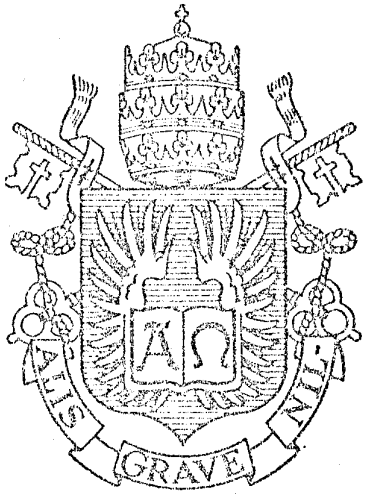


PUC

Série: Monografias em Ciência da Computação, Nº 20/88

O DESENVOLVIMENTO DE INTERFACES

Maria Helena B. Braz

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

IA MARQUÊS DE SÃO VICENTE, 225 – CEP 22453

RIO DE JANEIRO – BRASIL

PUC/RJ - DEPARTAMENTO DE INFORMÁTICA

Série: Monografias em Ciência da Computação, Nº 20/88
Editor: Paulo A. S. Veloso Dezembro, 1988

O DESENVOLVIMENTO DE INTERFACES*

Maria Helena B. Braz

* Apresentado pelo Prof. Carlos José Pereira de Lucena.
Trabalho parcialmente financiado pela FINEP.

Responsável por Publicações

Rosane Teles Lins Castilho
PUC/RJ-Depto. de Informática
Assessoria de Biblioteca, Documentação e Informação
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ

RESUMO

Este trabalho apresenta os principais contributos da Engenharia de Software no desenvolvimento de interfaces Homem-Máquina e apresenta exemplos de ambientes de desenvolvimento adequados à criação de sistemas interativos.

PALAVRAS-CHAVE

Interfaces com o Usuário; Ambientes de Desenvolvimento de Software; Sistemas Interativos.

ABSTRACT

In this work fundamental contributions of Software Engineering for the development of user-computer interfaces are analysed. Also some software environments which support the development of interative systems are discribed.

KEYWORDS

User Interfaces; Software Development Environments; Interactive Systems.

S U M Á R I O

I - INTRODUÇÃO.

Neste texto apresentamos uma análise sobre questões relativas a Interfaces Homem Máquina e Engenharia de Software.

Esta análise pretende dar resposta a quatro questões motivadas pelo seguinte problema:

"O projeto, implementação e verificação de um software que satisfaça um conjunto de especificações funcionais é, por si só, um problema muito complexo. Se à funcionalidade do software forem acrescentados os requisitos de comunicação Homem-Máquina, o problema de validação pode se tornar intratável"

Assim, com base neste problema, pergunta-se:

1 - A interface deve ser projetada junto com o software?

2 - E possivel desacoplar consideravelmente (até que ponto ?) o software da sua interface?

3 - O que significa e como se consegue validar uma interface?

4 - Como ferramentas para o projeto de interfaces tratam o problema acima?

Para dar respostas a estas questões iremos analisar na seção II deste trabalho os aspectos conceituais ligados ao desenvolvimento de interfaces procurando acentuar as posições de diferentes autores. Na seção III analisaremos as ferramentas propostas para apoiar esse desenvolvimento verificando como elas se enquadram nos princípios de desenvolvimento indicados na seção II.

Finalmente, na seção IV, apresentaremos as conclusões obtidas pela anàlise efetuada dando uma resposta sucinta para as questões colocadas.

II - O DESENVOLVIMENTO DE INTERFACES HOMEM MÁQUINA.

II.1 - Projeto Modular de Sistemas Interativos, suas consequências no desenvolvimento de Interfaces.

A notável expansão dos sistemas interativos pelas mais diversas áreas de aplicação e a necessidade de dirigi-los a usuários menos especializados, obrigaram ao estudo dos problemas relacionados com o desenvolvimento de interfaces Homem Máquina visando a criação de sistemas interativos de alta qualidade.

A importância desse estudo fica ainda mais ressaltada quando alguns trabalhos evidenciam que uma quantidade crescente de recursos é utilizada no desenvolvimento de interfaces chegando em alguns casos a atingir valores superiores a 50% dos recursos totais envolvidos.[12]

O alto custo de um sistema interativo de qualidade resulta em grande parte de se desenvolver as componentes responsáveis pela interface de forma "ad hoc" devido a não existirem ainda definidos princípios de desenvolvimento precisos e eficientes orientando esta atividade.

Na realidade o projeto de interfaces é ainda uma área pouco consolidada não só pela inexistência de estratégias de desenvolvimento, geralmente aceitas, mas também pela dificuldade de estabelecer princípios que necessariamente deverão incorporar conhecimento de áreas tão diversas como psicologia, ergonomia, engenharia de software, etc...

Na maioria dos casos os princípios utilizados dizem respeito apenas a aspectos ligados á interação Homem Máquina e

tentam orientar na solução de problemas de carater ergonômico tais como a melhor formatação de telas e o uso mais eficiente do teclado.

Estes principios resultam geralmente da generalização de resultados experimentais obtidos por observação do desempenho dos usuários dos sistemas interativos e, por isso mesmo, focam geralmente aspectos externos desses sistemas e pouco ou nada contribuem para o processo de desenvolvimento em si.

Na realidade o desenvolvimento de um sistema interativo necessitará a definição clara das funções da aplicação e também da forma como estas funções serão invocadas e se comunicarão com o usuário.

Constatando que, a funcionalidade da aplicação é distinta da funcionalidade do interface, o projeto modular de sistemas sugere então, como principio orientador do desenvolvimento, que se procure separar os módulos responsáveis pela comunicação Homem Máquina dos módulos que suportam as funções da aplicação.

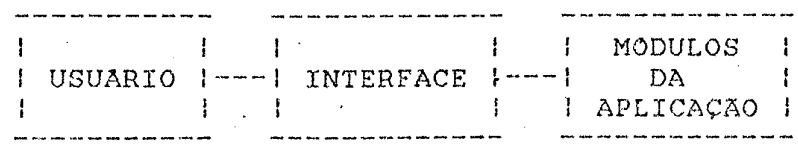


fig.1 - Estrutura de um Sistema Interativo.

Na realidade este princípio, geralmente proposto em toda literatura, define uma arquitetura para os sistemas interativos como se ilustra na figura 1 onde o usuário se comunica com os módulos de aplicação unicamente através da interface.

Desde que o protocolo de comunicação entre os módulos da interface e da aplicação tenha sido devidamente estabelecido, será possível um desenvolvimento em paralelo destes módulos.

As principais vantagens desta abordagem modular são fundamentalmente:

1) Os projetos da interface e dos módulos de aplicação podem ser realizados por especialistas em cada uma das áreas aumentando a qualidade dos componentes.

2) O desenvolvimento independente da interface, permite que esta seja ajustada de forma iterativa adaptando-se às necessidades, geralmente mal definidas, dos usuários.

3) O diálogo poderá ser partilhado por diferentes aplicações baixando o custo de desenvolvimento e garantindo ao usuário interfaces mais consistentes que diminuem o esforço de aprendizagem.

Esta estratégia de desenvolvimento de sistemas interativos é largamente aceita na literatura e é definida explicitamente em [1] [2] [4] [5] [7] [9] [10], ou aceita implicitamente em [3] [6] [8].

Assim, parece razoável tratar isoladamente o problema da interface considerando que ela será constituida por uma linguagem de entrada para o usuário, uma linguagem saida para a máquina e um protocolo de comunicação e de controle da execução

dos módulos da aplicação, tornando-a, então, responsável por toda a comunicacão entre a aplicação e o usuário.

II.2 - O Processo de Desenvolvimento.

Considerando o que já foi dito no ponto anterior uma interface poderá ser vista como um software capaz de aceitar uma linguagem de entrada do usuário, apresentar resultados numa outra linguagem e gerenciar protocolos para interação e para controle da execução dos módulos de aplicação.

Esta visão de interfaces, adotada no presente trabalho, não considera como pertencendo à interface aspectos perceptivos e cognitivos do usuário.

No atual estado estes aspectos influenciam apenas na medida em que orientam os projetistas na concepção das linguagens de comunicação e do protocolo de interação.

A medida que o processo de projeto de interfaces for sendo mais conhecido é provável que estes aspectos possam ser incluidos nas interfaces, permitindo um comportamento mais inteligente para os sistemas interativos.

II.2.1- O Ciclo de Vida.

Considerando o desenvolvimento de interface como um processo de desenvolvimento de software iremos analisar como se adequam a este caso os modelos propostos na literatura para o ciclo de vida de desenvolvimento de software.

O modelo tradicional prevê a existência das seguintes fases: Especificação; Projeto; Implementação; Avaliação e Testes; e Manutenção.

A aplicação deste modelo ao desenvolvimento de

interfaces prevê portanto a construção inicial de uma especificação precisa para a interface. No entanto a criação de tal especificação necessita incluir aspectos que são dependentes do usuário e que, por isso mesmo, poderão não ser bem conhecidos ou ser até variáveis no tempo.

Esta caracteristica da interface aponta então para a necessidade de um modelo de desenvolvimento baseado em prototipação. Segundo este modelo existirá então uma primeira fase onde é criado um protótipo permitindo avaliar as necessidades dos usuários no que diz respeito à interface e uma segunda fase onde é desenvolvido o software final de acordo com as necessidades estabelecidas e com a qualidade necessária.

Este modelo geralmente é o mais utilizado e para lhe dar suporte são criadas ferramenta permitindo uma prototipação rápida da interface e facilitando as alterações ao protótipo.

O uso de um protótipo para determinar os requisitos da interface de um sistema facilita a comunicação com o usuário e oferece-lhe uma forma mais real de avaliação do sistema proposto e de deteção de erros. Repare-se que como a especificação da interface pode ser vista como a definição das linguagens de entrada e saida e, portanto, dos seus aspectos léxicos, sintáticos e semânticos a um nível adequado ao projeto, tal definição deverá ser orientado pelas preferências do usuário e só um protótipo poderá exemplificar devidamente essas linguagens.

Do ponto de vista da Engenharia de Software o principal problema dessa abordagem resulta do fato do código de implementação do protótipo ser projetado sob o objetivo "rapidez" e, sendo dificil abandonar todo este código, haver a

tendência de o reutilizar partindo para um sistema final menos confiável e menos manutenível.

O enfoque de prototipação para o desenvolvimento de interfaces é amplamente defendido na literatura, [1],[4],[5],[9] e [10] são exemplos desse posicionamento.

Na realidade a diversidade de fatores a considerar na especificação, não só do ponto de vista dos equipamentos disponíveis e das aplicações pretendidas como dos usuários envolvidos, torna extremamente difícil a criação de regras gerais estabelecendo as características das interfaces.

Se o objetivo principal de uma interface é estabelecer a comunicação entre aplicações e usuários garantindo a satisfação destes últimos e a sua produtividade, seria necessário para poder equacionar regras formais de especificação possuir um modelo adequado e bem estabelecido do usuário o que pressupõe necessariamente um conhecimento muito mais profundo que o atual sobre o mesmo.

Esta barreira de conhecimento conduz então, à necessidade de experimentação sendo fundamental retirar de cada experiência contributos para um melhor conhecimento dos princípios que deverão orientar a construção de interfaces.

Em resumo podemos considerar que, do ponto de vista metodológico, o desenvolvimento de interfaces é ainda altamente experimental, orientado por recomendações sobre a construção do diálogo, das entradas e saídas e apoiado em ferramentas para prototipação rápida.

Essas recomendações são deduzidas de experiências no

desenvolvimento de interfaces e, quando utilizadas em situações similares às que lhe deram origem, procuram garantir na interface projetada os seguintes aspectos principais:

. facilidade de uso,

. rapidez na aprendizagem,

. robustez quanto a uso inadequado,

. alta produtividade e satisfação por parte do usuário.

Outro aspecto cujo estudo se torna necessário é a validação das interfaces. Será importante procurar estabelecer na fase de especificação critérios que permitam a sua posterior avaliação.

Não existindo regras claras para a especificação e, dada a diversidade de fatores envolvidos, existem propostas para que a especificação da interface se baseie em critérios de aceitação do tipo:

" Após 75 minutos de treinamento, 40 usuários típicos deverão conseguir executar 80% das tarefas estipuladas para aceitação do sistema, em 35 minutos, com menos de 12 erros " [11].

Esta abordagem permite deixar o projeto da interface sob a responsabilidade da equipe de desenvolvimento de software mas com objetivos explícitos para a qualidade esperada do sistema. A validação da interface poderá então, ser realizada de forma mais concreta.

Na realidade o teste da inteface deverá ser feito nas condições estipuladas por esses critérios e, quando eles não forem verificados considerar-se-á a ocorrência de uma falha na interface. A coleta de informação sobre a interface em testes realizados pelo usuário é, no entanto, um problema em aberto.

Na execução dos testes o usuário poderá sentir-se avaliado e, por isso, passar uma informação menos correta sobre o uso do sistema. Também a ocorrência de um erro da interface pode ser interpretado como um erro de utilização.

Este tipo de problema poderá ser minimizado recorrendo a questionários sobre uso cobrindo exaustivamente todas as partes da interface ou colocando observadores experimentados acompanhando os testes.

Um outro recurso poderá ser a criação de ferramentas registrando todas as ações do usuário e permitindo posterior análise do seu comportamento. Estes registros correspondem para o projeto da interface aos "Dumps" de memória usados na depuração de programas tradicionais.

Finalmente e, como último ponto a referir dentro do ciclo de vida temos a documentação. No caso das interfaces a documentação não se pode restringir à tradicional para os projetistas mas também deve prover as necessidades dos usuários. Para satisfazer os usuários deveremos ter em mente que a documentação deve fornecer as respostas que o usuário necessita, de uma forma rapidamente acessível.

O usuário recebe informação por diferentes canais como manuais, mensagens de erro, telas, e é necessário garantir que essas fontes de informação correspondem às necessidades do usuário tanto quanto ao tipo de informação como à oportunidade da mesma ser fornecida.

Analisamos nesta seção os principais problemas do ponto de vista de Engenharia de Software com que se defrontam os

projetistas de interface. Verificamos a necessidade de um desenvolvimento iterativo baseado em prototipação com alta participação dos usuários e, para diminuir o esforço de programação no desenvolvimento de interfaces, verificamos a necessidade de criar ferramentas que apoiem o processo de desenvolvimento.

Na seção seguinte analisaremos as abstrações a considerar em sistemas interativos para facilitar a definição de tais ferramentas.

II.2.2 - Abstrações para o projeto de interfaces com o usuário.

A comunicação da máquina com o usuário é feita através de dispositivos físicos cujas características são muito variáveis.

O desenvolvimento de interfaces obrigaria a enormes esforços de programação caso fosse necessário conhecer os detalhes de controle de tais equipamentos.

Para facilitar o uso destes dispositivos foram criadas primitivas gráficas que dão ao projetista uma visão abstrata desses dispositivos e escondem os detalhes particulares de cada um.

Dentro desta idéia base foram desenvolvidos pacotes gráficos isolando o software das aplicações do software para controle dos dispositivos físicos.

Atualmente existe uma norma internacional que estabelece um padrão gráfico designado por GKS-Grafical Kernel System. e, neste padrão, são utilizados conceitos como estação de

trabalho, entradas lógicas, primitivas de saida, etc...

Destes conceitos o mais interessante do ponto de vista de interfaces é o de entradas lógicas que foi estabelecido atendendo à definição de tarefas de interação básicas.

Essas tarefas são:

. Identificar uma posição;

. Identificar uma sequência de posições;

. Quantificar um valor;

. Escolher um valor entre um conjunto;

. Apontar um objeto;

. Fornecer uma cadeia de caracteres.

Verifica-se, no entanto, que entre este nível de abstração e o tipo de conceitos envolvidos numa interface gráfica de alta qualidade existe uma grande diferença. Assim, um caminho será criar, sobre as primitivas gráficas padrão, ferramentas implementando novos níveis de abstrações mais próximos das abstrações usadas ao nivel das aplicações.

Estas trabalham geralmente com abstrações do tipo: Menus; comandos; formulários de entrada; objetos com uma representação gráfica definida; valores numéricos; etc...

Torna-se necessário oferecer ao projetista da interface ferramentas permitindo criar e editar a representação externa dos objetos que o usuário irá ver durante a interação.

Para além destes aspectos estáticos, será necessário também apoiar os aspectos dinâmicos da interação como o eco ou o "feedback" relativos a uma ação do usuário.

Dada a diversidade de formas alternativas para a exe-

cução destes aspectos durante toda a comunicação com o usuário, é conveniente considerar a definição de estilos para a apresentação e dinâmica da interface.

Um estilo fixa e uniformiza tais aspectos, libertando o analista da definição de caracteristicas repetitivas em situações semelhantes de diálogo e assim, permite aumentar a consistência do mesmo.

Um estilo será implementado criando mecanismos que permitam que tarefas de interação mais complexas como "preenchimento de um formulário" ou "deslocar um objeto gráfico" sejam realizadas apresentando aspectos externos bem definidos e consistentes com uma decisão geral sobre formato e dinâmica.

No caso de existirem ferramentas suportando e utilizando a definição de estilos, as ferramentas que implementam as tarefas de interação complexas serão instanciadas com valores "default" definidos por este estilo.

O ambiente de desenvolvimento de interface deve ser então ser constituido recorrendo a primitivas que implementem abstrações intermediárias isolando o analista de detalhes de baixo nivel relativos aos dispositivos de entrada e saida e criando um nivel apropriado à especificação das interfaces.

Na próxima seção apresentaremos as principais caracteristicas de ambientes de desenvolvimento de interfaces.

II.2.3 - Ambientes de Desenvolvimento

De acordo com o que foi dito o desenvolvimento de interfaces deverá ser apoiado em ferramentas para prototipação

rápida e tratando dos detalhes de interação num nível de abstração próximo ao da aplicação, escondendo detalhes de mais baixo nível relativos aos dispositivos de entrada e saída.

Existem fundamentalmente 3 linhas de ação para tornar disponíveis, de forma integrada, tais ferramentas.

A primeira possibilidade corresponde ao desenvolvimento de novas linguagens de programação adequadas ao desenvolvimento de interfaces. Alguns esforços tem sido feitos nesse sentido, por exemplo, em [4] é descrita uma nova linguagem IFS desenvolvida de acordo com essas idéias.

Esta abordagem é limitada pois os comandos da linguagem implementando tais primitivas dificilmente possuem o nível de abstração necessário.

Dentro das linguagens de programação já existentes, a possibilidade de criar extensões é geralmente dificultada pela necessidade de normalização dessas linguagens e, assim, o que geralmente acontece é o desenvolvimento de bibliotecas de procedimentos que implementam as funções mais correntemente utilizadas nas interfaces.

A criação de tais bibliotecas corresponde então à segunda linha de ação que oferece rotinas já prontas com as quais será montada a interface.

Note-se que estas rotinas poderão satisfazer uma série de requisitos das interfaces como: consistência e suporte a diferentes estilos. Obrigam, no entanto, ao conhecimento da linguagem utilizada para poder montar o protótipo.

A terceira abordagem consiste na criação de um ambiente exclusivamente dedicado ao desenvolvimento de interfaces.

Vários trabalhos nesta linha indicam ser viável a geração .de uma interface a partir da especificação do diálogo usuário-computador realizado com um formalismo como Redes de Petri, Redes de Transição Aumentadas ou Gramáticas Formais. Na posse desta especificação a interface do sistema será gerada recorrendo a sucessivas traduções do código assistidas pelo computador.

Assim, a criação de tal ambiente poderá ser a resposta mais eficiente para o desenvolvimento de interfaces pois permitirá a geração, automática ou semi-automática, do respectivo código facilitando quaisquer transformações que se verifiquem necessárias.

O processo de desenvolvimento de uma interface, neste tipo de ambiente, seria então constituido pela especificação de um diálogo usuário-computador, seguida de sucessivas transformações até a geração de um protótipo. Este complementaria a fase de definição de requisitos e especificação da interface através de ajustes iterativos resultantes das sugestões do usuário e das experiências efetuadas no protótipo até que uma boa solução seja gerada.

Repare-se que até agora não foram mencionados os módulos de aplicação cujo desenvolvimento será feito em paralelo. Estes módulos serão executados a partir do diálogo e sob controle do mesmo.

O ambiente proposto configura-se então com um conjunto de ferramentas capazes de apoiar a especificação interativa dos diálogos, executá-los recorrendo a ferramentas de mais baixo

nível como gerenciadores de telas, bibliotecas de técnicas de interação, etc... e controlar a chamada das funções da aplicação. Este tipo de ambiente é geralmente designado por Sistemas de Gerencias de Interfaces de Usuário, SGIU.

Um SGIU opera objetos diretamente relacionados ao diálogo e assemelha-se a um SGBD (Sistema de Gerência de Banco de Dados) na medida em que generaliza e separa uma dada classe de função dos programas de aplicação. No entanto, ao contrário destes últimos, em vez de ser acionados pelos programas de aplicação é responsável pelo acionamento de seus módulos reduzindo o que era anteriormente chamado de aplicação a um conjunto passivo de funções e estruturas de dados.

Esta forma de abordar o desenvolvimento de interfaces apresenta-se como as mais promissora das apresentadas oferecendo as seguintes vantagens:

- Consistência das interfaces.
- Independência entre interface e módulo de aplicação
- Prototipação rápida para apoiar a especificação
- Geração de código de interface de forma automática ou semi-automática.
- Independência da linguagem dos módulos de aplicação.

III- ANALISE DAS FERRAMENTAS DESCRITAS NA LITERATURA.

Na literatura tem aparecido descritas várias ferramentas enquadrando-se nas linhas enunciadas, assim, em [1] [2] [3] [4] [5] [6] [10] são apresentados sistemas que poderão ser considerados SGIU, em [4] è descrita uma nova linguagem criada atendendo às necessidades de desenvolvimento de sistemas interativos e em [7] è mencionado um conjunto de ferramentas para apoio ao desenvolvimento de uma classe de aplicações enquadráveis sob a designação genérica de editores de objetos.

A criação de uma aplicação no sistema referido em [7] corresponde então à definição de objetos de 3 tipos possíveis:

- "Comandos" que representam as ações que o usuário pode tomar.
- "Representações" que são as formas gráficas dos objetos manipulados pela aplicação.
- "EZWIN" que representa a interface do usuário e è constituido por uma janela, um processo, uma repetição para interação e os estados internos da aplicação como um todo.

A definição de uma aplicação corresponde à definição de um objeto do tipo "EZWIN" possuindo listas com objetos dos tipos "representações" e "comandos".

A descrição dos objetos tipo "representações" define o domínio do problema e a definição dos objetos "comandos" estabelece o que o sistema pode fazer.

O sistema está implementado em ZETALISP usando um pacote orientado para objetos e possue previamente definido um conjunto de objetos do tipo "representações" como por exemplo círculo, retângulo, etc... que podem ser utilizados para formar

outros objetos, e um conjunto de "comandos" como COPY, DELETE, MOVE, etc..., que atuam sobre quaisquer objetos do tipo "representações".

Como este sistema se dirige para uma classe de aplicações restritas o seu intersse é menor que a abordagem tipo SGIU pois o seu uso fica limitado não só pelo tipo de aplicação a que se dirige como pela linguagem de programação que terá de ser usada na definição dessas aplicações.

A proposta contida em [4], na linha de novas linguagens, é bastante interessante pois a linguagem IFS, baseada numa rede de frames onde cada frame representa um módulo ou atividade lógica do sistema, oferece:

(1)- Construtores para interação usuário-computador tais como Menus e Formulários.

(2)- Construtores para interação entre programas como invocação de sub processos e comunicação entre coprocessos.

(3)- Construtores para interação e troca de informação entre Frames (conceito básico da linguagem).

(4)- Construtores usuais como quebra condicional, repetição, aritmética, operações com cadeias de caracteres e expressões para controle das interações (1),(2) e (3).

Esta linguagem suporta uma larga faixa de metodologias de desenvolvimento podendo um sistema ser desenvolvido quer com uma abordagem "top down" definindo primeiro as frames, suas interações e as interações destas como usuário e posteriormente as funções de baixo nível, ou com uma abordagem "botton up"

juntando programas já existentes e estabelecendo uma interface uniforme de alto nível baseada nas frames.

Nesta linguagem a programação de interação e a programação analítica são separadas já que a função principal das frames é coordenar o fluxo de informações entre módulos e o usuário. As computações específicas do programa podem ser executadas por outros programas. A organização das telas pode ser realizada de modo "default" ou adaptada ao usuário usando um editor de telas. No entanto não é referida a possibilidade de definição de icones nem de uso de diferentes dispositivos de entrada/saida como o tipo "mouse". Os exemplos apresentados são sempre de telas alfanuméricas.

Esta linguagem cria um ambiente onde os módulos de aplicação podem ser escritos numa outra linguagem e oferece construtores poderosos para a definição de Menus e Formulários. No entanto a limitação causada por não existência de representações gráficas nem suporte a diferentes dispositivos de E/S, associada à necessidade de conhecer esta linguagem e o seu ambiente, diminuem o interesse desta abordagem.

Em seguida apresentamos a análise das propostas de SGIU onde optamos por apresentar para cada uma delas os seguintes pontos:

. Nome do sistema

. Objetivos propostos

. Arquitetura

. Principais aspectos e restrições

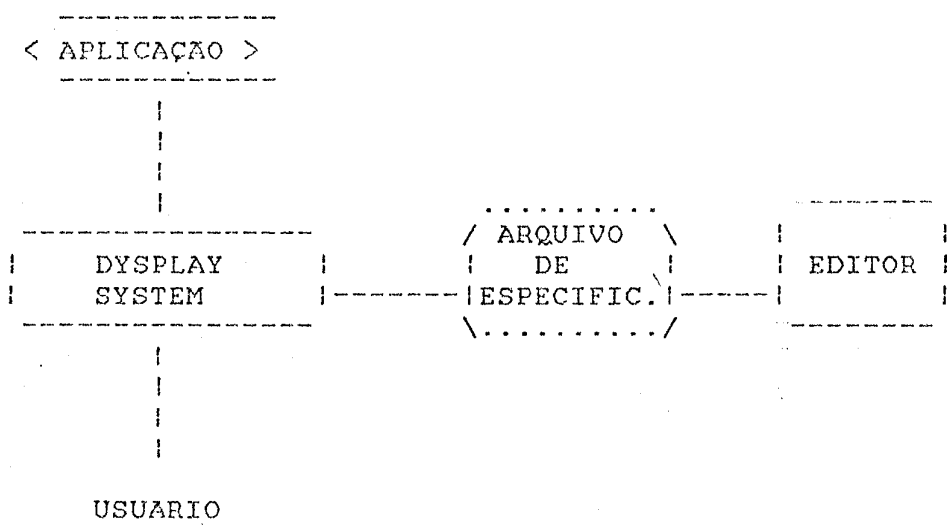
. Estado atual do Projeto.

NOME DO SISTEMA: Karlsruhe-Screen-Based Application Suport System. [6]

OBJETIVOS PROPOSTOS:

Proporcionar a definição fácil de aspectos geométricos, restrições de entrada de dados, facilidades computacionais e lógica de apresentação de forma independente da aplicação e acessível ao usuário final.

ARQUITETURA:



O sistema apresenta dois componentes principais:

. O Editor que permite ao usuário especificar de forma simples os detalhes geométricos das telas, os atributos dos valores apresentados e suas inter-relações e gera um arquivo em disco com estas informações.

. O Sistema de Display usa o arquivo gerado pelo Editor

e permite a especificação de comandos e a passagem de valores de e para a aplicação.

PRINCIPAIS ASPECTOS E RESTRIÇOES:

Este sistema permite de forma independente da linguagem utilizada nos módulos da aplicação gerenciar as telas de comunicação com o usuário e à sua sequenciação.

Uma vez especificados os valores e comandos correspondentes à aplicação é possivel construir uma interface e alterá-la sem modificar a aplicação.

Neste sistema construir uma interface corresponde a especificar recorrendo ao editor as características das telas e valores nelas exibidos. Esta especificação é feita de forma simples e poderá ser realizada pelo usuário final.

Na atual versão o sistema prevê apenas o uso de terminais alfanuméricos e não é feita a referência a modernos dispositivos de E/S como "Mouse" etc... .

No artigo não è descrita a forma de passar valores entre os módulos da aplicação e o Sistema de Display nem a sua ligação, referindo apenas que alterações a uma interface desde que não envolvam a semântica da aplicação, poderão ser realizadas sem necessidade de compilação ou linkedição.

ESTADO ATUAL DO SISTEMA:

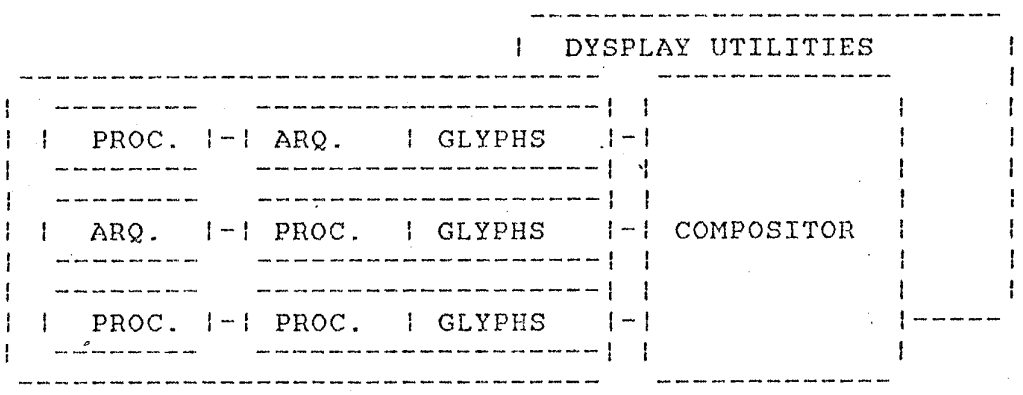
Foi implementado sob o sistema operacional VAX/VMS e tem sido utilizado com aplicações desenvolvidas em FORTRAN e PASCAL.

NOME DO SISTEMA: Descartes [5]

OBJETIVOS PROPOSTOS:

Proporcionar suporte à expecificação em execução dos sistemas e ferramentas para descrição de interfaces e construção de protótipos.

ARQUITETURA:



Um sistema interativo usando uma interface criada neste ambiente será composto por módulos da aplicação, o "compositor" responsável pela interação e desenvolvido especificamente para cada aplicação e utilitários genéricos partilhados pelas diferentes interfaces.

O "compositor" é responsável pela formatação das telas

de acordo com a especificação de interface, e pela sua atualização. Parece ser possível a geração automática a partir da especificação de parte substancial do seu código.

Para garantir a representaçao das informações necessárias foi criado um tipo abstrato de dados designado por Glyph que servirá como uma representação intermédia dos icones da aplicação.

PRINCIPAIS ASPECTOS E RESTRIÇOES:

O desenvolvimento deste sistema foi orientado por conceitos geralmente usados no desenvolvimento de linguagens de programação e procura obedecer a 3 princípios básicos:

- Alta ligação entre a representação na interface e a subjacente informação nos módulos de aplicação.
- Desacoplamento da aplicação e respectiva interface.
- Separação entre orientação genérica de estilos e instâncias.

Na atual versão existe uma linguagem para a especificação da interface e está criado o tipo Gliph e suas operações.

O " compositor " de uma aplicação é criado a partir especificação e prevê-se no futuro a sua geração de forma quase automática.

Os Gliphs têm uma estrutura capaz de representar as relações entre os vários elementos do display, os seus formatos e atributos dinâmicos, através de uma árvore onde cada nó terá os atributos definidos ou por herança dos seus antecessores, ou por combinação de atributos herdados com seus próprios, ou possuindo apenas atributos próprios.

Um estilo resultará assim, da definição de certos atributos herdados por todos os nós.

ESTADO ATUAL DO SISTEMA:

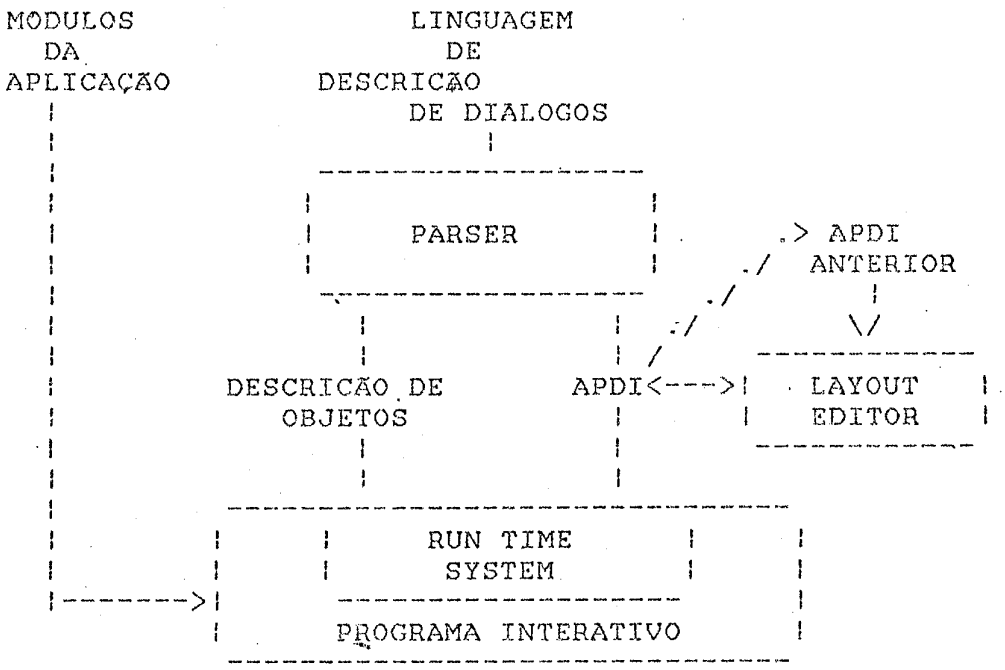
Foi implementada uma segunda versão de um protótipo em PERQ PASCAL com várias simplificações das idéias base. Neste protótipo não existem ferramentas sofisticadas para apoiar de forma interativa a criação da especificação. Os formatos de saida e a dinâmica de entrada são também primitivos.

NOME DO SISTEMA: GRINS [2]

OBJETIVOS PROPOSTOS:

Desenvolver um SGIU que além de oferecer facilidades para o desenvolvimento de sistemas interativos de alta qualidade permita estudar a ligação entre o tratamento da linguagem de entrada e o "feedback" gráfico associado.

ARQUITETURA:



Neste sistema o "Parser" é o responsável pela interpretação de uma linguagem para projeto de diálogos, gerando 2 arquivos. O primeiro contém a informação descrevendo objetos da aplicação, isto é, contém a parte computacional de definição de objetos com significado para a aplicação. O segundo contém a definição do diálogo sob a forma de Autômato Push-down

Interativo. Este último arquivo pode ser trabalhado pelo "Layout-Editor" outro dos componentes do Sistema, e assim é feita a definição interativa do aspectos gráficos das telas, menus, etc... . Finalmente o último componente é o "Runtime System" que interpreta esses dois arquivos e executa os módulos da aplicação.

PRINCIPAIS ASPECTOS E RESTRIÇOES:

O sistema proporciona desacoplamento entre a aplicação e a interface garantindo a ligação entre a informação da aplicação e a sua representação. O "Layout-Editor" permite não só a definição de atributos como cor, posição, forma, como também a criação e edição de Icones.

Na implementação dos componentes do Runtime System foi verificada a inadequação dos pacotes gráficos standart GKS e Core tendo no entanto sido utilizados os conceitos base do GKS.

A linguagem para definição de diálogos não é mencionada no artigo assim como não é abordada a ligação do Runtime System com os módulos de aplicação.

ESTADO ATUAL DO PROJETO:

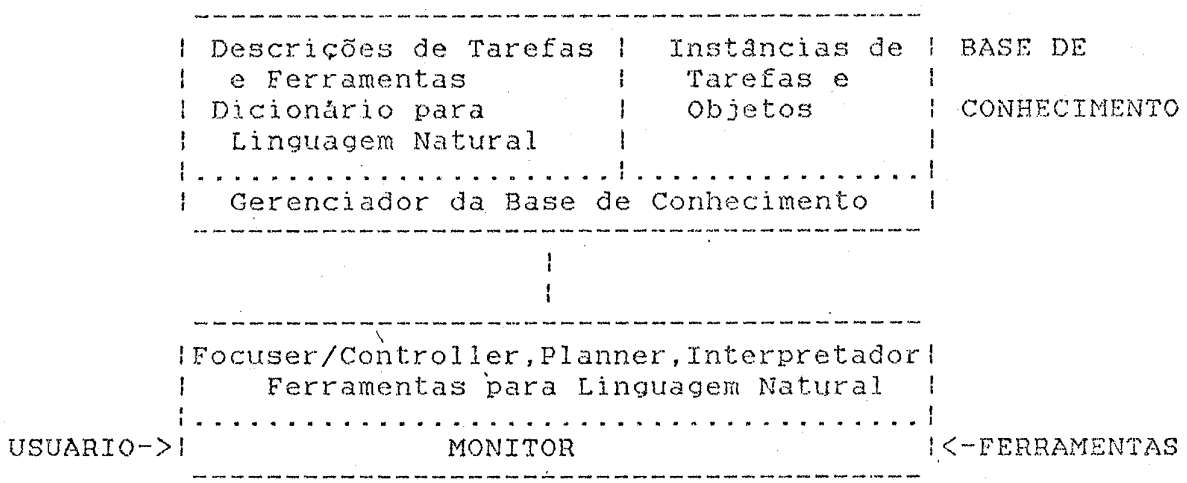
Os componentes do sistema estão implementados em Pascal e C sob o sistema operacional UNIX e usando tecnologia RASTER. Os autores consideram importante o progresso obtido com o sistema mas sentem necessidade de melhorar os algoritmos de atualização de telas usados no Runtime System.

NOME DO SISTEMA: POISE [3]

OBJETIVOS PROPOSTOS:

Oferecer facilidades para definir e suportar tarefas de alto nível predizendo as ações do usuário e prmitindo interpretações múltiplas e competitivas dessas ações.

ARQUITETURA:



A base de Conhecimento é composta por 2 partes:

A primeira corresponde a uma descição estática de tarefas, objetos e ferramentas num ambiente particular. Contêm também dicionários e outras informações para processamento de linguagem natural.

A segunda contêm instâncias descrevendo o estado dinâmico do sistema.

O gerenciador da base de conhecimento controla o acesso e fornece as operações necessárias para manipular o conhecimento.

O monitor serve de interface entre o usuário, as ferramentas e o sistema POISE.

O interpretador é o responsável por interpretar as ações dos usuários no contexto das descrições de tarefas.

O Focuser oferece mecanismos para backtraking quando ocorre um erro. O "Planner" usa os objetivos estabelecidos para as tarefas e orienta o usuário através da sequênciação de ações capazes de atingir esse objetivo.

PRINCIPAIS ASPECTOS E RESTRIÇOES:

Este sistema dirige-se para aplicações caracterizadas pela existência de conjuntos de ferramentas que suportam tarefas a executar pelo usuário num dado ambiente. Assim, o sistema assenta em 3 conceitos base: objeto, tarefa e ferramenta e considera que a execução de uma tarefa consiste em manipular objetos usando ferramentas e tomar decisões sobre aspectos não suportados por ferramentas.

Geralmente a descrição de uma tarefa representa a forma mais corrente de a executar mas várias exceções são possiveis pelo que a interface proposta procura suportar a execução das tarefas quer interpretando as ações dos usuários (modo interpretativo), quer a partir de um objetivo fixado pelo usuário determinar a sequência de ações que levam a tal objetivo (modo planejado)

O artigo preocupa-se fundamentalmente com a descrição da base de conhecimento necessária ao armazenamento das descrições das tarefas, dos objetos, das ferramentas e de instâncias dos objetos e tarefas criadas através do uso das ferramentas.

Não refere os tipos de interação a que é possível recorrer nem a forma de ligação entre as ferramentas e a interface, mas da arquitetura pode-se inferir que será uma interface com linguagem natural.

Na realidade este sistema corresponde a uma nova forma de implementar sistemas de informação e, ao contrário das outras, não pode ser visto como um simples gerador de interfaces para módulos de aplicações cujas funções são bem definidas.

ESTADO ATUAL DO PROJETO:

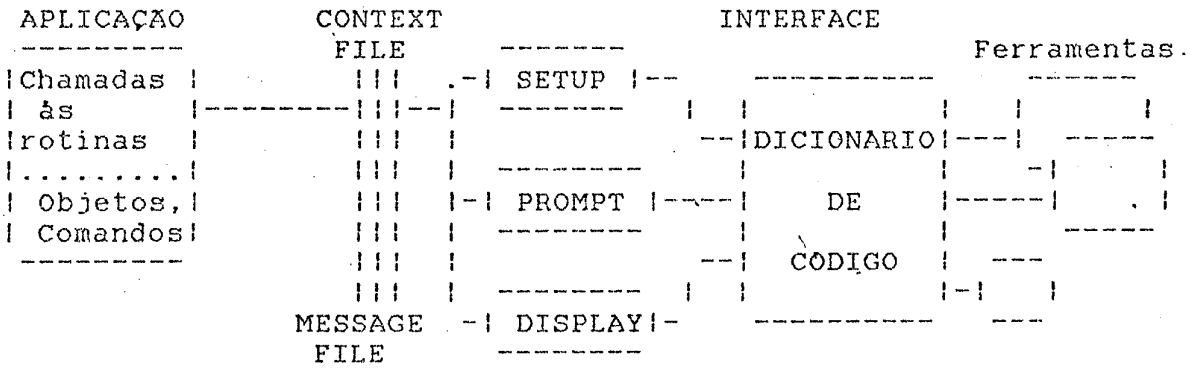
O sistema já está implementado e apenas para a base de conhecimento é indicada a utilização de uma linguagem designada por SRL (Schema Representation Language) baseada no conceito de frames.

NOME DO SISTEMA: AIDE [10]

OBJETIVOS PROPOSTOS:

Satisfazer as necessidades dos usuários e dos analistas, proporcionando um ambiente onde se podem criar e escolher diferentes interfaces para uma aplicação, ou usar a mesma interface em várias aplicações.

ARQUITETURA:



Este sistema compõe-se essencialmente de 3 rotinas.

A rotina "Set up" cria as áreas de interação e as ligações com os dados correspondentes utilizando as informações do "context file". Este arquivo é dependente da aplicação e estabelece a correspondência entre as ações do usuário e o seu significado para a aplicação.

A rotina "Prompt" interage com o usuário para definição

das entradas e traduz as ações deste para a aplicação usando as relações estabelecidas pela rotina "set-up". As mensagens utilizadas na comunicação com o usuário estão definidas no "message file".

A rotina "Display" apresenta ao usuário os resultados da aplicação.

Estas rotinas usam primitivas de alto nível cuja a representação em código será obtida através do dicionário de código, que as associa ao conjunto de ferramentas de mais baixo nível.

Neste conjunto de ferramentas existirão ,por exemplo, rotinas de E/S, isto é, tradutores de diferentes fontes de informação para informação codificada, rotinas gráficas como, por exemplo, primitivas geométricas, permitindo a composição e manipulação de objetos, gerenciadores de diálogo, etc... .

PRINCIPAIS ASPECTOS E RESTRIÇÕES:

Este sistema suporta a definição de estilos através da definição de tres componentes básicos: áreas de interação, entrada e saida. Cada aplicação terá para cada estilo um "context file" e das rotinas "Set-up" "Prompt" e "Display". Estas são geradas por um programa capaz de, a partir de um estilo, configurá-las de forma apropriada.

As principais vantagens deste sistema são o nível de desacoplamento entre os módulos de aplicação e a interface já que nos primeiros não existirá código com funções de interação e apenas se limitará a chamadas às rotinas "Prompt" ou "Display" quando necessitar alguma ação do usuário.

Como as rotinas de interação são desdobradas através do dicionário de código, será possível incorporar ferramentas gerenciando novos dispositivos de E/S como reconhecedor de voz ou de escrita. O seu uso seria possível através da tradução usando estas novas ferramentas.

O sistema, ao garantir um elevado grau de desacoplamento entre interfaces e aplicação, proporciona a possibilidade de usar a mesma interface para diferentes aplicações bastando modificar os arquivos "mensage file" e "context file".

ESTADO ATUAL DO PROJETO:

O ambiente descrito é apenas um modelo não estando ainda implementado e faltando especificar as gramáticas a utilizar na definição de estilos e na criação dos arquivos. Existindo, no entanto, uma versão simplificada dessas gramáticas usada como exemplo no artigo.

Analisando os cinco trabalhos na linha SGIU, verifica-se que a proposta [10] oferece um modelo que, além de satisfazer as necessidades de separação entre interface e aplicação, implementa o conceito de estilo uma vez que é possível usar para diferentes aplicações interfaces semelhantes e garante a forte ligação entre as representações usadas e os objetos representados através do arquivo "context file".

Prevê ainda a futura criação de novas formas de interação apoiadas, por exemplo, em reconhecedores de voz ou de escrita. E, portanto, a proposta mais abrangente.

No entanto, como o projeto não foi ainda implementado é possível esperar que restrições devidas à implementação venham diminuir a sua flexibilidade e abrangência.

O sistema "POISE", descrito em [3], tem características totalmente diversas dos restantes pois, a sua orientação não é tanto a necessidade de oferecer uma forma de construir interfaces amigáveis controlando a execução dos módulos de aplicação, mas sim dar suporte à execução de tarefas compostas por sequências de ações para as quais poderão existir ou não ferramentas de suporte.

Comparando estas duas abordagens, poderemos notar que, na primeira (SGIU tradicional), se procura estruturar, com a definição da interface, as diferentes sequências de tarefas que o usuário poderá realizar recorrendo a ferramentas que são os módulos da aplicação e cuja invocação será feita de forma determinada. Na abordagem do sistema "POISE" a idéia é partir do conhecimento das ferramentas existentes, dos objetos manipulados na aplicação e das tarefas que se pretende realizar para permitir

que o usuário controle livremente a execução das tarefas.

E possível perceber também que enquanto na abordagem tipo SGIU a geração de um sistema interativo é composta pela criação dos módulos da aplicação e de uma dada interface gerada pelo SGIU, na abordagem do sistema "POISE" a interface será sempre a mesma (oferecida pelo monitor) e o que difere de uma aplicação para outra são as descrições contidas na base de conhecimento e as ferramentas a invocar.

Resumidamente a abordagem SGIU apresentada anteriormente é essencialmente procedural enquanto a proposta "POISE" é lógico-dedutiva.

O uso de uma ou outra abordagem irá certamente depender do tipo de aplicação a desenvolver e do avanço na tecnologia relativa ao armazenamento e gerência do conhecimento necessário à construção da base de conhecimento desses sistemas.

IV- CONCLUSOES

Na introdução deste trabalho foram formuladas 4 questões sobre desenvolvimento de intrefaces que serviram de orientação ao presente estudo.

Como conclusão do trabalho apresenta-se, então, uma resposta para cada questão tentando sintetizar as idéias apresentadas anteriormente.

1 - A interface deve ser projetada junto com o software?

O projeto de uma interface deve ser realizado em paralelo ao projeto do software responsável pela funciónalidade da aplicação.

Deverão ser estudados inicialmente os pontos de interaçao com o usuário e definidas de forma modular as tarefas a executar pelo sistema interativo.

Uma vez estabelecidas as necessidades de interação, o projeto da interface pode ser separado do projeto do restante software.

O projeto da interface dependerá fundamentalmente da aplicação pretendida, dos tipos do usuário envolvidos e das ferramentas disponíveis para a construção da interface.

As vantagens principais da separação proposta são:

a) Simplificar a tarefa do projetista dos módulos. da aplicação, libertando de detalhes de entrada e saida, cujo o tratamento com as linguagens de programação tradicionais é geralmente penoso.

b) Possibilitar que a interface seja projetada por profissionais treinados na área de fatores humanos melhorando a qualidade da interface e o grau de satisfação dos usuários.

2 - E possivel desacoplar consideravelmente (até que ponto?) o software da sua interface?

Sim. Qualquer um dos trabalhos analisados no ponto anterior é exemplo de desacoplamento (em maior ou menor grau) entre interface e módulos de aplicaçao.

O artigo [10] leva este desacoplamento ao ponto de considerar que os módulos da aplicação são objetos capazes de enviar e receber mensagens de e para a interface sendo totalmente independentes desta em termos do código gerado e das representações apresentadas aos usuários.

A única ligação que necessàriamente terá de existir é a ligação entre os valores internos dos módulos de aplicação e as respectivas representações que devem refletir sempre o estado mais recente destes valores.

3 - O que significa e como se consegue validar uma interface?

Validar uma interface corresponde a verificar a sua correção do ponto de vista da especificação. Como esta especificação é geralmente apoiada por prototipação, validar significa na prática verificar a satisfação do usuário e a sua capacidade para usar o sistema em diferentes situações.

Como foi referido devem ser realizados testes pelos usuários e, destes testes, deverão resultar informações recolhidas, por exemplo, através de questionários que permitam verificar a adequação da interface implementada.

Medidas como tempo de aprendizagem, número de erros cometidos, rapidez de execusão de tarefas, servirão também para validar a interface garantindo que ela satisfaz nesses aspectos as expectativas dos usuários.

Como não existe ainda uma especificação formal para a interface o problema da validação é um problema em aberto tanto mais que envolve a avaliação de aspectos ligados a fatores humanos geralmente subjetivos e, por isso mesmo, dificilmente quantificáveis.

4 - Como ferramentas para o projeto de interfaces tratam o problema acima?

Dos artigos utilizados no presente trabalho apenas [1] e [8] abordam diretamente o problema de validação de software. No entanto se considerarmos que de alguma forma as ferramentas propostas facilitam o desenvolvimento de software (usando prototipação) a validação de interface será, como já foi dito, realizada de forma experimental procurando medir a satisfação dos usuários relativamente à interface criada no protótipo.

Como as diferentes ferramentas proporcionam a separação entre código da interface e código dos módulos de aplicação, a validação do sistema como um todo corresponderá à validação das partes, desde que ambas respeitem as restrições e o protocolo estabelecido para a comunicação entre interface e aplicação.

Por tudo que foi dito pode então concluir-se que o desenvolvimento de interfaces é um problema em aberto não existindo bem definidos princípios para projeto, linguagens de especificação e procedimentos para validação.

O aspecto que mais dificulta a definição destes pontos é o desconhecimento relativo ao usuário que condiciona a definição, sempre qualitativa, dos fatores humanos envolvidos.

Com a evolução da tecnologia de conhecimento, da qual o sistema [3] é exemplo no que diz respeito ao conhecimento sobre as tarefas a executar, é possível esperar a inclusão futura de conhecimento sobre os usuários, permitindo a criação de interfaces adaptáveis em tempo de execução.

Outra dificuldade sentida na implementação de interfaces corresponde á necessidade de possuir um sistema operacional e uma linguagem de programação permitindo a chamada de módulos desenvolvidos em diferentes linguagens, coprocessamento e alta portabilidade.

O conceito de objeto parece bastante útil para a definição de interfaces pois permite suportar a ligação adequada entre representações na interface e correspondentes informações na aplicação.

R E F E R E N C I A S

[1] - Software Engineering for User Interfaces - Stephen W. Draper and Donald A. Norman. IEEE Transactions on Software Engineering, Vol.SE-II, N 3, March 1985.

[2] - Input/Output Linkage in a User Interface Management System - Dan R. Olsen Jr, Elizabeth P. Dempsey, Roy Rogge. ACM Vol 19, N3 1975.

[3] - A Knowledg-Based Approach to Data Management For Intelligent User Interfaces - Carol A. Broverman, W. Bruce Croft. Proceedings of VLBD 85, Stockholm.

[4] - IFS - A Tool to Build Integrated, Interactive Application Software - K. P. Vo. AT&T Technical Journal Vol.64 N9, Novembro 1985.

[5] - Descartes: A Programming-Language Approach to Interative Display Interfaces. Mary Shaw, ed al. ACM SIGPLAN 1983.

[6] - A Generalized User Interface for Applications Programs - Leonard J. Bass and Ralph Bunker. Communications of ACM Vol.24 N12, December 1981.

[7] - There"s More to Menu Systems Than Meets the Screen - Henry Liederman. ACM SIGGRAPH"85 Vol.19 N3, 1985.

[8] - Programming in the Large - C.V. Ramamoorthy, ed al. IEEE Transations on Software Engineering, Vol.SE-12 N7, July 1986.

[9] - Abstrations for User Interfaces Design - Joelle Coutaz. Computer, September 1985.

[10] - Application Interface Development Environment - Robert F. Gordon, Barry E. Willner. Research Report, IBM Research Division, 1985.

[11] - The Future of Interactive Systems and the emergence of direct manipulation - Behavior Inform. Technol. Vol.1 pp. 237-256, 1982.

[12] - A Study of Display Generation and Management in Interactive Business Applications - RJ2392, IBM Research Division, 1977.