# Towards tools and languages for hybrid domain-specific kits

Martin L. Griss

Software Reuse Department, Software Technology Laboratory
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94301
Tel: (415) 857-8715
Email: griss@hpl.hp.com

**Abstract**

As part of HP Laboratories research into a systematic process for domain-specific reuse, we are exploring the notion of Domain-Specific Kits. Kits are comprised of compatible, domain-specific components, frameworks and glue languages, supported by a variety of technologies and tools, such as domain-specific languages, builders, generators and domain-tailored environments. We are particularly interested in *hybrid* kits, which combine both generative and compositional reuse, and domain-specific language tools to support the generative part. This paper describes our initial thinking and investigations.

**Keywords:** Reuse, kits, builders, generators, domain-specific languages, hybrid reuse.

**Workshop Goals:** Learn more about domain analysis/engineering and generative reuse methods; share and discuss information on our research program.

**Working Groups:** Kits, generative reuse, domain engineering

# 1 Background

I have been involved with HP software engineering products and processes since 1985, and with HP reuse efforts since 1989. In position papers at WISR'92 (Palo Alto) and IWSR'93 (Lucca), I described our multi-disciplinary reuse research program started at HP Laboratories in 1992[1]. This program complements HP's Corporate Reuse Program, described at IWSR'91, WISR'92 and IWSR'93. Key to our research program is an integrated approach to technology, method, process and organization issues. Our program has two major themes: *domain-specific-kits* and *flexible software factories*. The domain-specific kit research focuses on the technologies and methods for the production, use and support of kits, while the flexible software factory work concentrates on the processes, organization design, and software engineering and communication infrastructures for kit-based software development.

At WISR'92, the Domain Analysis Working Group[2] explored the different styles of domain analysis appropriate to either generative or compositional reuse. Purely generative approaches (e.g. Draco[3]) were deemed too complex for most cases, even though the payoff is high. Instead, one should try hybrid reuse, combining both generative and compositional approaches. However, current DA methods do not facilitate systematic design for hybrid reuse, or tradeoffs as to which route to take for which part of a complete application family.

At IWSR'93, the tutorial by Frakes, Batory and Devanbu on application generators did not really address these issues, nor how to design appropriate domain-specific languages for the level of generator technology chosen. Instead, it focused primarily on a few commercially available implementation techniques and tools. There were some useful experience reports, but no guidelines for hybrid reuse tradeoffs[4, 5].

# 2 Position

It is my belief that there are several important issues involved here, and that they should be addressed systematically. We need:

- methods and supporting technology to design and implement hybrid kits
- guidelines and examples of a variety of simple and more complex ways of using domain-specific languages and generators within hybrid kits
- techniques and mechanisms to ensure openness and extensibility of the kit, and to smooth the boundary between generative and compositional parts
- methods and common mechanisms to produce frameworks which will ensure that independently developed can interoperate.

This position paper will summarize some of our work on kits, hybrid reuse, domain-specific languages and related issues.

# 3 Domain-Specific Kits

Our goal is to develop for HP divisions the methods and technologies to dramatically improve application construction tasks using reuse. Most HP reuse situations involve the construction of families of closely related applications. To encourage the coherent design, implementation and packaging of a variety of comaptible domain-specific workproducts, we first identify and structure the domain ("domain engineering") and then use new methods ("kit engineering") to build a "domain-specific kit" consisting of domain-specific components, supporting infrastructure and tools. The kit is then used to build one or more applications.

A typical "hybrid domain-specific kit" will contain (most of):

- A set of *components*, which are well documented, tested and packaged sets of compatible software workproducts: C functions, C++ objects, generator templates, test-files, software-bus connected mega-components, etc.
- A *framework* within which compatible components can be combined with hand-written or generated glue-language, and the addition of non-kit workproducts. The framework provides common services, key mechanisms and core functionality, ensures appropriate interoperability and customizability, and establishes conventions and mechanisms to add new compoents.
- A *glue language* to combine components and add needed functionality. This could be general purpose, a flexible scripting language, or highly domain-specific.
- Exemplary *generic applications* are pre-packaged, ready-to-run, standard applications built with the kit, with interconnected framework, components and glue. A complete application is evolved or customized using built-in customization methods and adding or replacing components, supporting a prototyping development cycle[6, 7].
- A *construction environment* provides tools such as component browser, glue language editor, application builder and generator. Debugging, testing, and construction step recording tools aid in quick assembly and modification. Using a combination of techniques, one can start from components, generic applications or from previously built user applications.
- An *execution environment* provides support for customizing and running the application, with debugger, interpreter, graphical display and monitoring, etc.

# 4 Theory and Practice of Domain-Specific Kits

We are trying to produce a theory and practice of domain-specific kits, abstracting ideas from division studies and experiences in developing and using several domain-specific kits. A complete methodology should include:

- A philosophy of how and when to use domain-specific kits, and guidelines on how to balance the cost and benefits between the development of frameworks, reusable domain-specific components and languages, and the use of traditional programming.
- A taxonomy or catalog of kit models, styles, and implementation techniques. For example, kits may be "closed and complete", or "open and extensible", providing a some needed

workproducts, and instructions and mechanisms to create, declare and register new components.

- Processes and methods to be used by kit developers and users, workproduct supporters and managers, such as "domain engineering," "kit engineering," "framework design," "component management," "generator design and implementation," "using a kit," "augmenting a kit," etc.
- A set of (customizable) tools and core technology that can be used to define and develop kits, such as domain analysis tools, language and generator kits, library and browsing tools, glue language interpreters and compilers, software-bus services, user-programming and customizing language kits, etc.
- A set of case studies which illustrate the principles, the practice and the tradeoffs. It would be useful to know language sizes, number of components, implementation costs, productivity and quality benefits, etc.

### 4.1 Domain-specific languages for hybrid kits

Language technology can be used in several ways to include domain-specificity into a hybrid kit for different audiences and situation. There are several opportunies for domain-specific"little languages"[8], such as:

- Component development and generation (from templates or rules)
- Parameter generation for parameterized components
- Glue/config language for component interconnection
- Data file/table creation
- Loader/builder scripts
- Templates/declarations for registering new components in an open kit
- End-user programming/customization

The following list illustrates some implementation choices:

- Conventional language + domain-specific library of code

  Using Basic or C, the domain-specificity is offered as a library of parts.
- Object-oriented language + domain-specific class library

  C++ and Smalltalk make it easier to create frameworks and API's that shape components by inheritance and polymorphism.
- Extensible language or preprocessor + domain-specific library

  Using macros or syntax extension capabilities in LISP and TCL, or using CPP, AWK or PERL, Yacc, or STAGE[9] as preprocessor one adds concise, domain-oriented expressions to a standard language. These simplify parameter generation, consistent use of procedures and data, and creating data files, symbol-tables and other workproducts.

- Custom full domain-specific language and environment, with domain-specific library.

  Yacc and Lex, LISP-based meta-compilers, and C-based Tree-Meta have been used in HP to produce a variety of instrument system programming languages. These configure and select parts of an instrument, create front-panel display and remote-port drivers, declare bindings to measurement routines, establish interfaces to higher-level analysis routines, and fill in data tables.

Within HP, practical systems combine several of these techniques at the same time, leading to some issues of consistency and interoperability. One way to ensure that several languages be consistent or interoperate, is to implement them using the same language kit, which can be embedded with the development and delivery environments. In addition to Lex and Yacc, small embedded interpreters (Xlisp, TCL, PERL) and simple-meta compilers (Tree-Meta, Pmeta) are quite attractive for this purpose.

## 5 Next Steps and Summary

We are defining and prototyping a series of small domain-specific kits to help drive the development of our kit design and implementation methodology and tools.

Our first kit is in the domain of task/dolist managers, using our software bus[10] for component integration. Following a minimal domain analysis and kit design, components were written in several languages (LISP, C++, TCL/TK) for item display, definition, time management, and dolist item storage managment. Alternative components and options are selected, and data-structures are defined, using a simple (LISP-based) configuration language, from which a complete application is generated. A conversational rule-based tool uses data derived from our domain analysis to present decisions and consequences to the application builder to help generate the kit configuration file.

In our next cycle, we will refine our DA method, and be more systematic about the design of the kit architecture and style. We will use hypertext tools (Kiosk[11]) to browse and display domain and kit workproducts, issues and decisions and as an interface to the kit components and documents.

We are surveying the use of hybrid kits in HP, looking at language and generator style and supporting technologies, and to better understand divisional constraints and practices. We are collecting information on available language, builder and generator technologies, and will do some analysis and prototyping to understand several of these.

## 6 Acknowledgments

I had several useful discussions with Joachim Laubsch as I prepared this position paper. Dennis Freeze, Jon Gustafson, Joe Mohan and Kevin Wentzel gave useful comments on drafts of this position paper.

## References

[1] M. L. Griss, "A multi-disciplinary software reuse research program," in *Proceedings of the 5th*

*Annual Workshop on Software Reuse* (M. Griss and L. Latour, eds.), pp. Griss–1:8, Department of Computer Science, University of Maine, Nov. 1992.

[2] M. Griss and W. Tracz, "Workshop on software reuse," *Software Engineering Notes*, vol. 18, pp. 74–85, Apr. 1993.

[3] J. M. Neighbors, "The draco approach to constructing software from reusable components," *IEEE Transactions on Software Engineering*, vol. SE-10, pp. 564–574, Sept. 1984. Presented at the ITT Workshop on Reusability in Programming, Newport, RI, September 1983, published by UCI as report RTP019.

[4] B. Frakes, "Application generators." Lecture note slides, June 1993. (Private communciation).

[5] D. S. Batory, "The genesis database system compiler: Large scale software reuse from domain modeling," in *Proceedings of the 4th Annual Workshop on Software Reuse* (L. Latour, ed.), (Department of Computer Science, 222 Neville Hall, Orono, Maine 04469), pp. 1–6, University of Maine, University of Maine, Nov. 1991.

[6] G. Fischer, "Cognitive view of reuse and redesign," *IEEE Software*, vol. 4, pp. 60–72, July 1987.

[7] J. A. Johnson, B. A. Nardi, C. L. Zarmer, and J. R. Miller, "Ace: Building interactive graphical applications," *Communications of the ACM*, vol. 36, pp. 40–55, Apr. 1993.

[8] J. Bentley, "Little languages," *Communications of the ACM*, vol. 29, pp. 711–721, Aug. 1986.

[9] J. C. Cleaveland, "Building application generators," *IEEE Software*, vol. 4, pp. 25–33, July 1988.

[10] B. W. Beach, M. L. Griss, and K. D. Wentzel, "Bus-based kits for reusable software," in *Proceedings of ISS'92, UCI, Irvine, March 6*, pp. 19–28, Mar. 1992.

[11] M. Creech, D. Freeze, and M. L. Griss, "Using hypertext in selecting reusable software components," in *Proceedings of Hypertext'91*, (Palo Alto, CA), pp. 25–38, Software and Systems Laboratory, Dec. 1991.

# 7 Biography

**Martin L. Griss** is Principal Laboratory Scientist for Software Engineering at Hewlett-Packard Laboratories, Palo Alto. As manager of the Software Reuse Department, he leads research on software reuse, domain-specific kits and flexible software factories. He works closely with HP Corporate Engineering to systematically introduce software reuse into HP's software development processes. He was previously Director of HP's Software Technology Laboratory, researching expert systems, object-oriented databases, programming technology, human-computer interaction, and distributed computing. Before that, he was an Associate Professor of Computer Science at the University of Utah, working on computer algebra and portable LISP systems (PSL). He received a Ph.D. in Physics from the University of Illinois in 1971.