# Supporting Visual Editors using Reference Attributed Grammars

Niklas Fors

Department of Computer Science, Lund University, Lund, Sweden
niklas.fors@cs.lth.se

**Abstract.** Reference attributed grammars (RAGs) extend Knuth's attribute grammars with references. These references can be used to extend the abstract syntax tree to a graph. We investigate how RAGs can be used for implementing tools for visual languages. Programs in those languages can often be expressed as graphs.

## 1 Introduction

For certain applications domain-specific languages (DSLs) play an important role, since they can make it easier to solve problems in the domain in a more concise way than a general purpose language (GPL) [1]. DSLs are especially advantageous when the language user is a domain expert and not a programmer. Other advantages of DSLs over GPLs are easier domain analysis and optimizations [2]. However, there are several difficulties with DSLs as well. Developing DSLs often require compiler implementation knowledge, which is not necessary when creating a library. It is also desirable that the domain is well-understood, where abstractions are more likely to remain stable. Limited resources for creating and maintaining a DSL can be a problem, especially for small communities.

A DSL user may want different tools, such as a batch compiler and an integrated development environment (IDE). Such tools often share analysis, and reusing specification between them can reduce the implementation effort and help to keep the tools consistent with each other. One tool aiming for reuse of language and compiler specification is JastAdd [3], which we use in our research.

Often, new insights and knowledge about a particular domain arise over time, leading to new abstractions in the DSL. In such cases, the language developer typically wishes to extend the existing DSL in a backward compatible way. JastAdd has proven to be useful for extending languages; compilers for complex languages like Java and Modelica have successfully been implemented and extended with new language constructs [4,5].

In several domains, a visual notation is preferable to a textual notation, which can increase the accessibility for non-programmers. Such languages are called domain-specific visual languages (DSVLs). Some languages, such as Modelica [6], support both a textual and a visual syntax. This allows the software engineer and the domain expert to use the most convenient notation.

We want to investigate how the metacompilation system JastAdd, and its formalism reference attributed grammars [7], can be used for implementing tools for visual languages in general, and DSVLs in particular.

## 2 Background

There are several ways for defining the semantics, that is the meaning, of a programming language. Examples include denotational semantics [8,9], operational semantics [10,11] and attribute grammars [12].

In this work, we will use reference attributed grammars (RAGs), which is an extension to Knuth's attribute grammars [12]. RAGs allow the value of an attribute to be a reference to a node in the abstract syntax tree. For example, an identifier usage may have a reference attribute that refers to its declaration. The usage node can use this reference attribute to access attributes from the declaration node, for example, to obtain the type of the declaration. Using reference attributes, a graph can be superimposed on the AST, which can be used to model graph-based languages. Since graphs can be cyclic, the support of circular reference attributes [13] in JastAdd can be helpful, which solve circular attribute definitions using fixed-point iteration. Earlier work has shown that circular reference attributes are practical and provide a concise definition for the implementation of control-flow and data-flow for Java programs [14]. Other attribute grammar system supporting reference attributes include Silver [15] and Kiama [16].

For visual languages, there are several ways to define the syntax of a language. There is, however, no consensus on what formalism to use, in contrast to textual languages, where Chomsky's context-free grammars are widely used. Examples of formalisms are graph grammars [17,18,19], metamodeling [20,21] and constraint multiset grammars [22].

Erwig [23] proposed a separation of the concrete and abstract syntax for a visual language, and defined the semantics on the latter. The abstract syntax was modeled as a directed labeled multi-graph (there can be several edges between two nodes). Our approach differs from Erwig in that the base structure is the AST compared to a graph, and instead we use reference attributes to extend the AST to a graph. We also use RAGs to define the semantics. We think that a tree is practical when a textual syntax is desired, since it is straightforward to serialize and easy to create new concrete syntaxes. Rekers et al. [19] also suggest a separation between the concrete syntax and abstract syntax for visual languages.

Another related work is reusable visual patterns by Schmidt et al. [24], implemented with attribute grammars, but without use of RAGs. These patterns are predefined, and examples are lists, tables, sets, lines etc. The language developer can choose among the patterns and use the ones that matches the visual language, and can customize visual properties for the patterns. If a pattern is missing, a new can be added by defining attribute equations.

## 3 Research Proposal

We want to investigate how well suited RAGs are for defining the semantics for a visual language, and how to use RAGs for semantics-dependent editor interaction. In particular, we have the following research goals:

- Declarative semantic specification.
- Reuse of semantic specifiation.
- Semantic support in visual editor, for example, showing computed properties.
- Support languages with both a textual and visual syntax.
- Automatic incremental evaluation, using work by Söderberg et al. [25].
- Integration with other editor frameworks, such as JastGen [26].

### 3.1 Challenges

We have identified the following research challenges, based on experience from the preliminary work described below.

**Defining visual syntax.** When defining a visual language, there should be a way to define its syntax and the visual representation, that is, properties such as shape and color. One way is to have a DSL supporting that and which includes concepts such as nodes, connections and containment. To specify how nodes can connect, constraints can be used. Attributes are one way to specify such constraints. Another way is to use OCL as Bottoni et al. [20] have done or create a predicate language as Esser et al. [27]. It should also be possible to show attribute values. This can be used to show computed properties such as cycles in graphs, inferred types, etc.

Another approach for defining the syntax is by graph grammars [19], such as hypergraphs [17], where the visual representation is transformed to a graph, which is then parsed. One benefit of graph grammars is that they allow both free-hand and syntax-directed editing. However, earlier work [28] indicates that metamodeling using GMF [21] is easier to use but less expressive than hypergraphs, which is something we have experienced as well. We think it is desirable that the specification format is easy to use, so it is appealing for a software engineer.

We want to find and choose one approach and connect it to RAGs.

**Model consistency.** In the visual editor, we have both the view and the AST model, and a challenge is to keep these models consistent with each other. One way is to let the visual editor update the AST, which then informs the view about the change and is updated accordingly. If the editor supports both textual and visual editing, is this technique feasible? Another way is to support bidirectional transformations [29].

**Automatic layout.** We want to find out how to handle layout that can be sensitive to semantics. A language from ABB has this property, as described below. For these languages, automatic layout must take this semantics into consideration, in order to not inadvertently change the meaning of a diagram.

**Erroneous diagrams.** It can be helpful for the language user if it is possible to temporarily have diagrams that contain errors. For example, when two edges are being switched and it is forbidden to have more than one edge to a node, then it may be useful to temporarily have one edge pointing to nothing.

**Undo / redo functionality.** To increase the usability of editors for visual languages, functionality like undo and redo is often desirable. We would like to generate edit operations for the AST that automatically provide support for undo and redo functionality.

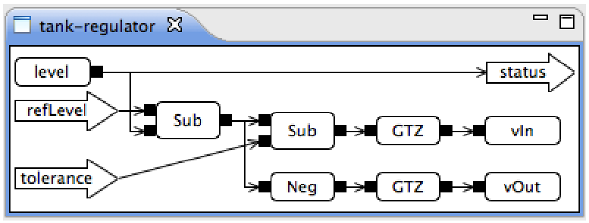
**Refactoring.** Another challenge is to support refactoring in visual languages with RAGs. Is it possible to extend the work done by Schäfer et al. [30]?

## 3.2 Methodology

The research methodology we use is similar to design science research in information systems. Instead of trying to understand the reality, as natural science does, design science tries to create new innovative artifacts that have utility [31]. We develop prototypes that are based on practical applications, as the preliminary work with ABB demonstrates. The research contribution is the technical artifact itself and the development of the foundations of RAGs. Design science consists of two activities, *build* and *evaluate*, and is an iterative process; the artifact is continuously evaluated to provide feedback to the building activity [32]. The evaluation will include performance measurements, for example, the response time for standard editing operations, as well as evaluation of specification size and modularity. We develop a prototype visual language together with ABB using RAGs, and it would be interesting to see if the engineers at ABB successfully can experiment with new language constructs for this language in a modular way.

## 3.3 Preliminary Work

We have started a collaboration with ABB, and designed a control language based on a industrial language from ABB [33]. This language has both a textual and a visual syntax. A diagram in the language contains blocks that are executed periodically and connections between them that specify the data flow. One interesting aspect of the language is that the semantics are influenced by the visual placements of the blocks. The execution order of the blocks is primarily defined by the data flow, and secondary by visual placements. To define the semantics without visual coordinates, the visual placements are reflected in the declaration order of the blocks in the textual syntax. We have implemented a visual editor for this language using RAGs and the Graphical Editing Framework (GEF), an Eclipse-based framework. A screenshot of the editor can be seen in Fig. 1, with the corresponding textual syntax.



```
diagramtype tank-regulator(
    Int refLevel, Int tolerance
    => StatusData status) {
  StatusMonitor level;
  Sub Sub_1;
  ...
  connect(refLevel, Sub_1.in1);
  ...
}
```

(a) Visual syntax (b) Textual syntax

Fig. 1: A model for regulating the liquid volume in a tank, using two valves, `vIn` and `vOut`.

## 4 Conclusions

We have in this paper described the opportunity to use RAGs for implementing tools for visual languages. Attribute gammars is a proven and convenient formalism to specify the semantics of a language. RAGs extends AGs with references, and makes it possibly to superimpose graphs on an AST, which can be used to represent programs of visual languages. Expected benefits of using RAGs include reuse of compiler specification between different tools and the possibility to experiment with new language constructs in a modular way. The expected contributions are the prototype system and development of the foundations of RAGs to support visual languages. Research goals have been described and research challenges identified. Finally, we mention design science research as our research methodology and describe the preliminary work with ABB that serves as a case study.

## Acknowledgements

We thank the anonymous reviewers for interesting and useful comments on a earlier draft of this paper.

## References

1. Deursen, A., Klint, P.: Little languages: Little maintenance? Journal of Software Maintenance: Research and Practice **10**(2) (1998) 75–92
2. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**(4) (2005) 316–344
3. Ekman, T., Hedin, G.: The JastAdd system - modular extensible compiler construction. Science of Computer Programming **69**(1-3) (2007) 14–26
4. Ekman, T., Hedin, G.: The Jastadd Extensible Java Compiler. In: OOPSLA 2007, ACM (2007) 1–18
5. Hedin, G., Åkesson, J., Ekman, T.: Extending languages by leveraging compilers: from Modelica to Optimica. IEEE Software **28**(3) (March 2010) 68–74

6. Modelica Association: Modelica - A Unified Object-Oriented Language for Systems Modeling - Language Specification Version 3.3. (2012) Available from: `http://www.modelica.org`.
7. Hedin, G.: Reference Attributed Grammars. In: Informatica (Slovenia). 24(3) (2000) 301–317
8. Scott, D.: Outline of a mathematical theory of computation. Oxford University Computing Laboratory, Programming Research Group (1970)
9. Scott, D., Strachey, C.: Toward a mathematical semantics for computer languages. Oxford University Computing Laboratory, Programming Research Group (1971)
10. Kahn, G.: Natural semantics. In: 4th Annual Symposium on Theoretical Aspects of Computer Sciences on STACS 87. Volume 247 of LNCS., Passau, Germany, Springer (1987) 22–39
11. Plotkin, G.D.: A Structural Approach to Operational Semantics. Technical report, Department of Computer Science, University of Aarhus (1981)
12. Knuth, D.E.: Semantics of Context-free Languages. Math. Sys. Theory **2**(2) (1968) 127–145 Correction: *Math. Sys. Theory* 5(1):95–96, 1971.
13. Magnusson, E., Hedin, G.: Circular Reference Attributed Grammars - Their Evaluation and Applications. Science of Computer Programming **68**(1) (2007) 21–37
14. Söderberg, E., Ekman, T., Hedin, G., Magnusson, E.: Extensible intraprocedural flow analysis at the abstract syntax tree level. Science of Computer Programming (2012)
15. Wyk, E.V., Bodin, D., Gao, J., Krishnan, L.: Silver: An extensible attribute grammar system. Science of Computer Programming **75**(1-2) (2010) 39–54
16. Sloane, A.M., Kats, L.C.L., Visser, E.: A pure object-oriented embedding of attribute grammars. Electr. Notes Theor. Comput. Sci. **253**(7) (2010) 205–219
17. Minas, M., Viehstaedt, G.: DiaGen: A generator for diagram editors providing direct manipulation and execution of diagrams. In: Proceedings of the IEEE International Symposium on Visual Languages, IEEE (1995) 203–210
18. Ehrig, K., Ermel, C., Hänsgen, S., Taentzer, G.: Generation of visual editors as Eclipse plug-ins. In: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. ASE '05, ACM (2005) 134–143
19. Rekers, J., Schürr, A.: Defining and parsing visual languages with layered graph grammars. Journal of Visual Languages and Computing **8**(1) (1997) 27–55
20. Bottoni, P., Costagliola, G.: On the definition of visual languages and their editors. Diagrammatic Representation and Inference (2002) 337–396
21. GMF: The Eclipse Graphical Modeling Framework http://www.eclipse.org/modeling/gmp/
22. Marriott, K.: Constraint multiset grammars. In: Visual Languages, 1994. Proceedings., IEEE Symposium on, IEEE (1994) 118–125
23. Erwig, M.: Abstract syntax and semantics of visual languages. J. Vis. Lang. Comput. **9**(5) (1998) 461–483
24. Schmidt, C., Kastens, U.: Implementation of visual languages using pattern-based specifications. Software: Practice and Experience **33**(15) (2003) 1471–1505
25. Söderberg, E., Hedin, G.: Incremental evaluation of reference attribute grammars using dynamic dependency tracking. Technical Report 98, Lund University (April 2012) LU-CS-TR:2012-249, ISSN 1404-1200.
26. Söderberg, E., Hedin, G.: Building semantic editors using jastadd. In: Proceedings of the of the 11th Workshop on Language Descriptions, Tools and Applications, LDTA 2011, ACM (2011)

27. Esser, R., Janneck, J.: A framework for defining domain-specific visual languages. Workshop on Domain Specific Visual Languages, ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA-2001) (2001)
28. Tveit, M.: Specification of graphical representations-using hypergraphs or meta-models? Norsk informatikkonferanse NIK (2008) 39–50
29. Antkiewicz, M., Czarnecki, K.: Design space of heterogeneous synchronization. In: Generative and Transformational Techniques in Software Engineering II. LNCS, Springer (2008) 3–46
30. Schäfer, M., Ekman, T., de Moor, O.: Sound and Extensible Renaming for Java. In Kiczales, G., ed.: 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2008), ACM Press (2008)
31. March, S.T., Smith, G.F.: Design and natural science research on information technology. Decis. Support Syst. **15**(4) (December 1995) 251–266
32. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**(1) (March 2004) 75–105
33. Fors, N., Hedin, G.: Handling of layout-sensitive semantics in a visual control language. In: Proceedings of the 2012 IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE (2012) poster paper to appear.