Interfaces Homem-Computador:
Uma Primeira Introdução

Fábio Nogueira de Lucena
fabio@dcc.unicamp.br
Hans K.E. Liesenberg
hans@dcc.unicamp.br

Relatório Técnico DCC–94-07

Setembro de 1994

Interfaces Homem-Computador: Uma Primeira Introdução

Fábio Nogueira de Lucena
fabio@dcc.unicamp.br

Hans K.E. Liesenberg
hans@dcc.unicamp.br

Projeto Xchart*
DCC/IMECC/UNICAMP
Caixa Postal 6065
13081-970 Campinas/SP, Brazil
e-mail: *xchart@dcc.unicamp.br*
WWW: *http://www.dcc.unicamp.br/projects/Xchart*

Sumário

Um destaque crescente tem sido dado às interfaces homem-computador. Neste texto são apresentadas evidências que justificam este fato. O conteúdo deste trabalho visa servir de leitura inicial sobre o assunto com ênfase em aspectos computacionais. Mais especificamente, como o tema relaciona-se com a área de engenharia de software. No entanto, são citadas contribuições de outras áreas para este tópico multidisciplinar. São definidos alguns conceitos primordiais, apresentadas técnicas, modelos, ciclo de vida e outras questões relevantes. Embora o texto seja sucinto, esperamos que a bibliografia apresentada indique outras fontes com mais informações.

1 Introdução

To users, the interface is the system.
Hix and Hartson [34]

Os primeiros computadores e softwares foram projetados por engenheiros para serem usados por engenheiros. Estes engenheiros, no entanto, não formam um grupo representativo da atual comunidade de usuários de sistemas interativos. O uso de computadores pela sociedade tem crescido continuamente. Em alguns casos são "invisíveis" ao usuário como, por exemplo, na alimentação eletrônica de automóveis. Em outros, pessoas não especializadas em computação estão usando diretamente computadores. Isto torna as interfaces homem-computador (interfaces, por simplicidade) um elemento relevante na composição de um sistema computacional.

Antes de apresentar indícios para esta relevância e como ela tem-se refletida no ambiente acadêmico e comercial, convém definir o que se entende por interface segundo a ênfase deste texto.

*Projeto que se concentra no desenvolvimento (especificação e implementação) de interfaces homem-computador. Por curiosidade: a 22ª letra do alfabeto grego ($\mathbf{X}$ e χ) é identificada em Inglês pela palavra CHI, que coincide com o acrônimo de *Computer-Human Interface*.

Interface: compreende todos os comportamentos do usuário e do computador que são observáveis externamente [9]. Há uma linguagem de entrada, uma de saída para refletir os resultados e um protocolo de interação (veja a Figura 1).

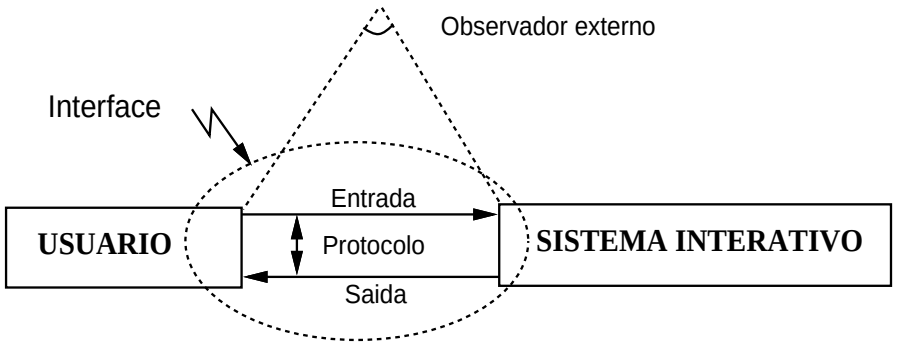

Figura 1: O conceito de interface [9].

Este conceito não fornece a perspectiva totalmente apropriada aos nossos mais voltados para aspectos relacionados com a área de engenharia de software. Podemos estabelecer um outro conceito de interfaces com uma definição fundamentada em software. A definição seguinte, ilustrada na Figura 2, fornece esta perspectiva.

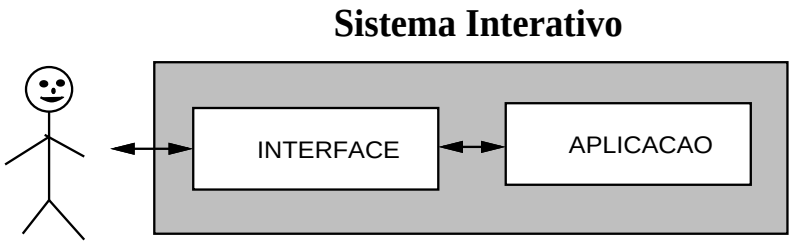

Figura 2: Visão simplificada de um sistema interativo.

Interface (software): parte do software de um sistema interativo responsável por traduzir ações do usuário em ativações das funcionalidades do sistema (aplicação), permitir que os resultados possam ser observados e coordenar esta interação. Em outras palavras, a interface é responsável pelo mapeamento das ações do usuário sobre dispositivos de entrada em pedidos de processamento fornecidos pela aplicação, e pela apresentação em forma adequada dos resultados produzidos.

A Figura 2 mostra um modelo simplificado de sistemas interativos. Nesta figura o componente **aplicação** é o elemento responsável pela parte funcional (algorítmica) do sis-

tema, ou seja, que transforma dados de entrada em dados de saída através da aplicação de uma função à entrada. O componente **interface** equivale à definição fornecida acima. A separação entre estes componentes é conhecida por independência de diálogo.

O termo diálogo geralmente é empregado com a mesma acepção de interface. É comum o uso de diálogo como a comunicação existente entre o ser humano e o sistema de computação, i.e., a troca de símbolos e ações entre o usuário e o computador. O termo interface também é comumente empregado como o software de suporte e hardware através do qual esta troca (diálogo) ocorre. A acepção destes termos é muito próxima, ambos referem-se somente às entradas do usuário e ao processamento localizado dessas entradas, juntamente com a apresentação dos resultados produzidos pela aplicação. Por outro lado, excluem o componente encarregado da transformação da entrada em saída. Esta função é de interesse do componente computacional, denominado aplicação.

A área de interação homem-computador envolve hardware, software, metodologias, técnicas, ferramentas e várias áreas do conhecimento humano. Estes itens são comentados sucintamente. A literatura fornecida ao longo do texto pode ser utilizada para obter detalhes mais específicos. Muitas referências apresentam estes itens como elementos estanques, sem a visão imprescindível do *processo* que os coloca em funcionamento durante o desenvolvimento de uma interface (ciclo de vida, métodos, técnicas e ferramentas). Tentamos, além de fornecer uma visão estanque de elementos da interação entre homem e computador, apresentar a relação entre eles e como empregá-los.

Aconselhamos [34] para um bom entendimento acerca do processo de desenvolvimento de uma interface. Referências que abrangem vários aspectos de interfaces em um só livro são [13, 31]. Embora este texto apresente informações destiladas de diversas referências, algumas fontes exerceram influência significativa no conteúdo aqui apresentado. São elas: [14, 30, 33] e [5]. [30] fornece uma visão geral da área. [5] enfatiza o desenvolvimento de software para interface.

O que é uma interface "amigável"?

O termo amigável (*user-friendly*) é comumente atribuído a interfaces. Abaixo seguem algumas características presentes em sistemas onde a interface recebe este adjetivo.

- *Facilidade de usar e aprender.* Embora seja auto-explicativo para os usuários, este termo precisa de um significado mais útil do que simplesmente "fácil" para projetistas. Do inglês *easy-to-learn* e *easy-to-use*. Geralmente são substituídos por *user-friendly* para caracterizar uma "boa" interface. Habermann [24] sugere uma semântica para *easy-to-learn* e *easy-to-use*: (1) a interface é invisível (o usuário pode concentrar-se nas tarefas que necessita realizar); (2) é previsível; (3) é flexível e (4) as pessoas gostam delas.

- *Taxa de erro mínima.* Em sistemas críticos (nucleares, controle de tráfico aéreo e outros) dificultar a ocorrência de erros cometidos pelos usuários é imprescindível. Convém relembrar que erros também afetam o desempenho dos usuários.

- *Recordação rápida.* O usuário esporádico, idealmente, não deve ter que recorrer a manuais quando for usar o sistema.

- *Atrativo.* Nem sempre o sistema mais poderoso é preferido pelo usuário. O usuário pode selecionar um sistema em que seu desempenho é inferior comparando com um outro, mas com o qual se sente mais "confortável."

Organização do Texto

A Seção 4.1 descreve várias áreas que contribuem para o desenvolvimento de interfaces. A Seção 3 descreve atividades a serem realizadas durante o desenvolvimento de interfaces.

No ciclo de vida de uma interface nota-se que seus primeiros estágios sofrem forte influência de outras áreas do conhecimento. A estes primeiros estágios dá-se o nome de projeto. O projeto é visto na Seção 4 e a implementação na Seção 5. Considerações finais são feitas na Seção 9.

2 Por que interfaces são importantes?

Newman e Sproull [56] há mais de uma década já descreviam a influência das interfaces no sucesso de sistemas interativos. Aprender a usá-los geralmente implica em investimento razoável de tempo. Uma boa interface torna a interação com o sistema mais fácil de aprender e usar, i.e., amigável (*user-friendly*). Em outras palavras, a interface pode influir na produtividade do usuário, que nem sempre prefere um sistema com mais recursos ou eficiência do ponto de vista computacional. Um exemplo real é o comentário de Bill Gates, grande acionista da Microsoft, acerca da frustação parcial do Word 2.0 [37]: *merecemos a culpa por não termos facilitado o seu aprendizado. No tocante aos recursos o produto era fantástico, mas no que se refere a facilidades dos primeiros passos não nos saímos muito bem.*

Reconhecendo o papel de destaque de uma interface em um sistema interativo, Curtis em [11] destaca vários projetos japoneses de longo prazo envolvendo investimentos consideráveis. Todos com a atenção centralizada no desenvolvimento de interfaces. Segundo Curtis, uma vez atingida certa uniformidade na confiabilidade dos produtos, as interfaces são o próximo passo para a distinção e vantagem comercial, o que justifica os investimentos japoneses. O projeto japonês Friend21, por exemplo, tem duração estimada em seis anos, envolve, 14 grandes companhias e totaliza um investimento de 120 milhões de dólares. *O projeto acredita que no século XXI todas as pessoas estarão utilizando os computadores em suas atividades diárias* [53]. Conforme [41], "gigantes" da computação como a IBM, Microsoft, Digital e Hewlett-Packard (HP) também têm dado grande ênfase para as interfaces.

O foco de atenção atualmente despendido com as interfaces possui uma linha simples de justificativa:

Primeiro. O uso de computadores tem crescido continuamente. Prevê-se que praticamente todo ser humano irá utilizar computadores no futuro de uma ou de outra forma.

Segundo. O mercado tem mostrado que, nas vendas entre produtos similares, sobressai o que melhor permite o acesso do usuário à funcionalidade fornecida pelo sistema. Convém ressaltar que em alguns casos a funcionalidade e o desempenho não são suficientes

para agradar o usuário, que faz opção por outro sistema com interface atrativa. Ou seja, se um produto deseja ser competitivo, necessariamente sua interface deve ser considerada de forma séria.

Conclusão. O mercado para sistemas interativos caminha para uma popularidade equivalente à dos televisores atualmente. Acredita-se que a distinção entre produtos dar-se-á pela interface. Isto torna compreensível a crescente quantidade de esforços nesta área. Ainda mais, estes esforços já mostraram que, como veremos ao longo do texto, há muito o que fazer pelo desenvolvimento complexo e difícil das interfaces.

Em [3] fica evidenciada a importância e o custo elevado das interfaces, que chegam a ser grande parte do código de um sistema interativo. Em [5, pág. v] afirma-se que a interface consome até 70% dos custos totais do ciclo de vida de um sistema interativo. Em [6] é relatado que para um sistema especialista poder-se-ia esperar que a maior parte de código fosse devotada à base de conhecimento e à máquina de inferência. Entretanto, a base de conhecimento e a máquina de inferência equivalem a apenas um terço do sistema. A interface é a maior parte do sistema, cerca de 42% do código. Esta não é uma situação atípica. Outros sistemas apresentam proporções similares.

Em recente pesquisa sobre dezenas de projetos [54], verifica-se que em média 48% do código de um sistema interativo é dedicado à interface. Quanto ao tempo total de desenvolvimento, aquele consumido na interface equivale a 48% do tempo de projeto de todo o sistema, 50% de toda a implementação e 37% da manutenção.

Ainda convém ressaltar que o preço de software tem-se tornado cada vez mais significativo com a queda do preço de hardware. Uma vez adquirido um sistema (hardware + software) ainda há custos envolvidos com o treinamento e uso diário por parte dos usuários. A interação com o sistema inclui, muitas vezes, o consumo de tempo desnecessário de operação se realizada de forma inadequada, ou ainda na correção de erros freq:uentemente cometidos pelos usuários. Estes custos devem estar envolvidos em uma análise custo/benefício no desenvolvimento de uma interface amigável. Conforme [53], economias da ordem de milhões de dólares podem ser realizadas em interfaces onde o usuário comete poucos erros, o tempo de descrição das tarefas a serem realizadas é mínimo, a distração do usuário é reduzida e os treinamentos são eliminados. Ou seja, a interface interfere economicamente na utilização de um sistema.

A qualidade da interface tem grande influência no sucesso comercial de um software. Os melhores sistemas de hardware e software podem se tornar ineficazes devido a uma interface imprópria com o usuário [55]. Enfim, sistemas interativos com boas interfaces são preferidos àqueles que apresentam barreiras quanto ao seu uso ou aprendizado. Infelizmente o desenvolvimento de boas interfaces é uma atividade que apresenta vários desafios [53]. Muito trabalho tem sido realizado visando a construção de boas interfaces a baixo custo. Este texto privilegia aspectos computacionais (engenharia de software) de uma interface ressaltando o seu desenvolvimento.

Os itens abaixo enfatizam alguns pontos que motivam trabalhos nesta área:

1. O custo de um sistema computacional não se limita a hardware + software. É preciso treinar usuários. Quanto mais difícil de aprender mais oneroso é o treinamento

e quanto mais difícil de usar, menor é o desempenho do usuário através de erros constantes, lentidão de operação do sistema e outros.

2. Softwares que apresentam dificuldades como as acima citadas tendem a ser rejeitados pelos usuários. Comercialmente, o sucesso de vendas de software interativos está intimamente relacionado à facilidade de uso e aprendizado do produto, adjetivos que acompanham praticamente toda propaganda de software hoje em dia.

3. Riscos fatais e implicações em sistemas críticos. Foi parcialmente atribuído a interface homem-computador a inabilidade do exército americano no combate dos mísseis Exocet iraquianos. A implementação errada de uma interface matou várias pessoas de doses excessivas de radiação, parcialmente em decorrência de erro no código de manipulação do cursor no Therac-25. Myers [53] comenta em mais detalhes e cita referências sobre estes e outros casos.

4. O desenvolvimento de interfaces é um processo caro, difícil, demanda tempo e que ainda há muito a ser empreendido. Ao longo do texto há informações que tornam mais palpáveis estas características.

5. Por último, todos estes itens tornam-se problemas cujas soluções são desejáveis, pois o número de usuários de computadores está se expandindo e com ele a demanda por sistemas interativos. As vendas, a descrição adequada e correta de tarefas, e inclusive a segurança de tais sistemas são influenciados pela interface, que consome 50% dos recursos de desenvolvimento de um sistema.

Conseq:uências

Há vários indicadores que, não oriundos apenas em ambiente acadêmico, reforçam o crescimento desta área. Informações sobre livros, periódicos, grupos de interesse específico e outros são apresentados na Seção 8.

1. Periódicos específicos ou pertinentes:
   (a) *ACM SIGCHI Bulletin.*
   (b) *ACM TOCHI.*
   (c) *ACM Interactions.*
   (d) *Human-Computer Interactions* (jornal).
   (e) *International Journal of Human-Computer Interaction.*
   (f) *Interacting with Computers.*
   (g) *Behaviour and Information Technology* (jornal).
   (h) *International Journal of Man-Machine Studies.*
2. Conferências específicas ou que apresentam trabalhos desta área:
   (a) *ACM User Interface Software Technology.*

(b) *ACM SIGCHI – Human Factors in Computing Systems.*

(c) *Human Factors Society* (encontro anual).

(d) *Computer Supported Cooperative Work.*

(e) *IFIP INTERACT.*

(f) *International Conference on Human-Computer Interaction*

(g) ACM SIGGRAPH

(h) *British Computer Society HCI: Human-Computer Interaction.*

(i) Eletrônicas:

i. netnews: comp.human-factors

ii. netnews: comp.cog-eng

iii. internet: students.chi@xerox.com (Administração: registrar.chi@xerox.com).

3. A ACM/IEEE incluem a disciplina de interação homem-computador na grade curricular da graduação recomendada por estas entidades.

4. Grandes projetos estão em desenvolvimento no Japão e outros em empresas como a IBM, Microsoft e HP, essencialmente voltados para o desenvolvimento de interfaces.

3 Desenvolvimento de interfaces

Because even the best usability experts cannot design perfect user interfaces in a single attempt, interface designers should build a usability-engineering life cycle around the concept of iteration.
Nielsen [57]

Esta seção fornece vários fundamentos necessários à construção de interfaces. Primeiro são elucidados alguns termos do âmbito de engenharia de software. A engenharia de software estabelece algumas qualidades desejáveis a serem atingidas por um software (correção, confiabilidade, robustez, facilidade de manutenção e outras) e um conjunto de princípios (modularidade, abstração e outros) que buscam, quando seguidos, atingir estas qualidades [21]. Para aplicar princípios, o engenheiro de software deve estar equipado com técnicas apropriadas. Técnicas podem ser agrupadas formando uma metodologia. O objetivo de uma metodologia é permitir o desenvolvimento sistemático de uma classe de sistemas através de uma abordagem ou modelo de solução para esta classe de sistemas. Ferramentas, por sua vez, são desenvolvidas para dar suporte à aplicação de técnicas, métodos e metodologias. Métodos são regras gerais que governam a execução de alguma atividade; são abordagens sistemáticas e disciplinadas. Técnicas são atividades mais mecânicas. Engenharia de software para interfaces pode ser obtida em [15].

O desenvolvimento de interfaces é um processo iterativo, que requer o projeto seguido de alguma forma de avaliação que realimenta o próprio projeto. Versões de projeto refinadas a cada teste com os usuários podem substancialmente melhorar a interação com o sistema [57]. O **projeto** pode ser influenciado por padrões, *guidelines*, experiência com outros sistemas e pelos próprios requisitos do sistema. Esta atividade produz uma especificação que

pode ser utilizada para a implementação ou um modelo formal. O emprego de um modelo formal permite que a interface em desenvolvimento possa ser avaliada sem a necessidade de implementá-la. A **implementação** pode envolver a construção de todo o sistema, parte do sistema, ou ainda a construção de um protótipo. A **avaliação** pode ser feita sobre um um modelo formal ou sobre a implementação do projeto. Note-se que apenas parte do projeto pode ser implementado para avaliação. A citação no início desta seção justifica a necessidade desta etapa.

Uma análise de várias abordagens para o projeto e implementação de interfaces pode ser vista em [12]. Quanto ao ciclo de vida da Figura 3, convém ressaltar que modelos mais tradicionais como o *Waterfall* (cascata) não são apropriados. A suposição de que as atividades envolvidas ocorrem de forma linear não é válida. Conforme [25], a interface pode não estar bem definida até poucos instantes antes da instalação de um sistema interativo! Em [28] é apresentado outro ciclo de vida para o desenvolvimento de interfaces exibido na Figura 4. O relacionamento entre notações, ferramentas e abordagens para o projeto de interfaces e os métodos tradicionais para desenvolvimento de sistemas é comentado em [70].

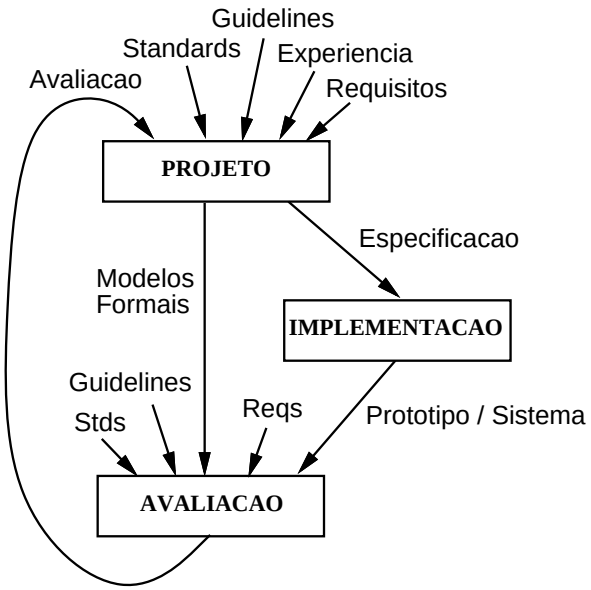

Figura 3: Ciclo de vida de desenvolvimento de interface [59].

4 Projeto

Definir projeto (*design*) não é uma tarefa simples. Em essência, **projeto** é a descrição de um objeto a partir da qual ele pode ser construído. Ou ainda um planejamento de uma construção. Note que projeto é comumente empregado ora como "processo" ora como "produto." No entanto, em ambos os casos referem-se a algo obtido antes do início da confecção de um produto. Antes que uma ponte seja construída é necessário que um projeto

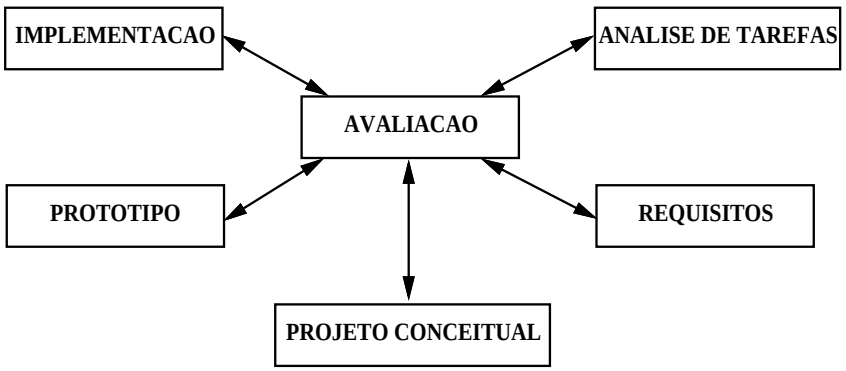

Figura 4: Ciclo de vida estrela [28].

correspondente seja realizado e avaliado. Pequenos empreendimentos podem ser realizados sem a descrição explícita de um projeto, ou seja, eles podem ser confeccionados a medida que são "imaginados." Outros, no entanto, não permitem esta abordagem em decorrência dos custos envolvidos na eliminação de falhas. Corrigir um erro seria caro, quando não impossível. Por outro lado, corrigir modelos e especificações é uma atividade bem menos onerosa do que os seus efeitos correspondentes sobre objetos reais. Por exemplo, com lápis e borracha pode-se reposicionar o botão liga/desliga do controle remoto de uma televisão, aumentar ou abaixar a altura de uma ponte, ou ainda corrigir a sua direção em alguns graus sem grandes dificuldades. No entanto, as conseq:uências destas mudanças sobre tais objetos em construção ou já acabados são claramente bem mais onerosas.

A construção de interfaces ainda não possui um padrão de desenvolvimento sistemático que possa ser aplicado com sucesso garantido. Definir uma interface com a qual o usuário irá interagir e, paulatinamente, refinar esta definição através da avaliação de protótipos/modelos formais até a obtenção de um projeto satisfatório é uma necessidade. Só então tem-se início a implementação. A implementação pode ainda revelar dificuldades que não foram adequadamente contempladas no projeto e, neste caso, novamente o projeto é realimentado com novos dados, modificado e a implementação tem prosseguimento. A Figura 3 ilustra esta evolução. Note que avaliação e iteração são também características no ciclo de de vida estrela da Figura 4.

O projeto de uma interface é equivalente a descrição do *look and feel* da interface. *Look and Feel* compreende a aparência e dinâmica de interação de uma interface. Um *Look and Feel* pode ser empregado por diferentes *toolkits* em diferentes plataformas, fornecendo a mesma aparência e comportamento para os usuários. Uma extensa e didática discussão sobre projeto é [4, 43]. Outras referências recomendadas incluem [7, 18, 71].

Projeto: define o comportamento e a apresentação de uma interface. A execução do projeto utiliza-se de psicologia, fatores humanos, tecnologia dos dispositivos de entrada e saída, tipos de diálogos, análise de tarefas, princípios, *guidelines*, padrões, regras, protótipos e avaliação de sistemas existentes. O resultado do projeto é a especificação do projeto, geralmente através de especificações formais, protótipos ou ainda documentos informais.

4.1 Uma atividade multidisciplinar

O projeto de uma interface envolve o conhecimento de distintas especialidades. Em geral, no entanto, sistemas são concebidos sem a presença de especialistas pertinentes a estas áreas e, em alguns casos, o conhecimento destas áreas é aplicado por especialistas em computação. Conforme [14], projetos desenvolvidos sem os conhecimentos de várias áreas, necessários a uma interface em particular, fazem uso da capacidade de adaptação do ser humano para compensar suas deficiências. Em [14] cada uma das áreas abaixo, relevantes para a definição e o estudo de interfaces, é comentada em mais detalhes.

Áreas de estudo relevantes para a interação homem-computador:

- *Ciência da Computação.* Fornece a estrutura tecnológica para facilitar o projeto e implementação de interfaces. Interface é considerada neste texto da perspectiva desta ciência.

- *Psicologia.* Preocupa-se com o entendimento do comportamento humano, percepção, cognição e outros. A psicologia permite derivar métodos para auxiliar o usuário humano nas suas atividades. O número mágico $7 +/- 2$ [47] e a razão áurea são alguns exemplos. Em [19, 20] é destacada a influência positiva de objetos de interação que apresentam a razão áurea.

- *Ergonomia.* Envolve-se com aspectos físicos da adaptação das máquinas para uso humano. Por exemplo, formatos de dispositivos físicos de interação são de interesse desta área.

- *Ling:uística.* A interação homem-computador exige uma linguagem. Ling:uística é o estudo da linguagem. A teoria de linguagens formais compartilha formalismos com a ciência da computação que são amplamente utilizados na especificação formal de diálogos. *Lex* (gerador de analisador léxico) e *Yacc* (gerador de analisador sintático) são ferramentas do ambiente UNIX utilizadas na implementação de interfaces [54].

- *Sociologia.* Neste contexto a sociologia preocupa-se com o impacto dos sistemas interativos na estrutura da sociedade.
  Groupware applications,[1] por exemplo, beneficiam-se das teorias sociais [16].

[1] Sistemas que dão suporte à grupos de pessoas envolvidas com a realização de uma tarefa comum e que fornecem uma interface para um ambiente compartilhado.

- *Desenho gráfico e tipografia.* A habilidade estética dessas áreas é importante a medida que a apresentação da interface torna-se "bonita" aos olhos dos usuários.

Não é esperado que um único indivíduo (projetista) possua conhecimentos em todas estas áreas. Uma alternativa mais realística é ter uma "consciência" (*awareness*, nível de entendimento) acerca das áreas, combinada com especialistas em uma ou outra área. Este nível de entendimento é particularmente essencial para os especialistas em computação que terão que desenvolver projetos de interfaces. O nível de entendimento fornece apenas diretivas (*guidelines*) para o projetista e não é suficiente para assegurar uma interface de alta qualidade, que requer um projeto mais rigoroso e avaliação realizados com os métodos pertinentes a cada área.

Por fim, convém ressaltar que o conhecimento dos especialistas em computação no projeto de interfaces é limitado. Em outras palavras, a formação básica típica deste profissional não contempla todos os conhecimentos necessários para o desenvolvimento desta atividade. Seja pelo caráter multidisciplinar ou ainda pela falácia da intuição egocêntrica. Esta falácia refere-se à ilusão de que se conhece o que determina o comportamento e satisfação do usuário. Conforme [42], mesmo quando um sistema é obviamente "difícil" a intuição pode não revelar as verdadeiras razões. O melhor é, sempre que possível, permitir que o especialista adequado desenvolva a atividade que lhe compete.

4.2 Executando o projeto

Os projetistas tentam satisfazer os requisitos humanos de uma sistema interativo através do projeto da interface, que envolve a aplicação de conhecimentos de várias áreas (veja Seção 4.1). Doravante são comentadas as várias entradas, i.e., informações que são utilizadas durante a realização do projeto de uma interface e o resultado do projeto (especificações). A Figura 3 (pág. 8) descreve o fluxo de tais informações ao longo do desenvolvimento de uma interface.

Análise de tarefas

A análise de tarefas (veja [14, cap. 5] acerca de um método para esta análise) é uma importante etapa nos primeiros estágios de um projeto. Esta análise permite obter informações que substituem aquelas oriundas da intuição do projetista acerca de tarefas e como as pessoas executam estas tarefas em um domínio particular.

Psicologia

Interfaces devem estar em conformidade com as capacidades e limitações humanas. Isto compreende a forma como os seres humanos processam informações, sensação, percepção, atenção, desempenho, aprendizado e memória.

Princípios, padrões, guidelines e regras

Princípios são metas gerais que podem ser úteis à organização de um projeto. Princípios, no entanto, não especificam métodos através dos quais podem ser obtidos de forma sistemática.

Guidelines são regras gerais a serem seguidas durante o projeto. Padrões compreendem princípios, regras e guidelines que devem ser seguidos por força de mercado ou "padrões" como *Windows*, IBM CUA e outros.

Abaixo seguem alguns princípios de projeto geralmente empregados. Embora em [64] seja feita referência a "regras de ouro" para denominar esta lista, nada assegura que estes princípios conduzirão a uma "boa" interface [48, 50]. Russell **et al.** [61] citam situações em que se tem conflitos entre eles. Em [14] é afirmado que contra-exemplos podem ser estabelecidos para cada princípio. Em [69] há uma extensa lista de diretivas (*guidelines*) para auxiliar o projetista de uma interface durante a atividade de projeto. Outra referência sobre este assunto é [18, págs. 391-414].

- *Consistência.* O diálogo deve seguir regras simples e não apresentar casos especiais ou exceções para operações similares.

- *Retroalimentação* (*feedback*). Ações do usuário devem gerar uma retroalimentação. Geralmente isto é obtido através de representações visuais que refletem, constantemente, o estado interno do sistema e, por conseguinte, suas alterações.

- *Minimizar possibilidades de erro.* Deve ser fornecido ao usuário somente os comandos passíveis de serem executados no instante da interação. Se uma operação não pode ser disparada então não deve ser acessível.

- *Fornecer um meio de recuperação de erros.* Entre operações desejáveis em uma interface comum podemos citar: refazer um contexto anterior a execução de um comando (*undo*), cancelar, interromper ou substituir um dado fornecido ou comando. Sem estas operações a dificuldade de corrigir um erro pode ser inaceitável, ou inibir a experimentação por parte dos usuários.

- *Tratar adequadamente usuários com habilidades diferentes.* Alguns sistemas são usados por uma grande variedade de pessoas, desde novatos até especialistas. A cada um destes grupos deve estar disponível um diálogo apropriado.

- *Minimizar a necessidade de memorização.* Quanto menos memória for exigida melhor a aceitação. Menus e *form-fill* são estilos utilizados com esta finalidade.

- *Metáforas.* Visam reduzir barreiras da interação utilizando ações, procedimentos e conceitos familiares ao usuário. Detalhes sobre metáforas podem ser obtidos em [8].

Dispositivos de entrada e saída

Dispositivos de entrada (p.ex., teclado, mouse) e saída (p.ex., impressora, tela) conectam os "sentidos" humanos aos canais utilizados pelo computador para comunicação com o meio externo. O projetista deve conhecer as tecnologias disponíveis e quando empregá-las.

Estilos de interação

Estilo refere-se à forma através da qual ocorre a interação com o computador. Em geral, vários estilos estão presentes em uma mesma interface. Abaixo são comentados vários estilos. Detalhes podem ser obtidos em [18] e [64]. Manipulação direta é um dos mais sofisticados e muitas pesquisas têm sido realizadas em busca de técnicas adequadas para especificá-lo adequadamente [17, 32, 39]. O projetista deve saber empregar adequadamente os vários estilos de interação. Vantagens e desvantagens de vários estilos podem ser encontradas em [14]. Uma abordagem para a implementação de uma interface em que vários estilos de interação coexistem e o usuário pode alternar entre eles durante a "descrição" de uma mesma tarefa é apresentada em [40]. Outros estilos de interação (p.ex., *dataglove*) são comentados em [74].

- *WYSIWYG.* Uma interface WYSIWYG (*What You See Is What You Get*) permite ao usuário observar na tela o resultado do processamento, ou seja, a saída. Por exemplo, um editor de texto WYSIWYG mostra na tela o que será impresso antes que o usuário obtenha a impressão.

- *Linguagem de comandos.* Meio de interação tradicional em que o usuário fornece, via teclado, uma seq:uência de caracteres correspondentes a entrada.

- *Linguagem natural.* Extensão do caso anterior onde o usuário não está limitado a um vocabulário exíguo de palavras e sintaxe rígida.

- *Manipulação direta.* Este termo está associado a algumas idéias centrais [63]: (1) visibilidade do objeto de interesse; (2) ações rápidas e reversíveis; (3) visualização imediata do resultado, e (4) substituição de estilos de interação como linguagem de comandos e menus pela manipulação direta do objeto de interesse. Em outras palavras, permite que o usuário manipule, geralmente com o uso do mouse, representações da realidade.

- *Demonstracional.* Do inglês *demonstrational.* Estilo no qual o usuário pode usar exemplos, fornecidos através de manipulação direta, para especificar operações abstratas [51]. O sistema usa inferências para "sugerir" generalizações. Por exemplo, em uma interface demonstracional na segunda vez que o usuário "arrasta" um arquivo com extensão `.bak` até uma lata de lixo, imediatamente o sistema conclui que provavelmente o usuário deseja remover todos os arquivos com esta extensão. Antes de realizar esta operação, no entanto, esta "inferência" é confirmada com o usuário.

- *Icônico.* Ícone é um símbolo gráfico que apresenta uma relação de semelhança ou analogia com um objeto, propriedade ou ação. Em interfaces icônicas conceitos são ligados a ícones e acionados através do mouse.

- *Menus.* Menu é uma lista de opções dispostas em uma coluna. Reduz a necessidade de memorização e limita o conjunto de opções. Há várias formas de apresentação de um menu. Os menus *pie* (menus circulares), por exemplo, permitem a mesma facilidade de acesso a todos os itens [35], o que não é possível com os menus tradicionais.

- *Form fill-in.* Estilo adequado para a entrada de dados composta por vários campos. Pode-se alternar entre campos, identificados por rótulos, e fornecer a entrada em qualquer ordem.

Protótipos

A complexidade das interfaces e o conhecimento ainda insatisfatório acerca de como construir interfaces obriga validar decisões de projeto. Geralmente isto é feito através da construção de protótipos que refletem de leiaute de tela e sintaxe de comandos. As ferramentas utilizadas para construir protótipos não são úteis, em geral, para a implementação. Técnicas para a construção rápida de protótipos por projetistas são descritas em [73].

O projeto de uma interface, geralmente, dura semanas até que algum resultado possa ser apresentado ao usuário. Uma forma alternativa de construir protótipos em pouco tempo de projeto é apresentada em [60]. Nesta abordagem, em vez das tradicionais ferramentas para confecção de protótipos é utilizado papel, canetas coloridas e outros materiais típicos de um escritório. A característica principal desta abordagem está na rapidez em que o usuário pode ter contato com um protótipo. Tradicionalmente os protótipos levam mais tempo para serem construídos e modificados. Usando esta técnica, em pouco tempo podem ser desenhados menus, telas, botões e caixas de diálogo. Todos estes desenhos podem ser utilizados durante uma execução simulada da interface projetada utilizando estes materiais. As vantagens, neste caso, residem na pouca resistência a alterações no projeto e a devida atenção dada aos aspectos mais relevantes, em detrimento de cores, formato de letras e outros que são, pelo próprio processo de confecção, eliminados.

Especificações de projeto (produto)

Em geral o projeto é documentado através de uma combinação de figuras, protótipos, descrição em linguagem natural ou mesmo através de linguagens formais. Por exemplo, a notação UAN foi desenvolvida especificamente para esta finalidade [29]. Outras exemplos são [25, 38, 39, 67]. Este tópico está intimamente relacionado aos UIMSs (veja Seção 5.2), pois estes sistemas, em geral, aceitam especificações em linguagens que podem ser executadas pelos mesmos e automaticamente geram a interface correspondente. Um tratamento extenso sobre este assunto pode ser obtido em [52].

5 Implementação

A implementação de uma interface é uma tarefa difícil [53, 68]. Entre as dificuldades:

1. Projeto iterativo. Não há "regras de ouro" para o projeto de interfaces. Em conseq:uência, o código é freq:uentemente modificado até que resultados satisfatórios possam ser obtidos [15]. Logo este código deve ser escrito de tal forma que possa ser modificado facilmente e, se possível, sem afetar outras partes do sistema.

2. Apresentação atrativa. Usar os pacotes gráficos e bibliotecas disponíveis na confecção de interfaces gráficas não é uma tarefa trivial. Obter um resultado satisfatório pode ser um desafio.

3. Entrada assíncrona. As interfaces que empregam o estilo de manipulação são guiadas pelas ações dos usuários, que podem ocorrer a qualquer momento. Torna-se necessário o emprego do controle externo que é uma estrutura de software diferente dos programas convencionais.

   No processo de desenvolvimento o programador deve fazer distinção entre operações de interesse da interface daquelas da aplicação e determinar a interação entre a aplicação e a interface. Uma decisão a ser tomada é a fonte de controle. Na Figura 5 vemos duas alternativas. Em aplicações tradicionais o controle reside na aplicação. Em termos de programação, o controle (*thread*) está na aplicação. Uma alternativa é colocar o controle na interface, como é o caso de muitas interfaces produzidas com o uso de ferramentas. Neste caso a interface transfere o controle para rotinas da aplicação em resposta às ações dos usuários na forma de *callbacks*. Do ponto de vista da interface a aplicação é uma biblioteca de procedimentos que implementa a funcionalidade do sistema. Em termos de programação o controle pertence à interface e é transferido para a aplicação na forma de chamada de função. Questões de controle e comunicação pertinentes ao software da interface são contemplados em [30].

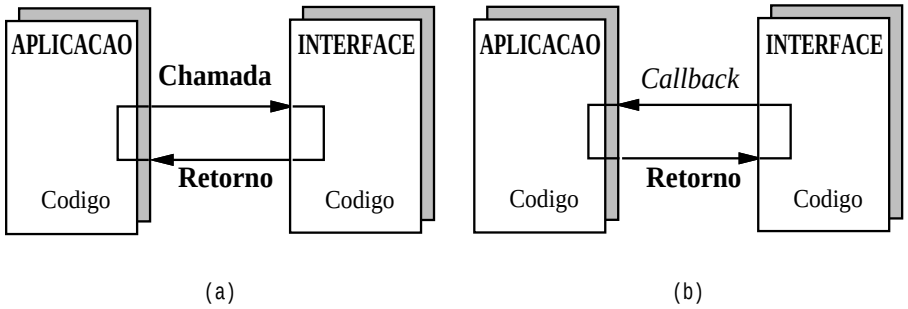

Figura 5: Fluxo de controle convencional (a) e existente em ferramentas (b).

4. Concorrência. Funções da aplicação podem consumir grande quantidade de tempo para as suas execuções. No entanto, a interface não deve esperar por tais execuções. Ela deve estar "pronta" para atender os pedidos dos usuários assim que forem requisitados. Em decorrência, o sistema deve ser organizado em múltiplos processos. Isto significa que o programador deverá tratar problemas de sincronização, exclusão mútua e outros.

5. Eficiência. Resposta imediata às ações dos usuários é desejável em uma interface. Obter esta reação imediata requer trabalho adicional do programador em alguns casos.

6. Manipulação de erro. Sistemas interativos não admitem o término de execução em decorrência de erro do usuário. É preciso informá-lo e, se possível, permitir que o erro possa ser corrigido.

7. Suspensão da execução, *undo* e ajuda. Geralmente sistemas interativos permitem que uma atividade seja suspensa em qualquer instante ou ainda, que possa ser "desfeita." Há ainda necessidade de ajuda ao usuário, i.e., *help on-line*. Estes requisitos acrescentam mais funcionalidade ao código da interface.

Há uma vasta quantidade de ferramentas utilizadas na construção de interfaces. Algumas disponíveis comercialmente, enquanto outras através de centros de pesquisa. Estas ferramentas distinguem-se umas das outras por vários motivos: funcionalidade que provêem, facilidade de uso da própria ferramenta, estilos de interação empregados (Motif™, OpenLook™), portabilidade (veja [2]) da interface gerada e outras [34]. Nesta seção abordaremos vários tipos de ferramentas para o desenvolvimento de interfaces. Note que existem ferramentas orientadas para a confecção de protótipos, projeto e avaliação. A implementação é apenas uma das atividades do desenvolvimento de interfaces que possuem ferramentas de apoio.

5.1 Toolkits

Toolkit é uma biblioteca de rotinas usadas por programadores para a implementação de detalhes de baixo nível. Em geral, quando se emprega um *toolkit* na implementação de uma interface, o controle reside no componente computacional (ou aplicação, conforme visto anteriormente). A implementação resultante é um conjunto de *widgets*. *Widget* é um objeto de interação com apresentação e comportamento bem definidos. Implementam elementos típicos de uma interface como botões, menus e outros. Um *toolkit* ainda pode possuir ferramenta que facilita o posicionamento destes objetos de interação (*widgets*) na tela [58]. A interface desenvolvida com o uso de um *toolkit* geralmente interage com a aplicação através de *callbacks*. Ou seja, funções fornecidas pelo programador para serem executadas em pontos específicos da interação do sistema com o usuário, em resposta a algum evento ocorrido.

5.2 UIMS

O modelo de Seeheim [22] é composto por módulos que se comunicam entre si (veja Figura 6). Cada um dos componentes possui uma função imprescindível de um UIMS e, portanto, uma descrição correspondente da interface deve existir para permitir a sua geração automática. Trata-se de um modelo lógico, não diz como um UIMS deve ser estruturado ou implementado. Cada componente deste modelo tem uma função diferente e requer uma técnica de descrição particular. Note ainda a separação desejável entre a interface e a aplicação.

UIMS (User Interface Management System): sistema voltado para o desenvolvimento e execução de interfaces entre homem e computador. Ajudam na especificação, projeto, construção de protótipos, implementação, execução, avaliação, modificação e manutenção de interfaces [33].

O componente de apresentação gerencia a tela (criação) e monitora os dispositivos de entrada. Mapeia as ações do usuário sobre os dispositivos de entrada em uma linguagem abstrata e interpreta operações abstratas de saída produzindo a tela. As entradas do usuário são *tokens* produzidos pelo componente de apresentação. A seq:uência em que estes *tokens* são entregues ao controle de diálogo guia o diálogo e produz informação para ser enviada à interface da aplicação. No sentido contrário, o controle de diálogo recebe informação da interface com a aplicação, seu estado pode ser modificado e provocar comandos abstratos de saída que serão interpretados pelo componente de apresentação. A interface da aplicação recebe pedidos do controle de diálogo que pode realizar mapeamentos antes de entregá-los à aplicação. Este componente pode responder diretamente aos pedidos do controle de diálogo quando o pedido não for sintaticamente válido, e transformar a saída da aplicação antes de transmiti-la ao controlador de diálogo.

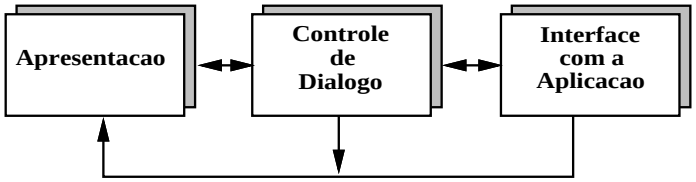

Figura 6: Modelo abstrato de um UIMS. Modelo de Seeheim [22]

A grosso modo pode-se considerar o componente de apresentação como o "nível léxico" do sistema interativo. Um analisador léxico pode ser utilizado para ocultar os detalhes de como os comandos e parâmetros podem ser fornecidos (p.ex., um comando pode ser originado da seleção de um menu e seu respectivo parâmetro de um ícone). O controlador de diálogo, "nível sintático", pode ser implementado como um analisador sintático, que recebe os *tokens* pré-processados do analisador léxico e verifica a correção sintática a medida que são obtidos.

5.3 Especificação de diálogo

Linguagens de especificação de diálogo podem ser classificadas segundo o modelo de controle de diálogo empregado. Três modelos são comentados abaixo, juntamente com notações típicas de cada modelo. Em [38] é apresentada uma lista de qualidades desejáveis em linguagens de especificação de diálogo. Geralmente as notações possuem ferramentas de apoio para construção e execução de descrições. Conforme [72], diagramas de transição e leiautes de tela são inadequados para o usuário na ausência de um interpretador destes diagramas.

Modelos de controle de diálogo

Comentários exaustivos sobre três modelos de diálogos (redes de transição, gramáticas e o modelo de eventos) podem ser obtidos em [23]. Abaixo são descritos os modelos e as notações típicas empregadas. Estes modelos podem ser usados como métodos para formalmente

especificar uma interface, possivelmente para servir de entrada para algum UIMS, que automaticamente gera o controle da interface.

A notação mais comum para interfaces é baseada em redes de transição. Diagramas de Transição de Estados (DTEs) é o exemplo mais significativo. Este modelo considera uma interface como uma coleção de estados ligados por arestas rotuladas com eventos. Estes eventos podem estar associados às ações do usuário, tempo e resultados (saída) da aplicação. Evento é a condição necessária para que ocorra uma transição entre estados. Uma característica positiva dos DTEs é a sua representação gráfica. *Statecharts* [26] é uma extensão dos DTEs empregada na especificação de sistemas reativos, em particular o controle de diálogo de uma interface [45]. *Statecharts* permitem descrições hierárquicas, descrever concorrência, comunicação e história.

Notações baseadas em gramáticas consideram o diálogo entre usuário e aplicação como como uma linguagem descrita por uma gramática. A gramática é especificada em BNF (Backus-Naur Form), que é amplamente empregada na especificação formal da sintaxe de linguagens de programação. A notação geralmente é estendida para permitir que código fornecido seja executado sempre que uma regra é utilizada.

Notações baseadas em eventos apresentam facilidades para a descrição de diálogos complexos, característicos de manipulação direta [17, 32]. Neste caso uma interface é vista como um conjunto de eventos e tratadores para estes eventos. Eventos podem ser gerados por ações do usuário sobre dispositivos de entrada, por tratadores de eventos ou em decorrência do resultado das aplicações.

5.4 Independência de diálogo

O desenvolvimento de sistemas interativos com pouca distinção entre o software do diálogo e o da aplicação conduz a interfaces de baixa qualidade, pois o projeto da interface é um processo cíclico envolvendo avaliação e correção. Isto prossegue até a obtenção de um projeto satisfatório. Se não há distinção entre estes elementos, há inúmeras dificuldades (resistências) que dificultam mudanças pois não podemos tratar questões de diferentes interesses de forma independente. Isto porque *separation of concerns* [21], um princípio básico, e sua conseq:uente realização em módulos distintos não são aplicados.

Em banco de dados há um problema similar: o isolamento desejável entre programas e dados, de tal forma que mudanças em um não afete o outro. A solução é conhecida por independência de dados. Uma solução análoga neste contexto é chamada de independência de diálogo, que é baseada na definição de um protocolo de comunicação entre a interface e a aplicação. A Figura 7 relaciona estes dois conceitos.

A independência de diálogo permite separar questões concernentes ao diálogo de um lado e a aplicação de outro. Note que a aplicação não se interessa na forma como dados de entrada foram obtidos (menus, linguagem de comandos e outros), mas nos dados propriamente ditos. Analogamente, a interface não se interessa pelo processamento dos dados, mas em como obtê-los do usuário e exibir os resultados das transformações neles realizadas.

A grande vantagem desta abordagem é manter a separação entre elementos de propósitos distintos (interface e aplicação). São vários os benefícios:

- Questões que dizem respeito a um componente tem influência limitada no outro.

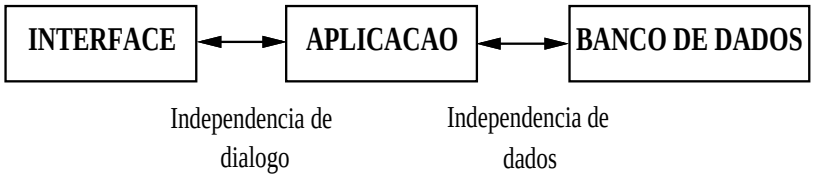

Figura 7: Conceitos análogos: independências de dados e diálogo.

- O desenvolvimento pode ser realizado independentemente, exceto quanto ao protocolo de comunicação entre eles.
- Interfaces diferentes para uma mesma aplicação podem ser desenvolvidas para grupos de usuários distintos. Por exemplo, por motivo de línguas diferentes, cultura ou experiência.

Como obter: isolar, em tempo de projeto, questões de diálogo e de aplicação [30].

Por fim, isto significa que um sistema interativo (Figura 2) é composto pelo componente de diálogo através do qual toda a comunicação entre o usuário e o sistema é realizada e o componente computacional, parte funcional do sistema que provê toda a semântica do sistema.

Em [30, 36] é comentado em mais detalhes as implicações desta separação.

5.5 Sistema de janela

Um marco nítido na história das atuais interfaces foi a criação do ambiente *Smalltalk*. Este ambiente permite o uso de várias janelas[2] sobrepostas.[3] As janelas podem ser selecionadas e transladadas na tela com o uso do mouse. A primeira versão deste ambiente, conforme [37], foi testada no Alto, um protótipo de computador desenvolvido no Xerox PARC (*Palo Alto Research Center*), um dos computadores mais fáceis de se usar até aquele momento. Conceitos empregados no Alto (*desktop, windows*, *WYSIWYG* e outros) foram refinados e novamente agrupados no XEROX Star em 1981. Usando conceitos apresentados nestas máquinas e com preço mais acessível surge o *Apple Macintosh* em 1984. Em 1985 é lançado o *Windows* — uma tentativa de fornecer aos usuários do computador mais difundido, o IBM-PC, facilidades semelhantes àquelas que os usuários dos computadores supracitados possuíam. O *Windows* consumiu 110 mil horas de programação. Resultado: em dezembro de 1989 já haviam sido vendidas 2 milhões de cópias. No fim de 1990 e início de 1991 eram

[2] Janela (*window*) é uma área geralmente retangular da tela. Através desta área um programa recebe entradas e emite os resultados do seu processamento. Geralmente os ambientes que permitem o uso de janelas são multiprogramados (*multiprogramming*). Neste caso cada janela pode corresponder a um processo executado concorrentemente.

[3] Janelas que podem ter uma interseção não nula, ao invés de janelas "azulejadas" (*tiled*).

vendidas 30 mil cópias por semana [37]. Grande parte deste sucesso é atribuído, sobretudo, à interface com o usuário. Para contrastar com o tempo de desenvolvimento do *Windows*, a interface para o Star consumiu 6 anos [10]. *Windows NT* é um sistema similar ao *Windows* do ponto de vista de apresentação. Por outro lado, acrescenta alguns mecanismos para melhor utilizar os recursos do equipamento utilizado. Muitas especulações [75] têm sido feitas sobre este produto.

Estes sucessos tornaram comum os sistemas de janelas. *X Window* [62] é muito utilizado em ambiente UNIX.

Sistema de Janela: gerencia recursos típicos de um ambiente de janela tais como eventos, criação e remoção de janelas, dispositivos de entrada e saída (*mouse*, teclado, vídeo e outros).

Um sistema de janela fornece mecanismos para gerenciar telas usando janelas. As facilidades fornecidas, no entanto, são de baixo nível e requer esforço substancial do programador para apresentar uma simples janela. O gerenciador de janela (*window manager*) permite o controle da tela pelos usuários através da manipulação de janelas. A interface fornecida pelo gerenciador de janela é percebida, incorretamente, como todo o sistema de janela. O gerenciador é apenas uma das interfaces de um sistema de janela! Uma taxonomia para gerenciadores de janela é apresentada em [49].

Há vários termos associados aos sistemas de janela. Um dos mais importantes refere-se ao modelo de imagem empregado.

Modelo de Imagem: modelo fornecido por um sistema de janela para expressar a saída, p.ex., no modelo de pixel (picture X element) a tela é vista como uma matriz de pontos que podem ter suas cores alteradas. Outros exemplos são GKS (*Graphical Kernel System*), PHIGS (*Programmer's Hierarchical Interface to Graphics*), *QuickDraw* e PostScript. Veja mais detalhes em [5, pág. 62].

---

6 Avaliação

The complexity of both humans and computing systems makes their interaction less predictable than we would like. Even the best intentions can result in unusable systems or, more often, in systems with problems.
Perlman [59]

Avaliação de projeto geralmente apresenta-se como uma atividade necessária no desenvolvimento de sistemas, não exclusivamente de software. No mundo real, a implementação de um sistema, seguida de sua avaliação e posterior correção de erros é impraticável na

grande maioria dos casos. Projetistas geralmente manipulam modelos, que só após um mínimo de "validação" são implementados. Esta regra também é aplicável ao desenvolvimento de interfaces. O papel da avaliação é verificar (projeto) se realmente o sistema comporta-se como esperado e atende os requisitos dos usuários. No contexto de interfaces há alguns indícios favoráveis a realização desta atividade. O "censo comum", por exemplo, é algo comumente empregado durante o projeto. No entanto, não pode ser aplicado de forma confiável e muitos resultados de pesquisas têm contradito este princípio. Outro indício diz respeito a posição do projetista, que em geral assume seu comportamento como representativo, o que é uma falácia. Estes e outros problemas são comentados em [14].

A avaliação pode ser executada durante todo o ciclo de vida do projeto. Ou antes da existência de um sistema executável envolvendo apenas os projetistas. No entanto, isto não substitui o teste com as pessoas para as quais o sistema é projetado: os usuários. Neste último caso, entretanto, há o envolvimento de implementação do sistema em alguma forma: desde simulação, protótipos até o sistema completamente implementado. Na pág. 14, veja **Protótipos**, é apresentado uma técnica alternativa de avaliação de projeto com o usuário final sem a necessidade de código executável ter sido gerado. Esta avaliação, no entanto, requer grande habilidade da equipe que a emprega.

Várias técnicas podem ser usadas na avaliação. Os métodos analíticos empregam modelos formais de interação. Os métodos empíricos variam de informais (observação do usuário) a formais (questionários, entrevistas e experimentos desenvolvidos formalmente). Abaixo são citados alguns métodos:

- *Cenários.* Fornece indicação de conceitos a serem capturados na interface.
- *Manual do usuário preliminar.* Durante a escrita de detalhes específicos pode-se descobrir melhores métodos de operação.
- *Mock-up.* Demonstra a aparência da interface.
- *Simulações prévias.* Demonstra o comportamento.
- *Demonstrações.* Mostra um limitado subconjunto da funcionalidade a ser implementada.
- *Think aloud.* Fornece uma idéia dos pensamentos do usuário enquanto está operando um protótipo ou versão preliminar do sistema.
- *Teste formal de protótipo.* Fornece informações acerca de quão bem tarefas podem ser realizadas. Usa técnicas de pesquisas em fatores humanos e estatística.

7 Orientação a objetos e interfaces

O paradigma de objetos é freq:uentemente relacionado a vários aspectos de interfaces, inclusive do ponto de vista metodológico de desenvolvimento [46]. Esta seção mostra exemplos deste relacionamento. Conforme [41], o paradigma é explorado para o desenvolvimento (projeto e implementação), apresentação e integração de interfaces.

1. *Projeto e implementação.* Devido à quantidade de rotinas e o grande número de parâmetros que geralmente apresentam, um programador não usa diretamente a interface de programação de um sistema de janelas. Ele faz uso de uma camada orientada a objeto sobre a interface de programação. Esta camada equivale a objetos de interação. Como exemplos existem MacApp para o Macintosh e Actor para *Windows*. Uma descrição destas duas camadas pode ser vista em [41].

   A camada orientada a objetos fornece uma hierarquia de classes predefinidas. Elas reduzem a quantidade de código a ser gerada pelo programador e fornecem consistência à interface. Encapsulam o comportamento e a apresentação de elementos comuns como menus, janelas, e ícones. O mecanismo de herança pode ser utilizado para herdar comportamento e estrutura de classes existentes. Os elementos herdados podem ser redefinidos ou estendidos. Mais detalhes de que tipo de facilidades e como podem ser usadas são apresentados na descrição do *InterViews* em [44].

2. *Apresentação.* Nas modernas interfaces o usuário geralmente seleciona ou move ícones e objetos sobre a tela. Estes objetos e ícones são representações gráficas de conceitos do mundo real e, portanto, reagem a mensagens recebidas bem como comunicam-se com outros objetos. Por exemplo, nestes sistemas, arquivos podem ser manipulados como objetos. Para remover um objeto a mensagem remova é enviada ao objeto correspondente.

3. *Integração.* Este emprego é mais sutil que os demais. Em vez de diretamente usufruir do paradigma de objetos, faz uso de conceitos como objeto complexo e identidade de objeto. Análogo à confecção de um objeto complexo, uma aplicação complexa pode ser obtida pela integração de informação de um conjunto de aplicações. O usuário pode desenvolver um relatório (objeto complexo) em que gráficos foram gerados de resultados obtidos de uma planilha eletrônica, que obteve informações de um banco de dados. Convém notar que os dados não são simplesmente transferidos de uma aplicação para outra. Uma alteração em um objeto causa a alteração do estado do objeto complexo.

8 Onde obter mais informações

SIGCHI Bulletin da ACM contém um variado elenco de informações (comentários e lançamentos de livros, projetos, lista de eventos e assim por diante) para os interessados nesta área. [30, 31] podem ser utilizadas como referências iniciais. *HCI Bibliography Project* mantém uma base de dados com milhares de referências sobre este assunto. Esta base de dados pode ser obtida via `ftp` anônimo no diretório `pub/hcibib` em `archive.cis.ohio-state.edu`.

O *ACM SIGCHI Curricula for Human-Computer Interaction* contém recomendações da ACM para a grade curricular de interação homem-computador [65].

O emprego de métodos formais em interfaces homem-computador é discutido em [1, 27].

A proposta de um UIMS é defendida em [66].

O Projeto Xchart encontra-se em andamento no Dept. de Ciência da Computação (IMECC/UNICAMP). Este projeto é orientada para a construção de ferramentas de apoio

a implementação de interfaces. Na primeira página são fornecidos endereços para contato com o grupo executor do projeto.

9 Considerações finais

Este texto aborda vários aspectos do desenvolvimento de interfaces, não só do ponto de vista do produto (p.ex., estilos de interação) quanto também do processo (p.ex., ciclo de vida). As informações apresentadas são comentadas sucintamente. No entanto, acreditamos que o "todo" sirva de arcabouço onde mais informações podem ser obtidos, conforme as referências fornecidas e o interesse do leitor. Neste sentido, buscou-se fornecer uma visão geral da área enfatizando aspectos de interesse dos profissionais de computação e sugerindo uma estrutura para guiar estes profissionais na vasta bibliografia existente.

10 Agradecimentos

Em particular, gostaríamos de agradecer as valiosas sugestões de Osvaldo Severino Júnior.

Referências

[1] Gregory D. Abowd. *Formal Aspects of Human-Computer Interaction*. PhD thesis, University of Oxford, University of York – Department of Computer Science — Heslington, York, Y01 5DD, England, June 1991.

[2] David M. Andersen and Bruce A. Sherwood. Portability and the GUI. *Byte*, 16(12):221–226, November 1991.

[3] Ronald M. Baecker and William A. S. Buxton, editors. *Readings in Human-Computer Interaction – A Muldisciplinary Approach*. Morgan Kaufmann Publishers, Inc., 1987. In "Introduction".

[4] Lon Barfield. *The User Interface - Concepts & Design*. Addison-Wesley Publishing Company,Inc., 1993. ISBN 0-201-54441-5.

[5] Len Bass and Joëlle Coutaz. *Developing Software for the User Interface*. SEI Series in Software Engineering. Addison-Wesley Publishing Company, Inc., 1991.

[6] Daniel G. Bobrow, Sanjay Mittal, and Mark J. Stefik. Expert Systems: Perils and Promise. *Communications of the ACM*, 29(9):880–894, September 1986.

[7] Susanne Bϕdker. *Through the Interface: A Human Activity Approach to User Interface Design*. Lawrence Erlbaum Associates, Inc., 1991. ISBN 0-8058-0570-2.

[8] John M. Carrol, Robert L. Mack, and Wendy A. Kellogg. Interface Metaphors and User Interface Design. In Martin Helander, editor, *Handbook of Human-Computer Interaction*. Elsevier Science - Publishers B.V, 1988.

[9] Uli H. Chi. Formal Specification of User Interfaces: A Comparison and Evaluation of Four Axiomatic Approaches. *IEEE Transactions on Software Engineering*, SE-11(8):671–685, August 1985.

[10] Joëlle Coutaz. Abstractions for User Interface Design. *IEEE Computer*, 18(09):21–34, September 1985.

[11] Bill Curtis. Japan's Research Focus Shifts to Interfaces. *IEEE Software*, November 1991.

[12] J. F. DeSoi and W. M. Lively. Survey and Analysis of Nonprogramming Approaches to Design and Development of Graphical User Interfaces. *Information and Software Technology*, 33(6):413–424, July 1991.

[13] Alan Dix, Janet Finlay, Gregory Abowd, and Russel Beale. *Human-Computer Interaction*. Prentice-Hall, 1993.

[14] Andy Downton, editor. *Engineering the Human-Computer Interface*. McGraw-Hill, 1991. ISBN 0–7-707321-5.

[15] Stephen W. Draper and Donald A. Norman. Software Engineering for User Interfaces. *IEEE Transactions on Software Engineering*, SE-11(3):252–258, March 1985.

[16] Clarence Ellis, Simon Gibbs, and Gail Rein. Groupware: Some Issues and Experiences. *Communications of the ACM*, 34(1):38–58, January 1991.

[17] Mark A. Flecchia and R. Daniel Bergeron. Specifying Complex Dialogs in ALGAE. In *Proceedings of the ACM CHI+GI'87*, pages 229–234, April 1987.

[18] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Company, Inc., second edition, 1990.

[19] Jason Gait. An Aspect of Aesthetics in Human-Computer Communications: Pretty Windows. *IEEE Transactions on Software Engineering*, SE-11(8):714–717, August 1985.

[20] Jason Gait. Pretty Pane Tiling of Pretty Windows. *IEEE Software*, pages 9–14, September 1986.

[21] Carlos Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice-Hall, Inc., 1991.

[22] Mark Green. Report on Dialogue Specification Tools. In Günther E. Pfaff, editor, *User Interface Management Systems*, pages 9–20. Springer-Verlag, 1985.

[23] Mark Green. A Survey of Three Dialogue Models. *ACM Transactions on Graphics*, 5(3):244–275, July 1986.

[24] Frits Habermann. Giving Real Meaning to 'easy-to-use' Interfaces. *IEEE Software*, pages 90–91, July 1991.

[25] Andrew Harbert, William Lively, and Sallie Sheppard. A Graphical Specification System for User-Interface Design. *IEEE Software*, 7(4):12–20, July 1990.

[26] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[27] Michael Harrison and Harold Thimbleby, editors. *Formal Methods in Human-Computer Interaction.* Cambridge University Press, 1990.

[28] H. Rex Hartson and Deborah Hix. Toward Empirically Derived Methodologies and Tools for Human-Computer Interface Development. *Int. J. Man-Machine Studies*, pages 477–494, 1989.

[29] H. Rex Hartson, Antonio C. Siochi, and Deborah Hix. The UAN: A User-Oriented Representation for Direct Manipulation Interface Designs. *ACM Transactions on Information Systems*, 8(3):181–203, July 1990.

[30] H.R. Hartson and D. Hix. Human-Computer Interface Development: Concepts and Systems for its Management. *ACM Computing Surveys*, 21(1):5–92, March 1989.

[31] Martin Helander, editor. *Handbook of Human-Computer Interaction.* North-Holland, 1988.

[32] Ralph D. Hill. Event-Response Systems — A Technique for Specifying Multi-Thread Dialogues. In *Proceedings of the ACM CHI+GI'87*, pages 241–248, April 1987.

[33] Deborah Hix. Generations of User-Interface Management Systems. *IEEE Software*, pages 77–89, September 1990.

[34] Deborah Hix and H. Rex Hartson. *Developing User Interfaces: Ensuring Usability Through Product & Process.* John Wiles & Sons, Inc., 1993. ISBN 0471-57813-4.

[35] Don Hopkins. The Design and Implementation of Pie Menus. *Dr. Dobb's Journal*, pages 16–26, December 1991.

[36] William D. Hurley and John L. Sibert. Modeling User Interface-Application Interactions. *IEEE Software*, pages 71–77, January 1989.

[37] Daniel Ichbiah and Susan Knepper. *MICROSOFT.* Editora Campus, Rio de Janeiro, 1992. Tradução do original: *The Making of Microsoft.*

[38] Robert J. K. Jacob. Using Formal Specifications in the Design of a Human-Computer Interface. *Communications of the ACM*, 26(4):259–264, April 1983.

[39] Robert J. K. Jacob. A Specification Language for Direct-Manipulation User Interfaces. *ACM Transactions on Graphics*, 5(4):283–317, October 1986.

[40] Elieser Kantorowitz and Oded Sudarsky. The Adaptable User Interface. *Communications of the ACM*, 32(11):1352–1358, November 1989.

[41] Setrag Khoshafian and Razmik Abnous. *Object Orientation: Concepts, Languages, Databases, User Interfaces*, chapter User Interfaces, pages 323–387. John Wiley & Sons, Inc., 1990.

[42] Thomas K. Landauer. Research Methods in Human-computer interaction. In M. Helander, editor, *Handbook of Human-Computer Interaction*, chapter 42, pages 905–927. Elsevier Science - Publishers B.V, North-Holland, 1988.

[43] Brenda Laurel, editor. *The Art of Human-Computer Interface Design*. Addison-Wesley Publishing Company, Inc., 1990. ISBN 0-201-51797-3.

[44] Mark A. Linton, Paul R. Calder, and John M. Vlissides. InterViews: A C++ Graphical Interface Toolkit. Center for Integrated Systems/Stanford, CA 94305.

[45] Fábio N. Lucena and Hans K.E. Liesenberg. Reflections on Using Statecharts to Capture User Interface Behaviour. *Proceedings of XIV Int. Conf. of the Chilean CSS*, October 1994.

[46] Susan E. McDaniel, Gary M. Olson, and Judith S. Olson. Methods in Search of Methodology – Combining HCI and Object Orientation. In *Human Factors in Computing Systems CHI'94 Conference Proceedings*, pages 145–151, 1994.

[47] G. A. Miller. The Magical Number 7, plus or minus 2: Some Limits of our Capacity for Processing Information. *Psychological Review*, 1956. Number 63.

[48] Rolf Molich and Jakob Nielsen. Improving a Human-Computer Dialogue. *Communications of the ACM*, 33(3):338–348, March 1990.

[49] Brad A. Myers. A Taxonomy of Window Manager User Interfaces. *IEEE Computer Graphics & Applications*, pages 65–84, September 1988.

[50] Brad A. Myers. Encapsulating Interactive Behaviors. In *Human Factors in Computing Systems, Proceedings SIGCHI'89*, pages 319–324, Austin,TX, April 1989.

[51] Brad A. Myers. Demonstrational Interfaces: A Step Beyond Direct Manipulation. *IEEE Computer*, pages 61–73, August 1992.

[52] Brad A. Myers, editor. *Languages for Developing User Interfaces*. J&B, 1992.

[53] Brad A. Myers. Challenges of HCI Design and Implementation. *Interactions*, 1(1):73–83, 1994.

[54] Brad A. Myers and Mary Beth Rosson. Survey on User Interface Programming. In *CHI'92 Proceedings*, pages 195–202, Monterey, California, May 1992.

[55] Peter G. Neumann. Flaws in Specification and What to do about Them. *ACM SIGSOFT Engineering Notes*, 14(3), May 1989.

[56] William M. Newman and Robert F. Sproull. *Principles of Interactive Computer Graphics*, chapter 28, pages 443–478. McGraw-Hill, Inc., second edition, 1979.

[57] Jakob Nielsen. Iterative User-Interface Design. *Computer*, 26(11):32–41, November 1993.

[58] Randy Pausch, Matthew Conway, and Robert DeLine. Lessons Learned from SUIT, the Simple User Interface Toolkit. *ACM Transactions on Information Systems*, 10(4):320–344, October 1992.

[59] Gary Perlman. User Interface Development. Technical Report SEI-CM-17-1.1, Software Engineering Institute, Carnegie Mellon University, 1989.

[60] Marc Rettig. Prototyping for Tiny Fingers. *Communications of the ACM*, 37(4):21–27, April 1994.

[61] Mattew D. Russell, Howard Xu, and Lingtao Wang. Action Assignable Graphics - A Flexible Human-Computer Interface Design Process. In *Human Factors in Computing Systems CHI'92 Conference Proceedings*, pages 71–72, Monterey, California, May 1992.

[62] Robert W. Scheifler and Jim Gettys. The X Window System. *ACM Transactions on Graphics*, 5(2):79–109, April 1986.

[63] Ben Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16(8):57–69, August 1983.

[64] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Publishing Company, Inc., 1987.

[65] ACM SIGCHI. Curricula for Human-Computer Interaction, 1992. ISBN 0-89791-474-0.

[66] Gurminder Singh. *Automating the Lexical and Syntactic Desigh of Graphical User Interfaces*. PhD thesis, University of Alberta, Edmonton, Alberta, 1989.

[67] Gurminder Singh. VU: Visual User-Interface Design. In *The Visual Computer*, pages 230–241. Springer-Verlag, 1990.

[68] H. W. Six and J. Voss. User Interface Development: Problems and Experiences. *Lecture Notes in Computer Science*, 555:306–319, June 1991.

[69] Sidney L. Smith and Jane N. Mosier. Guidelines for Designing User Interface Software. Technical Report ESD-TR-86-278, US Air Force, 1986. Distribution unlimited. Anonymous ftp: `archive.cis.ohio-state.edu`, `pub/hci/Guidelines`.

[70] R. Summersgill and D. P. Browne. Human Factors: Its Place in System Development Methods. *ACM SIGSOFT Engineering Notes*, 14(3):227–234, May 1989.

[71] Harold Thimbleby. *User Interface Design*. ACM Press, 1990. ISBN 0-201-41618-2.

[72] Anthony I. Wasserman, Peter A. Pircher, David T. Shewmake, and Martin L. Kersten. Developing Interactive Information Systems with the User Software Engineering Methodology. *IEEE Transactions on Software Engineering*, SE-12(2):326–345, February 1986.

[73] James wilson and Daniel Rosenberg. Rapid Prototyping for User Interface Design. In M. Helander, editor, *Handbook of Human-Computer Interaction*, chapter 39, pages 859–875. Elsevier Science Publishers B.V., 1988.

[74] Michael Wilson and Anthony Conway. Enhanced Interaction Styles for User Interfaces. *IEEE Computer Graphics & Applications*, 11(2):79–90, March 1991.

[75] Tom Yager and Ben Smith. Is Unix Dead? *Byte*, 17(9):134–146, September 1992.

Índice

Relatórios Técnicos – 1992

92-01 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

92-02 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

92-03 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

92-04 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

92-05 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

92-06 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

92-07 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

92-08 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

92-09 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

92-10 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

92-11 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

92-12 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

93-01 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

93-02 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

93-03 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

93-04 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

93-05 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

93-06 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

93-07 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

93-08 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

93-09 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

93-10 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

93-11 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

93-12 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

93-13 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

93-14 **Managing Time in Object-Oriented Databases,** *Lincoln M. Oliveira, Claudia Bauzer Medeiros*

93-15 **Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures,** *Nabor das Chagas Mendonça, Ricardo de Oliveira Anido*

93-16 $\mathcal{LL}$ – **An Object Oriented Library Language Reference Manual,** *Tomasz Kowaltowski, Evandro Bacarin*

93-17 **Metodologias para Conversão de Esquemas em Sistemas de Bancos de Dados Heterogêneos,** *Ronaldo Lopes de Oliveira, Geovane Cayres Magalhães*

93-18 **Rule Application in GIS – a Case Study,** *Claudia Bauzer Medeiros, Geovane Cayres Magalhães*

93-19 **Modelamento, Simulação e Síntese com VHDL,** *Carlos Geraldo Krüger e Mário Lúcio Côrtes*

93-20 **Reflections on Using Statecharts to Capture Human-Computer Interface Behaviour,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-21 **Applications of Finite Automata in Debugging Natural Language Vocabularies,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-22 **Minimization of Binary Automata,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-23 **Rethinking the DNA Fragment Assembly Problem,** *João Meidanis*

93-24 EGOLib — **Uma Biblioteca Orientada a Objetos Gráficos,** *Eduardo Aguiar Patrocínio, Pedro Jussieu de Rezende*

93-25 **Compreensão de Algoritmos através de Ambientes Dedicados a Animação,** *Rackel Valadares Amorim, Pedro Jussieu de Rezende*

93-26 GeoLab: **An Environment for Development of Algorithms in Computational Geometry,** *Pedro Jussieu de Rezende, Welson R. Jacometti*

93-27 **A Unified Characterization of Chordal, Interval, Indifference and Other Classes of Graphs,** *João Meidanis*

93-28 **Programming Dialogue Control of User Interfaces Using Statecharts,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-29 EGOLib – **Manual de Referência,** *Eduardo Aguiar Patrocínio e Pedro Jussieu de Rezende*

Relatórios Técnicos – 1994

94-01 **A Statechart Engine to Support Implementations of Complex Behaviour,** *Fábio Nogueira de Lucena, Hans K. E. Liesenberg*

94-02 **Incorporação do Tempo em um SGBD Orientado a Objetos,** *Ângelo Roncalli Alencar Brayner, Claudia Bauzer Medeiros*

94-03 **O Algoritmo KMP através de Autômatos,** *Marcus Vinícius A. Andrade e Cláudio L. Lucchesi*

94-04 **On Edge-Colouring Indifference Graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

94-05 **Using Versions in GIS,** *Claudia Bauzer Medeiros and Geneviève Jomier*

94-06 **Times Assíncronos: Uma Nova Técnica para o Flow Shop Problem,** *Hélvio Pereira Peixoto e Pedro Sérgio de Souza*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

reltec@dcc.unicamp.br