SmartTools: a Generator of Interactive Environments Tools [⋆]

Isabelle Attali, Carine Courbis, Pascal Degenne, Alexandre Fau, Didier Parigot, and Claude Pasquier

INRIA-Sophia - 2004 Route des Lucioles BP 93 06902 Sophia Antipolis Cedex
{First.Last}@sophia.inria.fr

Abstract. SmartTools is a development environment generator that provides a structure editor and semantic tools as main features. The well-known visitor pattern technique is commonly used for designing semantic analysis, it has been automated and extended. SmartTools is easy to use thanks to its graphical user interface designed with the Java Swing APIs. It is built with an open architecture convinient for a partial or total integration of SmartTools in other environments. It makes the addition of new software components in SmartTools easy. As a result of the modular architecture, we built a distributed instance of SmartTools which required minimal effort. Being open to the XML technologies offers all the features of SmartTools to any language defined with those technologies. But most of all, with its open architecture, SmartTools takes advantage of all the developments made around those technologies, like DOM, through the XML APIs. The fast development of SmartTools (which is a young project, one year old) validates our choices of being open and generic.The main goal of this tool is to provide help and support for designing software development environments for programming languages as well as application languages defined with XML technologies.

Key words. Program transformation, Software development, 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 [6, 7, 9, 14–16, 21]. It is imperative that the research community creates technologies to enhance the quality of software development and increase the developers productivity. Those goals are addressed by the SmartTools framework and research. It is composed of a set of generic and interactive software components organized in a modular architecture.

The reason for building a generic integrated environment, rather a collection of independent specific tools, is that integration enables the sharing of common services. It is also important, for each new language created, such as domain

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

specific languages to propose and generate an interactive and uniform environment. The quality of these interactive environments must be as close as possible to those proposed by industrial distributors like the Visual Studio environment of Microsoft. Finally, the architecture of this system must be conceived with the aim of facilitating the integration of research tools and widely used tools (e.g. standardised Application Programming Interfaces, APIs).

These above requirements have justified our choice of Java and XML technologies. The SmartTools system is completely written with the Java programming language and strongly uses XML technologies. The SmartTools framework is a natural successor of the Centaur system [9] and uses the same basic concepts, as the abstract syntax tree (AST) specification or formalism.

From an AST specification describing a given language L, SmartTools automatically generates a structure editor dedicated to this language. So the end-user (developer) can edit any L file thanks to this uniform structure editor with generic visualisation tools (see Figure 1). For example, we can have many views of a document: one for the text representation, one for the structure editing menu and another for graphical representation.

With this basic environment, the designer of a given language is able to easily define and implement using the generic tools, a set of specific applications for his language. As we are using the Java Swing API, the graphical user interface has a great quality and is easily configurable.

In the following, we will only focus on the important aspects of SmartTools framework: the modular and extensible architecture, powerful and simple semantics tools and the utilisation of XML technologies. The system is open to other developments and uses standard mechanisms as XML for data exchange.

2 The Architecture of the SmartTools Framework

SmartTools is made of several independent software components that communicate with each other through an asynchronous messaging system. One component, the message controller, is in charge of controlling the flow of messages and delivering them to their destinations. All other components have to register on the message controller to let it know what types of messages they want to receive. Then they will receive only those types of messages and never be disturbed by the others. Information carried in messages is serialised in XML format. The design of this messaging system has proven to be simple, efficient, and easy to maintain.

Thanks to that component-based architecture, we designed a document/views relationship that gives us the possibility to have several different views showing one document each with a different presentation.

If for some specific need one wants to design a new type of view, it is not necessary for the designer of this new view to deal with all the program APIs. It can be done by extending a default view component and adding some specific code to describe what happens when that view receives messages notifying any modification of the document. This design makes SmartTools seamlessly extensible for any kind of specific purpose.

It was not difficult using such a component-based architecture to take advantage of other tools developed at INRIA such as ProActive [4, 11] to make a distributed version of SmartTools. Another benefit is that SmartTools components can be used by other applications without the need of the whole system. We made some successful experiences of SmartTools components integration in third parties environment [17, 12, 8].

We also easily connected new components like XML parsers (Sax, Xerces) and some graphic components like a graph server based on the Koala Graphic toolbox [3].

3 Semantic Analysis within the SmartTools Framework

For any tree manipulation, it is important to have a simple and powerful programming technique. This technique may neither require a high level of expertise nor an advanced knowledge about the tool.

With Java, an Object Oriented language, it is natural to use a well-known programming technic: the Visitor Design Pattern [13, 18]. The advantage of this technique is the ability to reuse code and to specify dynamic semantic as an evaluator (see Figure 1). From the specification of an abstract syntax, it is possible to automatically generate Java source code, like a default depth-first traversal for example.

We have introduced a generic visitor concept to factorize identical behaviours applying to the nodes of a tree. With this technique we have defined one visitor only to graphically display the tree, for any language. We also use a Java Multi-Methods implementation [12] to fill many deficiencies [18, 19] due to the visitor implementation based on the Java reflect APIs. Additionally with this technique, we gain lisibility without losing efficiency.

We use DOM (level 2) as Tree API where each node has the same type. In SmartTools we extend this behaviour by typing (sub-classing) each node according to a given formalism (AST). Thus, we can use the Visitor Pattern technique on DOM Tree. Many existing applications use visitor patterns (Generic Java [10], Dynamic Java [2], ...) and connecting them in SmartTools is very easy.

4 Using XML technologies

As XML will be more and more used as a communication protocol between applications, we wanted to be able to handle any XML document in SmartTools [20]. Any XML document importing a DTD (Document Type Definition) has a typed structure. That DTD describes the nodes and their types, that is very similar to our AST formalism.

In order to obtain this result, we have specified and implemented a tool which converts a DTD formalism into an AST equivalent formalism. With this conversion, we automatically offer a structure editing environment for all languages defined with XML in the SmartTools framework. It is important to note that XML documents produced by SmartTools are well-formed.

So far, all the features of a DTD are not properly considered, by instance the importation of other DTDs (and namespaces), but that should be fixed in the near future. We are also studying XML schemas and RDF (Resource Description Framework) schemas, the successors of DTD.

Thus any application that respects the implementation of the APIs, can be XML-compliant. All the manipulated trees in SmartTools are Java DOM Trees to ease the integration with other tools and to have a very open data structure.

We offer a tool to automatically generate parsers. This tool can be useful for a designer to define a user-friendly concrete syntax for his language. But, extra data are required in the definition of the language.

We have also integrated the XSL (XML Style-sheet Language) specifications that describe the layout of a document as well as the XSLT (XSL Transformation).

5 Conclusion

SmartTools offers a quality development environment platform for research tools and benefits from large fields of applications thanks to XML technologies. The rapid evolution of SmartTools confirms our choices in term of modular architecture which facilitates the integration of other Java developments. In particular the choice of Java makes it possible to obtain a great quality graphical user interface with low development effort. Moreover the XML components, thanks to an open architecture, offers low cost advantages to SmartTools and broader application fields. We already have some examples of successful and easy integration of research tools [1, 8, 12], and technology transfer in industrial environment [5]. In both cases, the great quality of interactive environment, was the determinant element.

Acknowledgements

We have much benefited from discussions with Colas Nahaboo, Thierry Kormann and Stéphane Hillion on the topic of XML technologies. We would also like to thank Gilles Roussel, Etienne Duris and Rémy Forax for their helpful comments of their Java Multi-Methods implementation.

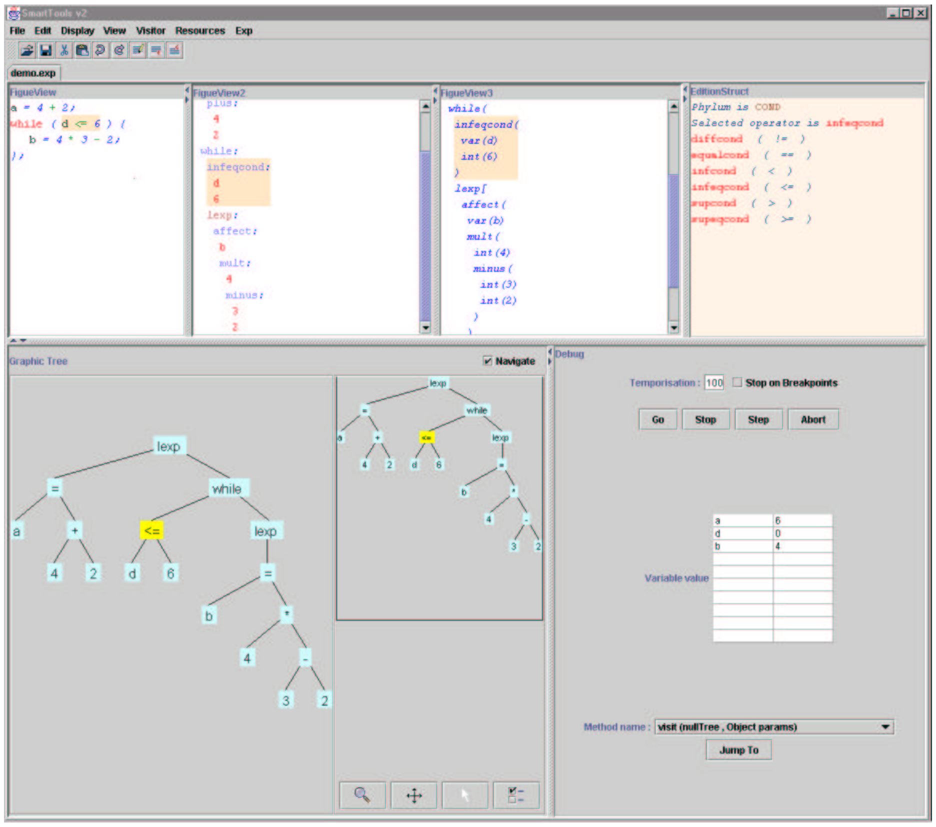
References

1. http://marcel.uni-mb.si/lisa/.
2. http://www-sop.inria.fr/koala/djava/index.html.

3. http://www-sop.inria.fr/koala/koala.html.
4. http://www-sop.inria.fr/sloop/javall/index.html.
5. http://www.cp8.bull.net/odyssey/javaa.htm.
6. 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.
7. Don Batory, Bernie Lofaso, and Smaragdakis. JTS: A tool suite for building genvoca generators. In *5th International Conference in Software Reuse*, June 1998.
8. 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.
9. 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.
10. Gilad Bracha, Martin Odersky, David Stoutamire, and Philip Wadler. Making the future safe for the past: Adding genericity to the java programming language. In *Proc. OPPSLA'98*, October 1998.
11. D. Caromel, W. Klauser, and J. Vayssiere. Towards seamless computing and metacomputing in java. In Geoffrey C. Fox, editor, *Concurrency Practice and Experience*, volume 10 of *Wiley and Sons, Ltd*, pages 1043–1061, September 1998.
12. 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.
13. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison Wesley, Reading, MA, 1995.
14. 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).
15. 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.
16. Paul Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering Methodology*, 2(2):176–201, 1993.
17. Marjan Mernik, Nikolaj Korbar, and Viljem Zumer. LISA: A tool for automatic language implementation. *ACM SIGPLAN Notices*, 30(4):71–79, April 1995.
18. 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.
19. 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.
20. Claude Pasquier and Laurent Théry. A distributed editing environment for xml documents. In *First ECOOP Workshop on XML and Object Technology (XOT2000)*, June 2000.
21. 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.

Annex



```
a = 4 + 2;
while ( d <= 6 ) {
   b = 4 * 3 - 2;
};
```

```
 plus:
  4
  2
while:
 infeqcond:
  d
  6
 lexp:
  affect:
   b
   mult:
    4
    minus:
     3
     2
```

```
while(
 infeqcond(
  var(d)
  int(6)
 )
 lexp[
  affect(
   var(b)
   mult(
    int(4)
    minus(
     int(3)
     int(2)
    )
```

```
Phylum is COND
Selected operator is infeqcond
diffcond  ( != )
equalcond  ( == )
infcond  ( < )
infeqcond  ( <= )
supcond  ( > )
supeqcond  ( >= )
```

Temporisation : 100 Stop on Breakpoints

Variable value

a	6
d	0
b	4

Method name : visit (nullTree , Object params)

Fig. 1. SmartTools