VisualLISA

Behind the Scenes

Universidade do Minho, Departamento de Informática
Nuno Oliveira, 2008

Outline

- Introduction
- Visual Language Grammar for VisualLISA
- DEViL
- Implementation
- Results

Introduction

- VisualLISA is a project under the UCE-15 of MSc in Informatics' 2nd year;
- VisualLISA is a graphical interface for the compilers-compiler LISA.
- VisualLISA will be used to edit visually LISA specifications.
- Can't VisualLISA be a generic VisualAG tool?

Visual Language Grammar

- Differences between visual and textual Grammars?

  + Operators to relate two or more symbols in space;
  + Attributes to compute symbol's position;

  – Notion of order of symbol's occurrence;

Visual Language Grammar

(…)

AG_ELEM → LEFT_SYMBOL | NON_TERMINAL | TERMINAL | SYNT_ATTRIBUTE
| INH_ATTRIBUTE | TREE_BRANCH | IV_ATTRIBUTE
| SYNT_CONNECTION | INH_CONNECTION| IV_CONNECTION

(…)

TERMINAL → over(**rectangle,text**)

(…)

TREE BRANCH → points_from(points_to(**line**,~TERMINAL),
~LEFT SYMBOL)
|points_from(points_to(**line**,~NONTERMINAL),
~LEFT SYMBOL)

DEViL – Development Environment for Visual Languages

Bad Aspects

- Complex Installation;
- Disperse Documentation and written in German;
- Generated Editor is only compatible with the SO where it is Generated;
- ...

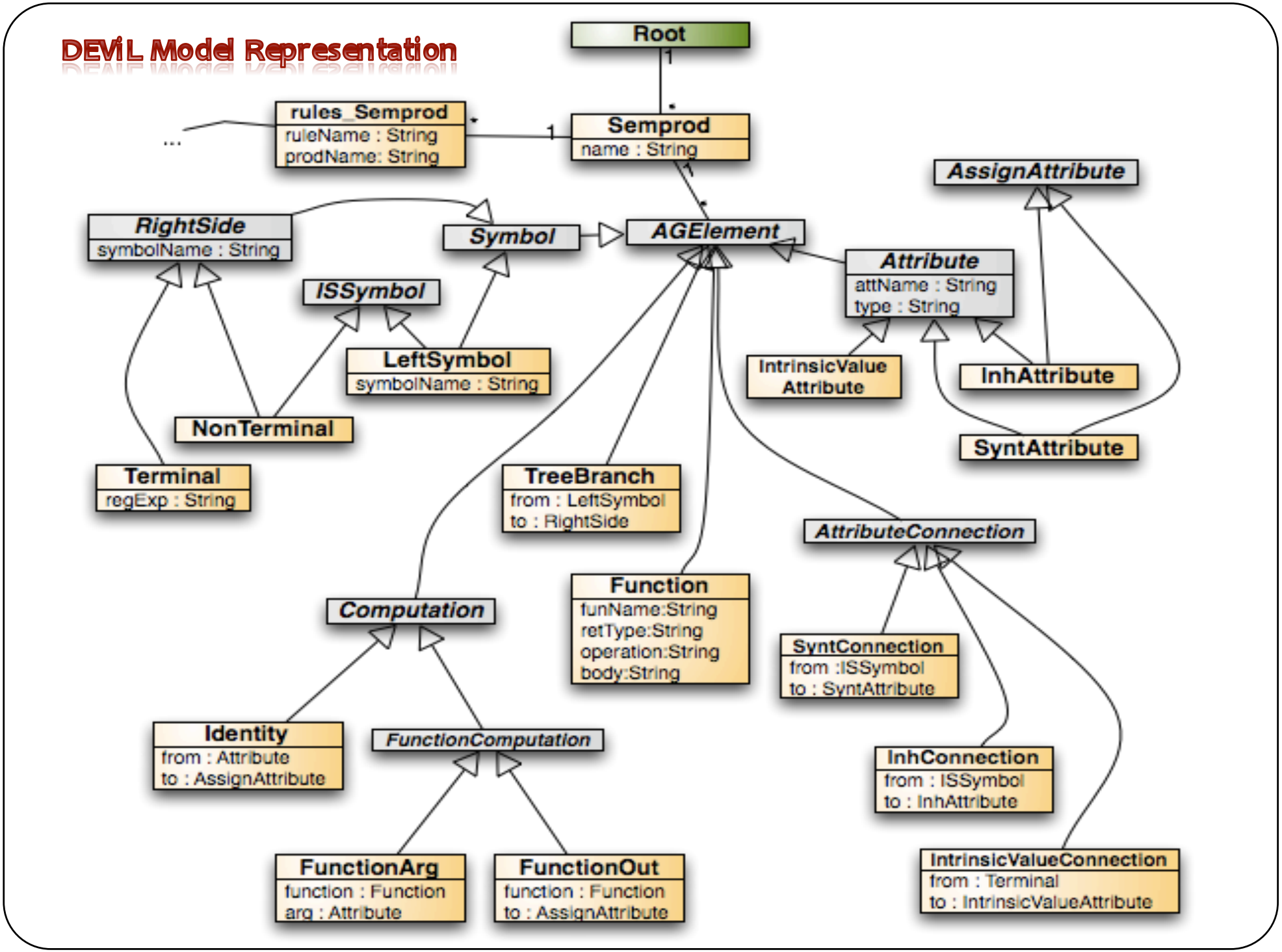
Good Aspects

- Very good support;
- Too much examples, addressing several DEViL features;
- Exists for MacOS, Windows and Linux
- ...

Implementation

- Visual Language Grammar into DEViL Model Representation
- Editor Generation
- Semantic Analysis
- Code Generation

DEViL Model Representation

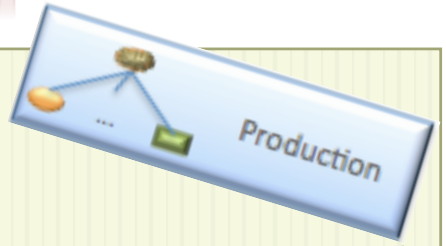


Editor Generation

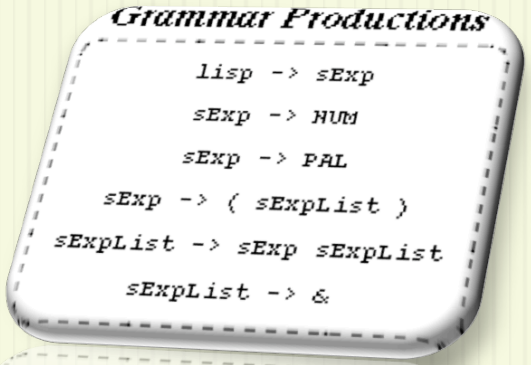
- Definition of Models to create different views:
  - Define buttons' Look & Feel;
  - Define buttons' behavior;
  - ...
- Definition of Canvas' Look & Feel:
  - Assign Visual Patterns to each grammar Symbol or Attribute;
  - Define computations to implement the Visual Pattern assigned (*ex: SYNT.drawing, SYNT.text, ...*);
  - Calculate values for Symbols' attributes;

Editor Generation

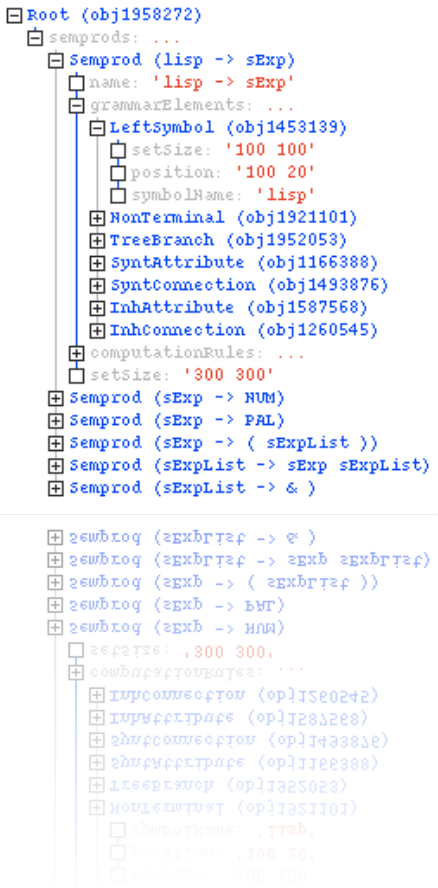
```
VIEW rootView ROOT Root {
    BUTTON IMAGE "img::btnSemprod"
        INSERTS Semprod
            INFO "Inserts a new Grammar Production";
}
```




```
SYMBOL rootView_Root INHERITS VPRootElement, VPForm
COMPUTE
        SYNT.drawing = ADDROF(rootViewDrawing);
END;

SYMBOL rootView_Root_semprods INHERITS
VPFormElement, VPSimpleList
COMPUTE
        SYNT.formElementName = "productions";
END;
```



Semantic Analysis



- DEViL builds a tree representation for our specification, based on the grammar specified;
- DEViL gives us a sort of a tree walker to perform computations given a tree context;
- The tree walker is named *addCheck*
- A tree context is any symbol or attribute in the tree representation;

Semantic Analysis

Semantic Contraint: A production must have one and only one root symbol

```
checkutil::addCheck Semprod {
        set n [llength [c::getList {$obj.grammarElements.CHILDREN[LeftSymbol]}]]
        set symbName [c::get {$obj.name.VALUE}]

        if { $n == 0 } {
                return "Production '$symbName' must have one Root symbol!"
        } elseif {$n > 1} {
                return "Production '$symbName' must have only one Root symbol!"
        }
        return ""
}
```

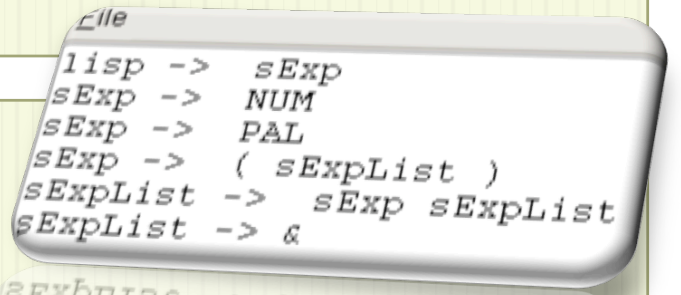
Code Generation

- Definition of code templates with IPTG/PTG files;
- Definition of code generation using LIDO files:
  - Use CONSTITUENTS to access/refer to symbols far down on the tree from the actual context;
  - Use INCLUDING* to access/refer to symbols far up on the tree from the actual context;
  - Use of SYNT attributes to calculate code or parts of code;

* INCLUDING reduces the need of using inherited attributes;

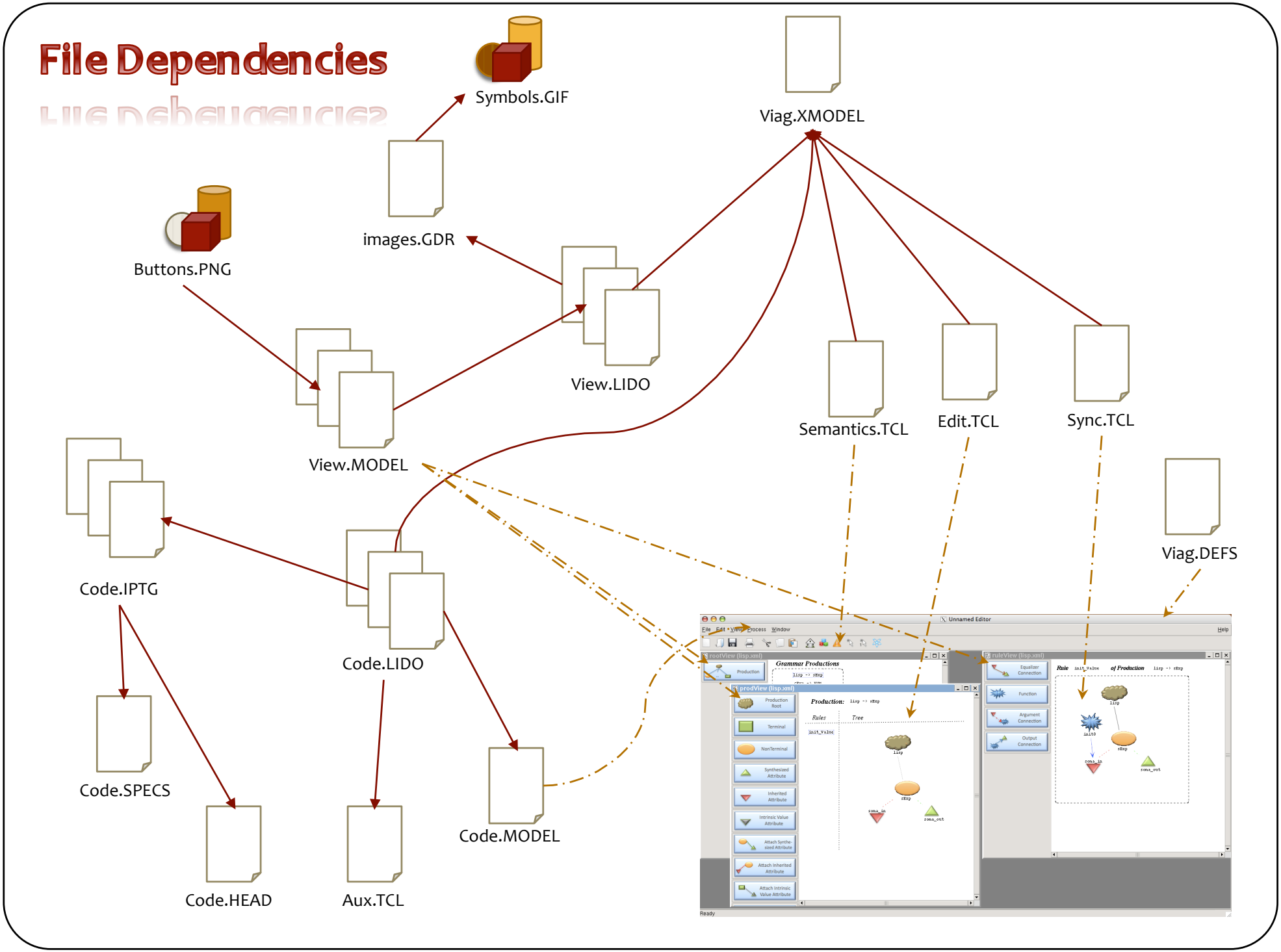
Code Generation

```
bnfProd(lhs, rhs):
   [lhs] -> [rhs]
```

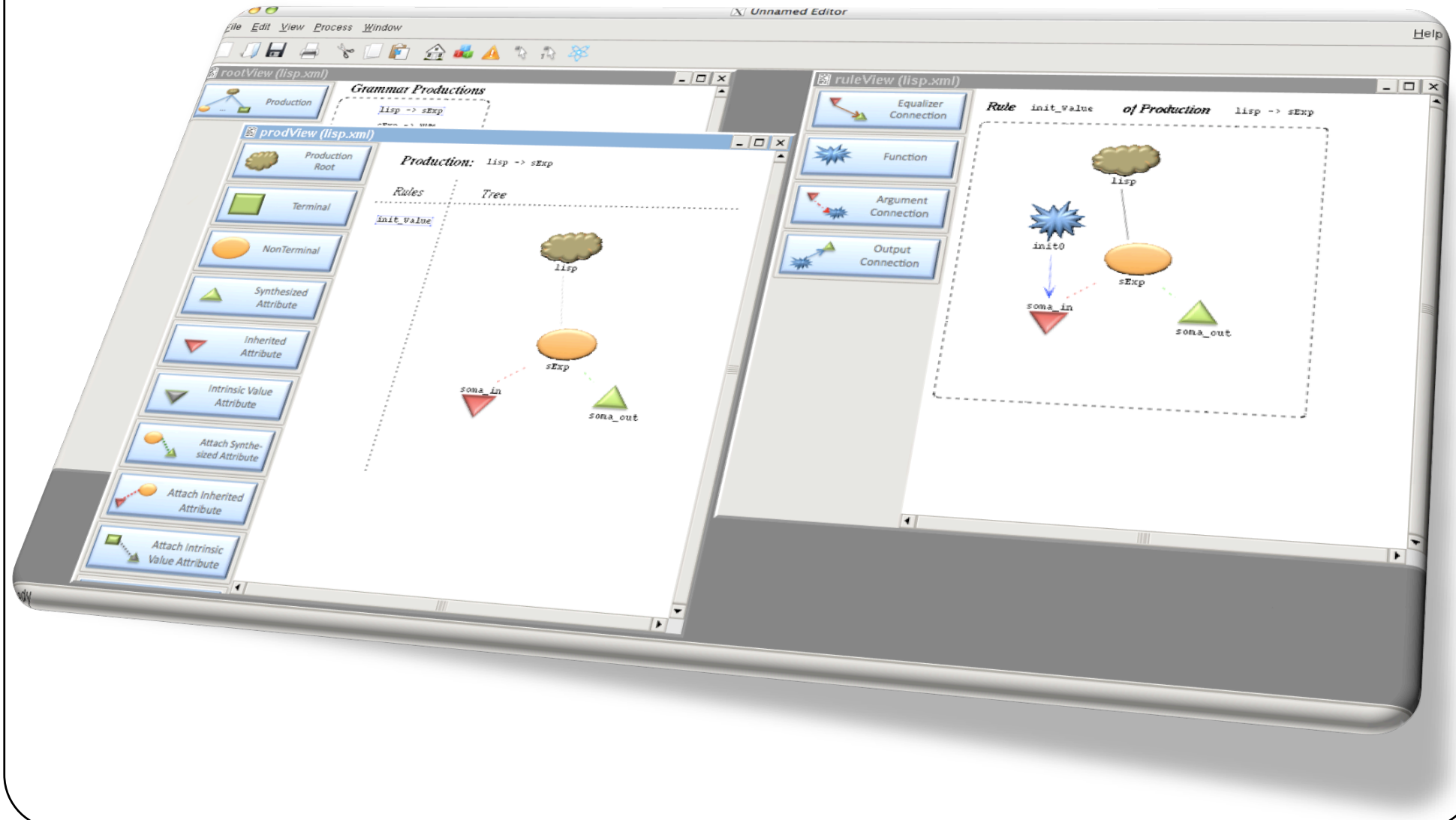



```
SYMBOL bnfgen_Semprod: bnfLHS : PTGNode;
SYMBOL bnfgen_Semprod: bnfRHS : PTGNode;
SYMBOL bnfgen_Semprod: bnfCode : PTGNode;
SYMBOL bnfgen_Semprod
COMPUTE
      SYNT.bnfLHS = CONSTITUENTS bnfgen_LeftSymbol.pers_symbolName
                        WITH(PTGNode, PTGNewLineSeq, PTGAsIs, PTGNull);

      SYNT.bnfRHS = PTGAsIs(VLString(SELECT(vlList(
                        "printBNFOrderedRHSElements",THIS.objId),eval())));

      SYNT.bnfCode = PTGbnfProd(THIS.bnfLHS, THIS.bnfRHS);
END;
```

File Dependencies



The Editor


Unnamed Editor
File Edit View Process Window
Help
rootView (lisp.xml)
Production
Grammar Productions
lisp -> sExp
prodView (lisp.xml)
Production Root
Terminal
NonTerminal
Synthesized Attribute
Inherited Attribute
Intrinsic Value Attribute
Attach Synthe-sized Attribute
Attach Inherited Attribute
Attach Intrinsic Value Attribute
Production: lisp -> sExp
Rules
Tree
init_value
lisp
sExp
soma_in
soma_out
ruleView (lisp.xml)
Equalizer Connection
Function
Argument Connection
Output Connection
Rule init_value of Production lisp -> sExp
lisp
init0
sExp
soma_in
soma_out

THE END

Universidade do Minho, Departamento de Informática
Nuno Oliveira, 2008