Replacing Copies With Connections: Managing Software across the Virtual Organization

Tobias Murer
TIK (Computer Engineering and Networks Laboratory)
ETH Zürich, 8092 Zürich, Switzerland
murer@acm.org

Michael L. Van De Vanter
Sun Microsystems Laboratories
901 San Antonio Road, UMTV29-112
Palo Alto, CA 94303 USA
Michael.VanDeVanter@sun.com

Abstract

Emerging technologies such as the Internet, the World Wide Web, Java™ technology, and software components, are changing the software business. Activities that have in the past been constrained by the need for intense information management increasingly involve cooperating organizations. Information management tools and techniques do not scale well in the face of this organizational complexity. An informal approach to information sharing, based largely on manual copying of information, cannot meet the demands of the task as size and complexity increase. Formal approaches to sharing information are based on groupware tools, but cooperating organizations do not always enjoy the trust or commonality of sophisticated infrastructure, methods, and skills that this approach requires. Bridging the gap requires a simple, loosely coupled, highly flexible strategy for information sharing. Extensive information relevant to different parts of the software life cycle should be interconnected in a simple, easily described way; such connections should permit selective information sharing by a variety of tools and in a variety of collaboration modes that vary in the amount of organizational coupling they require.

1. Introduction

The software business is changing, much as many other businesses have changed. The product life cycle increasingly takes place within so-called *virtual organizations* that require close cooperation across a variety of organizational boundaries [DM92]. In the software business this trend is supported and accelerated by emerging technologies: the World Wide Web, software components, and Java™ technology [GJS96].

Essential to the working of virtual organizations is the management and sharing of information, a task for which the increasingly connected world of the Internet is well suited. How the new technologies are actually used by organizations, however, is a different matter. Current approaches fall into two general categories: informal techniques based on *manual copying* of information across organizational boundaries, and formal arrangements based on shared tools that can be broadly called *groupware*. Neither approach is well adapted to the world of virtual software organizations, and neither address the whole software life cycle.

Manual copying, for example manually installing software from the Internet, from CD-ROMs, or via other media, fails as systems scale in complexity. It is unreliable, and it disconnects software from information that is important throughout the software life cycle. In contrast, highly evolved groupware tools have been developed to manage complex information reliably. Software configuration management systems permit concurrent non-interfering work, support specification of complex aggregations of parts, enforce process rules, and keep project history. Software managed by such systems resides in a complex web of information, without which it would be of little use to the organization. Although often difficult to use, no large software project can succeed without such tools.

Groupware tools are clearly important for virtual software organizations; efforts to date have focused on exploiting the new infrastructure to support distributed groupware tools [KD98]. This follows a common pattern where a new, emerging technology is initially used to extend existing mechanisms in a process that remains otherwise unchanged. But the new technologies and organizational patterns change the way people work. Groupware designed for tightly coupled organizations fails to address organizational and cultural issues such as autonomy, divergent tool preferences, and variations in methods and skills.

This suggests that a complementary strategy might fills the gap between the informality of copying and the tight coupling of groupware tools. Such a strategy, called the *application web* is being investigated jointly by the Forest project at Sun Microsystems Laboratories and the Virtual Software House project at ETH Zürich. An important characteristic of such a strategy is the notion, to which people are increasingly accustomed, that information (including software) should be interconnected. Useful and appropriate information *about* software, captured during its construction by development tools, should be continuously available throughout its life cycle. Finally, tools must be able to communicate across boundaries in a way that strikes a useful balance between cooperation and autonomy.

This paper discusses such a strategy, beginning with an assessment of the current situation and the forces at work. This is followed by observation of how these same forces conspire to limit the growth of virtual organizations and the quality of the products they can produce. The concept of the application web is then discussed general terms, followed suggestive scenarios based on current work with the JP and VSH prototypes.

2. Forces Changing the Software Business

The motivation for the proposed strategy arises from forces, both technological and organizational, that are reshaping the software business.

Technological Forces: the Connected World of Dynamic
Most compelling is the connectivity that accompanies widespread adoption of Internet technology, in particular the technical infrastructure provided by the World Wide Web: global naming system, simple protocols, and means to exchange and present information. The web permits creation of new technologies, gives leverage to others, and has emerged as the universal infrastructure for the kind of global cooperation required by virtual organizations.

A phenomenon of the Internet has been the widespread adoption of Java technology, a software platform whose dynamic behavior permits new kinds of flexibility and cooperation. The language is based on classes that can be composed into applications based on interfaces (APIs); interfaces, combined with platform neutrality and HTML-based documentation extracted from source code, have encouraged a great deal of code sharing. The common conduit for such sharing is downloading collections of classes over the Internet, for example Java libraries, plug-ins, tools, and applets.

The runtime behavior of a Java application is dynamic and flexible. Java classes are loaded lazily, as needed at runtime from a variety of sources: file systems, local and global networks. Each user of a Java application effectively defines a system composition by specifying a class search path: an ordered list of file system paths and URLs that are to be searched for required named class. This allows users to compose systems from classes that may not have been compiled and built together, as long as interface requirements and binary compatibility rules among are fulfilled. Users can extend and modify the behavior of applications by managing the runtime class path.

Java technology is predated by the component-based approach to system construction [Szy98], but the two now enjoy considerable synergy. Emerging component models such as plug-ins, Java Beans, and Enterprise Java Beans add more features to be used for system composition. The Java reflection facilities permit components to negotiate dynamically whether their interfaces match and whether required services are supported.

Organizational Forces: Rise of the Virtual Organization
A virtual organization is a dynamic network of organizations that cooperate for mutual benefit [DM92], a business model that presents new challenges. Geographic dispersion may increase, but even more important is the added autonomy represented by organizational boundaries. In this respect, one also sees virtual organizations within larger organizations whose parts often exercise similar autonomy. Significantly for the software business, organizational autonomy is typically reflected by heightened differences of infrastructure, methods, tool preferences, and skills when compared with single software development organizations.

Members of virtual software organizations cooperate more or less tightly at all phases of the software product life cycle: construction, deployment, runtime management, as well as services such as consulting. An application might consists of software developed by several organizations. It might be deployed and managed by two other organizations and supported by consultants from a third. For the purpose of discussion we define a Java *configuration* to be a collection of Java software that can move across organizational boundaries to support collaboration within a virtual software organization. A configuration might be a Java class library, a plug-in for some framework, a Java Bean, or a stand-alone application: any collection of software intended to be used, via a known interface, as a unit.[1] Although such units of code may be used in dynamic settings, for example plug-ins, it is understood that the units have static properties of interest (for example, who wrote them, and what level of quality assurance is given). The central issue at organizational boundaries is coordination and sharing of information *about* these properties, information without which configurations would be useless.

3. Limits Faced by the New Model

The very factors that engender synergy among emerging forces described in the previous section also create limits on the scale to which the new model can grow. The dimension of scale described here is not part of familiar problems: the size of a system's parts or the complexity of its assembly. This scale is characteristic of the virtual organization, where increased organizational complexity adds new problems to the old ones.

This section describes how current technologies fail in the presence of this challenge. Three inherent weaknesses in Java technology (described below) demonstrate how the whole software life cycle is affected: software construction (interface specification), deployment (configuration management), and management (runtime support). The common theme underlying these issues is that information *about* software must cross organizational boundaries, but that current approaches to sharing information actually *disconnect* software from essential information.

Java Interface Specification
Using classes created by others requires information about how they work. At one extreme (and the

most heavily used) are simple syntactic descriptions of the sort represented by Java interfaces and their embedded comments. But syntax is too weak (and comments seldom adequate) to assure interoperation. If Java interface syntax and comments were adequate then the avalanche of books describing Java platform classes would not be needed. At the opposite extreme, complete semantic descriptions of class behavior are widely understood to be infeasible in general, let alone scalable.

Intermediate mechanisms to address this problem have been proposed [WCOP97], but none have become widely used. A more organizational approach, suitable for the context discussed here, suggests that the key is to complement syntactic information with other information normally available only in the originating organization [MSW96]. Such "originator level" information specifies exactly and completely the context in which particular versions of a configuration have been constructed and are intended to be used, information essential throughout the life cycle. For all but the most widely used software, however, such information has been traditionally communicated informally, through informal documents, personal communication, and folklore. Unfortunately, this is precisely the kind of information from which software becomes disconnected when it crosses organizational boundaries.

Defining Java Configurations

Connecting pieces of software together is only part of the job of constructing software systems; they must be built, run, and tested together. For situations where the quality of delivered software is important, deployment takes place in aggregations (what we call configurations) accompanied by some kind of assurance from the originating organization. Unfortunately, Java technology offers no concrete notion of configuration and no tools for doing this reliably. Configurations are typically bundled simply as collections of Java classes, typically in file system hierarchies, and deployed in containers (JAR files) that mimic file systems. Installation involves copying such files into local storage. When a Java application requires more than one such configuration, the user is responsible for ensuring correct placement in local storage, and for constructing a correct class path.

The Java deployment model effectively delegates system composition to end users, who have no "originator level" information about configurations beyond what is present in a weak (and mutable) file system. There is no assurance, essential in many contexts, that configurations being used actually correspond to what is desired, to what was purchased, to what is assumed to have been built and tested. Furthermore it obscures responsibility when software does not perform as expected.

A simple hedge is to deploy Java applications as self-contained bundles, but this does not address the emerging component models such as beans and plug-ins that require such end-user configuration. Furthermore, this approach does not scale as applications and the Java platform grow in size. Some kind of sharing is essential.

Scalable solutions rely on configuration management tools, which variety of services to help with the construction of large software systems. However these tools are mainly designed for inter-organization coordination; such tools typically export the same kinds of class collections as in the manual scenario. Once again, the delivered configuration is disconnected from essential information.

Managing Applications

The information deficit worsens when a Java application is launched. Other than specification of the name for a main class, Java technology offers no runtime notion of what it means to be a application, nor does it offer any way to start them, stop them, and investigate their runtime state. The runtime configuration of an application is created dynamically by the automatic, on-demand loading of classes via searching through multiple sources. This makes it nearly impossible to answer such questions such as which versions of the classes are loaded, which configurations they were loaded from, where are documents that apply to them, does the application need any more classes, and if so where can they be found, and so on.

The runtime state of an application has essentially been disconnected from its origin, making it very difficult to analyze behavior and diagnose problems without more contextual information than is typically available when organizational boundaries have been crossed.

Summary: Losing Context at Organizational Boundaries
Virtual organizations collaborate because it is to their advantage. Unfortunately organizational boundaries tend to impede the flow of information important to that collaboration, as shown by three examples in the context of Java software development. As with all problems of complexity, this can be managed in the small, but it effectively limits how large virtual organizations can become and how reliable their products can be.

Tools such as configuration management systems are seldom practical in virtual organizations for variety of reasons. First, the design of such tools usually presume trusted participants, which is not the case in a context where cooperating organizations guard intellectual property and share information only as needed. Second, autonomous organizations differ in aspects such as culture, processes, methods, competences, and skills; the same tools will not be effective for all of them. Finally, such tools have buy-in overhead that is not always appropriate to the dynamic nature of virtual organizations.

Information flow across organizational boundaries continues to be dominated by informal, manual copying, accompanied by two serious drawbacks already mentioned. First, it is an unreliable way to cache information produced elsewhere. Second, it typically lacks the rich information available in the original context where software is constructed. Consequently software interoperation is be based on published interfaces alone; there is precious little information about configurations beyond simple aggregation; and issues arising at runtime cannot be resolved by tracing backward reliable from running software to additional information.

4. The Application Web: Connecting Virtual Software Organizations

A strategy for addressing these issues is being explored jointly by the Forest Project at Sun Microsystems Laboratories and the Virtual Software House (VSH) project at ETH Zürich. This section discusses this strategy in general terms.

The JP, GIPSY, and VSH Projects
The Forest project's JP software development environment addresses problems of scale in Java software development with fundamentally reliable and scalable Java software construction technologies [JV97]. These include a repository based on orthogonally persistent Java objects [Atk96], a configuration management system for all human-created information, a reliable builder based on functional programming, and Java-implemented tools that run in the repository. JP keeps a complete history of human work performed (each build is based on a closed-world, immutable prescription that can be rebuilt reliably at any time), automates management of *derived* information computed by builds (which the developer never sees unless needed), and connects tools such as editors to its repository [Van98]. JP is intended to support a federated software development model among multiple JP sites.

The VSH project complements the JP environment with focus on both the traditional scale of configuration complexity, as well as the emerging complexity of organizations. It is investigating new business opportunities made available by the emerging technology and the trend toward virtual organizations, based on earlier work project on software component development and cross-organization deployment investigated in the GIPSY project [Mur97,MS97]. A goal of the VSH project is a service architecture for electronic component-based software construction, deployment, marketing and consulting services in ways that will scale into the emerging development models.

From this collaboration has emerged a rethinking of the JP development model and its relevance to the entire software life cycle, as well as the addition of distributed services that would support the collaborative models proposed by the VSH project.

Connectivity Throughout the Life Cycle
Central to the application web strategy is rich interconnectivity among information that spans the life cycle of software products: construction, deployment, and management. This is based on the now familiar experience in other domains on the World Wide Web, a fundamentally simple model, but with the addition of restrictions and new services applicable to the realm of software. The application web might in some instances approximate "extranet" technology, where organizations selectively provide access to internal information, but it must be more dynamic and flexible than most current single-focus extranets.

This proposed notion of connectivity has two aspects. First, it is desirable to keep a configuration connected to all its information captured on the web during its whole life cycle. This makes it possible to query the application web for complete, precise, relevant information at all times. Second, it connects organizations with a set of useful services that strike the required balance between autonomy and cooperation on the part of participating organizations.

For this to succeed, an application web must be almost as simple to operate as the World Wide Web for basic operations such as deployment and management. At the same time it must be as powerful as JP, for example, in order to support complex operations such as shared development, configuration management, and building.

Autonomy
Organizations, even when collaborating within a virtual organization, try to retain as much autonomy as possible. The application web must find the right balance between granting autonomy and providing sufficient support for information sharing and cooperations. The application web approach addresses this challenge with two features. First, the shared model of data needed to exchange information is kept as simple as possible, following the lead of the World Wide Web. Second, information sharing can take place on more than one architectural layer, each appropriate to the task and particular organizational boundary.

Service Layers
The simplicity of the application web, through which collaborating organizations connect, is made possible by a layered approach that permit different degrees of coupling between organizations, depending on the phase of the life cycle involved and the kind of collaboration needed.

Closely collaborating development organizations might connect by a more complex shared application builder service such as JP federated building. In this model, build scripts (analogous to makefiles) in one JP repository can build against and aggregate software that may originate in another JP repository without requiring manual copying.

More loosely collaborating organizations might provide less access, for example the ability to run configuration directly out of its original repository, but with access to the rich contextual information available in the repository. Even more limited, and very simple, browsing services could be available, making available only a carefully controlled subset of repository information to suit particular relationships, for example a consultant simply browsing documentation related to a running application somewhere.

5. Application Web Services

The application web strategy is based on a layered approach in which a number of basic services are available to support a variety of collaboration models.

Naming and Versioning
A fundamental service of the web is a simple, globally scalable *naming system* for versioned configurations distributed across organizations The design of such a naming system, an extension of the JP naming system, is discussed in more detail elsewhere [VM99].

The JP approach is based on versioned packages of software, sources and related documentation, that are given globally unique names in an extension of Java name space. Even though there is no concrete notion of "package" in the Java language, one is provided by the tools so that software can be reliably bundled into globally agreed-upon names. It is intended that a versioned package name, e.g. com.sun.labs.forest.jp.util.7, have the same meaning everywhere. Not all JP versioned packages contain sources and documentation directly. Some packages serve only to aggregate other package versions, a mechanism permitting the recursive descriptions of systems of of any size. These kinds of packages play the role of "configurations" in the JP system.

Persistence, Immutability, and Caching
A requirement of any connected, sharing-based approach is strong guarantees about the lifetimes of important information. Participating organizations that export information must ensure that information be be durable, immutable, and accessible for as long as it is needed by its collaborators. With suitable restrictions of this sort, published information can be managed by reliable, automatic cache management for ensuring timely access by all concerned. Reliable automatic caching therefore replaces unreliable manual caching, as long as basic guarantees can be made.

Repositories that provide persistent bindings between versioned package names and their contents support, among other services, WWW browsing and querying throughout the application web's name space.

Configurations
As defined earlier, a "configuration" in the application web is simply the unit of software sharing, possibly containing a collection of sources and related documentation. But, as in the JP model, it might itself be composed recursively of other configurations, not all of which may have originated in the same organization.

Globally unique names, combined with persistent storage, allow configurations to be efficiently implemented without copying, carrying instead only references to included sub-configurations. In building, as for other services, copying is then done by automatic caching.

Reliable Building
Versioned packages contain sources from which derived information, for example Java class files, are computed. Just as the name of a package always refers to the same thing everywhere, so should

the derived form; this demands that building be location independent, a guarantee provided by the JP *build system*.[2] At the same time, the result of each build should be arranged to contain enough information for tools to trace back to the originating context in cases where more information is required.

This mechanism permits a wide variety of collaboration models at organizational boundaries where build systems can communicate. For example, a client organization might contract to use software from an originating organization with full source access, in which case build scripts in the client organization might simply import software as if it were local, relying on the build system to cache and build the sources in the local context. In other cases, the client organization might purchase only the right to request built-to-order packages, in which case part of each system build might take place in the repository of the originating organization, based on a call between the two builders that specifies the parameters of the build (e.g. compiler flags).

An Abstraction for Applications
Configurations may be libraries, but they may also be intended to code with more specific structure, for example plug-ins, Java Beans, and runnable applications. The application web supports abstractions for such cases, permitting interactions among many tools such as application loaders, inspectors, and debuggers.

Application Deployment and Management
Applications can be located on web, using browsing services, and launched by simply pressing a button without copying or installing anything. A *deployment service* precisely collects the relevant executable application parts directly from their source at the originating organization. An *application management service* supervises running applications, and manages links to the precise origins of all involved application parts. By following these links, a client can reliably locate additional information about the application directly from the source.

Application Inspection
An *inspection service* allows to remotely investigate the class and object structure of running applications. For every loaded class, a link leads back to the corresponding source at the providing organizations. An application provides hooks to navigate through the object structure of the application. Again, for every object, a link leads back to the corresponding source published on the web.

Managing Access to Information
Although the current exploration of this strategy has focused on providing more information to more parties throughout more parts of the software life cycle, the reverse is equally important. It must be possible to guarantee that shared information be reliably managed with respect to which parties can see which information at what times. The loosely coupled approach proposed here is a good start on this point, since it presumes that each organization autonomously manages a repository of information that represents its intellectual property.

6. Application Scenarios

A prototype has been used to explore the proposed concept of an application web. Scenarios from that prototype, described in this section, involve a web of four organizations and demonstrate how organizations exploit precise knowledge about applications and the connectivity provided by the web.

A Web Setting

This web spans four organizations. *HealthPro*, a health care organization, uses a graphical business process support application, *ProcessEdit*, provided by *ProcessPro*, a business process tool vendor. To implement this application, *ProcessPro* uses a graph layout package provided by *GraphPro*, an organization specialized in graph model and layout software. *HealthPro* contracts an external consulting organization, *ConsultPro*, which provides assistance as needed to operate the application. The scenarios in this connected world of organizations include activities such as developing, deploying, locating, running, managing and inspecting an application. A participating organization uses different services to accesses the web with respect to its interest into a particular phase of the application life cycle.

GraphPro offers two versions of a graph layout package (named com.graphpro.graph.layout.1 and com.graphpro.graph.layout.2). *ProcessPro* offers four versions of a business process support application, *ProcessEdit* (version 4 is named com.processpro.tool.processedit.4 and uses com.graphpro.graph.layout.2. *GraphPro* and *ProcessPro* provide federated browsing, development and deployment services on the web. *HealthPro* provides browsing, application management and inspection services, whereas organization *ConsultPro* offers online consulting services.

Scenario 1: Simple application management
A manager at *HealthPro* connects to the application web using the browsing service to locate version 4 of *ProcessEdit* published at *ProcessPro's* web server. The manager launches the application by pressing a button without any copying or prior installation. The deployment service collects the relevant executable application parts directly from *ProcessPro* (com.processpro.tool.processedit.4) and *GraphPro* (com.graphpro.graph.layout.2). The application management service at *HealthPro* lists the running application including hyperlinks to the web locations of all involved application parts. By following these links, the manager can locate precise additional information about the application provided by *ProcessPro* and *GraphPro*, such as documentation, sources or license information.

The scenario illustrates simple cooperations based on comprehensible and reliable services that take advantage of the shared knowledge and connectivity in the web. The shared, versioned name space allows for simple application location. The deployment and management service support application launching without any prior installation. Links from parts of a running application to their origins in the web allows to reliably access useful additional information about the application.

Scenario 2: Development and deployment of a customer special version
HealthPro is not satisfied with version 4 of *ProcessEdit* and asks for additional functions. A developer at *ProcessPro* receives the request (which names the configuration currently used) and builds a variant of that configuration using the JP federated development services of *ProcessPro* and *GraphPro*. The new version (com.processpro.tool.processedit.4.healthpro.1) is immediately available on the web and the manager can launch the new version immediately upon notification.

The scenario illustrates how the shared naming system allows for precisely name application parts including version and configuration information which is essential for cooperations across organizations. The JP federated development service is a more sophisticated service allowing for cooperative software development across organizations. The web concept including autonomous evolution allows to make new application versions immediately available and to launch them without installations.

Scenario 3: Remote online consulting
The manager at *HealthPro* contacts a consultant at *ConsultantPro* in order to learn more about the new version of *ProcessEdit* that is currently running. The consultant remotely connects to

HealthPro's application management and inspections services. Every part of the running application contains links to original information published directly by *ProcessPro* and *GraphPro*. This allows the consultant to find precise information about the running application such as sources, documentation, design artifacts, debug traces and more. Additionally, the object structure of the application can be investigated using the inspection service. Again, for every visited object, a hyperlink reliably leads to the source of its implementation.

The scenario illustrates how a consultant has remote access to extensive information about running applications at the customer site. This also includes links and access to the origin of the application. Applications can also be investigated at object granularity.The precise information is fundamental for effective consulting.

7. Related Work

The Vesta project [LM93]), from which some of JP's technologies are derived, pursued reliability in the face of scale, but only in the traditional dimensions of application size and compositional complexity

Goals for the JP project include those of Vesta, but added, among others, the ability to perform complex builds across multiple distributed sites[JV97,Van98]. However, this was understood to require that all sites be running JP software, and support did not extend through the full software life cycle.

Bischofberger et. al. make a related argument about capturing information available in the context where software originally gets created [BKMS95]. Their emphasis is on informal communication, not only more formal information such as sources and documents, and how it is very important when organizational boundaries are crossed.

Several projects have explored how to take software configuration management global. For example Noll and Scacchi propose an integration layer between autonomous repositories servers and their clients that would provide the appearance of a central repository [NS97]; this presumes much more infrastructure sharing and process-coupling than does the application web approach.

8. Conclusions and Outlook

We have been exploring a new strategy, called the application web, that will permit collaborating but autonomous organizations to reliably share information throughout the life cycle of software products. The design of the application web address the need for a balance between autonomy and cooperation, all in the context of reliable services.

The problem of information impedance at organizational boundaries, which arises where tight tool-based coupling is impractical, is addressed with a simple, but flexible approach to sharing information that permits a variety of collaboration models, including tightly-coupled tools, for example for federated building and possibly distributed configuration management, but also simpler relationships such as application downloading and WWW browsing. The unreliable drudgery of manual copying of information, an approach that does not scale well, is replaced with links that permit automatic caching.

Prototype versions of these services have been implemented in the context of the JP software development environment, whose repositories rely on an experimental system for persistent Java objects [Atk96]. The effect is to create a web of knowledge about each software artifact, quite a different view of the software life cycle than is customary. It adds kind of connectivity becoming common on the WWW, but it is still implemented on top of the kind of reliable services that large scale software development demands.

It is expected that such a strategy will open up new business opportunities for software development of the sort envisioned by the Virtual Software House: software services available on the Web; consistent, up-to-date, connected software catalogues; reliable software bundling and deployment; component seeking and matching; component interoperability checking; online consulting; pay per use; and many others yet to be imagined..

Footnotes

Footnote 1: The term "configuration" carries a great many meanings. Using it to mean "unit of software sharing" is somewhat non-standard, but it captures the intended notion that it is more than just a collection of Java classes. A configuration is assumed to have been constructed carefully by some organization for some purpose; it is very likely one of a family of versions, each representing successive refinement; it may be one of many variants, each representing special needs from its consumers; it presumably has associated documentation describing its use as well as interoperability restrictions; and it has presumably been subjected to some form of quality control.

Footnote 2: A corollary of the requirement that a package build be location independent is that it also be time independent: it always gives the same results. JP makes this guarantee by requiring that only immutable package version can be built, and that their build script be a completely closed functional program.

Acknowledgments

This work benefits greatly from the vision of Mick Jordan, Principal Investigator of the Forest Project at Sun Microsystems Laboratories and coauthor of the JP prototype development environment. The VSH project is supported by Prof. Albert Kündig at ETH Zürich and funded by the Swiss Priority Program of the Swiss National Science Foundation. Yuval Peduel made helpful comments on early drafts of this paper.

Trademarks

References

[Atk96] Malcolm Atkinson, Laurent Daynes, Mick Jordan, Tony Printezis and Susan Spence. An Orthogonally Persistent Java. *ACM SIGMOD Record*, Volume 25, Number 4, December 1996.

[BKMS] W.R. Bischofberger, T. Kofler, K.-U. Mätzel, and B. Schäffer. Computer Supported Cooperative Software Engineering with Beyond-Sniff. *Proc. 7th Conf. Software Eng. Environments (SEE)*, Noordwijkerhout, The Netherlands, IEEE CS Press, 1995, 135-143.

[DM92] W. Davidow and M. Malone. *The Virtual Organization: Structuring and Revitalizing the Corporation For the 21st Century*, Burlingame Books, 1992.

[KD98] Gail E. Kaiser and Stephen E. Dossick. Workgroup Middleware for Distributed Projects. *IEEE Seventh International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 1998, pp. 63-68.

[GJS96] James Gosling, Bill Joy, and Guy Steele. *The Java™ Language Specification*, Addison Wesley, 1996.

[JV97] Mick Jordan and Michael L. Van De Vanter. Modular system building with Java^{TM} packages. *Proceedings. 8th Conference on Software Engineering Environments* . IEEE Computer Society Press. 1997, pp.155-63. Los Alamitos, CA, USA.

[LM93] Roy Levin and Paul R. McJones. *The Vesta Approach to Precise Configuration of Large Software Systems*, SRC Research Report 105, June 14, 1993, Digital Equipment Corporation.

[Mur97] Tobias Murer. The Challenge of The Global Software Process. *Proceedings of the Second International Workshop on Component-Oriented Programming (WCOP-97)*, TUCS General Publication No. 5, September, 1997 and *ECOOP'97 Workshop Reader*, Springer LNCS Vol. 1357, 1998

[MS97] Tobias Murer and Daniel Scherer. Structural unity of product, process and organization form in the GIPSY process support framework. *Proceedings. 8th Conference on Software Engineering Environments* . IEEE Computer Society Press. 1997, pp.93-100. Los Alamitos, CA, USA.

[MSW96] Tobias Murer, Daniel Scherer and Andy Würtz. Improving Component Interoperability Information. *Workshop on Component-Oriented Programming at ECOOP'96 (WCOP-96),* June 1996

[NS97] J. Noll and W. Scacchi. Supporting Distributed Configuration Management in Virtual Enterprises. *Proc. 7th Int'l Workshop Software Configuration Management (ICSE 97 SCM-7)*, Springer LNCS 1235, 1997, 142-160.

[Szy98] Clemens S. Szyperski. *Component Software: Beyond Object-Oriented Programming.* Addison Wesley, 1998

[Van98] Michael Van De Vanter. Coordinated editing of versioned packages in the JP programming environment. *System Configuration Management. ECOOP'98 SCM-8 Symposium. Proceedings*. Springer. 1998, pp.158-73. Berlin, Germany.

[VM99] Michael L. Van De Vanter and Tobias Murer. Global Configuration Names: How Reliable Java Software Development and Deployment can cross Organizational Boundaries*. to be submitted to Ninth International Symposium on System Configuration Management (SCM-9), September 1999, Toulouse*

[WCOP97] Selected workshop papers. *Proceedings of the Second International Workshop on Component-Oriented Programming (WCOP-97)*, TUCS General Publication No. 5, September, 1997 and *ECOOP'97 Workshop Reader*, Springer LNCS Vol. 1357, 1998

--------------3FBB936F35A6CA381AECE5BC--