GAMA-X

MIU Programmer's Manual

José Creissac Campos
jfc@di.uminho.pt

Departamento de Informática
Universidade do Minho
Largo do Paço
4719 BRAGA Codex
PORTUGAL

1995

Contents

1 Introduction

The GAMA-X system is a semi-automatic generator of assisted interfaces[3]. It aims at being an UIMS (User Interface Management System) for the Camila system[1], capable of providing an user interface that can be used at all levels of the application development cycle, from the prototype to the final implementation.

The GAMA-X system is divided into two main modules: the MGI (Interface Generation Module) and the MIU (User Interaction Module). The MGI generates an user interface specification (or MIU specification) from the computational layer specification. The MIU creates an user interface based on the specification generated by the MGI.

The MIU specifications can be generated by the MGI or hand written from scratch. In this manual we show, with an example, how to write the MIU specification.

2 The Example

The example application will be a simple dictionary. The Camila specification of the dictionary is presented in Appendix A. From the specification we can see that there are five main operations:

- `INIT` - initializes the dictionary
- `INSWORD` - inserts a word
- `INSDIC` - joins a new dictionary
- `DELWORD` - deletes a word
- `DEFWORD` - finds the meaning of a word

We can also see that these operations manipulate three user defined types:

- `Dic`
- `Word`
- `Meaning`

To specify the user interface for the dictionary we must specify the behavior of the three components of the MIU (see fig. 1):

- the Application Model (section 3)

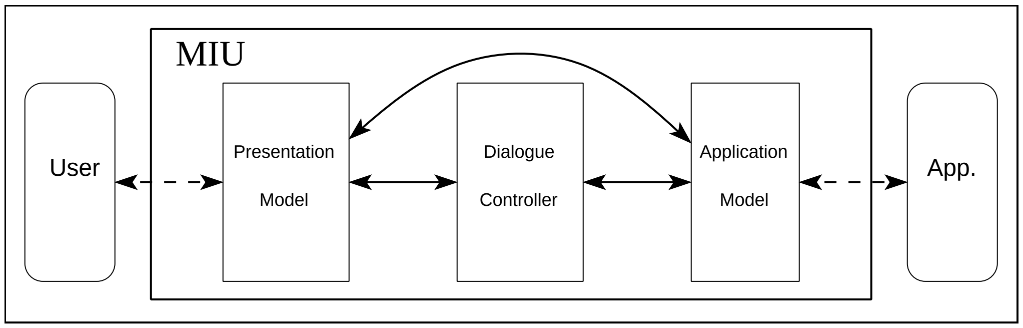

Figure 1: MIU architecture

- the Dialogue Controller (section 4)
- the Presentation Model (section 5)

3 The Application Model

The application model is the bridge between the dialogue controller and the application code. Thus, we will have different types of application models specification for different implementations of the computational layer. We will present the application model specification for a prototype in Camila.

The complete specification is shown in Appendix B. The specification is made in an operation named `init` that uses the `ModApl` type constructor to define a state named `ma`. `ModApl` has six arguments.

The first argument is a finite function defining the user defined types and their invariants. If the user type is defined using models its description is made with a list. The head of the list indicates the model:

- `FF` - finite functions;
- `REL` - relations;
- `SET` - sets;
- `LIST` - lists;
- `TUP` - tuples.

The tail lists the type(s) involved in the definition. In the case of tuples, the tail is made of pairs of name of selector/type.

The second argument is a finite function defining for each application variable that we want visible in the dialogue controller an operation that accesses its value. In this case we are not using this feature so the second argument of `ModApl` is empty.

The third argument is a finite function defining for each virtual operation known in the dialogue controller an actual operation of the application code. We must note here that, as the MIU representation of tuples is a finite function of name of field to value of field, when we have tuples we must write functions to make the translation. In Appendix G we give an example of an application model where we have tuples.

The last three arguments define the communication channels[1]:

- messages from the Dialogue Controller to the Application Model;
- messages from the Application Model to the Dialogue Controller;
- messages from the Application Model to the Presentation Model;

Finally the code of the prototype is included in the model:

4 The Dialogue Controller

The specification of the dialogue controller is made with *Guiões de Interacção*[2, 4] (see Appendix C for the complete specification using GIs). A compiler of GIs will be made available soon. For now we must write the specification directly in CAMILA (see D for the complete specification in CAMILA).

The specification is made in an operation named `init` that uses the `CD` type constructor to define a state named `cd`. `CD` has six arguments.

The first argument is a finite function and defines the GIs. The second argument must be the empty finite function. The last four are the communication channels[2]:

- messages from the Dialogue Controller to the Application Model;
- messages from the Application Model to the Dialogue Controller;
- messages from the Presentation Model to the Dialogue Controller;

[1]Their names must be coherent with the ones used in the specification of the other components of the MIU.

[2]Again, the names of the channels must be coherent with the ones used in the specification of the other components of the MIU.

- messages from the Dialogue Controller to the Presentation Model;

There are three basic types of GI constructors: `DECISION`, `SYNTH`, and `VALSYNTH`. For the handling of models there are five special constructors:

- `FFSYNTH` - for finite functions;
- `SETSYNTH` - for sets;
- `LISTSYNTH` - for lists;
- `RELSYNTH` - for relations;
- `TUPSYNTH` - for tuples.

The definition of the type of GI is made by the type of constructor used.
The five special constructors have only one argument: the type they will handle (see the `Dic` GI).
The basic constructors all have the same ten parameters:

- the set of symbols that identify the GI;
- the type of the result of the GI[3];
- the set of identifiers of external GIs used by this GI;
- the set of identifiers of subGIs of this GI;
- the declaration of variables of the GI;
- the context expression of the GI;
- the initialization code of the GI;
- the event sequence that defines the behavior of the GI;
- the description of the transitions associated with each event;
- the expression that should be evaluated on ending of the GI.

[3]For `DECISION` and `SYNTH` GIs this argument is not relevant.

4.1 Variables

Variables are declared in a finite function of variable identifier to variable definition. We have four constructors to define the variables:

- `VarUI` - for variables declared in `VAR-UI` or in `ARGS`;
- `VarCTRL` - for variables declared in `VAR-CTRL`;
- `VarAPL` - for variables declared in `VAR-APL`
- `VarAPLcopy` - for variables declared in `VAR-APL` using the form `a<=b`.

`VarUI` and `VarCTRL` have only one parameter: the type of the variable. `VarAPL` has two parameters: the type of the variable and its value (initially NIL). `VarAPLcopy` has two parameters: the type of the variable and the name of the application variable to copy.

4.2 Boolean Expressions

Boolean expressions are written using the `true` and `false` constants, application functions (see 4.3 on function calling), variables and the logical and relational operators listed:

- `Eq(a, b)` - $a = b$;
- `Ne(a, b)` - $a \neq b$;
- `Ge(a, b)` - $a \geq b$;
- `Le(a, b)` - $a \leq b$;
- `Gt(a, b)` - $a > b$;
- `Lt(a, b)` - $a < b$;
- `And(a, b)` - $a \wedge b$;
- `Or(a, b)` - $a \vee b$;
- `Not(a)` - $\neg a$;

4.3 Code

Code is written as a list of instructions. For writing instructions we have the following functions (see Appendix F for a complete listing):

- `Atrib(var, exp)` - assigns the value of expression `exp` to the variable `var`;
- `ProcCall(opr, args)` - calls procedure (operation with no return type) `opr` with the list of arguments `args`;
- `Out(str)` - sends the string `str` to the presentation;
- `If(g, t, el)` - if the boolean expression `g` evaluates to true then the code `t` is executed, else the code `el` is executed;
- `While(g, c)` - while the boolean expression `g` evaluates to true the code `c` is executed.

In expressions we can have:

- Integer expressions:
  - `Add(a, b)` - addition of `a` and `b`;
  - `Sub(a, b)` - subtraction of `a` and `b`;
  - `Div(a, b)` - integer division of `a` and `b`;
  - `Mul(a, b)` - multiplication of `a` and `b`;
  - `Mod(a, b)` - modulus of `a` and `b`;
- String expressions:
  - `StrExp(str)` - for now just constants;
- Boolean expressions - see 4.2;
- Constant values;
- Function Calling - `FunCall(fun, args)` the same as `ProcCall` but a result is returned.

4.4 Event Sequences

For writing the event sequence we have a function for each operation:

- the expression `a.b` is written `SeqDescr(a, b)`;
- the expression `a||b` is written `ConcSDescr(a, b)`;
- the expression `a|b` is written `ConcADescr(a, b)`;
- the expression `a+b` is written `OpcDescr(a, b)`;
- the expression `a*` is written `RepDescr(a)`.

Events are written with the `EvId` function. This function has two parameters:

1. the name of the GI associated with the event (if NIL then it is an `input` event);
2. the name of the variable associated with the event.

4.5 Transitions

To write the TRANS clause in Camila we have the `TransDescr` function. Its parameters are:

1. the boolean expression to test;
2. the code to execute if the condition is true;
3. the list of exceptions.

The exceptions are written with the `ExcepDescr` function, with parameters:

1. the condition to test;
2. the code to execute.

4.6 EXEC

To write the EXEC clause we have the `ExecDescr` function, with two parameters:

1. the operation to execute;
2. the list of parameters to the operation.

5 The Presentation Model

The complete specification of the presentation model for the dictionary is presented in Appendix E.

In the presentation model we define how each GI should be presented to the user. The specification is made in a function called `init`. The `init` function defines a state named `ma` with the `ModApr` type constructor. `ModApr` as seven parameters:

1. global information about the GIs;
2. description of the views[4];
3. default view;
4. initial GI;
5. channel from Presentation Model to the Dialogue Controller;
6. channel from Dialogue Controller to the Presentation Model;
7. channel from Dialogue Controller and Presentation Model to the Application Model;
8. channel from Application Model to the Presentation Model;

In the first parameter we use the `GlobalInfo` constructor to declare the presentation variables that each GI uses and their types and also the synonyms of each GI.

To define a view we must associate a lexical description to each GI: If we want the GI to look like a Dialogue Box (usually SYNTH and VALSYNTH GIs), we must use a `DB` description. If we want the GI to look like a Menu (usually DECISION GIs) we must use the `MENU` function.

In the `DB function` we define:

1. the lexical definition of the GI;
2. for each variable, its lexical type;
3. for each event that involves a sub or external GI, how it should be viewed in this GI;
4. for each command, its lexical definition.

[4]Each view is a complete description of the look of all the GIs. This way, by changing the view, we can change the look of the application

The lexical definition of the GI is done with the `LxDef` function. Its parameters are:

1. position;
2. name;
3. color.

For now only the second parameter is being considered.

To define the lexical type of a variable we must look at its syntactic type. If its syntactic type is defined using models, then the lexical type will be the GI that controls the interaction with the model (see section 4 and variable `d` of GI `GInsDic` in Appendix D). Otherwise we can use the following functions:

- `HimSelf` - text based interaction;
- `Scale` - a scale is presented (only for numeric values);
- `RadioBox` - a radio box is presented;
- `OptionMenu` - an option menu is presented.

The `HimSelf` function has only one argument: the lexical definition of the variable (done with the `LxDef` function). The `Scale` function has three arguments: the minimum and maximum values of the scale and its lexical definition (done with the `LxDef` function). The `RadioBox` and `OptionMenu` functions both have two arguments: a finite function of value name to value and its lexical definition (once again done with the `LxDef` function - see Appendix E for an example).

If the GI is a `FFSYNTH` or a `RELSYNTH` GI, then it has the variables `dom` and `ran`. `LISTSYNTH` and `SETSYNTH` GIs have only one variable: `elem`. `TUPSYNTH` GIs have one variable for each element of the tuple.

In the third parameter of `DB` we declare for each event that involves a sub or external GI how it should be presented. This is done with the `DBGIView` function. This function has two parameters:

1. the lexical definition of the event done with the `LxDef` function;
2. a boolean value that states how the GI associated with the event should be presented:
    - `true` - the lexical definition of the GI is automatically presented;
    - `false` - only a button is presented, when it is pressed the GI is shown.

The lexical definition of the commands are done with `LxDef`. `FFSYNTH`, `RELSYNTH`, `SETSYNTH` and `LISTSYNTH` GIs have the commands:

- `OK`;
- `Cancel`;
- `New` empty data;
- `Up` previous element;
- `Down` next element;
- `Del` delete element.

`TUPSYNTH` GIs have the `OK` and `Cancel` commands.

The `MENU` function has three parameters:

1. the lexical definition of the GI, done with the `LxDef` function;
2. the way to mark the selected item (not used);
3. a list with the name of each event of the GI.

6 Putting it all together - the gama-x command

For getting the application running we must put each one of the three definitions in a separate file. Let's say we call them:

- `dapl.n` - the application module
- `dcd.n` - the dialogue controller
- `dapr.n` - the presentation module

Now we must compile them with the `seca` compiler:

```
seca dapl.n dapl.met
seca dcd.n dcd.met
seca dapr.n dapr.met
```

We are now ready for running the application. We use the `gama-x` interpreter. It has three parameters:

1. the presentation model
2. the dialogue controller
3. the application module

and the options:

- `met` - generates a `vt100` interface;
- `win` - generates a window for each of the MIU components showing the message that they receive and send (good for debug!)

To generate a X11 interface to the dictionary application we write:

```
gama-x dapr dcd dapl
```

References

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

[2] José Creissac Campos. GAMA-X Geração Semi-Automática de Interfaces Sensíveis ao Contexto. Master's thesis, Escola de Engenharia, Universidade do Minho, 1993.

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

[4] Fernando Mário Martins. *Métodofs Formais na Concepção e Desenvolvimento de Sistemas Interactivos.* PhD thesis, Escola de Engenharia, Universidade do Minho, 1995.

A The Dictionary Specification

```
TYPE
 Dic = Word -> Meaning;
 Word = STR;
 Meaning = STR;
 Bool = SYM;
ENDTYPE

FUNC INIT((sigma)): (sigma)
STATE sigma <- [];

FUNC INSWORD(w: Word, m: Meaning, (sigma)): (sigma)
PRE not(EXISTWORD(w))
STATE sigma <- sigma + [w -> m];

FUNC INSDIC(d: Dic, (sigma)): (sigma)
STATE sigma <- sigma + d;

FUNC DELWORD(w: Word, (sigma)): (sigma)
PRE EXISTWORD(w)
STATE sigma <- sigma\{w};

FUNC DEFWORD(w: Word, (sigma)): Meaning
PRE EXISTWORD(w)
RETURN sigma[w];

FUNC EMPTYDIC ((sigma)): Bool
RETURN sigma == [];

FUNC EXISTWORD(w: Word, (sigma)): Bool
RETURN w in dom(sigma);

FUNC WordInv(w: Word|Meaning): Bool
RETURN w != "";
```

B The Application Module Specification

```
FUNC init((ma)):(ma)
STATE
ma<- ModApl(["Dic" -> TypeInfo(<"FF", "Word", "Meaning">, NIL),
             "Word" -> TypeInfo("STR", 'WordInv),
             "Meaning" -> TypeInfo("STR", 'WordInv)
            ],
            [],
            ["INIT" -> 'INIT,
             "INSWORD" -> 'INSWORD,
             "INSDIC" -> 'INSDIC,
             "DELWORD" -> 'DELWORD,
             "DEFWORD" -> 'DEFWORD,
             "EMPTYDIC" -> 'EMPTYDIC,
             "EXISTWORD" -> 'EXISTWORD
            ],
            channel("stsm"),
            channel("smst"),
            channel("smlx")
           );

#include mdic.n
```

C GIs for the Dictionary

```
DefGI Menu
  Declarations
    TYPE DECISON
  Behaviour
    EVSEQ (GInit + GManage + GDefWord)* + End
EndGI

DefGI End
  Declarations
    TYPE DECISION
  Behaviour
EndGI

DefGI GInit
  Declarations
    TYPE SYNTH
  Behaviour
    EXEC INIT()
EndGI

DefGI GManage
  Declarations
    TYPE DECISON
  Behaviour
    EVSEQ (GInsWord + GInsDic + GDelWord)* + End
EndGI

DefGI GInsWord
  Declarations
    TYPE SYNTH
    ARGS w: Word;
         m: Meaning
  Behaviour
    INIT w = "";
         m = ""
    EVSEQ input(w).input(m)
    TRANS input(w): (w != "") && not(EXISTWORD(w)) =>
                           EXCEP w=="" -> out("Empty Word!");
                                 w!="" -> out("Word Exists!")
          OK:
          CANCEL:
```

```
    EXEC INSWORD(w, m)
EndGI

DefGI GInsDic
  Declarations
    TYPE SYNTH
    ARGS d: Dic
  Behaviour
    EVSEQ input(d)
    TRANS CANCEL:
    EXEC INSDIC(d)
EndGI

DefGI GDelWord
  Declarations
    TYPE SYNTH
    ARGS w: Word
    VAR-UI m: Meaning
  Behaviour
    CONTEXT not(EMPTYDIC())
    INIT w = "";
         m = ""
    EVSEQ input(w)
    TRANS input(w): EXISTWORD(pal) => m = DEFWORD(w)
                           EXCEP out("Word does not Exist!")
          OK:
          CANCEL:
    EXEC DELWORD(w)
EndGI

DefGI GDefWord
  Declarations
    TYPE SYNTH
    SUBGI DoDefWord
    VAR-UI m: Meaning
  Behaviour
    CONTEXT not(EMPTYDIC())
    INIT m = ""
    EVSEQ DoDefWord(m)
    TRANS OK:
  SubGI
    DefGI DoDefWord: Meaning
      Declarations
```

```
        TYPE VALSYNTH
        ARGS w: Word
      Behaviour
        CONTEXT not(EMPTYDIC())
        INIT w = ""
        EVSEQ input(w)
        TRANS input(w): EXISTWORD(w) =>
                          EXCEP out("Word does not Exist!")
              OK:
              CANCEL:
        EXEC DEFWORD(w)
    EndGI
EndGI
```

D The Dialogue Controller Specification

```
FUNC init((cd)):(cd)
STATE
let(gi1 = DECISION({"Menu"},
                   NIL,
                   {},
                   {},
                   [],
                   true,
                   <>,
                   OpcDescr(EvId("End", NIL),
                            RepDescr(OpcDescr(EvId("GManage", NIL),
                                              OpcDescr(EvId("GInit", NIL),
                                                       EvId("GDefWord", NIL)
                                                      )
                                             )
                                    )
                           ),
                   [],
                   NIL
                  ),
    gi2 = SYNTH({"End"},
                NIL,
                {},
                {},
                [],
                true,
                <>,
                NIL,
                [],
                NIL
               ),
    gi3 = SYNTH({"GInit"},
                NIL,
                {},
                {},
                [],
                true,
                <>,
                NIL,
                [],
                ExecDescr("INIT", <>)
```

```
            ),
gi4 = DECISION({"GManage"},
               NIL,
               {},
               {},
               [],
               true,
               <>,
               OpcDescr(EvId("GInsWord", NIL),
                        OpcDescr(EvId("GInsDic", NIL),
                                 OpcDescr(EvId("GDelWord", NIL),
                                          EvId("End", NIL)
                                         )
                                )
                       ),
               [],
               NIL
              ),
trans5 = TransDescr(And(Ne("w", StrExp("")),
                         Not(FunCall("EXISTWORD", "Bool", <"w">))
                        ),
                    <>,
                    <ExcepDescr(Eq("w", StrExp("")),
                                <Out("Empty Word!")>
                               ),
                     ExcepDescr(Ne("w", StrExp("")),
                                <Out("Word Exists!")>
                               )
                    >
                   ),
gi5 = SYNTH({"GInsWord"},
            NIL,
            {},
            {},
            ["w"->VarUI("Word"),
             "m"->VarUI("Meaning")
            ],
            true,
            <Atrib("w", ""), Atrib("m", "")>,
            SeqDescr(EvId(NIL, "w"),
                     EvId(NIL, "m")
                    ),
            [EvId(NIL, "w") -> trans5,
                CmdId("$Cancel") -> TransDescr(true, <>, <>),
                CmdId("$OK") -> TransDescr(true, <>, <>)
               ],
               ExecDescr("INSWORD", <"w", "m">)
              ),
    gi6 = SYNTH({"GInsDic"},
               NIL,
               {},
               {},
               ["d"->VarUI("Dic")
               ],
               true,
               <>,
               EvId(NIL, "d"),
               [EvId(NIL, "d") -> TransDescr(true, <>, <>),
                CmdId("$Cancel") -> TransDescr(true, <>, <>)
               ],
               ExecDescr("INSDIC", <"d">)
              ),
    trans7 = TransDescr(FunCall("EXISTWORD", "Bool", <"w">),
                        <Atrib("m", FunCall("DEFWORD", "Meaning", <"w">))>,
                        <ExcepDescr(true,
                                    <Out("Word does not Exist!")>
                                   )
                        >
                       ),
    gi7 = SYNTH({"GDelWord"},
               NIL,
               {},
               {},
               ["w"->VarUI("Word"),
                "m"->VarUI("Meaning")
               ],
               Not(FunCall("EMPTYDIC", "Bool", <>)),
               <Atrib("w", ""), Atrib("m", "")>,
               EvId(NIL, "w"),
               [EvId(NIL, "w") -> trans7,
                CmdId("$Cancel") -> TransDescr(true, <>, <>),
                CmdId("$OK") -> TransDescr(true, <>, <>)
               ],
               ExecDescr("DELWORD", <"w">)
              ),
    gi8 = SYNTH({"GDefWord"},
                NIL,
                {},
                {"DoDefWord"},
                ["m"->VarUI("Meaning")
                ],
                Not(FunCall("EMPTYDIC", "Bool", <>)),
                <Atrib("m", "")>,
                EvId("DoDefWord", "m"),
                [EvId("DoDefWord", "m") -> TransDescr(true, <>, <>),
                 CmdId("$OK") -> TransDescr(true, <>, <>)
                ],
                NIL
               ),
    trans9 = TransDescr(FunCall("EXISTWORD", "Bool", <"w">),
                        <>,
                        <ExcepDescr(true,
                                    <Out("Word does not Exist!")>
                                   )
                        >
                       ),
    gi9 = VALSYNTH({"DoDefWord"},
                   "Meaning",
                   {},
                   {},
                   ["w"->VarUI("Word")
                   ],
                   true,
                   <Atrib("w", "")>,
                   EvId(NIL, "w"),
                   [EvId(NIL, "w") -> trans9],
                   ExecDescr("DEFWORD", <"w">)
                  )
   )
in cd <- CD(["Menu" -> gi1,
             "End" -> gi2,
             "GInit" -> gi3,
             "GManage" -> gi4,
             "GInsWord" -> gi5,
             "GInsDic" -> gi6,
             "GDelWord" -> gi7,
             "GDefWord" -> gi8,
             "DoDefWord" -> gi9,
             "Dic" -> FFSYNTH("Dic")
```

```
 ],
 [],
 channel("stsm"),
 channel("smst"),
 channel("lxst"),
 channel("stlx")
);
```

E The Presentation Module Specification

```
FUNC init((ma)):(ma)
STATE
let(global = ["GInsWord"  -> GlobalInfo(["w" -> "Word", "m" -> "Meaning"],
                                         {}
                                        ),
              "GInsDic"   -> GlobalInfo(["d" -> "Dic"],
                                         {}
                                        ),
              "GDelWord"  -> GlobalInfo(["w" -> "Word", "m" -> "Meaning"],
                                         {}
                                        ),
              "GDefWord"  -> GlobalInfo(["m" -> "Meaning"],
                                         {}
                                        ),
              "DoDefWord" -> GlobalInfo(["w" -> "Word"],
                                         {}
                                        ),
              "Dic"       -> GlobalInfo(["dom" -> "Word", "ran" -> "Meaning"],
                                         {}
                                        )
             ],
    gis1 = ["Menu"    -> MENU(LxDef("", "Principal", ""), "",
                               <Option(EvId("GInit", NIL), "Dicionario Vazio"),
                                Option(EvId("GManage", NIL), "Gestao"),
                                Option(EvId("GDefWord", NIL), "Consultar"),
                                Option(EvId("End", NIL), "Sair")
                               >
                              ),
            "End"      -> DB(LxDef("", "Fim", ""),
                              [],
                              [],
                              []
                             ),
            "GInit"     -> DB(LxDef("", "Vazio", ""),
                              [],
                              [],
                              []
                             ),
            "GManage"  -> MENU(LxDef("", "Gestao", ""), "",
                                <Option(EvId("GInsWord", NIL), "Inserir Palavra"),
                                 Option(EvId("GInsDic", NIL), "Inserir Dicionario"),
                          Option(EvId("GDelWord", NIL), "Remover Palavra"),
                          Option(EvId("End", NIL), "Sair")
                         >
                        ),
        "GInsWord" -> DB(LxDef("", "Insercao", ""),
                        ["w" -> HimSelf(LxDef("", "Palavra", "")),
                         "m" -> HimSelf(LxDef("", "Significado", ""))
                        ],
                        [],
                        [CmdId("$Cancel") -> LxDef("", "Cancelar", ""),
                         CmdId("$OK")     -> LxDef("", "Terminar", "")
                        ]
                       ),
        "GInsDic"  -> DB(LxDef("", "Insercao de Dicionario", ""),
                        ["d" -> "Dic"
                        ],
                        [],
                        [CmdId("$Cancel") -> LxDef("", "Cancelar", "")
                        ]
                       ),
        "GDelWord"  -> DB(LxDef("", "Remocao", ""),
                         ["w" -> HimSelf(LxDef("", "Palavra", "")),
                          "m" -> HimSelf(LxDef("", "Significado", ""))
                         ],
                         [],
                         [CmdId("$Cancel") -> LxDef("", "Cancelar", ""),
                          CmdId("$OK")     -> LxDef("", "Terminar", "")
                         ]
                        ),
        "GDefWord"  -> DB(LxDef("", "Consulta", ""),
                         ["m" -> HimSelf(LxDef("", "Significado", ""))
                         ],
                         [EvId("DoDefWord", "m") ->
                                 DBGIView(LxDef("", "Ler Palavra", ""),
                                          true
                                         )
                         ],
                         [CmdId("$OK")     -> LxDef("", "Terminar", "")
                         ]
                        ),
        "DoDefWord" -> DB(LxDef("", "Ler Palavra", ""),
                         ["w" -> HimSelf(LxDef("", "Palavra", ""))],
                         [],
                           []
                         ),
        "Dic"       -> DB(LxDef("", "Dicionario", ""),
                         ["dom" -> HimSelf(LxDef("", "Palavra", "")),
                          "ran" -> HimSelf(LxDef("", "Significado", ""))
                         ],
                         [],
                         [CmdId("$Cancel") -> LxDef("", "Cancelar", ""),
                          CmdId("$OK")     -> LxDef("", "Terminar", ""),
                          CmdId("$Up")     -> LxDef("", "Anterior", ""),
                          CmdId("$Down")   -> LxDef("", "Seguinte", ""),
                          CmdId("$Del")    -> LxDef("", "Apagar", ""),
                          CmdId("$New")    -> LxDef("", "Vazio", ""),
                         ]
                        )
       ],
gis2 = ["Menu"    -> MENU(LxDef("", "Main", ""), "",
                          <Option(EvId("GInit", NIL), "Empty Dictionary"),
                           Option(EvId("GManage", NIL), "Managment"),
                           Option(EvId("GDefWord", NIL), "Search"),
                           Option(EvId("End", NIL), "Quit")
                          >
                         ),
        "End"      -> DB(LxDef("", "End", ""),
                         [],
                         [],
                         []
                        ),
        "GInit"     -> DB(LxDef("", "Empty", ""),
                        [],
                        [],
                        []
                       ),
        "GManage"  -> MENU(LxDef("", "Managment", ""), "",
                           <Option(EvId("GInsWord", NIL), "Insert Word"),
                            Option(EvId("GInsDic", NIL), "Insert Dictionary"),
                            Option(EvId("GDelWord", NIL), "Delete Word"),
                            Option(EvId("End", NIL), "Quit")
                           >
                          ),
        "GInsWord" -> DB(LxDef("", "Insert", ""),
                         ["w" -> HimSelf(LxDef("", "Word", "")),
                          "m" -> OptionMenu(["Word0" -> "",
                                     "Word1" -> "This",
                                     "Word2" -> "is",
                                     "Word3" -> "an",
                                     "Word4" -> "Example"
                                    ],
                                    LxDef("", "Meaning", "")
                                   )
                 ],
                 [],
                 [CmdId("$Cancel") -> LxDef("", "Cancel", ""),
                  CmdId("$OK")     -> LxDef("", "OK", "")
                 ]
                ),
"GInsDic"  -> DB(LxDef("", "Insert Dictionary", ""),
                 ["d" -> "Dic"
                 ],
                 [],
                 [CmdId("$Cancel") -> LxDef("", "Cancel", "")
                 ]
                ),
"GDelWord"  -> DB(LxDef("", "Remove", ""),
                  ["w" -> HimSelf(LxDef("", "Word", "")),
                   "m" -> HimSelf(LxDef("", "Meaning", ""))
                  ],
                  [],
                  [CmdId("$Cancel") -> LxDef("", "Cancel", ""),
                   CmdId("$OK")     -> LxDef("", "OK", "")
                  ]
                 ),
"GDefWord"  -> DB(LxDef("", "Search", ""),
                  ["m" -> HimSelf(LxDef("", "Meaning", ""))
                  ],
                  [EvId("DoDefWord", "m") ->
                          DBGIView(LxDef("", "Read Word", ""),
                                   true
                                  )
                  ],
                  [CmdId("$OK")     -> LxDef("", "OK", "")
                  ]
                 ),
"DoDefWord" -> DB(LxDef("", "Read Word", ""),
                  ["w" -> HimSelf(LxDef("", "Word", ""))],
                  [],
                              []
                             ),
            "Dic"       -> DB(LxDef("", "Dictionary", ""),
                             ["dom" -> HimSelf(LxDef("", "Word", "")),
                              "ran" -> HimSelf(LxDef("", "Meaning", ""))
                             ],
                             [],
                             [CmdId("$Cancel") -> LxDef("", "Cancel", ""),
                              CmdId("$OK")     -> LxDef("", "OK", ""),
                              CmdId("$Up")     -> LxDef("", "Up", ""),
                              CmdId("$Down")   -> LxDef("", "Down", ""),
                              CmdId("$Del")    -> LxDef("", "Delete", ""),
                              CmdId("$New")    -> LxDef("", "New", ""),
                             ]
                            )
           ]
   )
in ma <- ModApr(global,
                ["View1" -> gis1, "View2" -> gis2],
                "View1",
                "Menu",
                channel("stlx"),
                channel("smlx"),
                channel("lxst"),
                channel("stsm")
               );
```

F Types For Writing Code

```
  Code = Exp-list;
  Exp = Atrib | ProcCall | Out | If | While;
; --- Assignment ---
  Atrib :: VAR: VarId
           EXP: TypedExp;
  TypedExp = IntExp | StrExp | BoolExp | Value; /* Mais Tipos ... */
  IntExp = Add | Sub | Div | Mul | Mod | Ident | FunCall | INT;
  Add :: IntArgs;
  Sub :: IntArgs;
  Div :: IntArgs;
  Mul :: IntArgs;
  Mod :: IntArgs;
  IntArgs :: ARG1: IntExp
             ARG2: IntExp;
  StrExp :: DREF: STR;
  BoolExp = Eq | Ne | Ge | Le | Gt | Lt | Ident | FunCall | Bool | Not |
            And | Or;
  Eq :: BoolArgs;
  Ne :: BoolArgs;
  Ge :: BoolArgs;
  Le :: BoolArgs;
  Gt :: BoolArgs;
  Lt :: BoolArgs;
  And :: BoolArgs;
  Or :: BoolArgs;
  Not :: ARG1: BoolExp;
  BoolArgs :: ARG1: TypedExp
              ARG2: TypedExp;
; --- Calling ---
  FunCall :: FU: Ident
             TYP: Ident
             ARGS: VarId-list;
  Ident = STR;
  ProcCall :: OPR: Ident
              ARGS: VarId-list;
  Call = ProcCall | FunCall;
; --- Out ---
  Out = STR;
; --- If ---
  If :: G: BoolExp
        T: Code
        El: Code;
; --- While ---
  While :: G: BoolExp
           C: Code;
```

G An Application Model with Tuples

```
FUNC init((ma)):(ma)
STATE
ma<- ModApl(["CliInfo" -> TypeInfo(<"TUP",
                                    <"N", "Nome">,
                                    <"M", "Morada">,
                                    <"F", "Alugados">
                                   >, NIL
                                  ),
             "Alugados" -> TypeInfo(<"FF", "CodFilme", "Data">, NIL),
             "FilmInfo" -> TypeInfo(<"TUP",
                                     <"N", "Nome">,
                                     <"R", "Realizador">,
                                     <"TOTAL", "Quantidade">,
                                     <"STOCK", "Quantidade">
                                    >, 'aplinvFilmInfo
                                   ),
             "Data" -> TypeInfo(<"TUP",
                                 <"D", "Dia">,
                                 <"M", "Mes">,
                                 <"A", "Ano">
                                >, 'aplinvData
                               ),
             "CodCliente" -> TypeInfo("INT", NIL),
             "CodClientes" -> TypeInfo(<"SET", "CodCliente">, NIL),
             "Nome" -> TypeInfo("STR", NIL),
             "Morada" -> TypeInfo("STR", NIL),
             "CodFilme" -> TypeInfo("INT", NIL),
             "Realizador" -> TypeInfo("STR", NIL),
             "Quantidade" -> TypeInfo("INT", NIL),
             "Dia" -> TypeInfo("INT", NIL),
             "Mes" -> TypeInfo("INT", NIL),
             "Ano" -> TypeInfo("INT", NIL)
            ],
            [], ["INIT" -> 'aplINIT,
                 "ALTERA_DATA" -> 'aplALTERAR_DATA,
                 "INS_CLI" -> 'aplINS_CLI,
                 "INS_FILM" -> 'aplINS_FILM,
                 "CONS_CLI" -> 'aplCONS_CLI,
                 "CONS_FILM" -> 'aplCONS_FILM,
                 "REM_CLI" -> 'REM_CLI,
                 "REM_FILM" -> 'REM_FILM,
                 "ALUGA" -> 'aplALUGA,
                 "DEVOLVE" -> 'aplDEVOLVE,
                 "CALOTEIROS" -> 'CALOTEIROS,
                 "EXISTCLI" -> 'EXISTCLI,
                 "EXISTFILM" -> 'EXISTFILM,
                 "EMPTYCLI" -> 'EMPTYCLI,
                 "EMPTYFILM" -> 'EMPTYFILM,
                 "STOCKOK" -> 'STOCKOK,
                 "CLIOK" -> 'CLIOK,
                 "ALUGOU" -> 'ALUGOU,
                 "FILMOK" -> 'aplFILMOK
                ], channel("c1"), channel("c2"), channel("c3"));


; Operations of the Application Model that make the translation of
; user types and call the application operations

FUNC aplinvFilmInfo(fi: X): SYM
RETURN invFilmInfo(refFilmInfo(fi));

FUNC aplinvData(dt: X): SYM
RETURN invData(refData(dt));

FUNC aplINIT(dt: Data, (video)): (video)
RETURN INIT(refData(dt));

FUNC aplALTERAR_DATA(dt: Data, (video)): (video)
RETURN ALTERAR_DATA(refData(dt));

FUNC aplINS_CLI(cod: CodCliente, info: CliInfo, (video)): (video)
RETURN INS_CLI(cod, refCliInfo(info));

FUNC aplINS_FILM(cod: CodFilme, info: FilmInfo, (video)): (video)
RETURN INS_FILM(cod, refFilmInfo(info));

FUNC aplCONS_CLI(cod: CodCliente, (video)): CliInfo
RETURN retCliInfo(CONS_CLI(cod));

FUNC aplCONS_FILM(cod: CodFilme, (video)): CliInfo
RETURN retFilmInfo(CONS_FILM(cod));

FUNC aplALUGA(codc: CodCliente, codf: CodFilme, (video)): Data (video)
RETURN retData(ALUGA(codc, codf));
```

```
FUNC aplDEVOLVE(codc: CodCliente, codf: CodFilme, (video)): Data (video)
RETURN retData(DEVOLVE(codc, codf));

FUNC aplFILMOK(info: FilmInfo, (video)): SYM
RETURN FILMOK(refFilmInfo(info));


; Functions of the Application Model that tranlate the user types

FUNC refData(dt: X): Data
RETURN Data(dt["D"], dt["M"], dt["A"]);

FUNC retData(dt: Data): X
RETURN ["D"->D(dt), "M"->M(dt), "A"->A(dt)];

FUNC refCliInfo(dt: X): CliInfo
RETURN
progn( princ(dt, "\n"),
CliInfo(dt["N"], dt["M"], refAlugados(dt["F"]))
);

FUNC retCliInfo(dt: CliInfo): X
RETURN ["N"->N(dt), "M"->M(dt), "F"->retAlugados(F(dt))];

FUNC refFilmInfo(dt: X): FilmInfo
RETURN FilmInfo(dt["N"], dt["R"], dt["TOTAL"], dt["STOCK"]);

FUNC retFilmInfo(dt: FilmInfo): X
RETURN ["N"->N(dt), "R"->R(dt), "TOTAL"->TOTAL(dt), "STOCK"->STOCK(dt)];

FUNC refAlugados(dt: X): Alugados
RETURN [cf -> refData(dt[cf]) | cf<-dom(dt)];

FUNC retAlugados(dt: CliInfo): X
RETURN [cf -> retData(dt[cf]) | cf<-dom(dt)];

#include video.n
```