Bus-Based Kits for Reusable Software

Martin L. Griss

Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94301
415-857-8715, griss@hplabs.hp.com

Abstract

Hewlett-Packard's interest in a reuse-based software construction process involves both a corporate reuse program and research into methods and technology. We describe research in domain-specific, architecture-driven, reuse-based software construction technology, specifically the notion of domain-specific kits. This work is explored in the context of distributed information management environments.

Keywords: software reuse, software bus, software construction kits, corporate reuse program, domain-specific reuse, distributed applications

1 Introduction

Hewlett-Packard's great interest in significantly improving its software process has led to several software initiatives, including a Corporate Reuse Program to make software reuse a systematic part of HP's software process for the 1990s[7]. This program is a broad, coordinated effort involving management, process, technology, education and research, for HP divisions developing *domain-specific* (or *application specific*) common architectures, frameworks and reusable components for families of related products, such as embedded software for instruments, printers, chemical and medical systems.

HP Labs and Corporate Engineering are developing a domain-specific reuse process for *develop for reuse*, *develop with reuse*, and *workproduct management*[4]. In this context, *domain* refers to set (or subset) of systems that share a significant number of common user requirements, and admit of common implementation(s). Different domains might be best implemented using different technologies, different tools, different architectures, and (hence) different development processes. For example, the use of a 4GL for the domain of certain financial problems might naturally lead to much more of a "prototyping" mode of developing. These guidelines and methods involve domain analysis, the definition of an architectural framework and components, and the construction, management and integration of these components. Corporate Engineering focuses on process, education and best practices, while HP Labs concentrates on research in processes, tools and environments.

This paper summarizes HP Labs research in component-oriented construction of novel, distributed applications, using kits, software-bus frameworks, and hypertext. Related work includes user-programmable UI components, OO methods and domain analysis.

2 New ways of building systems: Reuse based kits

Application development has become more complex, involving novel, distributed systems, multimedia, and support for collaboration, information sharing and manipulation. Use of complex, standard, object-oriented environments and interfaces, the desire for more end-user programmability and customizability, and the pressure to radically decrease time to market, require new approaches. Several technologies and methods must be used in concert to produce domain-specific application construction and execution environments, such as reuse, frameworks, generic applications, (reuse-based) prototyping, generators, user-programming languages, mega-programming, etc.

Application developers need kits of extensible components that can be combined into "domain-specific, 'yet still somewhat generic' applications", i.e., almost complete, customizable applications (or application skeletons) that need to be further tailored, or completed, by the end-user or local programmer. A kit will comprise several compatible sub-systems: a reuse library, a glue/extension language, a set of rules and constraints, a component integration framework, and supporting development/customization environment. Often a (partially domain specific) generic kit is extended with additional domain specific components and constraints, and released as a tailorable, domain specific kit. Examples are domain-specific stacks for (generic) HyperCard, which are still extensible via HyperTalk. Similar are the domain-specific AutoCAD drafting packages (e.g. landscaping), written in AutoLisp.

We are developing our first kit, using an object-oriented *software bus* architecture[2, 10, 9, 11] to produce distributed applications that are easy to customize, extend, and combine to produce application-oriented environments. Later, we will explore to what degree several kits can be themselves built from common *kit-building* technology.

3 Message based component integration

HP's CASE integration platform (SoftBench)[3] loosely couples independent software tools into powerful, customizable programming environments. Underlying control-integration technology for this 'mega-programming' [12] architecture is a Broadcast Message Server (BMS), and support to monitor or debug messages, and to encapsulate existing tools. Each tool (an independent 'mega'-component), registers its services and interests as message string patterns. Running tools broadcast their state or needs, or react to such messages, by starting other tools, logging results, or checking consistency. HP also uses BMS for other distributed applications.

Extending the BMS idea, we have developed a software bus framework (Bart)[2] for control and data integration, which provides a communication substrate that ALL components must use for data exchange and control coordination. A *Bus Manager* is invoked to establish communication paths between components, so that (subject to security) they can be monitored. Bart has three layers of service: multi-cast transport (synchronous and asynchronous) of typed messages, a database-like view mechanism for components to *publish* and *subscribe* to various information subsets and connections, and *software glue* that describes connections and message data-models. Components use ports (data and control) to permit a class of objects and its methods to be distributed across a set of components. Each component is event based, with a top loop waiting or polling for several input sources. Bart routes events via the control ports. These mechanisms make it easier to write components independently, and then combine them and their (incompatible) data-models with appropriately customized behavior.

4 Novel, distributed system prototypes

We have independently constructed several systems, with related but partially incompatible technology. Each system is evolving in similar directions. We would like to decrease redundant work and increase synergy.

Kiosk[5, 6] is a workstation-based single-user general-purpose hypertext framework built on Unix, C++ and InterViews[8]. Kiosk manipulates Unix file-based information, and interfaces well to other Unix tools and shell-scripts. It is a flexible base for experimenting with hypertext-based reuse tools and textual information management. Kiosk manages software workproducts, reusable libraries (e.g. InterViews) and other textual information (catalog entries, manual pages, header files, mail messages, etc.).

Kiosk has 3 major parts: a *semantic layer* implementing a hypertext-engine, with persistent, typed nodes and links, mapped to Unix files; a graphical and textual *browser/editor* built upon a general presentation layer; and, a schema-driven *hypertext builder*, which uses a structural description, a specification of input files and types, and rules (parsers and pattern-matchers) to identify and link 'interesting places' between files. The layers communicate via a *notification manager* which registers dependencies between semantic objects and (several) presentation objects to ensure the consistency of multiple views of the same objects.

We have built a small-team email-based Group Memory using Kiosk. Arriving email from team members, or external sources, is incrementally linked into a hypertext. It is mechanically processed for interesting places to link to, and for people to inform. It is put on a 'new' list, and an announcement sent to each person, to be displayed in an action panel on his workstation. The email can then be read, marked as seen, further linked into the global or private hypertext, etc. Multi-user aspects in the first prototype were handled via several scripts and processes attached to the notification manager in each Kiosk. While showing the correct behavior, this has proven clumsy and unreliable. We are restructuring Kiosk to use Bart so that multiple interacting/coordinated browser/editors can share one hypertext-engine (semantic layer). Our initial steps have been to connect Bart to the Kiosk notification manager, by introducting a *liaison presenter*, which appears as a presenter to the notification manager, but can accept socket based connections from the software-bus. We have interfaced both InterViews and Motif to Bart, and had to deal with differences in UI drivers and issues of 'who is in charge', 'where do events come from', and event loop conflicts.

The Physician's Workstation (PWS) is a multi-component environment to help several physicians and supporting functions deal with their daily activities. Independent components implemented in C for each activity (appointment calendar, patient record, prescription writer/drug interaction analyzer, lab record alert) interact via BMS and Unix sockets. Key components are a knowledge-base (KB), editor and global patient record database. Rules and facts describing patients, drugs and symptoms are recorded and modified here; messages from components are monitored by the KB, and inferences provide constraints and guidance to the physician. Issues involve the customizability of the systems to fit local preference, system configuration, security, etc. We have implemented a Bart model of the PWS. Originally, we tried simple re-engineering, replacing BMS calls with Bart calls. Instead, it was easier to first produce a *template* (an inner, component architecture) to correctly set up the appropriate components and ports, and then quickly embelish to emulate the behavior and communication patterns of the PWS. Within this PWS-model skeleton, we can now add more complete PWS functionality to each component.

The Matisse[1] team programming environment is a distributed, object-oriented, rule-based system for small teams of software developers. It manages fine-grain software representations as a software-hypertext and provides support for software process modeling and enactment. Matisse consists of several processes written in C. A central global database server encapsulates a relational database to present a multi-user low-level persistent object server. Developer workstations have several tool processes (editor, browser, etc.) connected to a local, active (AI) database. Rules (in SE-KRL, a software-engineering knowledge representation language) manage a local cache of objects, handling loading, flushing and locking of the cache and coordination between users and server, keeping multiple views consistent, etc. The local data-base also implements higher-level structured data-types, and performs rule-based computations, using a LISP-based forward chaining inference engine.

5 User progammable User Interface components

Driven by the goal of user-programmability, a companion project (ACEkit)[13, 14] has built a variety of higher-level, user-extensible UI 'widgets' in InterViews. Each of these widgets has associated with it a data-structure (a descriptor) that can be interrogated at execution time to determine parameters, modify presentation options, etc. Several basic InterViews objects are also wrapped with a descriptor (via multiple-inheritance). These descriptors are then used with interactive tools and interpreted glue language to permit simple applications to be extended and customized by the end-user using a combination of visual and procedural mechanisms. Widgets built so far are a general purpose table object (like an extended spreadsheet entity), a graph object (DAG and tree), and several editors.

6 Next steps

Each of the above systems has related technology, and similar needs for distribution, display, coordination, information sharing, locking, hypermedia, etc. We will extract useful technology and re-engineer it into bus-compatible components, to produce a common distributed, object-oriented hypertext platform. We also contemplate including ACE widgets and extension technology into a UIMS component for the environment. We will then reconstruct and extend several of these applications, and test our ideas on extensiblity and application compatibility by implementing additional components. We will complete the replatforming of Kiosk and Matisse on Bart. We are repackaging our software components for a set of useful components in the software construction/engineering domain. We will also work with pilot(s) to investigate applicability to other domains.

7 Acknowledgements

This work would not have been possible without the energy, insight, support and comments of members of the HP Labs Software Construction and Evolution projects, particularly Brian Beach, Mike Creech, Dennis Freeze, Jon Gustafson, Joe Mohan and Kevin Wentzel, and our summer students, Mark Gisi and Mark McAuliffe.

8 Biography

Martin L. Griss is Principal Laboratory Scientist for Software Engineering at Hewlett-Packard Laboratories, Palo Alto. He leads research on software reuse, software-bus frameworks, and hypertext-based reuse tools. 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. He serves on HP software engineering councils and is a consultant to HP management on software engineering. 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 has published numerous papers, and was an ACM national lecturer. He received a Ph.D. in Physics from the University of Illinois in 1971.

References

[1] Jim Ambras, Pankaj K. Garg, and Randy Splitter. The Workshop: A Team Programming Environment. In *Proceedings of the 1989 HP European Software Engineering Conference*, May 1989.

[2] Brian Beach. Connecting software components with declarative glue. Technical Report HPL-91-152, Hewlett-Packard Laboratories, Palo Alto, August 1991. Submitted to 14th International Conference on Software Engineering, Melborne, Australia, 1992.

[3] Martin R. Cagan. The HP Softbench environment: An architecture for a new generation of software tools. *Hewlett-Packard Journal*, pages 36–47, June 1990. (See other papers in this issue.).

[4] Patricia Collins. Towards a reusable domain analysis. Technical report, Hewlett-Packard Corporate Engineering, Porter Drive, Palo Alto, September 1991. To be presented at 4th Annual Workshop on Software Reuse, Herndon, VA, Oct 18-22, 1991.

[5] Michael Creech, Dennis Freeze, and Martin L. Griss. Kiosk: A hypertext-based software reuse tool. Technical Report SSL-TM-91-03, Hewlett-Packard Laboratories, Palo Alto, CA, March 1991.

[6] Michael Creech, Dennis Freeze, and Martin L. Griss. Using hypertext in selecting reusable software components. Technical Report SSL-91-59, Hewlett-Packard Laboratories, Palo Alto, CA, May 1991. To appear in Proceedings of Hypertext'91, Dec 1991.

[7] Martin L. Griss. Software reuse at Hewlett-Packard. Technical Report SSL-TM-91-01, Hewlett-Packard Laboratories, January 1991. Invited submission to the *First International Workshop on Software Reusability*, Dortmund, July 3-5, 1991.

[8] Mark A. Linton, John M. Vlissides, and Paul R. Calder. Composing user interfaces with InterViews. *IEEE Computer*, pages 8–22, February 1989.

[9] James Purtilo. The Polylith Software Bus. Technical Report CSD 2469, University of Maryland, College Park, MD 20742, 1990.

[10] James Purtilo, Aaron Larson, and Jeff Clark. A methodology for prototyping-in-the-large. In *Proceedings: 13th International Conference on Software Engineering*, pages 2–12, Los Alamitos, CA, May 1991. IEEE, IEEE Computer Society Press.

[11] Doris Ryan. *RAPID/NM, Reusable Architectures for Transaction Processing and Network Management Applications.* AT&T, 1990.

[12] Gio Wiederhold, Peter Wegner, and Stefano Ceri. Towards megaprogramming. Technical Report STAN-CS-90-1341, Stanford University, Stanford, CA, October 1990.

[13] Craig Zarmer. ACEKit overview. Technical Report STL-89-29, Hewlett-Packard Laboratories, Palo Alto, Ca., nov 1989. A brief overview of the ideas in Szekely and how they have been instantiated in C++ as ACEKit.

[14] Craig L. Zarmer. ACEKit: An Application Construction Toolkit version 1.0. Technical Report SSL-TM-90-08, Hewlett-Packard Laboratories, Palo Alto, CA, December 1990.