SmartTools: a Development Environment Generator based on XML Technologies*

Isabelle Attali[1], Carine Courbis[1], Pascal Degenne[1], Alexandre Fau[1], Joël Fillon[1], Didier Parigot[1], Claude Pasquier[1], and Claudio Sacerdoti Coen[2]

[1] INRIA-Sophia - 2004 Route des Lucioles BP 93 06902 Sophia Antipolis Cedex France
{First.Last}@sophia.inria.fr
[2] Department of Computer Science, University of Bologna, Mura Anteo Zamboni, 7, 40127 Bologna Italy
sacerdot@cs.unibo.it

Abstract. SmartTools is a development environment generator that provides a structure editor and semantic tools as main features. SmartTools is easy to use thanks to its graphical user interface. Being based on Java and XML technologies offers all the features of SmartTools to any defined language. The main goal of this tool is to provide help and support for designing software development environments for programming languages as well as domain-specific languages defined with XML technologies.
Keywords: Java, DOM, XML, BML, XSLT, Program transformation, Software engineering, Interactive environment.

1 Introduction

Producing high-quality software has become a major concern in industry. There is a long history of research about providing help and support during the development process [4, 5, 7, 13–15, 22]. It is imperative that the research community creates technologies to enhance the quality of software development and increase the developers productivity. These goals are addressed by the SmartTools framework and research [3]. It is composed of a set of generic and interactive software components organized into a modular architecture. With this basic environment, designers of languages are able to easily define and implement a set of specific applications for their languages using generic tools. The SmartTools system is completely written with the Java programming language and strongly uses XML technologies [23].

2 Abstract Syntax and Semantic Manipulation

SmartTools internally uses the Abstract Syntax Tree (AST) definitions of the manipulated languages. It can accept Document Type Definitions (DTDs) as AST definitions to define languages. When a DTD is imported, it automatically produces a set of Java

* This project is supported in part by Microsoft Research and Dyade, the Bull-Inria Research Joint Venture.

classes that extend a DOM implementation for each operator. It proposes a structure editing environment for the language defined by this DTD for free. During editing, the system constantly maintains the consistency of XML documents and guarantees that they remain valid and well-formed. When extra data are provided in DTDs, it can also automatically generate a set of pretty-printers and parsers to offer a user-friendly syntax for languages. In the future, we also plan to accept XML Schemas, Relax and TREK.

When an XML file is parsed, SmartTools constructs a strongly typed structure by mapping XML elements to instances of classes generated at DTD importation ([24, 11] use the same technique). Such a typed structure is necessary to apply the visitor design pattern technique [19, 20] (a well-known oriented-object programming technique) for tree manipulations. To ease the development of new visitors for any specific purpose, SmartTools generates a default visitor class. One can extend it by inheritance and override some of the visit methods to perform a new analysis on DOM trees like typechecking, evaluation or compilation. Additionally, it is possible to dynamically customize the behavior of visitors by using visitor-specific aspects [1] e.g., we use an aspect to implement a generic graphical debug mode. We have also introduced a generic visitor concept to be able to factorize identical behaviors applicable to all the nodes of a tree.

A challenging test-bench for this technique has been the application of DTDs developed by the HELM [2] project to describe a high-level logic used by the Coq proof assistant [12] to encode mathematical theories. A whole system of ten mutually recursive visitors has been easily implemented and applied to 27Mb of mutually linked XML files to reconstruct a valid input for Coq. The proposed technique for semantic manipulations proved to be easy to manage, efficient and scalable.

Finally, typed DOM structures and visitor patterns let designers easily create languages and their associated semantic tools. Thus, we have bootstrapped SmartTools by using it to design all its internal languages (our abstract syntax tree definition, pretty-printer definition, etc.) and their tools (compiler, typechecker, etc.). Approximately 40% of SmartTools source code is generated by SmartTools itself.

3 Interactive Environment

SmartTools includes a graphical interface which makes strongly use of XML technologies. It has multi-viewing capabilities (see Figure 1) and guarantees the consistency between the tree structure and its different views during the structure editing. Each view is built by the assembly of JavaBeans components. With this concept, a user can quickly and easily design new views by choosing different JavaBeans or by changing the way components are gathered together. In our system, representation of JavaBeans is expressed in an XML document using the Bean Markup Language (BML) [10] syntax. The XSL Transformation language (XSLT) [21] is used to express the transformation between the tree structure and its graphical views. Only a subset of XSLT instructions is used in order to make possible incremental updates of views.

A higher-level transformation language called Xpp has been defined on top of XSLT. Its features are very similar to those of XSLT but it is much more concise, more readable and can perform transformations only on subtrees for incremental purposes. An

Xpp specification is currently converted in XSLT. We plan to propose other implementations based on rewriting systems like TOM [17].

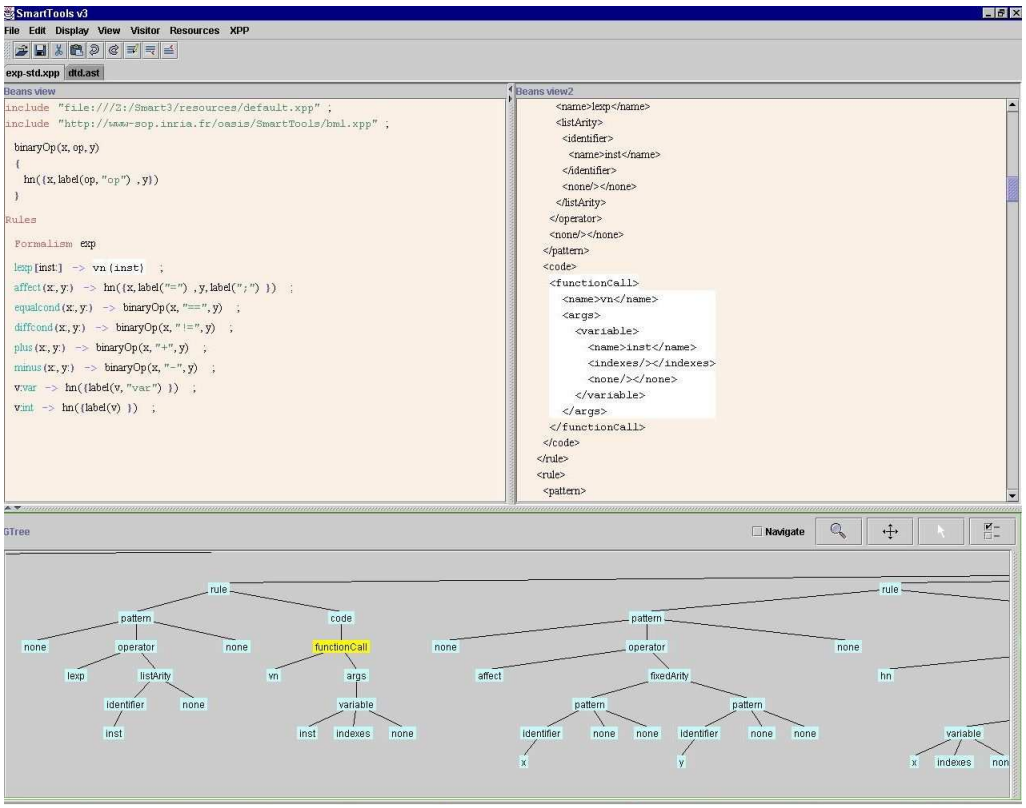

Fig. 1. SmartTools environment showing different views of the same tree structure

Xpp consists of a set of rule definitions which match patterns with explicit variables for subtrees. These variables are used in the code part for recursive calls. This constraint offers the capability of preserving incremental changes of the transformation. With Xpp, it is possible to define functions that avoid writing redundant code. It also provides an importation mechanism of others Xpp documents to reuse previously defined transformations.

4 The Architecture of the SmartTools Framework

SmartTools is made of several independent software components that communicate with each other through an asynchronous messaging system. We designed a message controller in charge of managing the flow of messages and delivering them to their

destinations. So far, information carried in messages is serialized in XML format. We plan to move a step forward and adapt our system to respect the W3C Simple Object Access Protocol (SOAP) [18] specifications in a close future. The design of this messaging system has proven to be simple, efficient, and easy to maintain. Another benefit is that SmartTools components can be used and integrated by other applications without the need of the whole system. Moreover with this architecture, it is very easy to obtain standalone version of tools for their external uses.

5 Conclusion

With SmartTools, it is possible to easily generate an interactive environment for programming or domain-specific languages. SmartTools automatically offers a well-known visitor pattern technique to specify semantic analysis on DOM tree structures. Its graphical part is mainly based on free existing implementations of standards (XSLT, BML). We have chosen to use non-proprietary APIs in the concern to be open and take advantage of future or external developments.

We already have some examples of easy and successful integration of research tools [16, 6, 9], and technology transfer in industrial environment [8]. Additionally, we hope to benefit from the large fields of applications that appear through XML technologies.

Acknowledgements

We have much benefited from discussions with Colas Nahaboo, Thierry Kormann and Stéphane Hillion about XML technologies.

References

1. Aspect-Oriented Programming. http://www.parc.xerox.com/csl/projects/aop/.
2. The Hypertextual Electronic Library of Mathematics. http://www.cs.unibo.it/~asperti/HELM/.
3. Isabelle Attali, Carine Courbis, Pascal Degenne, Alexandre Fau, and Didier Parigot. SmartTools: a Generator of Interactive Environments Tools. In *Compiler Construction CC'2001*, Lecture Notes in Computer Science, Genova, Italy, April 2001. Springer-Verlag. Tool demonstration, to appear.
4. Lex Augusteijn. The Elegant Compiler Generation System. In Pierre Deransart and Martin Jourdan, editors, *Attribute Grammars and their Applications (WAGA)*, volume 461 of *Lecture Notes in Computer Science*, pages 238–254. Springer-Verlag, New York–Heidelberg–Berlin, September 1990. Paris.
5. Don Batory, Bernie Lofaso, and Yannis Smaragdakis. JTS: A Tool Suite for Building GenVoca Generators. In *5th International Conference in Software Reuse*, June 1998.
6. Frédéric Besson, Thomas Jensen, and Jean-Pierre Talpin. Polyhedral Analysis for Synchronous Languages. In Agostino Cortesi and Gilberto Filé, editors, *Static Analysis*, volume 1694 of *Lecture Notes in Computer Science*, pages 51–68. Springer, 1999.
7. Patrick Borras, Dominique Clément, Thierry Despeyroux, Janet Incerpi, Gilles Kahn, Bernard Lang, and Valérie Pascual. CENTAUR: the System. *SIGSOFT Software Eng. Notes*, 13(5):14–24, November 1988.

8. Bull Cp8. Odyssey lab. http://www.cp8.bull.net/odyssey/javaa.htm.
9. Rémi Forax, Etienne Duris, and Gilles Roussel. Java Multi-Method Framework. In *International Conference on Technology of Object-Oriented Languages and Systems (TOOLS'00)*, November 2000.
10. IBM. Bean Markup Language. http://www.alphaworks.ibm.com/formula/bml.
11. IBM. XML Editor Maker. http://www.alphaworks.ibm.com/tech/xmleditormaker.
12. INRIA. The Coq proof assistant. http://coq.inria.fr/.
13. Martin Jourdan, Didier Parigot, Catherine Julié, Olivier Durin, and Carole Le Bellec. Design, implementation and evaluation of the FNC-2 attribute grammar system. In *Conf. on Programming Languages Design and Implementation*, pages 209–222, White Plains, NY, June 1990. Published as *ACM SIGPLAN Notices*, 25(6).
14. Uwe Kastens, Peter Pfahler, and Matthias Jung. The Eli system. In Kai Koskimies, editor, *Compiler Construction CC'98*, volume 1383 of *Lect. Notes in Comp. Sci.*, portugal, April 1998. Springer-Verlag. Tool demonstration.
15. Paul Klint. A Meta-Environment for Generating Programming environments. *ACM Transactions on Software Engineering Methodology*, 2(2):176–201, 1993.
16. Marjan Mernik, Nikolaj Korbar, and Viljem Zumer. LISA: A Tool for Automatic Language Implementation. *ACM SIGPLAN Notices*, 30(4):71–79, April 1995.
17. Pierre-Etienne Moreau, Christophe Ringeissen, and Marian Vittek. A Pattern-Matching Compiler. In *Workshop on Language Descriptions, Tools and Applications (LDTA)*, volume 42-2. Electronic Notes in Theoretical Computer Science, April 2001.
18. W3C Note. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/SOAP/, May 2000.
19. Jens Palsberg and C. Barry Jay. The Essence of the Visitor Pattern. In *COMPSAC'98, 22nd Annual International Computer Software and Applications Conference*, Vienna, Austria, August 1998.
20. Jens Palsberg, Boaz Patt-Shamir, and Karl Lieberherr. A New Approach to Compiling Adaptive Programs. In Hanne Riis Nielson, editor, *European Symposium on Programming*, pages 280–295, Linkoping, Sweden, 1996. Springer Verlag.
21. W3C Recommendations. XSL Transformation, version 1.0. http://www.w3.org/TR/xslt, November 1999.
22. Thomas Reps and Tim Teitelbaum. The Synthesizer Generator. In *ACM SIGSOFT/SIGPLAN Symp. on Practical Software Development Environments*, pages 42–48. ACM press, Pittsburgh, PA, April 1984. Joint issue with Software Eng. Notes 9, 3.Published as ACM SIGPLAN Notices, volume 19, number 5.
23. W3C. eXtensible Markup Language (XML). http://www.w3.org/XML/.
24. Brad Wait. Using XML in Oracle Database Applications. "http://technet.oracle.com/tech/xml/info/index2.htm?Info&htdocs/otnwp /about_oracle_xml_products.htm", November 1999.