Generating Semantic Editors using Reference Attribute Grammars

Emma Söderberg

Department of Computer Science, Lund University, Lund, Sweden
emma.soderberg@cs.lth.se

Abstract. In this short research plan we address the problem of generating modern semantic editors from formal specifications. We aim to tackle this problem using reference attribute grammars.

1 Problem Description and Motivation

There are a lot of programming languages around and the number keeps increasing. Most of these languages have small communities with limited resources. As a consequence, a lot of development in these languages is performed in simple text editors, in lack of better semantic tool support. At the same time, users of languages like Java with larger communities can choose from a set of high-quality semantic editors, like the Eclipse JDT, IntelliJ IDEA or NetBeans, with modern semantic services like context-sensitive name completion and refactorings [18].

Preferably, it should be simple and fast to develop semantic tools with modern services like these for languages with smaller communities or less resources. However, these tools are hand-crafted and developed over several years. One appealing approach to reduce the development time of semantic tools is to generate them from a formal specification [17]. This approach has several benefits in that it lets developers describe behavior on a higher conceptual level. Also, the specification is typically smaller than its manually implemented counterpart, which makes it easier to overview and easier to change. Another benefit is the possibility to check the specification for semantic errors, an activity which would require more cumbersome testing in a hand-coded implementation.

We can summarize the above need for semantic editors and the benefits of generation into a problem of *generating modern semantic editors from a formal semantic description*. This problem includes technical difficulties such as coping with growing languages [41] and extensibility, responsiveness and performance of a generated editor, and flexible descriptions of the views and services of an editor. The rest of this document aims to give a coarse overview of a plan for research addressing this problem.

2 Brief Overview of Related Work

There exists several formal means for specifying semantics. For example, attribute grammars (AGs) by Knuth [26], denotational semantics by Scott and Strachey [39,

42], natural semantics by Kahn [21], and algebraic semantics by Bergstra et al. [4]. We will focus on reference attribute grammars (RAGs) by Hedin [16], an extended form of AGs. A benefit of RAGs is their ability to explicitly express super-imposed graphs on top of an abstract syntax tree (AST). Super-imposed graphs like these can be used to describe for example inheritance and cross-references. RAGs have been shown useful for describing the semantics of complex languages like Java [13] and Modelica [2]. Some examples of systems supporting RAGs are JastAdd by Ekman et al. [19, 14], Silver by van Wyk et al. [44], Kiama by Sloane et al. [40], and Aster by Kats et al.[24].

Examples of earlier systems generating semantic editors from formal specifications include the PSG system by Bahlke et al. [3] using denotational semantics, the CENTAUR system by Borras et al. [6] using natural semantics, the ASF+SDF meta-environment [25] using algebraic semantics, the Synthesizer Generator by Reps et al. [35] using ordered attribute grammars (OAGs) by Kastens [22], and the Lrc system by Kuipers et al [28] using higher-order attribute grammars by Vogt et al. [46]. OAGs is a powerful subset of AGs enabling a static evaluation order.

One important property of a semantic editor is incremental updating of the semantic model. Both the Synthesizer Generator and the Lrc system support incremental updating. The statically known evaluation order of OAGs, supported by these systems, provide sufficient information for incremental updating of attribute values. RAGs have also been used in the context of editors, in the APPLAB system by Bjarnason et al. [5], but not in conjunction with incremental updating. In general, RAGs require dynamic evaluation and incremental updating of RAGs is an open problem.

In recent years, a number of tool generating systems have emerged which extend the Eclipse Platform. One example is the Eclipse Modeling Framework (EMF) by Budinsky et al. [8] which provides means for expressing structured data models (graphs). EMF can generate a basic graphical editor for these models and supports updating of a model via manual registration of model observers.

Another example is the IDE Meta-tooling Platform (IMP) by Charles et al. [10] which has a semi-automatic approach to the development of textual semantic editors. IMP semi-generates text editors using wizards and generation of code skeletons. Developers manually fill in language-specific behavior in these code skeletons. Parsing is supported by the LPG parser generator, but this is optional as shown in, for example, the Spoofax/IMP system by Kats et al. [23] which extends IMP using a different parsing technology. Spoofax provides a language workbench which uses strategic term rewriting [45] to express language semantics.

The EMF project also supports generation of textual editors via the xText project by Efftinger et al. [12]. In contrast to IMP, xText generates a more complete text editor based on a custom grammar format, using an EMF-based model and an ANTLR parser. xText uses a combination of the Object Constraint Language (OCL) [43] and dependency injection to implement semantics.

These frameworks could possibly be used as the target platform for a generated editor based on RAGs. A combination of EMF and JastAdd models have been explored by Bürger et al. in JastEMF [9]. Another example of a system supporting generation of textual editors is the MontiCore system by Krahn et al. [27]. MontiCore uses a com-

bined grammar format for concrete and abstract syntax supporting modular language extensions. This grammar format uses UML-like associations to describe semantics.

3 Proposed Solution

In order to address the problem posed in Section 1 we need a formal yet flexible way to describe the semantics of a language, including the abstract syntax. The semantic descriptions need to be modular in order to accommodate the need for extensibility. The semantic formalism also needs to be expressive to such a degree that the semantic information needed by the editing services can be computed. Beyond the need for pure semantic descriptions, we need a framework surrounding the underlying semantic model of a program and a means to describe services and views. These descriptions of services and views should seamlessly connect to the semantic descriptions.

We have chosen to use RAGs for semantic descriptions and we aim to construct a tool JedGen – *JastAdd-based semantic editor generator* supporting the remaining parts. These remaining parts include the framework surrounding a generated editor, means for describing services and views, and the actual generation of editors. We aim to support all languages that would benefit from static semantic analysis during development. To meet the demands on semantic development tools of today we have devised a list of three areas which a generated editor should support to be on par with hand-crafted modern semantic editors:

- *Incremental update* This is a highly desirable part of an interactive tool which affects performance and hence responsiveness. Incremental updating of RAG-based models is an open problem which we plan to address. A solution can possibly be based on work by Reps [34], by Jones [20], by Hedin [15], by Boyland [7], and by Acar et al. [1]. A solution to incremental updating of RAGs would be a contribution of this thesis.

- *Multiple views* Different development tasks benefit from different views of source code artifacts. Here, we include all views, editable or non-editable. This includes textual editors. Multiple views require a general architecture with support for synchronization and updating in a multi-threaded environment. Also, a generator needs to support a general way to specify the content and visualization of these views. Some work has been done on visualizing programs using RAGs [29] which we plan to extend along with a surrounding framework.

- *Modern semantic services* Inspection and modification of source code artifacts, requiring *context-sensitive static semantic information*. Some examples of services are code smell detection, context-sensitive metrics [11], cross-referencing, renaming and name completion. Promising work by Schäfer et al. [37, 38, 36] show that RAGs can be used to support sophisticated services like refactorings, and work by Nilsson-Nyman et al. [33] show how RAGs can be used to find dead code. The potential contributions of this work are further explorations of descriptions of semantic service information and service descriptions seamlessly connecting to RAG-based semantic descriptions.

4 Research Method

Our research method is constructive and experimental. We base our research on the hypothesis that "*RAGs can be used to generate modern semantic editors*", which we aim to demonstrate using a prototype. The development of a RAG-based generator prototype makes our research constructive.

The plan for the development of JedGen includes two phases – a *prototype framework* and a *prototype generator*. The purpose of the first phase is to build the general framework needed around a generated editor. This work can be separated into three subparts – the development of incremental updating of RAG-based ASTs, the development of mechanisms for access and updating of the ASTs in a general way and specification of views and services.

During the first phase the goal is to stepwise develop non-generated editor extensions to existing RAG-based compilers as a means for evaluation of the framework. Currently, we are working with editor extensions for Java and Modelica. We aim to evaluate these prototypes experimentally with regard to *behavior* (e.g., with regard to correctness), *coverage* (e.g., the range of errors that a generated editor can locate), *efficiency* (e.g., the performance of semantic analysis), and *effort* (e.g., line of code of an editor specification) The purpose of the second phase is to develop a prototype generator based on experiences gained in the previous phase. This includes a general description format for definition of editors based on an abstract syntax.

The JedGen tool is still in its first phase, but an alpha version of the tool supporting a semantic editing model has been used by Schäfer et al. in their exploration of refactorings [37, 38]. JedGen has also been used by several undergraduate students, as a part of their thesis work [31, 30, 32], and in a graduate course on RAGs.

5 Acknowledgements

A big thanks to all anonymous reviewers for valuable comments on an early version of this abstract.

References

1. Umut A. Acar, Guy E. Blelloch, and Robert Harper. Adaptive functional programming. *ACM Trans. Program. Lang. Syst.*, 28(6):990–1034, 2006.
2. Johan Åkesson, Torbjörn Ekman, and Görel Hedin. Development of a Modelica compiler using JastAdd. *Science of Computer Programming*, 75:21–38, January 2010.
3. Rolf Bahlke and Gregor Snelting. The PSG system: from formal language definitions to interactive programming environments. *ACM Transactions on Programming Langanguages and Systems (TOPLAS)*, 8(4):547–576, 1986.
4. Jan A. Bergstra. *Algebraic specification*. ACM, New York, NY, USA, 1989.
5. Elizabeth Bjarnason, Görel Hedin, and Klas Nilsson. Interactive language development for embedded systems. *Nordic Journal of Computing*, 6(1):36–54, 1999.
6. Patrick Borras, Dominique Clément, Th. Despeyroux, Janet Incerpi, Gilles Kahn, Bernard Lang, and V. Pascual. CENTAUR: The system. In *Software Development Environments (SDE)*, pages 14–24, 1988.

7. John Tang Boyland. Incremental evaluators for remote attribute grammars. In *Proceedings of the Second Workshop on Language Descriptions, Tools and Applications (LDTA 2002)*, volume 65 of *Electronic Notes in Theoretical Computer Science*, pages 9–29. Elsevier B.V., July 2002.
8. Frank Budinsky, Stephen A. Brodsky, and Ed Merks. *Eclipse Modeling Framework*. Pearson Education, 2003.
9. Christoff Bürger and Sven Karol. JastEMF, 2010. http://code.google.com/p/jastemf [Access September 2010].
10. Philippe Charles, Robert M. Fuhrer, Stanley M. Sutton, Jr., Evelyn Duesterwald, and Jurgen Vinju. Accelerating the creation of customized, language-specific ides in eclipse. *SIGPLAN Notices*, 44(10):191–206, 2009.
11. Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.
12. Sven Efftinge and Markus Völter. oAW xText: a framework for textual DSLs. In *Eclipse Summit Europe, Eclipse Modeling Symposium*, Esslingen, Germany, October 2006.
13. Torbjörn Ekman and Görel Hedin. The JastAdd extensible Java compiler. In *OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, pages 1–18, New York, NY, USA, 2007. ACM.
14. Torbjörn Ekman and Görel Hedin. The JastAdd system – modular extensible compiler construction. *Science of Computer Programming*, 69(1–3):14–26, December 2007.
15. Görel Hedin. *Incremental semantic analysis*. PhD thesis, 1992.
16. Görel Hedin. An overview of door attribute grammars. In Peter Fritzson, editor, *CC*, volume 786 of *Lecture Notes in Computer Science*, pages 31–51. Springer, 1994.
17. Jan Heering and Paul Klint. Semantics of programming languages: a tool-oriented approach. *SIGPLAN Notices*, 35(3):39–48, 2000.
18. Daqing Hou and Yuejiao Wang. An empirical analysis of the evolution of user-visible features in an integrated development environment. In *CASCON '09: Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, pages 122–135, New York, NY, USA, 2009. ACM.
19. The JastAdd Team. jastadd.org. http://jastadd.org/ [Access May 2010.
20. Larry G. Jones. Efficient evaluation of circular attribute grammars. *ACM Trans. Program. Lang. Syst.*, 12(3):429–462, 1990.
21. Gilles Kahn. Natural semantics. In Franz-Josef Brandenburg, Guy Vidal-Naquet, and Martin Wirsing, editors, *STACS*, volume 247 of *Lecture Notes in Computer Science*, pages 22–39. Springer, 1987.
22. Uwe Kastens. Ordered attributed grammars. *Acta Informatica*, 13(3):229–256, March 1980.
23. Lennart C. L. Kats, Karl T. Kalleberg, and Eelco Visser. Domain specific languages for composable editor plugins. In Torbjörn Ekman and Jurgen Vinju, editors, *Proceedings of the Ninth Workshop on Language Descriptions, Tools, and Applications (LDTA 2009)*, Electronic Notes in Theoretical Computer Science. Elsevier B. V., 2009.
24. Lennart C. L. Kats, Anthony M. Sloane, and Eelco Visser. Decorated attribute grammars: Attribute evaluation meets strategic programming. In Oege de Moor and Michael I. Schwartzbach, editors, *CC*, volume 5501 of *Lecture Notes in Computer Science*, pages 142–157. Springer, 2009.
25. Paul Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2(2):176–201, April 1993.
26. Donald E. Knuth. Semantics of context-free languages. *Journal Theory of Computing Systems*, 2(2):127–145, June 1968.
27. Holger Krahn, Bernhard Rumpe, and Steven Völkel. MontiCore: Modular development of textual domain specific languages. In Richard F. Paige and Bertrand Meyer, editors,

TOOLS (46), volume 11 of *Lecture Notes in Business Information Processing*, pages 297–315. Springer, 2008.
28. Matthijs F. Kuiper and João Saraiva. Lrc - a generator for incremental language-oriented tools. In Kai Koskimies, editor, *CC*, volume 1383 of *Lecture Notes in Computer Science*, pages 298–301. Springer, 1998.
29. Eva Magnusson and Görel Hedin. Program Visualization using Reference Attributed Grammars. volume 7, pages 67–86. Publishing Association Nordic Journal of Computing, 2000.
30. Jesper Mattsson. The JModelica IDE: Developing an IDE reusing a JastAdd compiler. Master's thesis, Lund University, Lund, Sweden, August 2009.
31. Erik Mossberg. Inspector – tool for interactive language development. Master's thesis, Lund University, Lund, Sweden, October 2009.
32. Philip Nilsson. Semantic editing compiler extensions using JastAdd. Master's thesis, Lund University, Lund, Sweden, June 2010. To be presented.
33. Emma Nilsson-Nyman, Torbjörn Ekman, Görel Hedin, and Eva Magnusson. Declarative intraprocedural flow analysis of Java source code. In *Proceedings of the Eight Workshop on Language Description, Tools and Applications (LDTA 2008)*, Electronic Notes in Theoretical Computer Science. Elsevier B.V., 2008.
34. Thomas Reps. *Generating Language-Based Environments*. PhD thesis, 1984.
35. Thomas Reps and Tim Teitelbaum. The Synthesizer Generator. In Peter B. Henderson, editor, *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, volume 19(5) of *SIGSOFT Software Engineering Notes*, pages 42–48, Pittsburgh, Pennsylvania, USA, May 1984. ACM.
36. Max Schäfer, Julian Dolby, Manu Sridharan, Emina Torlak, and Frank Tip. Correct Refactoring of Concurrent Java Code. In Theo D'Hondt, editor, *24th European Conference on Object-Oriented Programming (ECOOP '10)*, 2010.
37. Max Schäfer, Torbjörn Ekman, and Oege de Moor. Sound and extensible renaming for Java. In Gail E. Harris, editor, *OOPSLA*, pages 277–294. ACM, 2008.
38. Max Schäfer, Mathieu Verbaere, Torbjörn Ekman, and Oege de Moor. Stepping stones over the refactoring rubicon. In Sophia Drossopoulou, editor, *ECOOP*, volume 5653 of *Lecture Notes in Computer Science*, pages 369–393. Springer, 2009.
39. Dana Scott. Mathematical concepts in programming language semantics. In *AFIPS '72 (Spring): Proceedings of the May 16-18, 1972, spring joint computer conference*, pages 225–234, New York, NY, USA, 1972. ACM.
40. Anthony M. Sloane, Lennart C. L. Kats, and Eelco Visser. A pure object-oriented embedding of attribute grammars. In T. Ekman and J. Vinju, editors, *Proceedings of the Ninth Workshop on Language Descriptions, Tools, and Applications (LDTA 2009)*, Electronic Notes in Theoretical Computer Science. Elsevier B. V., 2009.
41. Guy L. Steele Jr. Growing a language. *Higher-Order and Symbolic Computation*, 12(3):221–236, October 1999.
42. Christopher Strachey. Towards a formal semantics. pages 198–216, 1966.
43. The Object Management Group (OMG). The Object Constraints Language (OCL), 2010. http://www.omg.org/technology/documents/formal/ocl.htm [Accessed May 2010].
44. Eric van Wyk, Derek Bodin, Jimin Gao, and Lijesh Krishnan. Silver: an extensible attribute grammar system. In *Proceedings of the Seventh Workshop on Language Descriptions, Tools, and Applications (LDTA 2007)*, Electronic Notes in Theoretical Computer Science. Elsevier B. V., 2007.
45. Eelco Visser. Stratego: A language for program transformation based on rewriting strategies. In *RTA '01: Proceedings of the 12th International Conference on Rewriting Techniques and Applications*, pages 357–362, London, UK, 2001. Springer-Verlag.
46. Harald Vogt, S. Doaitse Swierstra, and Matthijs F. Kuiper. Higher-order attribute grammars. In *PLDI*, pages 131–145, 1989.