CAPÍTULO 7

UM MÉTODO RIGOROSO DE DESENVOLVIMENTO DE SISTEMAS INTERACTIVOS

ÍNDICE

7.1.- Introdução.

Apresentam-se neste capítulo as etapas fundamentais do método que se propõe para a concepção e o desenvolvimento integrado de Sistemas Interactivos a par-tir da especificação formal da aplicação[1].

O método apresentado sugere como ponto de partida a especificação formal axiomática (ou teoria) do sistema a desenvolver. Esta primeira especificação, de elevado grau de abstracção, poderá imediatamente servir como auxiliar para a realização de análises de equivalência e complexidade de *operações* e *tarefas*.

A especificação formal por modelos da camada computacional é o passo se-guinte do método, correspondendo formalmente a uma concretização da especi-ficação anterior consistindo da construção de um modelo daquela. Desta espe-cificação por modelos é possível extrair informação fundamental para a concep-ção sistematizada da IU, em particular para a concepção do *modelo da aplicação* e do *controlador do diálogo*.

A prototipagem da camada computacional é um passo importante do méto-do, já que, conforme se pretende, deve auxiliar e estar ligada à própria prototi-pagem da IU. No entanto, a construção do protótipo da IU, quer seja realizada manualmente quer de forma semi-automática usando o sistema GAMA-X, cuja implementação é apresentada no capítulo seguinte, deve ter em consideração as informações e restrições semânticas recolhidas a partir da especificação da ca-mada computacional. Esta informação semântica irá permitir que o comporta-mento (ou *feel*) do protótipo da IU fique, desde logo, ajustado à semântica da camada computacional a ser criada, garantindo-se deste modo que, ainda que as concepções evoluam em separado, os seus comportamentos se ligam de for-ma coerente e natural.

Mostra-se assim que, a partir da especificação formal da camada computa-cional, e salvo iterações na concepção resultantes porventura de alterações nos requisitos, uma IU pode de imediato ser construída exibindo um comportamento que se pode considerar igual ao definitivo, podendo ficar apenas por definir al-gumas das características da sua apresentação.

O objectivo do método é indicar os sucessivos passos a realizar pelo imple-mentador, ou equipa de implementadores, desde a recepção dos requisitos, quer computacionais quer de interacção, até à rigorosa implementação quer da ca-mada computacional quer da respectiva camada interactiva, sendo certo que a maior parte da informação de comportamento da segunda é orientada pela pri-meira, dado que o objectivo fundamental é com esta estabelecer comunicação.

---

[1] Este requisito do método pode ser flexibilizado desde que a informação relevante possa ser recolhida por outro qualquer processo.

Distinguir-se-ão ainda duas diferentes estratégias, já experimentadas aliás, para a implementação de Sistemas Interactivos baseados no modelo MASS. Ainda que partindo ambas da especificação formal da camada computacional, a primeira estratégia privilegia uma aplicação sistemática do método, ainda que não recorrendo a quaisquer ferramentas para automatização do processo, ou seja, seguindo-o de forma empírica e não automatizada. A segunda estratégia baseia-se na utilização do sistema GAMA-X, o sistema de desenvolvimento de IU implementado, que serve de suporte à geração semi-automática de IU.

7.2.- O Método Proposto.

Para uma mais fácil apresentação dos diversos passos do método que se propõe, vamos considerar um problema concreto, que é também um clássico na área: a construção da IU para uma máquina ATM[2] ligada ao sistema bancário.

Consideremos então a especificação informal do problema, sendo apresen-tados em conjunto os requisitos funcionais e de interacção.

SISTEMA ATM: Requisitos.

Um banco adquire uma máquina ATM (vulgo máquina "MultiBanco") para a qual pretende desenvolver o seu próprio sistema interactivo, ou seja, a aplica-ção e a Interface com o Utilizador.

A máquina ATM deve permitir que qualquer utilizador, desse ou de outro banco ligado à rede, munido de um cartão de acesso, possa realizar transacções sobre a sua conta bancária usando o seu cartão. Cada máquina tem uma pequena base de dados local, onde é guardada informação sobre as bases de dados activas e sobre os pagamentos realizados.

Cada conta bancária possui um número único, um titular principal, um saldo, um "plafond" de saldo negativo e um "histórico" com os últimos movimentos realizados, admitindo-se ainda que não há limitação nos levantamentos, ex-cepto a que resulta da existência de saldo.

Cada cartão possui uma pequena banda magnética que guarda a informação de acesso ao sistema, designadamente, o PIN[3], ou código pessoal, e o NIB[4], número que identifica univocamente a conta bancária a que o cartão dá aces-so, pois identifica inequivocamente o banco, a dependência e, dentro desta, a conta.

---

[2] *Automatic Transaction Machine.*

[3] *Personal Identification Number.*

[4] Número de Identificação interBancária.

Considera-se ainda que cada cartão possui registado um "plafond off-line" permitindo ao seu utilizador realizar um ou mais levantamentos diários até tal quantia, mesmo que a base de dados do seu banco esteja em "off". O cartão contém também o número de tentativas disponíveis, num máximo de 3, para a introdução correcta do PIN.

Introduzido um cartão na máquina ATM, o utilizador tem um número de tentativas indicado pelo cartão para introduzir correctamente o PIN. Podendo o utilizador cancelar a qualquer momento esta, e qualquer outra, operação, caso esgote o número de tentativas disponíveis o cartão será apreendido, terminando a interacção.

Introduzido correctamente o código do cartão, a funcionalidade do sistema deve ser oferecida ao utilizador de modo simples, através de um sistema de menus, permitindo-lhe realizar as operações disponíveis, sendo de notar que a disponibilidade de algumas operações está condicionada pela possibilidade de acesso à base de dados onde estão sediadas as informações das respecti-vas contas.

- ***Consulta de Saldo**: caso seja possível o acesso à base de dados da conta a operação deverá estar disponível, caso contrário deverá apare-cer como inacessível. Se acessível a operação dá como resultado o saldo da conta.*

- ***Consulta de Movimentos**: se acessível, esta operação deverá dar como resultado uma lista dos últimos movimentos registados na conta.*

- ***Levantamento**: esta operação apresenta alguma complexidade. Em primeiro lugar, caso a máquina ATM, não tenha dinheiro disponível, tal deve ser imediatamente indicado, indisponibilizando a operação. Ainda que a máquina ATM tenha dinheiro, várias situações podem, ainda assim, acontecer após a introdução do cartão. Em primeiro lugar, a base de dados onde a conta se encontra sediada pode estar ou não "on-line". Se estiver "on-line", então, lida a quantia pretendida pelo utilizador, ha-verá que verificar se o saldo de conta, tendo em atenção o "plafond", permite tal levantamento. Caso o mesmo não seja possível, a adequada mensagem deve ser produzida, terminando-se a interacção. Caso tal im-portância seja, do ponto de vista da conta, levantável, haverá ainda que verificar se a máquina ATM tem uma combinação de notas capaz de to-talizar tal importância. Por outro lado, caso a base de dados da conta esteja "off-line", então, o utilizador poderá levantar uma importância que será, no máximo, o "plafond off-line" registado no próprio cartão. A quantia a levantar é seleccionada através de um menu de quantias.*

- ***Outros Levantamentos**: operação idêntica à anterior, sendo neste caso a quantia introduzida via teclado e não seleccionada num menu.*

? ***Pagamento***: *é registado na base de dados local o pagamento de uma quantia a uma entidade cujo código é introduzido. A quantia não é debitada na conta dado que a transacção só vai ser feita posteriormente.*

Dados os requisitos funcionais da aplicação a desenvolver e, tal como é usual, um conjunto mínimo de requisitos de interacção, o método segue o méto-do CAMILA/SETS, apresentado no capítulo 3, para o desenvolvimento da camada computacional. Assim, a primeira fase do método deverá levar o projectista até à construção do protótipo CAMILA da aplicação, protótipo desenvolvido a partir da especificação de um modelo matemático da aplicação.

FASE I : Dos Requisitos à Especificação e Protótipo.

Esta primeira fase corresponde ao ciclo apresentado na figura seguinte, segundo o qual, partindo dos requisitos funcionais da aplicação e possivelmente também de alguns requisitos de interacção apresentados, se realiza uma especificação algébrica-axiomática, criando uma *teoria* para o problema. Este nível de especifi-cação, dado o seu grau de abstracção e o seu carácter simbólico, pois baseia-se na reescrita de termos representativos de sequências de operações, pode ser até considerado um nível de abstracção ideal para representação e análise do que usualmente se designa por *tarefas do utilizador*.

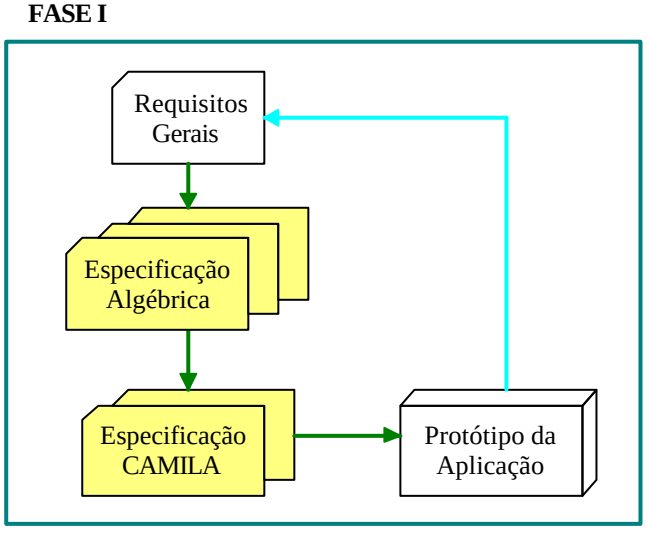

Fig. 7.1 - Dos Requisitos à Especificação e Protótipo.

Realizada, opcionalmente, esta especificação algébrica-axiomática, de facto um conjunto de teorias, i.é., assinaturas e conjuntos de axiomas,

passa-se en-tão à especificação por modelos matemáticos usando a linguagem CAMILA.

A linguagem CAMILA, na sua versão modular, permite que ?-álgebras, que são modelos das respectivas assinaturas, possam ser compostas usando opera-dores especiais de ligação designados *módulos*, dando como resultado uma ou-tra álgebra[5]. As expressões resultantes da composição de módulos, designadas por *expressões modulares*, são formalmente termos de uma meta-assinatura, cujas espécies são assinaturas ou *interfaces*, constantes são implementações e módulos são operadores. O tratamento formal de todos estes conceitos e cons-truções, que conciliam as especificações algébricas com especificações por mo-delos, ou seja, introduzindo a noção de *estado*, pode ser encontrado em [Oliveira 93].

Assim, quer a especificação venha a ser modularizada ou não, o ponto de partida deverá ser então a especificação das diversas interfaces dos modelos a construir, caso vários modelos venham a ser necessários para concretizar a im-plementação.

Comecemos então por especificar a *interface*, designada $?_{ATM}$, do problema, que define a linguagem do sistema, não se tendo procurado modularizar a espe-cificação dado o objectivo não ser esse mas sim a inferência de informação para a construção da IU.

interface I_{ATM} =

sorts ATM, InfCartão, PIN, Saldo, MovmList, Quantia, Entidade, Tentativas, QuantiaSel, QuantiaSelSet;

functions

saldo? : ATM ? Saldo
movimentos? : ATM ? MovmList
BdOn? : ATM ? Bool ; *auxiliar*
Pin_Ok? : ATM x PIN ? Bool ; *auxiliar*
NumTent? : ATM ? Tentativas ; *auxiliar*
pode_levantar? : ATM x Quantia ? Bool ; *auxiliar*
max_levantam? : ATM ? Quantia ; *auxiliar*
Seleccionáveis? : ATM ? QuantiaSelSet ; *auxiliar*
introduz_cartão : ATM x InfCartão ? ATM
introduz_pin : ATM x PIN ? ATM
pagamento : ATM x Quantia x Entidade ? ATM
levantamento : ATM x QuantiaSel ? ATM

[5] Ver definições detalhadas nos APÊNDICES A (CAMILA) e B (NOTAÇÃO E DEFINIÇÕES ALGÉBRICAS).

```
    outros_levantams : ATM x Quantia ?  ATM
    retira_cartão :  ATM ?  ATM
    captura_cartão : ATM  ?  ATM
end
```

Consideremos agora a especificação por modelos matemáticos do estado do sistema, estado esse designado por ATM.

State: ATM

```
ATM ::  Cartão : InfCartão                  ; informação do cartão
        StatusBd : CodigoBd ?  Status       ; tabela das BDs on e off
        InfBd : CodigoBd ?  Contas          ; contas das BDs acessíveis
        Contas : NConta ?  InfConta         ; dados de cada conta
        StatusAtm : InfAtm                  ; informação interna da ATM
        Pagamts : Pagamt-list               ; pagamentos a transferir
        Capturas : InfCartão-set            ; cartões apreendidos
        Dia : Data                          ; data do dia

InfConta ::  Tit : Titular
             Sld : Saldo
             Plf : Plafond
             Movs : MovmList

Movim ::     Dt : Data
             Descr : Descrição
             Imp : Quantia

MovmList = Movim-list

Pagamt ::    Dt : Data
             Ent : Entidade
             Val : Quantia

InfCartão ::  Pin : PIN                     ; código pessoal do cartão
              CodBd : CodigoBd              ; base de dados da conta
```

```
                NumCt : NConta          ; número da conta
                NumTent : Tentativas        ; tentativas possíveis
                SldOff : Saldo          ; saldo "off-line" do dia

InfAtm ::       SldAtm : Quantia        ; quantia existente na máquina
                Notas : ValorNota ? Nat ; tabela de notas
Saldo = Real
ValorNota = { 1, 2, 5, 10 }
PIN = Dígito-list
Dígito = { 0 .. 9 }
CodigoBd, NConta, Data, Entidade, Descrição : STR
Quantia, Plafond : Nat
Status = "OnLine" | "OffLine"
QuantiaSel = 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40
QuantiaSelSet = QuantiaSel-set
Tentativas = 0 .. 3
```

INVARIANTES:

O primeiro invariante visa garantir que, dada a tabela que indica qual o es-tado de cada base de dados, designadamente "on-line" ou "off-line", então o conjunto dos códigos das bases de dados ligadas (1) é coincidente com o domí-nio da função finita que dá acesso às respectivas contas (2).

```
FUNC inv1-ATM(atm: ATM) : Bool
; todas as bases de dados "on-line" têm as suas contas acessíveis
RETURNS
    (1)     let ( Bdon = { cdbd | cdbd ? dom(StatusBd(atm)) ;
                                StatusBd(atm)[cdbd] == "OnLine" } )
            in
    (2)         dom(InfBd(atm)) == Bdon;
```

O segundo invariante garante que nenhum cartão introduzido possui um número de tentativas igual a zero, em cujo caso já teria sido apreendido, que possui um código de base de dados reconhecido, e ainda que, caso tal base de dados esteja activa, o número de conta é reconhecido.

```
FUNC inv-InfCartão(atm: ATM) : Bool
RETURNS
        let ( tent = NumTent(Cartão(atm)),
              codbd = CodBd(Cartão(atm)),
```

```
                nconta = NumCt(Cartão(atm)),
                BdOn = { cdbd | cdbd ? dom(StatusBd(atm)) ;
                                StatusBd(atm)[cdbd] == "OnLine" },
                Bdligada = codBd ? BdOn )
        in
           ( tent > 0 ) ? ( codbd ? dom(StatusBd(atm)) ) ?
           if Bdligada
                then nconta ? ran(InfBd(atm)[codbd]) else true ;
```

O PIN a introduzir pelo utilizador deve ser uma sequência de caracteres de exactamente quatro dígitos.

```
FUNC inv-PIN(pin: PIN) : Bool
RETURNS
        length(pin) == 4;
```

Sobre as contas acessíveis à máquina ATM, deverá ser definido um invari-ante indicando que nenhuma destas pode alguma vez ter um saldo inferior ao "plafond" de débito definido. Este invariante, que especifica uma propriedade de consistência a que as contas acessíveis devem obedecer, impõe também restri-ções a todas as operações que podem alterar tais contas já que nenhuma destas operações poderá partir de um estado válido e conduzir o sistema a um estado inválido. Portanto, todas as operações que alterem contas bancárias devem pre-servar este invariante, sendo no método CAMILA/SETS para desenvolvimento da camada computacional esta prova matemática de carácter obrigatório.

```
FUNC inv-Contas(atm: ATM) : Bool
; nenhuma conta acessível possui um saldo inferior ao "plafond" de débito
; indicado na conta
RETURNS
        let ( BdOn = { cdbd | cdbd ? dom(StatusBd(atm)) ;
                              StatusBd(atm)[cdbd] == "OnLine" } )
        in
           all( codbd ? BdOn :
                all( nc ? dom(InfBd(codbd)) :
                        let  ( conta = InfBd(codbd)[nc] )
                        in   Sld(conta) + Plf(conta) 8 0 ));
```

Será ainda de considerar um invariante que garanta que a informação sobre a quantidade de notas existentes na máquina ATM coincide com o respectivo saldo, ou seja, que o somatório do valor das notas é igual ao saldo da máquina.

```
FUNC inv-Máquina(atm: ATM) : Bool
```

```
        ; o saldo disponível na máquina é igual ao somatório das notas
        RETURNS
                let (  infNotas = Notas(StatusAtm(atm)),
                       saldoAtm = SldAtm(StatusAtm(atm)),
                       total$ = +-ório < val * infNotas[val] | val ?  dom(infNotas)
>
                   )
                in
                   total$ == saldoAtm;
```

Finalmente, o invariante global do estado do sistema pode escrever-se como a conjunção dos invariantes atrás definidos, ou seja, como:

```
        FUNC inv-ATM(atm: ATM) : Bool
        RETURNS
                inv1_ATM(atm) ? inv_Contas(atm) ?
                inv_InfCartão(atm) ? inv_Máquina(atm);
```

Modelado o estado global do sistema e especificados os invariantes de tipos de dados e outros, o passo seguinte consiste na especificação de cada uma das operações, indicando-se explicitamente as suas pré-condições, ou seja, as con-dições contextuais semânticas em que as mesmas podem ser executadas com garantia de execução correcta. Tal como a informação sobre invariantes, a infor-mação sobre as pré-condições é, no método apresentado, crucial para o correcto desenvolvimento da IU.

OPERAÇÕES:

Apresentam-se a seguir as especificações formais das operações do sistema. Ao longo da apresentação destas especificações, para além dos comentários ne-cessários à sua compreensão, introduzem-se comentários relativos a algumas regras para a inferência de informação de interacção que na fase seguinte irá auxiliar na escrita dos Guiões de Interacção. Deve desde já notar-se que, não contendo as especificações das operações qualquer informação de interacção, é a partir da análise e consideração dos invariantes e das pré e pós condições das operações que se pode obter a informação necessária.

A primeira operação especifica a alteração do estado do sistema resultante da introdução na máquina ATM do cartão de um utilizador. Lida automatica-mente a informação contida na banda magnética do cartão, esta informação é guardada no estado do sistema. Esta operação, ainda que não seja invocável pe-lo utilizador, pode associar-se ao evento de introdução de cartão na máquina.

```
        FUNC introduz-cartão(atm: ATM, infc: InfCartão) : ATM
```

```
; operação que regista a introdução de um cartão e sua informação
RETURNS
        [ infc / Cartão ]atm;[6]
```

A operação especificada a seguir é a que corresponde à introdução pelo uti-lizador do código pessoal, que vai ser comparado com o PIN do cartão já intro-duzido, sabendo-se que o cartão será apreendido caso esta seja a última tenta-tiva possível e os códigos não coincidam, .

```
FUNC introduz-pin(atm: ATM, pin: PIN) : ATM
; operação de validação do código pessoal segundo as condições indicadas.
RETURNS
        let (  pincode = Pin(Cartão(atm)),
               tentativ = NumTent(Cartão(atm)) )
        in
           if pincode == pin
              then atm
              else
                 if tentativ == 1
                       then captura_cartão(atm)
                       else  retira_cartão(atm);
```

As operações *captura_cartão(atm)* e *retira_cartão(atm)*, dado não terem interferência na camada interactiva, não serão aqui especificadas, sendo óbvio que o deverão ser para a construção do protótipo da aplicação. É também de notar que esta operação *introduz_pin*, dado ser obrigatória e do controlo da máquina, isto é, não invocável pelo utilizador, não apresenta qualquer pré-condição e não tem qualquer dependência do contexto. A qualquer momento, durante a intro-dução do PIN, o utilizador pode cancelar a operação, em cujo caso o cartão lhe é devolvido.

A operação seguinte é uma operação de consulta à base de dados onde a in-formação sobre a conta associada ao cartão reside. Esta base de dados poderá estar ou não activa, sendo esta a pré-condição a testar para que a operação pos-sa ser realizada, e, do ponto de vista interactivo, seleccionável.

```
FUNC saldo?(atm: ATM) : Saldo
; operação de consulta do saldo de uma conta
PRÉ: CodBd(Cartão(atm)) ?  dom(InfBd(atm))     ; ou BdOn?(atm)
RETURNS
```

[6] Esta notação, que especifica a alteração do campo do tuplo identificado, passando este campo a assumir o valor indicado, deve ser considerada como uma simplificação da notação existente em CAMILA, mais explícita cf. *ATM(infc, StatusBd(atm), InfBd(atm), Contas(atm), StatusAtm(atm), Pagamts(atm))*, que obriga à indicação do valor de cada um dos campos do tuplo a construir. A linguagem gerada, *xmetoo*, suporta a construção utilizada, através de uma função de reescrita de tuplos, designada *plusq* [Almeida e Barbosa 91].

```
        let   (  bd = CodBd(Cartão(atm)),
              numct = NumCt(Cartão(atm)),
              contas = InfBd(atm)[bd],
              infconta = contas[numct] )
        in
              Sld(infconta);
```

Note-se que, por não conter a especificação da camada computacional qual-quer informação sobre aspectos interactivos, a condição de selectibilidade de uma operação deve ser inferida da pré-condição. A pré-condição impõe, em geral, restrições aos valores que os argumentos da operação podem assumir. Sempre que tal acontece, podemos inferir que é da responsabilidade da camada interactiva a garantia da correcção de tais valores, dado que estes devem ser li-dos por seu intermédio.

Porém, situações podem acontecer em que na pré-condição não são referi-dos argumentos da operação mas apenas componentes do estado da aplicação. Quando tal acontece, deve inferir-se que a pré-condição especifica as condições de contexto que devem existir para que a operação seja seleccionável. Esta infor-mação deverá ser transcrita para a cláusula de contexto do guião da operação.

No exemplo, e no sentido de facilitar a determinação do valor da condição de contexto, será neste caso importante que, ao nível da aplicação, uma função particular pudesse ser definida com o objectivo de testar se determinada base de dados, cujo código está registado no cartão, está ou não activa.

```
FUNC BdOn?(atm: ATM) : Bool
; operação que verifica se a Base de Dados do cartão está ligada
RETURNS
        CodBd(Cartão(atm)) ?  dom(InfBd(atm));
```

A operação de consulta de movimentos, tal como a anterior, só estará dispo-nível se a base de dados a que o cartão dá acesso estiver ligada. A pré-condição é neste caso escrita usando a função auxiliar anteriormente apresentada.

```
FUNC movimentos?(atm: ATM) : MovmList
; operação de consulta dos movimentos da conta
do cartão
PRÉ: BdOn?(atm)
RETURNS
        let   (  bd = CodBd(Cartão(atm)),
              numct = NumCt(Cartão(atm)),
              contas = InfBd(atm)[bd],
              infconta = contas[numct] )
```

```
        in
              Movs(infconta);
```

A operação de pagamento implica a leitura de uma quantia e de uma enti-dade, que pode ou não existir, sendo tal pagamento registado no sistema para posterior actualização da conta. A operação limita-se a registar na lista local de pagamentos que um novo pagamento foi realizado, admitindo-se que um outro processamento interno ao sistema realizará posteriormente a actualização das contas. A semântica é plausível ainda que outras pudessem ser consideradas.

```
FUNC pagamento(atm: ATM, qt: Quantia, ent: Entidade) : ATM
; operação de pagamento
RETURNS
        let   (  pagam = Pagamt(Data(atm), ent, qt),
              pagams = Pagamts(atm) )
        in    [ pagams ^ <pagam> / Pagamts ]atm;
```

No entanto, e dado que a operação é de carácter interactivo, algumas consi-derações deverão ser feitas. Em primeiro lugar, a correcção da operação não de- pende da quantia introduzida, dado que, qualquer que esta seja, será sempre correcta. Quanto ao código da entidade duas situações poderiam ser considera-das. Uma primeira situação possível seria admitir-se que qualquer máquina ATM possui, localmente, uma lista das entidades reconhecidas para pagamentos automáticos. Do ponto de vista interactivo, para que a introdução da entidade fosse sempre correcta bastaria que a IU garantisse que apenas entidades em tal lista poderiam ser seleccionadas. Neste caso a operação seria total, ou seja, não produziria nunca erros.

A segunda situação, adoptada no presente exemplo porque mais real, consi-dera que tal lista é inexistente ao tempo da interacção, o que implica que a vali-dação total da transacção deva ser feita em diferido e por um outro programa. De tal decisão resulta a inexistência de pré-condição.

A operação seguinte, de levantamento de uma dada quantia seleccionada a partir de um menu de quantias, é uma operação complexa, mas exactamente por isso, muito interessante do ponto de vista do seu estudo enquanto operação interactiva. Consideremos a sua especificação funcional:

```
FUNC levantamento(atm: ATM, qtsel: QuantiaSel) : ATM
; operação de levantamento via menu de quantias
PRÉ:let   (  sldAtm = SldAtm(StatusAtm(atm)),
          codbd = CodBd(Cartão(atm)),
          bdon? = StatusBd(atm)[codbd] == "OnLine",
          pode_levantar = if bdon?
                          then
                             let   (  nconta = NumCt(Cartão(atm)),
                                   contas = InfBd(atm)[codbd],
```

```
                                        conta = contas[nconta],
                                        saldo = Sld(conta), plf = Plf(conta) )
                                    in saldo + plf
                                else
                                    SldOff(Cartão(atm)) )
            in
                ( sldAtm > qtsel ) ? ha_notas_para(qtsel) ? ( pode_levantar > qtsel
)
        RETURNS
                let ( codbd = CodBd(Cartão(atm)),
                      bdon? = StatusBd(atm)[codbd] == "OnLine")
                in
                    if bdon?
                        then
                            let ( codbd = CodBd(Cartão(atm)),
                                  nconta = NumCt(Cartão(atm)),
                                  contas = InfBd(atm)[codbd],
                                  conta = contas[nconta],
                                  novaconta = [Sld(conta) - qtsel / Sld]conta,
                                  novascontas = contas ? [nconta ? novaconta],
                                  novaAtm = [novascontas / Contas]atm )
                            in novaAtm
                        else
                            let ( infcart = Cartão(atm),
                                  saldoOff = SldOff(infcart),
                                  novoSaldo = saldoOff -qtsel,
                                  novoCartão = [novoSaldo/SldOff]infcart )
                            in
                                [novoCartão/Cartão]atm;
```

sendo desde já de notar a complexidade da pré-condição, provocada pelas diver-sas situações que podem ocorrer nas quais esta operação não deve poder ser in-vocada e, portanto, disponibilizada a nível interactivo. Em primeiro lugar deve contemplar-se a possibilidade de, seleccionada uma das quantias do menu, se verificar que a máquina não possui tal valor disponível (cf. *sldAtm > qtsel*). Por outro lado, e considerada a função auxiliar utilizada, *ha_notas_para(qtsel),* que determina se é possível encontrar uma combinação das notas disponíveis que totalize a importância pedida, pode acontecer que tal quantia, ainda que inferior ao saldo da máquina, não possa ser totalizada com o conjunto de notas existen-te. Finalmente, quer a base de dados da conta esteja ou não activa, é necessário que a quantia pedida esteja dentro do saldo disponível, quer este seja o saldo "on-line" ou o saldo "off-line", o que é determinado encontrando-se primeiro tal valor (cf. *pode_levantar*) e comparando-o de seguida com a quantia pretendida (cf. *pode_levantar > qtsel*).

No entanto, e por muito complexa que seja a pré-condição da função, ela, só por si, não pode garantir qualquer obediência aos princípios de sensibilidade ao contexto e de assistência semântica anteriormente

expressos e aconselhados. De facto, esta especificação da pré-condição da operação de levantamento, começa por admitir que o parâmetro de entrada correspondente à quantia seleccionada é uma correcta selecção, dentro de um menu de opções. No entanto, tal não é uma correcção em termos de semântica dinâmica, i.é., tendo em conta o actual estado interno do sistema, mas tão só sintáctica ou, quando muito, de semânti-ca estática. De facto, a quantia seleccionada só poderia estar incorrecta nestes termos se a operação fosse invocada em modo "batch", em cujo caso poderia até acontecer que, esperando-se um valor para o argumento dentro do conjunto de valores especificado, pudesse aparecer um argumento do tipo "string"
ou fora da gama de valores aceite.

Estas situações não ocorrerão a nível interactivo, se a selecção da quantia for feita sobre um menu. Porém, e afastado este tipo de possibilidade de erro, tal não é ainda suficiente para se garantir a efectiva correcção do argumento, já que para além do menu que limita o conjunto de opções, poderá acontecer que, ain-da assim, possam ocorrer selecções incorrectas tendo em conta o actual estado do sistema. Apenas a aplicação do modelo que foi já advogado, dado ter em con-sideração assistência semântica, pode garantir que apenas são apresentadas no menu de quantias aquelas que, em função do contexto do momento, designada-mente o saldo da conta se a respectiva base de dados estiver activa, ou o saldo do cartão no caso contrário, possam ser seleccionáveis.

Não sendo esta solução garantida pela pré-condição[7], mas dela podendo ser inferida, conforme se procurou mostrar, a solução encontrada deverá ser assim garantida na especificação que vier a ser feita para a camada interactiva relati-vamente a esta operação.

Assim, conforme se verifica com esta operação, torna-se importante que em situações como a apresentada, o sistema interactivo possua um comportamento não tanto de "feedback", ou seja de retorno semântico "a posteriori", mas antes de "feedforward"[8], ou seja, disponibilizando "a priori" informação semântica em função da qual as selecções do utilizador são realizadas, garantindo deste modo que estas são, seguramente, semanticamente correctas, logo executáveis com a garantia de não ocorrência de erros.

Matematicamente, esta diferença de perspectiva corresponde à diferença que existe entre considerar-se que o menu, ou lista de opções a fornecer ao utiliza-dor, é um conjunto finito e estático de valores, tal como em

```
let input = ppickone( { q | q ? QuantiaSel } )
```

[7] nem o podendo ser em qualquer circunstância.

[8] Termo aqui introduzido pelo autor, procurando significar, relativamente ao termo usual "feedback", a possibilidade de se possuir semântica disponível ao nível da IU "antes" das esperadas acções do utilizador.

ou, em contrapartida, um conjunto construído em contexto, usando um predica-do que define as opções disponíveis a partir de um conjunto contendo todas as opções de base, cf. a expressão,

let input = ppickone({ q | q ? QuantiaSel ; *P(q)* })

onde *P(q)*, representa um predicado que calculado em tal contexto determina quais das quantias do conjunto QuantiaSel podem nesse caso ser seleccionadas. Este cálculo deverá ser realizado "a priori" relativamente à apresentação ao utilizador das opções disponíveis, para que, qualquer que seja a selecção feita, possa ser tratada sem erros pela camada computacional. No exemplo da opera-ção *levantamento*, em análise, o predicado deverá ser construído a partir da pré-condição da operação, considerando desta as restrições impostas sobre o argu-mento *qtSel*.

Esta noção simples de "feedforward", sempre que aplicável, faz com que a IU possa garantir que a operação da aplicação, sempre que invocada, corresponda a uma *função total*, isto é, uma função aplicável a todos os pontos do seu domí-nio. A IU servirá assim de *front-end* de protecção e prevenção de erros, pelo que cada função da camada computacional pode passar a ser concebida e abordada como uma *função total*.

A consideração do contexto de selecção de uma operação e da informação a fornecer "a priori" da interacção, às quais se junta informação sobre invariantes de dados, constituem informação fundamental para a implementação do com-portamento "profiláctico" advogado para a IU.

Vamos considerar agora a operação que permite realizar levantamentos de importâncias introduzidas interactivamente através de um teclado. Esta opera-ção, que é uma operação complementar à anterior, visa dar acesso ao utilizador ao levantamento de quantias não contempladas no conjunto discretizado e finito de quantias, QuantiaSel, que lhe é apresentado pela operação anterior. Temos portanto, neste caso, uma situação de escolha sobre um conjunto infinito, situa-ção que conduz, em geral, à aceitação do valor introduzido, e à sua posterior va-lidação.

Note-se a semelhança da especificação desta operação com a especificação da operação anterior.

```
FUNC outros_levantam(atm: ATM, qt: Quantia) : ATM
; operação de levantamento via teclado
PRÉ:let   ( sldAtm = SaldoAtm(StatusAtm(atm)),
          codbd = CodBd(Cartão(atm)),
          bdon? = StatusBd(atm)[codbd] == "OnLine",
          pode_levantar = if bdon?
                          then
                             let   ( nconta = NumCt(Cartão(atm)),
                                   contas = InfBd(atm)[codbd],
                                   conta = contas[nconta],
```

```
                                        saldo = Sld(conta), plf = Plf(conta) )
                                    in saldo + plf
                                else
                                    SldOff(Cartão(atm))
            )
        in
            ( sldAtm > qt ) ? ha_notas_para(qt) ? ( pode_levantar > qt )
    RETURNS
            let ( codbd = CodBd(Cartão(atm)),
                  bdon? = StatusBd(atm)[codbd] == "OnLine")
            in
               if bdon?
                  then
                     let ( codbd = CodBd(Cartão(atm)),
                           nconta = NumCt(Cartão(atm)),
                           contas = InfBd(atm)[codbd],
                           conta = contas[nconta],
                           novaconta = [Sld(conta) - qt / Sld]conta,
                           novascontas = contas ? [nconta ? novaconta],
                           novaAtm = [novascontas / Contas]atm )
                     in novaAtm
                  else

                     let ( infcart = Cartão(atm),
                           saldoOff = SldOff(infcart),
                           novoSaldo = saldoOff -qt,
                           novoCartão = [novoSaldo/SldOff]infcart )
                     in
                        [novoCartão/Cartão]atm;
```

No entanto, e apesar da semelhança das operações do ponto de vista computacional, do ponto de vista interactivo existe grande diferença entre elas, dado terem diferentes requisitos de interacção. Considerando a concepção da interac-ção para esta operação, uma primeira solução consiste em aceitar-se um qual-quer valor e depois realizar a sua validação em função do contexto da aplicação. Esta é, naturalmente, uma solução aceitável e tratável exclusivamente ao nível da IU, conforme se defende.

Porém, alternativas, dentro deste mesmo esquema que não pode fugir à va-lidação posterior do valor lido, podem ser encontradas em auxílio do utilizador. Por exemplo, calculada a quantia máxima levantável num dado contexto, o utilizador poderia ser informado deste valor, diminuindo-se certamente a ocor-rência de erros. A expressão geral de entrada para este caso obedece ainda à ex-pressão anteriormente apresentada, onde um dado predicado é inserido, diferin-do apenas quanto ao efeito que a avaliação de tal predicado provoca ao nível da IU.

```
let input = ppickone( { q | q ? Quantia; 0 ? q ? pode_levantar } )
```

Dentro das considerações anteriormente realizadas relativamente ao referên-cial de interacção seguido, poder-se-á dizer que esta solução é mais adequada que a anterior, pela diminuição das distância expressiva e observacional do sis-tema.

Para além de se deverem ter em conta os requisitos de interacção associados ao sistema interactivo a desenvolver, procurando ajustá-los ao modelo de inter-acção MASS a que a concepção deve obedecer, a consideração destas distâncias é também um factor importante, podendo até ser este o critério decisório relati-vamente a soluções que se apresentem como alternativas.

Finalmente, é de notar que ainda que baseada e orientada por informação resultante da especificação da camada computacional, o que introduz alguma sistematização que se procurará melhor ilustrar na descrição da fase seguinte, a concepção da IU, como qualquer processo de concepção, é um acto criativo ain-da que baseado em regras e conhecimento. Tal parece ter ficado claro das diver-sas considerações de interacção realizadas nesta primeira fase.

Note-se que as únicas funções não consideradas nesta análise da IU para esta aplicação ou são funções auxiliares, logo escondidas do utilizador, ou são funções associadas a eventos não interactivos, i.é., não visíveis para o utiliza-dor, tais como *retira_cartão* e *captura_cartão.*

FASE II.I : Especificação da IU.

Realizada a especificação formal da camada computacional e gerado um protó-tipo, o projecto prossegue agora com a especificação da IU, com o objectivo de se construir igualmente um protótipo desta.

A construção do protótipo da IU pode ser realizada de forma semi-automáti-ca, com a ajuda do sistema GAMA-X cuja arquitectura e funcionamento se apre-sentam em detalhe no capítulo seguinte. Começa-se pela especificação dos Gui-ões de Interacção que irão representar o comportamento da IU, descrevendo em notação própria, anteriormente apresentada, a informação extraída da especifi-cação da camada computacional. Em particular, é utilizada nos GI informação de contexto, pré-condições, dependências entre os argumentos de cada uma das operações (que pode condicionar a sua ordem de entrada), a possibilidade de se interromper a síntese de um comando para alteração de parâmetros do sistema, invariantes de dados, e, como vimos, informação não directamente resultante da especificação mas de requisitos e de decisões de desenho.

A especificação da apresentação, ainda que de forma simplificada, é realiza-da também nesta fase, recorrendo a *Descritores da Apresentação*, que são no en-tanto dependentes da tecnologia de apresentação usada. Os descritores da apre-sentação aqui usados são bastante simples, dado ser a

pretensão principal des-te capítulo a apresentação da metodologia e não a discussão do que se pode es-pecificar em tais descritores.

A figura 7.2 procura ilustrar os passos contemplados nesta fase do projecto, em particular, a especificação dos GI recolhendo informação da especificação (e outra) bem como a especificação dos descritores da apresentação, quer em fun-ção dos requisitos, quer em função da restante informação recolhida.

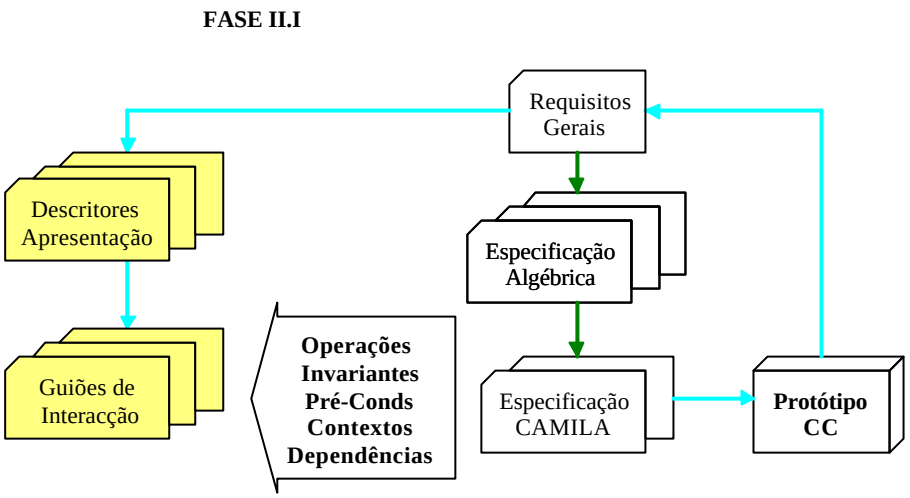

Fig. 7.2 - Especificação da IU.

Retomando o projecto da parte do sistema ATM que se acabou de especificar e de analisar em termos interactivos, vamos agora definir para cada uma das operações o seu guião de interacção.

As operações que devem ser disponibilizadas ao utilizador têm caracterísi-cas diferentes, sendo algumas meros eventos detectados no sistema, ainda que resultantes de iniciativas do utilizador, tais como as operações *introduz_cartão* e *introduz_pin*, enquanto outras irão implicar transacções efectivas no sistema, como *levantamento*, *pagamento*, *saldo?*, etc. Operação a operação, e em função de todas as informações recolhidas da fase anterior, vamos especificar cada um dos seus GI.

A operação de introdução do cartão de utilizador é, do ponto de vista do sis-tema, correspondente à ocorrência de um evento que se traduz na acção auto-mática de leitura da informação contida no cartão. Porém, este é o evento indis-pensável à abertura do sistema às subsequentes possíveis transacções. Assim, um guião particular deverá ser associado a esta operação do utilizador, guião esse que, como qualquer outro, reconhece a ocorrência de um evento e despole-ta uma acção no sistema. O guião será então representado por,

DefGI *Cartão_in*
Declarations
VAR-CTRL *infc: InfCartão*
Behaviour
EVSEQ *input_cartão* . *lê_InfCartão(infc)*
EXEC *introduz_cartão(infc)*[9]
EndGI

Os dois eventos considerados no guião não são eventos-utilizador pelo que o guião não apresenta qualquer tipo definido.

A actividade interactiva seguinte consistirá da introdução do código pessoal, cancelável a qualquer momento, e que deve ser comparado com o código pessoal lido pelo sistema após a introdução do cartão. Quem aceita o código e controla tal introdução é o controlador do diálogo. Porém, quem realiza a sua validação é a camada computacional (cf. a função *Pin-Ok?(pin)*).

DefGI *Leitura_pin*
Declarations
VAR-CTRL *pin: PIN; numttv: Tentativ*
Behaviour
EVSEQ *input(pin)*
TRANS
input(pin): Pin_Ok?(pin) => Null
EXCEP numttv := Numtent?();
Cancel:
Ok:
EndGI

Passemos agora à especificação dos diálogos para as operações verdadeira-mente interactivas do sistema, começando pela especificação da operação de consulta do saldo de uma conta. A pré-condição da operação *saldo?*, dado que a operação não possui argumentos, exprime-se unicamente em função do estado da aplicação, em particular invocando a função *BdOn?*.

[9] É conveniente referir que o parâmetro *atm* passa a ser omitido, por *currying* assumido a nível da camada computacional, onde qualquer função do tipo f: ATM x A ? B é transformada numa operação que assume *atm* como sendo o estado da máquina ATM, tendo a funcionalidade f'_{ATM}: A ? B.

Sendo um predicado envolvendo apenas o estado da aplicação, este predicado vai fazer parte da cláu-sula CONTEXT do respectivo GI.

```
DefGI Saldo
Declarations
        TYPE ValSynth
        VAR-UI sld: Saldo
Behaviour
        CONTEXT BdOn?()
        TRANS
                Start : true => sld := saldo?()
EndGI
```

O GI *Saldo* apenas é activável caso a base de dados do cartão introduzido esteja acessível. Caso esta operação, em tal contexto, faça parte de um qualquer menu de opções deverá estar então assinalada como não acessível. Caso possa ser activado, o guião especifica que um valor deve ser colocado ao dispor da IU, valor esse que resulta da invocação da operação de consulta de saldo. Dado ser um guião para síntese de um valor, é do tipo *ValSynth*.

A operação de consulta de movimentos de conta tem semelhanças com a an-terior, exceptuando-se o facto de que o valor a encontrar para passar à IU é de um tipo estruturado. Este facto é, no entanto, irrelevante do ponto de vista da especificação do controlador, como se verifica pelo guião da operação.

```
DefGI Movimentos
Declarations
        TYPE ValSynth
        VAR-UI movs: MovmList
Behaviour
        CONTEXT BdOn?()
        TRANS
                Start : true => movs := movimentos?()
EndGI
```

A operação seguinte, *pagamentos*, ainda que seja uma operação de alteração do estado, apresenta-se sem restrições. Não depende de qualquer contexto e não impõe restrições aos seus argumentos para além das que possam ser estabeleci-das pelos seus próprios invariantes.

```
DefGI Pagamentos
Declarations
        TYPE Synth
        VAR-CTRL ent: Entidade: qt: Quantia
Behaviour
        EvSEQ input(ent) | | input(qt)
        TRANS
                Cancel: Ok:
        EXEC   pagamento(qt, ent)
EndGI
```

O guião especifica que a ordem de entrada dos valores é irrelevante, o que resulta do facto de não existirem dependências entre eles, que a qualquer mo-mento a operação deve poder ser cancelada, e ainda que o fim da interacção se faz com a ocorrência do evento *Ok*, sendo então invocada a respectiva operação da aplicação.

Especifiquemos agora o GI da operação de levantamento de uma quantia in-troduzida através do teclado. Consideremos em primeiro lugar a especificação deste guião considerando a primeira solução equacionada, ou seja, correspon-dente à aceitação de uma qualquer quantia fazendo-se depois a sua validação. A operação apenas é activável, conforme indica a cláusula de contexto, se a má-quina tiver disponível uma quantia superior a zero.

```
DefGI Outros_levantamentos
Declarations
        TYPE Synth
        VAR-CTRL qt: Quantia
Behaviour
        CONTEXT SaldoAtm?() > 0
        EvSEQ input(qt)
        TRANS
                input(qt): pode_levantar?(qt) => Null
                                EXCEP out("Levantamento Impossível !")
                Cancel: Ok:
        EXEC   outros_levantam(qt)
EndGI
```

A segunda solução apontada é em tudo semelhante a esta quanto ao trata-mento da situação de erro, porém será diferente quanto à apresentação, já que, como se disse, apresentará ao utilizador os limites mínimo e máximo da quantia a introduzir, podendo este, no entanto, não satisfazer a indicação. Neste caso, porém, os valores destas duas variáveis devem ser passadas ao modelo de apre-sentação, pelo que devem ser colocados em VAR-UI.

```
DefGI Outros_levantamentos1
Declarations
    TYPE Synth
    VAR-CTRL qt: Quantia; qtmin: Quantia; qtmáx: Quantia;
    VAR-UI   qtmin: Quantia; qtmáx: Quantia
Behaviour
    INIT qtmin = 0
    CONTEXT SaldoAtm?() > 0
    EvSEQ input(qt)
    TRANS
        Start: qtmáx := pode_levantar?(qt)
        input(qt): 0 ? qt ? qtmáx => Null
                EXCEP out("Levantamento fora dos limites !")
        Cancel: Ok:
    EXEC outros_levantam(qt)
EndGI
```

É de notar também que se declaram as mesmas variáveis ao nível do contro-lador. Tal deve-se por um lado a razões de coerência com as definições ante-riormente apresentadas para as cláusulas dos GI e, por outro, pelo facto de que, não se encontrando de momento implementados quaisquer mecanismos de opti-mização, entre usar uma variável do controlador ou invocar mais do que uma vez uma função da aplicação, no caso a função *pode_levantar?(qt)*, a solução pa-rece ser mais correcta[10].

Finalmente, uma terceira possibilidade de interacção poderá ser considera-da e especificada. Esta vai basear-se na utilização de uma ligação complementar e não explícita entre o guião, que especifica o comportamento da IU para dada operação, e o respectivo *Descritor da Apresentação* (DA), que especifica uma pos-sível apresentação associada. Naturalmente que, sendo a junção textual de um com o outro a mais simples e clara solução de especificação, tal significa que, em situações mais complexas, esta deve ser mesmo a regra, ou seja, especificar comportamento juntamente com a

[10] Em particular tendo em atenção a implementação das comunicações entre módulos realizada no sistema GAMA-X que, por se apresentar apenas no capítulo seguinte, não deve ser ainda considerada.

especificação da apresentação. Usaremos tal regra nesta situação de especificação da interacção.

Vamos agora considerar que os valores das quantias mínima e máxima pos-síveis de serem levantadas são, não apenas passadas à camada de apresentação para informação do utilizador, mas por esta usadas para garantir que "apenas" um valor em tal gama é devolvido ao controlador do diálogo e à aplicação. Para que tal seja possível, é então necessário especificar, ao nível do controlador, que tais valores devem ser passados à camada de apresentação, conforme o guião,

---

DefGI *Outros_levantamentos2*
Declarations
TYPE *Synth*
VAR-CTRL *qt: Quantia; qtmin: Quantia; qtmáx: Quantia;*
VAR-UI *qtmin: Quantia; qtmáx: Quantia*
Behaviour
INIT *qtmin = 0*
CONTEXT *SaldoAtm?() > 0*
EVSEQ *input(qt)*
TRANS
Start: true => qtmáx := max_levantam?()
Cancel: Ok:
EXEC *outros_levantam(qt)*
EndGI

---

que especifica que, seleccionada a operação, deve de imediato ser atribuído à variável do controlador e da apresentação o valor máximo que pode ser levanta-do, e que após a introdução da quantia pelo utilizador a operação da camada computacional é automaticamente invocada, não existindo qualquer teste a esta quantia introduzida, assumindo-se portanto a sua correcção. Para que tal seja possível é necessário que a nível da camada mais próxima do nível léxico, ou se-ja do I/O, tais garantias possam ser dadas. Neste caso, por exemplo, se a cama-da de I/O, ou de apresentação, puder garantir a correcção das entradas, muito se pode simplificar ao nível das outras, designadamente, do controlador e da ca-mada computacional.

Para tal, e relativamente a esta operação, ter-se-ia que definir um DA ade-quado à garantia de tais condições, conforme, por exemplo o DA[11],

---

DefDA *Outros_levantamentos2*

---

[11] A sintaxe e semântica dos Descritores da Apresentação (DA) serão introduzidos apenas no capítulo seguinte, enquanto descritores de implementações de apresentações.

Declarations
TYPE *DB* *; de Caixa de Diálogo*
NAME *"Outros Levantamentos"*
VARS *qt (TYPE : Scale,*
VALUES : (qtmin TO qtmáx)
NAME : "Limites do Levantamento !")
EndDA

que especifica que o valor a introduzir deve ser controlado por um objecto inter-activo do tipo *Scale*, com valores limitados entre *qtmin* e *qtmáx*. Garantindo a ca-mada de apresentação que apenas um valor entre tais limites é introduzido ou seleccionado, o controlador do diálogo pode, de forma segura, invocar a opera-ção da camada computacional, ou seja, sem preocupações de validação, i.é., de controlo de erros.

Resta finalmente apresentar o guião da operação *levantamento*, que segundo as preocupações de assistência semântica anteriormente apresentadas, deverá apresentar ao utilizador no menu de quantias seleccionáveis apenas aquelas que garantem a totalidade da operação de *levantamento*. Assim, o guião deve es-pecificar que apenas são apresentadas ao utilizador as quantias que podem ser correctamente seleccionáveis por este, em função do estado do sistema. Tal im- plica, como se afirmou anteriormente, a utilização de uma função capaz de cal- cular as quantias acessíveis ao utilizador no contexto da selecção.

Esta função deverá ter em consideração a informação do cartão e as notas disponíveis na máquina, e determinar o conjunto de quantias que poderão ser disponibilizadas ao utilizador de forma interactiva, tendo ainda em atenção que tais quantias fazem parte de um conjunto bem definido de quantias, associadas ao tipo QuantiaSel.

A função *Seleccionáveis?*, que se apresenta no guião, realiza tal cálculo. De-terminado o conjunto de valores seleccionáveis, este é colocado numa variável da apresentação para que seja posteriormente mostrado ao utilizador. O descri-tor da apresentação associado à operação garante que tal é realizado.

DefGI *Levantamentos*
Declarations
TYPE *Synth*
VAR-CTRL *qt: Quantia*
VAR-UI *qtSels : QuantiaSel-set*

Behaviour
CONTEXT *SaldoAtm?() > 0*

EvSeq *input(qt)*
Trans
Start: true => qtSels := Seleccionáveis?()
Cancel: Ok:
Exec *levantamento(qt)*
EndGI

O descritor da apresentação usa um *OptionMenu*, contendo os valores que foram anteriormente calculados como seleccionáveis, para a leitura da variável *qt* que será o argumento da operação de levantamento.

DefDA *Levantamentos*
Declarations
Type *DB*
Name *"Levantamento"*
Vars *qt (Type : OptionMenu,*
Values : qtSels
Name : "Quantias")
EndDA

Esta última especificação mostra como a interacção efectiva resulta da con-jugação da informação contida nos GI e nos DA. Sempre que "feedforward" deva ser fornecido, então o GI deve realizar o cálculo da informação a apresentar e tal informação colocada na área de dados da apresentação, de modo a que o DA associado a possa utilizar na apresentação ou introdução especificada.

Resta agora, para terminar a especificação do controlador, definir o guião principal que determina a sequência de eventos e respectivas operações no sis-tema. O acesso ao sistema ATM é dado após a introdução do cartão do utiliza-dor e da introdução correcta do respectivo PIN. Após estas duas operações obri-gatórias, o sistema apresenta um menu de operações invocáveis pelo utilizador, que por cada introdução do cartão pode apenas realizar uma transacção. Após a transacção seleccionada o cartão é automaticamente retirado da máquina. Em qualquer momento, a transacção seleccionada pode ser anulada.

Comecemos por especificar o menu de operações a ser apresentado pelo sis-tema após a introdução do cartão.

DefGI *Menu1_ATM*
Declarations
Type *Decision*
External *MConsultas, MLevantamentos, Pagamentos, MOutras_Op*

Behaviour
EvSEQ *MConsultas + MLevantamentos + Pagamentos + MOutras_Op*
EndGI

Este menu principal dá acesso ao menu de consultas, ao menu de levanta-mentos e ao menu de outras operações, bem como à operação de pagamentos. Os guiões que especificam estes sub-menus são apresentados a seguir.

DefGI *MConsultas*
Declarations
TYPE *Decision*
EXTERNAL *Saldo, Movimentos*
Behaviour
EvSEQ *Saldo + Movimentos*
EndGI

DefGI *MLevantamentos*
Declarations
TYPE *Decision*
EXTERNAL *Outros_levantamentos2, Levantamentos*
Behaviour
EvSEQ *Outros_levantamentos2 + Levantamentos*
EndGI

DefGI *MOutras_Op*
Declarations
TYPE *Decision*
EXTERNAL ...
Behaviour
EvSEQ ...
EndGI

```
DefGI Anular
Declarations
        TYPE Synth
Behaviour
        EXEC  CloseApl()
EndGI
```

Teremos finalmente o guião que especifica o comportamento total do siste-ma ATM,

```
DefGI ATM
Declarations
        TYPE
Behaviour
        EvSEQ Cartão_in . Leitura_pin . ( Menu1_ATM | | Anular )
EndGI
```

Para completar a especificação da camada interactiva, resta apenas especi-ficar a informação da camada computacional necessária à IU, designadamente os identificadores de tipos e os tipos associados, bem como possíveis invariantes sobre estes tipos. Com esta informação, a camada interactiva vai poder garantir que os valores de dado tipo que sejam lidos são de imediato validados. As decla-rações **DefVar** e **DefType** que se apresentam na secção seguinte cumprem este objectivo.

É ainda de salientar, caso tal não tenha resultado claro do exemplo seguido, que as pré-condições podem ainda fornecer informação sobre a sequência ideal de introdução dos argumentos de uma dada operação (*dependências*). Por exem-plo, dada a especificação de uma operação *OP(a, b)* na qual o argumento *a* deve satisfazer a uma condição qualquer explícita na pré-condição, e assumindo-se que a assistência semântica não foi neste caso implementada ao nível da IU, é então necessário garantir que uma ordem lógica de introdução destes dados seja estabelecida. Neste caso, se o argumento *a* está condicionado e o argumento *b* está livre, então a ordem lógica da introdução destes argumentos será uma or-dem sequencial, isto é, *a* **.** *b* (*a* seguido de *b*). Por outro lado, caso na pré-condi-ção da operação surjam predicados sobre os argumentos ligados entre si pelo operador lógico *andif* (cf. por exemplo na expressão *p1(a)* ö *p2(b)*), tal pode ser igualmente entendido como uma dependência entre os argumentos, já que ape-nas fará sentido ler e validar o segundo argumento caso o valor lido para o pri-meiro satisfaça ao predicado correspondente.

Estas considerações sobre o grau de dependência entre os argumentos de uma operação ou entre os valores dos campos de uma “form”, são indicações

importantes para a escrita da especificação apresentada nas cláusulas EVSEQ e TRANS dos guiões.

Deve igualmente ser notado que a informação que facilmente se recolhe da especificação formal da camada computacional, pode igualmente ser recolhida caso o método de desenvolvimento empregue seja um outro qualquer. De facto, em muitos métodos informais ou semi-formais de desenvolvimento, noções se-melhantes às noções de pré-condição, invariante de dados, etc., são usadas, ainda que, na maioria dos casos, documentadas em linguagem natural. Tal não impede, no entanto, que a partir da interpretação destas, o mesmo processo de especificação da IU seja utilizado, fazendo corresponder as definições informais à notação dos GI e dos DA, se não para a geração da IU usando o GAMA-X, pelo menos para que documentação sobre a construção da IU possa ser associada à restante documentação de projecto, o que, em geral, não acontece.

FASE II.II : Protótipo da IU.

A fase seguinte da construção da camada interactiva consiste na geração de um protótipo da IU usando o gerador GAMA-X, capaz não só de apoiar a produção das especificações anteriores como de, a partir destas, construir uma IU que automaticamente se liga ao protótipo da aplicação. Pode também ser construído um protótipo de forma manual, desde que seguindo o modelo de interacção su-gerido, o MASS, e interpretando as especificações produzidas na fase anterior.

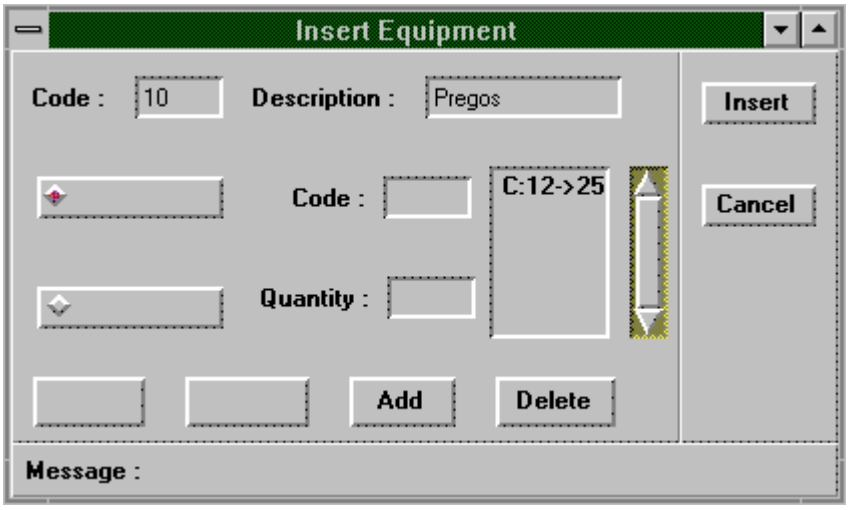

Fig. 7.3 - Caixa de Diálogo numa Aplicação com MASS.

Sendo aqui, naturalmente, colocada a tónica nas vantagens da utilização do sistema que gera semi-automaticamente as IU, pretende-se com a figura

7.3 mostrar que, mesmo seguindo a vertente manual, resultados de qualidade po-dem ser conseguidos[12].

A utilização do sistema GAMA-X permite que, a partir das descrições da IU que se apresentaram anteriormente, uma IU seja gerada de modo semi-automá-tico, e semi-automático apenas porque questões finais de apresentação devem ainda ser consideradas numa implementação final. Por outro lado, o sistema apresenta a vantagem de permitir que a ligação ao protótipo da camada compu-tacional possa ser feita numa fase ainda muito inicial do projecto. Assim sendo, a IU pode ser iterada e melhorada, em função dos requisitos dos utilizadores, em isolamento das questões relacionadas com a efectiva implementação da ca-mada computacional, que pode prosseguir em relativo[13] isolamento das ques-tões ligadas à concepção da IU.

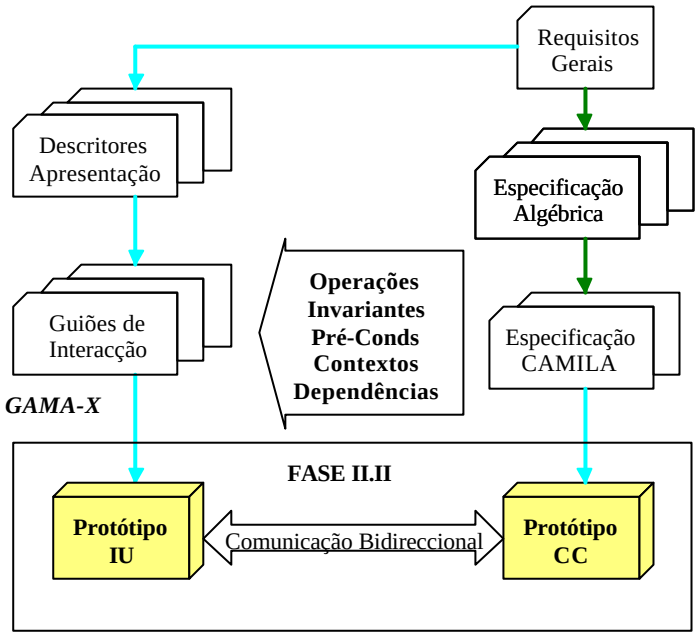

Fig. 7.4 - Ligação entre Protótipos.

A figura 7.4 ilustra o resultado esperado desta fase, na qual os protótipos desenvolvidos se ligam, permitindo a simulação do comportamento final do

[12] A figura 7.3 representa uma "caixa de diálogo" para uma operação de inserção de equipamentos, extraída de um ecrã de um dos projectos do 5º ano da LMCC de um sistema interactivo de Planeamento de Produção, especificado em CAMILA e posteriormente implementado em Oracle e Access, e cuja IU foi desenvolvida manualmente seguindo o MASS. Note-se a assistência semântica dada pela "ListPane", e a sensibilidade ao contexto evidente nos botões disponibilizados ou não. De notar ainda que a IU desenvolvida para comunicação com o protótipo CAMILA é, conforme os exemplos disponíveis, indistinguível da IU que se liga aos SGBD usados, em concreto Oracle e Access.

[13] Tenha-se em atenção a necessidade de criação de algumas funções auxiliares adicionais.

sis-tema interactivo numa fase ainda bastante inicial do seu efectivo desenvolvi-mento ou implementação.

FASE III : Implementação da CC e da IU.

Esta fase corresponde ao desenvolvimento final da aplicação, ou seja, da passa-gem rigorosa do protótipo da camada computacional para uma efectiva imple-mentação. O processo de reificação de dados e refinamento das operações é rea-lizado usando *refinamento transformacional* [Oliveira 92] [Oliveira 93]. Em resul-tado, é construído o código final da aplicação, sendo também definidas as duas funções que relacionam os objectos abstractos com os objectos concretos. No sentido objectos concretos-objectos abstractos uma *função de abstracção* terá sido calculada. No sentido inverso a chamada *função de reificação* deverá ser escolhida[14], permitindo que um valor abstracto possa ser feito corresponder ao respectivo valor concreto que o representa.

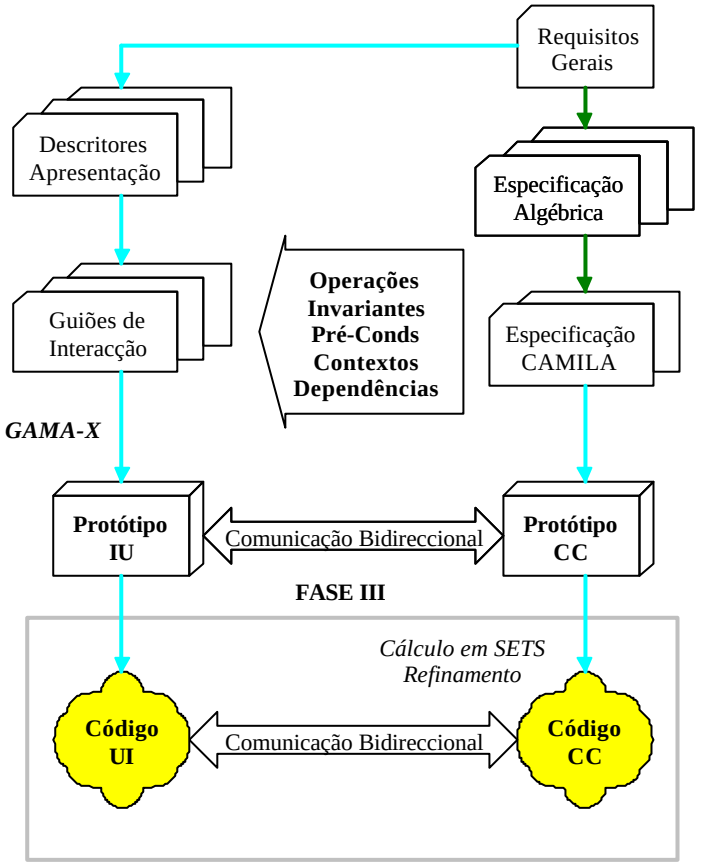

Fig. 7.5 - Implementação da IU e da CC.

[14] Como a *função de abstracção* raramente é injectiva, terá que ser escolhida uma dentre as suas várias funções inversas.

A figura 7.5 procura ilustrar esta fase da construção do sistema interactivo, discutindo-se em consequência as alterações a realizar no protótipo da IU para que este passe a ser também uma IU definitiva.

Encontrado o código definitivo da aplicação, torna-se agora necessário ana-lisar como tal código pode ser ligado ou ao código do protótipo da IU, ou ao có-digo da implementação definitiva desta. Duas soluções podem ser encontradas para realizar esta ligação que dará origem ao sistema interactivo final.

A primeira solução passa por se considerar que ao protótipo desenvolvido para a IU são fornecidas as duas funções, de *reificação* e de *abstracção*, resultantes do refinamento da especificação da camada computacional. Com estas funções, o protótipo da IU pode ser de imediato considerado código definitivo, já que passa a ser capaz de comunicar com a camada computacional, aplicando as funções fornecidas na conversão de representações.

A segunda solução, possivelmente mais eficiente, passa por se ajustar o Mo-delo da Aplicação da IU à implementação da camada computacional. Experiên-cias já realizadas[15] demonstraram que a simples alteração de um ficheiro de de-finições ao nível do código C, que representa parte da API entre a IU e a camada computacional, é suficiente para que o protótipo da IU passe a ser a IU efectiva ligada à camada computacional definitiva.

Ficam indiciadas duas ligações possíveis entre o protótipo gerado da IU, ou até o seu código final, e o código final da camada computacional. Qualquer que seja então a ligação final entre estas duas camadas, deverá no entanto ficar claro da análise feita que tal não será um obstáculo à implementação final do sistema interactivo, e que, qualquer que seja a solução encontrada, ela não será nunca complexa em termos de implementação, podendo no entanto conduzir a diferentes resultados em termos de "performance", afirmação que se baseia em experimentação já realizada em diferentes ambientes e com diferentes graus de exigência em termos de qualidade final. Em definitivo, teremos que a implemen-tação final da camada computacional não afecta o "look" e o "feel" da IU, dado que estes foram concebidos, desde o início, num contexto que emula o contexto final da implementação. Finalmente, o "currying" do estado (cf. nota de rodapé nº 9 da página 238) elimina a necessidade, e a correspondente sobrecarga, de ter que se implementar no sistema uma função de abstracção que traduz valores concretos do estado para valores abstractos do mesmo, viabilizando assim todo o processo.

7.3.- Síntese do Método.

[15] conforme a experiência referida anteriormente e realizada em múltiplos ambientes e usando diferentes ferramentas.

O método apresentado baseia-se, em primeiro lugar, na recolha de informações a partir das especificações formais da camada computacional. Usando estas in-formações, os requisitos de interacção encontrados e os princípios associados ao modelo MASS, é possível sistematizar a concepção e especificação da IU.

A especificação formal vai produzir a assinatura do sistema, i.é., o seu con-junto básico de operações, os objectos e seus tipos, os axiomas, os invariantes de tipos de dados e as pré-condições das operações. Filtrando deste conjunto de operações aquelas que não serão acessíveis a partir da IU, é sobre as restantes e respectivos objectos que se deve centrar a concepção e recolha da informação.

Sintetizam-se de seguida as relações que se estabeleceram entre componen-tes da especificação formal e componentes da especificação da IU.

? **Assinatura.**

A assinatura contém informação que é fundamental para a criação de uma parte do Modelo da Aplicação. O modelo da aplicação, além de outras funções, é a interface entre a linguagem da IU e a linguagem interna da aplicação. Assim, deverá conter não apenas os identificadores das operações da camada computa-cional acessíveis da camada interactiva como também a sua aridade, ou seja, os tipos dos argumentos de entrada e do resultado. À assinatura resultante da es-pecificação formal devem juntar-se algumas funções auxiliares resultantes da concepção da IU, e que são exclusivamente funções de acesso à camada aplica-tiva para consulta de informação.

? **Invariantes de Dados.**

Os invariantes de dados são predicados que se aplicam às variáveis de um dado tipo para verificação da correcção dos seus valores. Representam portanto restrições à gama de valores que as variáveis de um dado tipo podem assumir.

Em consequência, sempre que os argumentos de uma operação têm que ser recolhidos a partir do utilizador, devem-lhes ser aplicadas as funções que repre-sentam os seus invariantes para verificação da sua correcção. Tendo a entrada de dados a ver com a estrutura do Modelo de Apresentação, ou seja, com o nível de I/O, os invariantes de dados, sob a forma de funções invocáveis que aplica-das a um valor lido retornam o valor verdadeiro ou falso, devem ser disponibili-zadas ao Modelo da Apresentação. Assim, e em geral, uma leitura terá a seguin-te expressão

$$\text{input(x)} \;?\; \text{let x = ppickone(\{ x | x ? T ; inv-T(x) \})}$$

ou seja, sempre que um valor **x** do tipo **T** deve ser lido, então tal consiste na se-lecção de um valor do tipo **T** que deve pertencer à classe dos valores que

sa-tisfazem o invariante de dados desse tipo. Não se tecem aqui considerações tem-porais ou outras relacionadas com a validação posterior do valor lido ou com o fornecimento prévio dos valores correctos. Note-se no entanto que a adopção de um dado Modelo de Interacção significa uma interpretação particular destas ex-pressões matemáticas que são atemporais. Assim, e em conclusão, invariantes são funcionalmente vistos como validadores de dados ou como filtros para a de-terminação dos valores correctos entre os quais deve ser feita a selecção.

Os invariantes de dados podem ter ainda informação relevante para a escri-ta da cláusula EVSEQ dos guiões. Por exemplo, se relativamente a um objecto com estrutura, seja ele um registo, existirem restrições envolvendo vários dos seus campos, então estas serão descritas no invariante, e uma cláusula de EVSEQ adequada deverá ser associada à leitura de um valor de tal tipo estrutu-rado de dados. O mesmo tipo de consideração pode ser feita caso no invariante sejam utilizados operadores *andif* que, tal como foi anteriormente observado, se associam a uma ordem temporal de obtenção de dados.

A sequência de eventos especificada é uma representação, em termos de eventos, da navegação correcta que deve ser realizada sobre a estrutura, assu-midas as restrições que podem ser apenas dinamicamente consideradas. Um tí-pico exemplo é uma qualquer “form” (ou registo) na qual a existência de campos opcionais e de campos dependentes faz com que a navegação correcta sobre a sua estrutura seja dependente da própria história da navegação realizada. Note--se aqui a diferença entre este tipo de problemas e a navegação estática que era suficiente nos arquétipos.

? **Pré-Condições.**

Tal como os invariantes, as pré-condições são dos elementos da especifica-ção da camada computacional que trazem a si associada mais informação útil para a concepção da camada interactiva. No entanto, o tipo de informação dada pela pré-condição deve ser interpretada, enquanto informação de interacção, de três modos diferentes, gerando diferentes cláusulas de especificação da IU:

a) *Predicados sobre o Estado.*

Todos os predicados ou sub-predicados introduzidos na pré-condição de da-da operação que se aplicam exclusivamente ao *estado* da aplicação, devem ser interpretados como restrições contextuais à aplicação da operação, pelo que se a operação vai ser interactivamente invocável, a adequação do contexto à sua invocação deve ser testada na IU. Em resultado, todos estes predicados devem ser transpostos para a cláusula CONTEXT do respectivo guião.

b) *Predicados sobre um Argumento.*

Todos os predicados ou sub-predicados introduzidos na pré-condição de da-da operação que envolvem individualmente os argumentos da operação impõem sobre estes restrições e condições a serem verificadas. Se o argumento é asso-ciado pelo predicado ao estado, então tal significa que o valor do argumento de-pende, de alguma forma, do contexto actual. Por exemplo, uma pré-condição típica

Cod ? dom(BaseDados)

que significa que para uma dada operação o código introduzido deve fazer parte dos códigos registados na base de dados, é uma restrição com dependência con-textual. Nestes casos, o cálculo do contexto correcto e sua apresentação ao uti-lizador garante a pré-condição. Porém, tal esforço deve ser realizado ao nível da IU, pelo que esta informação deve ser especificada. Como se verificou no exem-plo apresentado, esta informação é introduzida quer no GI quer do DA. Outra alternativa, menos assistencial, consiste em aceitar o evento *input(Cod)* e, só de-pois, usando a cláusula EXCEP da cláusula TRANS, tratar a situação de erro. A decisão entre uma solução e a outra é uma decisão de concepção, que pode ou não obedecer aos princípios do modelo sugerido, mas que continua a ter trata-mento na especificação dos GI.

c) *Predicados sobre vários Argumentos.*

Quando na pré-condição surgem predicados relacionando entre si os argu-mentos da operação, tal significa que uma relação de dependência entre os seus valores foi estabelecida, necessitando agora de uma interpretação, do ponto de vista interactivo, temporal linear. Por exemplo, considere-se que numa pré-con-dição de uma dada operação com dois argumentos A e B, surge a relação A < B.

Do ponto de vista da interacção, várias hipóteses podem ser consideradas. Por exemplo, poder-se-ia considerar a introdução paralela dos dois valores e, só no final realizar a verificação da propriedade. Por outro lado, poderia ser consi-derada a possibilidade de introdução sequencial de A seguido de B, de tal forma que, após introduzido o valor de A, apenas valores de B tais que B > A poderiam ser considerados válidos, ou vice-versa. Isto significa que, por um lado, em si-tuações deste tipo, uma cláusula EVSEQ deve ser encontrada em função do tipo de relação entre argumentos, e de seguida, em função de decisões de concep-ção, um DA em conformidade ou não com o MASS completará a descrição. Tendo em consideração o exemplo, poder-se-ia ter no GI da operação a sequência,

EVSEQ *input(A)* **.** *input(B)*

complementada por um descritor no qual a leitura de B fosse limitada à leitura de valores tais que,

$$input(B) = ppickone(\{ B \mid B\ ?\ T : B > A \})$$

onde T representa o tipo de A e de B, sobre o qual está definida a ordem >.

Caso a EVSEQ fosse decidida de forma diferente, então o ajuste respectivo no DA teria que ser feito. Assim, e em conclusão, relações entre argumentos terão sempre implicações quanto à respectiva cláusula EVSEQ, tendo sempre em con-sideração que no DA respectivo se deve considerar a especificação complemen-tar que é a consequente lógica da especificação anterior dentro do modelo que foi considerado.

7.4.- Sumário.

Neste capítulo, a partir de um exemplo e seguindo um conjunto de regras espe-cíficas, apresentou-se um método rigoroso (porque baseado em construções for-malizadas) quer para a especificação da camada computacional quer para a es-pecificação da camada interactiva. O método encontra-se sistematizado segun-do um conjunto de fases e regras para a construção da IU a partir da especifica-ção da camada computacional.

Não sendo automático, o método baseia-se na inferência de informação so-bre a camada interactiva a partir de uma especificação formal por modelos da camada computacional. Tal implica, apesar de tudo, a adopção de decisões de concepção, advogada, ainda que não necessariamente, seguindo o modelo MASS.

Para além de todas as regras de sistematização apresentadas, um processo básico pode ser identificado, consistindo da interpretação temporal, crucial para a construção da IU, de especificações atemporais da camada computacional.

De facto, e em última análise, todas as regras propostas e todas as decisões a tomar têm tais características, ou seja, visam inferir da especificação formal e atemporal da camada computacional, uma ordem temporal lógica para as ac-ções interactivas. Esta ordem temporal a inferir visa, segundo o MASS, garantir uma maior usabilidade das IU, pela diminuição das distâncias articulatória e observacional (cognitivas) e pela eliminação de potenciais erros.

O mecanismo de “feedforward” semântico sugerido é um exemplo de como tais objectivos podem ser satisfeitos em simultâneo, sem que tal envolva uma sobrecarga exagerada para a tecnologia subjacente.