O Sistema GAMA

Arquitectura e Implementação

José Francisco Creissac F. de Campos

Departamento de Informática

Universidade do Minho

jfc@di.uminho.pt

Fernando Mário J. Martins

Departamento de Informática/INESC

Universidade do Minho

fmm@di.uminho.pt

31 de Janeiro de 1994

Resumo

Este artigo apresenta o Sistema GAMA, um Sistema de Geração de Interfaces Humano-Computador com um elevado grau de assistência semântica no âmbito de uma metodologia rigorosa para o desenvolvimento de Sistemas Interactivos.

O sistema é composto por um módulo de geração semi-automática da especificação da interface (MGI) e outro de animação da interface com base na especificação anterior (MIU). A implementação do MIU é feita com base no Modelo de Seeheim, utilizando três processos, um para cada um dos componentes do modelo. Cada um dos componentes é apresentado bem como os protocolos de comunicação entre eles.

É também mostrado como o sistema permite, facilmente, obter tanto uma interface VT100 como uma X11.

1 Introdução

É hoje ponto assente que uma boa Interface Humano-Computador é um factor fundamental para o sucesso de qualquer sistema *software* interactivo.

Para além da complexidade inerente à concepção de um sistema *software*, o desenvolvimento de uma interface humano-computador, deve ainda ter em conta o factor humano com as suas características de não-determinismo, imprecisão, etc. O conceito de Interface em Modo Assistido[8] surge desta necessidade. Uma Interface Assistida age de um modo preventivo: evita que o utilizador cometa erros, solicita informação necessária ainda não fornecida, etc. Tal implica, necessariamente, que a camada interactiva do sistema tenha informação sobre a semântica da camada computacional.

Com o projecto GAMA[5, 3, 6] pretende-se complementar a linguagem de especificação CAMILA[2] e o método de refinamento a ela associado, com um Sistema de Gestão de Interfaces com o Utilizador que permita especificar e gerar interfaces, em Modo Assistido, tanto para os protótipos como para as aplicações finais resultantes do processo de refinamento da especificação.

2 Arquitectura do Sistema

O sistema pode ser dividido (ver fig. 1) numa componente de geração semi-automática (o Módulo de Geração de Interfaces - MGI), responsável pela geração e manutenção da especificação da interface, e por uma componente de *runtime* (o Módulo de Interacção com o Utilizador - MIU) que animará a interface a partir da especificação produzida pela primeira, gerindo todos os aspectos do diálogo entre o Utilizador e a Componente Computacional do Sistema Interactivo.

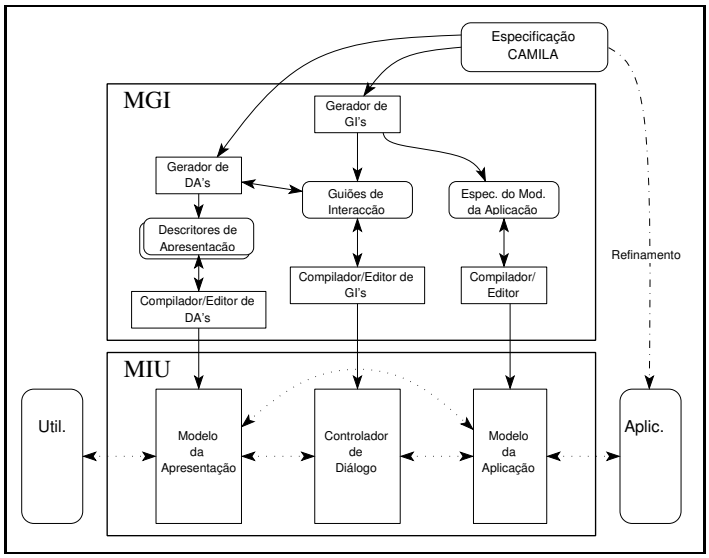

Figura 1: Arquitectura do GAMA

Sendo importante o conceito de validade semântica (cf. Modo Assistido), a separação rígida inicialmente proposta pelo Modelo de Seeheim[7] entre componentes Léxica, Sintáctica e Semântica torna-lo-iam inadequado para ser utilizado como Abstracção Arquitectural de uma interface em Modo Assistido. Deste modo, na arquitectura proposta para o Módulo de Interacção com o Utilizador, embora existam um Modelo da Aplicação, um Controlador de Diálogo e um Modelo da Apresentação, eles terão tarefas ligeiramente diferentes do proposto no modelo, estando a semântica da aplicação presente nos três. A especificação dos três componentes é feita com recurso a formalismos desenvolvidos para o efeito. Em particular, o Controlador de Diálogo é especificado por Guiões de Interacção[3].

O funcionamento do MGI processa-se em duas fases. Numa primeira fase, são geradas automaticamente versões base das especificações de cada componente (gerador de DA's e gerador de GI's). De seguida, um conjunto de editores/compiladores permitem a edição dessas especificações e posterior compilação das versões definitivas.

3 Guiões de Interacção

Os Guiões de Interacção (GI) são o formalismo desenvolvido para a especificação de Controladores de Diálogo no âmbito do sistema GAMA.

Um GI descreve um subdiálogo da interface. A título de exemplo apresenta-se um GI correspondente à síntese do comando **REMPAL(pal)**, que remove uma palavra de um dicionário. Note-se que a operação só é válida se o dicionário não estiver vazio e dele constar a palavra que se pretende remover.

Guião 1 (Guião de Interacção)

```
DefGI GRemPal
  Declarations
    TYPE SYNTH
    SYMBOL {GRemPal, rempal}
    ARGS pal: Pal
    VAR-UI sig: Sig
  Behaviour
    CONTEXT not(EMPTYDIC())
    INIT pal = "";
         sig = ""
    EVSEQ input(pal)
    TRANS input(pal): EXISTPAL(pal) => sig = CONSPAL(pal)
                                          EXCEP out("Erro!")
          OK:
          CANCEL:
    EXEC REMPAL(pal)
EndGI
```

Como se pode ver pelo exemplo apresentado, um GI divide-se em duas componentes principais: **Declaractions** e **Behaviour**. Cada uma delas está dividida, por sua vez, em várias cláusulas.

A componente **Declarations** engloba a declaração do tipo de Guião[1] (TYPE), dos símbolos que o identificam (SYMBOL), de Guiões externos ou de sub-Guiões que ele utiliza (EXTERNAL e SUBGI), dos argumentos da operação a executar (ARGS) e de variáveis locais, da aplicação, ou da apresentação que manipula (STATE-CTRL, STATE-APL e STATE-UI respectivamente).

A componente **Behaviour** engloba a declaração de condições de contexto necessárias para que a utilização do GI seja válida (CONTEXT), de acções de inicialização (INIT), da sequência de eventos que o GI aceita (EVSEQ), de condições a verificar e acções a executar para cada evento (TRANS) e da expressão a enviar à Aplicação para execução (EXEC).

Para descrever, em EVSEQ, as sequências possíveis de eventos existem os operadores de sequência (.), paralelismo (com e sem sincronização, | e ||), repetição (*) e escolha (+). Os eventos possíveis são, ou o evento especial **input** (**input(v)** corresponde à leitura da variável **v**), ou Guiões de Interacção (correspondendo ao processamento dos mesmos)[2].

Na cláusula TRANS é ainda possível declarar eventos especiais, correspondentes a comandos pré-definidos: cancelamento do GI (CANCEL), confirmação do diálogo (OK), reinicialização do GI (RESET) e repetição do GI (APPLY).

4 Implementação do MIU

A implementação actual do MIU está feita em NYAGSL[1] e é composta por três processos, um para cada componente do MIU. Para permitir a comunicação entre os diferentes componentes desenvolveram-se protocolos de comunicação. De seguida, descrevem-se cada um dos componentes, bem como os protocolos de comunicação entre eles.

4.1 Controlador de Diálogo

O Controlador de Diálogo é, basicamente, um animador de Guiões de Interacção e foi desenvolvido sobre um animador de *Petri Nets*[9] (o formalismo utilizado para implementar os Guiões). É composto, ainda, por um interpretador da linguagem de comandos utilizada nos Guiões, por módulos de tratamento dos modelos de dados (funções finitas, relações, conjuntos, listas e tuplos) e por um módulo responsável pela comunicação com os restantes componentes.

Animador de *Petri Nets*

Sendo o comportamento dos Guiões de Interacção modelado por *Petri Nets*, o Controlador de Diálogo foi desenvolvido sobre um animador dessas redes que aqui se apresenta.

Formalmente as redes utilizadas podem ser modeladas da seguinte forma:

[1] Existem diferentes tipos de Guiões de Interacção correspondentes a diferentes fases do diálogo.

[2] No caso do exemplo a expressão não faz uso dos operadores, pois trata-se do caso mais simples da leitura de uma variável.

```
PetriNet :: B : Case
            Ev: Events
            Fi: Flow
            Fo: Flow
            Cb: Case
            Ce: Case;
Conditions = CId -> Bool;
Events = Event-set;
Flow = Event -> Case;
Case = CId-set;
```

B é o conjunto de todos os lugares da rede, **Ev** é o conjunto de todas as transições, **Fi** e **Fo** são os fluxos de entrada e saída dessas transições (de que lugares retiram *tokens* e em que lugares os colocam) e **Cb** e **Ce** são as marcações inicial e final, respectivamente. Como a definição da *Petri Net* será partilhada por todas as instâncias de um Guião ela não inclui a marcação actual, devendo cada instância conservar a sua.

A utilização do animador é feita por meio das seguintes funções:

startpn: PetriNet × InstId → Conditions

que inicializa a rede cuja descrição lhe é passada como parâmetro, devolvendo a marcação daí resultante - a necessidade de indicar a instância prende-se com a verificação das condições necessárias para que as transições se possam efectuar -;

firepn: PetriNet × Conditions × Event × InstId → Conditions

que implementa o disparo de uma transição - neste caso é passada a descrição da rede, a marcação actual, qual o evento e a instância, sendo devolvida a marcação resultante da transição - e

endpn: PetriNet × Conditions → Bool

que serve para verificar se uma dada marcação é a marcação final da rede referida.

Adicionalmente existe a função:

valideventspn: PetriNet × Conditions × InstId → Event-set

para calcular todos os eventos possíveis para uma dada marcação da rede.

O Interpretador de Instruções

Outra tarefa, bem definida, que o Controlador de Diálogo deve realizar, é a execução das instruções e cálculo das expressões colocadas em INIT e TRANS. Para o efeito foi desenvolvido um interpretador de instruções.

Na sua versão actual estão previstas expressões envolvendo atribuições, invocações de operações (com ou sem resultado), a instrução **out**, as instruções condicional, **if...then...else...** e cíclica, **while...do...**. Foi ainda incluído o tratamento de expressões do tipo inteiro, *string* e booleano.

A utilização do interpretador é feita através das operações

docode: Code × InstId →

para executar uma sequência de instruções e

dotypedexp: TypedExp $\times$ InstId $\times$ OpcGISym $\rightarrow$ Value

para calcular o valor de uma expressão. O identificador de instância passado às operações é o da instância à qual se referem as expressões a calcular e é necessária para se fazer o acesso às variáveis do Guião.

A Inclusão dos Tipos

Ao nível da especificação os valores dos argumentos são sempre considerados como *tokens* léxicos, independentemente do seu tipo. Na implementação, no entanto, torna-se necessário controlar a leitura desses valores. Se os tipos básicos (inteiros, *strings*, etc.) não apresentam problemas, o mesmo já não se pode dizer dos que são definidos à custa de modelos. A leitura de uma Função Finita ou de um Conjunto não é uma tarefa trivial, existem vários valores a serem lidos e condições que devem ser verificadas[3]. Coloca-se, então, o problema de decidir quem/como controlar esse diálogo.

A solução implementada foi colocar esse controlo no Controlador de Diálogo. Para tal, desenvolveu-se uma série de pseudo-Guiões, um para cada modelo. Assim, quando uma variável é de um tipo definido com recurso a modelos estruturados, o seu tipo léxico é substituído pela indicação do GI que deverá controlar a interacção a ela referente. No Controlador de Diálogo, por sua vez, o controlador "desvia" o processamento dos eventos para um módulo de tratamento apropriado. Este processo é transparente uma vez que o protocolo de comunicação continua a ser o mesmo.

Este modo de funcionamento permite ter vários métodos de leitura para cada um dos modelos, bastando para tal definir tipos de Guiões e os respectivos módulos de processamento associados. Actualmente estão definidos tipos de Guiões para a leitura de Funções Finitas (FFSYNTH), Relações (RELSYNTH), Listas (LISTSYNTH), Conjuntos (SETSYNTH) e Tuplos (TUPSYNTH). A definição dos módulos de processamento para cada um destes Guiões baseou-se, em parte, na experiência adquirida em [4].

Os Guiões do tipo FFSYNTH e RELSYNTH geram um diálogo em que os pares domínio/contradomínio (variáveis **dom** e **ran**) são apresentados/lidos um a um, sendo disponibilizados, para além de OK e CANCEL, os comandos NEW (inicializar a vazio), UP (par anterior), DOWN (par seguinte) e DEL (apagar o par actual). Internamente, o valor é representado por quatro listas de valores:

- **dup** - valores do domínio anteriores ao valor actualmente apresentado;
- **rup** - valores do contradomínio correspondentes;
- **ddown** - na cabeça está o valor actualmente apresentado e na cauda os restantes;
- **rdown** - valores do contradomínio correspondentes.

[3]Por exemplo, no caso das Funções Finitas, não podem existir repetições no domínio.

Os Guiões do tipo SETSYNTH e LISTSYNTH são semelhantes aos anteriores mas a variável utilizada para apresentar os valores é **elem**.

Por último, os Guiões TUPSYNTH geram um diálogo correspondente à expressão

input(sel1) || **input(sel2)** || ... || **input(seln)**[4]

com os comandos OK e CANCEL e em que **sel1** a **seln** são os selectores do tuplo. Importa notar que a representação sintática utilizada para os tuplos é uma função finita de nome do selector para valor correspondente.

Estrutura Geral do Controlador

A informação necessária ao Controlador de Diálogo consiste na definição dos Guiões de Interacção, na descrição das instâncias existentes e nos canais de comunicação com os outros componentes de MIU. Temos então:

```
CD :: DEFS:  GISym -> GIdef
      INSTS: InstId -> InstDescr
      OUTSM: CHAN
      INSM:  CHAN
      INLX:  CHAN
      OUTLX: CHAN;
GIdef = DECISION | SYNTH | VALSYNTH | FFSYNTH | RELSYNTH |
        SISTSYNTH | SETSYNTH | TUPSYNTH;
FFSYNTH :: DREF: TypeId;
InstDescr :: FATHER:  NIL | InstId
             VARS:    VarId -> Value
             EVSEQ:   Conditions
             CMDLINE: NIL | CmdLineDescr;
```

O seu funcionamento consiste basicamente em:

- criar as instâncias, quando tal é pedido;
- idem para a sua remoção;
- validar e efectuar, na *Petri Net* da instância indicada, as transições relativas aos eventos que lhe são comunicados;
- indicar ao Modelo da Apresentação quais os eventos válidos em cada momento;
- indicar, também, alterações nos valores das variáveis conhecidas pelo Modelo da Apresentação;
- construir e enviar a frase ao Modelo da Aplicação para avaliação e receber o resultado, passando-o ao Modelo da Apresentação.

Na fig. 2 são apresentados os ficheiros que compõem o Controlador de Diálogo. Note-se que as setas representam uma relação de inclusão textual, pelo que podem existir (e existem de facto) referências cruzadas entre os ficheiros.

[4]Leitura em paralelo de todos os valores do Tuplo.

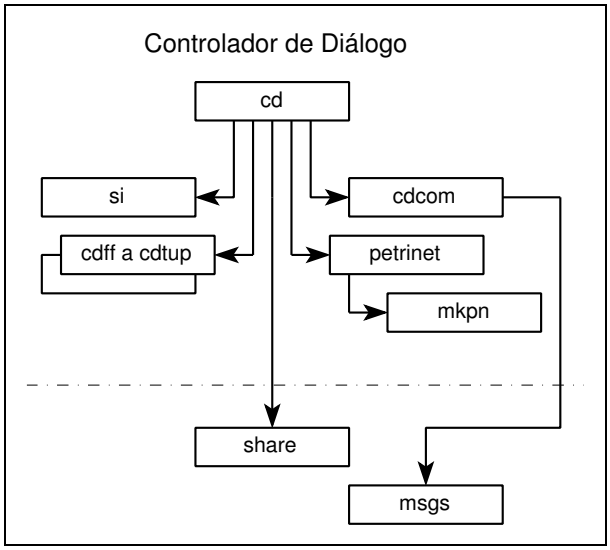

Figura 2: Composição do Controlador de Diálogo

O ficheiro **cd.n** contém o corpo principal do Controlador. Em **cdcom.n** estão as funções de comunicação e este por sua vez inclui **msgs.n** onde são definidos os protocolos de comunicação; **petrinet.n** é o animador de *Petri Nets* e **mkpn.n** um gerador das mesmas a partir das definições dos GI's; **si.n** é o interpretador de instruções e **cdff.n** a **cdtup.n** são os módulos de tratamento dos modelos.

Finalmente, em **share.n** são feitas definições globais aos três componentes do MIU. Tanto este ficheiro como **msgs.n** são partilhados pelos três componentes do sistema.

4.2 Modelo da Apresentação

O Modelo da Apresentação pode ser dividido num módulo genérico de processamento de eventos (ModApr) - cuja principal responsabilidade é implementar as comunicações com o Controlador de Diálogo - e um *frontend* que implementará o aspecto gráfico da interface e a tradução dos diferentes eventos de e para o utilizador. Deste modo, por simples substituição do *frontend* utilizado, podemos passar de uma interface VT100 para outra em X11.

Nas figuras 3 e 4 é apresentada uma interface gerada pelo sistema. No primeiro caso, é utilizado o *frontend* NYAGSL, obtendo-se uma interface VT100. No segundo caso, utilizou-se uma versão experimental de um *frontend* X11 que está em desenvolvimento.

Um novo grau de liberdade, introduzido pela separação entre a especificação do Controlador de Diálogo e a do Modelo da Apresentação, consiste na possibilidade de especificar diversas apresentações para a mesma interface, sendo possvel, em *runtime*, alterar a apresentação que está a ser utilizada. Podemos, assim, substituir, por exemplo, *dialog boxes* por *option menus* e leituras de valores via teclado por *scales* durante uma sessão de trabalho e sem necessidade de abandonar a aplicação.

Na fig. 5 são apresentados os ficheiros que compõem o Modelo da Apresentação.

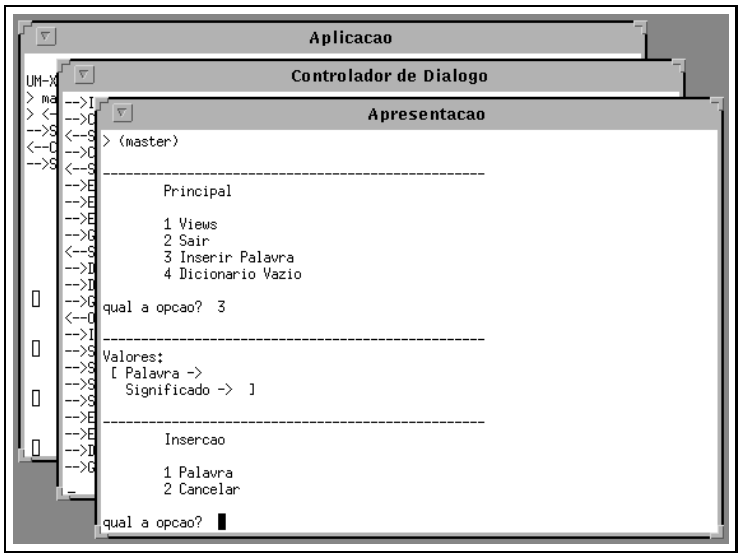

Figura 3: *Frontend* VT100

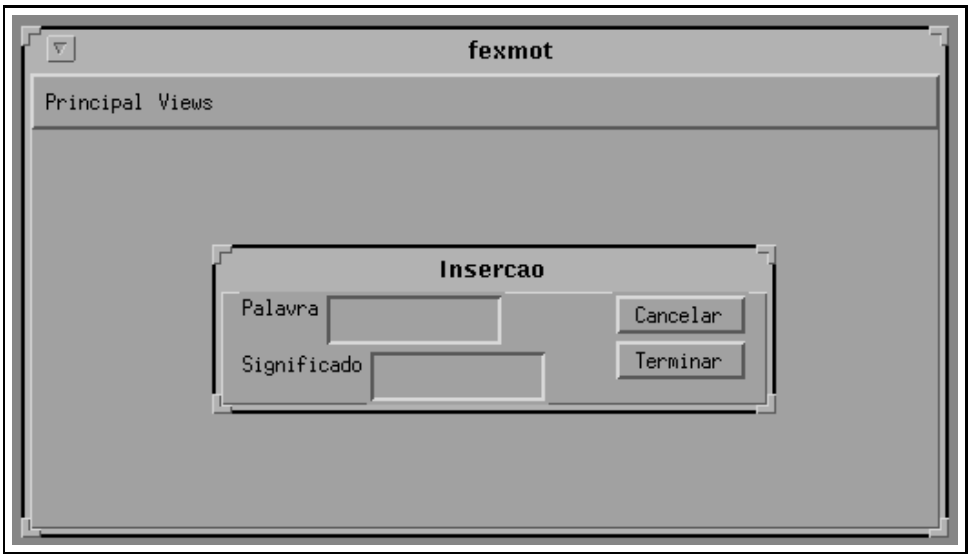

Figura 4: *Frontend* X11

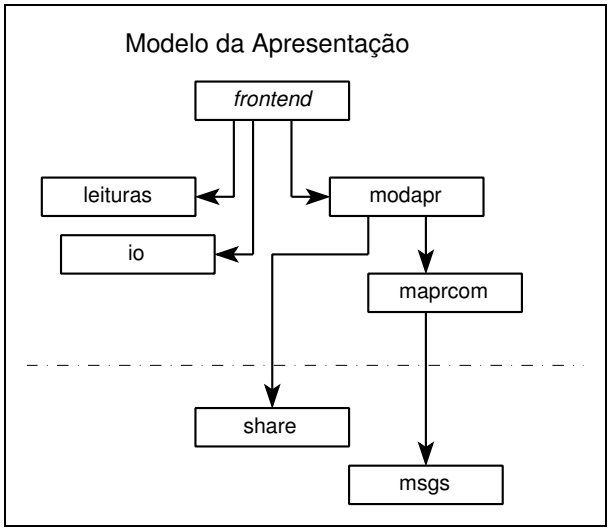

Figura 5: Composição do Modelo da Apresentação

Os ficheiros **io.n** e **leituras.n** implementam o *input/output*. O ficheiro **fexm.n** é o corpo principal do *front-end* e **modapr.n** o do ModApr, **maprcom.n** implementa as comunicações.

4.3 Modelo da Aplicação

O Modelo da Aplicação é composto por uma parte fixa, que será apresentada, e pelas funções de tradução de valores entre as representações da Aplicação e do CD (representação NYAGSL), que devem ser fornecidas pela equipa de desenvolvimento da camada computacional e que, por serem dependentes da aplicação em causa, não serão abordadas. Deste modo, quando referirmos o Modelo da Aplicação, estaremos apenas a falar da componente referida em primeiro lugar.

Esta componente da interface é a mais simples das três, limitando-se a ser um servidor de pedidos feitos pelas outras duas:

- pedidos para obtenção das definições dos tipos;
- pedidos para verificação de invariantes;
- pedidos para invocação de operações;
- pedidos para obtenção dos valores de variáveis da aplicação.

Como a fig. 6 mostra, a estrutura de ficheiros reflecte esta simplicidade. São apenas necessários, para além dos ficheiros comuns a todas as componentes do MIU, o corpo principal do servidor (**modapl.n**) e as rotinas de comunicação (**maplcom.n**).

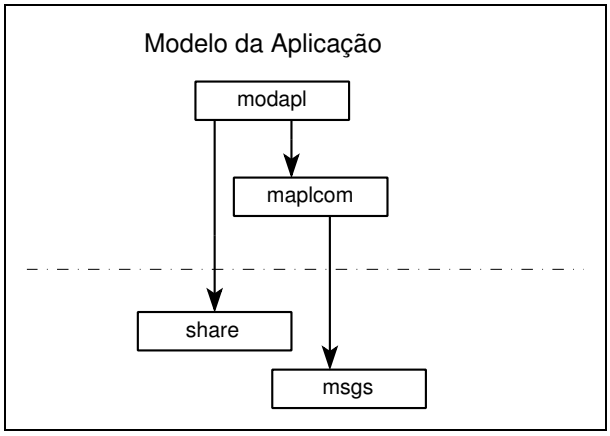

Figura 6: Composição do Modelo da Aplicação

4.4 Protocolos de Comunicação

Estando a informação semântica dispersa pelos três componentes do sistema, é necessário que exista colaboração entre eles. Tal é conseguido recorrendo a protocolos de comunicação inter-processos desenvolvidos sobre *channels* UNIX[5].

O tipo de mensagens que cada componente pode receber/gerar depende do tipo de tarefas que ele realiza.

O Modelo da Apresentação é responsável pela recepção e envio de eventos válidos do utilizador ao Controlador de Diálogo, o que dá origem às seguintes mensagens no sentido Modelo da Apresentação - Controlador de Diálogo:

- **CreateMsg** - pedido para que seja criada uma instância de um GI;
- **OpenMsg** - pedido para que seja activada uma instância;
- **KillMsg** - pedido para que seja eliminada uma instância;
- **StartMsg** - início de um determinado evento;
- **EndMsg** - fim de um evento e, opcionalmente, qual o valor por ele devolvido;
- **CancelMsg** - cancelamento de um evento.
- **CmdMsg** - selecção de um dos comandos pré-definidos;
- **SetValMsg** - envio de um valor a uma instância.

Por seu lado, o Controlador de Diálogo deve processar os eventos e indicar ao Modelo da Apresentaão quais os próximos eventos válidos e quais os valores que devem ser apresentados ao utilizador. O protocolo de comunicação no sentido Controlador de Diálogo-Modelo da Apresentação é:

[5]Ou melhor, utilizando as primitivas de comunicação NYAGSL que estão desenvolvidas sobre *channels* UNIX.

- **InstMsg** - identificação de uma instância recém criada;
- **EnableMsg** - tornar um determinado evento válido;
- **DisableMsg** - tornar um determinado evento inválido;
- **OutMsg** - corresponde às instruções **out()**;
- **ShowMsg** - atribuição de um valor a uma variável;
- **StopMsg** - final bem sucedido de um Guião;
- **AbortMsg** - cancelamento de um Guião;
- **GoMsg** - terminou o processamento da mensagem recebida.

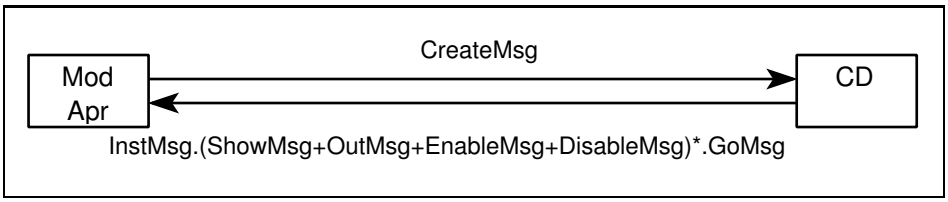

Figura 7: Comunicações correspondentes a CreateMsg

Na fig. 7 é apresentado, a título de exemplo, o esquema correspondente às comunicações geradas pelas mensagem **CreateMsg**. Depois de recebida a mensagem, o Controlador de Diálogo indica o identificador da instância criada (**InstMsg**) e, de seguida, gera zero ou mais mensagens **ShowMsg**, **OutMsg**, **EnableMsg** e **GoMsg** para a inicializar; finalmente, a mensagem **GoMsg** é utilizada para indicar ao Modelo da Apresentação que o Controlador de Diálogo está pronto para receber uma nova mensagem.

O Controlador de Diálogo necessita também comunicar com o Modelo da Aplicação. Neste caso, o protocolo é mais simples. As mensagens que o Controlador de Diálogo envia ao Modelo da Aplicação são:

- **GetVarMsg** - para a obtenção do valor de uma dada variável da Aplicação;
- **GetTypeMsgSt** - para a obtenção da definição de um dado tipo;
- **CallMsg** - para a execução de um dado comando;
- **HaltMsg** - informa o Modelo da Aplicação de que deve terminar a execução.

Na sentido inverso existem apenas as mensagens:

- **SetValMsg** - para enviar valores ao Controlador de Diálogo, quer como resposta a **GetValMsg**, quer a **CallMsg**;
- **DefTypMsg** - devolução da definição de um tipo.

Faltam agora as comunicações entre o Modelo da Apresentação e o Modelo da Aplicação. O Modelo da Apresentação envia as mensagens:

- **GetTypeMsgLx** - para a obtenção da definição de um dado tipo;
- **InvMsg** - para a verificação de invariantes.

O Modelo da Aplicação responde com:

- **SetValMsg** - para devolver o resultado do teste de um invariante;
- **DefTypMsg** - para devolução da definição de um tipo.

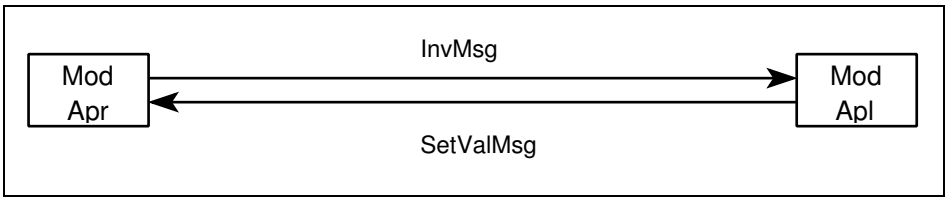

Figura 8: Comunicações correspondentes a InvMsg

Na fig. 8 estão representas as mensagens correspondentes a um pedido de verificação de invariante.

5 Conclusão

No artigo é apenas descrita a implementação da componente de *runtime* do sistema (o MIU). Em relação ao MGI, apenas o editor/compilador de Guiões de Interacção foi desenvolvido[10], sendo ainda necessário desenvolver todos os outros componentes.

Ao contrário de muitos sistemas tradicionais, durante o desenvolvimento do GAMA esteve presente a necessidade de proporcionar ao utilizador interfaces com um elevado grau de assistência semântica. Esta caracterstica reflecte-se no sistema de dois modos. Por um lado, a informação semântica está dispersa pelos diferentes componentes do sistema e não concentrada no Modelo da Aplicação. Por outro, existe uma grande colaboração entre os diferentes componentes, nomeadamente por parte do Modelo da Aplicação que funciona não só como servidor de pedidos para execução de operações da Aplicação, mas também para validação de dados.

Se o Princpio da Separação introduziu um grau de liberdade ao permitir ter a mesma aplicação com interfaces diferentes, a separação entre a Apresentação e o Controlador de Diálogo veio permitir ter para a mesma interface, apresentações diferentes. A actual versão do Modelo da Apresentação, a correr em XMetoo, não é muito interessante do ponto de vista estrito de utilização, mas permitiu testar rapidamente o funcionamento do sistema. Uma versão para X11 está, neste momento, em desenvolvimento.

Referências

[1] J. J. Almeida, J. B. Barros, P. M. Castro, and F. C. Madeira. Pré-processador para YARPT. Relatório jnict:pmct:87/66:jja1, Departamento de Informática, Universidade do Minho, Outubro 1989.

[2] L. Barbosa and J. J. Almeida. CAMILA by Example. Relatório interno, DI/INESC, Universidade do Minho, 1991.

[3] J. C. Campos. GAMA-X Geração Semi-Automática de Interfaces Sensíveis ao Contexto. Dissertação de mestrado, Departamento de Informática, Universidade do Minho, 1993.

[4] J. C. Campos and F. M. Martins. IAPF - Interfaces Assistidas para Protótipos Funcionais. Technical report, Universidade do Minho/INESC, 1992.

[5] J. C. Campos and F. M. Martins. GAMA-X - Uma Arquitectura Software para o Desenvolvimento Semi-Automático de Interfaces Utilizador-Sistema. In *5º Encontro Português de Computação Gráfica*, pages 197–209, Fevereiro 1993.

[6] J. C. Campos and F. M. Martins. Automatic Generation of User Interfaces at Prototype Level. Technical report, Project C19 - Olivetti Ricerca/INESC-Braga, 1994.

[7] M. Green. A Survey of Three Dialogue Models. *ACM Transactions on Graphics*, 5(3):243–275, Julho 1986.

[8] F. M. Martins and J. N. Oliveira. Archetype Oriented User Interfaces. *Computer & Graphics*, 14(1):17–28, 1990.

[9] W. Reisig. *Petri Nets - An Introduction.* Springer-Verlag.

[10] F. Rocha. Um Editor/Compilador de Guiões de Interacção. Relatório de estágio, Universidade do Minho/Departamento de Informática, 1993.