Jakarta:
A Tool Suite for Constructing Software Generators

Don Batory, Dan Miranker, David Brant
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712
{batory, miranker, brant}@cs.utexas.edu
EDCS Contract Number: F30602-96-2-0226

1 Introduction

Software component (or *software building block*) *technologies* enable customized applications or their subsystems to be synthesized quickly and inexpensively from component reuse libraries. Software that is synthetically produced is inherently more evolvable than software produced by other means: a revised system is specified by a more appropriate composition of components and then regenerated. Generators are tools that convert component compositions into optimized source code and that validate compositions to ensure consistency. From our experience in building component technologies and generators for a variety of domains [Bat92-97], there is a serious lack of support from industrial programming languages and tools to simplify generator construction. 60% of the effort in building domain-specific generators involves the creation of largely domain-independent programming infrastructure (e.g., languages for component specification, languages for component composition, etc.). Jakarta is a project to provide this common infrastructure: the *Jakarta Tool Suite (JTS)* is designed for constructing software component technologies and their generators.

Among the features of Jakarta are:

- **Extensible Languages**. Generators are compilers for domain-specific languages or compilers for general-purpose languages with domain-specific extensions. The JTS simplifies the evolution of languages and their compilers.
- **Compiler Generation**. JTS allow both base grammars of languages and their extensions to be encapsulated as primitive building blocks. This is an advance over existing domain-specific compiler-construction tools in that Jakarta merges domain-specific language/compiler technologies with software component/generator technologies. Rather than having languages become progressively more bloated and unwieldy, Jakarta makes it possible to select the specific features that one needs to develop domain-specific applications, and to produce a customized (lean) language and its compiler for writing such applications quickly and cheaply through component composition/generation. Language components may include domain-independent extensions, such as templates and lexically-scoped macros, or they may encapsulate entire domain-specific generators (that include libraries of reusable domain-specific components).
- **Extensible Java**. JTS is written entirely in the Java language, or more accurately, in a superset of the Java language that is suited for building generators. We have defined component extensions to Java that support meta-programming (i.e., LISP backquote and comma) for representing and manipulating programs as data and generation scoping [Sma97] (a form of hygienic, lexically-scoped macros that is appropriate for generators) to help build the Jakarta tool set. These components, like all GenVoca components, can be easily removed from or installed into Java.

- **Automated Design Patterns**. JTS is geared for providing a solid foundation for automating object-oriented program transformations known as *design patterns* [Gam95, Tok95]. By automating the application of design patterns, one can more easily evolve existing OO applications.
- **Adaptable Software**. *GenVoca*, the software building-blocks model on which Jakarta is based, expresses systems as equations, where components are primitive terms [Bat92]. A significant form of domain knowledge that cannot be encapsulated within individual components is how target systems that meet requirements specifications can be expressed as compositions of components. We believe such knowledge can be captured by domain-specific rewrite rules that transform the design of one system (i.e., equation) into an another (equation). We are developing domain-specific algebras to express equivalences among software designs and are implementing this algebra using a high-performance expert-system technology [War96]. A software tool that encapsulates a domain-specific algebra, called a *design wizard*, will automatically apply its rewrite rules to critique hand-crafted system designs (i.e., equations), and will provide specific explanations on what can be improved and why. When integrated with components that profile target system executions (that collect performance data that is needed for system optimization), design wizards may enable *adaptable software systems*, systems that tune and evolve themselves automatically to meet the demands of applications with changing workloads.

2 Project Status and Availability

The Jakarta Tool Suite will run on **Solaris**, **Windows 95**, and **Windows NT** platforms. A beta-release is expected by the end of 1997. Demonstrations of JTS will be given this summer. For current information, release announcements, and the latest technical reports, please check our web page **`http://www.cs.utexas.edu/users/schwartz`** or contact Don Batory at **`batory@cs.utexas.edu`**.

This research is sponsored by the Defense Advanced Research Projects Agency. Technical and contractual management are provided by Rome Laboratory, USAF, under Cooperative Agreement F30602-96-2-0226.

3 References

[Bat92] D. Batory and S. O'Malley. “The Design and Implementation of Hierarchical Software Systems with Reusable Components”. *ACM Trans. on Software Engin. and Methodology*, October 1992.

[Bat95] D. Batory, L. Coglianese, M. Goodwin, and S. Shafer. “Creating Reference Architectures: An Example from Avionics”, *Symposium on Software Reusability*, Seattle Washington, April 1995.

[Bat96] D. Batory and B.J. Geraci. “Validating Component Compositions in Software System Generators”, *International Conference on Software Reuse*, Orlando, Florida, 1996.

[Bat97a] D. Batory and J. Thomas. “P2: A Lightweight DBMS Generator”. Accepted for publication in the *Journal of Intelligent Information Systems*, 1997.

[Bat97b] D. Batory, “Intelligent Components and Software Generators”, Software Quality Institute *Symposium on Software Reliability*, Austin, Texas, April 1997.

[Gam95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Pattens: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[War96] L.B. Warshaw and D.P. Miranker, "A Case Study of the Venus Approach to Rule-Based Modularity", *Conferenece on Information and Knowledge Management*, (CIKM-96), 317-325.

[Sma97] Y. Smaragdakis and D. Batory, “Scoping Constructs for Program Generators”. Technical Report 96-37, Department of Computer Sciences, University of Texas at Austin, December 1997.

[Tok95] L. Tokuda and D. Batory, "Automated Software Evolution via Design Pattern Transformations". *3rd Int. Symposium on Applied Corporate Computing*, Monterrey, Mexico, October 1995.