SmartTools: a Development Environment Generator based on XML Technologies*

Isabelle Attali, **Carine Courbis**, **Pascal Degenne**, **Alexandre Fau**, **Joël Fillon**, **Didier Parigot**
INRIA Sophia - OASIS Project
2004, Route des Lucioles BP 93
06902 Sophia-Antipolis, France
+33 4 92 38 75 56
First.Last@sophia.inria.fr

Claude Pasquier
Bull CP8
68, Route de Versailles
78430 Louveciennes, France
+33 4 92 38 71 64
Claude.Pasquier@sophia.inria.fr

Claudio Sacerdoti Coen
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 [7, 8, 10, 15, 16, 17, 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 [6]. 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 uses XML technologies [24] extensively.

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

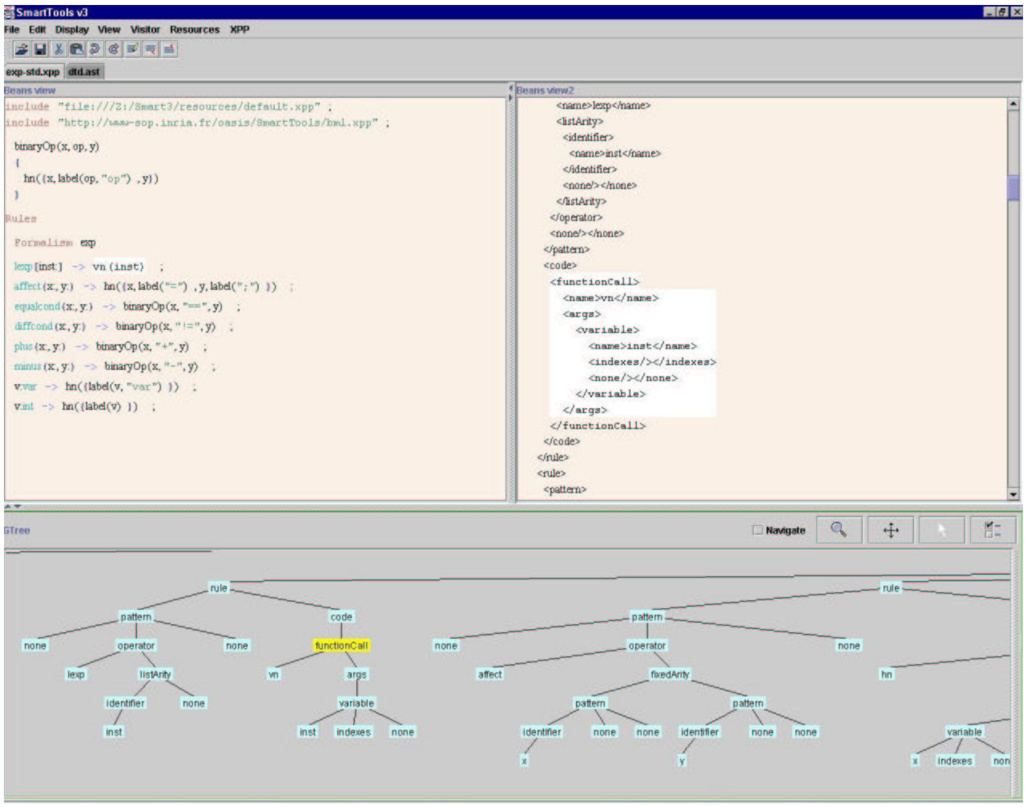
2 ABSTRACT SYNTAX AND SEMANTIC MANIPULATION

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 classes that extend a DOM implementation for each operator. It offers a structure editing environment for the language defined by this DTD for free. During editing, the system guarantees that XML documents remain valid and well-formed. When additional information is provided in DTDs, it can automatically generate a set of pretty-printers and a parser to offer a user-friendly syntax for languages. In the future, SmartTools will 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 during DTD importation ([25, 13] use the same technique). Such a typed structure is necessary to apply the visitor design pattern technique [20, 21] (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 [6]. Additionally, it is possible to dynamically customize the behavior of visitors by using visitor-specific aspects [4] e.g., an aspect is used 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 [5] project to describe a high-level logic used by the Coq proof assistant [14] 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 de-



```
include "file:///Z:/Smart3/resources/default.xpp" ;
include "http://www-sop.inria.fr/oasis/SmartTools/bml.xpp" ;

 binaryOp(x, op, y)
 {
   hn({x, label(op, "op") , y})
 }

Rules

 Formalism exp

 lexp[inst:] -> vn(inst)  ;
 affect(x:, y:) -> hn({x, label("=") , y, label(";") }) ;
 equalcond(x:, y:) -> binaryOp(x, "==", y)  ;
 diffcond(x:, y:) -> binaryOp(x, "!=", y)  ;
 plus(x:, y:) -> binaryOp(x, "+", y)  ;
 minus(x:, y:) -> binaryOp(x, "-", y)  ;
 v:var -> hn({label(v, "var") })  ;
 v:int -> hn({label(v) })  ;
```

```
      <name>lexp</name>
      <listArity>
        <identifier>
          <name>inst</name>
        </identifier>
        <none/></none>
      </listArity>
    </operator>
    <none/></none>
  </pattern>
  <code>
    <functionCall>
      <name>vn</name>
      <args>
        <variable>
          <name>inst</name>
          <indexes/></indexes>
          <none/></none>
        </variable>
      </args>
    </functionCall>
  </code>
</rule>
<rule>
  <pattern>
```

Figure 1: SmartTools environment showing different views of the same tree structure

signers easily create languages and their associated semantic tools. Thus, SmartTools is bootstrapped by using itself 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 extensive 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 that know how to calculate (thanks to a few specific constraints) their relative position in the tree structure. 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) [12] syntax. The XSL Transformation language (XSLT) [2] 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 [19].

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. It is planned to move a step forward and adapt the system to respect the W3C Simple Object Access Protocol (SOAP) [3] 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

From the abstract syntax definition of programming (e.g. Java) or domain-specific languages, it is possible to easily generate an interactive environment with SmartTools. This latter 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. Thus, we can focus on semantics tools [23] (visitor technics, aspect-oriented programming).

There are already some examples of easy and successful integration of research tools [18, 9, 11], and technology transfer in industrial environment [1]. 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] Bull CP8, Odyssey Lab. http://www.cp8.bull.net/odyssey/javaa.htm.

[2] W3C recommendations, XSL Transformation, version 1.0. http://www.w3.org/TR/xslt, November 1999.

[3] W3C note, simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/SOAP/, May 2000.

[4] Aspect-Oriented Programming. http://www.parc.xerox.com/csl/projects/aop/.

[5] The Hypertextual Electronic Library of Mathematics. http://www.cs.unibo.it/˜ asperti/HELM/.

[6] I. Attali, C. Courbis, P. Degenne, A. Fau, and D. Parigot. SmartTools: a Generator of Interactive Environments Tools. In *Compiler Construction CC'2001*, volume 2027 of *Lecture Notes in Computer Science*, Genova, Italy, April 2001. Springer-Verlag. Tool demonstration.

[7] L. Augusteijn. The Elegant Compiler Generation System. In P. Deransart and M. 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, Sept. 1990. Paris.

[8] D. Batory, B. Lofaso, and Y. Smaragdakis. JTS: A Tool Suite for Building GenVoca Generators. In *5th International Conference in Software Reuse*, June 1998.

[9] F. Besson, T. Jensen, and J.-P. Talpin. Polyhedral Analysis for Synchronous Languages. In A. Cortesi and G. Filé, editors, *Static Analysis*, volume 1694 of *Lecture Notes in Computer Science*, pages 51–68. Springer, 1999.

[10] P. Borras, D. Clément, T. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. CENTAUR: the System. *SIGSOFT Software Eng. Notes*, 13(5):14–24, Nov. 1988.

[11] R. Forax, E. Duris, and G. Roussel. Java Multi-Method Framework. In *International Conference on Technology of Object-Oriented Languages and Systems (TOOLS'00)*, Nov. 2000.

[12] IBM. Bean Markup Language. http://www.alphaworks.ibm.com/formula/bml.

[13] IBM. XML Editor Maker. http://www.alphaworks.ibm.com/tech/xmleditormaker.

[14] INRIA. The Coq proof assistant. http://coq.inria.fr/.

[15] M. Jourdan, D. Parigot, C. Julié, O. Durin, and C. 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).

[16] U. Kastens, P. Pfahler, and M. Jung. The Eli system. In K. Koskimies, editor, *Compiler Construction CC'98*, volume 1383 of *Lect. Notes in Comp. Sci.*, portugal, Apr. 1998. Springer-Verlag. Tool demonstration.

[17] P. Klint. A Meta-Environment for Generating Programming environments. *ACM Transactions on Software Engineering Methodology*, 2(2):176–201, 1993.

[18] M. Mernik, N. Korbar, and V. Zumer. LISA: A Tool for Automatic Language Implementation. *ACM SIGPLAN Notices*, 30(4):71–79, Apr. 1995.

[19] P.-E. Moreau, C. Ringeissen, and M. 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.

[20] J. Palsberg and C. B. Jay. The Essence of the Visitor Pattern. In *COMPSAC'98, 22nd Annual International Computer Software and Applications Conference*, Vienna, Austria, Aug. 1998.

[21] J. Palsberg, B. Patt-Shamir, and K. Lieberherr. A New Approach to Compiling Adaptive Programs. In H. R. Nielson, editor, *European Symposium on Programming*, pages 280–295, Linkoping, Sweden, 1996. Springer Verlag.

[22] T. Reps and T. Teitelbaum. The Synthesizer Generator. In *ACM SIGSOFT/SIGPLAN Symp. on Practical Software Development Environments*, pages 42–48. ACM press, Pittsburgh, PA, Apr. 1984. Joint issue with Software Eng. Notes 9, 3.Published as ACM SIGPLAN Notices, volume 19, number 5.

[23] M. van den Brand, M. Mernik, and D. Parigot, editors. *LDTA'01 First Workshop on Language Descriptions, Tools and Applications*, Electronic Notes in Theoretical Computer Science. ETAPS'2001, Elsevier, April 2001. Genova, Italy.

[24] W3C. eXtensible Markup Language (XML). http://www.w3.org/XML/.

[25] B. 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.