Mecanismos para Especificação e Prototipagem de Interfaces Utilizador – Sistema

F. Mário Martins
J. João Almeida
Pedro R. Henriques
UMINHO/INESC
R. D. Pedro V. 88-2

Junho de 1990

Resumo

Uma das principais lacunas na concepção e desenvolvimento de sistemas interactivos diz respeito à incapacidade existente em reflectir na camada interactiva questões relacionadas com a dependencia desta relativamente à camada computacional. Em resultado, quer a concepção quer a prototipagem de Interfaces com o Utilizador (IU), contemplam apenas aspectos estáticos destas, designadamente, aspectos visuais e ergonómicos, deixando por tratar aspectos comportamentais. Estes prendem-se fundamentalmente com o estado global do sistema e, em particular, com dependencias contextuais.

Neste artigo, apresentam-se mecanismos que permitem, a especificação de controladores de diálogo que tomam em consideração dependências de contexto (GUIÕES de INTERACÇÃO), bem como mecanismos que possibilitam a prototipagem de IU dependentes do contexto (GRAMÁTICAS INTERACTIVAS GUARDADAS).

Tais mecanismos não aparecem isoladamente, antes se constituindo como diferentes passos de uma metodologia visando a geração automática de IU

1 Introdução

1.1 Objectivos

O problema geral abordado neste artigo posiciona-se em torno da reconhecida lacuna de conhecimento existente quanto ao desenho de diálogos utilizador -sistema em particular, e que se estende ao desenho da camada dos sistemas interactivos designada por *Interface com o Utilizador* .

Em consequência, do ponto de vista dos utilizadores tal traduz-se em dificuldade e desagrado na utilização dos produtos; do ponto de vista dos desenhadores, traduz-se por intenso e desordenado esforço, mesmo que possam e saibam usufruir de ambientes intermediários tais como UIMS ou "Application Frameworks"; finalmente, do ponto de vista dos gestores dos projectos, tal traduz-se por custos que atingem mais de metade dos custos totais.

Reconhecida esta situação, os vectores de complexidade que têm sido identificados são os seguintes:

a) A área é **multidisciplinar**, envolvendo engenharia de factores humanos, engenharia de "software" e engenharia do conhecimento;

b) Os desenhadores de IU são obrigados a especificar os diálogos com o utilizador a muito baixo nível, ou seja, são necessários **mecanismos de abstracção**;

c) Quase inexistência de ferramentas que auxiliem ao desenho e possibilitem a **construção de protótipos** e a sua experimentação;

d) Os desenhadores de IU não têm ao seu dispor metodologias, contrariamente ao que já se passa noutras áreas;

As técnicas e formalismos apresentados neste artigo, designadamente, **Guiões de Interacção** (GI) para a especificação de diálogos, e **Gramáticas Interactivas Guardadas** (GIGs) para representação intermédia possibilitando a prototipagem rápida, visam sobretudo dar uma resposta às b) e c).

1.2 Revisão do Estado dos Conhecimentos

A proliferação de computadores pessoais e "workstations" no mercado dos computadores veio suscitar a necessidade de estudo e desenvolvimento de tecnologia de interacção apropriada ao novo tipo de utilizadores, em particular a melhoria da Interface com o Utilizador (IU) [Or68,GS86]. Mecanismos de interacção consistentes são hoje em dia fundamentais como garantia do sucesso de sistemas, utilitários ou aplicacões [GU*80].

A área designada por HCI (de "Human-Computer Interaction"), em português designada IHC, por analogia semântica (cf. Interacção Humano-Computador), como ciência é ainda imatura e, consequentemente, o desenho de IU é ainda feito com base em tecnologia existente mais do que com base em teorizações [Th87].

De uma forma geral, existe concordância quanto ao facto de que IHC é intrinsecamente uma área complexa, dado que à complexidade conhecida dos processos de

engenharia de "software" se junta a complexidade inerente à existência de uma componente humana a ser considerada [Sc80].

Como resultado, a definição de princípios de engenharia e de regras para a concepção de IU é praticamente inexistente, pelo que ainda não existem metodologias que promovam a integração do seu desenho no ciclo de desenvolvimento de sistemas interactivos [HH87,Su89].

No entanto, muito trabalho vem sendo desenvolvido, nem sempre seguindo um encadeamento que permita a sedimentação de conhecimentos adquiridos, mas procurando explorar as múltiplas vertentes que podem contribuir para o desenvolvimento na área.

Assim, *modelos de interacção*, descrições abstractas baseadas na decomposição da interacção em diferentes camadas, a maioria dos quais apresentados informalmente, têm sido desenvolvidos, apresentando níveis de detalhe variáveis [Mo81,FvD82,Ni86].

Conforme a camada relativamente à qual possibilitam maior descrição podem ser classificados como *conceptuais* [CMN83,Mo81], *analíticos* [CMN80,Re81], de *desenho* [Sc83,MB86], e de *implementação* [Cou87,GR84].

Princípios de desenho têm igualmente sido propostos de modo informal por diversos autores [Sc80,Th85].

O *princípio de separação* é talvez o primeiro princípio estabelecido e aceite no desenho de IU [CD82,Ka82], apesar de ser, de forma geral, difícil de conseguir. Basicamente, o princípio estabelece a independência no desenho das componentes interactiva e computacional de um sistema interactivo. Tal separação, estendendo-se mesmo à implementação, veio possibilitar a construção dos chamados Sistemas de Gestão de Interfaces com o Utilizador (SGIU), "User Interface Management Systems" (UIMS), especializados em tratarem dos aspectos de comunicação entre os utilizadores e as aplicações [Pf85], fornecendo um conjunto de facilidades de interacção e possibilitando mesmo a prototipagem rápida da IU.

A existência de modelos de interacção é, particularmente neste caso, crucial dado que os SGIU baseiam as suas implementações num determinado modelo [Gr86]. O modelo designado por modelo de Seeheim [Pf85], um dos exemplos de modelo genérico, divide a IU em três componentes: *apresentação*, *controlador do diálogo* e *modelo da aplicação*.

Destas, apenas a componente de controlo do diálogo tem merecido um tratamento mais profundo e formal, através da utilização de técnicas de especificação diversas, desde as derivadas das linguagens formais [Re81,Sc82b], às redes de transição de estados [Wa85] até outras especificamente desenvolvidas com tal propósito [vdBPH83]. Infelizmente, poucas são as que possibilitam a descrição de diálogos concorrentes [vdBPH83, Ma88a], ainda que a níveis diferentes.

No entanto, tem sido igualmente apresentada a opinião de que o *princípio da separação* pode conduzir a resultados deficientes dado que o desenho da IU deverá, sempre que possível, ser realizado tomando em consideração o seu contexto real, isto é, utilizador e aplicação [So87,HH87].

A ligação metodológica entre o desenho da aplicação e da IU, preocupação menos voltada para o utilizador final mas mais para o desenhador de IU, carece de estudo mais aprofundado, tal como vem sendo sugerido por vários autores [Th85,HH87,Su89], para que de futuro se possa abordar a concepção de sistemas interactivos de uma forma integrada.

De momento, e em resultado do estado da arte nesta área, os utilizadores podem já encontrar interfaces tecnológicamente atractivas, resultantes do desenvolvimento de produtos tais como Gestores de Janelas (cf. X-WINDOWS e outros), "Application Frameworks" (cf. MacApp, Open Look e outros), enquanto os desenhadores de IU procuram ainda princípios de desenho e ferramentas que os auxiliem nessa tarefa [Ma90].

Capitalizando em técnicas próprias das áreas da IHC e da Engenharia de "software", torna-se pois necessário encontrar paradigmas unificadores, procedendo eventualmente a uma remodelação do ciclo tradicional de desenvolvimento.

Da engenharia de "software" parece ser relevante importar o rigor conferido pelos métodos formais de especificação [Th86] e a versatibilidade das técnicas de prototipagem rápida enquanto que na área de IHC devem ser considerados modelos de interacção mais genéricos e técnicas de especificação de diálogos mais expressivas [Th87,MO85,MO86].

Em resumo, passada a fase empírica, a IHC entrou na fase de criação de modelos e metodologias (teorizações) indispensáveis à entrada definitiva no período em geral designado por automatização [Ayr68,Ma81]. O recurso aos métodos formais e às técnicas de prototipagem rápida é o caminho a investigar para se obter tal salto qualitativo na área.

1.3 Estrutura do Artigo

Na secção 2 abordam-se as questões relacionadas com a *independência do diálogo*, propriedade importante que serve de suporte ao *princípio da separação*, e caracteriza-se ainda um aspecto em geral esquecido no desenho de IU, a *dependência do contexto*.

Na secção 3 introduz-se o formalismo designado por Guião de Interacção, que facilita a descrição do comportamento da IU a alto nível, possibilitando a expressão de diálogos concorrentes e dependências contextuais.

Na secção 4 apresentam-se as Gramáticas Interactivas Guardadas (GIGs), um outro formalismo, neste caso de base gramatical. As GIGs têm poder expressivo quase equivalente aos GI, sendo capazes de especificar diálogos sequenciais entrando em consideração com as referidas dependências.

As GIG, dada a sua base gramatical, tornam-se adicionalmente importantes pois facilitam a criação de protótipos de IU, dada a facilidade com que podem ser tratadas por ferramentas de tradução existentes.

Na secção 5, através do estudo de um caso, mostra-se a ligação entre os GI e as GIGs.

2 Dependência do Contexto em IU

2.1 Independência do Diálogo

Nas primeiras abordagens ao desenho de sistemas interactivos, preocupações com o isolamento do código computacional do código de I/O eram praticamente inexistentes. Tipicamente, quando muito, mensagens de erro ou similares eram agrupadas em sub-rotinas.

O elevado grau de interdependência entre as duas camadas, tanto maior quanto mais proximo da implementação final se encontrasse o desenvolvimento, inviabilizava a possibilidade de reutilização do código, tornava as alterações praticamente impossíveis de realizar e os tempos e custo de projecto insustentáveis.

Na área das bases de dados, para problemas semelhantes (a possibilidade de alterar os dados sem interferência nos programas), os investigadores desenvolveram o conceito de *independência de dados* (cf. "data independence") [SAFW72]. Uma descrição formal dos dados (à qual se faz corresponder uma estrutura em "run-time") permite tornar independentes os programas das instâncias de dados.

"Independência do diálogo" [EH81] foi o conceito análogo encontrado na área dos sistemas interactivos. A descrição formal da comunicação entre a IU e a camada computacional permite a sua relativa independência. Deste modo, decisões de desenho que afectem apenas a IU são isoladas das que afectem apenas a camada computacional ou aplicação. A inexistência de preocupações do ponto de vista da independência do diálogo faria com que se tornasse da responsabilidade da camada computacional toda a lógica de controlo de um dado sistema que, por exemplo se baseasse numa interacção com o utilizador via linguagem de comandos. É fácil imaginar que, caso o desenho do sistema não seguisse o modelo da independência do diálogo, a alteração desta interface para uma baseada em menus e caixas de diálogo implicaria modificações dramáticas na camada computacional, perfeitamente injustificadas aliás, dado que a funcionalidade da aplicação seria exactamente a mesma. Particularmente sensível seria o código necessário à validação das entradas do utilizador.

Baseando o desenvolvimento do sistema no princípio da independência do diálogo, a modificação anteriormente referida poderia ser confinada a alterações afectando apenas a camada de diálogo. Adicionalmente, poderia até ser possível, tendo por base a mesma camada computacional, oferecer as duas diferentes perspectivas ou "visões" de interacção: *assistida* (menus e caixas de diálogo) e *por comandos*.

O chamado *princípio da separação* [CD82] baseado na noção de independência do diálogo, e porventura um dos primeiros princípios estabelecidos na área, veio tornar possível o desenvolvimento dos primeiros Sistemas de Gestão de Interfaces com o Utilizador (SGIU), ou UIMS (de "User Interface Management Systems"), visando facilitar o desenho da camada interactiva (a componente de desenho) e gerir o diálogo entre utilizador e aplicação em tempo de execução (a componente de "run-time") [Ka82,Pf85].

Para diálogos sequenciais uma estratégia de sincronismo entre as entradas do utilizador, as transformações de estado internas do sistema e suas consequentes respostas, facilita a separação no desenho das duas camadas em questão. Para diálogos concorrentes ou por manipulação directa, a separação pode tornar-se difícil de conseguir, já que, não só há a necessidade de representações de objectos partilhados pelas duas camadas, como também diálogo e cálculo passam a estar ligadas a um nível mais atómico.

De facto, e em geral, se a "aparência" da interface se pode considerar independente da aplicação, o mesmo não se poderá dizer do seu *comportamento*. Este é muitas vezes ditado pela camada computacional, em particular pelo *estado da aplicação*.

Deste modo, assumindo-se do ponto de vista das ciências da computação, que as rotinas da camada computacional são totais, isto é, que lidam com conjuntos válidos de valores de entrada, gerando resultados válidos, compete à camada de interacção garantir essa validade. Mas dependerá tal validade apenas de verificações estáticas (temporalmente imutáveis) tais como tipos dos argumentos, gamas aceitáveis, etc? A resposta é obviamente negativa.

A validade dos argumentos poderá depender também do "estado actual da aplicação" entendendo-se por "actual" o instante de interacção.

Tal pode ser estendido à própria operação já que, função do estado da aplicação (em sentido lato), tal operação pode não ser invocável.

Assim, e em resumo, parece óbvio concluir-se que o comportamento da camada interactiva é, apesar de todos os princípios, muito dependente do estado da aplicação (ou camada computacional) e até das interacções já realizadas. Designaremos tal dependência por **dependência do contexto**.

2.2 Dependências do Contexto em IU

Dentre as principais razões justificativas da ainda má qualidade das IU, a falta de metodologias e técnicas para o seu desenvolvimento assumem um peso importante. Mas outras existem.

De facto, o desenvolvimento de um sistema interactivo passa hoje em dia quase inevitavelmente pela interligação entre um UIMS ou "toolkit", a aplicação, um sistema de Gestão de Janelas e ainda, eventualmente, um pacote gráfico.

Sistemas de Gestão de Janelas (cf. X-Windows, etc) e pacotes gráficos (cf. GKS e PHIGS) são hoje ferramentas tecnologicamente adultas, daí a sua disponibilidade comercial e até standardização. O mesmo não se poderá dizer dos UIMS que apresentam ainda problemas não resolvidos [GR90].

Um dos mais significativos prende-se com a forma como os UIMS e ferramentas similares abordam a possibilidade de se construirem protótipos de IU.

Na sua grande maioria tais sistemas, permitem apenas a prototipagem dos aspectos estáticos da IU, em particular os relacionados com a apresentação da interface, deixando de fora todos os aspectos que se relacionam com o comportamento, tais como, reacções à activação de botões do "rato", selecções em "menus", etc. De uma forma geral, tal significa que tais sistemas apenas possibilitam a prototipagem de interfaces insensíveis ao contexto [Ol86].

Em resultado, os protótipos de IU assim desenvolvidos serão sempre necessariamente pobres em "*feedback semântico*" e portando irreais. O problema consiste em não se ter em consideração no desenho da IU, em particular do seu comportamento, questões relacionadas com *dependências de contexto*, dado em geral não serem trazidas para a especificação da IU quaisquer informações relacionadas com a camada computacional.

Assim, e a título de exemplo, "menus sensitivos" (ou menus sensíveis ao contexto), nos quais certas operações são vedadas à selecção em função do contexto de interacção, são em geral impossíveis de desenvolver ao nível da prototipagem, se determinada informação não estiver presente.

Para que o desenvolvimento de protótipos de IU possa incluir aspectos dinâmicos (porventura os mais importantes), é necessário que informação sobre a camada computacional possa ser tomada em consideração (o chamado "modelo da aplicação"). Esta informação, essencial sobre a aplicação, em conjunto com a informação intrínseca à própria IU, de apresentação e de controlo do diálogo, estabelecem o "contexto" no âmbito do qual um evento utilizador (p.ex. selecção num menu, accionamento de um botão do "rato" ou comando) devem ser interpretados.

Uma interface insensível ao contexto pode ser considerada a menos "adaptada" à aplicação. De facto uma IU deste tipo não tem possibilidade de "observar" mudanças na camada computacional e, portanto, qualquer que seja o estado interno desta, apresentará sempre o mesmo comportamento interactivo, limitando-se o utilizador a observar diferenças nos resultados gerados pela camada computacional.

Como se pode observar também, a sensibilidade ao contexto para além do conduzir ao desenho de interfaces mais adaptadas e por isso mais inteligentes, concorre também para uma melhoria do tratamento de erros.

Numa perspectiva de desenho integrado poder-se-á mesmo trabalhar com uma camada computacional "total", no sentido de que nunca gera erros, deixando à IU todo o trabalho de "prevenção" e tratamento destes (cf. o modelo "dialog dominant control").

Uma área interessante de investigação com vista à geração automática de IU é a definição de uma metodologia que possibilite a desenho integrado das duas camadas. Em [Ma90,MO90] alguns passos no sentido do estabelecimento de uma metodologia de base formal são apresentados, mostrando-se como a partir da especificação da camada computacional muita da informação relevante para o desenho da IU pode ser imediatamente inferida.

Neste artigo apresentam-se técnicas desenvolvidas no âmbito do estudo metodológico referido em [Ma90], em particular uma abstracção de controlo designada GUIÃO DE INTERACÇÃO [MO90] e a sua representação sob a forma de um formalismo gramatical, as GRAMÁTICAS INTERACTIVAS GUARDADAS [Ma88b,AHM90], cuja implementação recorrendo a GRAMÁTICAS DE ATRIBUTOS vem colmatar uma das grandes dificuldades atrás apontadas no desenvolvimento de IU : a prototipagem de IU sensíveis ao contexto.

f) EVSEQ: nesta secção indicam-se os eventos externos ao guião necessários à sua transição de estado até atingir o seu estado final, seja este a invocação do comando, ou a anulação da sequência interactiva descrita pelo mesmo (em caso de anulação por exemplo). A ordem destes eventos pode não ser necessáriamente sequencial. Diálogos sequenciais são expressáveis pelos GI, tal como podem ser expressos diálogos concorrentes (síncronos ou assíncronos). Um exemplo comum de expressão de diálogos assíncronos é a possibilidade de, a qualquer momento de um diálogo interactivo, se poder retornar ao estado inicial pelo accionamento de uma tecla especial, por exemplo, <ESC>:

g) TRANS: secção na qual, através de um conjunto de regras da forma

$$< estado1 >:< evento >.< condicao > \rightarrow < estado2 >$$

se descrevem as transições de estado interno do GI associadas ao diálogo interactivo. Cada regra deverá ser interpretada do seguinte modo: Partindo do <estado1>, a ocorrência de <evento> satisfazendo a <condição> faz com que o estado actual passe a ser <estado2>:

h) STATES: nesta secção associa-se a cada estado do diálogo controlado pelo GI o conjunto de acções a executar logo que o mesmo é atingido.

Vejamos de seguida, via exemplo, como a descrição do comportamento interactivo pode ser feita num GI de forma não procedimental, simplesmente através de um sistema de regras, favorecendo assim um estilo mais decalarativo.

A sequência de eventos necessária à síntese interactiva de um comando que, num sistema de gestão académica, possibilite inscrever o aluno número **na** no curso **c**, poderia descrever-se informalmente e textualmente por:

a) verificar se no contexto actual a operação é invocável:

b) se a) é verdadeira *inicializar* senão *terminar*:

c) a introdução de <ESC> a qualquer momento termina o diálogo:

d) por uma ordem arbitrária, ler o *número do aluno* (na) e o *código do curso* (c), aceitando apenas valores válidos;

e) se a) foi completada invocar a operação **inscreve(na, c)**:

f) terminar

Embora à primeira vista esta descrição informal do diálogo possa parecer normal e de simples satisfação, na realidade alguns problemas interessantes se nos deparam.

Em primeiro lugar, a a) impõe uma pré-condição contextual à elegibilidade da operação. Em segundo lugar, a c) introduz, através da expressão "a qualquer momento", características de assíncronismo e diálogo não sequencial que devem ser devidamente expressas. Finalmente, na alínea d), a expressão "por uma ordem arbitrária"

leva-nos a tomar em atenção o facto de que a entrada dos valores dos argumentos pode ser, tal como no exemplo, completamente arbitrária (ie. livre), o que nos conduz a três possibilidades, designadamente **a** seguido de **b** (denotado por $a \rightarrow b$) ou **b** seguido de **a** (denotado por $b \rightarrow a$), o que em conjunto se pode denotar por $a - b$, e ainda **a** em paralelo com **b** (denotado por a||b).

No entanto, o requisito informal da alínea d) é, como poderia ser mostrado, ilógico e incorrecto.

De facto ambos os argumentos a introduzir devem satisfazer certas condições, designadamente:

- Qualquer que seja o *Número de Aluno* introduzido, nenhum aluno se pode inscrever 2 vezes no mesmo curso ou em mais do que um curso; introduzido o *Número de Aluno* deverá verificar-se que este não foi ainda inscrito a qualquer curso;

- O *Código de Curso* deve ser válido, isto é, referenciado na Base de Dados de Cursos e, para este, existirem ainda vagas.

O *Código de Curso* aparece portanto mais restringido que o *Número de Aluno* a inscrever, pelo que se considera que a ordem deverá ser $c \rightarrow na$.

Desta forma, fixado um **c** podemos realizar diversas iterações para a introdução de um **na** válido, dependendo a validade deste apenas de **c**.

No caso inverso, a validade de **c** depende não só do **na** lido mas também de outros factores.

O GI correspondente aos requesitos desta operação poderia ser escrito como:

```
GI : InscreveAluno
SYMBOL
    com = {inscAluno, inscreve}  default 'inscAluno'
ARGS
    na : NumAluno default ""
    c : CodCurso default ""
STATE - APL
    lc : CodCurso - set
    bdc : CodCurso — NumAluno - set
    ncl : CodCurso — NumClausus
STATE - UI
    h : str - list
CONTEXT
    ∃ c ∈ dom bdc.#bdc[c] < nlc[c]
EVSEQ
    input(c) — input(na)
STATES
    start
    erro1 ⇒ out("Curso Invalido.Reintroduza"); c = NULL
    erro2 ⇒ out("Aluno ja inscrito"); na = NULL
    end ⇒ inscreve(na, c); h — +"inscreve(na, c)"
TRANS
    start : input(c), c ∈ dom bdc ∧ #bdc[c] < ncl[c] — leAluno
    start : input(c), ¬ (c ∈ dom bdc ∧ #bdc[c] < ncl[c]) — erro1
    erro1 : NULL, NULL — start
    leAluno : input(na), na ∉ bdc[c] — end
    leAluno : input(na), na ∈ bdc[c] — erro2
    erro2 : NULL, NULL — leAluno
```

Um GI define portanto um determinado comportamento interactivo, seja à custa de um conjunto de eventos por ele internamente reconhecidos e temporalmente estruturados (cf. cláusula EVSEQ), seja à custa de uma "expressão de comportamento" construída à custa de outros GI, sejam estes LOCAIS ou EXTERNOS (cf. cláusula EXTERNAL adiante).

Expressões envolvendo GI podem ser construídas à custa dos operadores dos quais os mais comuns são os seguintes:

; sequência

|| paralelismo

* repetição

+ alternativa

— ordem arbitrária

Definindo um GI associado ao tratamento do evento correspondente ao accionamento da tecla <ESC>, como

```
GI : Escape
ARGS
    k : key
TRANS
    start : input(k), k = 'ESC' — end
```

podemos agora construir um guião para a operação completa que passe a aceitar "a qualquer momento" o evento <ESC>, em paralelo com o guião atrás definido.

Sendo ambos externos, uma cláusula EXTERNAL deverá ser usada como declaração bem como expressão do tipo de composição (neste caso paralelismo ||).

```
GI : Inscricao
EXTERNAL
    InscreveAluno||Escape
```

4 Gramáticas Interactivas Guardadas

4.1 Introdução às GIGs

Uma **Gramática Interactiva Guardada** (abrev. GIG) proposta numa versão inicial em [Ma88b], é uma adaptação das gramáticas tradutoras -GT- (gramáticas independentes de contexto -GIC- com acções semânticas) com vista à especificação de diálogos utilizador-sistema. Com este formalismo alvejamos a geração automática de Interfaces. Sendo os Guiões de Interacção mecanismos abstractos para a especificação de IU, as GIGs surgem da necessidade de se encontrarem formalismos capazes de expressarem as descrições dos GI facilitando a geração automática das IU, dada a sua base gramatical.

É propósito desta secção introduzir sucintamente as GIGs e referir, de passagem, a geração automática de interfaces a partir dessas gramáticas. Para maior detalhe sobre estes temas, remete-se o leitor para [AHM90].

Qualquer formalismo para descrever interfaces com o utilizador, deve ser independente do modelo de interacção[1] — esse modelo deverá estar todo concentrado no Gerador dos Interfaces (GI), o qual o reflectirá no programa gerado. Nessa perspectiva, procuramos propor um formalismo que se acomode a vários GIs, o que irá permitir gerar, a partir da mesma descrição, interfaces diferentes (correspondendo a filosofias de

[1] Entenda-se, a estratégia a seguir no que respeita à forma de apresentar a informação ao Utilizador e de proceder à recolha das informações que este envia, bem como as técnicas de implementação dessas trocas.

Exemplo 2 *Na especificação dum editor de texto, verifica-se que existem vários comandos distintos que actuam sobre blocos. Admitindo que o conceito de bloco é denotado pelo símbolo NT* DefineBloco, *surgiria então a seguinte produção alternativa para o definir:*

```
<"Definicao de Intervalo">DefineBloco
      : <"linha actual">actual  {$$.ini=$$.fim=linhaact();}
      | <"linha actual ate ao fim">resto
                                {$$.ini=linhaact(); $$.fim=ultima();}
      | <"outro intervalo">outro Lelinha Lelinha
                                {$$.ini=$2; $$.fim=$3;}
```

Ainda extraída da mesma GIG para especificar um editor de texto, veja-se abaixo a produção que define os três modos alternativos de seleccionar o "drive" de trabalho (repare-se na guarda que precede uma das alternativas, indicando que essa escolha só é válida se o utilizador actual tiver privilégios de **superuser**!*):*

```
<"DIR: ",(08,58)>SeleDir
      : <"Dir A:">dira {$$.dir = 'A';}
      | [superuser()] <"Dir C:">dirc {$$.dir = 'C';}.
      | <"Dir D:">dird {$$.dir = 'D';}.
```

□

Como se disse, um Símbolo (Terminal, ou Não-Terminal) denota uma frase (elementar, ou complexa) enviada pelo Utilizador:

- Os vocábulos da linguagem usada pelo Utilizador (as palavras com as quais este formará as suas mensagens) serão denotadas por símbolos T.
- Cada conjunto de termos do mesmo tipo (isto é, as diversas sub-mensagens, ou a mensagem completa, transmitida pelo Utilizador) será denotado por um símbolo NT.

Exemplo 3 *Alguns exemplos poderão ajudar a esclarecer estes conceitos:*

a) *São exemplos de "termos" (denotados por símbolos NT): a* Escolha duma opção a partir dum menu*; a* Identificação de Utilizador *(p. ex. um par Nome e Palavra-Reservada); a* Ficha de um aluno *(com todos os seus items de informação).*

b) *São exemplos de "vocábulos" (representados por símbolos T):* a Opção escolhida; o Nome de utilizador, o Numero Interno de aluno.

□

Para caracterizar as interacções (Pre- e PosAcções) definiu-se um conjunto de atributos, que se julgam genéricos e suficientes. Ao escrever a gramática, bastará indicar o valor desses atributos; caso nada seja dito, ser-lhes-á associado um valor por defeito. Os exemplos 1 e 2 ilustram também o uso destas atribuições.
Alguns dos atributos definidos são:

- O **titulo** da operação.
- A **posição no ecran** da janela associada a essa operação.
- A **ficha**, ou **máscara de ecran**, que define o formato segundo o qual pretendemos que sejam trocadas informações.
- A **condição de validação** que deve ser verdadeira para se considerar terminada a leitura de uma ficha.
- O **valor por defeito** a associar a um dado item, na produção argumentos (antes de ser lido, deve mostrar-se no ecran o item em causa preenchido com esse valor, depois da leitura, o item materá o valor se apenas for digitado 'return').
- Os **atributos visuais** dum item, na produção argumentos (p.ex., leitura sem eco).

Além destes atributos de interacção prédefinidos, pode associar-se a qualquer NT um outro atributo que contenha o valor semântico desse NT, sintetizado (calculado) durante o respectivo reconhecimento (esse atributo pode ser de qualquer tipo).
Assume-se que a cada símbolo T estará sempre associado um atributo intrínseco (fornecido pelo analisador léxico) que define o seu valor, isto é, corresponde à "string" de caracteres que formam o símbolo.

4.2 Comparação com aproximações afins

A ideia básica subjacente ao desenvolvimento das GIGs pode enunciar-se como sendo o uso das gramáticas como método de programação de interfaces.
Este mesmo princípio geral —*usar gramáticas como paradigma de programação*— foi já proposto por alguns outros autores. Nomeiem-se, por exemplo, Christiansen (cf. [Chr88]), que alvitra o uso de *gramáticas generativas* e Nakata et al (cf. [YN88a, YN88b]), onde se sugere o uso de *Coupled Context Free Grammars* como método de programação.
Apesar de uma GIG poder ser vista como uma gramática generativa, na medida em que dá origem a diferentes gramáticas em função do estado global do processador em cada instante (devido às *guardas*), a GIG não é semelhante ao formalismo de Christiansen [Chr85] — de acordo com [Chr88], as GIGs podem ser reunidas com as gramáticas

Estado: ATM

$InfCartao :: CP : CodigoPessoal$ - dados a serem lidos
$CBD : CodigoBd$ - apos introducao do
$NC : NumConta$ - do cartao
$NT : NumTentativas$
$SO : SaldoOffline$ - plafond em "off line"

$StatusBd = CodigoBd \xrightarrow{} online|offline$ - estados das BDs

$InfBd = CodigoBd \xrightarrow{} Contas$ - Contas por BD

$Contas = NumConta \xrightarrow{} InfConta$ - inf. de conta

$InfConta :: TC : Titular - set$
$SC : Saldo$

$SaldoAtm = Interger$

$NotasAtm = ValorNota \xrightarrow{} Quantidade$

$ValorNota = 1|5|10$

Outros tipos auxiliares são definidos por:

$Saldo : Real$
$Quantia : RealPos$

Torna-se agora possível especificar os tipos das operações principais consideradas, e as condições necessárias a que conduzam a uma transição válida no estado da aplicação (designadas pré-condições)[1], bem como as condições de contexto, sob a forma de **guardas**, para que possam ser interactivamente seleccionadas.

A cláusula de tipo, sempre que se trate de operações sobre o estado global do sistema (estado indentificado por ATM), indicará um objecto argumento do tipo ATM (o estado inicial), os tipos ordenados dos argumentos da operação e o resultado da mesma, necessariamente do tipo ATM (o estado final).

Por exemplo, a cláusula

$$Tipo : ATM \times InfCartao \xrightarrow{} ATM$$

especifica uma qualquer operação que recebe como argumento o estado do sistema (ATM) e um tuplo (ou record) do tipo InfCartao e dá como resultado um novo valor para o estado.

[1] Não se especificam por serem obvias as pós-condições, ou seja, as alterações ao estado resultante da execução da operação.

Assumindo ao longo da especificação ATM como o identificador de tipo associado ao estado da aplicação, escreveremos simplesmente, assumindo-o como argumento e resultado de modo implícito:

Tipo : InfCartao —

Funções auxiliares e funções da aplicação, porque não alteram o valor do estado, podem recebê-lo como argumento mas, obviamente, terão como resultado o tipo do resultado da função (que não será pois ATM).

Assim, uma função que determine se uma dada Base de Dados se encontra "on line", dando nesse caso o valor *true* como resultado, teria por cláusula de tipo:

Tipo : ATM × CodigoBd — Bool

Passemos, após esta explicação do significado da notação, à especificação de cada operação da aplicação atrás descrita.

Op : le — CodigoPessoal
Contexto = TRUE
Tipo : CodigoPessoal —
Pre : TRUE
Post : .....

A operação é invocável em qualquer contexto e a pré-condição indica que o valor do argumento, desde que obedeça ao tipo indicado e ao invariante desse tipo, é (em termos sintáticos e léxicos) válido, podendo a operação ser executada.

Op : consultaSaldo
Contexto = bdligada( )
Tipo : — Saldo
Pre : TRUE
Post : .....

Esta operação é invocável apenas se o resultado da invocação da função auxiliar **bdligada** (que cf. o *CodigoBd* lido do cartão e a informação em *StatusBd*, feita a correspondência, retorna V ou F) for TRUE, isto é, se a base de dados da conta estiver "on line".

Op : consultaMovim
*/ * o mesmo que o anterior apenas com resultado diferente * /*

Op : pagamento
Contexto = TRUE
Tipo : InfPagam —
Pre : TRUE
Post : .....

como opções de levantamento as correspondentes às operações, LEV1, LEV2, LEV3, LEV5 e LEV10 e também "outros LEVANTAMENTOS", correspondendo à operação originalmente especificada e que pede ao utilizador a introdução da quantia desejada.

Temos assim concebida uma IU função de certas características particulares da aplicação, apresentando uma correcta adaptabilidade ao contexto e para a qual conhecemos até os requisitos quanto ao modelo da aplicação. Sabemos quais os objectos da aplicação que devem ser acedidos, como e quando, e as correspondências (de momento não explicitadas ainda) entre operações da IU e operações da aplicação.

O estabelecimento destas correspondências e a definição da estrutura do diálogo (até aqui irrelevante e não detalhada) far-se-á agora na fase de especificação formal da IU à custa de Guiões de Interacção (GIs). No entanto, e desde já, o desenho de parte da IU foi integrado no desenho da camada computacional!

5.2 Especificação da IU usando GIs

Conhecidas as operações da aplicação, os contextos próprios para a sua invocação, os tipos dos seus argumentos e as suas pré-condições, os **Guiões de Interacção** poderão ser agora escritos adicionando a esta informação a especificação do fluxo do diálogo utilizador-IU-aplicação.

Vejamos então os GIs correspondentes às operações principais, começando pela operação le-CodigoPessoal, a mais rica em termos interactivos.

```
GI : le - CodigoPessoal
VARS
    cp : CodigoPessoal                                  - buffer
    c : char
    k : key
STATE - APL
    nt <= ATM.InfCartao.NT                              - num.tentativas
    cod <= ATM.InfCartao.CP                             - cod.pessoal
EVSEQ
    NULL - (input(d)*^{0..4} input(k)*^{0..$} NULL)
STATES
    start ⇒ cp = NULL.out("introduzaCodigo...")
    ledigito ⇒ cp = cp + d
    novatent ⇒ nt = nt - 1; GravaCartao(nt)
    end2 ⇒ apreendeCartao()
TRANS
    start  NULL.NULL — ledigito1
    ledigito1 : input(d).len(cp) < 3 — ledigito
    ledigito : input(d).len(cp) = 3 — validacp
    ledigito.ledigito1 : input(k).k = 'corrigir' — start
    ledigito : input(k).k = 'continuar' — ledigito
    ledigito1 : input(k).k = 'continuar' — ledigito1
    valicacp : NULL.cp = cod — end1
    valicacp : NULL.cp ≠ cod — testatent
    testatent : NULL.nt > 1 — novatentativa
    novatentativa : NULL — start
    testatent : NULL.nt = 1 — end2
```

O GI especifica que, a qualquer momento da introdução de um dígito, accionar a tecla 'corrigir' corresponde a limpar o "buffer" e reiniciar o processo de introdução. Accionando a tecla 'continuar' pura e simplesmente não altera o estado interactivo e logo que forem lidos 4 dígitos o código é validado. Note-se que dado o teclado ser apenas numérico (ou com teclas de função) a entrada não necessita de reconhecimento nem de validação especial. Ou é um caracter representativo de um dígito ou uma "string" resultante de accionamento de uma tecla de função.

No entanto, uma situação não se encontra aqui contemplada. De facto, a qualquer momento da introdução do código pessoal o accionamento da tecla 'anular' faz com que a ATM faça sair o cartão do utilizador. Sendo tal evento assíncrono, o correspondente GI deverá permanecer activado em paralelo com o guião anterior e com qualquer guião com o qual este coexista.

Definindo o GI ANULAR como,

[Mo81] T.P. Moran. *The Command Language Grammar - a representation for the user interface of interactive computer systems*. Int. J. Man-Machine Studies, 15(1), 1981.

[MO85] F.M. Martins, J.N. Oliveira. *Graphics Programming with Archetypes - A Preliminary Study*. Proc. of the EUROGRAPHICS'85 Conference, Nice, France, North-Holland Ed., 1985.

[MO86] F.M. Martins, J.N. Oliveira. *On the Specification of Archetype-Oriented Graphics Editors*. EUROGRAPHICS'86 Special Session, Lisboa, Portugal, Aug. 1986, Int. Rep. CCES:FMM-JNO/R2-86, Univ. do Minho, Braga, Portugal.

[MO90] F.M. Martins, J.N. Oliveira. *Archetype-Oriented User Interfaces*. Computer & Graphics, 14(1), Jan. 1990.

[Ni86] J.Nielsen. *A virtual protocol model for computer-human interaction*. Int. J. Man-Machine Studies, 24(3), 1986.

[Ol86] D. R. Olsen. MIKE: the menu interaction kontrol environment. *IEEE Transactions on Graphics*, 5(4), Oct. 1986.

[Or68] W. Orr. *Conversational Computing*. John Wiley, New York, 1968.

[Pf85] G. E. Pfaff. *User Interface Managemente Systems*. Springer-Verlag, Berlin, 1985.

[Re81] P. Reisner. *Formal Grammar and Human Factors Design of an Interactive Graphics System*. IEEE TSE, SE-7(2), Mar. 1981.

[SAFW72] M. E. Senko, E. B. Altman, P. L. Fehder, and C. P. Wang. *A data independent architectural model: Four levels of description from logical structures to physical structures*. Technical Report, IBM corp., Feb. 1972.

[Sc80] B. Schneiderman. *Software Psychology: Human Factors in Computer and Information Systems*. Little Brown and Co., Boston, Mass., 1980.

[Sc82a] B. Schneiderman. *The future of interactive systems and the emergence of direct manipulation*. in Human Factors in Interactive Computer Systems, Vassilou Y., Ed., Ablex Pub., New Jersey, 1982.

[Sc82b] B. Schneiderman. *Multiparty grammars and related features for defining interactive systems*. IEEE TSMC, SMC-12(2), 1982.

[Sc83] B. Schneiderman. *Direct Manipulation: A Step Beyond Programming Languages*. Computer, 16(8), Aug. 1983.

[So87] S. Sommerville. *Review of Formal Methods in HCI*. in Colloquium on Formal Methods and Human-Computer Interaction, IEE Computing and Control Division, London, UK, 1987.

[St84] R. Struder. *Abstract Models of dialogue concepts*, in Proc. 7. Int. Conf. Software Eng., IEEE, 1984.

[Su82] B. Sufrin. *Formal specification of a display-oriented text editor*, Science of Computer Programming, 1, 1982.

[Su89] A. Sutcliffe. *Task analysis, systems analysis and design: Symbiosis or Synthesis?*, Interacting with Computers, 1(1), April 1989.

[Th85] H. Thimbleby. *User Interface Design: Generative User Engineering Principles*, in Fundamentals of Human-Computer Interaction, Academic Press, London, 1985.

[Th86] H. Thimbleby. *User Interface Design and Formal Methods*, Computer Bulletin, Sept. 1986.

[Th87] H. Thimbleby. *in Colloquium on Formal Methods and Human-Computer Interaction*, IEE Computing and control Division, London, UK 1987.

[vdBPH83] Jan van den Bos, M. J. Plasmeijer, and P. H. Hartel. Input-output tools: a language facility for interactive and real-time systems. *IEEE Transactions on Software Engineering*, 9(3):247–259, 1983.

[Wa85] A. I. Wasserman. *Extending State transition diagrams for the specification of human-computer interaction*, IEEE TSE, SE-11(8), 1985.

[YN88a] Y. Yamashita and I. Nakata. *Programming in Coupled Context-Free Grammars*. Research Report ISE-TR-88-70, Univ. of Tsukuba, Institute of Information Sciences and Electronics, Jun. 1988.

[YN88b] Y. Yamashita and I. Nakata. *Programming in Gramp: A programming language based on CCFG*. Research Report ISE-TR-88-73, Univ. of Tsukuba, Institute of Information Sciences and Electronics, Junho 1988.