VisualLISA: a Visual Interface for an Attribute Grammar based Compiler-Compiler*

Maria João Varanda Pereira[1], Marjan Mernik[2] Daniela da Cruz[3], and Pedro Rangel Henriques[3]

[1] Polytechnic Institute of Bragança
Campus de Sta. Apolónia, Apartado 134 - 5301-857, Bragança, Portugal
mjoao@ipb.pt
[2] University of Maribor
Faculty of Electrical Engineering and Computer Science
Smetanova ul. 17, 2000 Maribor, Slovenia
marjan.mernik@uni-mb.si
[3] University of Minho - Department of Computer Science,
Campus de Gualtar, 4715-057, Braga, Portugal
{danieladacruz,prh}@di.uminho.pt

Abstract. The research work that we discuss in this position paper, is concerned with the implementation of a visual front-end for LISA tool in order to make easier and more attractive the work of writing an attribute grammar for a new language. LISA (Language Implementation System based on Attribute grammars) is a compiler-compiler, or a system that generates automatically a compiler/interpreter from a formal language specification based on attribute grammars. The main idea is to design a visual language to draw derivation rules in conjunction with the associated semantic rules and to develop a visual compiler to transform that graphical representation into LISA notation.

KEYWORDS: visual programming languages, compiler-compiler tools

1 Introduction

LISA is a compiler generator based on object-oriented attribute grammars developed at University of Maribor [1, 2]. It can be regarded also as a generic interactive environment [3] for programming language development since from the formal language specifications of a particular programming language LISA produces a set of related tools. LISA and the generated environment are written in Java which enables high portability to different platforms [4].

An Attribute Grammar (AG) can be seen as a generalization of Context-Free Grammar (CFG), in which each symbol has an associated set of attributes with semantic information, and each production have an associated set of semantic

* This work is supported by a grant for a Bilateral Collaborative project between Portugal and Slovenia.

rules with attribute computation (in LISA the semantic rules are Java assignment statements). Being a truly attribute grammar based compiler generator, LISA provide us a system capable to develop a language using synthesized and inherited attributes.

LISA specification language provides constructions for: regular expression definitions (lexical part); attribute definitions; and grammar rule definitions, which are generalized syntax rules (described using a variant of the BNF notation) that encapsulate semantic rules and methods (written in Java).

As we want to design a new visual notation for writing grammar rules, it is important to notice that LISA rules are written in a block { } which start with keyword rule followed by the rule name. Inside brackets appear the rule syntactic definition (in BNF notation), and after the reserved word compute appears a new block { } that contains the Java assignments to compute the output attributes of that rule.

```
rule RuleName {
     NAME ::= DEFINITION compute {<attribute computation>};
     }
```

Besides that, LISA derives other tools: editors to help the final users to create and modify source programs; inspectors that are useful to understand the behavior or debug the generated language processor; visualizers/animators, similar to inspectors, useful to understand the meaning of the source program that is being processed. So we can say that LISA is an interactive and visual environment for both the generation and testing of language processing tools — and it is precisely that kind of interface that we want to enhance and extend.

In our work, we want to upgrade LISA tool allowing the visual construction of attribute grammar productions. Visual languages play an important role in software specification specially in specific domains. The very well known Attribute Grammar Specification [5] can be seen as an interesting case study for the creation of a new visual programming language.

2 Our proposal: VisualLISA

When we create a new visual language [6] we must assure that: it has expressiveness; it is intuitive; it is easy to learn and to use; it avoids ambiguity, syntax and semantic errors; it allows program dependencies visibility; it allows program structure visibility; it allows program logic visibility; and it reduces scalability problems.

A processor for a visual language [7] consists of a graphical front-end that analyzes and transforms visual programs. Therefore, the construction of a visual language processor requires a wide range of conceptual and technical knowledge.

In [6] the authors define the static representation of a visual program as the set of every item of information about a program that can be displayed simultaneously on a screen. But a visual programming environment must also

allow dynamic navigational devices (menus, scroll bars, etc.) and animations, sound annotations and other alternative views.

We intend to create a new *visual programming language environment* that will be a front-end for LISA system. The VisualLISA architecture can be seen in Fig. 1.

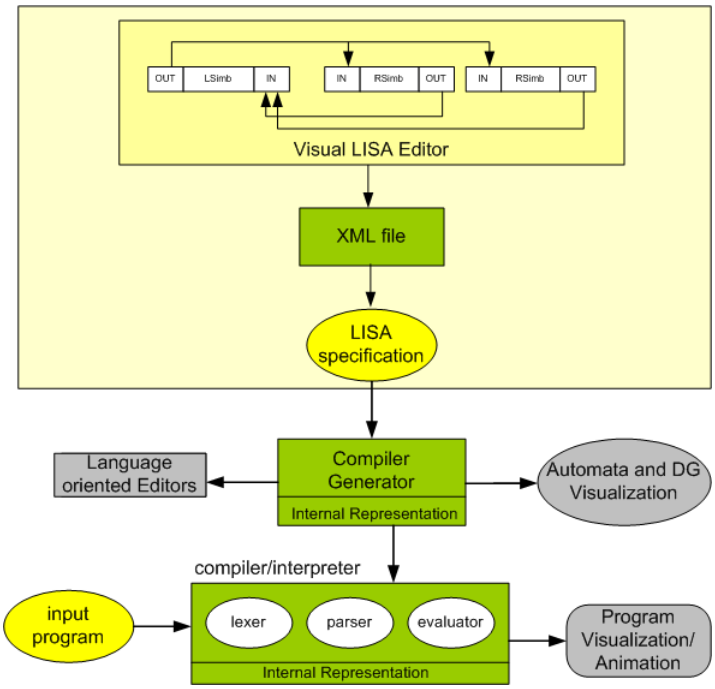

Fig. 1. Architecture of VisualLISA

According to Fig. 1, LISA system generates some visualizations related to the compiler construction and other visualizations related with the program that is compiled by that compiler. On the other hand, the new architectural component that we have to implement is a graphical editor that offers a set predefined icons (elementary drawings) that are the terminal symbols of the new visual language, which can be combined just in some specific ways defined by the visual grammar syntactic rules. That graphical editor, after validate the visual specification (the spatial composition of those icons) will generate an XML file that is a textual description of that graphical sentence.

This XML file is compiled into a valid LISA specification, that is an attribute grammar written in LISA standard textual notation. Then, after that translation, all LISA features can be used to build the compiler and all other language-based tools. Notice that we have adopted XML as an intermediary language (the

interface between the new graphical editor and the traditional LISA front-end). We could generate directly LISA specification but the generation of a XML file increases the versatility of the system allowing a functional separation between the visual editor and the compiler-compiler tool. A XML specification is simple, clear, expressive, easy to generate, and easy to process[4].

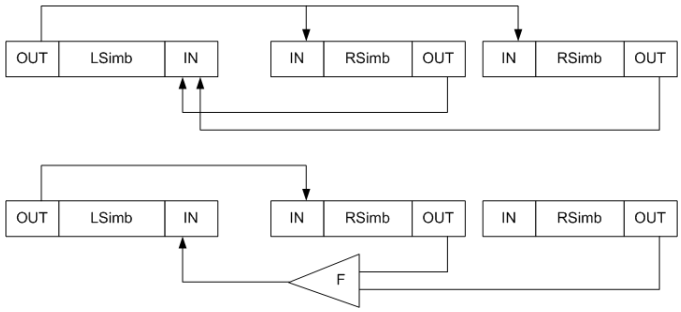

Fig. 2. Production Graphical Representation

The new VisualLISA language (designed to allow the specification of attribute grammar rules) includes icons to specify symbols, synthesized and inherited attributes (associated with those symbols) , syntactic dependencies between symbols, and attribute evaluation rules. Fig. 2 sketches what a VisualLISA grammar rule should look like. That figure contains two sample rules; the second of them includes the use of a function F() to compute a synthesized attribute of the root (the left hand symbol). This function `F` will be probably defined using a textual editor.

As a simple example, consider the following grammar rule (`init`) written in standard LISA notation:

```
rule init
     { STUDENT ::= #NAME #AGE compute { STUDENT.n= #NAME.value();
                                        STUDENT.a= #AGE.value();};
     }
```

Its representation in VisualLISA can be seen in Fig. 3.

In Fig. 3, the symbol `STUDENT` of the production left hand side has two attributes `n` and `a`. The attribute `n` receives the value of `#NAME` symbol and the attribute `a` receives the value of `#AGE` symbol.

To develop VisualLISA, i.e., define this new visual language and build the above referred graphical editor we will, of course, use a visual compiler-compiler. After looking for *tools for visual language implementation*, we found (among many

[4] For those reasons, XML is, nowadays, the most popular and widely used information/knowledge representation schema to support inter-operability.

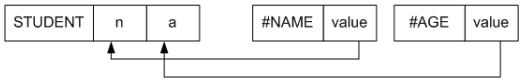

Fig. 3. Graphical Representation of the Student Production

other prototypes) two important systems: VLDesk [8] based on VLCC tool; and Devil [9] based on Eli tool.

Devil [7] is based in attribute grammars, it has a clear separation between abstract grammar, code generation and graphical specification. It provides very good functionalities for specify attribute evaluation and pretty-printing.

At moment we are working on a first prototype of a graphical editor for VisualLISA, using Devil and taking into account the visual representation exemplified in Fig. 2. To be more specific, we present in the rest of the section small extracts taken from Devil specification files that are needed to generate automatically the desired VisualLISA editor.

In the abstract grammar below, we are specifying that our visual grammar can have several productions. The components of each production are: `name`, `leftside` (one left hand side symbol), `rightside` (several right hand side symbols) and `computations`. The symbols `LSimb`, `RSimb` and `Computation` will be also defined in this file.

```
CLASS Root {
    semprods: SUB Semprod*;
}
CLASS Semprod {
    name:  VAL VLString;
    leftside: REF LSimb;
    rightside: SUB RSimb*;
    computations: SUB Computation*;
    setsize: VAL VLPoint INIT "600 600" EDITWITH "None";
}
...
```

The code generation is specified using PTG tool that allows the use of combinator functions specified in another file. The attribute `code` (associated to the root) is synthesized from other `code` attribute values.

```
ATTR code: PTGNode;

SYMBOL codegen_Root
  COMPUTE
    SYNT.code = PTGOutXml(CONSTITUENTS codegen_Semprod.code
                          WITH (PTGNode, PTGDoubleNewLineSeq, IDENTICAL,PTGNull));
    SYNT.fsimbattrDefined = CONSTITUENTS codegen_TSimbattr.tsimbDefined;

    PTGOutWindow("Generated Specification", THIS.code);
```

```
END;
...
```

The graphical specification is used to associate the graphical elements to the abstract grammar symbols. These drawings will be combined based on the abstract grammar productions and on some predefined entities to indicate the spacial position of these elements. In this example, the figure `ProductionDrawing` is been associated to the symbol `Semprod`.

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

3 Conclusion

Along the paper we introduced a ongoing research work on specification and automatic generation of Visual Language Processors.

We have pointed out the concrete use of Devil tool (an Eli based generator of graphical editors for visual language) applied to the development of VisualLISA, a graphical front-end for LISA, a classic Compiler Generator (based on object-oriented attribute grammar language specifications).

That project, that is still on the beginning, requires the formal definition of the new visual language to describe the attribute grammar syntactic-semantic rules, and the systematic and automatic derivation of the graphical front-end to support the editing of the visual sentences and their translation to a textual format recognizable by LISA system.

We firmly believe that VisualLISA will be a more intuitive and user-friendly notation to write attribute grammars, and that its inclusion on LISA environment will extend smoothly the present graphical (windows based) interface available for the development and testing of grammar-based language processing tools.

Now we have to complete the specification files, generate the graphical editor and test the usability of this new visual environment.

References

1. Mernik, M., Korbar, N., Žumer, V.: LISA: A tool for automatic language implementation. ACM SIGPLAN Notices **30**(4) (April 1995) 71–79
2. Mernik, M., Lenič, M., Avdičaušević, E., Žumer, V.: A reusable object-oriented approach to formal specifications of programming languages. L'Objet **4**(3) (1998) 273–306
3. Mernik, M., Lenič, M., Avdičaušević, E., Žumer, V.: LISA: An Interactive Environment for Programming Language Development. In Horspool, N., ed.: 11th International Conference on Compiler Construction. Volume 2304., Lecture Notes in Computer Science, Springer-Verlag (2002) 1–4

4. Mernik, M., Novak, U., Avdičaušević, E., Lenič, M., Žumer, V.: Design and implementation of simple object description language. In: ACM Symposium on Applied Computing, SAC'2001. (2001) 590–594
5. Aaltonen, T., Helin, J.: Attribute grammars - a formalism for structure-directed computation (2004)
6. Yang, S., Burnett, M., DeKoven, E., Zloof, M.: Representation design benchmarks: a design-time aid for vpl navigable static representations. Journal of Visual Languages and Computing **8**(5/6) (1997) 563–599
7. Kastens, U., Schmidt, C.: Vl-eli: A generator for visual languages. Electronic Notes in Theoretical Computer Science **65**(3) (2002)
8. Costagliola, G., Deufemia, V., Polese, G.: Visual language implementation through standard compiler-compiler techniques. Journal of Visual Languages & Computing **18** (2007)
9. Kastens, U.: Devil tool. http://ag-kastens.uni-paderborn.de/forschung/devil/ (2008)