Software Factory on top of Eclipse: SmartTools

Agnès Duchamp[1], Shouhéla Farouk Hassam[1], Stephanie Mevel[1] and Didier Parigot[2],

[1] Ecole PolyTech'Nice Sophia
930, Route des Collles, PB 145
F-06903 Sophia-Antipolis CEDEX, France
name@polytech.unice.fr

[2] INRIA Sophia-Antipolis
2004, route des Lucioles - BP 93
F-06902 Sophia-Antipolis CEDEX, France
Didier.Parigot@inria.fr,
Tel: 33 4 92 38 50 01
http://www.inria.fr/smartool/

Abstract

For tree years, notions of Software factory or of Rich Client Platform have appeared in order to automate the development process. Similarly, since 2000, our research *team* has designed an approach driven by DSL. The SmartTools prototype is a result of this approach. It did not consider IDE (Integrated Development Environment) aspects. In this article, describe the first experience on how our tool has been integrated into the Eclipse Environment. First results show a perfect complementarily. Indeed, we did not modify our application code. In addition to that, this approach enables at makes it possible to produce plug-ins quicker or faster (especially plug-ins for a DSL). Our motivation is to offer generation mechanism to Eclipse plug-in developers.

Introduction

With the increasing use of computer science within the information society, it is necessary to reconsider the way in which software is developed in order to quickly produce families of flexible and reliable ubiquitous applications. Concepts such as Component-Oriented Programming [1] (OGSi), Service-Oriented Architecture (SOA), Aspect-Oriented Programming (AOP) [13], and Model-Driven Engineering (MDE) [6] have been proposed to address these challenges. They can even be merged to create the notion of Software Factory [5][7] (Microsoft Domain-Specific Language Tools) or Rich Client Platform [3] (Eclipse Foundation).

Since 2000, inspired by our previous research works, our team developed a Software factory approach named SmartTools [8][9][10][14]. The goal of this factory is to automate application development linking Aspect-Oriented Programming approach [13], DSL-Oriented Programming approach [2] and Component-Oriented Programming approach. However, the aspects only focused on development environment were never given priority. Thus, to have Eclipse at ours disposal is an ideal context for our approach's merger. In this article, we explain our feasibility study for this merger.

Our first results show that this integration does not imply a reorganization of our approach. In addition, the ability of our generation tools to produce plug-in seems to be very efficient. Indeed, our tools are able to generate a model (Abstract syntax Tree based on XML Document Object Model, DOM) for a language from the declarative description of the abstract syntax. SmartTools accepts various formats like DTD, XML Schema, abstract syntax or UML [6].

Furthermore SmartTools is able to generate various concrete syntax or graphical views. Indeed we defined a language to link firstly the abstract syntax and the concrete one, and secondly various graphical views. As results for a specific DSL, on one hand we are able to generate a plug-in linked to the concrete syntax parser and on the other hand, several XSL transformations are available to edit the DSL. In that way, we are able to transform our components (about thirty in the SmartTools context) in Eclipse plug-ins thanks to few code lines. The first part of the article describes briefly the progress report of SmartTools. The second part insists on the strong features of the merger. Then we will conclude by the extension of this study.

1. Short presentation of SmartTools Software Factory

SmartTools [8] (http://www-sop.inria.fr/smartool), our research prototype, is a concrete example of a Software Factory. Our main research interests hinge around the three following concepts:

- A development driven by the specification of Domain-Specific Language (DSL);
- A separation of concerns (AOP) directly defined on these DSLs;
- And a Service-Oriented Architecture (SOA) based on dynamically extensible components.

One of the main ideas behind the design of SmartTools (and consequently behind the design of the generated DSL tools) is to model the business logic of each concern in a technology-free manner which can then be used to generate platform-specific code (see `Figure 1`Figure `1:`).

The following five concerns have been taken into consideration:

- The description of the structure of the language (with a meta-language, named Absynt DSL,;
- The description of the method signatures and the traversal of the semantics analyses, (Xprofile DSL);
- The graphical representation of the model, (Cosynt DSL); Several views of a data model can be defined, such as a structured editor in order to more easily create and update instances of this model (programs or documents).
- The component meta-model, (Cdml DSL), It is as tightly integrated as possible with the application requirements. In particular, it enables to specify the provided and required services.
- The GUI meta-model, (Lml DSL); It describes a possible configuration of a GUI for a given application.

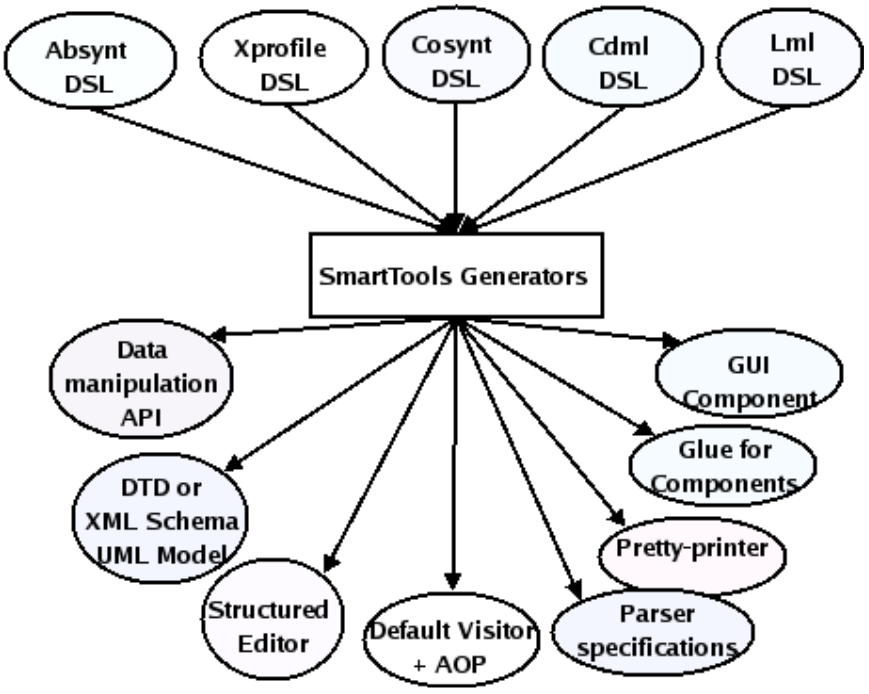

Figure 1: SmartTools DSLs and MDE approach.

SmartTools is heavily bootstrapped; that is it internally uses its technology to develop its own models. Through the development of these models, our approach in integrating the mentioned paradigms and technologies has been intensively tested and refined. Since then, SmartTools has been used to produce tools for many diverse languages such as SVG, DTD, XML schema, CSS, WSDL, and BPEL. The SmartTools framework represents approximately 100 000 lines of Java source code before the generation stage and 1 000 000 lines after. This ratio shows the efficiency of this development approach based on generative programming.

2 SmartTools components to Eclipse plug-ins

This integration has two goals:

- To help the developing plug-in with our generation tools
- It provides a real development Environment for our DSL's thanks to Eclipse in order to make the use of our approach easier.

As the two systems SmartTools and Eclipse are complex, we split up our approach into three directions:

- Development of complete environment for each DSL of SmartTools;
- Immersion of SmartTools basic components for a first instantiation of our approach;
- Integration of the several concepts or techniques developed in SmartTools into Eclipse framework.

2.1 Transformation of basic SmartTools components into Eclipse plug-ins

For the second direction, it is necessary to embed SmartTools Core into Eclipse. In order to make it, we created a first plug-in containing the core of SmartTools (st-core plug-in, see Figure 4). This plug-in will certainly be divided into many shorter plug-ins later. SmartTools was already structured as component with a particular organisation. For instance, all Java source files, XSL or CSS files generated by SmartTools are under a specific directory called “generate” in order to differ from other data of the component. Into this organisation, we just had to add the information link to the Eclipse plug-in concept (plugin.xml, .project, Manifest.mf description files).
After that**,** the code produced by SmartTools (and also generate by SmartTools for each component) was accessible via Eclipse. In this way, it is possible to use the SmartTools functionalities, such as code generation in Eclipse. More precisely, the link generator to our Cdml language (notion close to the plug-in notion) makes it possible to call for each description (DSL) the associated generator.
At this stage**,** SmartTools technologies are not used for each plug-in produced. Especially, it was necessary to make the link between the text of Eclipse editor (associated to one of our DSLs) and the tree (model or AST) given in input of the DSL SmartTools generators.

2.2 Integration of the SmartTools generated model (AST)

From a meta-model or a language description (DTD, XML Schema, or our Absynt meta-language), SmartTools produce a *logical model*. This model is based on the W3C‘s model DOM (Xml Document Objects Model). We wanted to check the possibility of merging this model generation into Eclipse. Indeed, we wanted to produce for each DSL of SmartTools a basic IDE in Eclipse environment. It is possible to call our generator (small compiler) from edited text in Eclipse editor. To check the validity of a produced model, we associated the Outline view (specific Eclipse view) to each editor (see Figure 2). All of our models are based on DOM. So, we created generic classes in order to offer this Outline view automatically for all the plug-ins. In a classical way, the developer has to write these classes by himself to obtain this Outline view.

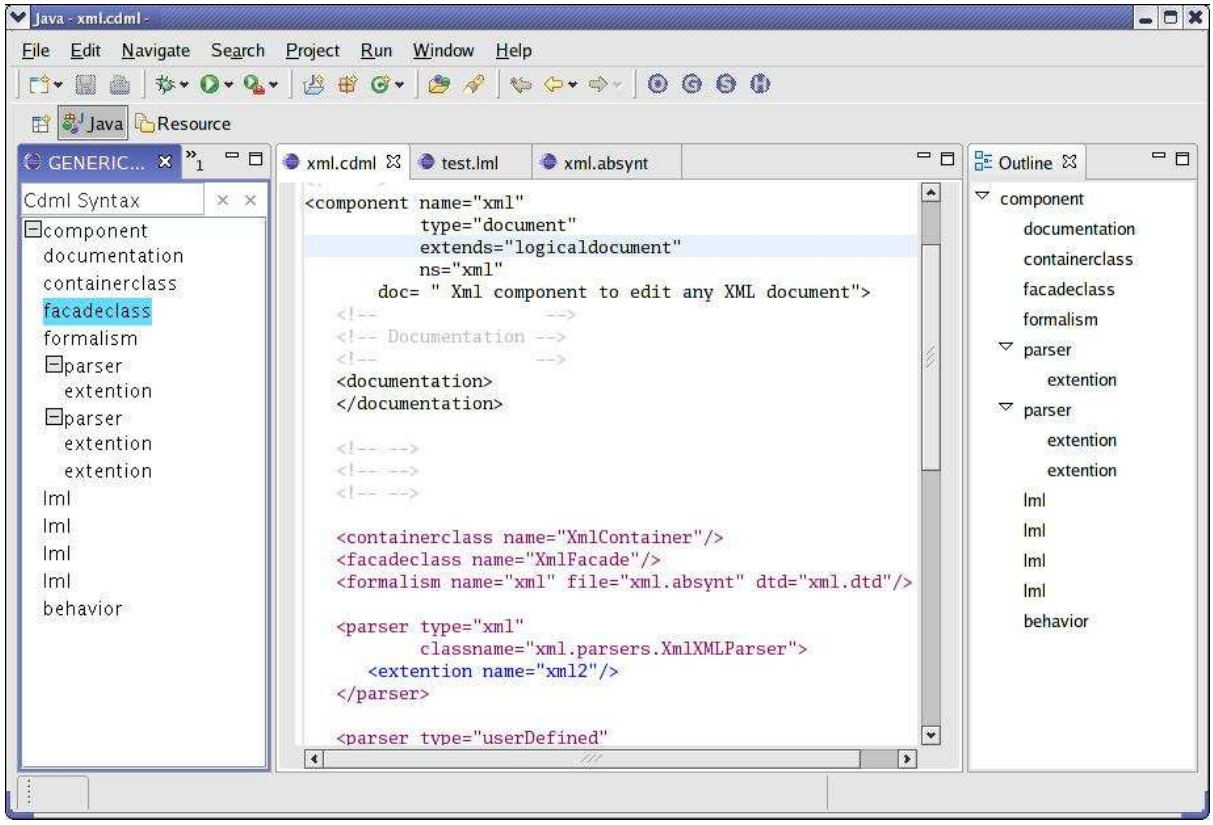

Figure 2: Generic view, Editor view and Outline view of Cdml DSL

2.3 Integration of the « graphical view » model generated by SmartTools

A declarative specification (Cosynt DSL) makes it to create different concrete forms and several graphical views for a given language. The generator of Cosynt produces automatically the parser to create the model from the text and several XSL transformations to offer graphical views (structural edition). This technology was integrated into Eclipse thanks to the Swing technology.

We can highlight a mechanism of SmartTools, the graphical object selection which enables the views to communicate by the means of the logical model (by using XPath for instance). In fact, when an item is selected in a graphical view, the item calculates its path to the root and a selection message is sent to the logical model. Then, when the message is received, the logical model informs all the graphical views linked to this model that the item is selected. For instance, the Figure 3 show three graphical views (SmartTools views) of xml.cdml with a selection on a Container item (node). As the result, this mechanism of selection driven by the logical model is very simple and offers for free to plug-ins developers. Moreover, the syntactical colours are directly defined on the notion of the logical model, the AST. Furthermore, each entity of the AST is described by a particular CSS (colour, height, font …) which use by SmartTools graphical view

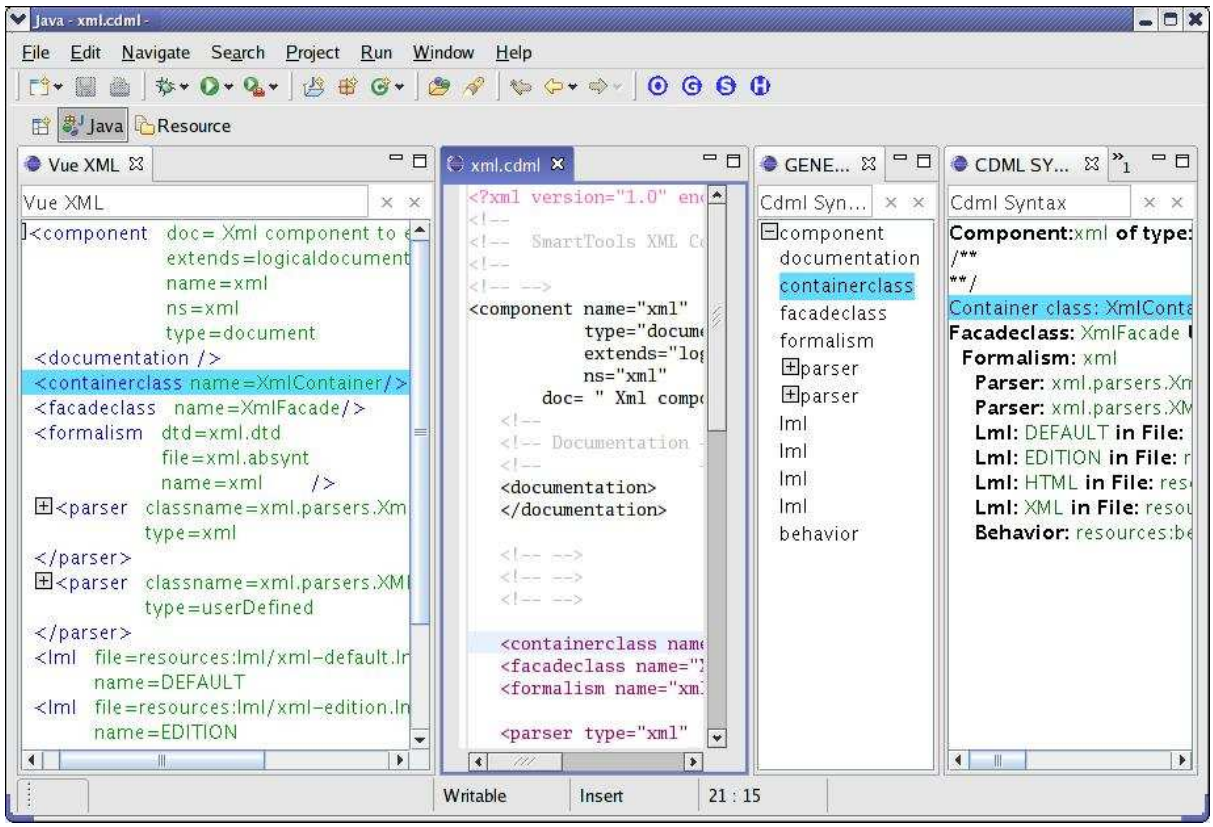

Figure 3: Generic XML view, Eclipse Editor, Generic View, Concrete Syntax view of an Cdml DSL example: xml.cdml .

2.4 Generic plug-ins conception

In order to encapsulate these two techniques of generation, we made basic plug-ins (generics ones). Our plug-ins (made thanks to our approach) depend on these generic plug-ins (for instance st_editor plug-in in Figure 4). This factorization enables to reduce the code to write in order to obtain all the functionalities (see in Figure 4, the size of Editor class with this factorization). For instance, to make a graphic view, we only need to write a single line of code.

2.5 PDE extension within the SmartTools context

In order to get a better integration of our automatic production of plug-ins, we have extended the Plug-in Development Environment (PDE) for our context (our generated code). Especially, this extension calls

SmartTools generation code during the plug-in creation and produces java source files in order to obtain an editor.

2.6 Transformed SmartTools components:

In this context, we have transformed and produced, in few months, ten plug-ins: (see `Figure 4`)

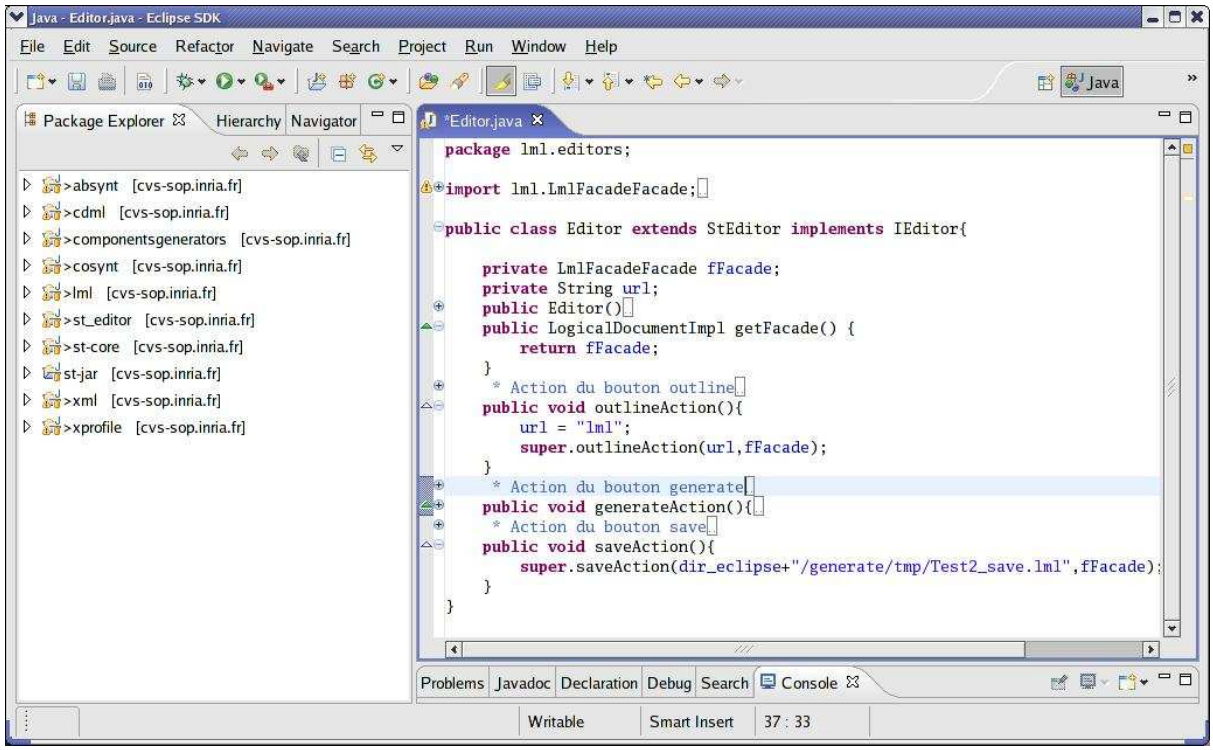

Figure 4: List of plug-ins of SmartTools components and the Editor Class of Lml DSL plug-ins.

But over all, thanks to the integration of these two technologies, all our components and theirs views can be easily integrated into Eclipse. This represents more then thirty plug-ins.

3 Conclusion

This first immersion shows perfectly that the Eclipse environment and our approach are complementary. Indeed, this integration does not require any modification of the SmartTools code. For each plug-in, the developed code corresponds to associate parts of Eclipse environment. Moreover, we have developed in no time our plug-ins associated to our DSLs thanks to generation. So we are planning a new distribution of SmartTools thought a set of plug-ins.

However, we have not worked on plug-ins communications yet. Especially extensions and entry points associated to plug-ins are currently studied to transpose our Component approach. Indeed, our Service-Oriented Architecture might be transposable to these notions. We want to offer the OSGi platform support to our components. After this first study, we have to work on: using all of Eclipse's capacity and diffusing our approach in Eclipse environment. Then our research on Aspect-Oriented Programming and program automatic transformation will be downloading by this way.

References

[1] C. Szyperski with D. Gruntz and S. Murer, *Component Software : Beyond Object-Oriented Programming*, Addison-Wesley/ACM press 2002, ISBN 0-201-74572-0

[2] Arie van Deursen , Paul Klint , Joost Visser, Domain-specific languages: an annotated bibliography, ACM SIGPLAN Notices, v.35 n.6, p.26-36, June 2000

[3] Eclipse. http://www.eclipse.org

[4] J. Arthorne and C. Laffra, *Official Eclipse 3.0 FAQs,* Eclipse series, p. xxxiv + 386, pubAW, 2004.

[5] Software Factories. http://www.softwarefactories.com.

[6] F. Budinsky, D. Steinberg, R. Ellersick and B.Grose, *Eclipse Modeling Framework*, Addison Wesley Professional, 2003.
[7] J.Greenfield, K. Short, S. Cook and S. Kent, *Software Factories : Assemblig Applications with Patterns, Models, Frameworks, and Tools,* John Wiley & Sons, 2004.
[8] D. Parigot and C. Courbis, *Domain-driven developpment: the SmartTools Software Factory,* Technical Report RR-5588, INRIA, Sophia Antipolis, July 2005.
[9] C. Courbis, P. Degenne, A. Fau et D. Parigot; *Un modèle abstrait de composants adaptables.* Système à composants adpatables et extensibles, numèro thématique de la revue TSI, 23(2), 2004.
[10] C Courbis, P. Degenne, A. Fau et D. Parigot. L'apport des technologies XML et Objets pour un générateur d'environnement: SmartTools. Revue l'Objet, numéro thématique XML et les Objets , 9(3), 2003
[11] K. Czarnecki and U. W. Eisenecker, *Generative Programming : Methods, Techniques, and Applications*, Addison-Wesley, June 2000.
[12] E Gamma R. Helm, R. Johnson et J. Vlissides. *Design Patterns*. Addison Wesly, Reading. MA, 1995.
[13] G.Kiczales, J. Lamping, A Menhdhekar. C. Maeda, C. Lopez. J-M. Loingtier et J. Irwin. Aspect-Oriented Programming, ECOOP '97, volume 1241 de LNCS, pages 220-242, Juin 1997.
[14] D. Parigot, C. Courbis, P.Degenne, A. Fau, C. Pasquier, J. Fillon, C. Held, and I. Attali, Aspect and XML-oriented Semantic Framework Generator: SmartTools, In ETAPS'2002, LDTA workshop, Grenoble, France, April 2002, ENTCS