tools in (4). We would encourage professors to have students produce these for the community under their guidance.

SIGLINK is the most logical professional organization to define and construct a fully-functional hypermedia-supported digital library. Let's lead the way by implementing the infrastructure to publicly demonstrate our community's concepts and then populate it with our community's research and annotations. Simultaneously, let's work further to craft this digital library into the core resource of a full Hypermedia Networked Improvement Community.

Please contact me if you are interested in working on these ideas, or if you have any suggestions. (bieber@njit.edu)

Desiderata for a Java-based Hyperlink API: The Hypertext '98 Java Users' Birds of a Feather Session

Peter J. Wasilko, Esq., J.D., LL.M.
Director, The Continuity Project
futurist@cloud9.net

"Are we here yet?"—Background to the BOF

Hypertext '98 will no doubt be remembered as a seminal event in which our community paused to reflect on its past and to reassess its future direction. We have a long tradition that dates back to the visionary work of Vannevar Bush, Ted Nelson, and Douglas Engelbart. We have produced numerous research systems over the years demonstrating a depth and breadth of functionality that goes far beyond the static navigational paradigm of the Web.

But HT '98 was a time to reconcile where we are with where we might have been. Our conference series didn't accept the first paper on the World Wide Web, though it should have. Our technologies, in their full richness, have not been transferred beyond the lab, though they should have been. Indeed, as John Leggett observed in his opening keynote, our community is not Hyperliterate, since we do not yet use our own tools in the creation and access of our literature.

Douglas Engelbart challenged us to think of ourselves as a Networked Improvement Community and to remember that the introduction of novel high impact technology will inevitably necessitate some measure of user training. However, he cautioned us to avoid the trap of limiting our designs to the lowest common denominator of today's skill levels since people can and will learn in the future just as they have in the past.

In my view, our community came away from HT '98 with a renewed sense of direction, a general consensus to make the interoperability and transferability of our technologies a key priority, and a determination to lead by example. In the months ahead you should see the emergence of a number of exciting new initiatives with their roots in HT '98.

Java and Technology Transfer

I proposed and led this year's Java BOF in anticipation of such developments which are making the Java Platform a particularly appealing environment for future hypertext research. In the days of old, when computing machinery was slow, large, expensive, and the province of large organizations, the availability of CPU Cycles dictated our choice of development environments. If an organization later wanted to deploy the fruits of our research, we could reasonably assume that it would already have access to or be willing to commit the funds needed to replicate our development environment.

So a lot of early work found its home on high end low volume systems built around custom hardware, like the LISP Machine. Over time, as these systems were phased out or failed to penetrate the commercial market technologies dependent upon them stagnated and atrophied. This occurred despite the concomitant explosion in desktop computing power, because the vendors of the underlying systems did not perceive a large enough market to justify porting their full development environments to the desktop. Moreover, they did not have the expertise to produce fast, tight runtimes that could offer end users of systems constructed to run on them a consistent native user interface experience.

More recently, some vendors have followed the rise of processing power to the desktop with limited cross-platform solutions, but they tend to focus on preserving legacy code and operate on the assumption that their users' primary object is to create commercial code targeted for each platform's native environment.

Sun's Java platform is predicated on a different set of choices, which are perhaps more closely attuned to the needs of the research community. The cost of entry into Java development is dramatically lower than competing solutions (free if one downloads just the Java Development Kit from Sun). By focusing on the creation of a world class Virtual Machine and a series of high level cross-platform Application Programmer Interfaces rather than a full Integrated Development Environment, Sun has taken on a more manageable task. The virtual machine architecture encourages third parties to boost the performance of the system through bytecode optimizations and advanced interpretation techniques while providing an appealing target for the designers of other programming languages.

Java offers us memory management, multiple threads of execution, exception handling, a persistence mechanism, the ability to create reusable cross-platform graphic user interface components, and enough high level networking support to put the design of distributed collaborative hypertext systems within the grasp of an advanced undergraduate. Of course the language is not perfect and many might argue that Scheme offers a more elegant and concise conceptual framework on which to base computer science instruction. But as a research development environment its benefits outweigh its disadvantages, indeed even those preferring Prolog or Scheme can find solutions under development to interpret these languages in Java or compile them directly into Java classes which could then be invoked from Java proper.

At present, Java is too slow for some applications on common hardware, client-side Java still isn't completely stable across virtual machines, and the language will never match the speed of compiled C or hand optimized assembler for **all** classes of computation. However, these issues will soon fade as we run our Java under Dynamic Compilation or with next generation Just-In-Time Compilers with more compliant virtual machines on ever-faster hardware. Indeed it would not be unreasonable to assume that Java's runtime overhead will represent a sufficiently small percentage of CPU load in a few years that a major system being written today could safely introduce another level or two of interpretation without becoming unwieldy.

However, from a technology transfer perspective, the cross platform promise, if realized, offers us the greatest benefit. For it will guarantee that what we write on high-end systems in the lab today will be able to run with ease on a student's or customer's personal computer in a few years. Moreover, if network computing takes hold on campus, even a stock PC in today's average computer cluster could serve as a thin client to support interaction with our Java-based server side solutions.

Are We Still Doing Hypertext Wrong?

After spending some time on these issues, I raised the question of what sort of direct support for hypertext we might like to see added to the standard Java libraries. I took Norman Meyrowitz's short but highly salient contribution to Edward Barrett's classic anthology *The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information* (MIT Press, 1989) as a point of departure. In "The Missing Link: Why We're All Doing Hypertext Wrong" he found that the failure of hypertext to reach its full potential stems from our development of "**insular, monolithic packages** that demand the user disown his or her present computing environment to use the functions of hypertext and hypermedia." (p. 112) By contrast, Meyrowitz argued that we should try to emulate the success of the cut, copy, and paste paradigm noting that:

"This paradigm caught on for four reasons: 1) powerful things could be done with this paradigm; 2) the paradigm was extremely easy to motivate and teach to end-users; 3) the toolbox vendors touted the copy and paste protocol as an important integrating factor that all software developers should include in their applications; and 4) most importantly, the toolbox supporters provided the framework for copy and paste deep in the system software and provided developers the protocols that enabled them to incorporate the paradigm into their software with relative ease." (p. 112-113)

"...Linking functionality must be incorporated, as a fundamental advance in application integration, into the heart of the standard computing toolboxes... and application developers must be provided with the tools that enable applications to 'link up' in a standard manner."

The Intermedia project took this approach with limited success as it lacked the visibility and market penetration to induce developers to embrace its API. Unfortunately, none of the platform developers have taken heed of Meyrowitz's challenge and to this day we do not have system wide hypertext facilities.

Apple's Initiatives

Given the adroitness with which Apple Computer commercialized Xerox's *Cut, Copy, and Paste* research, I took some time at our session to carry the case study forward. Apple's *Drag and Drop* and *Publish and Subscribe* models, whose APIs parallel that of the standard clipboard, have enjoyed comparable though somewhat more limited success with the basic text and graphic clipping types.

But programmers and users have come to recognize the limitations of the underlying core data types. Specifically, most new applications can now import and export raw text, styled text, and "PICT" graphics. However, many older applications can't handle text style information and the "PICT" graphic format does not support many advanced visual effects. In general users seem to tolerate this mismatch under *Cut and Paste*, but it clearly breaks down under *Publish and Subscribe* because one can edit a live graphic with all effects preserved in its native application which is then printed as a screen resolution bitmap by the subscribing application. Thus the seamless integration of the data interchange is notably broken.

This led to the ill-fated OpenDoc effort, which attempted to provide an object oriented framework that could factor out editing from printing and viewing and to eliminate the need for data to be reduced to a system level common denominator before being incorporated in other documents. OpenDoc failed for two reasons: 1) executive level mismanagement sent the company into a steep downturn costing Apple market share and developer trust at the time of its release and 2) given its 50 unfamiliar API calls and dubious cross platform support, it could not be adopted by developers with *relative ease*. (See "Building an OpenDoc

Part Handler" by Kurt Piersol in *Develop: The Apple Technical Journal*, Issue 19, September 1994, pp. 6-16)

Moving Forward with Java

These experiences suggest that, to be widely useful, a Hyperlink API ought not widely diverge from the dominant application organization paradigm of its host system. It should have a small well-factored API that can be implemented with minimal changes to existing code. And perhaps, it should be associated with a composite document model so arbitrary source elements can be *transcluded* (i.e. linked & embedded) without loss of formatting. Moreover, any programmatically generated or scripted elements ought to have their implementations fully encapsulated so their destinations need not actively trigger or participate in the execution of their evaluation mechanism.

These final points imply support for first class access and ability to manipulate nested components, raising some interesting knowledge representation issues[†] which could be safely deferred to a future point in time, provided appropriate interfaces are put in place from the outset.

We can further envision two classes of users of such a link API: 1) Developers of such functional domain specific modules as Day Planners, who would like to support the importation of arbitrary link pointers (e.g. URL's) to embed in their container classes (e.g. Timeslots) while exporting references to, and self-contained representations of, their own application specific substantive classes (e.g. Appointment objects); and 2) Hypertext researchers and developers looking to provide advanced link management, authoring, and analysis. The former group would be best served by a minimalist API building on existing data interchange services with user controls provided through a pre-built reusable GUI Widget. The later group would want to retain full control over all elements of the user interface, perhaps have the ability to bind custom data structures to system wide link objects, and be able to modify the functional semantics of link creation, selection, traversal, etc. without sacrificing interoperability with other packages. Fortunately, these goal sets are complimentary.

Java is particularly well suited to the development of such a framework, as its high level constructs already draw heavily on the work of Design Patterns community (see *Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides with a Forward by Grady Booch. Addison Wesley, 1995) which has identified a number of mechanisms that can be readily integrated to endow a system with just these kinds of functionality. Thus Java is well positioned to help solve our software engineering problems.

Conclusions—The Open Hypermedia Connection

But we would still need a semantic foundation on which to structure a Java Hyperlink API to insure interoperability with native services in other environments, for while Java provides an elegant solution to creating cross-platform hypermedia clients we need to embrace legacy systems as well.

Fortunately, our session had a number of participants who have been actively working with both Java and Open Hypermedia Systems. This Open Hypermedia Systems Working Group has drawn on the perhaps somewhat dated, but still descriptive, *Dexter Hypertext Reference Model* to implement cross-platform link services. They were able to address these larger interoper-ability issues and point to their ongoing standardization effort that we could leverage

[†] I noted that Professor Ken Haase at the MIT Media Lab had done some interesting work on large-scale distributed knowledge bases with Java support that could provide some useful insights (see "FramerD: Representing knowledge in the large" by K. Haase in *IBM Systems Journal*, Volume 35, Numbers 3&4, 1996, pp. 381-397). Unfortunately he has been out of touch, and presumably focusing on other Media Lab business since his last note to me was back in February so I am unable to report on FramerD's current status.

in our explorations of the design space for a future Java-based Hyperlink API.

Ultimately, we gathered some initial sense of how Java might evolve to better support our needs while coming to recognize that future collaborations between the Open Hypermedia Systems Working Group and the Java developer community would offer us the greatest opportunities to advance our mutual objectives.

In closing, I would like to thank the HT '98 Program Committee for providing us with this forum to meet and to thank each and every one of our session's participants for sharing their thoughts. ❖

Online Resources

JavaWorld—IDG's magazine for the Java community: `http://www.javaworld.com`

Java Developer Connection Home Page: `http://developer.javasoft.com`

Languages for the Java VM: `http://grunge.cs.tu-berlin.de/~tolk/vmlanguages.html`

Open Hypermedia Systems Working Group: `http://www.csdl.tamu.edu/ohs/`, `http:// www.ohswg.org` (This later site was down at the time of this writing)

Trip Report: ACM Hypertext '98

Deena Larsen
textra@chisp.net, deenalarsen@acm.org

Hyperliteracy

The keynotes began and ended in a dance around the concept of hyperliteracy. I have always thought in hypertext—but I suspect that this is merely a private quirk in my thought patterns. Thinking linearly is like holding onto a bedstead while someone pulls the corset tight—it crushes my ideas until they faint.

I need to think both of the idea and its connections at the same time or the thought is lost. Thus this trip report is as nonlinear as you can get in a newsletter (My real conception of Hypertext 98 is all contained in the graphic, which shows the interstices of the three "camps").

A Common Language

John Leggett's image from the opening keynote, of scattered tribes on a desolate plain, held sway throughout the conference. To create a civilized hyperliterate world, we first need a common language. John spoke of runners between the camps. Mark Bernstein began to address the challenge of developing vocabulary that describes and names the structures and paths we are finding in actual hypertexts. Francisco Ricardo stalked the edges of meaning as he spoke about the meaning that links have in and of themselves. These new ways of forming and thinking about communication will lay the groundwork for developing hyperliterate systems.

Reef Accretions

Accretion. **n.** An increase by external addition or accumulation (as by adhesion of external parts or particles).

The reef metaphor for the panels opened up the way for fantastic accretions of knowledge, insights, and collaboration. If thoughts could grow shells like coral, then the panel reef would surely be a multifaceted wonder of the seven seas.