CAPÍTULO 8

PROTOTIPAGEM
E
GERAÇÃO AUTOMÁTICA

O SISTEMA GAMA-X

ÍNDICE

8.1.- Introdução.

A arquitectura "software" designada por GAMA, *Gerador semi-Automático de Modo Assistido*, GAMA-X na versão actual sobre X-Windows [Creissac 93] [Creissac e Martins 94], é o sistema de desenvolvimento de IU (SDIU) que foi construído com o objectivo de suportar a construção sistemática e a geração semi-automá-tica de protótipos da IU, segundo o método proposto. Para tal, o sistema inclui um conjunto de ferramentas de apoio.

Por razões que se prendem com a necessidade de gerarmos IU satisfazendo a requisitos de interacção tais como assistência semântica e sensibilidade ao contexto, várias especializações tiveram que ser adicionadas às funções básicas dos componentes típicos de um SGIU. Outro aspecto muito importante do dese-nho e da apresentação de um SGIU, dado muito ter a ver com a separação e mo-dularidade dos componentes, é o seu *modelo de dados*, ou seja, a forma como os dados da IU e da aplicação se distribuem pelos vários componentes do sistema interactivo.

Por outro lado, a IU deve poder comunicar com a camada computacional em diferentes fases do desenvolvimento desta, desde a fase em que a aplicação é apenas uma especificação executável (ou protótipo), até à fase em que a mesma, segundo o método referido na secção 3.5, atinge o seu código final.

Naturalmente que, assim sendo, dois problemas diferentes são colocados:

1) *Construir de forma semi-automática, a partir da especificação formal da aplicação, o módulo MIU (MÓDULO INTERFACE UTILIZADOR) respectivo, ou seja uma IU obedecendo ao modelo MASS, tendo por ferramentas auxiliares de geração as do módulo MGI (MÓDULO GERADOR DE INTERFACES), e por camada computacional um protótipo da aplicação resultante da directa execução da sua especificação formal; A IU gerada deverá ser, quanto à sua apresentação mas principalmente quanto ao seu comportamento interactivo, idêntica à IU definitiva;*

2) *Possuindo já uma IU que, por evolução independente usando a infor-mação da especificação, deverá atingir o seu estado definitivo, coloca-se de seguida o problema de colocar tal IU em comunicação com uma apli-cação que deve ser observacionalmente equivalente, do ponto de vista do utilizador e da própria IU, à anterior, mas que, igualmente por evolução, se encontra agora completamente desenvolvida e codificada numa outra lin-guagem que não a de especificação.*

São os problemas referentes a estas questões que se tratam neste capítulo, onde se apresenta a arquitectura do GAMA-X, a funcionalidade de

cada um dos seus componentes e ainda a forma de distribuição da informação.

8.2.- GAMA-X: Arquitectura Funcional.

Apresentam-se nesta secção os dois módulos principais do GAMA-X, designada- mente, o módulo de "runtime" do GAMA-X, o MIU (*Módulo Interface Utilizador*), onde é implementada a IU, e o módulo MGI (*Módulo Gerador de Interfaces*), que representa o conjunto de ferramentas auxiliares construídas para a geração se-mi-automática dos diferentes componentes do MIU. Ainda que não se pretenda que o sistema GAMA-X seja um SGIU, o módulo MGI corresponde, de facto, a um usual UIDE (*"User Interface Development Environment"*), enquanto que o módulo MIU é o usual UIS (*"User Interface System"*).

A arquitectura funcional abstracta do MIU segue o modelo de Arch-Slinky [Bass et al. 91], reconhecendo-se os seguintes componentes, com as seguintes funções típicas:

- *Camada Computacional* (CC): a aplicação ou núcleo computacional;
- *Modelo da Aplicação* (MApl): este componente agrega dados da aplicação em estrutras de mais alto nível, e serve de interface entre o controlador do diálogo e a aplicação;
- *Controlador do Diálogo* (CD): este componente medeia entre a aplicação e a apresentação, estabelecendo as necessárias correspondências. Deve ga-rantir coerência, por exemplo, entre o estado das variáveis da aplicação e as diferentes *views* que estas possam ter, controlando ainda a sequência das interacções.
- *Modelo da Apresentação* (MApr): contém os *logical interaction objects*, que são descrições abstractas da apresentação de vários objectos da aplicação que, por serem abstractas, permitem independência dos "toolkits";
- *Toolkit de Interacção* (TI): esta é a componente que gere a interacção ao ní-vel físico entre o utilizador e o computador, controlando os dispositivos físicos de I/O e a apresentação.

A figura 8.1, de sentido apenas estrutural, representa as ligações existentes entre componentes e módulos, que serão posteriormente detalhadas em termos funcionais.

A figura torna evidente a constituição interna do MIU, contendo o módulo MA que implementa o Modelo da Aplicação (MApl), o Controlador do Diálogo (CD) e o módulo AP do Modelo da Apresentação (MApr). Note-se, desde já, que o "toolkit" (TI) e a camada computacional (CC) se encontram, naturalmente, fora do MIU ainda que em comunicação com este.

Por outro lado, o MGI deve ser considerado um módulo auxiliar, sendo tem-poralmente um antecessor do MIU e para a descrição do qual incluirá utilitários importantes tais como editores/compiladores dos Guiões de Interacção, edito-res/compiladores de Descritores da Apresentação, e editores/compiladores de descrições do Modelo da Aplicação.

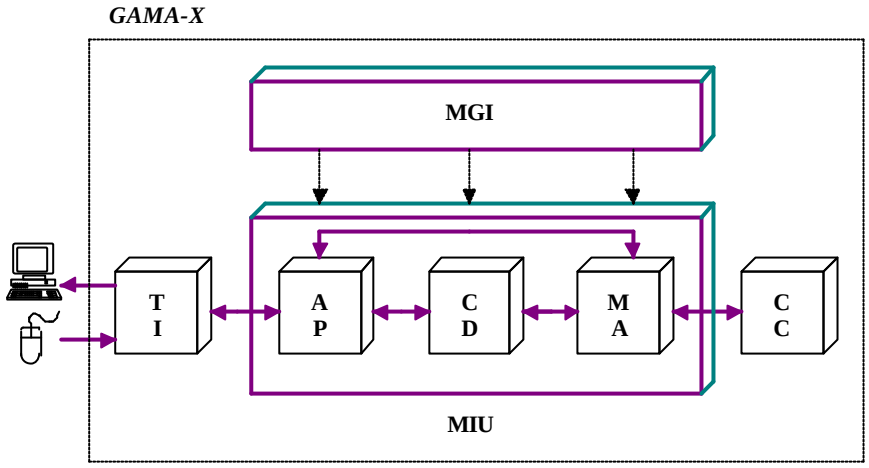

Fig. 8.1 - O Sistema GAMA-X.

Nas secções seguintes serão analisadas as capacidades funcionais de cada módulo, como se interligam, e a forma como são construídos. Uma descrição mais pormenorizada de decisões e detalhes da implementação efectiva do siste-ma pode ser encontrada em [Creissac 93].

8.3.- O Módulo MIU.

Como se referiu atrás, para a geração semi-automática de IU em modo MASS, característica que deixaremos a partir de agora de sublinhar, o ponto de partida quanto a informação sobre a aplicação a desenvolver, é uma especificação for-mal na linguagem CAMILA e, quanto à modelação, um protótipo executável na linguagem funcional *xmetoo* [TCG 90] resultante da compilação da especificação em CAMILA.

Todo o Sistema Interactivo deve ser construído a partir destes elementos. A geração da IU usando as ferramentas auxiliares desenvolvidas, deve basear-se nas informações recolhidas ao nível da especificação formal, segundo regras que se salientaram no capítulo anterior onde o método a aplicar foi apresentado. A IU que, segundo o método, assim vier a ser construída, deve, numa primeira fa-se, "dialogar" com um modelo da aplicação existente sob a forma de um protóti-po escrito na linguagem funcional *xmetoo*.

Desconhecendo-se nesta fase qual a "linguagem-objecto" na qual a aplica-ção virá a ser implementada, mas procurando-se alguma coerência

com o méto-do mais geral em vista (cf. secção 3.5), achou-se por bem desenvolver notações de alto nível para a descrição independente dos vários componentes do MIU (cf. por exemplo os GI para a descrição do CD). Estas descrições são depois, atra-vés do suporte dado pelo MGI, compiladas para *xmetoo,* garantindo-se compati-bilidade imediata de comunicação com o protótipo da aplicação, por mais com-plexo que este seja. O Modelo da Aplicação (MA) assume o papel de "adaptador" entre a estrutura interna do MIU e o código da aplicação. Assim, enquanto a aplicação for representada pelo protótipo desenvolvido, um Modelo da Aplicação (MA) ajustado deverá estar presente. Quando se pretender realizar a ligação da IU desenvolvida ao código final da aplicação, bastará realizar a adaptação de tal componente.

Por outro lado, e visando garantir independência e modularidade, cada um dos componentes do MIU será um processo *xmetoo* independente, comunicando estes processos entre si através de protocolos específicos que foram definidos.

A separação entre componentes e a sua relativa independência é conseguida através da adopção de um modelo de dados distribuído. Assim, passamos a ter três representações diferentes para os dados no sistema interactivo. Uma é a re-presentação *semântica,* ou seja, como estes são, num dado instante, representa- dos na camada computacional. Outra, é a representação dos dados ao nível do controlador do diálogo, a que chamaremos representação *sintáctica.* Finalmente, teremos ainda a representação dos valores no Modelo da Apresentação, podendo ter uma de várias possíveis apresentações (*views*). No entanto, como veremos, a informação semântica relevante para que a IU tenha um comportamento perma-nentemente sensível ao contexto e com assistência semântica, é colocada à dis-posição dos componentes que dela necessitam, em particular o Modelo da Apre-sentação e o Controlador do Diálogo.

Vejamos de seguida como cada componente é construído e qual a sua fun-cionalidade específica.

8.3.1.- Controlador do Diálogo.

O Controlador do Diálogo é construído a partir dos Guiões de Interacção que es-pecificam o seu comportamento, estado interno e ligações a estabelecer com os outros componentes. Todo o comportamento do CD resulta da execução das Redes de Petri geradas a partir dos GI especificados, para o que um animador de Redes de Petri foi desenvolvido. O ciclo genérico de execução do CD consiste na recepção de eventos do módulo de apresentação, no processamento de tais eventos conforme a especificação contida nos GI, gerando possivelmente outros eventos que são comunicados aos outros módulos.

O CD trabalha sobre os TAD que resultaram da especificação da aplicação, ou seja, usando tuplos, conjuntos, sequências, funções finitas e

relações biná-rias. Tornou-se assim necessário, dadas as particularidades estruturais destes objectos, desenvolver mecanismos pré-definidos para a sua leitura e apresenta-ção. Foram pois criados guiões especiais para síntese de valores estruturados de cada um destes tipos, nomeadamente, *TupSynth, SetSynth, ListSynth, FFSynth, RelSynth* [Creissac 93].

A leitura dos valores destes objectos é realizada sob controlo do CD e não, como todas as outras, feita sob o controlo do Modelo da Apresentação que reali-za o tratamento léxico. Eventos correspondentes à execução destes GI especiais são comunicados ao CD que assim realiza o controlo de tal execução.

Ao longo da execução do diálogo, e sempre que um guião é activado, uma nova *instância* do mesmo é criada, pelo que é com estas diversas instâncias que o CD comunica, sendo responsável pelo controlo do espaço de variáveis destas várias instâncias e pela execução das respectivas Redes de Petri.

Tendo já sido apresentado detalhadamente o funcionamento dos guiões de interacção que especificam o comportamento do Controlador do Diálogo, apre-sentam-se agora as diferentes mensagens trocadas entre os outros componentes do MIU e o CD no sentido da invocação da funcionalidade deste.

Apresenta-se em primeiro lugar a comunicação entre o Modelo da Apresen-tação (AP) e o CD, ou seja, as mensagens trocadas no sentido de AP para CD, para invocação dos serviços deste:

- A criação de novas instâncias de guiões é realizada a pedido do AP, que envia ao CD uma mensagem para a criação de uma nova instância de um guião, tendo por "pai", caso seja SUBGI ou EXTERNAL, um outro guião identificado, cf. *CeateMsg(GIpai, GInovo).* Em resultado, o CD comunica ao AP o identificador da instância criada, para que o MA se lhe possa, de fu-turo, referir;
- A activação de uma instância já criada, cf. *OpenMsg(InstId),* é outro servi-ço invocado pelo AP;
- A desactivação de uma instância é um serviço invocado através da men-sagem *KillMsg(InstId);*
- O AP comunica ainda ao CD os eventos de *start, end* e *cancel,* através das mensagens *StartMsg, EndMsg* e *CancelMsg,* no contexto de uma dada instância, invocando os serviços do CD para a respectiva actualização do estado da Rede de Petri desta;
- A mensagem *CmdMsg(InstId, CmdId),* corresponde à selecção de um dado comando a nível da apresentação que é comunicada ao CD, que deve ac-tualizar a instância identificada;
- A mensagem *SetValMsg(InstId, Value),* permite transmitir ao CD um valor cuja leitura havia sido solicitada ao AP;

Em síntese, no sentido AP-CD podem ser identificadas as seguintes mensagens:

Mens-AP-CD = CreateMsg | OpenMsg | StartMsg | EndMsg | ComMsg | CancelMsg | KillMsg | SetValMsg ;

Apresentam-se agora as mensagens enviadas pelo CD para o AP, não só as geradas em resposta às anteriores, como também as que vão garantir sincro-nização de estados entre os módulos.

- A mensagem *InstMsg(InstId),* corresponde à comunicação que o CD faz ao AP em resposta à mensagem de criação de uma nova instância;

- Sempre que uma mudança de estado é realizada ao nível do CD, este deve comunicar ao AP quais os eventos ou comandos que passam a ser aceitáveis em tal contexto. Para tal efeito são usadas as mensagens que indicam que uma dada instância apenas pode aceitar, ou deixa de acei-tar, certos eventos, cf. as mensagens *EnableMsg(InstId, TransId_list)* e *DisableMsg(InstId, TransId_list);*

- Sempre que uma instrução de *output* é executada pelo CD, uma mensagem *OutMsg(InstId, String)* é enviada ao AP para que tal valor seja apresentado;

- Sempre que uma variável da apresentação vê o seu valor alterado, o CD envia a mensagem *ShowMsg(InstId, VarId, Value)* para o AP, por forma a que a apresentação possa reflectir de imediato tal alteração;

- Quando uma Rede de Petri terminou a sua execução, ou seja, um guião chegou ao fim com sucesso, o AP é notificado pelo CD que lhe envia a mensagem *StopMsg(InstId, OpcValue)*;

- Se, por qualquer razão, uma Rede de Petri terminou anormalmente a sua execução, por exemplo por ter ocorrido um evento assíncrono de cancela-mento, o CD notifica o AP com a mensagem *Abort(InstId)*;

- Finalmente, e porque o CD e o AP funcionam entre si de modo síncrono, uma mensagem *GoMsg()* é enviada pelo CD ao AP logo que o processa-mento da mensagem anteriormente recebida for realizado;

Em síntese, no sentido CD-AP podem ser identificadas as seguintes mensa-

gens:

Mens-CD-AP = InstMsg | OutMsg | EnableMsg | DisableMsg | StopMsg | GoMsg | ShowMsg | AbortMsg

O CD estabelece também comunicação com o Modelo da Aplicação (MA). O protocolo é neste caso mais simples dado que apenas será necessário invocar a execução de funções da aplicação, consultar o valor de determinada variável, saber o tipo de um dado identificador ou terminar a execução da aplicação.

- Para determinar o valor de uma dada variável da aplicação, o CD envia a mensagem *GetVarMsg(InstId, VarId*) ao MA, respondendo este, após a cha-mada à respectiva rotina da aplicação, com o valor desejado;

- Para determinar a definição de um dado identificador de tipo, o CD envia a mensagem *GetTypeMsg(InstId, TypeId*) ao MA, respondendo este, com o tipo desejado;

- Para a execução de um dado comando da aplicação, o CD envia ao MA a mensagem *CallMsg(InstId, Opr, ArgValue-list, Ret*), contendo o contexto de execução de tal comando, o nome da operação a executar, a lista de valores dos seus argumentos, e uma variável lógica que indica se o CD vai ficar ou não à espera da devolução de um resultado, ou seja, se sin-croniza ou não com a operação despoletada. Porque as operações invoca-das, sejam elas de consulta ou de alteração do estado da aplicação, têm implicações ao nível do diálogo e da apresentação, o usual é que a IU se mantenha síncrona com as alterações de estado da aplicação.

- O CD tem ainda a missão de informar a aplicação do fim da sessão, para o que pode usar a mensagem *HaltMsg(integer)*, contendo um parâmetro de tipo inteiro de carácter geral, utilizável ou não;

Em resumo, no sentido CD-MA podem ser identificadas as seguintes men-sagens:

Mens-CD-MA = GetVarMsg | CallMsg | GetTypeMsg | HaltMsg ;

Vejamos finalmente as mensagens trocadas no sentido MA-CD, ou seja, as mensagens que correspondem a respostas às mensagens enviadas pelo CD ao MA. Estas podem ser descritas simplesmente indicando o seu tipo. De facto, te-remos apenas as mensagens,

Mens-MA-CD = SetValMsg | DefTypeMsg ;

? Para enviar ao CD os valores solicitados por *GetVarMsg(InstId, VarId*) ou por *CallMsg(InstId, Opr, ArgValue-list, Ret*), o MA utiliza uma única mensagem, *SetValMsg(InstId, Value)*;

? Para enviar ao CD os valores solicitados por *GetTypeMsg(InstId, TypeId*), o MA utiliza uma única mensagem, *DefTypeMsg(InstId, TypeDef)*;

Para além destes serviços prestados ou invocados pelo Controlador do Diá-logo, é também necessário que ele execute as acções descritas em INIT e TRANS. Para tal, um interpretador foi desenvolvido permitindo de momento a expressão de instruções como atribuições, instruções condicionais, instruções repetitivas e instruções de saída.

O CD contém também o interpretador e animador de Redes de Petri. Este in-terpretador aceita uma Rede de Petri modelada da forma seguinte:

```
PetriNet ::     B: Places
                Ev: Events
                Fi: Flow
                Fo: Flow
                Cb: Places
                Ce: Places;
Events = Event-set;
Event = StartEvent | CancelEvent | EndEvent | CmdEvent | NullEvent;
StartEvent, CancelEvent, EndEvent :: Dref: EvId;
CmdEvent :: Dref: CmdId;
NullEvent :: Dref: STR;
Flow = Event ?  Places;
Places = PlaceId-set;
Conditions = PlaceId ?  Bool;       marcação da rede (externa)
```

onde B é o conjunto de todos os lugares da rede, Ev é o conjunto de todas as transições, Fi e Fo os fluxos de entrada e saída das transições definidas, e Cb e Ce as marcações inicial e final da rede. Note-se que esta modelação da Rede de Petri não contém a marcação actual. Tal deve-se ao facto de que, sendo cada re-de partilhada por todas as instâncias dos GI que lhes deram origem, a marcação actual, ou estado da rede, é guardada nas correspondentes instâncias dos GI. A figura 8.6 apresenta, a título de exemplo, o código das Redes de Petri automati-camente gerado pela "ferramenta" de edição estruturada de GI, podendo obser-var-se a obediência de tal código ao modelo anteriormente apresentado.

Apresenta-se na figura 8.2 a estrutura de ficheiros necessária à criação au-tomática do CD final para uma dada aplicação.

O ficheiro *cdmain* contém o núcleo do controlador. O ficheiro *cdcom* contém todas as funções de comunicação. Em *pnet* encontra-se o código do animador das Redes de Petri. Por razões de optimização da implementação final, o código correspondente às cláusulas TRANS, CONTEXT e EXCEP não foi incluído no código das redes mas antes colocado sob tratamento pelo CD, invocando o animador das redes as necessárias operações do CD, garantindo este um comportamento observacionalmente equivalente ao que foi especificado. Destas decisões resul-tam vantagens de implementação dado que o animador das Redes de Petri é simplificado, bem como a geração automática destas, sendo ainda diminuída alguma redundância de informação já que, em vez de termos as mesmas condi-ções e instruções repetidas ao longo de diversos lugares da rede, elas são cen-tralizadas nas próprias definições dos guiões.

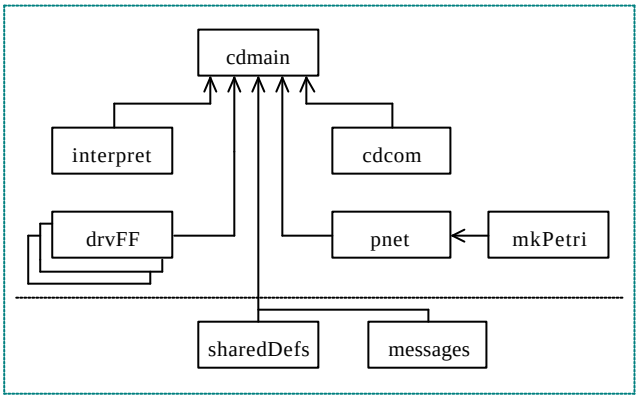

Fig. 8.2. - Ficheiros para a geração do CD.

No ficheiro *interpret* encontra-se o código do interpretador de instruções. O ficheiro *mkPetri* pode não existir caso as Redes sejam geradas automaticamente. Os módulos de *drvFF* a *drvTup* são os módulos pré-existentes para o tratamento dos objectos que se associam aos modelos matemáticos pré-definidos a realizar pelo CD.

Os ficheiros *sharedDefs* e *messages* contêm definições que passarão a estar acessíveis aos outros componentes do MIU.

8.3.2.- Modelo da Apresentação.

Na figura 8.1, onde se apresentou a arquitectura do GAMA-X, procurou tornar-se claro o facto de que um "toolkit" para a implementação dos aspectos relaciona-dos com a apresentação e com o "input" é utilizado pelo sistema. Na versão actual, é utilizado um *frontend* em X11R4, desenvolvido recorrendo à utilização de Tcl/Tk [Ousterhout 91].

O Modelo da Apresentação implementará o módulo de interface entre o MIU e o "toolkit", sendo o módulo responsável pela gestão do aspecto gráfico da IU e pela tradução dos diversos eventos de e para o utilizador. O AP contém também as descrições internas, abstractas, da apresentação de cada menu ou objecto da aplicação com ligação à IU, permitindo descrições "por omissão" e a definição de diferentes apresentações ou *views* para um dado objecto, ainda que de forma relativamente simples. Estas descrições abstractas, são especificadas usando *Descritores da Apresentação.* Por serem descrições lógicas, elas garantem algum grau de independência relativamente aos "toolkits".

Por exemplo, os menus a apresentar são descritos por uma estrutura sim-ples contendo o identificador do menu, atributos de posição e cor, o conjunto de opções seleccionáveis e acções associadas, e a cor da opção seleccionada. Por exemplo, para descrever um menu com seis operações seleccionáveis, às quais temos associados guiões, poderíamos ter o seguinte descritor:

```
DefDA Menu
        TYPE MENU
        NAME "Principal"
        SEL  "black"
        OPTIONS
            GInitStack      "Inicializa Stack"
            GPush       "Insere no Topo"
            GTop            "Consulta Topo"
            GPop            "Remove Topo"
            GList       "Lista Stack"
            GClose      "Fim"
EndDA
```

Os descritores de menus são modelados pelo tuplo

```
DMenu ::    Name: STR
            Pos: [STR]          ; posição
            Color: [STR]        ; cor do menu
            Sel: STR            ; cor da selecção
            Opcs: Option-list   ; lista de opções
Option ::   GI: EvId
            Nm: STR
```

Descritores do tipo DB, de "dialogue box", destinam-se a descrever as apre-sentações que se associam a cada um dos GI, ainda que estas possam ser recor-rentes dado que alguns argumentos do comando a sintetizar podem

ser de ti-pos estruturados para os quais “dialogue boxes” são igualmente necessárias. To-das estas situações são contempladas na actual implementação.

```
DefDA ConsProduct
        TYPE DB
        NAME "Consulta Produto"
        VARS cprod ( TYPE: ItSelf,
                     NAME: "Product Code:",
                     COLOR: "white")
        ...
EndDA
```

Para a descrição gráfica de variáveis, foram criados mecanismos de apresen-tação pré-definidos. *ItSelf* associa, "por omissão", uma apresentação para uma variável correspondente ao seu tipo. *RadioBoxes*, *OptionMenus* e *Scales* podem ser também declaradas como apresentações para certas variáveis. Porém, como se referiu antes, caso as variáveis pertençam a tipos estruturados, a sua leitura e apresentação encontra-se definida em guiões particulares pré-definidos.

Os descritores do tipo DB obedecem ao modelo seguinte:

```
DDB ::      Name: STR
            Pos: [STR]                  ; posição
            Color: [STR]                ; cor
            Vars: VarId ? ViewType      ; apresentação das variáveis
            GIs: EvId ? DBGIView        ; apresentações dos sub-DB
            Cmds: CmdId ? LexDef        ; léxico dos comandos
Option ::   GI: EvId
            Nm: STR
ViewType = PreDefView | GISym           ; tipo de apresentação
PreDefView = ItSelf | RadioBox | OptionMenu | Scale;
```

A apresentação é construída recorrendo aos *widgets* que implementam cada um dos objectos abstractos de interacção anteriormente referidos. Dada a exis-tência de uma semântica intrínseca aos *widgets* fornecidos pelos “toolkits”[1], que acaba por corresponder a uma tomada do controlo do diálogo por parte destes à revelia do próprio controlador, foi necessário criar *widgets*

[1] por vezes referida como "micro-diálogo".

especiais para alguns dos objectos abstractos de interacção. Por exemplo, a cada guião para síntese de um comando associa-se uma "janela" contendo uma "caixa de diálogo" para a leitura dos seus argumentos. Porém, não só porque a leitura destes argumentos deve ser realizada por uma ordem determinada gerida pelo controlador, mas também porque, de forma recorrente, a síntese de um argumento estruturado pode implicar a apresentação de uma outra "caixa de diálogo", tal como se refe-riu anteriormente, os comportamentos dos *widgets* disponibilizados pelo "tool-kit" têm que ser adaptados, tendo-se optado por criar *widgets* específicos. Por um lado, a cada passo do diálogo devem devolver o controlo do diálogo ao CD, que, em função da instância de guião em questão, da especificação do diálogo para tal guião e do contexto actual de interacção, determina um novo contexto e, possivelmente, a respectiva alteração na apresentação.

Por questões de optimização, o AP mantém igualmente uma estrutura que representa a sua informação sobre as instâncias dos guiões para as quais gere o "input/output". Esta estrutura é especificada a seguir.

```
InstInfo ::   Father: [InstId]                  /* instância "pai" */
              EvMask: EvId-set                  /* eventos a cada momento aceites */
              Var: [VarId]                      /* variável-resultado opcional */
              Vals: VarId ? [Value]             /* valores das variáveis */
              Children: EvId ? InstId           /* instâncias descendentes */
              Active: Bool;                     /* activa ou não ? */
```

Por cada instância criada, esta estrutura contém a informação sobre o seu posicionamento na hierarquia de guiões, indicando a instância "pai", o conjun-to de eventos (*event mask*) a cada momento reconhecidos (cf. informação actua-lizada fornecida pelo CD usando as mensagens anteriormente apresentadas), os valores actuais das variáveis, as sub-instâncias desta, e, ainda, se a instância se encontra ou não activa.

Em circunstâncias normais de funcionamento, é exactamente sobre o con-junto das instâncias activas que é realizada a selecção do utilizador. Dentro da instância seleccionada, o utilizador poderá passar a interactuar com todos os objectos que geram eventos válidos nesse contexto.

Para além do funcionamento em MASS, o MIU permite que o utilizador possa em qualquer momento invocar o *modo-comando*, passando, neste caso a ser in-diferentes as descrições de apresentação construídas, ainda que algumas vali-dações de dados permaneçam. A comutação entre o *modo-comando* e o modo MASS faz-se automaticamente sempre que o comando introduzido tem um qual-quer erro ou foi introduzido não completamente instanciado. No entanto, mesmo em *modo-comando*, se um argumento está definido como *Scale*, *RadioBox* ou *Option Menu*, o seu valor é verificado relativamente aos valores válidos em tal estado.

8.3.3.- Modelo da Aplicação.

O MA é descrito de forma bastante simples atendendo à informação que deve conter, designadamente, para além das operações disponíveis, os tipos das vari-áveis usadas, os seus invariantes e, caso se acedam a variáveis da aplicação, as operações de consulta que lhes são associadas.

Estas declarações obedecem aos esquemas-tipo que se apresentam de segui-da. O esquema para declaração de tipos de variáveis e seus invariantes é basea-do em duas cláusulas apresentadas no esquema genérico exemplificativo.

DefType
idvar1 = type *; declaração simples*
idvar2 = type WITH invariant *; declaração com invariante*
EndType

Nesta declaração genérica apresenta-se a regra que permite indicar o tipo de dados de uma dada variável de interesse para a IU (cf. *idvar1*). A segunda regra não só associa um identificador de tipo de dados à variável como também a fun-ção que determina se um valor desse tipo é válido ou não (*invariante*).

Para se poderem atribuir valores actualizados a certas variáveis do controla-dor, variáveis que foram declaradas como associadas a variáveis da aplicação, é necessário que no esquema de definição de variáveis se associe a cada uma destas uma função para obtenção de tais valores, conforme o esquema geral,

DefVar
idvar1 = nomefunc1
idvar2 = nomefunc2
EndType

A manutenção de um modelo da aplicação estruturalmente simples, de facto apenas declarações sintácticas das operações invocáveis, de associações entre identificadores de tipos e funções que determinam se um determinado valor é correcto (*invariantes*) e associações entre variáveis e as funções que determinam, por consulta da aplicação, os seus actuais valores, permite que a construção da IU seja relativamente independente da implementação final da camada compu-tacional. De facto, e conforme algumas experiências relatadas no capítulo 7, um simples ficheiro da IU, em particular do MA, deverá ser adaptado se a imple-mentação deixar de ser realizada na linguagem A e passar a ser feita na lingua-gem B. Esta

alteração acaba sempre por acontecer desde que a implementação não se faça na linguagem de prototipagem, o que será a regra e não a excepção.

8.3.4.- Modelo de Dados do MIU.

Um dos problemas mais complexos no desenho de sistemas para a construção de IU, é considerar uma distribuição da informação que possa garantir separa-ção e modularidade nos sistemas construídos.

Carlsen, em [Carlsen 92], distingue três diferentes modelos de dados. O modelo designado *simbólico-directo*, mais usado em sistemas antigos, especifica que apenas informação simbólica, tal como ícones ou menus, deve ser manipu-lada pela componente UIS. A informação *directa*, ou seja, com valor semântico, é manipulada exclusivamente pela aplicação, ainda que com ligação ao sistema gráfico, que a deve receber e apresentar. O modelo designado de *partilha* propõe que toda a informação da camada computacional relevante para a IU seja colo-cada numa estrutura de dados partilhada. Ainda que possa auxiliar quanto à implementação de *retorno semântico*, esta solução restringe a camada computa-cional quanto à modelação dos dados, sendo ainda um problema em sistemas distribuídos onde uma separação lógica e física é importante. O modelo deno-minado *distribuído* sugere que estes dados sejam distribuídos de tal modo que a camada computacional é a única a ter acesso aos seus dados, implementando no entanto *views* sobre tal informação sob a forma de um conjunto de *funções de acesso*. Quando esta *view* muda, comunicação explícita entre a camada com-putacional e a IU deve ser realizada. Assim, a camada computacional não tem que se submeter a um modelo de dados particular, e, por outro lado, o *retorno semântico* necessário pode ser facilmente conseguido através da concepção cui-dadosa de um protocolo de comunicação. O modelo favorece ainda soluções dis-tribuídas. A figura 8.3 esquematiza o modelo *distribuído* de dados.

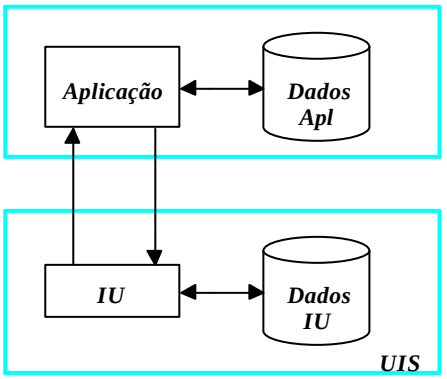

Fig. 8.3 - Modelo Distribuído de Dados em SGIU.

Conforme se pode verificar pelas anteriores descrições dos componentes do módulo MIU, parece evidente que no sistema GAMA-X o modelo de dados adopta-do é o *modelo distribuído*. Ainda que o GAMA-X não tenha peocupações imediatas em garantir suporte a IU por manipulação directa, de grande exigência em ter-mos de *retorno semântico*, é no entanto evidente que a implementação do MASS coloca dificuldades semelhantes no GAMA-X.

Ainda que alguma sobrecarga[2] de comunicação possa ser reconhecida, pare-ce, no entanto, equilibrado o protocolo de comunicação encontrado no GAMA-X, dado permitir a satisfação de todos os requisitos da IU recorrendo a um conjun-to simples e bem identificado de mensagens.

8.4.- O Módulo MGI.

O módulo MGI deveria, idealmente, gerar automaticamente todas as descrições necessárias à geração automática dos componentes do MIU, ou seja da IU. O es-quema ideal seria então poder extrair de forma automática, a partir da especifi-cação CAMILA da aplicação, as necessárias descrições para a geração dos com-ponentes. Naturalmente que alguns destes objectivos, colocados desta forma, só podem ser, em termos responsáveis e realistas, considerados inatingíveis.

Por exemplo, os Descritores da Apresentação podem ser construídos de vá-rias formas, utilizando várias ferramentas auxiliares específicas, não sendo po-rém inferíveis de qualquer especificação formal. Mesmo os que definem menus devem ser construídos em resultado de um processo de concepção da apresen-tação, seguindo até regras próprias, e não a partir da especificação. No entanto, é possível, definida uma notação para a descrição de apresentações possivel-mente dependente do "toolkit" a usar, construir ferramentas que auxiliem a sua descrição.

Outra dificuldade consiste, tal como se viu no capítulo anterior, em analisar a especificação e dela extrair, automaticamente, operações e suas funcionalida-des, importantes para o Modelo da Aplicação, e pré-condições, contendo infor-mação importante para diferentes cláusulas dos GI. Finalmente, a definição de tipos e invariantes realizados em CAMILA pode oferecer informação para a ex-tracção automática das definições de *DefType*, informação necessária ao Modelo da Apresentação. Porém, para que tal seja facilitado, algumas simples mas labo-riosas alterações devem ser realizadas ao nível da sintaxe do próprio CAMILA.

Dado que muitas destas necessidades foram constatadas em função do pró-prio objectivo proposto de procurar gerar automaticamente IU a partir de especi-ficações formais por modelos em CAMILA, o MGI não tem, neste momento, qual-quer ligação automática à especificação formal da camada computacional. Isso deve-se a uma limitação da actual sintaxe do

[2] *overhead*.

prototipador CAMILA que, dada a sua actual estrutura sintáctica, não facilita a inferência automática da informa-ção necessária à geração da respectiva IU. Por tal razão nos temos vindo a refe-rir à *geração semi-automática* de IU a partir de especificações formais.

Foram no entanto desenvolvidas ferramentas que, independentemente da especificação formal, mas contando com informação retirada desta de forma sis-tematizada pelo construtor da IU, permitem a construção automática de alguns dos componentes do MIU, em particular o Controlador do Diálogo, especificado usando os GI.

A figura 8.4 apresenta a estrutura interna actual do MGI, distinguindo-se os passos que são tomados de forma automática por serem já suportados por "fer-ramentas" com tal propósito desenvolvidas, e os que, quer por não justificarem de momento automatização, ou apresentarem ainda certo grau de impossibilida-de de automatização, são ainda realizados de forma manual, ainda que com al-gum grau de sistematização.

Editores estruturados semelhantes ao que foi desenvolvido para os Guiões de Interacção constituirão as "ferramentas" que permitirão a geração automática dos Modelos da Apresentação e da Aplicação. A necessidade de se realizar um estudo adequado que permita que tais geradores possam ser parametrizados, por exemplo relativamente ao “toolkit” a usar ou à linguagem da aplicação, têm adiado a realização da sua implementação definitiva.

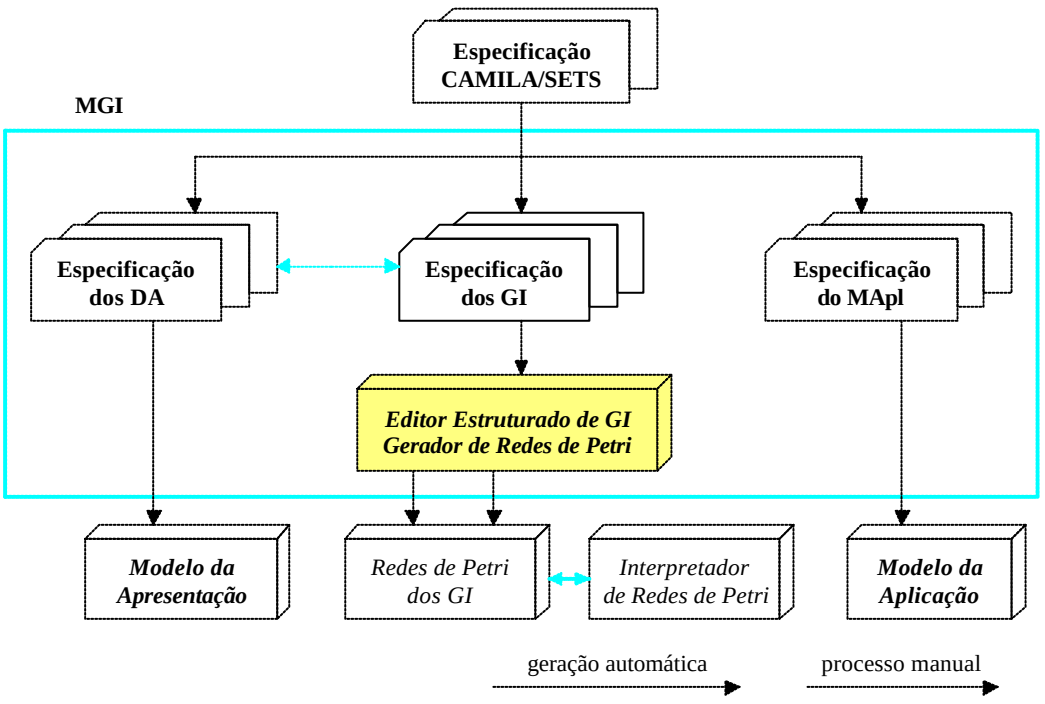

Fig. 8.4 - O módulo MGI.

A ferramenta principal consiste num editor estruturado de GI que, além de permitir a correcta construção destes, possibilita a geração automática das res-pectivas Redes de Petri que representam o controlo da sua execução [Rocha 93]. A geração automática das Redes de Petri permite que, usando o interpretador desenvolvido no âmbito do GAMA-X, se possam animar estas especificações. Esta ligação é ilustrada na figura 8.4.

Atendendo à importância desta ferramenta no sistema GAMA-X, dado permi-tir automatizar uma das tarefas mais importantes - a geração automática do controlador do diálogo -, apresentam-se a seguir duas "janelas" da "ferramenta", representativas das duas fases mais importantes do processo, designadamente, a edição dos GI e a geração automática do código das Redes de Petri.

A figura 8.5 representa a fase de edição estruturada dos GI de um sistema, tendo já sido definido o guião correspondente ao menu principal, sendo visível na cláusula EVSEQ a expressão de comportamento global do sistema. Este guião inclui um subguião *Fim* ainda não completamente definido, procurando-se com o exemplo ilustrar a flexibilidade concedida pelo editor pelo facto da edição ser incremental.

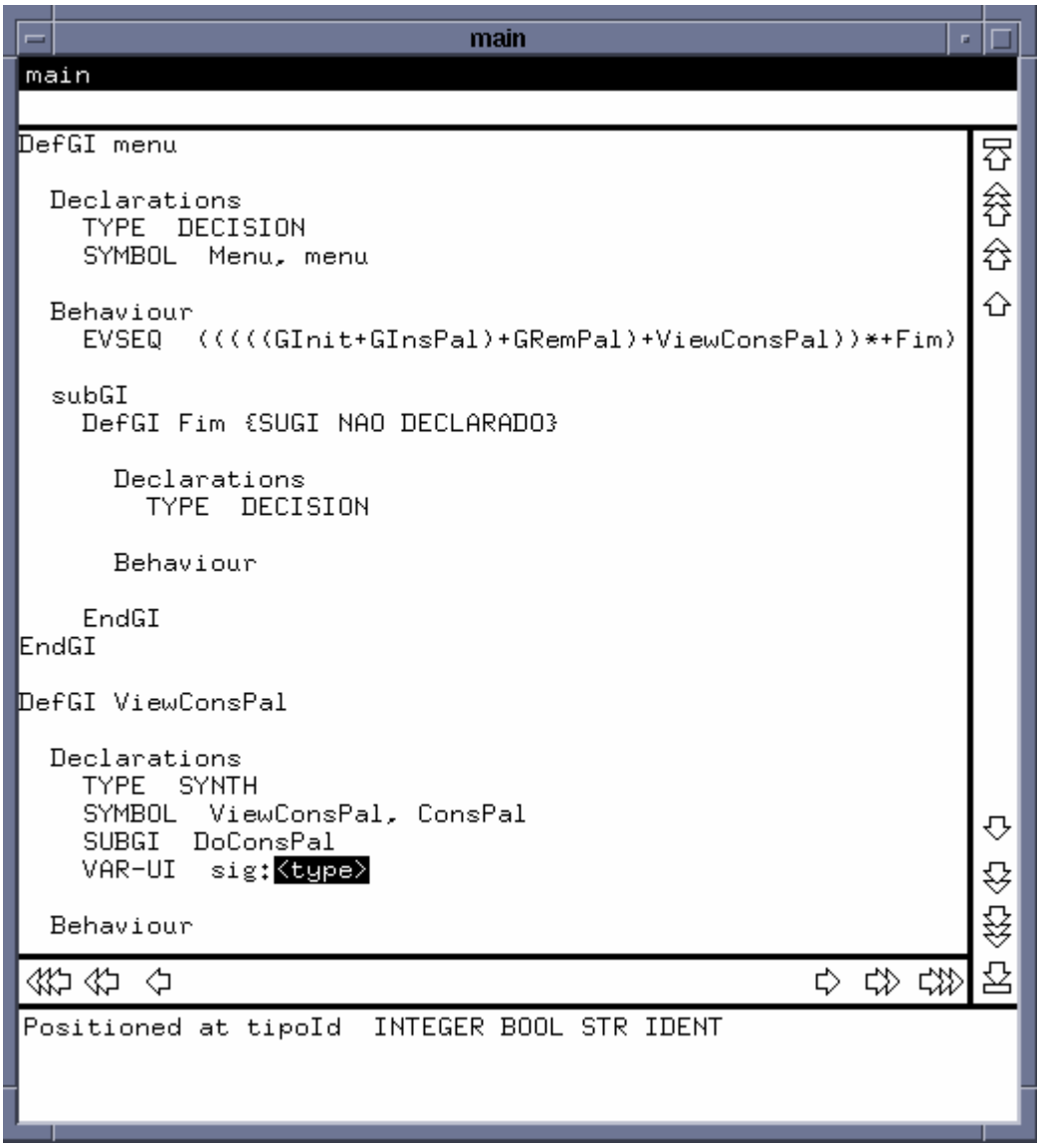

Fig. 8.5.- Editor Estruturado de Guiões.

Note-se a assistência estrutural sintáctica e de semântica estática fornecida ao utilizador pelo editor estruturado[3], que em cada momento da introdução de valores informa o utilizador dos tipos dos objectos a introduzir ou mesmo do conjunto de opções sintácticas possíveis. No exemplo, ao ter que introduzir o tipo correcto de uma dada variável, o utilizador é informado pelo editor dos tipos aceitáveis para a definição da mesma no contexto (cf. Integer, Bool, Str e Ident).

Na figura 8.6 é apresentada a "janela" que corresponde à visualização do có-digo automaticamente gerado das Redes de Petri, correspondente aos vários GI introduzidos. Este código é posteriormente gravado num ficheiro

[3] gerado automaticamente pelo SYNTHESIZER GENERATOR [Reps e Teitelbaum 89].

que será lido e constituirá uma das fontes de informação do CD e do interpretador das redes.

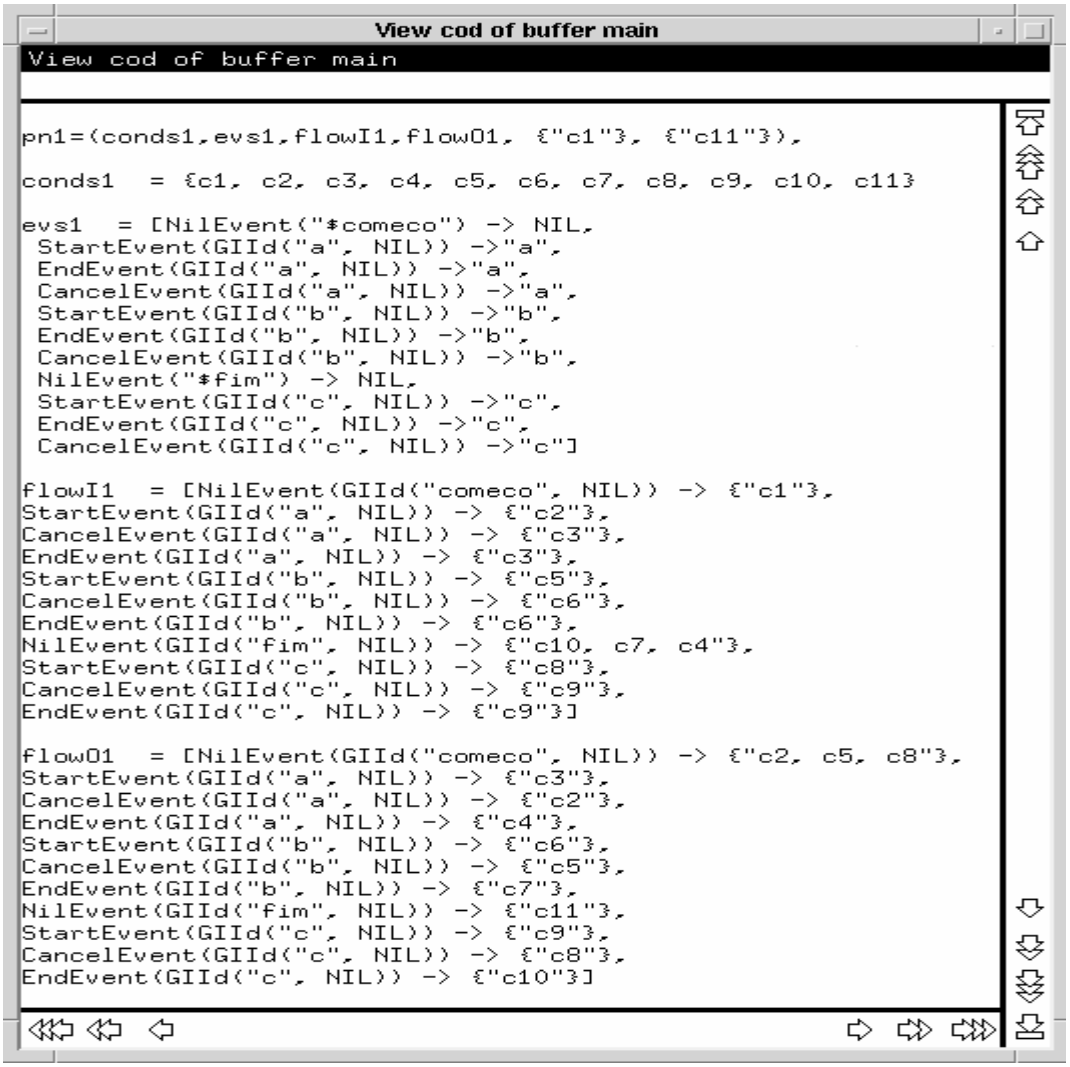

Fig. 8.6.- Geração Automática das Redes de Petri.

É de notar a conformidade da estrutura do código automaticamente gerado com a estrutura abstracta do código de uma Rede de Petri tal como especificada anteriormente. Tal semelhança não é casual e resulta de se ter considerado que, tendo que ser encontrada uma linguagem para a descrição das Redes de Petri e tendo a possibilidade de as representar em CAMILA, então poderíamos ter simul-taneamente descrições rigorosas e imediatamente executáveis se necessário.

8.5.- Geração Semi-Automática com GAMA-X: Exemplo.

Procura-se nesta secção dar apenas uma simples ideia das reais capacidades de geração semi-automática do GAMA-X, apresentando dois exemplos da IU

gerada a partir da especificação do sistema ATM que, a título de exemplo, se realizou no capítulo 7.

A "janela" que se apresenta na figura 8.7 corresponde à "janela" principal, automaticamente gerada, do sistema ATM tal como especificado, sendo de sali-entar a *sensibilidade ao contexto* evidente da inacessibilidade, em tal contexto, à operação de *levantamentos*. Note-se ainda a existência de um botão identificado como *Change Mode* que permitirá comutar entre o *modo-comando* e o MASS.

Finalmente, note-se a informação sobre a *view* activa.

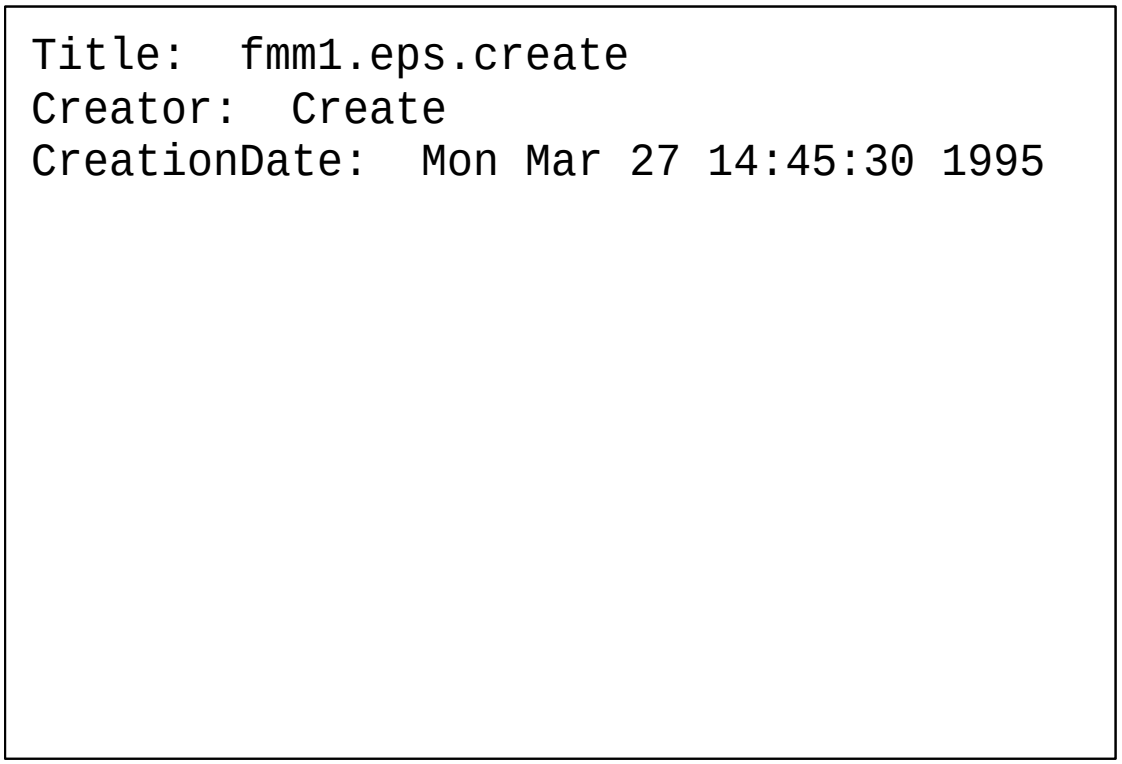

Fig. 8.7 - ATM: Menu Principal.

A "janela" que se apresenta na figura 8.8 corresponde à "janela" associada à operação de *Levantamentos*, quando a mesma se encontrar disponível, tal como automaticamente gerada pelo GAMA-X partindo da respectiva especificação. É de salientar não só a apresentação do caminho interactivo que conduziu à selecção escolhida pelo utilizador, como também a assistência semântica tornada eviden-te pela apresentação de um menu de quantias seleccionáveis, no qual algumas das possibilidades se encontram não acessíveis (supostamente porque a máqui-na ATM possui, naquele instante, apenas notas de dois mil escudos).

Esta janela ilustra também os problemas anteriormente referidos de que um controlo local das instâncias dos guiões deve ser mantido no MA,

permitindo que alguma parte do diálogo seja realizada sem interferência ou invocação do CD. Quando se torna necessário reflectir contexto ao nível da IU, como aconte-ce no exemplo com as quantias seleccionáveis, é o CD que é responsável quer por tal cálculo quer pela comunicação de tais resultados ao MA. No exemplo, o valor do argumento *Quantia* deverá ser escolhido dentre um conjunto de quan-tias elegíveis em tal contexto como seleccionáveis. Sendo essas, e só essas, apre-sentadas na lista fornecida ao utilizador, tal garante de imediato a correcção se-mântica deste argumento da operação.

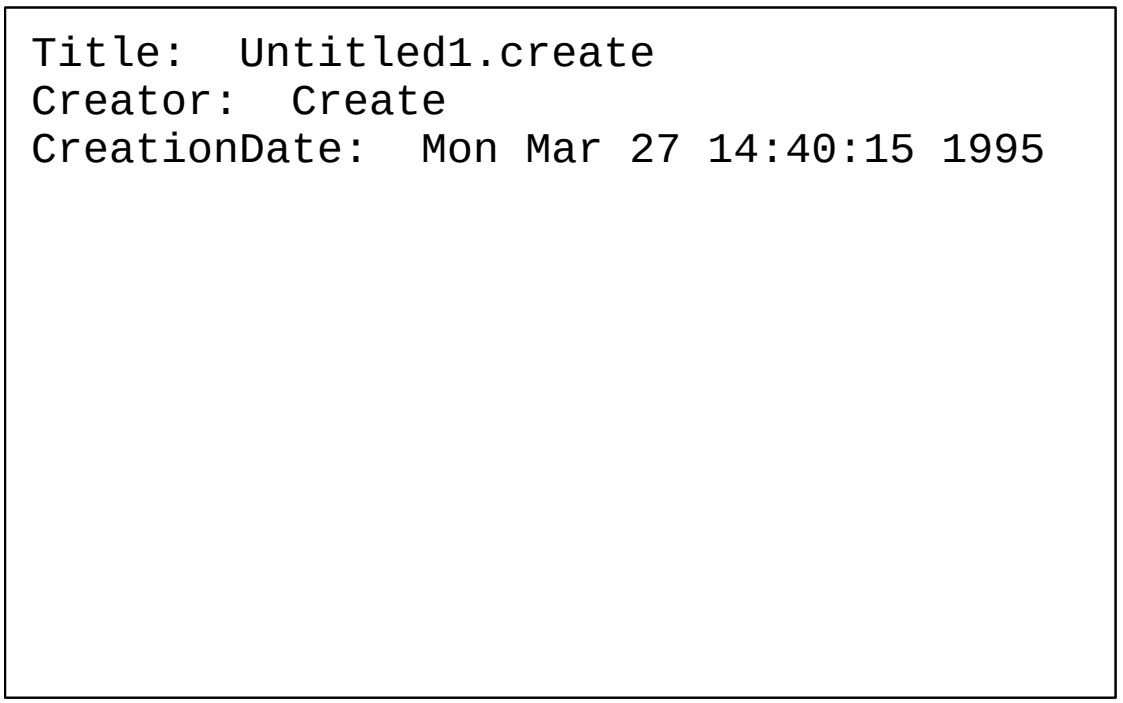

Fig. 8.8 - Sensibilidade ao Contexto na Selecção de um Valor.

O objectivo da apresentação destas duas "janelas" da IU gerada pelo sistema GAMA-X para uma aplicação anteriormente concebida e especificada, é, simples-mente, por um lado completar tal exemplo e, por outro, dar uma ideia de como as propriedades e características de interacção referidas são de facto reflectidas no comportamento final da IU gerada.

A qualidade do “look” final da IU, que não do seu comportamento, é, como se compreende, dependente do “toolkit” utilizado, podendo ser depois ajustada a partir da apresentação gerada.

8.6.- Comparação com alguns SGIU.

Ainda que o objectivo final do trabalho apresentado nesta tese não seja a cons-trução de mais um SGIU, sendo antes o GAMA-X uma "ferramenta" de apoio a to-do o método proposto, parece todavia interessante que se ponham em confronto as soluções encontradas no GAMA-X com as soluções propostas

noutros SGIU, com este comparáveis dado possuirem algumas características comuns, quer na forma quer nos objectivos.

Não se comparam arquitecturas completas dado que, para que tal pudesse ter significado, os objectivos deveriam ser semelhantes. Assim, comparam-se principalmente notações, abordagens e resultados.

MIKE.

MIKE (*Menu Interaction Kontrol Environment*) [Olsen 86] é um SGIU que, tal como o GAMA, tem por metáfora de interacção de alto-nível a *síntese de um comando* da aplicação. No MIKE, a definição da IU baseia-se na descrição de um conjunto de tipos de dados, das operações que podem operar sobre os valores destes ti-pos e, com base nestes, na descrição sintáctica de cada comando da aplicação, realizada usando um editor de comandos. Esta informação é equivalente à assi-natura da camada computacional que no GAMA constituirá uma parte da infor-mação do Modelo da Aplicação. No MIKE não são porém considerados invarian-tes de tipos de dados nem pré-condições, ainda que mecanismos simples de ac-tivação e desactivação de comandos sejam contemplados. No entanto, informa-ção semântica sobre os dados não é considerada (cf. invariantes).

Olsen afirma que MIKE é um sistema baseado na informação semântica da aplicação, desta forma permitindo-se desprezar a sintaxe. Do ponto de vista do autor, esta perspectiva não é correcta por duas razões. Em primeiro lugar, a assinatura da aplicação não constitui a semântica desta mas antes a sintaxe, ou seja, o que em geral se designa por API[4], e que são os meios sintácticos disponi-bilizados para acesso à semântica. Em segundo lugar, e dado que o primeiro passo na definição da IU em MIKE é, precisamente, determinar qual a sintaxe dos comandos da aplicação, tal como no GAMA, a sintaxe não pode ser conside-rada como desprezada, quando muito abstraída ou escondida do utilizador fi-nal. É no entanto certo que o estilo de interacção proposto permitirá esconder os detalhes sintácticos da linguagem de comandos, baseando-se numa navegação estrutural puramente sintáctica sobre as definições gramaticais dos comandos.

No sistema MIKE, um "parser" foi implementado para a síntese dos coman-dos da aplicação, "parser" que é orientado aos tipos de dados dos argumentos dos comandos, i.é., que dado um comando procura estruturalmente sintetizar um valor para cada um dos seus argumentos[5]. Porém, em MIKE informação se-mântica não é considerada, excepto inibir ou desinibir comandos, o que, con-forme os argumentos anteriormente expostos, conduz a IU que, ainda que muito rapidamente produzidas, não poderão apresentar um comportamento represen-tativo do seu comportamento final

---

4 de *Application Programmer´s Interface*.

5 estratégia nesta tese formalizada no capítulo 5 recorrendo ao modelo algébrico dos arquétipos.

esperado. Em MIKE, adicionalmente, a apli-cação deverá estar já completamente construída para que a IU possa ser prototi-pada.

UIDE.

UIDE (*User-Interface Design Development*) [Foley 87] [Foley et al. 89] é um SGIU que procura também a geração automática de IU a partir de especificações de alto nível. Na base do UIDE encontra-se uma descrição da semântica da aplica-ção realizada numa linguagem procedimental do tipo Pascal. As especificações realizadas nesta linguagem não descrevem a IU desejada mas sim as funções da aplicação e as suas pré e pós condições. A partir desta informação semântica, que é guardada numa base de conhecimento, o sistema permite a realização de testes de completude e coerência, e a geração de uma IU. Porém, dado que não são definidas propriedades sintácticas nem léxicas para esta IU, o que é de facto gerado é uma IU genérica a ser posteriormente ajustada quanto a apresentação e fluxo do diálogo. Interessante é o facto de o sistema UIDE possibilitar que al-gumas transformações possam ser realizadas sobre a IU, produzindo IU funcio-nalmente equivalentes. Uma transformação permite, por exemplo, que um objec-to esteja permanentemente seleccionado em vez de seleccionar um objecto para cada operação. Outra transformação típica corresponde a substituir uma opera-ção polimórfica (por exemplo *delete*) por funções especializadas para cada tipo a que a anterior poderia ser aplicada (por exemplo, *delete-square*, *delete-point*, etc.). A prototipagem da IU é garantida por um SGIU simples, porém, e tal como noutras abordagens, ou em independência da camada computacional e visando apenas aspectos de apresentação, ou em presença da camada computacional já completamente desenvolvida. Dado que UIDE apenas representa a informação semântica anteriormente indicada e, sendo esta manifestamente insuficiente, a prototipagem não é neste caso um processo de iteração propriamente dito, mas antes de construção das camadas léxica e sintáctica da IU, até então não espe-cificadas.

Em [Sukaviriya e Foley 90] as pré e pós condições das operações são usadas na geração automática de "help" para o utilizador.

ITS.

ITS [Wiecha et al. 90] é, tal como UIDE, um sistema para a geração automática de IU baseado em regras, popularizado dado ter sido utilizado na criação do sis-tema de informação de apoio aos visitantes da Expo-92 de Sevilha. A filosofia do ITS não tem semelhanças com o GAMA, sendo aqui apresentado não em termos comparativos mas em termos de complementaridade.

No sistema ITS todas as decisões de concepção da IU são codificadas como regras, por forma a poderem ser posteriormente usadas quer noutras concep-ções quer para análise de dada concepção. À medida que mais regras são intro-duzidas, a tarefa do "desenhador"[6], coerente com tal filosofia, é facilitada. Edi-ção gráfica não é apoiada dado que, deste modo, não seria possível registar as decisões tomadas nem as insuficiências detectadas em concepções anteriores que tiveram que ser resolvidas de tal forma.

A forma como regras construídas e registadas que acabaram por conduzir a concepções sem sucesso são editadas e corrigidas não é referida. ITS procura, de certo modo, construir sistemas periciais[7] contendo regras para a concepção de IU.

GIGA.

GIGA (*Generatori di Interfacce Grafico ed Automatico*) [Bordegoni et al. 90], de-senvolvido com o objectivo de permitir a geração automática de sistemas CAD, logo com todas as exigências de interacção inerentes, apresenta algumas seme-lhanças de arquitectura, de funcionamento e, até (quanto à escolha de notações de especificação) com o sistema GAMA, que devem ser aqui analisadas com al-gum detalhe. A abordagem em GIGA consiste em considerar a geração de IU di-vidida em duas fases: especificação da IU usando linguagens de alto nível e ge-ração automática do código da IU.

Tal como no GAMA, no GIGA uma particular linguagem de descrição foi defi-nida para cada um dos componentes da IU seguindo o modelo de Seeheim. O GIGA define de uma forma simples a camada de apresentação, usando objectos interactivos tais como menus, “forms”, “graphical areas”, “icons”, etc. Cada um dos objectos interactivos é identificado e associado a um objecto da aplicação. Este módulo, designado *Presentation Techniques Module*, é baseado num *Layout Editor*, de momento inexistente no sistema GAMA. É no entanto pouco clara a forma como cada apresentação interactiva é associada a cada objecto da aplica-ção.

O *Dialogue Control Module* do GIGA é especificado usando directamente Re-des de Petri do tipo Condição/Evento [Reisig 85], redes que apresentam alguma complexidade dado descreverem não apenas o fluxo do diálogo mas também a invocação de rotinas da aplicação e aspectos da apresentação. No GAMA, a des-crição do diálogo é bastante mais modular, tendo-se atingido maior separação, e, além disso, é realizada numa notação de muito mais alto nível, sendo as Re-des de Petri automaticamente geradas. O poder descritivo é, no entanto, equiva-lente, já que em ambos os sistemas diálogos “multi-threaded” podem ser imple-mentados. Do ponto de vista da arquitectura de implementação, GIGA usa tam-bém uma arquitectura multi-processo, sendo

---

6 do inglês *designer*, em português *o que concebe*.

7 *expert systems*.

cada um dos três componentes da IU implementado num processo separado, comunicando estes processos entre si através de filas de eventos.

O GIGA adopta uma estratégia semelhante à dos “frameworks” para a gera-ção do sistema final, dado que, partindo das descrições dos componentes conti-das em ficheiros, gera o código específico do sistema interactivo que é ligado ao código de um "esqueleto" pré-existente. Não existe portanto qualquer estrutura a interpretar em tempo de execução. Ainda que supostamente alguma melhoria de “performance” possa ser obtida, esta estratégia não permite a prototipagem rápi-da da IU, o que constitui uma evidente desvantagem.

CHIRON.

Chiron [Taylor e Johnson 93] baseia-se numa arquitectura cliente-servidor, existindo tipicamente um cliente que é a aplicação. Chiron, tal como GAMA, se-para claramente a aplicação do resto do sistema. O objectivo é minimizar a sen-sibilidade às alterações ambientais. A semelhança entre o Chiron e o GAMA re-sulta do facto de que também em Chiron a aplicação exporta um conjunto de TAD que são geridos por “dispatchers” que, por sua vez, comunicam com os de-signados “artists” que controlam as apresentações abstractas dos TAD contidas no servidor. Apesar das designações, existem semelhanças funcionais com as entidades implementadas no GAMA. Os TAD e os “dispatchers” correspondem no Chiron ao Modelo da Aplicação, os “artists” correspondem aos Guiões do GAMA e, portanto, implementam o Controlador do Diálogo. Os outros componentes re-sidem no processo “servidor” e correspondem aos *descritores da apresentação* do GAMA. No entanto, a separação de funcionalidade do Chiron parece ser pouco clara, não sendo fácil de determinar a divisão de responsabilidades na gestão do diálogo entre TAD e “artists” [Abowd et al. 93].

8.7.- Sumário.

Apresentou-se neste capítulo o sistema GAMA-X, o sistema de desenvolvimento de IU (SDIU) construído como "ferramenta" de suporte à prototipagem e desen-volvimento de sistemas interactivos segundo o modelo de interacção e o método sistemático de desenvolvimento propostos em capítulos anteriores.

As características arquitecturais e de funcionalidade do GAMA-X foram apre-sentadas, bem como o utilitário de edição de Guiões de Interacção e de geração automática das respectivas Redes de Petri que irão constituir o Controlador do Diálogo. Os Modelos da Apresentação e da Aplicação do GAMA-X, conforme a ter-minologia e divisão funcional sugerida pelo modelo de Seeheim e seguida pela maioria dos SGIU, foram igualmente apresentados, quer quanto à sua descrição quer relativamente à sua funcionalidade.

Tendo sido adoptada no GAMA-X a estratégia de que cada componente da IU seja implementada num processo individual, apresentou-se também o protocolo estabelecido entre estes componentes (ou processos), ou seja, o conjunto de mensagens que cada um é capaz de receber e a que é capaz de responder.

Comparações entre decisões, notações e possibilidades do GAMA-X e decisões tomadas noutros sistemas, sejam SGIU ou SDIU, foram também apresenta-das, ainda que com a relativização correspondente aos contextos e objectivos es-pecíficos dos respectivos desenvolvimentos.

A principal diferença a registar entre o GAMA-X e a maioria dos sistemas com que o mesmo foi comparado, consiste no facto de que o GAMA-X possibilita a construção de protótipos da IU "a priori" e de forma independente, ou seja, mui-to antes do desenvolvimento do código da aplicação e de forma coerente com a implementação final desta.

Por outro lado, e adicionalmente, apenas MIKE [Olsen 86] e UIDE [Foley 87] são, quanto aos objectivos finais, comparáveis com o GAMA-X, dado que apenas estes sistemas estruturam o controlo do diálogo relativamente a unidades de informação directamente inferidas da camada computacional. De facto, e salvo estas excepções, a maioria das abordagens vê a camada computacional como uma fonte de informação apenas "a posteriori", ou seja, após o desenvolvimento completo desta, o que não pode ser, com a tecnologia actualmente disponível, metodologicamente aceitável.

Deste ponto de vista, o sistema GAMA-X, sendo tecnologicamente uma "ferra-menta" que suporta o modelo de interacção e o método proposto, permite que IU possam ser geradas em fases iniciais do desenvolvimento de sistemas interacti-vos, sem que para tal deva existir mais do que um protótipo da camada compu-tacional, permitindo assim iteração sobre esta IU semi-automaticamente gerada, numa fase muito inicial do projecto do sistema interactivo. Em geral, o que fica por "ajustar" é a apresentação da IU (o seu "look" final) já que o seu comporta-mento (o "feel"), mesmo dentro das exigências estabelecidas, será observacional-mente equivalente ao comportamento final, salvo alterações nos requisitos.