Experiments in Hypermedia Support for the "Understanding for Reuse" Problem

Larry Latour

University of Maine

Department of Computer Science

222 Neville Hall

Orono, Maine, 04469

Tel: (207) 581-3523

Email: larry@gandalf.umcs.maine.edu

Fax: (207) 581-1604

Abstract

Generic architecture schemas, documentation schemas that capture all aspects of the development of system families, are complex, multi-view structures that require hypermedia support in order to be complete with respect to a domain specific toolkit [6]. We have experimented with hypermedia representations of integrated Ada component libraries, and have recently been exploring the use of hypermedia tools in university courses such as operating systems, database management systems, and software engineering. We've noted a number of parallels between the university learning environment and a generic architecture development environment. While they are different in many ways, in both cases the issue of separating view-dependent information from underlying "raw material" is an important one. We hint at some of our experiments in this position paper.

Keywords: hypermedia, generic architecture, view independence, software engineering databases

Workshop Goals: to discuss aspects of generic software architectures that will provide insight into the construction of hypermedia webs.

Workshop Groups: domain specific software architectures, domain specific toolkits, generic code architectures, formal methods.

1 Background

It has long been obvious (to most of us, I hope) that a software system is more than just code. Indeed, code is just one small part of an overall documentation schema that IS the system [11]. Extended to system families, code for a specific system should be thought of as a generic code architecture instantiated within the context of a particular environment [*The 3Cs*: 5,8,14]. This generic code architecture in turn is one small part of the documentation schema for a system family, recently being referred to with the buzzphrase *domain specific software architecture* [15]. This documentation schema is augmented with a wide variety of support tools to provide developers with *domain specific toolkits* [6]. In order for a development team to successfully build a system, they need to properly construct, assimilate and apply the knowledge of one of these *generic documentation schemas*. This IS the problem, not the reuse of code components.

At the University of Maine we have been looking at various aspects of this problem. Specifically, we dealt with the construction of generic code architectures in [9], using the 3C model as a guideline for organizing collections of interconnected code components, and we have experimented with hypermedia support for integrated component libraries in [7]. Recently I have been involved in three projects that I consider to be closely related. The first concerns multi-media systems and applications, and the development of a multi-media minor on the University campus. The second concerns the use of "non-linear" classroom/lab support tools in my computer science department courses, and the third concerns the further development of hypermedia based software engineering databases to support the construction and understanding of complex documentation schemas. It is the latter project that concerns us here, but the first two projects have provided me with a good deal of insight into the problems inheritent in a project such as this. I describe some of my experiences in my position.

2 Position

2.1 Hypermedia Support for Operating Systems

After (somewhat) successfully using the results of my hypermedia experiments with integrated Ada software component libraries in my graduate software engineering class, I decided to extend these experiments into the domains of operating systems and database management systems. My operating systems experiments were motivated by the use of Andrew Tanenbaum's Minix Operating System [13], the complete C source code of which is provided to the student, accompanied by a poorly written textbook as the only source of documentation. The DBMS experiments were less extensive but were motivated by a more reuse related project, Don Batory's GENESIS configurable database system composer [2]. In both cases the courses were "architecture driven". That is, both operating systems and DBMS syllabi revolved around the generic properties of their respective subsystem components. A number of interesting lessons learned accompanied the development of hypercard-based stacks for these courses. After describing a few interesting pieces of my "documentation webs", I will discuss a few lessons learned that seem to apply equally well to hypermedia-based reuse environments.

My first example is how user processes interact with the kernel during interrupt handling. It's interesting that long after developing these stacks I recalled similar experiments by Ted Biggerstaff while he was with MCC, developing hypermedia system software support specifically as an aid to reusability [3]. The essence of this particular set of cards is to take the student through the interrupt handling process step-by-step, stopping along the way to visit a number of interesting side issues (or parallelling major issues!). Slicing two cards from the middle of this set doesn't truly demonstrate the usefulness of such a tool in class (or in reuse

understanding), but with a bit of imagination the reader should be able to envision where we came from to get to the card, and where we're going when we finish.

In figure 1, process A has made a fork system call, which was translated to a software interrupt, subsequently invoking the assembler handler s_call. After saving the state of the processor in the process table, s_call then invokes the C procedure sys_call, which adjusts the process table blocked and ready lists and handles message transmission. This is currently the state of figure 1. Subsequently s_call regains control of the processor, and a new process is loaded and started.

Figure 1: Process Interaction with the Kernel

The diagram above serves two purposes. First, it gives the user information about what is happening at this specific moment in the interrupt handling process. But, more interestingly, it provides a c*urrent framework*, or *view*, by which the student can gain more information about any of the visible objects on the screen. It is, in a sense, a graphical query language. The user can, for example, click on the general model button (the up-arrow on the right), transitioning to the general interrupt handling model of figure 2, corresponding to this step in the fork system call.

Figure 2: General Interrupt Interaction with the Kernel

Note that the source of the general interrupt is defined only as "Hardware Interrupt", and the description of this stage in the interrupt handling process should apply equally well to any interrupt handler.

Referring again to figure 1, every visible object has meaning and can be interrogated by the user. This is an important issue in hypermedia systems. For example, click on the processor object and a description of the processor architecture is made visible. Similarly one can explore the structure of the process table, the design of the s_call and sys_call interrupt handling procedures, the library procedures, and information regarding both the process sending a fork system call and the server process within the Minix system proper that receives and processes the system call.

A second, related example, is the protocol in which the fork system call, together with related calls wait, exec, and exit, is used in an application program (most commonly in the implementation of a typical Unix shell). Figure 3 presents one step in the interaction between a parent and child process after the child has been created with a fork system call.

Figure 3: Protocol of System Call Usage

Note that the double click request indicates a degree of animation, essentially focusing on the transformations between each step in the protocol between parent and child. Again, as with the interrupt handler view, all visible items on the screen can be interrogated. An interesting note here is that this view can be arrived at from a number of sources, one indirectly being the view of interrupt handling in figures 1 and 2. The return button is context sensitive in that regard.

2.2 Lessons Learned

A number of interesting issues arose in the construction of the web of information surrounding Minix, which incidently is an ongoing project, neither complete nor correct with respect to my current thinking. I list them below, in no particular order of importance (they all seem to be!)

- **View Independence**: an important issue in any hypermedia web is whether the basic information is independent of a particular view one wants to study it from. My initial linear approach to developing these stacks was seriously flawed in this regard. Specifically, informaton about process usage (operating system services and user applications, specification (the system interface) and implementation (the kernel), needs to be defined to a large extent independent of the "direction" one views that information from. This relates somewhat to the redundancy probem prevalent in poorly designed database schemas, but there is also a modelling issue so important to understanding. The old saying that you don't really know something until you've seen it from at least two perspectives is an important issue here.

- **Completeness**: if a hypermedia web is incomplete with respect to the system it is attempting to model, students quickly tire of it. For example, a primary issue that came up was whether or not the web could be taken down to the code level, up to the device driver, file system, and memory manager levels, or to generalizations of these subsystems. The answer in my case is, to an extent, but not completely. For example, I am currently considering a hypercard front-end on our PC network, accessing and encapsulating the Minix code database on our SUN network. Related here are the issues of view independence and full integration.

- **Full Integration**: an issue that came when discussing what a system family web would look like was the extent to which an instantiated system member was integrated into the family web. That is, is the mode of interaction to access the web and mine for artifacts, or to integrate the new system into the web. This raises subtle issues of version control and configuration management that are central to any large software management effort.

- **Simulation vs. the Real Thing**: multi-media systems such as Hypercard and Macro-Media Director, as well as CASE tools such as State-Mate provide a rich set of visual simulation tools. In contrast to this, we have discussed providing the hypermedia environment with hooks to explore the dynamics of the system as it's being constructed. This is an important issue when considering the following bullet:

- **Person-Hours of Development Time**: hypermedia webs take large amounts of time to develop, which I suppose is an issue that has hindered the development of good "static" documentation throughout the software ages. The issue here, and the domain speficic toolkit developers should have a good handle on this, is how this development work can be "amortized" across the system family.

- **View Formalism**: Considering each screen as a view, is each based on a fundamentally sound model? Furthermore, is the model formal, or does it have a formal underpinning? In our hypermedia experiments with integrated component libraries, we defined Ada specification views that were based on Larch formal interface specifications [Liskov86], which in turn were based on a hierachy of formal algebraic models [Wing90]. An issue that we considered, but didn't pursue, was to take advantage of Ada and Larch specification language formalism to define much more granular connections between artifacts, a task that would require automatic link generation support.

- **Web Growth**: An issue that I've considered and rejected for the present with my student hypermedia experiments is the ability of the student to add novice knowledge to the web in a way that grows and is refined as progress through the material is made. One such class project currently involves the implementation of a kernel along with accompanying memory management and file management

support for a small virtual machine. There clearly are expert/novice issues involved when constructing a complete hypermedia web within a multi-person software development effort.

3 Comparison

There are a number of hypermedia-related efforts currently ongoing in the WISR community, including the work at Hewlett-Packard on Domain Specific Toolkits [6] and the KAPTUR work CTA [1]. In addition a good deal of hypermedia-reuse research was done by the MCC group [3]. Related work in semantic network construction and navigation was done at Unisys [12] and AT&T [4].

In addition to the hypermedia-specific work mentioned above, hypermedia systems will benefit greatly from a more complete understanding of the structure, formalism, and dynamics of domain specific software architectures and their associated toolkits.

There are, of course, many differences between a typical software development environment and a University learning environment. The definition of "novice" and "expert" is much more refined in a software development environment. But there are similarities as well. The difference actually might very well be one of view rather than view independent knowledge.

References

[1] Bailin, S.C., and Henderson, S., "Towards a Case-Based Software Engineering Environment", *Fifth Annual Workshop on Software Reuse*, Palo Alto, CA., 1992.

[2] Batory, D., "On the Difference Between Very Large Scale Reuse and Large Scale Reuse", *Fourth Annual Workshop on Software Reuse,* Reston, VA., 1991.

[3] Biggerstaff, T., "Hypermedia as a Tool to Aid Large Scale Reuse", *Workshop on Software Reuse, Rocky Mountain Institute of Software Engineering*, Boulder, CO., October, 1987.

[4] Devanbu, P., "Re-use of Software Knowledge: A Progress Report", *Third Annual Workshop: Methods and Tools for Reuse*, June, 1990.

[5] Edwards, S., "The 3C Model of Reusable Software Components", *Third Annual Workshop: Methods and Tools for Reuse*, Syracuse, 1990.

[6] Griss, M., "A Multi-Disciplinary Software Reuse Research Program", *Fifth Annual Workshop on Software Reuse*, Palo Alto, CA., 1992.

[7] Latour, L., and Johnson, E.,"SEER: A graphical retrieval system for reusable Ada software modules", *Third International IEEE Conference on Ada Applications and Environments*, Manchester, NH, May, 1988.

[8] Latour, L., Wheeler, T., and Frakes, W., "Descriptive and Predictive Aspects of the 3C Model: SETA1 Working Group Summary", *Third Annual Workshop: Methods and Tools for Reuse*, Syracuse, 1990.

[9] Latour, L.,"Layered Generic Architectures for Reuse Engineering", *First International Workshop on Software Reusability*, Dortmund, Germany, 1991.

[10] Liskov, B., and Guttag, J., *Abstraction and Specification in Program Development*, McGraw Hill, 1986.

[11] Parnas, D.L. and Clements, P.C., "A Rational Design Process: How and Why to Fake It", *IEEE Transactions on Software Engineering*, Vol. 12, No. 2, February, 1986.

[12] Solderitsch, J., "An Organon: Intelligent Reuse of Software Assets and Domain Knowledge", *Fourth Annual Workshop on Software Reuse*, Reston, VA., November, 1991.

[13] Tanenbaum, A.S., *Operating Systems: Design and Implementation*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.

[14] Tracz, W.J., and Edwards, S., "Implementation Working Group Report", *Reuse in Practice Workshop*, Pittsburgh, PA., 1989.

[15] Tracz, W.J., "A CASE for Domain-Specific Software Architectures", *Fifth Annual Workshop on Software Reuse*, Palo Alto, CA, November, 1992.

[16] Wing, J.M., "A Specifier's Introduction to Formal Methods", *IEEE Computer*, September, 1990.

4 Biography

Larry Latour is an Associate Professor of Computer Science at the Univeristy of Maine, having received his PhD degree in Computer Science from Stevens Institute of Technology in 1985. His work was in the development of a model theoretic approach to concurrency control, and he has since looked at how algebraic specifications of abstract objects can enhance concurrency control in object databases. At the same time he developed his software engineering interests at the Ft. MOnmouth Center for Software Engineering. He was introduced to reuse in 1986 at the Syracuse University Annual Software Engineering Workshop in Minnowbrook, NY, where he and a small group of Tools and Environments working group members began what is currently the National WISR workshop series on software reuse. In conjunction with this workshop he manages the WISR repository at Maine. As well as hypermedia systems his interests include formal methods in reuse, generic code architectures, quality university teaching, and computing at the K-12 level.