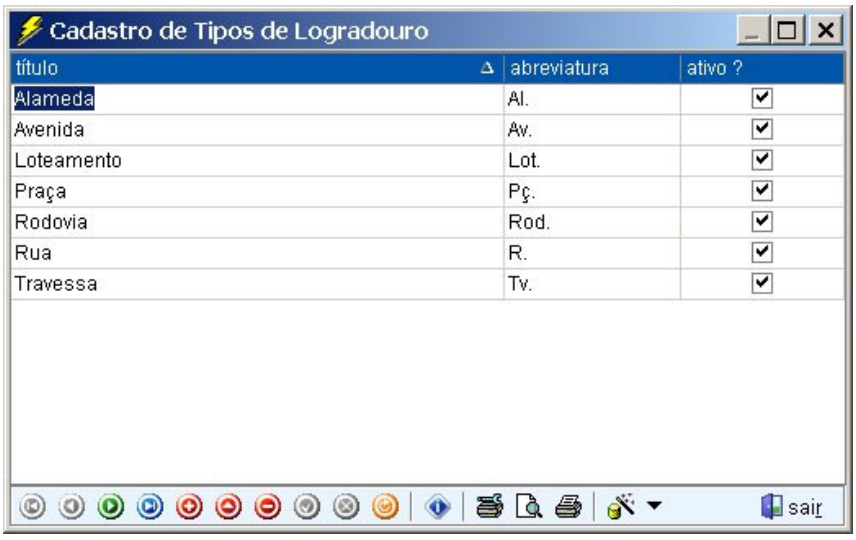


Figura 43 – Exemplo: Grade

A Figura 43 apresenta uma implementação característica de Grade. Os dados estão dispostos claramente de forma tabular, em linhas e colunas. As colunas identificam as partes da informação através de Identificadores no cabeçalho da planilha. Uma Barra de Controles é empregada para permitir a navegação e manipulação dos dados. A implementação de referência suporta o padrão Grade automaticamente.

5.1.11 Grade Contextualizada

A Grade Contextualizada é um padrão baseado na Grade, mas com uma diferença fundamental: o volume e a natureza dos dados manipulados. Na Grade Contextualizada, os dados estão naturalmente bem agrupados em torno de uma ou mais de suas partes. O volume total de dados pode ser grande, mas quando considerados a partir dos agrupamentos individuais, são suficientemente limitados para que se possa empregar o padrão Grade. Assim, ao invés de apresentar os dados em sua totalidade, a Grade Contextualizada limita os dados através de uma filtragem prévia. Em geral, essa filtragem ocorre através do emprego de um ou mais *widgets* que implementam o padrão Lookup Local. Variações podem utilizar outros controles para estabelecer o contexto.

O uso da Grade Contextualizada permite explorar as vantagens da Grade em

conjuntos de dados que, sem a filtragem prévia, seriam inadequados para o uso de um modelo de interface baseado na Grade. Com o uso da Grade Contextualizada, a visão dos dados fica fragmentada nos diversos contextos e, por isso, o analista deve certificar-se de que essa visão não traga problemas significativos à operação do sistema ou à interpretação de seus dados. Esse padrão é extremamente útil em sistemas de informação para o preenchimento de tabelas para as quais se espera uma quantidade significativa de registros, mas cujos dados se permitem contextualizar sem maiores implicações.

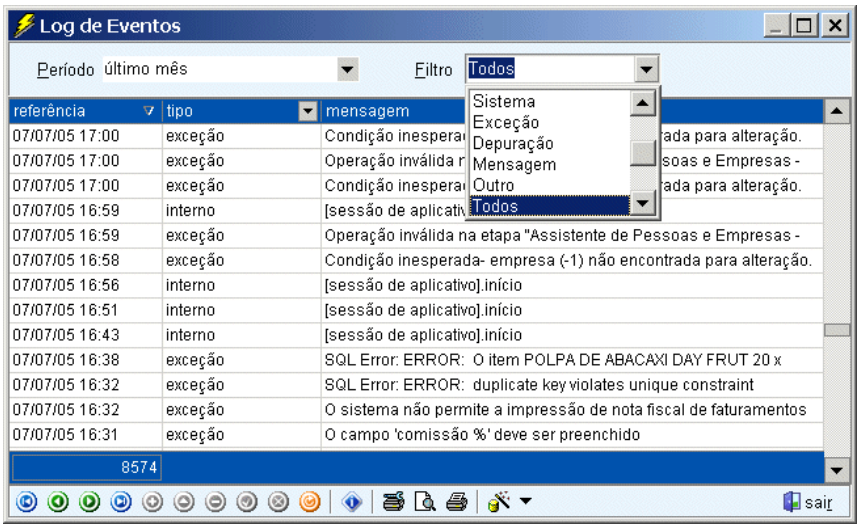


Figura 44 – Exemplo: Grade Contextualizada

A Figura 44 apresenta uma implementação típica de Grade Contextualizada com informações acerca dos eventos de um sistema. Os controles de *lookup* na barra superior da tela permitem estabelecer o contexto de visualização dos eventos: o período (último mês) e o tipo de evento (todos). A implementação de referência suporta parcialmente a Grade Contextualizada. Além de alguma programação para definir novos comportamentos, é preciso especificar a interface visual no construtor de interfaces a partir de uma subclasse concreta da classe TclGridForm do modelo (herança visual).

5.1.12 Grade de Pesquisa Livre

A Grade de Pesquisa Livre é um padrão análogo à Grade Contextualizada. A diferença fundamental entre os dois padrões está na forma através da qual as informações são previamente filtradas. Como, na Grade Contextualizada, as informações estão naturalmente agrupadas, o emprego de um ou mais filtros bem definidos é possível. Na Grade de Pesquisa Livre, no entanto, não há agrupamento natural da informação e sua contextualização é definida através de um texto livre de pesquisa digitado pelo usuário. Como essa filtragem é definida pelo usuário, o resultado de uma pesquisa pode retornar mais registros do que o adequado para uso com a Grade. Em tais casos, cabe à implementação do modelo oferecer uma alternativa- sugerir ao usuário uma pesquisa mais específica, pedir sua confirmação para carregar o volume de dados ou cancelar a pesquisa. Um aspecto relevante na implementação das interfaces concretas é garantir que os textos de pesquisa filtrem os dados a partir de suas principais partes, levando em conta o modelo conceitual do sistema.

O uso da Grade de Pesquisa Livre permite explorar as vantagens da Grade Contextualizada, sem as limitações impostas pela definição prévia das dimensões que podem ser filtradas. Com o uso da Grade de Pesquisa Livre, a visão dos dados fica fragmentada de forma mais natural do que com o uso da Grade Contextualizada; no entanto, o analista deve também certificar-se de que essa visão não traga problemas significativos à operação do sistema ou à interpretação de seus dados. Esse padrão é extremamente útil nas partes dos sistemas de informação baseadas no paradigma localizar-editar, onde as tarefas do sistema são precedidas da localização prévia das entidades com as quais se deseja trabalhar.

Figura 45 – Exemplo: Grade de Pesquisa Simples (1 de 2)

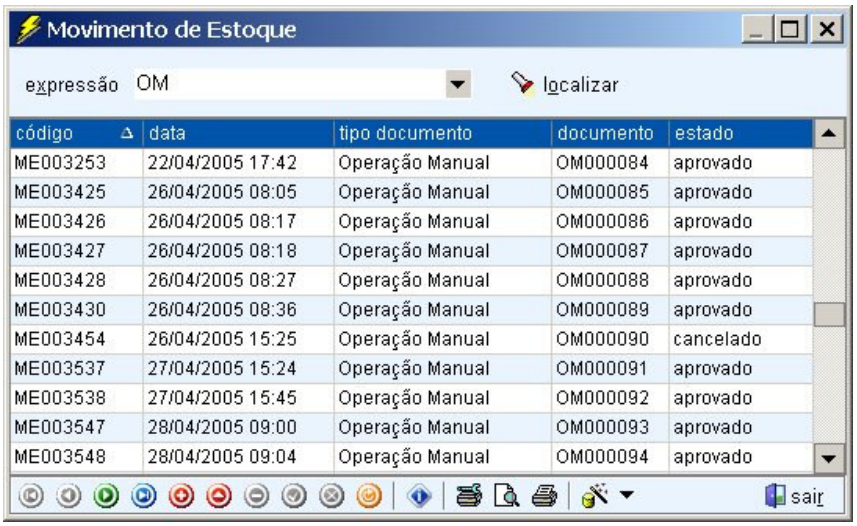


Figura 46 – Exemplo: Grade de Pesquisa Simples (2 de 2)

As Figuras 45 e 46 ilustram o funcionamento da Grade de Pesquisa Livre. A expressão de pesquisa livre é digitada num *widget* disponível na barra superior. Nas figuras, estão apresentadas duas pesquisas de registros de movimento de estoque com as expressões de pesquisa “aprovado” e “OM”. Na primeira pesquisa, o filtro ocorreu no campo “estado” e foram retornados apenas os registros de movimentos aprovados; na segunda, o filtro ocorreu no campo “documento” e foram retornados apenas os movimentos de estoque gerados a partir de operações manuais

(documento começando com “OM”). Através da implementação concreta, a expressão de pesquisa pôde ser adequadamente utilizada para retornar ao usuário os resultados que ele esperava, de acordo com o modelo conceitual. A implementação de referência suporta o padrão Grade de Pesquisa Simples automaticamente.

5.1.13 Grade Mestre-Detalhe

Acessar dados de uma só Grade nem sempre é suficiente. Em certas ocasiões, o usuário precisa acessar, simultaneamente, dados hierarquicamente dependentes de duas ou mais Grades. A Grade Mestre-Detalhe deve oferecer uma visão dos dados de cada Grade e de sua relação de dependência. Basicamente, o padrão define a composição de uma interface a partir de Grades associadas através de relações mestre-detalhe entre seus dados. Na sua forma mais tradicional, as Grades são dispostas lado a lado, horizontal ou verticalmente. A posição de cada Grade determina sua relação na hierarquia. Quando o usuário seleciona uma Grade, as demais mudam seus estados para indicar que estão indisponíveis (somente leitura) para edição e filtram seus dados para o contexto adequado (dado pelo registro mestre, na relação mestre-detalhe dos dados).

O uso da Grade Mestre-Detalhe permite a manipulação de dados relacionados através da mesma interface. Com o uso desse padrão, o usuário tem não apenas a visão geral de cada informação individual, mas também de suas relações de dependência. O resultado é uma maior identificação com o modelo conceitual e a maior agilidade na manipulação de dados associados que não precisam ser acessados em telas individuais ou editados com o auxílio de *lookups* para as chaves estrangeiras. Esse padrão é extremamente útil em sistemas de informação para o preenchimento de tabelas dependentes para as quais se espera uma quantidade limitada de registros.

Figura 477– Exemplo: Grade Mestre-Detalhe

A Figura 47 apresenta uma implementação típica de Grade Mestre-Detalhe com as Grades dependentes dispostas verticalmente na tela. Ao selecionar uma "avaliação" na primeira Grade (Intensivo), o usuário passa a ver apenas as "partes" disponíveis para essa avaliação e, ao selecionar uma dessas partes (Part B), apenas as "notas" associadas a essa parte são exibidas. Apenas a Grade do meio está ativa. As demais, visualmente diferenciadas com cor de fundo cinza, estão indisponíveis para edição. Para ativar outra Grade, basta que o usuário a selecione com o auxílio do *mouse* ou do teclado. A partir de então, poderá editar seus dados normalmente, como se fosse uma Grade independente. A Barra de Controle está sempre corretamente contextualizada para a Grade selecionada. A implementação de referência suporta parcialmente a Grade Mestre-Detalhe. Além de alguma programação para definir novos comportamentos é preciso especificar a interface visual no construtor de interfaces a partir de uma subclasse concreta da classe TclGridForm do modelo (herança visual).

5.1.14 Grades Alternativas

Em algumas situações, o usuário precisa acessar, através de uma mesma interface, conjuntos de informações não relacionadas. O padrão Grades Alternativas permite que cada informação seja acessada independentemente através de uma

Grade, ao mesmo tempo em que permite ao usuário a fácil e rápida alternância entre as diferentes Grades. Assim como a Grade Mestre-Detalhe, trata-se da composição de uma interface a partir de diversas Grades. Contudo, os dados das Grades não são relacionados entre si. Na sua forma mais tradicional, um *widget* é utilizado como um índice visual através do qual a seleção da Grade ativa é realizada. Na ativação de uma Grade, todos os demais elementos da interface são atualizados de modo a garantir que o contexto de trabalho refere-se à Grade ativa.

O uso das Grades Alternativas permite a manipulação de dados não relacionados semanticamente através da mesma interface. Com o uso desse padrão, o usuário tem fácil acesso a um conjunto de Grades cuja ocorre em conjunto. O resultado é a maior agilidade na operação das diversas Grades. Esse padrão é extremamente útil em sistemas de informação com muitas tabelas pequenas, tipicamente tabelas de domínios, para as quais se espera uma quantidade limitada de registros. Em geral, a alimentação e manutenção dessas tabelas é realizada ao mesmo tempo, por um mesmo grupo de usuários.

A Figura 48 apresenta uma implementação típica de Grades Alternativas com um índice baseado numa lista de textos e imagens. Ao selecionar um item no índice, a Grade correspondente é carregada na área à direita da tela. No exemplo, a Grade ativa é a de “bancos”. Uma outra forma de implementar a alternância poderia empregar um *widget* de controle de páginas como um *tab control* (guia de abas). A implementação de referência suporta parcialmente as Grades Alternativas. Além de alguma programação para definir novos comportamentos, é preciso especificar a interface visual no construtor de interfaces a partir de uma subclasse concreta da classe TclGridForm do modelo (herança visual).

Figura 48 – Exemplo: Grades Alternativas

5.1.15 Grade de Pesquisa Avançada

A Grade de Pesquisa Avançada é muito semelhante à Grade de Pesquisa Simples. A diferença fundamental entre esses padrões está na forma através da qual as informações são previamente filtradas. Na Grade de Pesquisa Simples, a contextualização é definida através de um texto livre de pesquisa digitado pelo usuário. Na Grade de Pesquisa Avançada, os filtros são definidos dinamicamente pelo usuário, através de uma interface com um construtor visual de consultas. Através desse construtor, o usuário define uma ou mais cláusulas de consulta combinando uma parte da informação, um operador e seus argumentos. Pesquisas arbitrariamente complexas podem ser construídas a partir da concatenação de várias cláusulas de pesquisa. Somente na implementação do aplicativo é que são definidas de fato as partes da informação que poderão ser utilizadas para a construção das consultas. Um aspecto relevante na implementação das interfaces concretas de pesquisa é garantir que o construtor de consultas disponibilize ao usuário as principais partes da informação, levando em conta o modelo conceitual do sistema.

O uso da Grade de Pesquisa Avançada permite explorar as vantagens da Grade de Pesquisa Simples, de forma menos limitada, já que os critérios das pesquisas são definidos pelo usuário, ao invés do programador. Com o uso da Grade de Pesquisa Avançada, a visão dos dados fica fragmentada de uma forma

pouco natural, já que a maioria das pesquisas tende a ser extremamente específica. O analista deve certificar-se de que essa a Grade de Pesquisa Avançada esteja disponível apenas aos usuários com entendimento técnico e do modelo conceitual suficientes para sua operação e interpretação de seus resultados. Esse padrão é útil para gerar consultas *ad hoc* (de uso pontual, em situações específicas) e para usuários avançados que realizam a extração e depuração dos dados nos sistemas de informação.

A Figura 49 ilustra uma implementação de Grade de Pesquisa Avançada com um construtor de consultas na parte superior da tela. A primeira etapa ao utilizar a Pesquisa Avançada do exemplo é definir a origem dos dados através do *widget* “origem”, na parte inferior da Barra de Controles. Então, o usuário volta sua atenção para o construtor de consultas, na parte superior da tela. Ele seleciona as partes da informação, os critérios e os parâmetros, um a um, até que sua consulta esteja completa. No exemplo, o usuário definiu o último critério de pesquisa como: “valor efetivo entre 250 e 450”. Esse critério faz parte de um conjunto maior de critérios, visível na caixa de texto maior, através de um fragmento de comandos SQL gerado automaticamente pelo construtor. Os dados na Grade mostram o resultado da execução da consulta que, por rápida inspeção, respeitam todos os critérios de pesquisa definidos. A implementação de referência não suporta a Grade de Pesquisa Avançada.

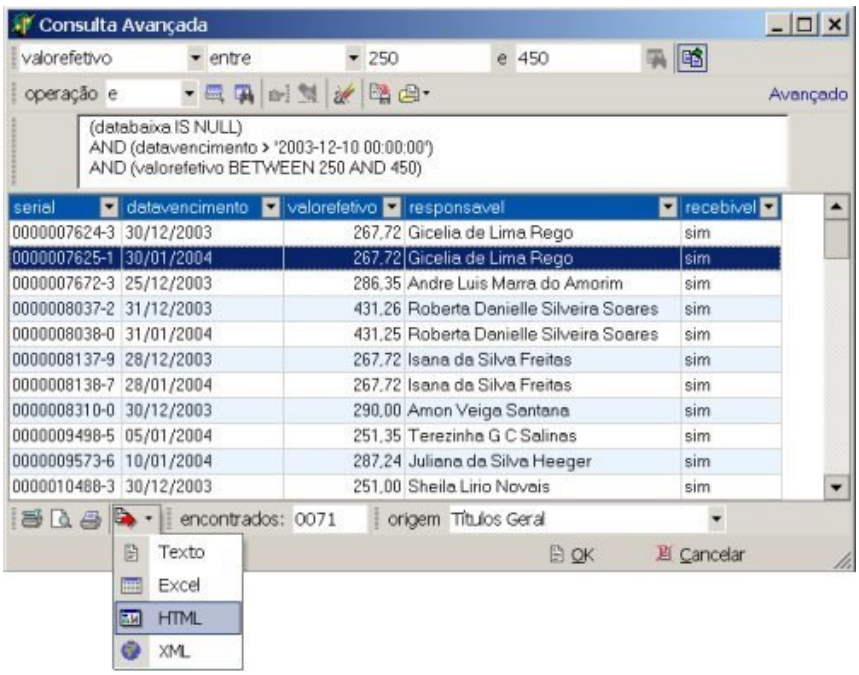


Figura 49 – Exemplo: Grade de Pesquisa Avançada

5.1.16 Formulário

O Formulário é o padrão através do qual os usuários acessam informações de complexidades moderada e alta. A informação é apresentada no seu todo, através de cada uma de suas partes. Em geral, um Identificador e um *widget* específico são utilizados para apresentar cada parte da informação; partes da informação podem ser agrupadas de modo a destacar sua importância das demais. Assim como na Grade, partes conceitualmente mais importantes devem ter maior destaque (precedência ou outro destaque visual), enquanto partes conceitualmente associadas devem ser agrupadas (sucessão ou outro agrupamento visual). Geralmente, o Formulário é utilizado em conjunto com outros padrões: com Identificadores, para permitir a identificação dos dados de cada parte da informação; com uma Barra de Controles, para realizar as diversas operações de manipulação de seus dados; com Lookups Local e por Delegação, para manipular dados de pesquisa. O Formulário Mestre-Detalhe é utilizado em conjunto com Grades, para compor um padrão mais complexo. O Formulário Segmentado é definido a partir do Formulário e goza da maioria de suas características. O Assistente é um padrão

complexo que pode ser interpretado como o encadeamento de uma série de formulários individuais, dependentes entre si, coordenados para a realização de uma tarefa.

O uso do Formulário permite a manipulação rápida e prática de uma única informação complexa. Além disso, expõe os usuários à complexidade da informação e assim o aproxima do modelo conceitual do sistema. Esse padrão é amplamente empregado em sistemas de informação para o preenchimento de informações complexas que vão desde cadastros de entidades a formulários de natureza comercial, financeira, administrativa, etc.

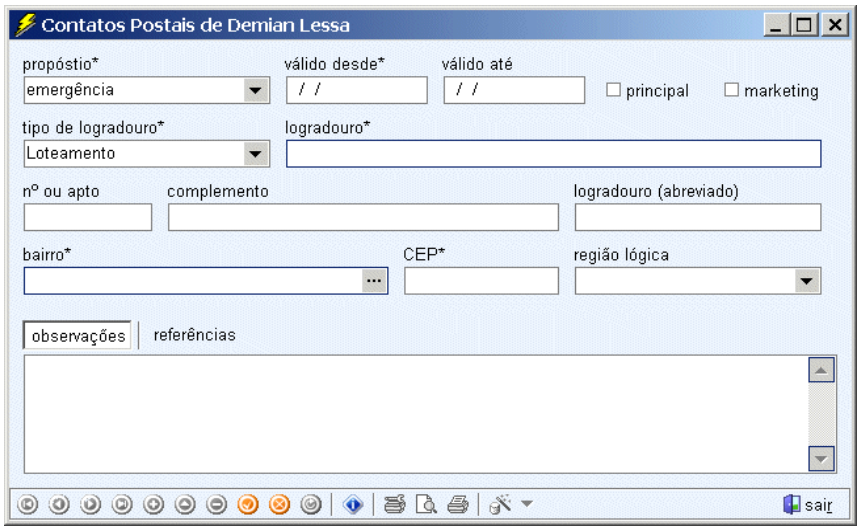


Figura 50 – Exemplo: Formulário

A Figura 50 apresenta uma implementação de Formulário. Os diversos *widgets* da interface são identificados através de títulos. Os dados estão distribuídos na interface de forma a associar as partes relacionadas da informação. Por exemplo: “tipo de logradouro” e “logradouro”; “válido desde” e “válido até”. Uma Barra de Controles é empregada para permitir a manipulação dos dados. A implementação de referência suporta parcialmente os Formulários. Além de alguma programação para definir novos comportamentos, é preciso especificar a interface visual no construtor de interfaces a partir de uma subclasse concreta da classe TclSimpleForm do modelo (herança visual).

5.1.17 Formulário Segmentado

O Formulário Segmentado é uma variação do Formulário na qual a quantidade de partes da informação é grande demais para que seja possível exibi-las todas numa mesma área. A interface é dividida em regiões lógicas que são acessadas uma por vez. A seleção de um segmento da interface apresenta apenas os *widgets* associados àquele segmento, ocultando todos os demais. Formas conhecidas de implementar a segmentação incluem o emprego de *widgets* como controles paginados (*page control*).

O uso do Formulário Segmentado mantém os benefícios do Formulário e permite a manipulação informação ainda mais complexa. No entanto, a necessidade de segmentação dificulta a visão da informação como um todo. Esse padrão é empregado de forma quase tão freqüente quanto o Formulário.

As Figuras 51, 52 e 53 apresentam três segmentos de um típico Formulário Segmentado. Cada segmento agrupa conjuntos relacionados de dados. Em cada segmento, os dados são agrupados de forma ainda mais específica, através do uso dos Identificadores em azul escuro. Os segmentos são dispostos em ordem de importância, estendendo a regra para a arrumação das partes das informações no Formulário. A implementação de referência suporta parcialmente os Formulários Segmentados. Além de alguma programação para definir novos comportamentos, é preciso especificar a interface visual no construtor de interfaces a partir de uma subclasse concreta da classe TclSimpleForm do modelo (herança visual).

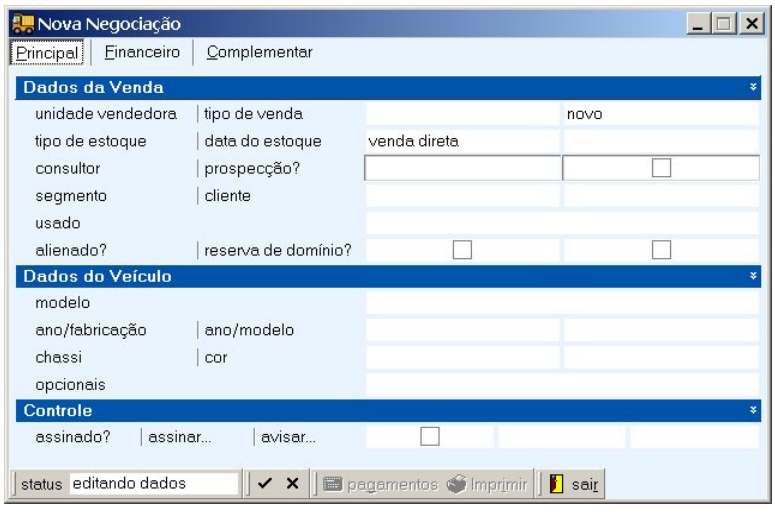


Figura 51 – Exemplo: Formulário Segmentado (1 de 3)

Figura 52 – Exemplo: Formulário Segmentado (2 de 3)

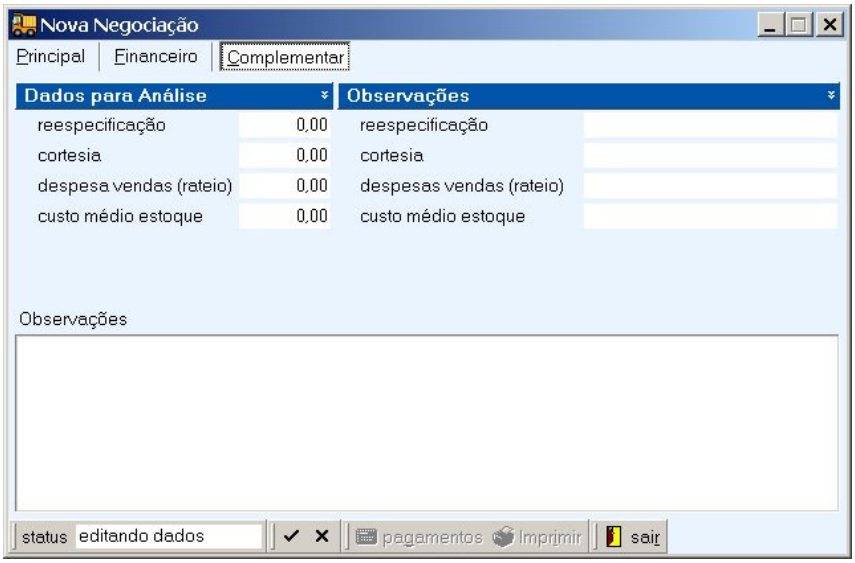


Figura 53 – Exemplo: Formulário Segmentado (3 de 3)

5.1.18 Formulário Mestre-Detalhe

Acessar dados de um Formulário nem sempre é suficiente. Em certas ocasiões, o usuário precisa acessar, simultaneamente, as informações do Formulário e outros conjuntos de informação dependente. O Formulário Mestre-

Detalhe resolve esse problema oferecendo uma visão dos dados do Formulário combinada com suas informações dependentes através do emprego da Grade. Basicamente, o padrão define a composição de uma interface a partir de Grades dependentes dispostas na área de um Formulário. Na sua forma mais tradicional, as Grades são dispostas na parte inferior da interface. Quando houver mais de uma Grade dependente, é possível empregar a segmentação da exibição, de forma análoga àquela empregada pelo padrão Formulário Segmentado.

O uso do Formulário Mestre-Detalhe permite a manipulação de uma informação principal (mestre), através do padrão Formulário, e suas informações dependentes (detalhes), através do padrão Grade. Combinados num mesmo padrão, oferecem ao usuário uma visão geral da informação individual, assim como de suas relações de dependência. O resultado é uma maior identificação com o modelo conceitual e a maior agilidade na manipulação de dados associados, que não precisam ser acessados em telas individuais ou editados com o auxílio de lookups para as chaves estrangeiras. Esse padrão é extremamente útil em sistemas de informação para o preenchimento de tabelas dependentes para as quais se espera uma quantidade limitada de registros. Formulários comerciais e financeiros são naturalmente implementados através desse padrão.

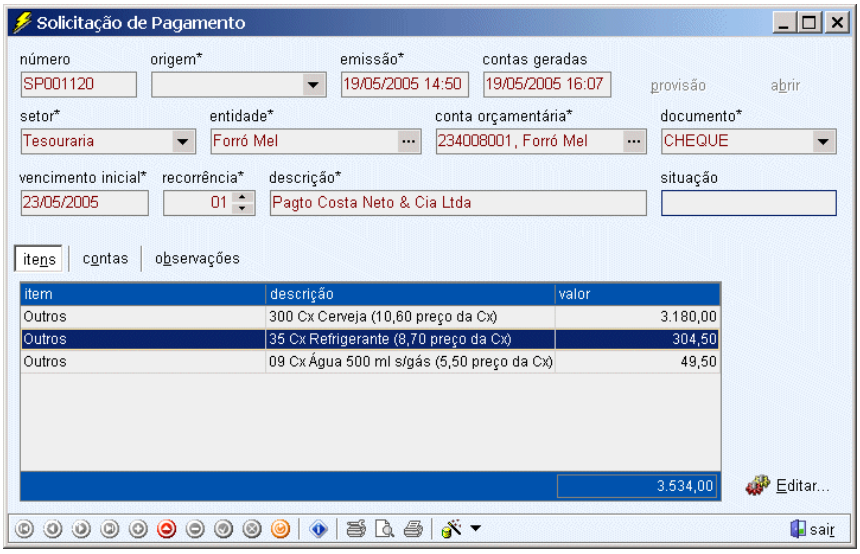


Figura 54 – Exemplo: Formulário Mestre-Detalhe

A Figura 54 ilustra um típico Formulário Mestre-Detalhe. Na parte superior da

interface, os dados principais que caracterizam o Formulário. Na parte inferior, três Grades estão disponíveis, uma por vez, através da segmentação da exibição. O segmento visível, de “itens”, relaciona três registros dependentes da informação principal apresentada. A implementação de referência suporta parcialmente os Formulários Mestre-Detalhe. Além de alguma programação para definir novos comportamentos, é preciso especificar a interface visual no construtor de interfaces a partir de uma subclasse concreta da classe TclSimpleForm do modelo (herança visual).

5.1.19 Assistente

Os padrões baseados em Grades e Formulários resolvem uma quantidade razoável de problemas de interface em sistemas de informação. No entanto, eles são naturalmente orientados a dados e não a tarefas. Para realizar tarefas mais complexas, que dependem de uma série de decisões intermediárias, é preciso empregar outro padrão. Nesse padrão, o usuário é apresentado à resolução de um problema em partes ou etapas. A tarefa principal é decomposta em uma série de tarefas menores que são executadas sequencialmente, em etapas bem definidas, com o auxílio de um Assistente. Em cada etapa, o Assistente apresenta os elementos de interface necessários a sua conclusão. O usuário então passa prossegue à etapa seguinte, retorna à etapa anterior, para ajustar alguma informação, ou cancela a execução do Assistente. A interface dos Assistentes é composta por uma região de interação, onde são construídas as interfaces de cada etapa, e uma região de controle, através da qual se dá a navegação pelas diversas etapas, o cancelamento e o acesso ao sistema de ajuda. As interfaces de cada etapa do Assistente são independentes entre si e podem empregar outros padrões como Formulários e Grades. Uma das características mais marcantes dos Assistentes é seu modo de guiar o usuário passo a passo na execução das tarefas, limitando as formas de interação e disponibilizando apenas o necessário à execução de cada etapa, reduzindo assim a incidência de erros ao longo do processo.

O emprego de Assistentes para a realização de tarefas de complexidade moderada e alta é ideal, pois, permite que diversas tarefas menores sejam

combinadas em seqüência para a execução de uma tarefa maior. Ao invés de executar uma única tarefa complexa ou uma série de tarefas não associadas, o usuário realiza várias tarefas menores, simples, em uma ordem natural ao modelo conceitual. A memorização dos procedimentos é alcançada com menos esforço, pela repetição e pelo maior entendimento do sistema. Usuários iniciantes ganham rapidamente perspectiva a respeito do modelo e, com isso, aprendem a usar o sistema mais facilmente. Esse padrão é repetidamente empregado em sistemas de manipulação direta em geral, não sendo específico a sistemas de informação. Em sistemas de informação, seu emprego é uma alternativa indicada para a implementação de procedimentos avançados ou cadastros de alta complexidade, cujas informações estejam distribuídas em muitas partes e em ordinalidades variáveis.

As Figuras 55, 56 e 57 apresentam três etapas de um Assistente. Na primeira, faz-se a localização da entidade com a qual se deseja trabalhar. A interface apresenta aspectos de Grade Contextualizada, com as caixas de marcar atuando como filtros, caixas de texto com opções para digitação do “nome” e do “documento” da entidade e a Grade, na parte inferior da tela, com o resultado da pesquisa. Ao avançar para a etapa seguinte, o usuário passa a interagir com os principais dados da entidade, dispostos numa típica configuração de Formulário. Ocupando a parte esquerda da tela, um controle para a contextualização das informações. Através desse *widget* especializado, é possível selecionar a parte da informação que se deseja trabalhar. Na Figura 57, o usuário selecionou uma outra parte da informação para trabalhar. A interface é, tipicamente, a de um Formulário Mestre-Detalhe. A implementação de referência suporta parcialmente os Assistentes. As classes que controlam o comportamento do Assistente são genéricas e, para cada etapa de um assistente, é preciso definir uma classe descendente para implementar o comportamento específico. Para cada interface, de cada etapa do Assistente, é preciso especificar a interface visual no construtor de interfaces. A interface geral do assistente está disponível a partir de uma subclasse concreta da classe TfmWizard do modelo (herança visual).

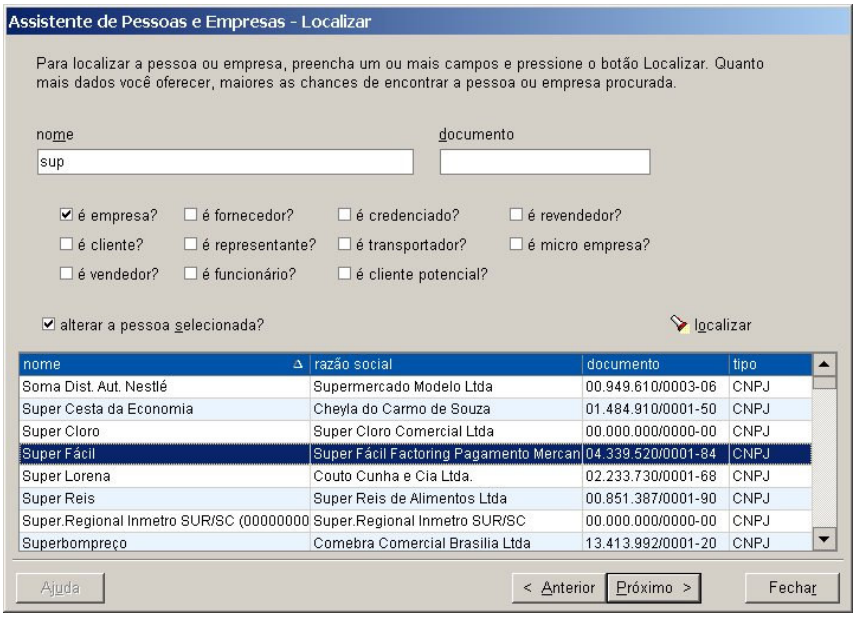


Figura 55 – Exemplo: Assistente (parte 1)

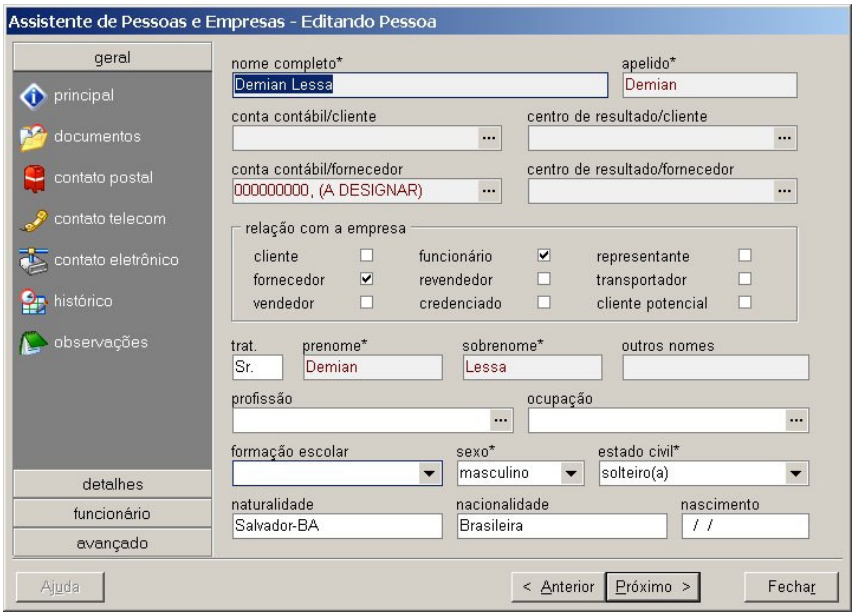


Figura 56 – Exemplo: Assistente (parte 2)

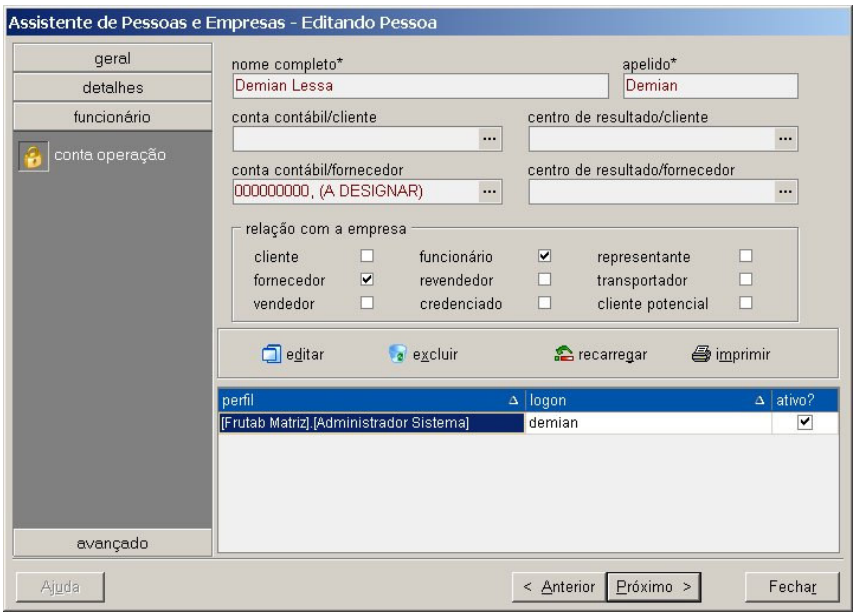


Figura 57 – Exemplo: Assistente (parte 3)

5.1.20 Listagem Simples

O único padrão de saída do sistema é a Listagem Simples. Através do emprego desse padrão, saídas permanentes de um conjunto de informações podem ser geradas. Essas informações estão normalmente disponíveis através de uma das variações do padrão Grade. A saída deve representar fielmente os dados, conforme apresentados na Grade específica. A Alternância de Cores deve ser empregada em conjunto com a Listagem Simples, de modo a facilitar a leitura dos dados da planilha. A Listagem Simples deve cuidar de todos os aspectos visuais da saída, de modo a garantir a apresentação completa e legível dos dados, levando ainda em conta a formatação visual da Grade associada.

O emprego de Listagens Simples facilita o trabalho de documentação do usuário, que passa a contar com a possibilidade de imprimir dados de qualquer grade do sistema. Com essas Listagens, as tarefas manuais que precisam de informação em papel para sua execução também passam a ser contempladas pelo sistema. Em sistemas de informação, o emprego desse padrão é uma alternativa intermediária entre relatórios personalizados e a visualização de dados na tela. Com a disponibilidade de Listagens Simples automaticamente associadas às Grades do

sistema, relatórios individuais mais complexos passam a ser implementados para uma quantidade menor de entidades.

5.2 IMPLEMENTAÇÃO DE REFERÊNCIA DA COLEÇÃO DE PADRÕES

A biblioteca de geração de interfaces baseadas em padrões, desenvolvida como parte deste trabalho, é composta por 45 classes públicas distribuídas em 39 arquivos. Esses, por sua vez, são dispostos em duas pastas principais a partir da raiz do código fonte e organizados em 7 pastas ao todo, somando 12.233 linhas de código na sua última compilação, em julho de 2005; nessa mesma data, a biblioteca maior da qual faz parte totalizava 46.554 linhas compiladas. Foi desenvolvida no ambiente Borland Delphi 5.0. O resumo dessa composição é apresentado nas Tabelas 5 e 6.

O Diagrama 1 mostra as relações de herança e associação entre as classes que definem a arquitetura de geração: TclForm, TclDataForm, TclDataEditor (abstrata, representada em itálico no diagrama), TclGridEditor e TclSearchEditor. As classes TclForm e TclDataForm representam janelas; a primeira implementa o suporte a padrões como *Somente-Leitura* e *Foco*; a segunda define o protocolo de comunicação entre as camadas de interface e manipulação de dados através da ligação com a classe abstrata, TclDataEditor. As classes de editores representam formas de manipular os dados através da interface. Três formas são atualmente suportadas pela biblioteca: a primeira, baseada no padrão Ficha ou Formulário (TclDataEditor); a segunda, na Grade (TclGridEditor); a última, na Grade de Pesquisa Livre (TclSearchEditor). A classe TclGridEditor, derivada de TclDataEditor, implementa sua planilha de dados através do uso de uma instância concreta da classe abstrata TclDataGridProxy. Essa abordagem garante que a implementação do padrão Grade seja independente do objeto utilizado para representar a planilha de dados na prática. No diagrama, é possível observar que uma implementação concreta está disponível: TclQuantumGridProxy. De forma análoga, TclSearchEditor estende a classe TclGridEditor de modo a incluir as facilidades para a implementação da Pesquisa Livre. Estabelece a dependência com uma barra de pesquisa abstrata, através da classe TclSearchBarProxy. Uma barra concreta está

disponível: TclDevExpressSearchBarProxy. A outra classe do diagrama, TclVisualDataEditor, representa um editor de dados genérico que pode ser representado por qualquer janela descendente de TclDataForm. Seu objetivo é permitir que uma Grade ou Grade de Pesquisa ofereça, durante sua execução, um outro editor para manipular um ou mais de seus próprios dados.

O Diagrama 2 mostra a relação das classes que implementam os serviços mais diretamente ligados à interface gráfica. Primeiramente, uma nova classe de janela é estabelecida como a classe base da hierarquia de janelas utilizadas pela biblioteca. A opção por não utilizar TclDataForm diretamente promove a independência entre o comportamento das janelas e sua interface visual. TfmAutoGUI é a janela base a partir da qual ocorre a geração automática das interfaces na biblioteca. Essa classe estabelece os elementos mínimos da interface visual e, através da comunicação com diversos editores de dados, tem sua interface gráfica construída dinamicamente. Os editores utilizados com TfmAutoGUI são: TclSimpleEditor, para janelas do padrão Ficha, TclSimpleGridEditor, para janelas do padrão Grade e, para janelas do padrão Grade de Pesquisa Livre, a classe TclSimpleSearchEditor. Essas três classes são privadas e servem apenas para alterar certas propriedades dos editores a partir dos parâmetros de chamada no código. Outras duas classes de janela constam do Diagrama 2. TclSimpleForm e TclGridForm, controladas pelos editores TclSimpleEditor e TclGridEditor, respectivamente, são classes de janelas que implementam os padrões Ficha e Grade. Diferentemente do caso da geração automática, essas classes estão à disposição na biblioteca para a utilização de herança visual, um mecanismo de programação suportado pelo Delphi através do qual as mudanças realizadas na interface visual de uma janela em tempo de desenvolvimento são propagadas para todas as janelas de classes descendentes, ainda em tempo de desenvolvimento. Ao usar uma dessas classes de janelas para esse fim, o programador terá duas vantagens: aproveitará o mecanismo de herança visual oferecido pelo Delphi e, ao mesmo tempo, terá todo o controle do comportamento realizado pelas classes de editores da biblioteca. Essas classes podem ser, portanto, utilizadas como ponto de partida para a implementação das variações dos padrões da Grade e da Ficha não suportadas no modo automático pela biblioteca.

O Diagrama 3 apresenta as classes envolvidas no desenvolvimento de

assistentes. O que a biblioteca oferece através dessas classes é um modelo baseado, novamente, na separação entre controle e interface. A classe TfmWizard representa a estrutura básica da interface de um assistente e, através do uso de herança visual, a interface gráfica dos assistentes podem ser criadas. Cada etapa do assistente é controlada por uma instância do tipo TclWizardStep e o conjunto de etapas por uma instância de TclWizard. A história, isto é, a sucessão de etapas que o usuário da interface concluiu, são mantidas por um objeto TclWizardHistory.

O Diagrama 4 mostra a relação entre as classes de informações contextuais. Essas representam a forma e o conteúdo da informação que é exibido ao usuário caso o programador do aplicativo permita acesso às informações de contexto. Diferentemente de ajuda, essas informações trazem informações sobre o modelo do aplicativo e não sobre sua operação. Na forma padrão, através da classe TclInfoFormGUI, apenas as informações sobre os dados ativos são oferecidas; a classe TclPgInfoGridFormGUI, por outro lado, busca as informações diretamente do dicionário de dados de um banco de dados PostgreSQL (http://postgresql.org/) e as exibe numa grade.

A classe TclDAConfig, apresentada no Diagrama 6, é, em geral, a única classe da biblioteca utilizada pelo programador. Essa classe contém uma série de parâmetros para a configuração das interfaces que serão geradas. Tudo que é preciso para que a biblioteca gere uma interface é passar uma instância dessa classe para uma das chamadas “Show...” em TfmAutoGUI (Diagrama 5), que então instancia um editor adequado e inicia a execução da interface. TclDAConfig oferece dois métodos para facilitar a geração das interfaces, estabelecendo valores padrão consistentes para todos os parâmetros a serem passados aos editores através de TfmAutoGUI.

Na Tabela 6, a coluna *interface visual* indica se um padrão é implementado pela biblioteca. Exceto pela Grade de Pesquisa Avançada, que não é atualmente suportada pela biblioteca, todos os demais padrões de Editores de Dados são plenamente suportados em termos de funcionalidade. Em termos de interface visual, o programador precisa utilizar o construtor de interfaces para finalizar a interface dos padrões implementados parcialmente. Os *lookups* locais são *widgets* normalmente implementados pelo *toolkit* ou biblioteca visual utilizada.

A íntegra da biblioteca encontra-se à disposição para *download* na Internet a

partir do URL: <http://demianlessa.com/download/cl2.zip>.

Tabela 5 – Módulos e Classes da Biblioteca

Módulo	Arquivos
	00, cl/clDB.pas
	01, cl/clDBAggrated.pas
	03, cl/clFile.pas
	04, cl/clRegistry.pas
	05, cl/clStrings.pas
	06, rad/core/clConfig.pas
	07, rad/core/clDAConfig.pas
	08, rad/core/clDBForms.pas
	09, rad/core/clEdGrid.pas
	10, rad/core/clEditors.pas
	11, rad/core/clEdSearcg.pas
	12, rad/core/clEvents.pas
	13, rad/core/clForms.pas
	14, rad/core/clGrids.pas
	15, rad/core/clLogon.pas
	16, rad/core/clLookup.pas
	17, rad/core/clMail.pas
	18, rad/core/clProfile.pas
	19, rad/core/clPropLoader.pas
	20, rad/core/clSearchBar.pas
	21, rad/core/clSecurity.pas
	22, rad/core/clSession.pas
	23, rad/core/clSysProps.pas
	24, rad/core/clWizard.pas
	25, rad/core/db/zeos/clZeosModule.pas
	26, rad/gui/clAutoGUI.pas
	27, rad/gui/clGridGUI.pas
	28, rad/gui/clGUI.pas
	29, rad/gui/clInfoGridGUI.pas
	30, rad/gui/clInfoGUI.pas
	31, rad/gui/clMainConfig.pas
	32, rad/gui/clPgInfoGridGUI.pas
	33, rad/gui/clSimpleGUI.pas
	34, rad/gui/clWizardGUI.pas
	35, rad/gui/dx/clQGrid.pas
	36, rad/gui/dx/clQSearchBar.pas
	37, rad/gui/qr/clQQRGenericConfig.pas
	38, rad/gui/qr/rGenerico.pas
	Total de Linhas de Código

Linhas de Código	Classes Públicas
383	TclDataSetProxy, TclQueryProxy, TclDataSetProxyFactory, TclQueryProxyFactory
339	TDBAggregateData, TDBAggregate
188	———
52	———
969	TclRegistryDataset
215	———
143	———
276	TclDAConfig
292	TclDataForm, TclDataFrame, TclVisualDataEditor
492	TclGridEditor
1.270	TclLookupList, TclDataEditor
190	TclSearchEditor
105	TclEventManager
544	TclShortCutList, TclForm
460	TclDataGridProxy
41	TclLogonManager
207	TclLookupController
230	TclMailManager
139	TclUserPermission, TclUserProfile
86	TclPropertyLoader
52	TclSearchBarProxy
283	TclSecurityManager
268	TclSessionManager
49	———
706	TclWizardHistory, TclWizard, TclWizardStep
620	TdmZeos
443	TfmAutoGUI
166	TclDAConfig
86	TclDataFormGUI
75	TclInfoGridFormGUI
107	TclInfoFormGUI
232	TfmConfiguracoes
126	TclPgInfoGridFormGUI
380	TclSimpleForm, TclSimpleEditor

Arquivos

123	TfmWizard
902	TclQuantumGridProxy
147	TclDevExpressSearchBarProxy
226	TfxGenericoConfig
621	TqrReport
12.233	

Tabela 6 – Padrões Implementados na Biblioteca

Grupo	**Padrão**	**Interface Visual**
Indicador Visual	Identificador	não (nativo)
Indicador Visual	Obrigatoriedade	sim
Indicador Visual	Somente-Leitura	sim
Indicador Visual	Foco	sim
Indicador Visual	Zebrado	sim
Controle	Barra de Controles	sim
Controle	Menu Contextual	sim
Acesso a Dados	Lookup Local	não (nativo)
Acesso a Dados	Lookup por Delegação	sim
Editor de Dados	Grade	sim
Editor de Dados	Grade Contextualizada	parcial
Editor de Dados	Grade de Pesquisa Livre	sim
Editor de Dados	Grade Mestre-Detalhe	parcial
Editor de Dados	Grades Alternativas	parcial
Editor de Dados	Grade de Pesquisa Avançada	não
Editor de Dados	Formulário	parcial
Editor de Dados	Formulário Segmentado	parcial
Editor de Dados	Formulário Mestre-Detalhe	parcial
Editor de Dados	Assistente	parcial
Saída de Dados	Listagem Simples	sim

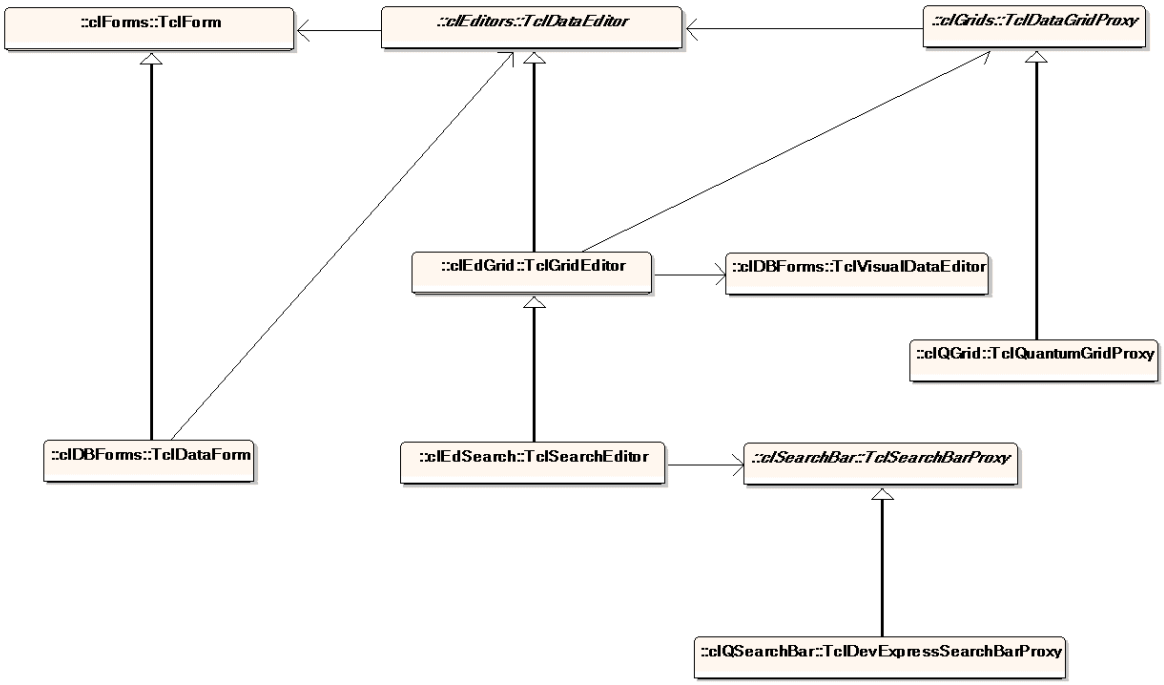


Diagrama 1 – Principais Classes de Arquitetura de Padrões

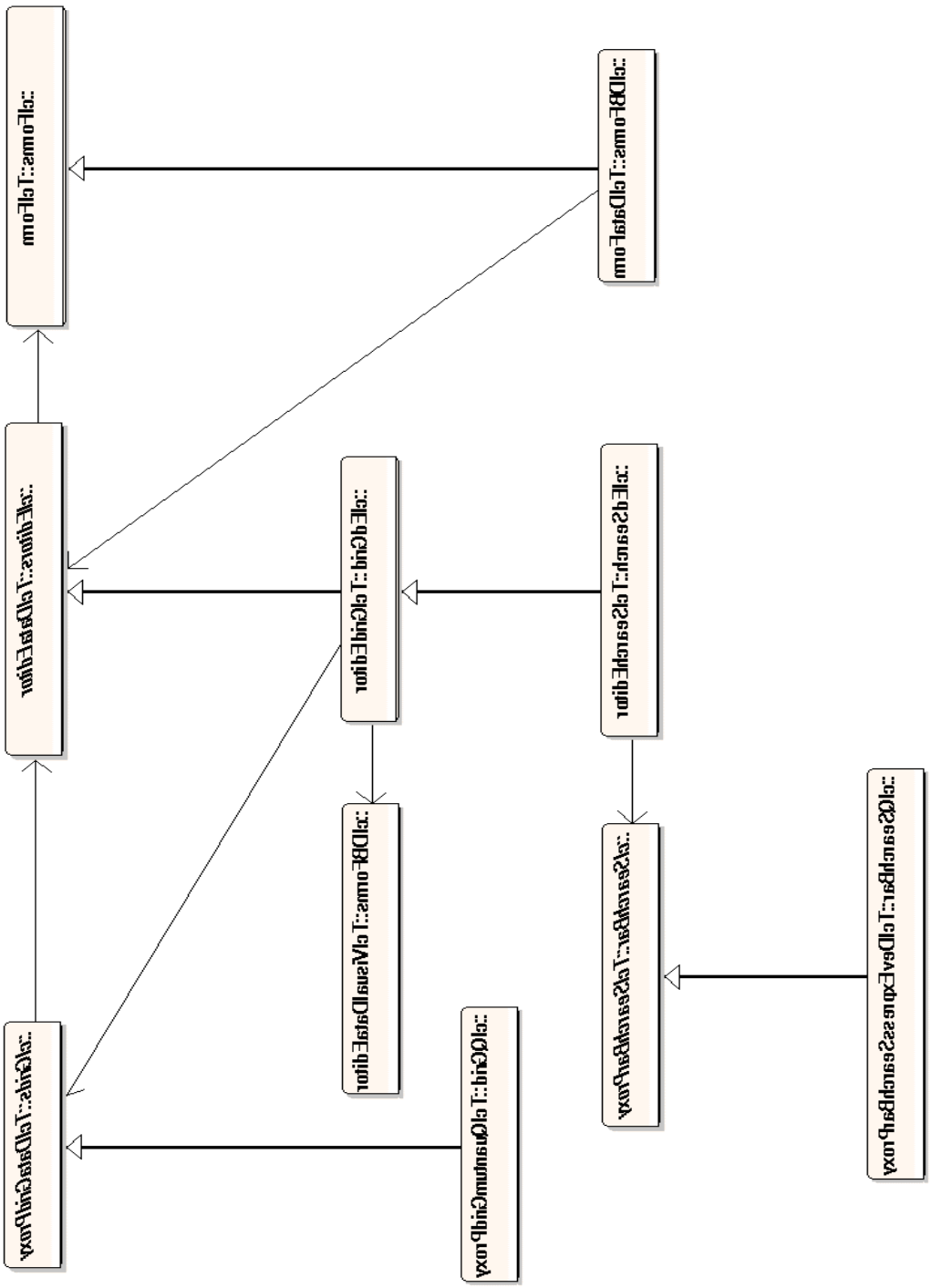


Diagrama 2 – Classes Concretas para Programação de Interfaces

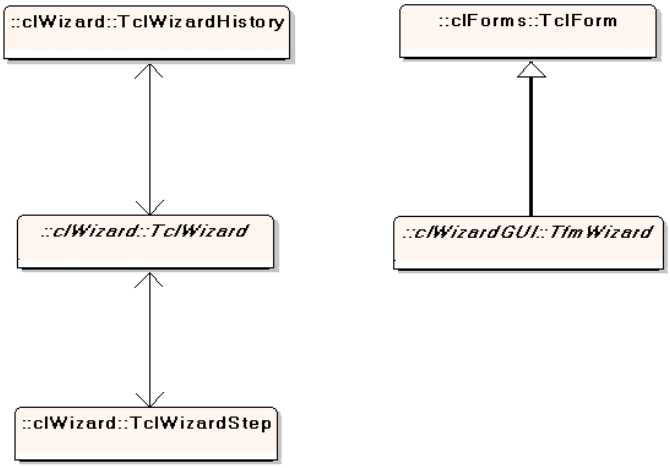


Diagrama 3 – Classes de Apoio à Implementação de Assistentes

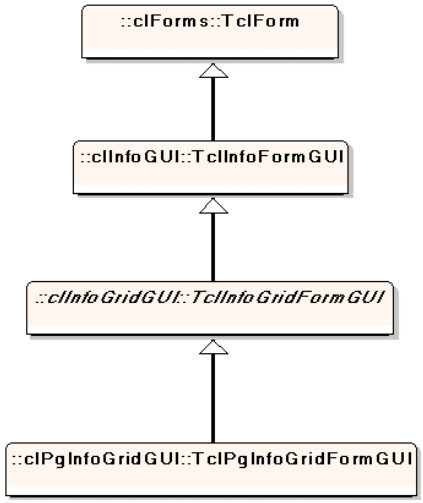


Diagrama 4 – Classes de Exibição de Informações Contextuais de Editores

Diagrama 5 – A Classe TfmAutoGUI

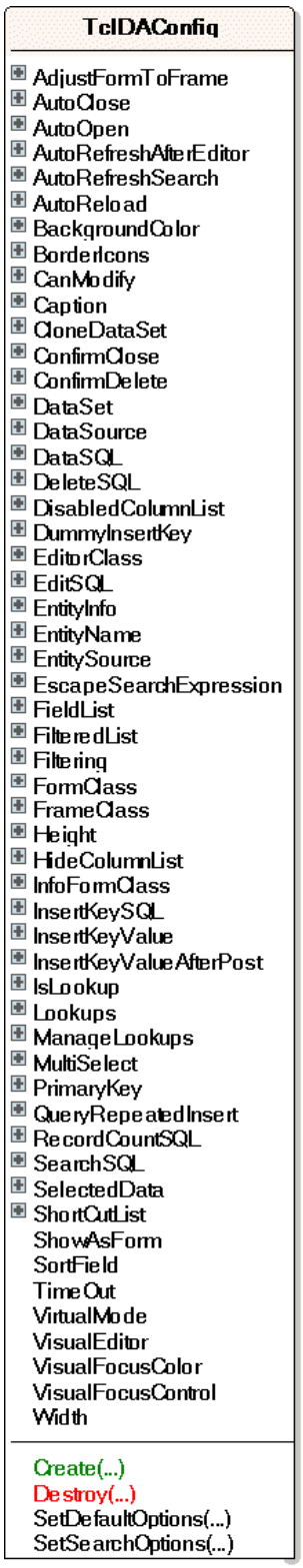


Diagrama 6 – A Classe TclDAConfig

5.3 USANDO OS PADRÕES

Um dos aspectos mais importantes relacionados ao uso dos padrões é o modelo de desenvolvimento empregado. O mais marcante desse modelo é a divisão de responsabilidades e tarefas no processo de desenvolvimento das interfaces. Quatro grupos estão envolvidos nesse modelo: profissionais de interface, analistas, programadores de ferramentas e programadores de aplicativos. Profissionais de interface e analistas elaboram as coleções de padrão e especificam as interfaces dos sistemas. Programadores de ferramentas utilizam as coleções de padrão e desenvolvem o *software* de apoio ao uso desses padrões nos aplicativos. Finalmente, os programadores de aplicativos utilizam as interfaces especificadas a partir dos padrões e as implementam nos aplicativos. O modelo de trabalho é simples e há uma escala de responsabilidades bem definida.

O profissional de interfaces passará a concentrar seus esforços mais no trabalho com padrões- sua identificação, criação e avaliação. Poderão participar de visitas e entrevistas em campo com a equipe de análise, a fim de coletar informações para a elaboração de novos padrões. O analista oferecerá auxílio ao profissional de interfaces na identificação de padrões e na avaliação das interfaces, indicando sua adequação para a finalidade proposta. Quando preciso, programadores podem ser acionados para a implementação de protótipos do modelo para facilitar sua avaliação. Uma vez elaborada uma coleção de padrões, o programador de ferramentas é acionado para dar início a sua implementação. Em geral, a implementação é realizada no ambiente de desenvolvimento utilizado na empresa para o desenvolvimento dos sistemas de informação. Caso precise de orientações além daquelas contidas na coleção, o programador solicitará esclarecimentos ao analista, para a parte funcional, e ao profissional de interfaces, para os aspectos relacionados à interação humano-computador e à parte visual dos padrões. A ferramenta desenvolvida implementará as propriedades e os comportamentos gerais da coleção de padrões através de uma biblioteca de classes. Os detalhes concretos da implementação de cada interface serão definidas em código específico no aplicativo

Durante o processo de análise, as interfaces do sistema de informação serão especificadas com o auxílio da coleção de padrões, que será utilizada para

comunicar a funcionalidade de cada padrão e demonstrar seu uso aos futuros usuários. O analista poderá precisar especificar interfaces não cobertas pela coleção. Nesses casos, as interfaces serão estudadas detalhadamente e novos padrões poderão surgir, incorporando-se à coleção. Uma vez especificadas todas as interfaces do sistema, o programador do aplicativo inicia seu trabalho. Com os resultados da análise, a especificação das interfaces, a coleção de padrões e a implementação da coleção, o programador está pronto para iniciar o desenvolvimento do aplicativo.

O modelo descrito incentiva a separação de responsabilidades entre profissionais de interface, analistas, programadores de bibliotecas e desenvolvedores de sistemas. A tarefa de elaboração de coleções ocorre paralelamente às demais, num processo contínuo, já que será sempre possível estender e depurar padrões existentes, incluir novos padrões na coleção, além de melhorar o próprio idioma de descrição dos padrões. Da mesma forma, a biblioteca deve ser atualizada regularmente para incorporar as atualizações da biblioteca, corrigir erros de modelagem e implementação. Os trabalhos de análise e desenvolvimento de sistemas, ao contrário, ocorrem somente quando surge um novo contrato. Essas tarefas dependem da estabilidade da coleção de padrões e sua implementação. Por isso, é importante que tanto a coleção de padrões quanto o código da biblioteca sejam mantidos através de um sistema de controle de versões.

6 DISCUSSÃO DO TRABALHO

Nas seções a seguir, a coleção de padrões elaborada e a implementação de referência serão discutidas em pormenor. A validade do trabalho é então analisada através da discussão de um sistema de informações desenvolvido através do emprego da biblioteca de classes implementada neste trabalho.

6.1 A COLEÇÃO DE PADRÕES

A linguagem de padrões elaborada neste trabalho descreve soluções para uma série de problemas que ocorrem repetidamente no desenvolvimento de sistemas de informação.

Os padrões do grupo Indicador Visual representam soluções a problemas comuns da interface visual, como identificar e estabelecer visualmente características de unidades de informação, como nos casos dos padrões Indicador, Obrigatoriedade, Somente-Leitura, e Foco, ou facilitar o uso ou interpretação da interface, como no caso do padrão Alternância de Cores. Esses são os únicos padrões independentes do escopo da coleção. São e utilizados em diversos tipos de aplicação e não apenas em sistemas de informação.

Dois padrões de Controle foram identificados: a Barra de Controles, cujo papel está relacionado ao controle das operações disponíveis através da interface visual, e o Menu de Contexto, cujo papel é controlar aspectos específicos da interface visual.

Outros dois padrões foram identificados para Acesso a Dados. Ambos realizam a mesma função, do ponto de vista conceitual: selecionar um valor. A forma de implementação é bem diversa e destaca uma característica interessante dos padrões. Se, na identificação e descrição dos padrões, o problema não tivesse sido especificado de forma precisa, esses padrões teriam sido, provavelmente, identificados como duas soluções alternativas para o mesmo problema: seleção de um único valor.

Os dez padrões seguintes implementam Editores de Dados. São tipicamente

os padrões que as equipes de desenvolvimento de sistemas de informação estão interessadas. Ocorrem em três categorias distintas: grades, fichas e assistentes. As grades são baseadas em uma planilha de dados e possuem três características fundamentais: suas unidades de informação são homogêneas; a visão das informações se dá através de suas partes e a quantidade de partes de cada informação é limitada. Duas variações da grade sofisticam a forma de recuperar os dados a exibir e outras duas ampliam o escopo da grade original, permitindo o acesso alternado a conjuntos distintos de informação. O princípio básico do formulário é a visão de uma unidade de informação complexa por vez, através de suas partes. A versão Formulário Segmentado é uma extensão do Formulário que permite a distribuição da informação ao longo de segmentos na interface. A versão Mestre-Detalhe amplia a complexidade do Formulário, através da integração de conjuntos dependentes de informações. Finalmente, o Assistente, que é um editor de dados especializado no apoio à execução de tarefas complexas.

O único padrão identificado para Saída de Dados foi a Listagem Simples, que representa uma visão permanente de um conjunto de dados com as mesmas características dos dados representados pela Grade.

É importante perceber que os padrões nem sempre ocorrem isoladamente. Dentre os padrões de Editor de Dados, por exemplo, é possível identificar o emprego de outros padrões da própria coleção. As relações de associação entre os padrões da coleção poderiam ter sido definidas como parte do idioma, através da inclusão de duas novas dimensões: usado por e usa. Quando um padrão utiliza outro, listaria o padrão usado na dimensão usa e seria listado na dimensão usado por pelo padrão usado. Por exemplo: Grade usa Barra de Controles e Alternância de Cores é usado por Listagem Simples.

Outro aspecto que não fica imediatamente claro na consulta à coleção de padrões e que poderia ser incluído no idioma refere-se à extensão do padrão na interface. Alguns padrões estão geralmente relacionados a um único *widget* da interface, como Obrigatório ou Somente-Leitura; outros, a um conjunto bem definido e limitado de *widgets*, como Barra de Controles; existem ainda padrões como o Assistente, que operam em toda a área disponível para a interação com o usuário.

Outras duas dimensões que poderiam auxiliar no correto emprego dos padrões são: contraindicação e contraexemplo. A contraindicação permitiria identificar circunstâncias nas quais é sabido que o emprego do padrão é

inadequado. Por exemplo, a utilização de uma Alternância de Cores numa Grade, combinado com Somente-Leitura, pode gerar uma interface difícil de interpretar: enquanto o Alternância de Cores produziria uma alternância vertical nas cores das linhas da Grade, o uso do Somente-Leitura mudaria a cor de todas as colunas não editáveis, resultando numa grade difícil de acompanhar. No entanto, o uso da Grade com Somente-Leitura ou da Grade com Alternância de Cores, são ambos, nos seus respectivos contextos, soluções interessantes. Outra forma mais ilustrativa de contraindicação é o contraexemplo. Enquanto a contraindicação descrever situações onde o uso de um padrão é inadequado, o contraexemplo, em analogia ao exemplo, apresenta uma ou mais imagens demonstrando possíveis efeitos e não aplicar o padrão no contexto do problema, ou, ainda, o emprego inadequado do padrão.

Os padrões identificados deverão ser prontamente reconhecidos, ainda que parcialmente, pela maioria dos programadores e usuários de sistemas de informação. A contribuição mais significativa da coleção não é a apresentação de padrões originais, mas a sistematização das descrições dos padrões desse tipo de sistema. A esse respeito, os padrões foram apresentados em linguagem clara e direta, num formato bem estruturado, permitindo a rápida comparação entre os padrões pela inspeção de suas diversas propriedades.

Na medida em que pesquisadores, profissionais e empresas concentrarem esforços no desenvolvimento e utilização de coleções de padrão, como a descrita no presente trabalho, tais coleções serão gradualmente refinadas e poderão ter novas dimensões e novos padrões acrescentados. Uma forma de agilizar esse processo é através da publicação e divulgação dessas coleções, tornando-as acessíveis a uma maior parte da comunidade interessada. Outra forma de avançar os desenvolvimentos na área de padrões é através da análise comparativa de padrões que procuram resolver os mesmos problemas, em uma mesma coleção e em coleções distintas, e da crítica dos idiomas empregados na descrição dos padrões em diversas coleções.

A coleção de padrões está disponível para *download* na Internet através do URL: <http://demianlessa.com/tecno/ui.php>.

6. 2 A IMPLEMENTAÇÃO DE REFERÊNCIA

A biblioteca de referência desenvolvida neste trabalho demonstrou a viabilidade da implementação de padrões como unidades reutilizáveis de código.

O modelo de classes da biblioteca permitiu o desacoplamento, em grande parte, entre a interface visual e o controle da telas através da separação das hierarquias de classes de janela e editores. Foram definidas três classes de editores e suas características visuais essenciais representadas através de classes abstratas, a fim evitar a determinação de aspectos da interface visual que limitariam as possíveis interfaces concretas suportadas pela biblioteca.

Para facilitar a programação de editores de dados, foi oferecida uma classe concreta que definia os aspectos mais básicos de consistência visual: esquema de cores e fontes, barra de controles, suporte a padrões de indicação visual como Foco e Somente-Leitura. Uma outra classe foi disponibilizada para configurar as características principais dos editores. Assim, a tarefa de programação de interfaces concretas no aplicativo ficou reduzida à definição das propriedades da classe de configuração e a subsequente chamada de um serviço da biblioteca. Essa foi a forma preferencial de programação das interfaces suportadas automaticamente pela biblioteca, isto é, aquelas que a biblioteca implementava integralmente e para as quais não havia necessidade de programação visual através do construtor de interfaces. No entanto, outros padrões, implementados parcialmente, dependiam do uso do construtor. Para utilizar esses padrões, o programador tinha que completar a edição visual das telas no construtor e codificar uns poucos comportamentos.

A observação do uso da biblioteca em sistemas de teste e de produção permitiu constatar que o tempo médio gasto tanto em interfaces suportadas automaticamente, quanto naquelas suportadas parcialmente, foi reduzido de forma considerável. Da mesma forma, a quantidade de código escrito pelo programador foi reduzida de maneira significativa.

Num primeiro momento, a biblioteca mostrou-se um tanto difícil de ser utilizada pelos programadores de aplicativos- aqueles menos familiarizados com a metodologia de orientação a objetos. Para esses, foi preciso explicar o uso da biblioteca de forma detalhada e apresentar exemplos ilustrativos antes que começassem a utilizar a ferramenta de forma produtiva. A ressalva que se faz,

porém, é que esse problema tem pouca relação com a arquitetura e a funcionalidade da biblioteca e mais como a falta de documentação apropriada.

Outro aspecto que poderia ser melhorado na biblioteca é a questão do acoplamento. Apesar do acoplamento entre as hierarquias de interface e controle se dá apenas nas suas classes básicas das hierarquias, o uso de interfaces poderia ter reduzido ainda mais essa dependência. No entanto, a opção por não utilizar interfaces deveu-se à plataforma adotada, cujo suporte a interfaces julgou-se inapropriado.

Por fim, o uso da biblioteca poderia ser ainda mais facilitado através do uso de arquivos externos de configuração ao invés do emprego de uma classe específica para a passagem de parâmetros de configuração. Com o uso de arquivos externos, os programadores deixariam de realizar chamadas para a geração de cada tela; ao invés disso, um ou mais arquivos seriam definidos e carregados para que uma determinada tela fosse gerada. Com isso, as telas poderiam ser facilmente alteradas, através de propriedades nos arquivos, sem a necessidade de recompilar o sistema para que as alterações fossem percebidas no sistema, como hoje é necessário fazer.

6. 3 VALIDAÇÃO DO TRABALHO

A coleção de padrões elaborada neste trabalho foi utilizada durante o processo de análise e desenvolvimento de um sistema de informações real, do segundo semestre de 2004 até o final de março de 2005. O sistema, responsável pela informatização de uma indústria de sucos no interior da Bahia, conta com mais de vinte usuários distribuídos entre os setores de recepção de cargas, produção, almoxarifado, administração comercial, administração financeira e planejamento estratégico. Encontra-se em produção plena há mais de três meses depois de um período de aproximadamente cinco meses de testes. O aplicativo conta com mais de 150 telas para entrada e manipulação dos dados do sistema: mais de 200 tabelas gerenciadas por um banco de dados PostgreSQL rodando sobre o Linux. Todos os padrões identificados na coleção são utilizados no sistema, que faz uso extensivo da biblioteca implementada neste trabalho.

Os padrões foram extremamente úteis no processo inicial de análise. No caso

específico desse projeto, o novo sistema substituiria um outro existente. A apresentação da coleção de padrões serviu para demonstrar como as telas baseadas nos padrões substituiriam as telas do sistema existente. Foram exploradas as funcionalidades dos padrões, assim como a consistência resultante do emprego sistemático de interfaces baseadas nos padrões. As idéias foram tão bem aceitas que a contratante solicitou a inclusão do documento de padrões na proposta comercial apresentada à empresa.

Os padrões foram colocados à prova nas sucessivas interações do desenvolvimento do sistema, quando novas versões eram entregues. A utilização dos padrões na maioria das telas do sistema e o treinamento de um técnico da contratante capaz de multiplicar o conhecimento adquirido na operação dessas telas permitiram o rápido engajamento dos usuários com o sistema. A inclusão de um sistema de coleta de informações de uso, ainda que mínimas, permitiu verificar e corrigir alguns pontos isolados de dificuldade na operação do sistema. Nenhum desses problemas, no entanto, esteve ligado à utilização dos padrões. A maior dificuldade dos usuários deveu-se à mudança na forma de interação com as janelas do sistema: o novo aplicativo operava as janelas de forma *modal* (para ativar uma nova janela é preciso fechar a janela ativa), enquanto o anterior as operava de forma não-*modal* (para ativar uma nova janela basta selecioná-la).

Desde que entrou em produção, os mais de 20 usuários do sistema interagem diariamente com suas diversas interfaces, por aproximadamente 8 horas por dia. Como o sistema cobre os processos dos mais diversos setores de uma indústria, desde a produção, passando pelo almoxarifado e chegando a administração financeira, o perfil dos usuários é bem heterogêneo. Ainda assim, apenas um setor da fábrica precisou abandonar o uso dos padrões em uma de suas telas em favor de uma interface alternativa, menos usual. De uma forma geral, esse projeto mostrou uma boa adaptação dos usuários ao uso das telas baseadas nos padrões elaborados.

Um sistema de informações de médio porte baseado na coleção de padrões elaborada foi implementado utilizando a biblioteca apresentada, demonstrando sua eficácia nos objetivos propostos. A rápida adaptação dos usuários ao uso do sistema, mesmo com a mudança na forma de manipulação das janelas, é um indicativo de que o emprego dos padrões é uma abordagem que traz benefícios e deve ser explorada em maior profundidade.

7 CONCLUSÃO

Esta dissertação propôs a elaboração de uma coleção de padrões de interfaces gráficas e sua implementação através de uma biblioteca de classes, ambos para uso específico no desenvolvimento de sistemas de informação. Para viabilizar a elaboração da coleção de padrões, foi definido um idioma com sete dimensões para sua descrição; foram identificados vinte padrões através de coleções disponíveis, da literatura e da experiência do próprio autor; foram classificados e descritos esses padrões. Antes de implementar a biblioteca, buscou-se subsídio através de um levantamento para identificar o tipo de apoio ao desenvolvimento de interfaces gráficas atualmente oferecido pelos ambientes de desenvolvimento (IDEs). Após sua implementação, buscou-se a validação conjunta da coleção de padrões e da biblioteca através do desenvolvimento de um sistema de informações para a indústria, baseado nos padrões e na biblioteca.

7.1 Contribuições do Trabalho

O trabalho realizou uma pesquisa com 49 ambientes de desenvolvimento para caracterizar o tipo de apoio oferecido no desenvolvimento de interfaces gráficas por essas ferramentas (seção 3.10). Foi constatado que, de um modo geral, o apoio é reduzido e, quando existe, é geralmente limitado ao uso de um construtor de interfaces apenas. Não foram encontradas ferramentas que suportassem o processo de análise para auxiliar no desenvolvimento das interfaces.

Foi elaborada uma coleção de padrões de interfaces gráficas para apoio ao desenvolvimento de sistemas de informação (seção 5.1). A coleção identificou vinte padrões em cinco categorias: Identificador Visual, Controle, Acesso a Dados, Editor de Dados e Saída de Dados. Esses padrões foram também apresentados no apêndice A, num formato especificamente elaborado para facilitar a comunicação com os diversos profissionais envolvidos no uso dos padrões.

Além da coleção de padrões propriamente dita, foi desenvolvido um idioma para a descrição dos padrões (seção 4.2.2). O idioma determina as dimensões

comunicadas através de cada padrão. O idioma proposto pode ser utilizado para descrever outras coleções de padrões, independentemente, até mesmo em escopos diferentes daquele estabelecido para a coleção deste trabalho.

Uma biblioteca de classes com aproximadamente 12.500 linhas foi desenvolvida para implementar a coleção de padrões elaborada neste trabalho (seção 5.2). A plataforma de desenvolvimento selecionada foi o Microsoft Windows com o Borland Delphi. Dos padrões identificados, onze foram implementados de forma completa na biblioteca, sete de forma parcial, um já estava disponível de forma nativa no ambiente selecionado e apenas um não foi implementado.

Finalmente, foi desenvolvido um sistema de informações completo, baseado na biblioteca de padrões, para a informatização de uma indústria no estado da Bahia. Nesse sistema, todos os padrões implementados pela biblioteca são utilizados. Praticamente todas as telas do sistema, do total aproximado de 150, utilizam um ou mais dos padrões identificados e descritos neste trabalho.

7.2 LIMITAÇÕES DO TRABALHO

O levantamento sobre os ambientes de desenvolvimento ficou restrito a uns poucos IDEs comerciais e várias ferramentas *open source*- o que não pode se afirmar ser a distribuição correta de ferramentas no mercado de desenvolvimento de sistemas.

A coleção de padrões de interfaces foi desenvolvida especificamente para emprego no desenvolvimento de sistemas de informação. Poderia ter incluído mais padrões- desde os mais elementares, abordando interações em pequena escala, como identificadores visuais, aos mais complexos, oferecendo soluções completas à manipulação de dados, como alguns dos padrões de edição de dados identificados.

O idioma proposto poderia ter considerado outras dimensões, como: “usado por” e “usa”, para estabelecer o relacionamento entre os padrões; “contra-indicação”, para identificar quando um padrão não deve ser empregado; “contra-exemplo”, para ilustrar formas incorretas de uso do padrão. Algumas dessas novas dimensões só foram percebidas tarde, após o desenvolvimento e emprego dos padrões para o desenvolvimento da biblioteca e do sistema de informações. Outras resultariam num

esforço adicional para a elaboração dos padrões que não compensaria os benefícios adicionais de sua inclusão para este trabalho.

A plataforma de desenvolvimento selecionada, Windows e Borland Delphi, estabelece, na prática, que apenas esse ambiente pode ser utilizado caso se deseje utilizar a biblioteca desenvolvida. Essa escolha pareceu lógica, por combinar um IDE amplamente utilizado no desenvolvimento de sistemas de informação, com o sistema operacional mais utilizado pelas empresas contratantes desse tipo de sistema em Salvador-BA. Plataformas alternativas poderiam ter sido consideradas para a implementação da biblioteca, como .NET, Java e Web.

A biblioteca implementou apenas os padrões da coleção e, da mesma forma, foi planejada para ser usada no desenvolvimento de sistemas de informação apenas. Poderia ter ido além, oferecendo suporte total aos padrões parcialmente desenvolvidos. Optou-se por não fazer isso no presente trabalho, devido ao custo adicional em termos de tempo e esforço de codificação. Quando comparados os benefícios adicionais de tal implementação com seu custo, achou-se melhor deixar esse esforço para uma etapa futura.

7.3 TRABALHOS FUTUROS

O idioma e a abordagem empregados na elaboração da coleção poderiam ser utilizados em outros contextos, fora dos sistemas de informação, e os resultados comparados. Padrões de outros contextos poderiam sugerir abordagens alternativas para diversas soluções propostas.

Uma ferramenta poderia ser desenvolvida para auxiliar no desenvolvimento dos padrões. Ela poderia oferecer uma interface que integrasse um editor de interfaces, para edição do modelo visual do padrão, com uma folha de propriedades para cada interface. O responsável pela coleção de padrões construiria a interface no editor e descreveria o padrão através de sua folha de propriedades que, adicionalmente, poderia permitir a seleção ou definição do idioma empregado na descrição. Uma vez prontos os padrões, a ferramenta poderia ser utilizada para imprimir ou exportar a coleção, ou ainda, para gerar o código fonte de sua implementação, numa linguagem suportada.

A abordagem baseada em padrões poderia ser avaliada através de um estudo de campo. Dois grupos de uma mesma organização poderiam ser selecionados e, por um período de tempo estabelecido, utilizariam, em essência, a mesma metodologia de desenvolvimento de sistemas de informação, exceto pelo emprego de padrões, que ocorreria em um dos grupos apenas. Os processos de análise e desenvolvimento dos sistemas seriam observados e as dimensões de consistência da interface, aprendizado, retenção de aprendizado, linhas de código e tempo de desenvolvimento seriam analisadas e, ao final do período, os resultados comparados.

A última sugestão é o emprego de abordagem de padrões em outras disciplinas, como na modelagem e programação de bancos de dados. Da mesma forma que o estudo e a discussão dos padrões de projeto de engenharia de software (GAMMA *et al*, 1995) serve como fonte para o estudo dos padrões de interface, a extensão da abordagem para outras áreas poderia gerar benefícios semelhantes. Modelos concretos e técnicas utilizadas na prática pelos profissionais da área de bancos de dados poderiam ser catalogados e disponibilizados para a comunidade. Os padrões de modelagem seriam baseados em problemas como: “é preciso acessar com freqüência uma parte (pequena) dos campos de uma entidade que possui uma quantidade (muito) grande de campos”, “é preciso acessar a referência principal de uma entidade que possui diversas referências associadas”. Os exemplos, nesse caso, seriam dados através do modelo DER da solução, acompanhado de uma descrição. Os padrões de programação seriam baseados em problemas como: “é preciso acessar um único registro de uma entidade, baseado no valor agregado (MAX, MIN, AVG) de um campo em outra entidade relacionada”, “é preciso acessar um único registro de uma entidade, baseado no valor agregado (MAX, MIN, AVG) de um campo em outra entidade não relacionada”. O contexto, nesse caso, poderia incluir a parte essencial de um modelo de dados genérico para ilustrar a relação entre as entidades. Os exemplos poderiam ser dados através de instruções SQL, acompanhados de uma descrição.

REFERÊNCIAS

ABOWD, G. et al. *User interface languages*: a survey of existing methods. Technical Report Prg-Tr-5-89. Oxford University Computing Laboratory Programming Research Group, Oct. 1989. Disponível em: <http://citeseer.ist.psu.edu/abowd89user.html>. Acesso em: 5 jul. 2005.

ALIASING. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Aliasing>. Acesso em: 5 jul. 2005.

APPLE. *Cocoa.* [ca. 1996]. [*Site* oficial do produto]. Disponível em: <http://developer.apple.com/cocoa/>. Acesso em: 5 jul. 2005.

AREND, V. U. User interface patterns: components for user interfaces. *SAP Design Guild,* [S.l.], n.8, nov. 2004. Disponível em: <http://www.sapdesignguild.org/ editions/edition8/patterns.asp>. Acesso em: 5 jul. 2005.

AARON Marcus; VAN DAM, Andries. User-interface developments for the nineties. *Computer*, v. 24, n. 9, p. 49-57, sep., 1991.

ASR33. In: WIKIPEDIA the free enclyclopedia. Disponível em: <http://en.wikipedia.org/wiki/ASR33>. Acesso em: 5 jul. 2005.

BÄUMER, D. et al. User interface prototyping: concepts, tools, and experience. In: THE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 18., 1996, Berlin. *Proceedings*... Washington, DC: IEEE Computer Society, 1996. p.532-541. ISBN: 0-8186-7246-3. Disponível em: <http://doi.ieeecomputersociety.org/10.1109/ICSE. 1996.493447>; e <http://portal.acm.org/ citation.cfm?id= 227726.227841>. Acesso em: 29 jun. 2005.

BETTS, B. et al. Goals and objectives for user interface software. *SIGGRAPH Computer Graphics*, Nova Iorque, v. 21, n. 2, p. 73-78, apr. 1987. ISSN: 0097-8930. Disponível em: <http://doi.acm.org/10.1145/24919.24920>. Acesso em: 5 jul. 2005.

BIG. paybacks from 'discount' usability engineering. Full text, Publisher Site. Source, *IEEE Software archive,* v. 7, n. 3, may 1990.

BORLAND. Borland Delphi. 1995. [*Site* oficial do produto]. Disponível em: <http://www.borland.com/delphi>. Acesso em: 5 jul. 2005.

BORLAND. *Manuais do Borland Delphi*. [Em formato eletrônico, disponível em qualquer versão do produto]. Disponível em: <http://www.borland.com/delphi>. Acesso em: 5 jul. 2005.

BRICLIN, D. *VisiCalc:* information from its creators, Dan Bricklin and Bob Frankston. 1999. Disponível em: <http://www.bricklin.com/visicalc.htm>. Acesso em: 5 jul. 2005.

BROOKS JR. F. P. *The mythical man-month*: essays on software engineering. 20th anniversary ed. [Reading, MA]: Addison-Wesley, 1995. Chap. 16-17, p.179-226, ISBN 0-201-83595-9.

BROOKSHEAR, J. G. *Computer science***:** an overview. 6th ed. [Reading, MA]: Addison-Wesley, 2000. p.11-14, 119-166; 225-283. ISBN 0-201-35747-X.

BUSH, V. As we may think. *The Atlantic Monthly*, [S.l.], p. 101-108, July 1945. Disponível em: <http://sloan.stanford.edu/mousesite/Secondary/Bushframe.html>.

Acesso em: 5 jul. 2005.

BUTLER, K. A.; JACOB, R. J.; JOHN, B. E. Introduction & overview to human-computer interaction. In: CONFERENCE COMPANION ON HUMAN FACTORS IN COMPUTING SYSTEMS: COMMON GROUND, 1996, Vancouver, British Columbia, New York, NY: ACM Press, 1996. p. 328-329. ISBN: 0-89791-832-0. Disponível em: <http://doi.acm.org/10.1145/257089.257352>. Acesso em: 5 jul. 2005.

BUXTON, W.; LAMB, M. R.; SHERMAN, D.; SMITH, K. C. Towards a comprehensive user interface management system. In: THE ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, 1983, Detroit, Michigan. 10., *Proceedings*... New York, NY: ACM Press, 1983. p. 35-42. ISBN: 0-89791-109-1. Disponível em: <http://doi.acm.org/10.1145/800059.801130>. Acesso em: 5 jul. 2005.

BYOUS, J. *Java technology*: the early years. Sun Developer Network, may 1998. Disponível em: <http://java.sun.com/features/1998/05/birthday.html>. Acesso em: 5 jul. 2005.

CARLISLE, M. C.; MAES, P. RAPID: a free, portable GUI design tool. In: THE 1998 ANNUAL ACM SIGADA INTERNATIONAL CONFERENCE ON ADA, 1998, Washington, DC. *Proceedings...* New York, NY: ACM Press, 1998. p.158-164. ISSN: 1094-3641. Disponível em: <http://doi.acm.org/0.1145/289524. 289570>. Acesso em: 5 jul. 2005.

CATHODE ray tube. In: WIKIPEDIA THE FREE ENCLYCLOPEDIA. Disponível em: <http://en.wikipedia.org/wiki/Cathode_ray_tube>. Acesso em: 5 jul. 2005.

COLDEWEY, J.; KRÜGER, I. Form-based user interface: the architectural patterns: a pattern language. In: THE 1997 EUROPEAN PATTERN LANGUAGES OF PROGRAMMING CONFERENCE, 1997, Irsee, Germany. *Proceedings...:* Siemens Technical Report 120/SW1/FB. Disponível em: <http://citeseer.ist.psu.edu/ 240474.html>. Acesso em: 5 jul. 2005.

COMMAND line interface. In: WIKIPEDIA THE FREE ENCLYCLOPEDIA. Disponível em: <http://en.wikipedia.org/wiki/Command_line_interface>. Acesso em: 5 jul. 2005.

COMPUTER display. In: WIKIPEDIA THE FREE ENCLYCLOPEDIA. Disponível em: <http://en.wikipedia.org/wiki/Computer_display>. Acesso em: 5 jul. 2005.

COMPUTER HISTORY MUSEUM. Mountain View, CA: 1996-2005. [Apresenta um grande acervo de artefatos, imagens e documentos relacionados à computação, inclusive software]. Disponível em: <http://www.computerhistory.org/>. Acesso em: 5 jul. 2005.

COMPUTER MUSEUM. School of Computer Science and Engineering, Monash University. Disponível em: <http://www.csse.monash.edu.au/museum/>. Acesso em: 5 jul. 2005.

COMPUTER terminal. In: WIKIPEDIA THE FREE ENCLYCLOPEDIA. Disponível em: <http://en.wikipedia.org/wiki/Computer_terminal>. Acesso em: 5 jul. 2005.

COMPUTER workstation. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Computer_workstation>. Acesso em: 5 jul. 2005.

COMPUTER. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://www.britannica.com/eb/article?tocId=216029>. Acesso em: 5 jul. 2005.

CTSS. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/CTSS>. Acesso em: 5 jul. 2005.

DA CRUZ, F. *Columbia university computing history*: a chronology of computing at Columbia University. New York, NY: 2001. [Apresenta a cronologia da história da computação]. Disponível em: <http://www.columbia.edu/acis/history/>. Acesso em: 5 jul. 2005.

DA SILVA, P. P.; GRIFFITHS, T.; PATON, N. W. Generating user interface code in a model based user interface development environment. In: THE WORKING CONFERENCE ON ADVANCED VISUAL INTERFACES, [s.d.], Palermo, Italy. *Proceedings*... New York, NY: ACM Press, [s.d.]. p. 155-160. ISBN: 1-58113-252-2. Disponível em: <http://doi.acm.org/10.1145/345513.345301>. Acesso em: 5 jul. 2005.

DATABASE INTELLIGENCE, INC. *DBASE.* [*Site* oficial do produto]. Disponível em: <http://www.dbase.com/>. Acesso em: 5 jul. 2005.

DBASE. In: WIKIPEDIA the free encyclopedia. Disponível em: <http:// en. wikipedia.org/ wiki/DBASE>. Acesso em: 5 jul. 2005.

DE BAAR, D. J.; FOLEY, J. D.; MULLET, K. E. Coupling application design and user interface design. In: THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1992, Monterey, California. *Proceedings*... New York, NY: ACM Press, 1992. p.259-266. Disponível em: <http://doi.acm.org/10.1145/ 142750.142806 >. Acesso em: 5 jul. 2005.

DESMARAIS, M. C.; HAYNE, C.; JAGANNATH, S.; Keller, R. A survey on user expectations for interface builders. In: CONFERENCE COMPANION ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1994, Boston, Massachusetts. New York, NY: ACM Press, 1994. p. 279-280. ISBN: 0-89791-651-4. Disponível em: <http:// doi.acm.org/10.1145/259963.260491>. Acesso em: 5 jul. 2005.

DEWAN, P.; SOLOMON, M. An approach to support automatic generation of user interfaces. *ACM Transactions on Programming Language and Systems*, New York, v. 12, n.4, p.566-609, Oct. 1990. ISSN: 0164-0925. Disponível em: <http://doi.acm.org/ 10.1145/88616.214518>. Acesso em: 5 jul. 2005.

DIRECT manipulation interface. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Direct_manipulation_interface>. Acesso em: 5 jul. 2005.

DRAPER, S. W.; NORMAN, D. A. Software engineering for user interfaces. In: THE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 7., 1984, Orlando, Florida. *Proceedings...* Piscataway, NJ: IEEE Press, 1984. p. 214-220. ISBN~ISSN: 0270-5257,0-8186-0528-6. Disponível em: <http://doi.acm.org/10.1145/ 800054.801972>. Acesso em: 5 jul. 2005.

DTSS. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en. wikipedia.org/wiki/DTSS>. Acesso em: 5 jul. 2005.

ECLIPSE Project. *Eclipse SWT*. [ca. 2004]. [*Site* oficial do produto]. Disponível em: <http://www.eclipse.org/swt/>. Acesso em: 5 jul. 2005.

ENGELBART. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Engelbart>. Acesso em: 5 jul. 2005.

ENIAC. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.

wikipedia.org/wiki/ENIAC>. Acesso em: 5 jul. 2005.

FINKEL, R. What is an operating system? In: TUCKER, A. B. (Ed.). *Computer science handbook*. 2nd. ed. [S.l.]: Chapman & Hall/CRC, 2004. Chap. 80. ISBN 1-58488-360-X.

FLANAGAN, D. *Java in a nutshell***:** a desktop quick reference. 4th ed. Sebastopol, CA: O'Reilly, 2002. p.3-18. ISBN 0-596-00283-1.

FOLEY, J. et al. *Computer graphics***:** principles and practice. 2nd ed. [Reading, MA]: Addison-Wesley, 1990. p.1-24; 145-200; 391-469. ISBN 0-201-12110-7.

GAJOS, K.; WELD, D. S. SUPPLE: automatically generating user interfaces. In: THE 9TH INTERNATIONAL CONFERENCE ON INTELLIGENT USER INTERFACE, 2004, Funchal, Madeira. *Proceedings…* New York, NY: ACM Press, 2004. p. 93-100. ISBN: 1-58113-815-6. Disponível em: <http://doi.acm.org/10.1145/964442.964461>. Acesso em: 5 jul. 2005.

GAMMA, E. et al. *Design patterns:* elements of reusable object-oriented software. [Upper Saddle River, NJ], 1995. p.207-219. ISBN: 0-201-63361-2.

GNOME Project. *GLADE User Interface Builder*. 1998. [*Site* oficial do produto]. Disponível em: <http://glade.gnome.org/>. Acesso em: 5 jul. 2005.

GNOME Project. *GNOME*. 1997. [*Site* oficial do produto]. Disponível em: <http://gnome.org/>. Acesso em: 5 jul. 2005.

GNU Project. *GTK+ the GIMP toolkit*. 1995. [*Site* oficial do produto]. Disponível em: <http://www.gtk.org/>. Acesso em: 5 jul. 2005.

GOMAA, M. *Towards a better model based user interface development environment:* a comprehensive survey. North Dakota: State University, [ca. 2004]. Disponível em: <http://www.cs.ndsu.nodak.edu/~gomaa/Towards%20a%20better%20model%20based%20user%20interface%20development%20%20%20%20%20%20environment.doc>. Acesso em: 5 jul. 2005.

GRANLUND, A.; LAFRENIÈRE, D.; DAVID, C. A. In: THE HCI INTERNATIONAL 2001, 9TH INTERNATIONAL CONFERENCE ON HUMAN-COMPUTER INTERACTION, 2001, New Orleans, Louisiana. *Proceedings*... [S.l.]: [s.n.], [ca. 2001]. Disponível em: <http://www.sm.luth.se/csee/csn/publications/HCIInt2001Final.pdf>. Acesso em: 5 jul. 2005.

GRAPHICAL user interface. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Graphical_user_interface>. Acesso em: 5 jul. 2005.

GRUDIN, J. Interface: an evolving concept. *Communications of the ACM*, New York, v. 36, n. 4, p. 110-119, Apr. 1993. ISSN: 0001-0782. Disponível em: <http://doi.acm.org/10.1145/255950.153585>. Acesso em: 5 jul. 2005.

GRUDIN, J. The case against user interface consistency. *Communications of the ACM,* v. 32, n. 10 p. 1164-1173, Oct. 1989. ISSN: 0001-0782. Disponível em: <http://doi.acm.org/10.1145/67933.67934>. Acesso em: 5 jul. 2005.

GRUDIN, J. The computer reaches out: the historical continuity of interface design. In: THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS: EMPOWERING PEOPLE, 1990, Seattle, Washington. New York, NY: ACM Press, 1990. p.261-268. ISBN: 0-201-50932-6. Disponível em: <http://doi.acm.org/10.1145/97243.97284>. Acesso em: 5 jul. 2005.

HARTSON, H. R.; HIX, D. Human-computer interface development: concepts and systems for its management. *ACM Computing Surveys*, New York, v.21, n. 1, p. 5-92, Mar.1989. ISSN:0360-0300. Disponível em: <http://doi.acm.org/10.1145/62029.62031 >. Acesso em: 5 jul. 2005.

HARVARD Mark I. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Harvard_Mark_I>. Acesso em: 5 jul. 2005.

HENRY, T. R.; HUDSON, S. E. Using active data in a UIMS. In: THE ANNUAL ACM SIGGRAPH SYMPOSIUM ON USER INTERFACE SOFTWARE, 1., 1988, Alberta, Canada. *Proceedings…*New York, NY: ACM Press, 1988. p. 167-178. ISBN: 0-89791-283-7. Disponível em: <http://doi.acm.org/10.1145/62402.62429>. Acesso em: 5 jul. 2005.

HISTORY of computing hardware (1960s-present). In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/History_of_ computing_hardware%281960s-present%29>. Acesso em: 5 jul. 2005.

HISTORY of computing hardware. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/History_of_computing_hardware>. Acesso em: 5 jul. 2005.

HISTORY of operating systems. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/History_of_operating_systems>. Acesso em: 5 jul. 2005.

HISTORY of the graphical user interface. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/History_of_the_GUI>. Acesso em: 5 jul. 2005.

HIX, D. Generations of user-interface management systems. *IEEE Software*, 7, n.5, p.77-87, 1990.

HOLUB, A. Building user interfaces for object-oriented systems: part 1. *Java World,* [S.l.], July 1999. Disponível em: <http://www.javaworld.com/javaworld/jw-07-1999/jw-07-toolbox.html>. Acesso em: 29 jun. 2005.

HOLUB, A. Building user interfaces for object-oriented systems: part 2: the visual proxy architecture. *Java World,* sept. 1999. Disponível em: <http://www.javaworld.com/javaworld/jw-09-1999/jw-09-toolbox.html>. Acesso em: 29 jun. 2005.

HOLUB, A. More on getters and setters. *Java World*, [S.l.], jan. 2004. Disponível em: <http://www.javaworld.com/javaworld/jw-01-2004/jw-0102-toolbox.html>. Acesso em: 29 jun. 2005.

HOLUB, A. Why extends is evil. *Java World,* [S.l.], aug. 2003. Disponível em: <http://www.javaworld.com/javaworld/jw-08-2003/jw-0801-toolbox.html>. Acesso em: 29 jun. 2005.

HOLUB, A. Why getters and setters are evil. *Java World*, [S.l.], sept. 2003. Disponível em:<http://www.javaworld.com/javaworld/jw-09-2003/jw-0905-toolbox.html>. Acesso em: 29 jun. 2005.

HUTCHINS, E. L.; HOLLAN, J. D.; NORMAN, D. A. *Direct manipulation interfaces*: Human-Computer Interaction. [S.l.]: Lawrence Erlbaum, 1985. v. 1, p 311-338.

HYPERCARD. In: WIKIPEDIA the free encyclopedia. Disponível em:

<http://en.wikipedia.org/wiki/Hypercard>. Acesso em: 5 jul. 2005.

ICON (computing). In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Icon_%28computing%29>. Acesso em: 5 jul. 2005.

JAVA programming language. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Java_programming_language>. Acesso em: 5 jul. 2005.

KAZMAN, R.; BASS, L. Evaluating user interface tools. In: CONFERENCE COMPANION ON HUMAN FACTORS IN COMPUTING SYSTEMS, Boston, Massachusetts, 1994. New York, NY: ACM Press, 1994. p. 429-430. ISBN: 0-89791-651-4. Disponível em: <http://doi.acm.org/10.1145/259963.260427>. Acesso em: 5 jul. 2005.

KAZMAN, R.; BASS, L. Evaluating user interface tools. In: CONFERENCE COMPANION ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1994, Boston, Massachusetts. New York, NY: ACM Press, 1994. p. 429-430. ISBN: 0-89791-651-4. Disponível em: <http://doi.acm.org/10.1145/259963.260427>. Acesso em: 5 jul. 2005.

KDE Project. *KDE*. 1996. [*Site* oficial do produto]. Disponível em: <http://kde.org/>. Acesso em: 5 jul. 2005.

KELLER, R. K. User Interface Tools: a survey and perspective. In: THE WORKSHOP ON SOFTWARE ENGINEERING AND HUMAN-COMPUTER INTERACTION, 1994, [S.l.], 1994. *Proceedings...* London: Springer-Verlag, 1994. p. 225-231. ISBN: 3-540-59008-0. Disponível em: <http://citeseer.ist.psu.edu/keller95user.html>. Acesso em: 5 jul. 2005.

LATZINA, M. Software design with user interface patterns: centered on the user. *SAP INFO*, [S.l.], n. 104, apr. 2003. Disponível em: <http://www.sap.info/en/go/19932>. Acesso em: 5 jul. 2005.

LESSA, D. *Página principal dos padrões de ui*. 2004. [Apresenta um conjunto de padrões de interfaces com textos explicativos]. Disponível em: <http://demianlessa.com/tecno/ui.php>. Acesso em: 5 jul. 2005.

LESSTIF. 1995. [*Site* oficial do produto]. Disponível em: <http://www.lesstif.org/>. Acesso em: 5 jul. 2005.

LINDEN, T. A. The use of abstract data types to simplify program modifications. In: THE 1976 CONFERENCE ON DATA: ABSTRACTION, DEFINITION AND STRUCTURE,1976, ake City, Utah. *Proceedings…* New York, NY: ACM Press,1976. p.12-23. ISSN: 0163-5808. Disponível em: <http://doi.acm.org/10.1145/984344.807113>. Acesso em: 5 jul. 2005.

LINEBACK, N. *Nathan's Toasty Technology Page*. [S.l]: 2001-2005. [Apresenta descrições e imagens de diversos sistemas gráficos ao longo da história]. Disponível em: <http://toastytech.com/guis/index.html>. Acesso em: 5 jul. 2005.

LÖWGREN, J. History, state and future of user interface management systems. *SIGCHI Bulletin*, New York, v. 20, n. 1, p. 32-44, jul. 1988. ISSN: 0736-6906. Disponível em: <http://doi.acm.org/10.1145/49103.491 5 >. Acesso em: 5 jul. 2005.

MANDELKERN, D. GUIs: the next generation. Communications of the *ACM*, New York, v. 36, n. 4, p. 36-39, Apr. 1993. ISSN: 0001-0782. Disponível em: <http://doi.acm.org/10.1145/255950.153574>. Acesso em: 5 jul. 2005.

MANDELKERN, D. GUIs: the next generation. *Communications of the ACM,* v. 36, n. 4, p.36-39, apr.1993. ISSN: 0001-0782. Disponível em: <http://doi.acm.org/10.1145/255950.153574>. Acesso em: 5 jul. 2005 .

MCAMIS, D. *Totally RAD:* we road test five IDEs. [s.l.]: Builder AU, sept. 2004. Disponível em: <http://www.builderau.com.au/program/0, 39024614,39131183,00.htm>. Acesso em: 29 jun. 2005.

MEYER, B. *Object Oriented Software Construction.* 2nd ed. [Upper Saddle River, NJ]: Prentice Hall, 1997. p.37-160. ISBN 0-13-629155-4.

MICROSOFT Windows. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Microsoft_Windows>. Acesso em: 5 jul. 2005.

MICROSOFT. *Microsoft Foundation Classes*. 1992. [*Site* oficial do fabricante]. Disponível em: <http://msdn.microsoft.com/>. Acesso em: 5 jul. 2005.

MICROSOFT. *Microsoft Visual Studio*. 1997. [*Site* oficial do produto]. Disponível em: <http://msdn.microsoft.com/vstudio/>. Acesso em: 5 jul. 2005 .

MORSE, A.; REYNOLDS, G. Overcoming current growth limits in UI development. *Communications of the ACM,* New York, v. 36, n. 4, p. 72-81, apr. 1993. ISSN: 0001-0782. Disponível em: <http://doi.acm.org/10.1145/255950.153579>. Acesso em: 5 jul. 2005 .

MÜLLER-PROVE, M. *Vision & reality of hypertext and graphical user interfaces.* 2002. 122f. Dissertação (Mestrado) – Departamento de Informática, Universidade de Hamburgo. Disponível em: <http://www.mprove.de/diplom/index.html>. Acesso em: 5 jul. 2005.

MULTICS. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/MULTICS>. Acesso em: 5 jul. 2005.

MYERS, B. A. A brief history of human-computer interaction technology. *Interactions,* New York, v. 5, n. 2, p. 44-54, mar. 1998. ISSN: 1072-5520. Disponível em: <http://doi.acm.org/10.1145/274430.274436>. Acesso em: 5 jul. 2005.

MYERS, B. A. Challenges of HCI design and implementation. *Interactions*, New York, v. 1, n. 1, p. 73-83, jan. 1994. ISSN: 1072-5520. Disponível em: <http://doi.acm.org/10.1145/174800.174808>. Acesso em: 5 jul. 2005 .

MYERS, B. A. Demonstrational interfaces: a step beyond direct manipulation. *Computer*, Los Alamitos, v. 25, n. 8, p. 61-73, aug. 1992. ISSN: 0018-9162. Disponível em: <http://dx.doi.org/10.1109/2.153286>. Acesso em: 5 jul. 2005 .

MYERS, B. A. Graphical user interface programming. In: TUCKER, A. B. (Ed.). *Computer science handbook*. 2nd ed. [S.l.]: Chapman & Hall/CRC, 2004. Chap. 48. ISBN 1-58488-360-X.

MYERS, B. A. User interface software tools. *ACM Transactions on Computer-Human Interaction,* New York, v. 2, n. 1, p. 64-103, mar. 1995. ISSN: 1073-5 16. Disponível em: <http://doi.acm.org/10.1145/200968.200971>. Acesso em: 5 jul. 2005.

MYERS, B. A. et al. Garnet: comprehensive support for graphical, highly interactive user interfaces. *Computer*, Los Alamitos, v. 23, n. 11 p. 71-85, nov. 1990. ISSN: 0018-9162. Disponível em: <http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=60882>. Acesso em: 5 jul. 2005.

MYERS, B. A et al. Strategic directions in human-computer interaction. *ACM*

Computing Surveys, New York, v. 28, n. 4, p. 794-809, Dec. 1996. ISSN: 0360-0300. Disponível em: <http://doi.acm.org/10.1145/ 242223.246855>. Acesso em: 5 jul. 2005.

MYERS, B. A.; HUDSON, S. E.; PAUSCH, R. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction*, New York, v.7, n.1, p.3-28, Mar.2000. ISSN:1073-5 16. Disponível em:<http://doi.acm.org/ 10.1145/344949.344959>. Acesso em: 5 jul. 2005.

MYERS, B. A.; OLSEN, D. R. User interface tools. In: CONFERENCE COMPANION ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1994, Boston, Massachusetts, New York, NY: ACM Press, 1994. p. 421-422. ISBN: 0-89791-651-4. Disponível em: <http://doi.acm.org/10.1145/259963.26 5 35>. Acesso em: 5 jul. 2005.

MYERS, B. A.; ROSSON, M. B. Survey on user interface programming. In: THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1992, Monterey, California. *Proceedings…* New York, NY: ACM Press, 1992. p.195-202. ISBN: 0-89791-513-5. Disponível em: <http://doi.acm.org/10.1145/ 142750.142789>. Acesso em: 5 jul. 2005.

MYERS, B. A. User-interface tools: introduction and survey. *ACM Transactions on Computer-Human Interaction (TOCHI), v.2,* p. 64-103, 1995.

MYERS, B. A. Window interfaces: a taxonomy of window manager user interfaces. *IEEE Computer Graphics & Applications*, v. 8, n.5, p.65-84, 1988.

NICHOLS, J.; MYERS, B. A.; LITWACK, K. Improving automatic interface generation with smart templates. In: INTERNATIONAL CONFERENCE ON INTELLIGENT USER INTERFACE, 9., 2004, Funchal, Madeira. *Proceedings…* New York, NY: ACM Press, 2004. p. 286-288. ISBN: 1-58113-815-6. Disponível em: <http://doi.acm.org/10.1145/964442.964507>. Acesso em: 5 jul. 2005.

NIELSEN, J. Noncommand user interfaces. *Communications of the ACM,* New York, v.36, n.4, p.83-99, Apr.1993. ISSN: 0001-0782. Disponível em: <http://doi.acm.org/ 10.1145/255950.153582>. Acesso em: 5 jul. 2005.

NLS (computer system). In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/NLS_%28computer_system%29>. Acesso em: 5 jul. 2005.

NLS (computer system). In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/On-Line_System>. Acesso em: 5 jul. 2005.

NOVELL. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Novell>. Acesso em: 5 jul. 2005.

OLIVEIRA, T. M. V. Amostragem não probabilística: adequação de situações para uso e limitação de amostras por conveniência, julgamento e quotas. *Administração On Line*: Prática Pesquisa e Ensinos, [São Paulo], v.2, n.3, jul./ago./set. 2001. Disponível em: <http://www.fecap.br/adm_online/art23/tania2.htm>. Acesso em: 5 jul. 2005.

OLSEN, D. R et al.. Whiter (or wither) UIMS?. In: THE SIGCHI/GI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS AND GRAPHICS INTERFACE, 1987,Toronto, Ontario. *Proceedings…* New York, NY: ACM Press, 1987. . 311-314. ISSN: 0736-6906. Disponível em: <http://doi.acm.org/10.1145/29933.275649>. Acesso em: 5 jul. 2005.

OLSEN, D. R.; KLEMMER, S. R. The future of user interface design tools. In: CHI ' 5 EXTENDED ABSTRACTS ON HUMAN FACTORS IN COMPUTING SYSTEMS, 2005, Portland, Oregon. New York, NY: ACM Press, 2005. p. 2134-2135. ISBN: 1-59593-002-7. Disponível em: <http://doi.acm.org/10.1145/156808.157124>. Acesso em: 5 jul.2005.

OPEN GROUP. *Motif.* [ca. 1989]. [*Site* oficial do produto]. Disponível em: <http://www.opengroup.org/motif>. Acesso em: 5 jul. 2005.

OPERATING systems timeline. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Operating_systems_timeline>. Acesso em: 5 jul. 2005.

OPERATING systems. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Operating_system>. Acesso em: 5 jul. 2005.

PARNAS, D. L. Designing software for ease of extension and contraction. In: THE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 3., 1978, Atlanta, Georgia. *Proceedings…* Piscataway, NJ: IEEE Press, 1978. p. 264-277. Disponível em: <http://doi.acm.org/10.1145/800099.803218>. Acesso em: 5 jul. 2005.

PARNAS, D. L. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, New York, v.15, n.12, p.153-158, Dec. 1972. ISSN:0001-0782. Disponível em: <http://doi.acm.org/10.1145/361598.361623>. Acesso em: 5 jul. 2005.

PARNAS, D. L. The influence of software structure on reliability. In: THE INTERNATIONAL CONFERENCE ON RELIABLE SOFTWARE, 1975, Los Angeles, California. *Proceedings...* New York, NY: ACM Press, 1975. p. 358-362. Disponível em: <http://doi.acm.org/10.1145/390016.808458>. Acesso em: 5 jul. 2005.

PARNAS, D. L.; SHORE, J. E; WEISS, D. Abstract types defined as classes of variables. In: THE 1976 CONFERENCE ON DATA: ABSTRACTION, DEFINITION AND STRUCTURE, 1976, Salt Lake City, Utah. *Proceedings...* New York, NY: ACM Press, 1976. p.149-154. ISSN: 0163-5808. Disponível em: <http://doi.acm.org/10.1145/984344.807133>. Acesso em: 5 jul. 2005.

PAWSON, R.; MATTHEWS, R. *Naked Objects.. [*S.l.]: John Wiley & Sons, 2002a. ISBN: 0-47084-425. Disponível em: <http://www.nakedobjects.org/static.php?content=content.html>. Acesso em: 5 jul. 2005.

Pawson, R.; Matthews, R. Naked Objects. In: COMPANION OF THE 17TH ANNUAL ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS, 2002, Seattle, Washington. New York, NY: ACM Press, 2002. p. 36-37. ISBN: 1-58113-626-9. Disponível em: <http://doi.acm.org/10.1145/985072.985091>. Acesso em: 5 jul. 2005.

PETRIE, M. *WORDSTAR Resource Site***!**. 1999. [*Site* não oficial do produto. Apresenta a história e outros recursos sobre o WordStar]. Disponível em: <http://www.wordstar.org/>. Acesso em: 5 jul. 2005.

PIXELATION. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Pixelation>. Acesso em: 5 jul. 2005.

PIZANO, A.; SHIROTA, Y.; IIZAWA, A. Automatic generation of graphical user interfaces for interactive database applications. In: INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, 2., 1993, Washington, DC.

Proceedings... New York, NY: ACM Press, 1993. p. 344-355. ISBN: 0-89791-626-3. Disponível em: <http://doi.acm.org/ 10.1145/170088.170166>. Acesso em: 5 jul. 2005

POWERS, M. K. Ensemble: a graphical user interface development system for the design and use of interactive toolkits. In: ANNUAL ACM SIGGRAPH SYMPOSIUM ON USER INTERFACE SOFTWARE AND TECHNOLOGY, 2., 1989, Williamsburg, Virginia. *Proceedings...* New York, NY: ACM Press, 1989. p. 168-178. ISBN: 0-89791-335-3. Disponível em: <http://doi.acm.org/10.1145/73660.73681>. Acesso em: 5 jul. 2005.

PUERTA, A. R.; MAULSBY, D. Management of interface design knowledge with MOBI-D. In: INTERNATIONAL CONFERENCE ON INTELLIGENT USER INTERFACES, 2., 1997, Orlando, Florida. *Proceedings...* New York, NY: ACM Press, 1997. p.249-252. ISBN: 0-89791-839-8. Disponível em: <http://doi.acm.org/10.1145/ 238218.238337>. Acesso em: 5 jul. 2005.

PUERTA, A. R.; SZKELEY, P. Model-based interface development. In: CONFERENCE COMPANION ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1994, Boston, Massachusetts. New York, NY: ACM Press, 1994. p. 389-390. ISBN: 0-89791-651-4. Disponível em: <http://doi.acm.org/10.1145/259963.26519>. Acesso em: 5 jul. 2005.

PUERTA, A.; EISENSTEIN, J. Towards a general computational framework for model-based interface development systems. In: INTERNATIONAL CONFERENCE ON INTELLIGENT USER INTERFACES, 4., 1999, Los Angeles, California. *Proceedings...* New York, NY: ACM Press, 1999. p. 171-178. ISBN: 1-58113-098-8. Disponível em: <http://doi.acm.org/10.1145/291080.291108>. Acesso em: 5 jul. 2005.

PUNCH card. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Punch_card>. Acesso em: 5 jul. 2005.

RAYMOND, E. S.; LANDLEY, R. W. *The art of Unix usability*. [S.l.]: Pearson Education, Apr. 2004. Disponível em: <http://www.catb.org/~esr/writings/taouu/ html/index.html>. Acesso em: 5 jul. 2005.

REFLEXIVE user interface. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Reflexive_user_interface>. Acesso em: 5 jul. 2005.

REIMER, J. A history of the GUI. *Ars Technica*, May 2005. Disponível em: <http://arstechnica.com/articles/paedia/gui.ars>. Acesso em: 5 jul. 2005.

RHYNE, J. et al. Tools and methodology for user interface development. *SIGGRAPH Computer Graphics,* New York, v. 21, n. 2, p. 78-87, Apr. 1987. ISSN: 0097-8930. Disponível em: <http://doi.acm.org/10.1145/24919.24921>. Acesso em: 5 jul. 2005.

ROGERS, W. P. Encapsulation is not information hiding. *Java World*, [S.l.], May 2001. Disponível em: <http://www.javaworld.com/javaworld/jw-5-2001/jw-518-encapsulation.html>. Acesso em: 29 jun. 2005.

ROSENBERG, J. et al. UIMSs: threat or menace? In: THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1988, Washington, DC. *Proceedings...* New York, NY: ACM Press, 1988. p.197-200. ISBN: 0-201-14237-6. Disponível em: <http://doi. acm.org/10.1145/57167.57200>. Acesso em: 5 jul. 2005.

ROSSON, M. B.; MAASS, S.; KELLOGG, W. A. Designing for designers: an analysis

of design practice in the real world. In: THE SIGCHI/GI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS AND GRAPHICS INTERFACE, 1987, Toronto, Ontario. *Proceedings...* New York, NY: ACM Press, 1987. p. 137-142. ISSN: 0736-6906. Disponível em: <http://doi.acm.org/10.1145/29933.30873>. Acesso em: 5 jul. 2005.

SCHWALLER, T. Aus dem hohen Norden. *Linux-Magazin,* [S.l.], Mai, 1996. Disponível em: <http://www.linux-magazin.de/Artikel/ausgabe/1996/5/Qt/qt.html>. Acesso em: 5 jul. 2005.

SOEGAARD, M. *Interaction Styles*. [Apresenta formas tradicionais de estilos de interação]. Disponível em: <http://www.interaction-design.org/encyclopedia/ interaction_styles.html>. Acesso em: 5 jul. 2005.

SOEGAARD, M. UIMS (User Interface Management System) or User Interface Architecture. [Apresenta uma definição de UIMS]. Disponível em: <http://www. interaction-esign.org/encyclopedia/uims_user_interface_management_system.html>. Acesso em: 5 jul. 2005.

STIM, *Science & Technology in the Making*. [Desenvolvido pela Standford University, 1997-1998. Apresenta textos, vídeos e outros recursos sobre a história recente da ciência e da tecnologia]. Disponíveis em: <http://sloan.stanford.edu/mousesite>. Acesso em: 29 jun. 2005.

SUN Microsystems. *Abstract Window Toolkit (AWT).* 1995. [*Site* oficial do produto]. Disponível em: <http://java.sun.com/products/jdk/awt/>. Acesso em: 5 jul. 2005.

SUN Microsystems. *SWING.* [ca. 1998]. [*Site* oficial do produto]. Disponível em: <http://java.sun.com/products/jfc/>. Acesso em: 5 jul. 2005.

SUTHERLAND, I. *Sketchpad: a man-machine graphical communication system*. MIT, 1963. Report #296. Disponível em: <http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-574.pdf>. Acesso em: 5 jul. 2005.

SYSTEM/360. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/System/360>. Acesso em: 5 jul. 2005.

SZEKELY, P. A. Retrospective and Challenges for Model-Based Interface Development. In: DESIGN, SPECIFICATION AND VERIFICATION OF INTERACTIVE SYSTEMS. [S.l.]: Springer, 1996. Disponível em: <http://citeseer.ist.psu.edu/szekely96retrospective.html>. Acesso em: 5 jul. 2005.

SZEKELY, P. A. User interface prototyping: tools and techniques. In: INTERNATIONAL CONFERENCE OF SOFTWARE ENGINEERING, WORKSHOP ON SE-HCI, 1994, Sorrento, Italia. [S.l.]: Springer, 1994. p. 76-92. ISBN: 3-540-59008-0. Disponível em: <http://citeseer.ist.psu.edu/pedro94user.html>. Acesso em: 5 jul. 2005.

SZEKELY, P.; LUO, P.; NECHES, R. Beyond interface builders: model-based interface tools. In: THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1993, Amsterdam, The Netherlands. *Proceedings...* New York, NY: ACM Press,1993. p.383-390. ISBN: 0-89791-575-5. Disponível em: <http://doi.acm.org/10.1145/169 5 9.1693 5 >. Acesso em: 5 jul. 2005.

TANENBAUM, A. S. *Modern operating systems*. 2nd. ed. Upper Saddle River, NY: Prentice-Hall, 2001. p.1-67; 671-758. ISBN 0-13-031358-0.

TANGIBLE user interface. In: WIKIPEDIA the free encyclopedia. Disponível em:

<http://en.wikipedia.org/wiki/Tangible_User_Interface>. Acesso em: 5 jul. 2005.

TATSUKAWA, K. Graphical toolkit approach to user interaction description. In: THE SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS: REACHING THROUGH TECHNOLOGY, 1991, New Orleans, Louisiana. *Proceedings…* New York, NY: ACM Press, 1991. p. 323-328. ISBN: 0-89791-383-3. Disponível em: <http://doi.acm.org/10.1145/108844.108934>. Acesso em: 5 jul. 2005.

TELEPRINTER. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Teleprinter>. Acesso em: 5 jul. 2005.

TEXT user interface. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Text_user_interface>. Acesso em: 5 jul. 2005.

TIDWELL, J. *Common Ground*: a pattern language for human-computer interface design. [S.l.]: [s.n.], 1998. Disponível em: <http://www.mit.edu/~jtidwell/ common _ground.html>. Acesso em: 5 jul. 2005.

TIDWELL, J. *UI patterns and techniques*: introduction. [Apresenta um catálogo de padrões de interfaces gráficas para a *web* e para o *desktop*]. Disponível em: <http://time-tripper.com/uipatterns/>. Acesso em: 5 jul. 2005.

TIME-SHARING. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Time-sharing>. Acesso em: 5 jul. 2005.

TIMELINE of computing 1950-1979. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Timeline_of_computing_1950-1979>. Acesso em: 5 jul. 2005.

TIMELINE of computing 1980-1989. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Timeline_of_computing_1980-1989>. Acesso em: 5 jul. 2005.

TIMELINE of computing 1990-forward. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Timeline_of_computing_1990-forward>. Acesso em: 5 jul. 2005.

TODD, E.; KEMP, E.; PHILLIPS, C. What makes a good user interface pattern language? In: THE FIFTH CONFERENCE ON AUSTRALASIAN USER INTERFACE, 2004, Dunedin, New Zealand. *Proceedings… Darlinghurst*, Australia, Australian Computer Society, v.28, p. 91-100, 2004. Disponível em: <http://doi.acm.org/10. 1145/976310.976322>. Acesso em: 5 jul. 2005.

TROLLTECH. *QT Designer*. 2000. [*Site* oficial do produto]. Disponível em: <http://www.trolltech.com/products/qt/designer.html>. Acesso em: 5 jul. 2005.

TROLLTECH. *QT.* 1994. [*Site* oficial do produto]. Disponível em: <http://www.trolltech.com/products/qt/>. Acesso em: 5 jul. 2005.

TSCHATER, M. Plataform Independent Software Development. *LinuxFocus*, [S.l.]: Oct. 2004. Disponível em: <http://ldp.hughesjr.com/linuxfocus/English/October2004/ article350.shtml>. Acesso em: 5 jul. 2005.

TUCK, M. The real history of the GUI. *Sitepoint*, Aug. 2001. Disponível em: <http://www.sitepoint.com/print/real-history-gui>. Acesso em: 5 jul. 2005.

UNIT record equipment. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Unit_record_equipment>. Acesso em: 5 jul. 2005.

UNIX. In: WIKIPEDIA the free encyclopedia. Disponível em:<http://en. wikipedia.org/ wiki/Unix>. Acesso em: 5 jul. 2005.

USABILITYNET. Interface Design Patterns. *UsabilityNet*: Design Overview, [ca. 2003]. Disponível em: <http://usabilitynet.org/tools/design_pattern.htm>. Acesso em: 5 jul. 2005.

USER interface. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Human-Machine_Interface>. Acesso em: 5 jul. 2005.

VAN DAM, A. Post-WIMP user interfaces. *Communications of the ACM*, New York, v. 40, n.2, p.63-67, feb.1997. ISSN: 0001-0782. Disponível em: <http://doi.acm.org/10. 1145/253671.253708>. Acesso em: 5 jul. 2005.

VAN DAM, A. Post-WIMP user interfaces. *Communications of the ACM*, v. 40, n. 2, p. 63-67, feb. 1997. ISSN: 0001-0782. Disponível em: <http://doi.acm.org/10.1145/ 253671.253708>. Acesso em: 5 jul. 2005.

VAN DAM, A. The shape of things to come. *SIGGRAPH Computer Graphics*, v. 32, n.1, p.42-44, feb. 1998. ISSN: 0097-8930. Disponível em: <http://doi.acm.org/ 10.1145/279389.279446>. Acesso em: 5 jul. 2005.

VAN DAM, A. User interfaces: disappearing, dissolving, and evolving. *Communications of the ACM*, New York, v. 44, n. 3, p. 50-52, mar. 2001. ISSN: 0001-0782. Disponível em: <http://doi.acm.org/10.1145/365181.365192>. Acesso em: 5 jul. 2005.

VAN DAM, A. User interfaces: disappearing, dissolving, and evolving. *Communications of the ACM, v.* 44, n. 3, p. 50-52, mar. 2001. ISSN: 0001-0782. Disponível em: <http://doi.acm.org/10.1145/365181.365192>. Acesso em: 5 jul. 2005.

VAN WELIE, M. Patterns for designers? In: CHI 2002 PATTERNS WORKSHOP: position paper. [S.l.]: [s.n.], [ca. 2002]. Disponível em: <http://www.welie.com/ patterns/chi2002-workshop/Patterns%20for%20Designers.pd f>. Acesso em: 5 jul. 2005.

VAN WELIE, M. *Patterns in Interaction Design*. 2003-2005. [Apresenta uma coleção de padrões de interfaces para a *web* e para o *desktop*]. Disponível em: <http://www.welie.com/patterns/>. Acesso em: 5 jul. 2005.

VAN WELIE, M.; TRÆTTEBERG, H. Interaction patterns in user interfaces. In: PATTERN LANGUAGES OF PROGRAMS CONFERENCE, 7., Allerton Park Monticello, Illinois, 2000. [S.l.]: [s.n.], [ca. 2000]. Disponível em: <http://citeseer.ist.psu.edu/ vanwelie00interaction.html>. Acesso em: 5 jul. 2005.

VAN WELIE, M.; VAN DER VEER, G. C.; ELIËNS, A. *Patterns as tools for user interface design*. [S.l.]: [s.n.], [ca. 2000]. Disponível em: <http://www.cs.vu.nl/ ~martijn/gta/docs/TWG2000.pdf>. Acesso em: 5 jul. 2005.

VANDERDONCKT, J. Automatic generation of a user interface for highly interactive business-oriented applications. In: CONFERENCE COMPANION ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1994, Boston, Massachusetts. New York, NY: ACM Press, 1994. p.123-124. ISBN:0-89791-651-4. Disponível em: <http://doi.acm.org/10.1145/259963.260104>. Acesso em: 5 jul. 2005.

VANNEVAR Bush. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Vannevar_Bush>. Acesso em: 5 jul. 2005.

VERING, M. More than just a pretty interface. Nov. 2004. [Entrevistador: Johannes Gillard]. *SAP INFO,* [S.l.], nov. 2004. Disponível em: <http://www. sapdesignguild. org/ editions/edition8/leading_article.asp>. Acesso em: 5 jul. 2005.

VLECK, T. V. (Ed.). *The multicians*. 2000-2005. [Apresenta a história e as características do sistema MULTICS]. Disponível em: <http://www.multicians.org/>. Acesso em: 5 jul. 2005.

WADLOW, T. A. The Xerox Alto Computer. *Byte*, [S.l], n.9, p.58-68, 1981. Disponível em: <http://www.aresluna.org/guidebook/articles/thexeroxaltocomputer>. Acesso em: 5 jul. 2005.

WALOSZEK, G.; EBERLEH, E. Introduction to user interface patterns at SAP. *Design Tidbits*, [S.l.], nov. 2003. Disponível em: <http://www.sapdesignguild.org/ community/design/patterns_sap.asp>. Acesso em: 5 jul. 2005.

WESSON, J.; COWLEY, L. Designing with patterns: possibilities and pitfalls. In: WORKSHOP ON SOFTWARE AND USABILITY CROSS-POLLINATION: THE ROLE OF USABILITY PATTERNS, INTERACT, 2., 2003, Zurique, Suíça. [S.l.]: [s.n.], [ca. 2003]. Disponível em: <http://wwwswt.informatik.uni-rostock.de/deutsch/Interact/5 WessonCowley.pdf>. Acesso em: 5 jul. 2005.

WICHARI, M. *GUIdebook*. 2002-2005. [Apresenta diversos recursos relacionados com interfaces gráficas]. Disponível em: <http://www.aresluna.org/guidebook/guis>. Acesso em: 5 de jul. 2005.

WIDGET (computing). In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Widget_%28computing%29>. Acesso em: 5 jul. 2005.

WIDGET TOOLKIT. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Widget_toolkit>. Acesso em: 5 jul. 2005.

WIECHA, C. et al. ITS: a tool for rapidly developing interactive applications. *ACM Transactions on Information Systems*, New York, v. 8, n. 3, p. 204-236, jul. 1990. ISSN 1046-8188. Disponível em: <http://doi.acm.org/10.1145/98188.98194>. Acesso em: 5 jul. 2005.

WIECHA, C.; SZEKELY, P. Transforming the UI for anyone. anywhere: enabling an increased variety of users, devices, and tasks through interface transformations. In: CHI '01 EXTENDED ABSTRACTS ON HUMAN FACTORS IN COMPUTING SYSTEMS, 2001, Seattle, Washington. New York, NY: ACM Press, 2001. p. 483-484. ISBN: 1-58113-340-5. Disponível em: <http://ndoi.acm.org/10.1145/634067. 634344>. Acesso em: 5 jul. 2005.

WIMP (computing). In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/WIMP_%28computing%29>. Acesso em: 5 jul. 2005.

WINDOWS 1.0. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Windows_1.0>. Acesso em: 5 jul. 2005.

WINDOWS 2.0. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Windows_2.0>. Acesso em: 5 jul. 2005.

WINDOWS 3.x. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Windows_3.x>. Acesso em: 5 jul. 2005.

WORDSTAR. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/WordStar>. Acesso em: 5 jul. 2005.

WXWIDGETS. 1992. [*Site* oficial do produto]. Disponível em: <http://www.wxwidgets.org/>. Acesso em: 5 jul. 2005.

X WINDOW System. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/X_Window_System>. Acesso em: 5 jul. 2005.

X.ORG FOUNDATION. X11 Technologies. [ca. 2004]. [Apresenta as tecnologias e os padrões em torno do X Windows; é responsável pela implementação de referência do X Windows]. Disponível em: <http://x.org/>. Acesso em: 5 jul. 2005.

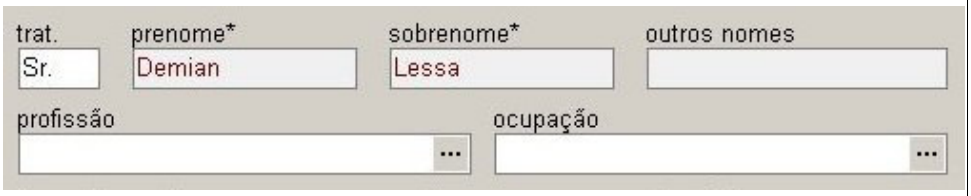
XEROX Alto. In: WIKIPEDIA the free encyclopedia. Disponível em: <http://en.wikipedia.org/wiki/Xerox_Alto>. Acesso em: 5 jul. 2005.

ZIEGLER, J. Interactive techniques. *ACM Computing Surveys,* New York, v. 28, n. 1, p.185-187, mar.1996. ISSN: 0360-0300. Disponível em: <http://doi.acm.org/ 10.1145/ 234313.234392>. Acesso em: 5 jul. 2005.

APÊNDICE A - Coleção de Padrões

Este apêndice apresenta a coleção de padrões de interfaces desenvolvida como parte deste trabalho, na forma tabular, como deve ser utilizada por profissionais de interfaces. Seja no uso prático da coleção para o trabalho de análise, na comparação de seu idioma ou de seus padrões com outras coleções, ou simplesmente na sua leitura, a utilização da coleção sob a forma tabular favorece a rápida referência aos padrões, como a eventual consulta a qualquer de suas características específicas.

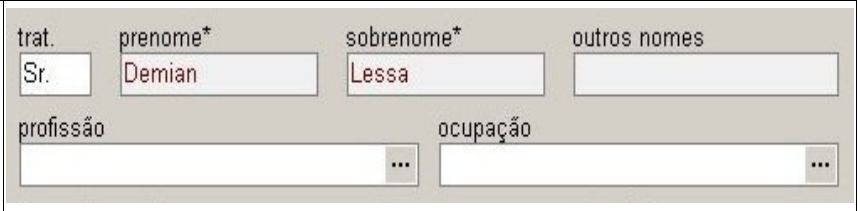
Nas páginas que seguem, a coleção de padrões de interface é apresentada através da forma tabular, formato original previsto para seu uso.

Nome	Identificador (também conhecido como Título ou Rótulo)
Grupo	Indicador Visual
Problema	O usuário não sabe o que representa, no modelo conceitual, o conteúdo de um determinado artefato.
Contexto	O usuário precisa identificar, de forma rápida e inequívoca, o que representa o conteúdo associado a um determinado artefato. Sabendo o que esse conteúdo representa, o usuário poderá interagir corretamente com seu artefato.
Solução	Acrescentar um título ou texto explicativo que apresente ao usuário o significado do conteúdo do artefato.
Benefícios	Maior agilidade na interação do usuário com a interface. Elevação na qualidade e consistência da informação preenchida pelo usuário através da interface.
Exemplo	 O fragmento de UI acima é composto por seis widgets e seis títulos. Os títulos são localizados acima e alinhados à esquerda de seus widgets associados. O objetivo dos títulos é identificar o significado do conteúdo que o widget associado exibe. No exemplo acima, “trat.” (tratamento), “prenome”, “sobrenome”, “outros nomes”, “profissão” e “ocupação”.

Indicador Visual: Identificador.

Nome	Obrigatoriedade
Grupo	Indicador Visual
Problema	O usuário não sabe quais artefatos representam informações obrigatórias e qu representam informações opcionais.
Contexto	O usuário precisa identificar prontamente na interface quais artefatos represent informação obrigatória e quais representam informação opcional. Sabendo que d conteúdo é obrigatório, o usuário irá procurar preenchê-lo.
Solução	Artefatos cujos dados são de preenchimento obrigatório devem ser destaca através de algum identificador visual de obrigatoriedade. Uma forma bem conhecida representação de obrigatoriedade emprega um '*' (asterisco) nos identificadores artefatos cujos dados são de preenchimento obrigatório.
Benefícios	Maior agilidade na interação do usuário com a interface. Menos erros por omissão informação essencial.
Exemplo	 O fragmento de UI acima é composto por dois conteúdos obrigatórios e qua opcionais. Os títulos dos conteúdos obrigatórios são decorados com um asterisco final. No exemplo acima, os conteúdos “prenome” e “sobrenome” são am obrigatórios.

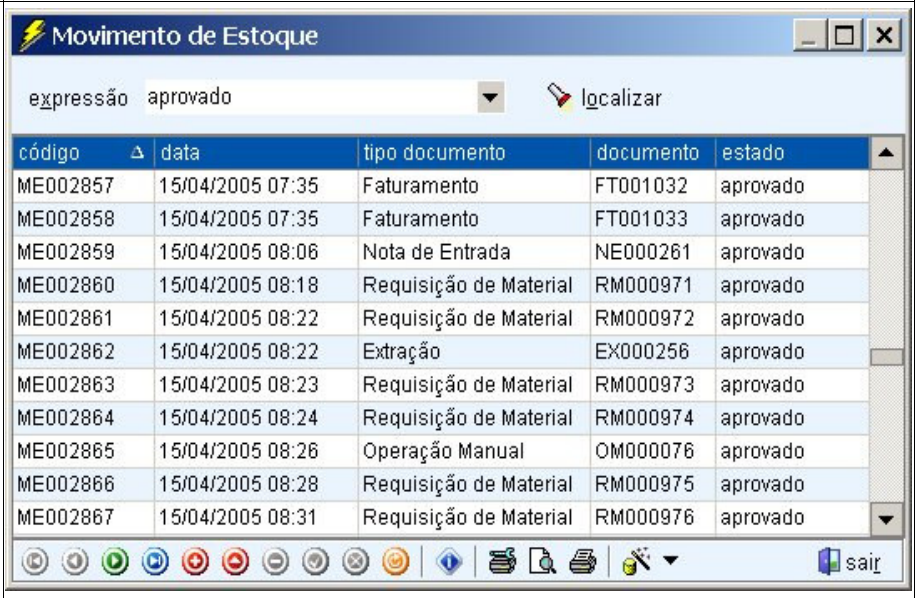
Indicador Visual: Obrigatoriedade.

Nome	Somente-Leitura
Grupo	Indicador Visual
Problema	O usuário não sabe quais artefatos representam informações que não podem ser alteradas.
Contexto	O usuário precisa identificar prontamente na interface quais artefatos representam informação que não pode ser alterada. Sabendo que dado conteúdo é somente-leitura, o usuário não tentará interagir com seu artefato desnecessariamente.
Solução	Artefatos cujos dados não podem ser alterados devem ser destacados através de algum identificador visual. Uma forma bem conhecida de representação de artefatos somente-leitura emprega cores de fundo e de texto características para o artefato.
Benefícios	Maior agilidade na interação do usuário com a interface.
Exemplo	 O fragmento de UI acima apresenta três conteúdos somente-leitura. As cores de fundo do texto dos widgets somente-leitura foram alteradas em relação ao padrão, fundo branco e texto preto, para fundo cinza claro e texto marrom.

Indicador Visual: Somente-Leitura.

Nome	Foco (também conhecido como Seleção)
Grupo	Indicador Visual
Problema	O usuário não identifica prontamente qual o artefato da interface está com o foco do teclado, isto é, recebe a entrada do teclado.
Contexto	O usuário precisa identificar prontamente na interface qual dos artefatos está com o foco do teclado. A identificação do artefato com o foco do teclado serve como um alerta visual, reduzindo a quantidade de erros causados pelo preenchimento equivocado de informações.
Solução	O artefato com o foco deverá ser destacado na interface através da mudança no seu estado visual (cor de fundo, cor do texto ou outro aspecto visual) ou no estado visual de seu identificador (cor do texto do identificador). Na perda do foco do teclado, o artefato deve ter seu estado visual restaurado ao original de modo a garantir que apenas um dos artefatos da interface possua destaque visual.
Benefícios	Maior agilidade na interação do usuário com a interface. Menor quantidade de erros.
Exemplo	 O fragmento de UI acima destaca visualmente um widget dos demais. Esse widget, em azul claro, identifica o artefato com o foco. Nos casos em que o padrão somente-leitura for empregado por um widget, ele jamais receberá o foco.

Indicador Visual: Foco.

Nome	Zebrado (também conhecido como Alternância Visual)
Grupo	Indicador Visual
Problema	O usuário é apresentado a um conjunto de informações em forma tabular e precisa acompanhar os valores em cada linha sem equívoco.
Contexto	O usuário precisa distinguir uma linha de outra facilmente, sem a necessidade de utilizar outros auxílios visuais como réguas ou apontadores. Além disso, deve poder ler o conteúdo de cada linha sem se perder ao longo da leitura.
Solução	A informação é apresentada de modo a distinguir visualmente uma linha da outra através do uso alternado de cores na apresentação. Linhas são agrupadas em blocos alternados e as linhas em cada bloco são apresentadas com a mesma cor de fundo.
Benefícios	Agilidade na visualização da informação.
Exemplo	 A interface apresentada acima ilustra bem as características do padrão zebrado. A grade exibe diversas linhas, agrupadas uma a uma, onde cada grupo é exibido com a cor de fundo em branco ou azul claro, alternadamente.

Indicador Visual: Alternância de Cores.

Nome	Barra de Controles
Grupo	Controle
Problema	O usuário precisa executar rapidamente e de forma rotineira uma série de tarefas baseadas nos dados apresentados pela UI.
Contexto	O usuário precisa executar tarefas comuns sobre os dados como: incluir, excluir e alterar os dados; navegar entre os dados; exibir meta-informação; imprimir ou exportar dados, etc. A combinação de artefatos numa Barra de Controles para disparar a execução de tais tarefas permite a padronização do acesso às mais comuns operações sobre os dados.
Solução	Acrescentar uma Barra de Controles através da combinação de uma série de artefatos independentes. Cada artefato é associado a uma única operação, que está acessível a depender do estado dos dados apresentados pelos artefatos da UI. Uma forma bem aceita de Barra de Controles envolve o uso de botões, identificados por imagens, combinados lado a lado para formar a barra, que é então apresentada pela interface em uma das extremidades da tela, normalmente na parte inferior.
Benefícios	Maior agilidade na interação do usuário com a interface. Maior consistência na operação das interfaces.
Exemplo	 O fragmento de UI acima apresenta uma Barra de Controles formada por diversos botões, cada um identificado por uma imagem distinta. Alguns dos botões estão acessíveis e outros não; essa distinção apresenta-se através do uso de cores- quando o botão está em tons de cinza, ele não está disponível. Especificamente nesta barra, os botões têm o seguinte significado (da esquerda para a direita): voltar ao primeiro registro, voltar ao registro anterior, avançar ao próximo registro, avançar ao último registro, incluir um novo registro, editar o registro atual, excluir o registro atual, salvar as alterações, cancelar as alterações, recarregar os dados, exibir informações dos meta-dados, configurar impressão, visualizar impressão, imprimir e exportar para um formato (como HTML, XML, Excel, etc).

Controle: Barra de Controle.

Nome	Menu Contextual
Grupo	Controle
Problema	O usuário precisa, de forma excepcional, alterar a forma de apresentação dos dados apresentados pela UI.
Contexto	O usuário precisa, esporadicamente, alterar a quantidade, ordem ou disposição dos dados apresentação pela UI. A utilização de um menu contextual para acessar as operações permite que usuários avançados realizem suas tarefas sem interferir com a forma padrão de interação da UI.
Solução	Acrescentar um menu contextual ao artefato que permite a configuração de sua apresentação.
Benefícios	Maior flexibilidade e agilidade na operação dos usuários avançados com a interface.
Exemplo	 O fragmento de UI acima apresenta um menu contextual que permite selecionar as colunas que devem estar visíveis no interface; além disso, conta também com uma funcionalidade avançada para calcular o melhor ajuste visual para a apresentação dos dados na grade. O menu de contexto apresentado está acessível através do clique do botão direito do mouse ou do pressionamento da tecla de menu de contexto.

Controle: Menu Contextual.

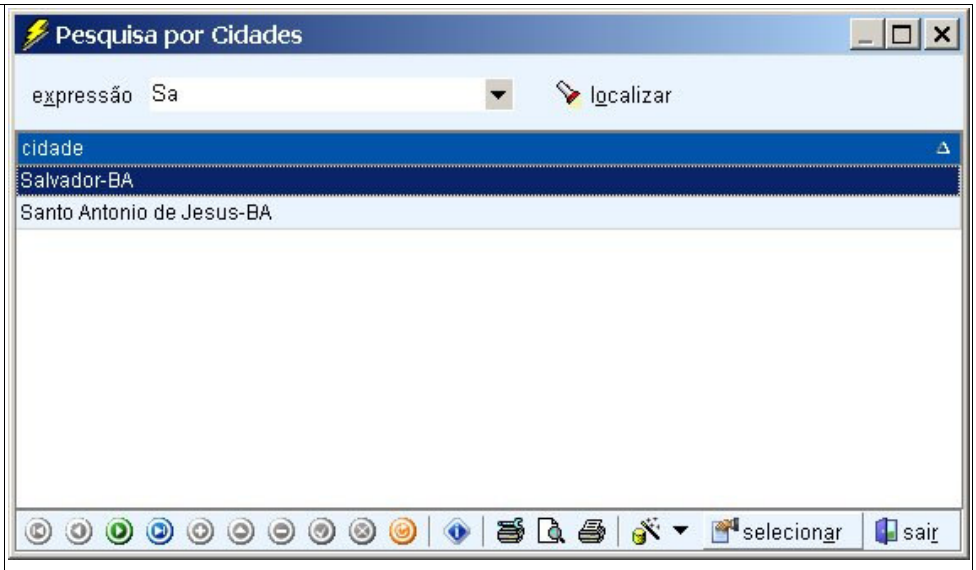
Nome	Lookup por Delegação (Delegate Lookup)
Grupo	Acesso a Dados
Problema	O usuário precisa selecionar um único valor mas a lista de possíveis valores é muito grande para ser carregada e exibida para ele selecionar.
Contexto	O usuário precisa selecionar o valor desejado mas a quantidade de possíveis valores é muito grande. Assim, para que o usuário identifique e selecione o valor desejado, é preciso quebrar a operação em duas etapas- pesquisa e seleção.
Solução	O widget responsável pela exibição do valor selecionado permite o acionamento de um artefato de pesquisa, que suporta o uso de critérios para restringir a quantidade de dados na pesquisa. O usuário então identifica, seleciona um valor e retorna. Em seguida, o widget responsável processa e exibe o valor selecionado. Uma forma de implementar o lookup por delegação é através de um widget de caixa de edição com um botão anexado. Ao clicar o botão, um dos padrões de grade de pesquisa é acionado, a pesquisa corre por conta da grade e o valor resultante é retornado. Ou seja, a seleção propriamente dita é delegada a um outro artefato.
Benefícios	Maior agilidade na interação do usuário com a interface, uma vez que menos dados são trafegados e, portanto, a resposta do sistema é mais rápida.
Exemplo	

	No primeiro exemplo, um lookup é utilizado localmente num formulário para selecionar um dos possíveis valores do sexo de uma pessoa. Os valores são previamente conhecidos e a quantidade é pequena. No segundo exemplo, um lookup é utilizado localmente em uma grade para selecionar um dos possíveis valores da parte de uma avaliação. Apesar da quantidade de possíveis partes de uma avaliação ser variável, o conhecimento do modelo conceitual permite antecipar que essa quantidade será relativamente pequena, mesmo em se passando um longo tempo.

Nome	Lookup por Delegação (Delegate Lookup)
Grupo	Acesso a Dados
Problema	O usuário precisa selecionar um único valor mas a lista de possíveis valores é muito grande para ser carregada e exibida para ele selecionar.
Contexto	O usuário precisa selecionar o valor desejado mas a quantidade de possíveis valores é muito grande. Assim, para que o usuário identifique e selecione o valor desejado, é preciso quebrar a operação em duas etapas- pesquisa e seleção.
Solução	O widget responsável pela exibição do valor selecionado permite o acionamento de um artefato de pesquisa, que suporta o uso de critérios para restringir a quantidade de dados na pesquisa. O usuário então identifica, seleciona um valor e retorna. Em seguida, o widget responsável processa e exibe o valor selecionado. Uma forma de implementar o lookup por delegação é através de um widget de caixa de edição com um botão anexado. Ao clicar o botão, um dos padrões de grade de pesquisa é acionado, a pesquisa corre por conta da grade e o valor resultante é retornado. Ou seja, a seleção propriamente dita é delegada a um outro artefato.
Benefícios	Maior agilidade na interação do usuário com a interface, uma vez que menos dados são trafegados e, portanto, a resposta do sistema é mais rápida.
Exemplo	

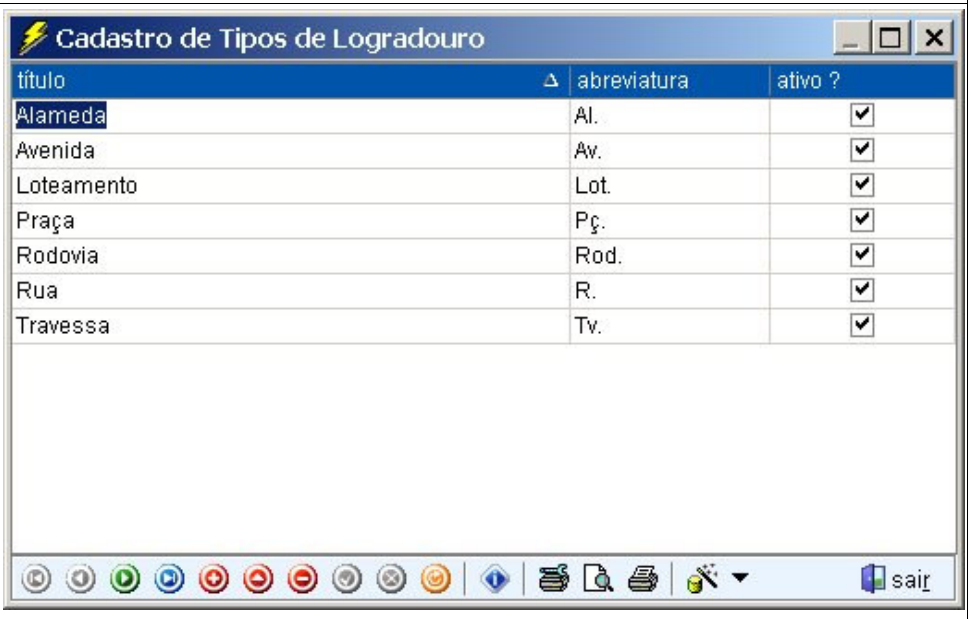
Acesso a dados: Lookup por Delegação.

Exemplo (cont.)

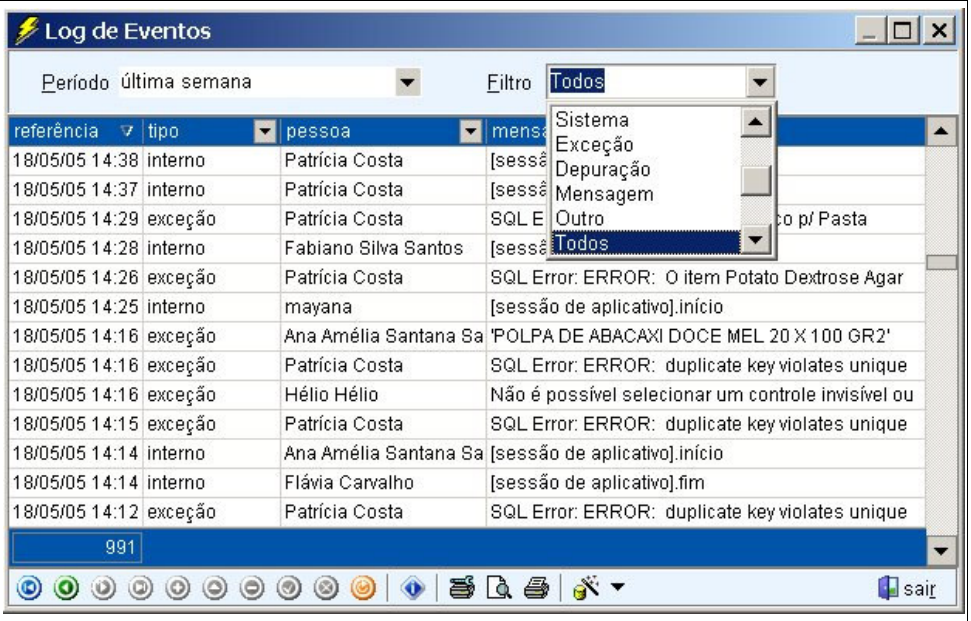


As três telas acima ilustram a típica seqüência envolvida no uso de um lookup por delegação. Na alteração dos dados do lookup (nesse caso trata-se da seleção da cidade associada ao bairro “Pernambués”), um botão está disponível para acionar o artefato de pesquisa. Uma vez acionado, o controle do aplicativo passa para esse artefato- no caso em particular, uma grade de pesquisa livre. O usuário entra com a expressão de pesquisa e, quando identificar a informação desejada, a seleciona na grade e aciona o botão selecionar. Na tela final, o lookup delegate já recebeu o valor selecionado pela grade e o está exibindo na UI.

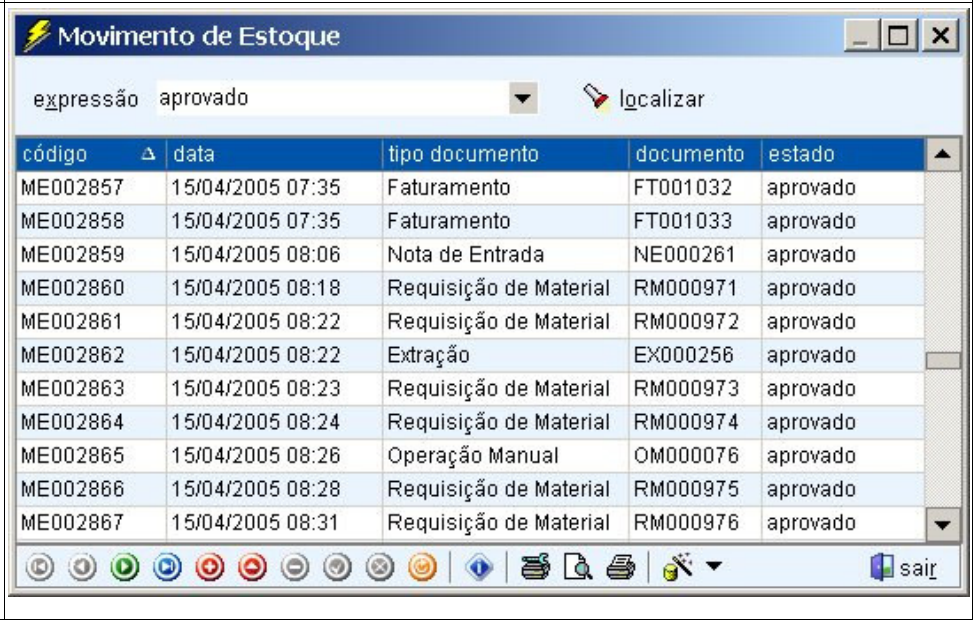
Acesso a Dados: Lookup por Delegação (cont.).

Nome	Grade (também conhecido como Tabela ou Planilha)
Grupo	Editor de Dados
Problema	O usuário precisa acessar um conjunto de informações, de estrutura simples e homogênea, individualmente e através de suas partes.
Contexto	O usuário precisa ter a visão do conjunto de informações ao mesmo tempo em que precisa ter acesso a cada informação individualmente através de suas (poucas) partes. Por exemplo, o acesso ao conjunto dos possíveis tipos de logradouro deve permitir o acesso a cada tipo de logradouro através do título do tipo de logradouro e de sua abreviatura.
Solução	A informação é apresentada de forma tabular onde linhas representam unidades independentes de informação e colunas, propriamente identificadas, representam as partes de cada informação. Partes conceitualmente mais importantes da informação devem ter um maior destaque (precedência ou outro destaque visual) enquanto partes conceitualmente associadas devem ser agrupadas (sucessão ou outra forma de agrupamento visual).
Benefícios	Agilidade na manipulação de conjuntos de informação.
Exemplo	 A interface apresentada acima ilustra bem as características da grade. Os dados estão dispostos de forma claramente tabular, em linhas e colunas. Cada coluna utiliza um rótulo para identificar o conteúdo apresentado. O título do editor, na barra superior, identifica a informação trabalhada- nesse caso, tipos de logradouro. O editor ainda utiliza-se do padrão barra de controles e assim permite, com um idioma simples e intuitivo, que a grade seja navegada e seus dados manipulados.

Editor de Dados: Grade.

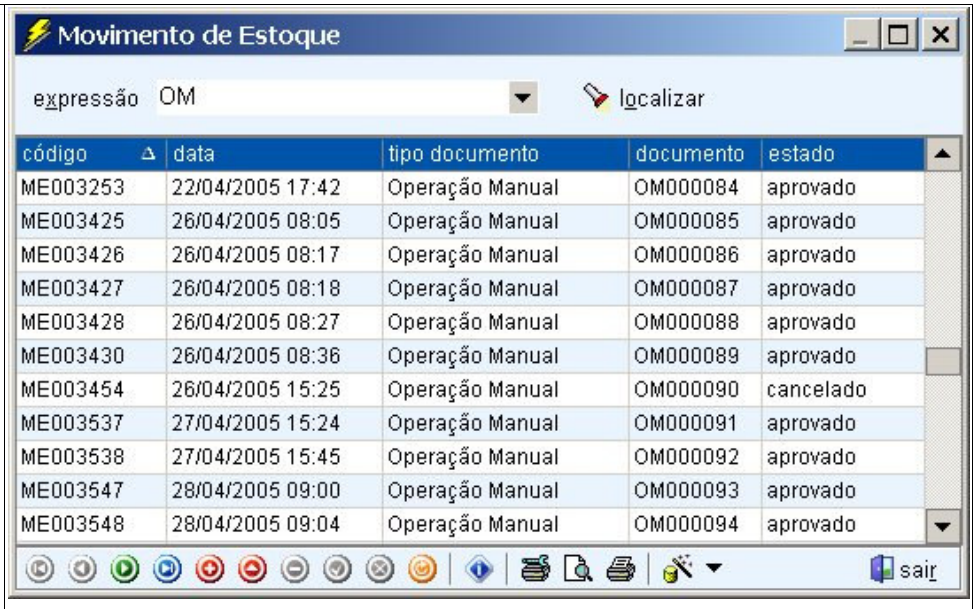
Nome	Grade Contextualizada (também conhecido como Grade com Filtro)
Grupo	Editor de Dados
Problema	O usuário precisa acessar um conjunto de informações, de estrutura simples e homogênea, individualmente e através de suas partes. O conjunto de informações está naturalmente agrupado em uma ou mais de suas partes e a contextualização prévia das informações é essencial.
Contexto	O usuário precisa ter a visão do conjunto de informações já contextualizado, ao mesmo tempo em que precisa ter acesso a cada informação individualmente através de suas (poucas) partes. Por exemplo, em um log de eventos, as informações são naturalmente agrupadas em período (quando o evento ocorreu) e tipo de evento; o acesso ao conjunto de eventos deve permitir a contextualização do período e do tipo de evento.
Solução	Esse padrão estende o padrão da grade permitindo a seleção de um ou mais valores para contextualizar a apresentação dos dados na grade. Em geral, acrescenta-se uma região de contextualização na UI onde os widgets contendo os possíveis contextos são apresentados.
Benefícios	Redução no tamanho do conjunto de dados a manipular, conferindo maior agilidade na manipulação das informações.
Exemplo	 A interface apresentada acima ilustra bem as características da grade com filtro. A diferença marcante entre este padrão e a grade simpes está na inclusão da região para a contextualização dos dados apresentados. No caso específico, os dados estão contextualizados em relação a duas dimensões: período em que os eventos ocorreram e tipo de evento. São apresentados na grade todos os eventos ocorridos ao longo da última semana.

Editor de Dados: Grade Contextualizada.

Nome	Grade de Pesquisa Livre
Grupo	Editor de Dados
Problema	O usuário precisa acessar um conjunto de informações, de estrutura simples e homogênea, individualmente e através de suas partes. O conjunto de informações não está particularmente bem agrupado através de qualquer de suas partes, mas é preciso contextualizar a informação através de uma ou mais de suas partes.
Contexto	O usuário precisa ter a visão do conjunto de informações já contextualizado, ao mesmo tempo em que precisa ter acesso a cada informação individualmente através de suas (poucas) partes. Por exemplo, em um movimento de estoque, o usuário digita o estado dos movimentos, o documento que gerou o movimento ou o código do movimento para ser então apresentado ao conjunto de dados de movimento de estoque já contextualizados.
Solução	Esse padrão estende o padrão da grade permitindo a entrada de um texto livre para contextualizar a apresentação dos dados na grade. Em geral, acrescenta-se uma região de contextualização na UI onde um ou mais widgets são acrescentados para controlar a entrada da expressão de contextualização.
Benefícios	Maior identificação com o modelo conceitual, já que a contextualização utiliza expressões dependentes do modelo. Redução no tamanho do conjunto de dados a manipular, conferindo maior agilidade na manipulação das informações.
Exemplo	

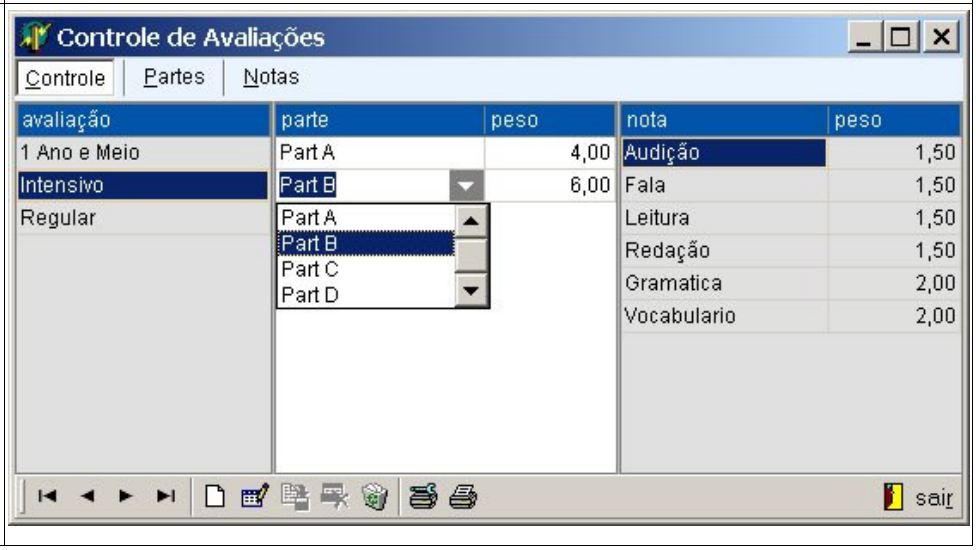
Editor de Dados: Grade de Pesquisa Livre.

Exemplo (cont.)



As interfaces apresentadas acima ilustram bem as características da grade de pesquisa livre. A diferença marcante entre este padrão e a grade simpes está na inclusão da região para a contextualização dos dados apresentados. No primeiro caso, os dados estão contextualizados em relação à expressão “aprovado” enquanto no segundo caso, em relação à expressão “OM”. Conhecendo o modelo conceitual, sabe-se que a expressão de pesquisa “aprovado” só faz sentido no contexto do “estado” do movimento e que a expressão de pesquisa “OM” só faz sentido no contexto do “documento” do movimento. O aplicativo então utiliza a expressão e contextualiza a visão dos movimentos de estoque conforme esperado.

Editor de Dados: Grade Pesquisa Livre (cont.).

Nome	Grade Mestre-Detalhe (também conhecido como Grade com Hierárquica)
Grupo	Editor de Dados
Problema	O usuário precisa acessar simultaneamente mais de um conjunto de informações relacionadas. Cada um dos conjuntos possui estrutura simples e homogênea (mas heterogênea entre os conjuntos), e sua informação pode ser acessada individualmente e através de suas partes.
Contexto	O usuário precisa ter a visão dos conjuntos de informações, das relações entre eles, ao mesmo tempo em que precisa ter acesso a cada informação individualmente através de suas (poucas) partes. Por exemplo, o acesso a diversas avaliações cujas partes são segmentadas em tipos.
Solução	Esse padrão estende o padrão da grade incluíndo uma ou mais grades na UI- tantas quantas forem as relações de dependência. Na sua forma mais comum, as grades são dispostas lado a lado, horizontal ou verticalmente, e a disposição visual das grades determina sua relação hierárquica. Quando uma informação em uma grade é selecionada, as grades imediatamente abaixo dela na hierarquia sincronizam suas apresentações. Quando uma grade é selecionada, as demais grades mudam seus estados para indicar que estão indisponíveis (somente-leitura); todos os widgets da UI que dependem de contexto são sincronizados para trabalhar apenas no contexto da grade selecionada.
Benefícios	Maior identificação com o modelo conceitual, já que a hierarquia de grades identifica relações de dependência no modelo conceitual. Redução no tamanho do conjunto de dados a manipular, conferindo maior agilidade na manipulação das informações.
Exemplo	

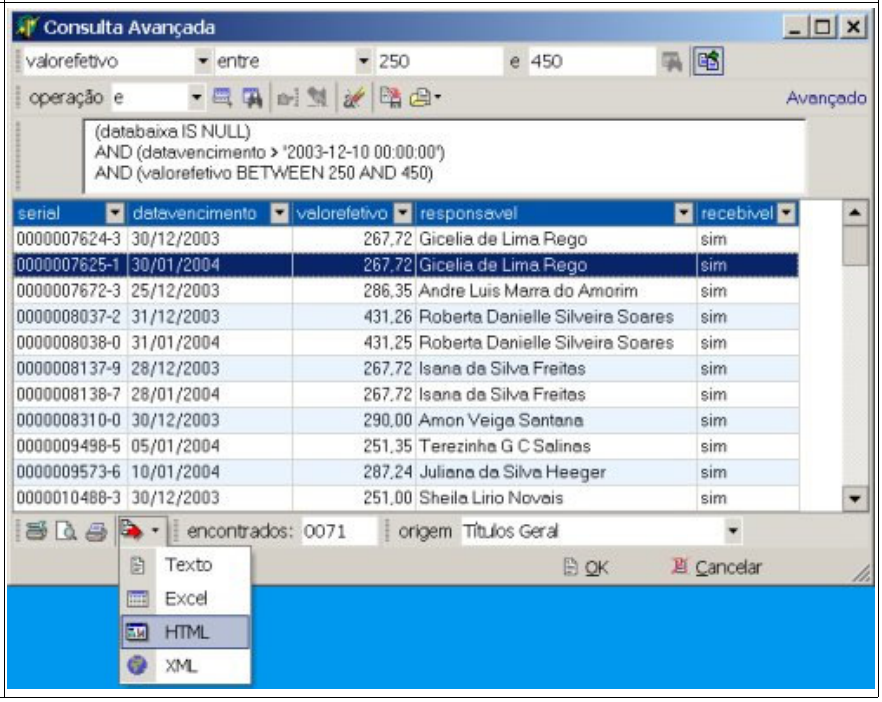
Editor de Dados: Grade Mestre-Detalhes.

Exemplo (cont.)	A interface apresentada acima ilustra bem as características da grade hierárquica. A diferença marcante entre este padrão e a grade simples está na inclusão de mais grades, verticalmente na UI. No caso específico, ao selecionar uma avaliação, suas partes são exibidas e, ao selecionar uma parte, os tipos de notas são exibidos. As grades com as quais o usuário não está manipulando ficam sombreadas para indicar visualmente sua indisponibilidade. A barra de controles está sempre ligada à grade ativa.

Editor de Dados: Grade Mestre-Detalhe (cont.).

Nome	Grades Alternativas (também conhecido como Múltiplas Grades)
Grupo	Editor de Dados
Problema	O usuário precisa acessar simultaneamente mais de um conjunto de informações não relacionadas. Cada um dos conjuntos possui estrutura simples e homogênea (mas heterogênea entre os conjuntos), e sua informação pode ser acessada individualmente e através de suas partes. São, em geral, conjuntos com pouca informação que tendem a crescer muito pouco com o tempo.
Contexto	O usuário precisa ter a visão dos conjuntos de informações, ao mesmo tempo em que precisa ter acesso a cada informação individualmente através de suas (poucas) partes. Por exemplo, o acesso a bancos, descontos, bairros e profissões.
Solução	Esse padrão estende o padrão da grade permitindo o acesso alternado a várias grades- tantas quantas forem necessárias. Na sua forma mais comum, um widget é utilizado como índice visual para as diversas grades. Ao selecionar um item nesse índice, a grade associada é apresentada. Todos os widgets da UI que dependem de contexto são sincronizados para trabalhar apenas no contexto da grade selecionada.
Benefícios	Agilidade na manipulação de vários conjuntos de informação não relacionados.
Exemplo	 A interface apresentada acima ilustra bem as características das grades alternativas. A diferença marcante entre este padrão e a grade simpes está na inclusão da região para a seleção da grade apresentada. No caso específico, o índice é apresentado como uma lista de textos e imagens. Ao selecionar um dos itens no índice, a grade correspondente é apresentada. No caso em particular, a grade selecionada é a de bancos.

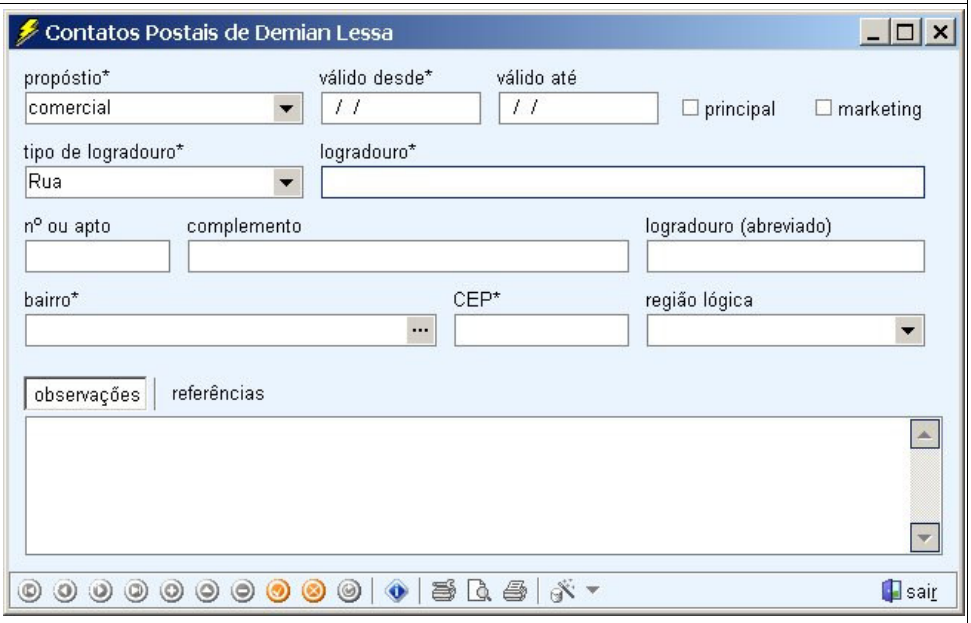
Editor de Dados: Grades Alternativas.

Nome	Grade de Pesquisa Avançada
Grupo	Editor de Dados
Problema	O usuário precisa acessar um conjunto de informações, de estrutura simples e homogênea, individualmente e através de suas partes. O conjunto de informações não está particularmente bem agrupado através de qualquer de suas partes e não há contextualização possível. No entanto, o usuário precisa fixar os valores parciais (uma ou mais partes) da informação que deseja acessar.
Contexto	Como não é possível antecipar uma contextualização ideal, o usuário precisa ter um mecanismo de filtrar o conjunto de informações com o qual deseja trabalhar de modo dinâmico. Por exemplo, dado um conjunto das partes de uma informação, o usuário poderá definir critérios em relação a essas partes para que somente as informações que satisfaçam esses critérios sejam apresentadas na UI.
Solução	Esse padrão estende o padrão da grade de pesquisa livre permitindo ao usuário selecionar as partes e os critérios que deseja utilizar para apresentar as informações na grade. Em geral, acrescenta-se uma região na UI onde um conjunto de widgets é acrescentado para que o usuário possa definir seus critérios de contextualização.
Benefícios	Maior identificação com o modelo físico, já que a contextualização pode utilizar elementos do modelo físico de dados. Flexibilidade na seleção do conjunto de dados a trabalhar.
Exemplo	

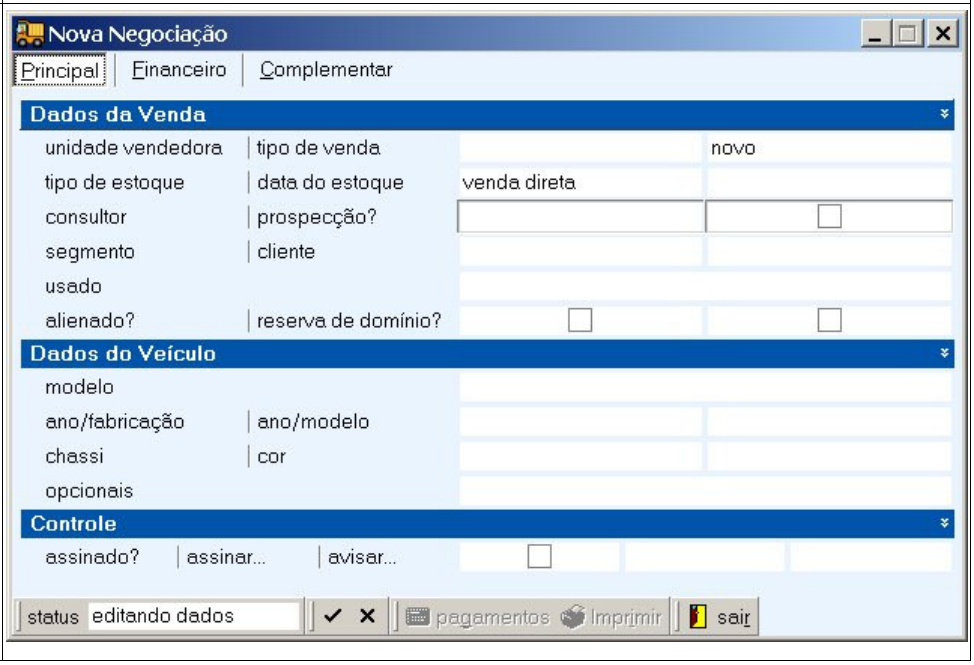
Editor de Dados: Grade de Pesquisa Avançada.

Exemplo (cont.)	A grade de pesquisa avançada apresentada acima demonstra a complexidade com a qual o usuário poderá ter que lidar ao utilizar um padrão como esse. Ele precisa definir antecipadamente os dados que deseja acessar- isso é representado na UI pelo widget identificado por “origem”, na parte inferior da barra de controles. O usuário então seleciona uma parte da informação, através do widget na parte superior esquerda da UI. Em seguida, utilizando os widgets à direita desse, define o critério para filtrar essa parte da informação. E repete isso tantas vezes quanto necessário. Enquanto o usuário interage com a UI, os comandos resultantes são montados pelo aplicativo. A exibição desses comandos não faz parte do padrão e pode ser excluída da UI sem prejuízos ao resultado final. Como esse é um padrão normalmente usado por usuários avançados, a exibição dos pode ser desejável.

Editor de Dados: Grade de Pesquisa Avançada (cont.).

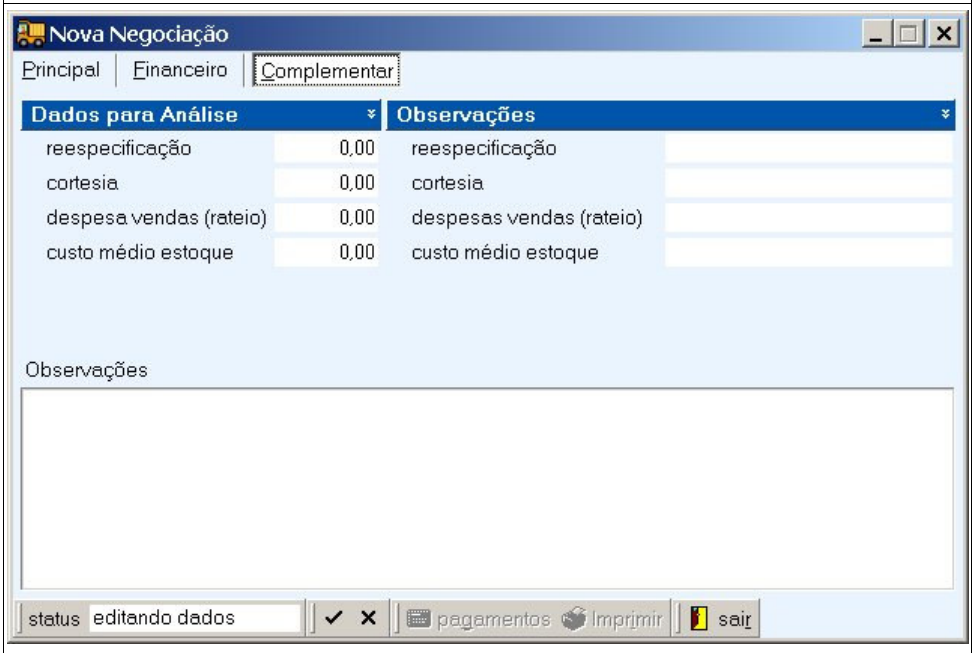
Nome	Formulário (também conhecido como Ficha)
Grupo	Editor de Dados
Problema	O usuário precisa acessar uma informação complexa, estruturada em partes.
Contexto	O usuário precisa ter uma visão geral de uma informação complexa, ao mesmo tempo em que precisa ter acesso a cada uma de suas (muitas) partes. Por exemplo, o acesso a: tipo de logradouro, logradouro, complemento, bairro, cidade, estado, país e CEP (partes) de um endereço (informação complexa).
Solução	Cada parte da informação é apresentada através de artefato e identificador próprios. Partes conceitualmente mais importantes da informação possuem maior destaque (precedência ou outro destaque visual); partes conceitualmente associadas são agrupadas (sucessão ou outra forma de agrupamento visual).
Benefícios	Lidar com a informação complexa como um todo aumenta o entendimento do modelo conceitual.
Exemplo	 A interface apresentada acima ilustra bem as características do formulário. Várias partes de uma mesma informação foram dispostas visualmente como uma ficha. Os dados foram dispostos naturalmente na ordem em que normalmente são obtidos das pessoas. Partes associadas foram mantidas próximas, como “tipo de logradouro” e “logradouro”, “válido desde” e “válido até”.

Editor de Dados: Formulário.

Nome	Formulário Segmentado
Grupo	Editor de Dados
Problema	O usuário precisa acessar uma informação complexa, estruturada em partes e o número dessas partes é muito grande.
Contexto	O usuário precisa ter a visão do conjunto de informações ao mesmo tempo em que precisa ter acesso a cada informação individualmente através de suas (poucas) partes. Por exemplo, o acesso ao conjunto dos possíveis tipos de logradouro deve permitir o acesso a cada tipo de logradouro através do título do tipo de logradouro e de sua abreviatura.
Solução	A informação é segmentada em partes conceitualmente associadas e, através da utilização de um mecanismo seletor de segmentos, é apresentada ao usuário de forma que apenas o conteúdo de um único segmento é visível a cada instante. A seleção de um segmento resulta na apresentação dos artefatos associados àquele segmento como se fossem um formulário independente. Formas bem conhecidas de segmentar um formulário utilizam widgets como “page control” ou “tab sheet”.
Benefícios	Mantendo os mesmos benefícios do padrão formulário, permite trabalhar com informações bem mais complexas.
Exemplo	

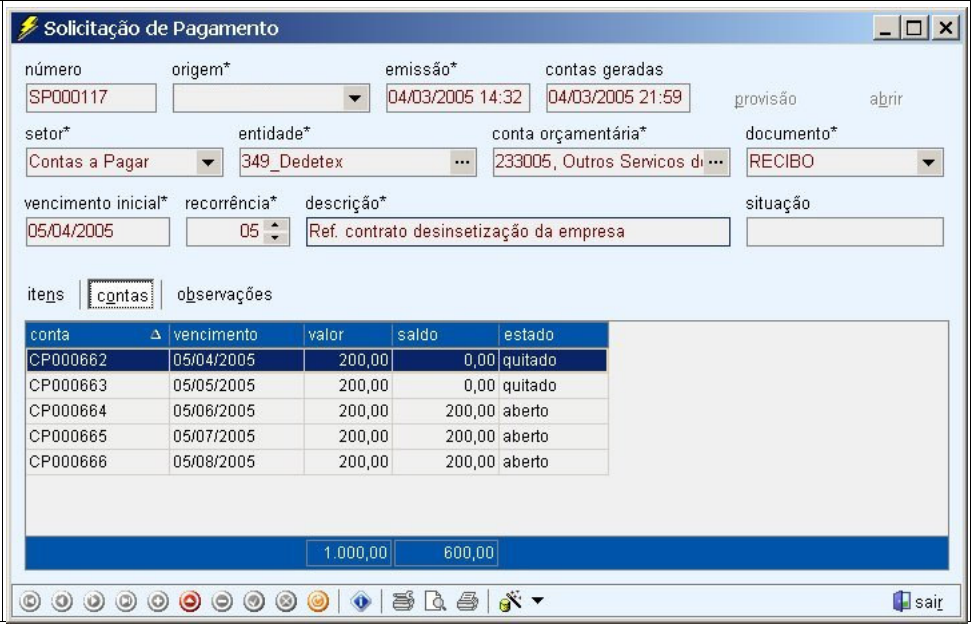
Editor de dados: Formulário Segmentado.

Exemplo (cont.)



A variação segmentada do formulário é bem representada nas imagens acima. Na primeira, os dados principais de uma negociação; na segunda, os dados financeiros e, na terceira, dados complementares da negociação. As partes da informação foram segmentadas de acordo com o modelo e, em cada segmento, tem-se a impressão de se estar vendo um formulário independente. Em cada segmento as informações foram segmentadas visualmente, conforme previsto no padrão formulário.

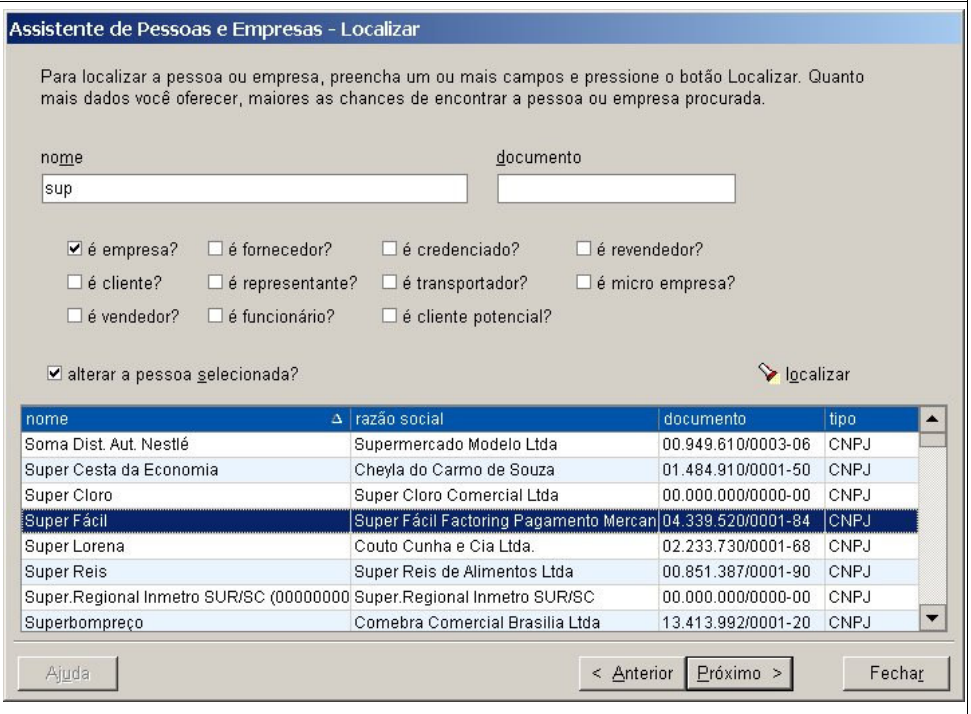
Editor de Dados: Formulário Segmentado (cont.).

Nome	Formulário Mestre-Detalhe (também conhecido como Formulário Hierárquico)
Grupo	Editor de Dados
Problema	O usuário precisa acessar uma informação complexa, estruturada em partes, que possui outras informações dependentes que também precisam ser exibidas. Essas informações dependentes possuem estrutura simples e homogênea, e devem ser acessadas individualmente, através de suas partes.
Contexto	O usuário precisa ter simultaneamente a visão da informação complexa assim como das informações dependentes. Por exemplo, as informações básicas de um pagamento (informação complexa) e as contas geradas para quitar esse pagamento (informações dependentes).
Solução	A informação é apresentada parcialmente num formulário e parcialmente numa grade ou em um artefato que permita a exibição alternada de múltiplas grades, no caso de ser preciso apresentar múltiplos conjuntos de informações dependentes. Formas conhecidas de apresentar o formulário hierárquico utilizam um formulário com um widget segmentador (como “page control” ou “tab sheet”) na parte inferior da UI para apresentar cada um dos conjuntos de informação dependente.
Benefícios	Combinando os benefícios dos padrões formulário e grade, permite-se trabalhar com informações complexas com vários conjuntos de informações dependentes. Maior identificação com o modelo conceitual, já que a associação entre informações expõe detalhes acerca do modelo conceitual.
Exemplo	

Editor de Dados: Formulário Mestre-Detalhe.

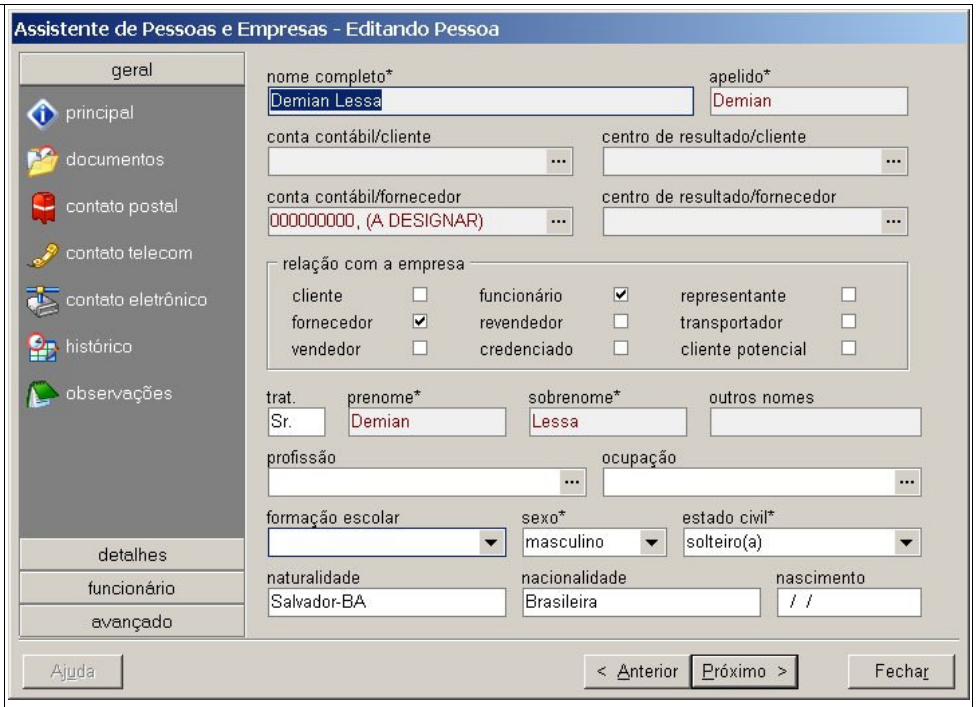
Exemplo (cont.)	A interface apresentada acima ilustra bem as características do formulário mestre-detalhe. A diferença marcante entre este padrão e o formulário simples está na inclusão de uma região onde são exibidos as informações dependentes através do padrão grade. No caso específico, os dados da solicitação de pagamento podem ser vistos na parte superior da tela. Ao fundo, um controle paginado permite visualizar os detalhes da solicitação- um conjunto por vez. Os detalhes visíveis no exemplo são as contas geradas no financeiro relacionadas com a solicitação de pagamento visível na parte superior da UI.

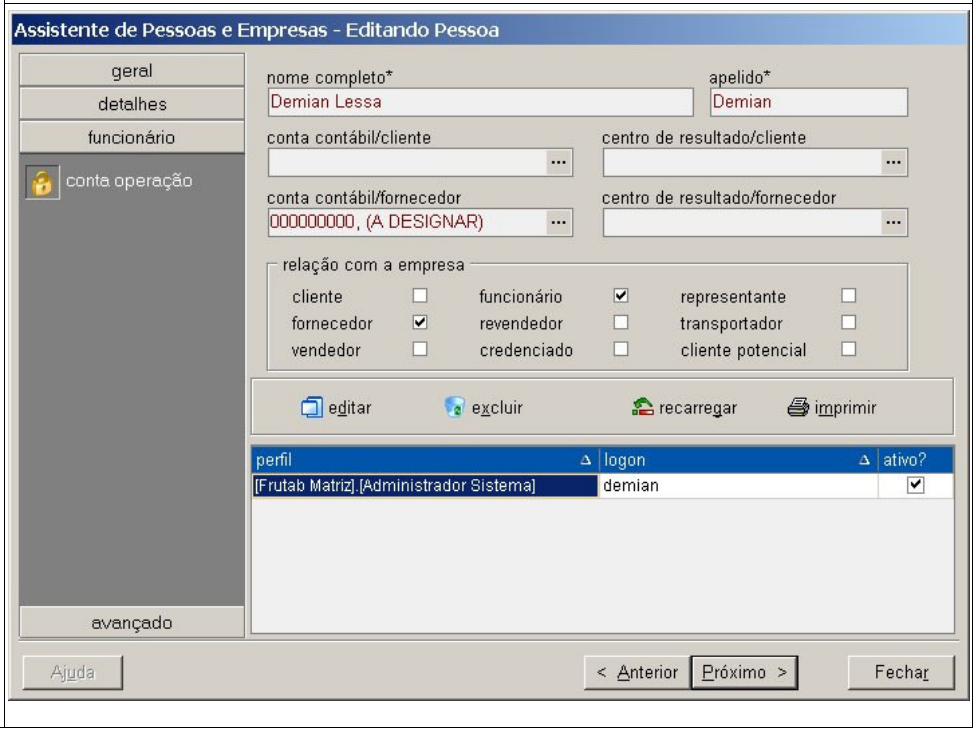
Editor de Dados: Formulário Mestre-Detalhe (cont.).

Nome	Assistente (também conhecido como Expert)
Grupo	Editor de Dados
Problema	O usuário precisa realizar uma tarefa complexa de dados que depende de uma série de outras decisões/tarefas menos complexas.
Contexto	O usuário precisa realizar uma operação complexa composta de uma seqüência (linear) de tarefas menores, onde uma série de decisões têm que ser tomadas em cada etapa. A quantidade de tarefas menores é relativamente pequena, girando tipicamente entre 3 e 10.
Solução	A solução permitirá ao usuário realizar uma tarefa por vez, recebendo instruções de como proceder em cada uma dessas etapas sempre que necessário. Soluções mais avançadas permitem identificar qual que partes da tarefa já foram realizadas e que partes ainda precisam ser completadas. Sempre que possível, o usuário poderá retornar a um ponto anterior do processo, desfazendo as tarefas até aquele ponto. Em geral, a UI é composta por uma região de interação com o usuário, onde são colocados os elementos da UI com os quais o usuário interage para realizar sua tarefa, e outra região para o controle do assistente- navegação, ajuda, conclusão, etc.
Benefícios	Facilita a memorização da tarefa. Contribui no entendimento do modelo conceitual através da incorporação das diversas etapas envolvidas numa atividade complexa.
Exemplo	

Editor de Dados: Assistente (1 de 3).

Exemplo (cont.)





Editor de dados: Assistente (1 de 2).

Exemplo (cont.)	As interfaces acima ilustram etapas de um mesmo assistente. Na primeira tela, o assistente guia o usuário na localização de uma entidade no sistema. Caso o usuário encontre a entidade que ele busca, ele pode optar por alterá-la em seguida. Na segunda tela, o usuário, depois de ter encontrado a entidade que buscava, chega à etapa de edição dos dados da entidade. Na UI, os dados principais da entidade. À esquerda, um widget que permite a seleção de outras categorias de dados da entidade a consultar/editar. Na terceira tela, depois de selecionada uma categoria no widget à esquerda, a UI mostra os detalhes relacionados com as informações de funcionário da entidade. Nas telas 2 e 3 os dados podem ser meramente consultados como alterados. Durante a execução do assistente, o usuário conta com uma série de controles de navegação na parte inferior da UI. As opções “Anterior” e “Próximo” permitem que o usuário alterne entre as diversas etapas do processo. O botão “Ajuda” exibe ajuda contextual para a etapa do assistente. Finalmente, o botão “Fechar” permite que o assistente seja interrompido num ponto qualquer de sua execução.

Editor de Dados: Assistente (3 de 3).

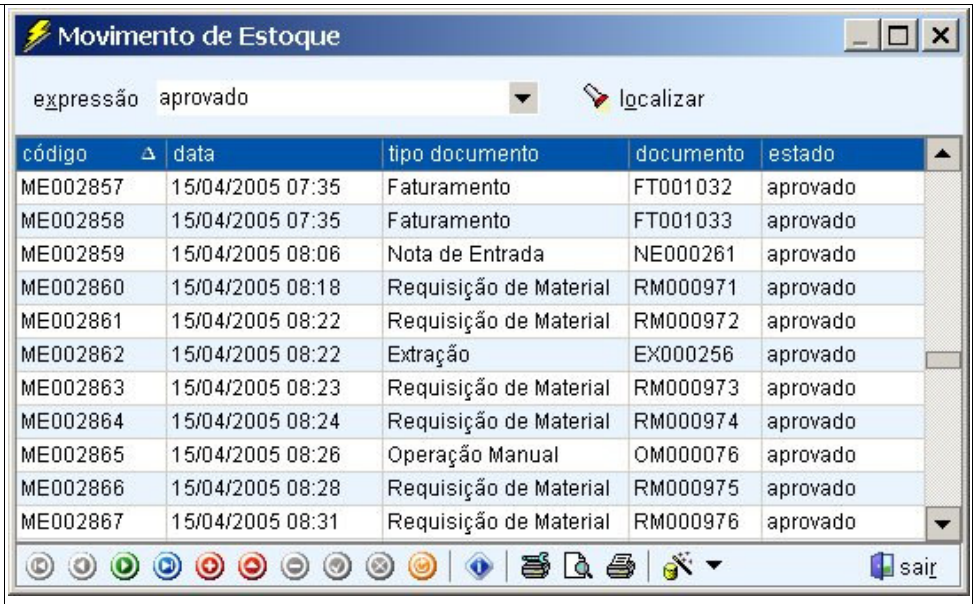
Nome	Listagem Simples
Grupo	Saída de Dados
Problema	O usuário precisa gerar uma saída permanente de um conjunto de informações, de estrutura simples e homogênea.
Contexto	O usuário precisa ter um registro permanente da visão atual de um conjunto de informações, normalmente de dados exibidos através de uma das variações do padrão grade. As informações devem ser apresentadas através de suas partes e dispostas de forma tabular.
Solução	A solução é gerar automaticamente uma listagem tomando por base as informações literalmente na forma como são visualizadas na UI (em geral uma variação da grade). O padrão zebrado deverá ser utilizado ou pelo menos disponibilizado para uso, para que os dados dispostos em forma tabular sejam facilmente lidos. As partes da informação devem ser identificadas e as larguras devidamente calculadas de modo a minimizar corte ou sobreposição dos dados exibidos.
Benefícios	Facilita a documentação. Agiliza tarefas manuais que precisam de saídas em papel para sua execução.
Exemplo	

Movimento de Estoque

código	data	tipo documento	documento	estado
ME000003	28/02/2005 13:11:06	Operação Manual	OM000003	aprovado
ME000004	28/02/2005 15:11:38	Extração	EX000001	aprovado
ME000005	28/02/2005 15:49:02	Extração	EX000002	aprovado
ME000006	01/03/2005 08:57:01	Operação Manual	OM000004	aprovado
ME000007	01/03/2005 09:38:32	Extração	EX000003	aprovado
ME000008	01/03/2005 09:40:29	Faturamento	FT000002	aprovado
ME000009	01/03/2005 09:40:56	Faturamento	FT000001	aprovado
ME000010	01/03/2005 09:52:42	Extração	EX000004	aprovado
ME000011	01/03/2005 10:13:39	Envasamento	EN000001	aprovado
ME000012	01/03/2005 10:17:48	Extração	EX000005	aprovado
ME000013	01/03/2005 10:23:06	Envasamento	EN000001	aprovado
ME000014	01/03/2005 11:26:52	Extração	EX000006	aprovado
ME000015	01/03/2005 11:31:02	Faturamento	FT000005	aprovado
ME000016	01/03/2005 11:45:57	Envasamento	EN000002	aprovado
ME000017	01/03/2005 11:50:33	Envasamento	EN000002	aprovado
ME000018	01/03/2005 14:54:46	Extração	EX000007	aprovado
ME000019	01/03/2005 15:10:30	Faturamento	FT000003	aprovado
ME000020	01/03/2005 15:13:58	Faturamento	FT000004	aprovado
ME000021	01/03/2005 15:47:22	Envasamento	EN000010	aprovado
ME000022	01/03/2005 15:49:21	Envasamento	EN000010	aprovado
ME000023	01/03/2005 15:51:00	Envasamento	EN000009	aprovado
ME000024	01/03/2005 15:54:13	Envasamento	EN000009	aprovado
ME000025	01/03/2005 15:59:33	Faturamento	FT000006	aprovado
ME000026	01/03/2005 15:59:57	Faturamento	FT000007	aprovado
ME000027	01/03/2005 16:00:19	Faturamento	FT000008	aprovado

Saída de Dados: Listagem Simples.

Exemplo (cont.)



código	data	tipo documento	documento	estado
ME002857	15/04/2005 07:35	Faturamento	FT001032	aprovado
ME002858	15/04/2005 07:35	Faturamento	FT001033	aprovado
ME002859	15/04/2005 08:06	Nota de Entrada	NE000261	aprovado
ME002860	15/04/2005 08:18	Requisição de Material	RM000971	aprovado
ME002861	15/04/2005 08:22	Requisição de Material	RM000972	aprovado
ME002862	15/04/2005 08:22	Extração	EX000256	aprovado
ME002863	15/04/2005 08:23	Requisição de Material	RM000973	aprovado
ME002864	15/04/2005 08:24	Requisição de Material	RM000974	aprovado
ME002865	15/04/2005 08:26	Operação Manual	OM000076	aprovado
ME002866	15/04/2005 08:28	Requisição de Material	RM000975	aprovado
ME002867	15/04/2005 08:31	Requisição de Material	RM000976	aprovado

No exemplo acima, uma listagem (primeira figura) foi gerada automaticamente a partir dos dados exibidos na grade (segunda figura). A listagem gerou uma visão correspondente aos dados exibidos pela grade, sem exceção- título, identificadores de colunas, dados e zebrado; além disso, incluiu números de página e data e hora de impressão, para documentação histórica. Uma tal listagem pode ser gerada, por exemplo, pelo acionamento do botão imprimir na barra de controle de uma grade.

Saída de dados: Listagem Simples (cont.).