# Supporting Distributed Workflow Using HTTP

Peter J. Kammer
Gregory Alan Bolcer
Richard N. Taylor
Arthur S. Hitomi

Department of Information and Computer Science
University of California, Irvine, CA 92697

Technical Report 98-05

February 19, 1998

### Abstract

Frequently process workflows are distributed collections of activities that involve groups of individuals at disparate locations. To coordinate these tasks, a process support system should provide for distributed process execution and integration with tools across networks.

We describe the use of the Hypertext Transfer Protocol (HTTP), an increasingly ubiquitous technology, to provide a coordination mechanism for distributed process execution and tool integration. Building on the Endeavors process support system, we discuss extensions to an HTTP server to provide support for communication and coordination between system components as well as integration with external tools. We also examine several progressively more distributed and flexible approaches to distribution using HTTP and Endeavors.

# Supporting Distributed Workflow Using HTTP

**Peter J. Kammer** **Gregory Alan Bolcer** **Richard N. Taylor** **Arthur S. Hitomi**

Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425 USA
+1 714 824 8438
{pkammer, gbolcer, taylor, ahitomi}@ics.uci.edu

## Abstract

Frequently process workflows are distributed collections of activities that involve groups of individuals at disparate locations. To coordinate these tasks, a process support system should provide for distributed process execution and integration with tools across networks.

We describe the use of the Hypertext Transfer Protocol (HTTP), an increasingly ubiquitous technology, to provide a coordination mechanism for distributed process execution and tool integration. Building on the Endeavors process support system, we discuss extensions to an HTTP server to provide support for communication and coordination between system components as well as integration with external tools. We also examine several progressively more distributed and flexible approaches to distribution using HTTP and Endeavors.[1]

## Keywords

Software process and process improvement, hypermedia, project management, computer supported cooperative work (CSCW) and software engineering, distributed and parallel systems, environments: organization and integration principles.

## 1 Introduction

Complicated process workflows are inherently distributed. They involve a wide range of people and disparate locations. Heterogeneous mixes of hardware and software are combined in activities that cross workgroup and organizational boundaries, possibly across networks to locations around the world.

A process support system must not only support and coordinate widely removed people and activities, but also support the heterogeneous mix of tools and applications they use. Endeavors [4] is a process support system designed to be easily integrated with various off-the-shelf software tools and technologies. It provides a customizable process infrastructure designed to support distribution, multiple views, and incremental adoption of system components.

Hypertext Transfer Protocol (HTTP) [2, 7] is a widely supported protocol that underlies the World Wide Web (WWW). Tools supporting it are quickly becoming ubiquitous technologies. We describe here our use of HTTP as a distribution mechanism for the Endeavors system. Included are progressively more powerful and flexible approaches that extend the capability of an HTTP (web) server to support additional functionalities.

Section 2 provides an overview of the Endeavors system and its general architecture. Section 3 explains our criteria for selecting a distribution mechanism. Section 4 discusses our rationale for using an extensible HTTP server. Section 5 describes a number of approaches to utilizing HTTP. Section 6 explains how distribution using HTTP extends the ability to integrate with other systems. Section 7 discusses related work, and finally Section 8 presents conclusions and future work.

## 2 Overview Of The Endeavors System

Endeavors provides an open, extensible process support infrastructure. The system uses a layered object model to provide for the object-oriented definition and specification of process artifacts, activities, and resources. Interaction between system components is event driven. Behavior of process objects is specified through the use of handlers: code invoked by the object in response to events received. Store locally or loaded from a remote source, handlers are bound to objects at runtime and may be changed dynamically in the course of process execution.

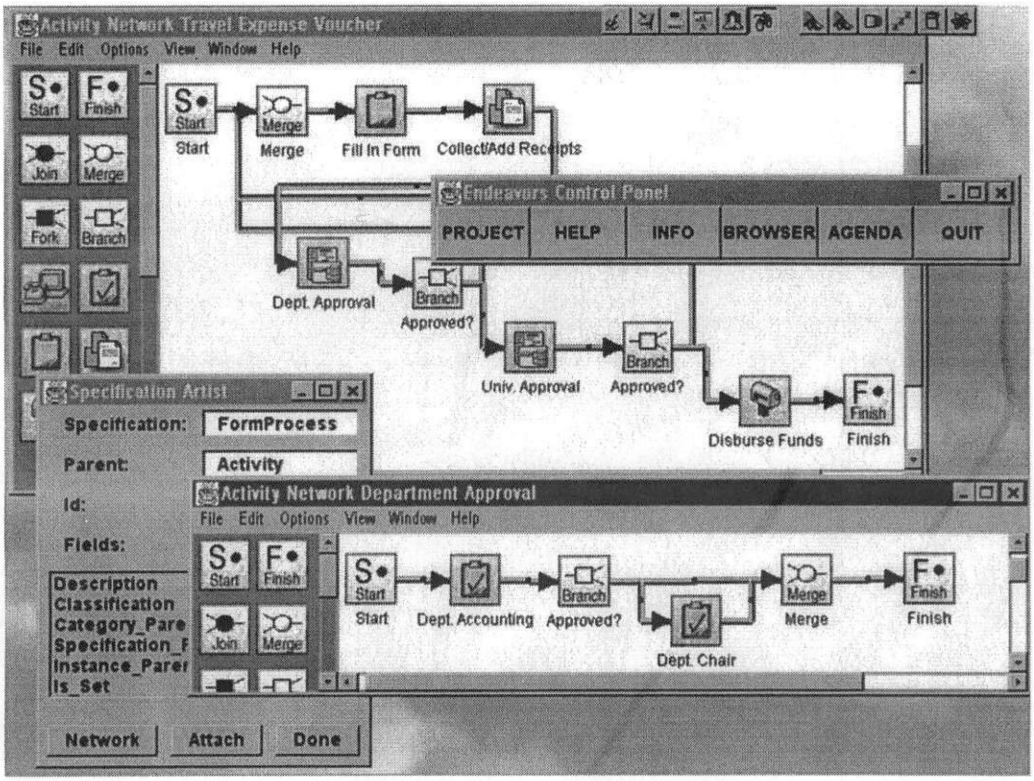
Activity networks associate activities by control-flow, data-flow, and resource-flow relationships. A rich user interface provides for their dynamic specification and control of their execution. Networks are executed by interpreters that traverse a network and send appropriate events to objects to invoke the objects' behaviors.

Figure 1 shows one example view of an Endeavors activity network. The largest window shows the top level process. The activity *Dept. Approval* is expanded into the sub-network shown below the larger main network. Also visible is the main control panel and a dialog for editing the individual attributes of an activity.

[1] Effort sponsored by the Defense Advanced Research Projects Agency, and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-2-0021. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Air Force Research Laboratory or the U.S. Government.

Approved for Public Release - Distribution Unlimited



Specification: FormProcess

Parent: Activity

Id:

Fields:

Description
Classification
Category_Pare
Specification_F
Instance_Parei
Is_Set

**FIGURE 1. One view of Endeavors activity networks**

Figure 2 shows a high-level view of the system architecture and functional breakdown. Endeavors has a three-tiered architecture. Each layer has its own object model and responsibilities. The *user* layer provides the interface for human interaction with the underlying system and processes. It monitors events from the underlying layers and maintains a coordinated view of the system and the processes specified. User actions are translated into lower level operations and events. Beneath the user level, the *system* level provides the domain object model. At this layer, artifacts, activities, and resources may be programmatically created, manipulated, associated in networks, and executed by an interpreter. The *foundation* layer implements the Class-Metaclass model [22], managing the loading of objects and handlers as well as their persistent storage. This layer triggers an object's handlers in response to received events.

Endeavors provides an open architecture that simplifies integration with external tools, providing for two-way communication. Handlers may activate and manipulate external tools through existing or custom APIs. External entities may access Endeavors through its open interfaces provided at each layer of the system's virtual machine architecture, allowing manipulation of process state or of the processes themselves (subject, of course, to authorization controls).

Finally, Endeavors leverages off the capabilities of the Java programming language [8] in which it is entirely written. The system is highly componentized to facilitate incremental adoption and reuse. System functionality may be downloaded to process execution sites as required without the need for explicit installation processes.

Additional detail about Endeavors may be found in [4] and [22].

## 3 Criteria For Selecting A Distribution Mechanism

We focused on several criteria in choosing a distribution mechanism. In addition to examining how closely the functionality provided matched the needs our application, we were also concerned with matters of availability, ease of adoption, and ability to integrate with a wide scope of external tools.

Our initial functional requirements were that the technology allow the sharing of object repositories and communication of events through active message passing. The flexibility to readily extend the technology with additional (possibly unanticipated) functionality was also necessary.

Other desirable characteristics included low adoption cost, wide availability, and broad support from third-party tools. These qualities provide the greatest flexibility in supporting incremental adoption and ease of integration within existing environments.

Finally, rather than needing to support and promote the technology, we hoped to leverage off the efforts of others in supporting its ongoing development.

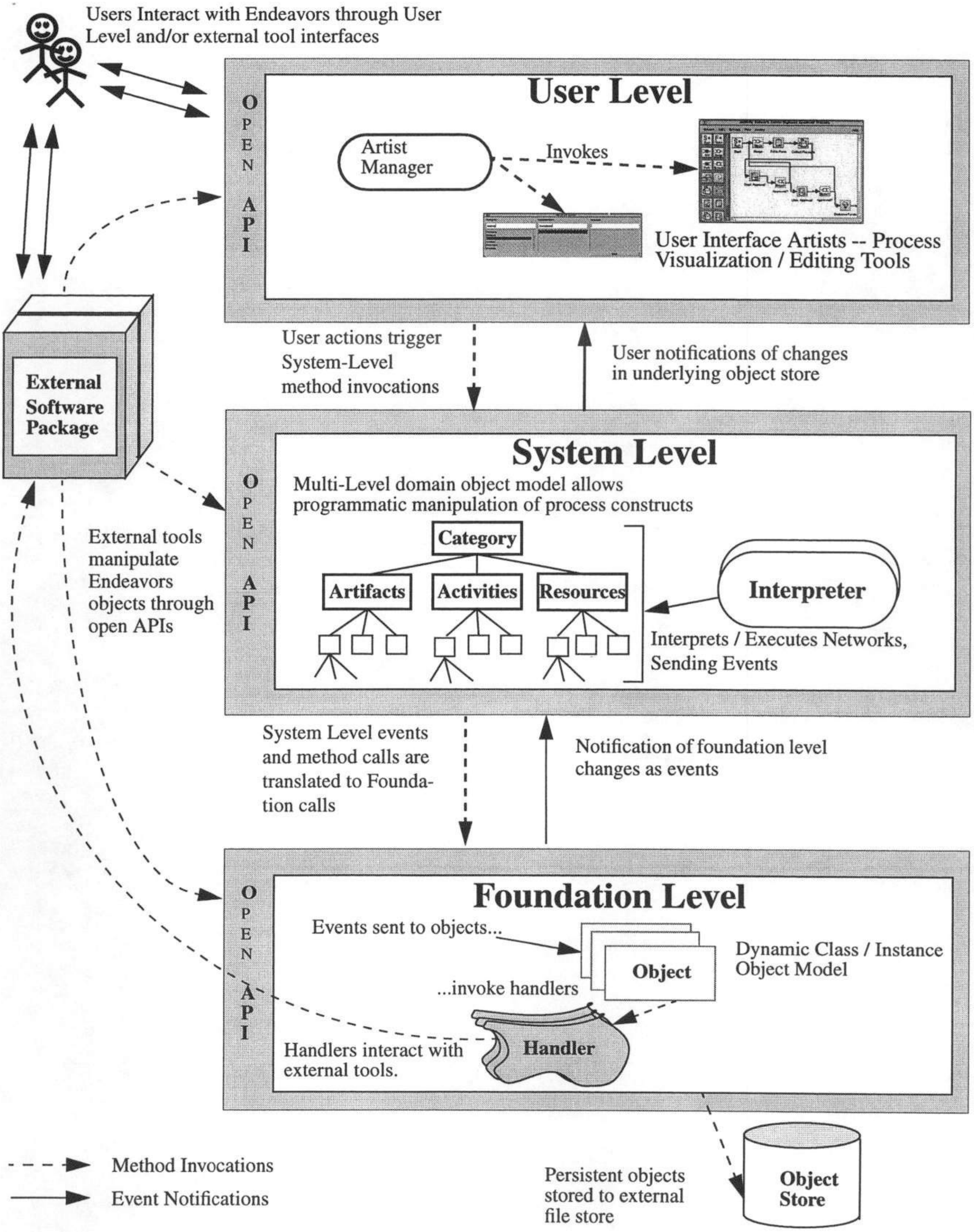

Users Interact with Endeavors through User Level and/or external tool interfaces
OPEN API
User Level
Artist Manager
Invokes
User Interface Artists -- Process Visualization / Editing Tools
User actions trigger System-Level method invocations
User notifications of changes in underlying object store
External Software Package
OPEN API
System Level
Multi-Level domain object model allows programmatic manipulation of process constructs
Category
Artifacts
Activities
Resources
Interpreter
Interprets / Executes Networks, Sending Events
External tools manipulate Endeavors objects through open APIs
System Level events and method calls are translated to Foundation calls
Notification of foundation level changes as events
OPEN API
Foundation Level
Events sent to objects...
Object
Dynamic Class / Instance Object Model
...invoke handlers
Handler
Handlers interact with external tools.
Persistent objects stored to external file store
Object Store
Method Invocations
Event Notifications

**FIGURE 2. Endeavors Layered System Architecture**

## 4 The Choice Of HTTP

We explored a number of mechanisms for providing communication between distributed Endeavors systems. Issues associated with the choice of HTTP are considered in Sections 4.1 through 4.3; some alternative technologies are briefly considered in Section 4.4. HTTP is an increasingly ubiquitous technology driven by its widespread use in WWW applications. The protocol is widely supported and allows simple integration with the distribution mechanisms of many other tools. In addition, many systems that manage the interaction between local and external networks, such as proxies and fire-walls, are designed to support and optimize HTTP interactions. Other approaches may require the additional investment of modifying or replacing these systems.

### 4.1 Further Evolution of HTTP

One of the appealing strengths of HTTP is the ongoing effort to improve the protocol, both in terms of efficiency and functional capability. Examples of this effort are the release of the HTTP 1.1 draft standard [7] and the ongoing work in distributed authoring and versioning [18, 21].

#### *4.1.1 HTTP 1.1*

HTTP 1.1 provides a number of improvements to the protocol's overall efficiency that will speed data transfer between client and server as the standard makes its way into implementations. Persistent connections will reduce the overhead for multiple transactions between the same client and server. Further, the improved caching specification will reduce the number of needed transactions.

#### *4.1.2 Distributed Authoring and Versioning (WebDAV)*

WebDAV provides a number of functional extensions to HTTP to support collaborative authoring of web resources. Once WebDAV reaches implementation, leveraging off these functionalities will provide added flexibility.

*Write locking* allows a single client to obtain an exclusive write lock on a resource. This will not only facilitate the interaction of clients, but also allow the integration of third-party tools in a well-behaved standardized way. Combined with versioning, write locking will facilitate maintaining a consistent object store, even while it is accessed concurrently by multiple clients.

*Versioning* allows multiple instances of a resource to follow separate parallel development paths, perhaps later combined by a merge tool. This not only allows multiple clients to follow parallel process execution threads with copies of the same resource, but also supports the possibility of "process rollback" to a previous state by reverting the object store to earlier versions.

*Metadata* provides information about resources such as authoring, ownership, and age. *Name space management* provides access to collection hierarchies (such as file systems). These technologies can be utilized to handle locating and managing resource attributes and associations at the HTTP level.

### 4.2 Extending an HTTP Server via Servlets

In surveying the available web servers, we found that we needed to extend a server to provide added functionality. For example, most web-servers did not support the HTTP "PUT" operation, or writing back to the server, for security reasons. A readily extensible server also provides the flexibility for additional functionality in the future as needed.

Traditional web-servers have provided executable content on the server through auxiliary programs accessed via CGI (Common Gateway Interface). In this approach, subprograms are executed by the server in separate processes, each with distinct memory space. Each time the functionality of the subprogram is called for, a new process and version of the program is instantiated. Context information, such as parameters and path information, are provided to the subprogram by variables in the execution environment.

*Servlets* [20] provide an alternative method for extending the functionality of a web server. Supported by a number of web servers, servlets are similar to the familiar *applets*, the small executable components that add interactivity to many web pages. While applets are downloaded and run in the browser, however, servlets are executed on the server machine, with access to the web-server environment and file system.

Servlets provide a number of advantages over CGI for our application. They are loaded into the server the first time they are invoked and share the server's process and memory space. This eliminates the overhead of creating a separate process each time the functionality is required. Because the servlets are initialized only once, they can retain persistent state between invocations. Further, since servlets all share the common memory space of the server, communication between servlets may be achieved through shared data structures. CGI applications, running in separate processes are forced to communicate through either the file system or interprocess communication.

We found that servlets provided a simple, flexible way to extend server functionality. We will discuss specific extensions in Section 5.

### 4.3 Drawbacks to HTTP

The most significant drawback to the use of HTTP is that it primarily follows a traditional client-server model of interaction. That is, the server responds to commands from the client but does not establish connections or transfer information without a request. This makes it problematic to have the server notify running clients of events or messages that may be of interest. The approach taken to address this problem will be discussed in detail in Section 5.2.

### 4.4 Other Technologies for Supporting Distribution

#### *4.4.1 Java Remote Method Invocation (RMI) and Serialization*

Java RMI [19] is a native Java functionality that invokes methods on remote servers. Values of parameters, return

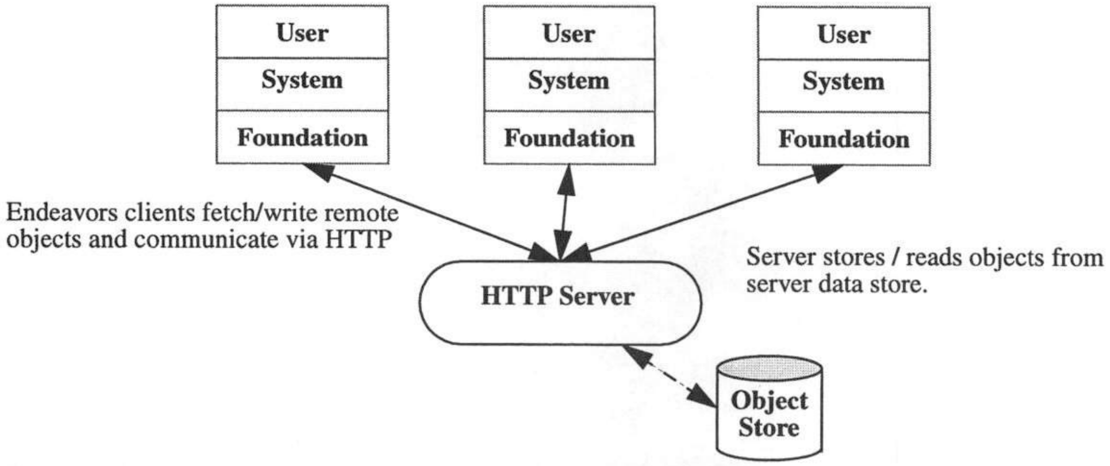

User
System
Foundation
User
System
Foundation
User
System
Foundation
Endeavors clients fetch/write remote objects and communicate via HTTP
HTTP Server
Server stores / reads objects from server data store.
Object Store

**FIGURE 3. Multi-User configuration with a single remote server providing data storage and message passing.**

values and their relationships are serialized and passed as a stream from client to server. RMI is now a standard component in the Java language version 1.1. This approach promises to provide a more elegant, and perhaps more efficient, communication mechanism. RMI is Java-specific however, and does not provide the potential for interaction with the variety of tools that support HTTP. Further RMI has not been fully supported in all Java execution environments, particularly some WWW browsers.

#### *4.4.2 Client / Server with Centralized Database*

One traditional approach to distribution is a centralized database provided by a server and accessed by remote clients. A number of such systems also provide HTTP interfaces. This approach provides the power and efficiency of a full database management system, but it does not, however, provide the necessary flexibility to extend the server with additional functionality. Further, installing a database system and server is likely to significantly increase the cost of adoption. For our work, the additional power of a client-server database was not justified by this loss of flexibility and the additional investment that such a system would require. Nevertheless, we have adopted several key ideas from this approach and discuss them in Section 5. Moreover, our approach does not preclude integrating a database management system with the foundation layer (on either client or server) as a persistence mechanism.

#### *4.4.3 Common Object Request Broker Architecture (CORBA)*

CORBA [17] is an open standard developed by the Object Management Group (OMG) to provide a specification for the interaction of heterogeneous objects through specified interfaces. Similar to client-server databases, installing and maintaining a CORBA product increases the entry barrier and discourages incremental adoption. Further, CORBA is not yet a pervasive technology, limiting the tools, technologies and platforms that will support it. The technology does have numerous merits, however, and we will continue to reevaluate our choices as the context evolves.

## 5 Approaches To Configuring Distribution

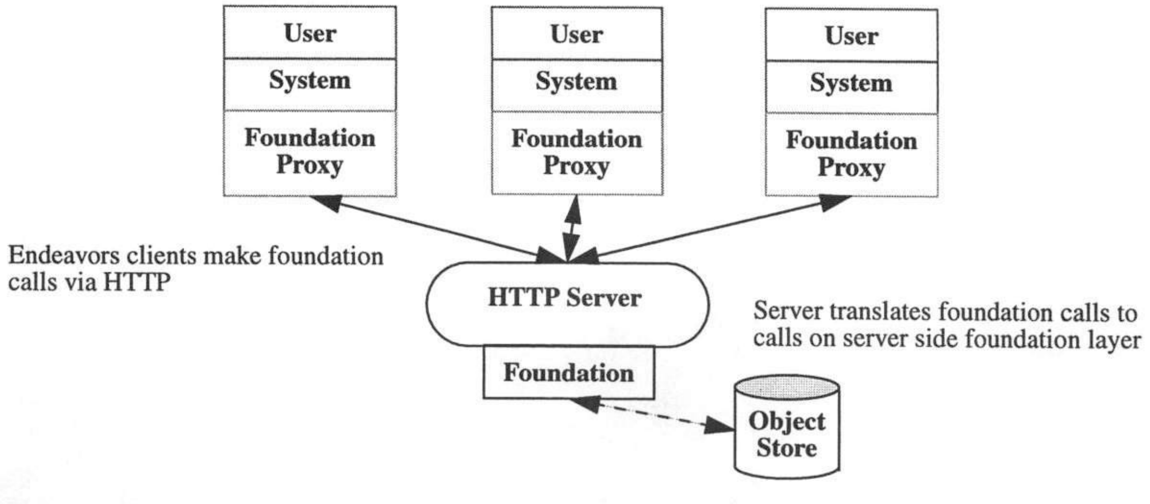
A variety of increasingly more flexible and powerful approaches to supporting distribution using HTTP have been explored in a series of prototypes using Endeavors. We present these in the following sections. Our intent has been to support a wide range of configurations with varying degrees and kinds of distribution. Depending on the needs of the user, each approach has its sphere of appropriate application.

### 5.1 Stand-alone

Figure 2 shows the base system configuration without distributed components. A user interacts with the system through the user-interface provided at the user level or through interfaces provided by external tools. Direct method calls to lower layers of the system invoke object behaviors in response to the user's actions. Persistence is provided by a local file-store. As process activities are executed by the interpreter, handlers are invoked on the local machine to interact with tools which then may make API method calls to access Endeavors objects. This configuration is appropriate for individual users manipulating their own processes as well as users wishing simply to model, simulate, analyze, and experiment with processes without actually executing distributed activities. It may also be appropriate for small groups of users utilizing a single machine.

### 5.2 Multi-User with a Single Remote Data-Store

Our initial extension to the single user model was to provide concurrent access to a single data repository to support groups of users working on a common project. Users on remote machines share an object server and maintain parallel views of the same underlying system state. Figure 3 illustrates this configuration. Each client is a complete Endeavors system (as shown in Figure 2). The HTTP server replaces the functionality of the local data store by providing remote data access. In addition, it supports communication between the independent clients. Process interpretation, handler execution, and implementation of the Endeavors object models takes place on the client side.



User
System
Foundation Proxy
User
System
Foundation Proxy
User
System
Foundation Proxy
Endeavors clients make foundation calls via HTTP
HTTP Server
Server translates foundation calls to calls on server side foundation layer
Foundation
Object Store

**FIGURE 5. Multi-User configuration with a remote server providing foundation services.**

Similar to the way a web-browser caches pages, in this design Endeavors caches content received from the server. Once the data is obtained, a transaction need only occur to write it back or to obtain an update if the item is changed by another client. This approach is efficient when a large number of fine-grain operations are going to be performed because it limits the number of necessary HTTP transactions.

There were two principle extensions to the server to provide for this added functionality; these are shown in Figure 4. First, we added functionality to write back to the server file store and implemented a simple locking mechanism to prevent conflicts when accessing files. A simple security extension limits access to particular machines. Second, we added support for message passing. This was a more complex task. The HTTP server is essentially a passive entity. It responds to requests from clients, but it cannot actively connect to a client. We worked around this by using a mechanism whereby clients register for messages with the server and then maintain an open HTTP connection. Execution threads in both the client and server block, waiting for input on the open connection. When a message is received for the registered client, it is fed through the open connection to the waiting client whose thread resumes execution and processes the message, resuming its passive monitoring state when done.

This approach is more efficient than a polling method which would waste resources on both the client and server side. The connected threads are passive until activated by a message. It is, however, less robust as it requires that the connection remain open.

### 5.3 Moving Additional Functionality to the Server

In the previous approach, with the server providing a mechanism for distributed access to files, individual Endeavors clients each have their own complete foundation layer. Each time a new copy of an object is required or an object is changed, the entire object must travel between client and server. Further, each foundation caches its own copies of objects so keeping clients synchronized is a formidable task requiring additional communication.

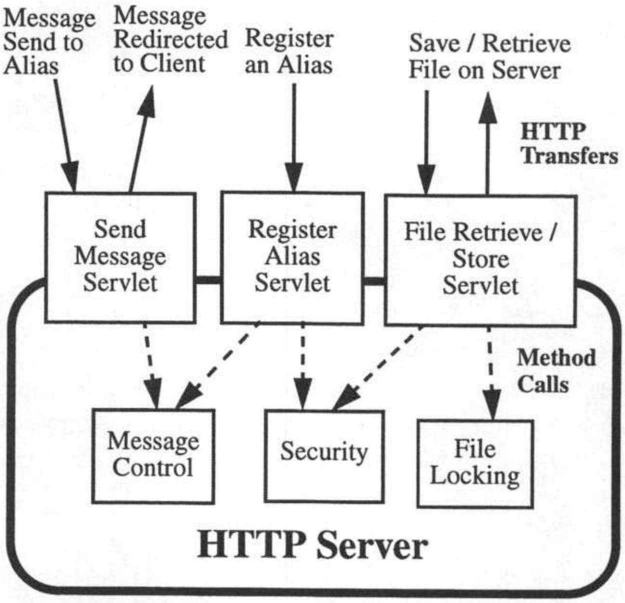


**FIGURE 4. Extensions to web server to provide additional functionality**

A further improvement to our distribution model, therefore, is to move the foundation functionality to the server. The server maintains the system state for multiple processes. The individual clients make calls to a local foundation proxy which in turn makes requests, via HTTP, to the server which executes the requests (returning any values). Figure 5 shows this configuration.

One effect of this division is that handlers triggered by events on objects are now executed on the server rather than the client. (We will look at a more flexible approach in Section 5.4.) The process interpreter, at the system level, still executes on individual clients.

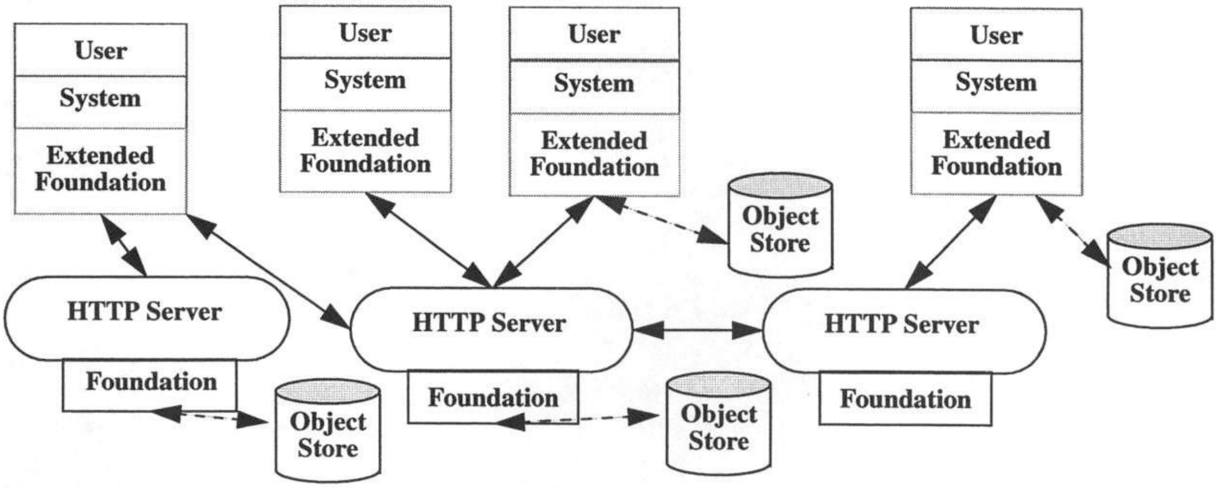


**FIGURE 6. Multi-user, multi-server configuration with distributed foundation services**

In contrast to the previous approach, this configuration requires a separate HTTP transaction for each operation on an object. This may be more efficient for situations where large objects are accessed with only a few operations as the entire object does not need to be transmitted to the client.

### *5.3.1 Server Extensions*

This approach required extending the server in a less conventional way. While the previous configuration only required that we be able to read and write files as well as pass simple messages, this configuration requires the proxy foundation on the client translate foundation calls, with parameters, into text HTTP requests for the server. The server then translates the request back into a method call, responding with any resulting return value from the call, also converted to text. The proxy then converts it back to the appropriate data type.

This approach emphasizes simplicity, both for ease of development and integration with external tools, but this also produces drawbacks. The translation of complex data types to text, transmitted via HTTP, sacrifices type information. Client and server must be consistent in their interfaces and interpretation of the data, limiting the robustness of this methodology. In addition, the steps of converting from typed data to text and then back represent additional overhead for each transaction.[1]

### *5.3.2 Application*

We have utilized this configuration in two of our process workflow applications. In the first, a document approval process, developed to support the requirements engineering activity of a large telecommunications company, one participant interacts with applets in a web-browser to submit a document for approval. The document and associated information are stored on the server while execution passes to other clients whose users also participate in the process. Each user's actions manipulate the status of the object on the server. This is accomplished through the underlying HTTP interfaces.

Our second application also reflects an instance where there are multiple participants in the process with different roles. In this web-based-learning (WBL) process, developed for a workstation manufacturer, there are "course designers" who provide the overall structure of the activities, "content providers" who provide the information for the courses, "students" who review the material and are tested on it, and finally "course managers" who monitor students' progress and the use of the course in general. These different roles require different visualizations of the underlying process taking place, as well as different subprocesses. This subprocess execution and individual viewpoint is provided by the individual clients that the users access the system through, interacting with the single server, via HTTP, to obtain a common set of data.

## 5.4 Multi-User, Multi-Server

Finally, we discuss an approach whose prototype is currently under active development. By providing the capabilities of both of the previous approaches—allowing manipulation of objects on either client or server—we increase the flexibility and, potentially, the efficiency of our design.

Rather than completely remove the foundation capabilities from the client, as was done in Section 5.3 above, we expand them to provide communication with the extended HTTP server. Object references provided to the foundation are given in terms of Uniform Resource Locators (URLs) [3]. The extended foundation interprets the URL and, depending on the location and the foundation's configuration, locates the object in the local data-store or makes a request of one of several remote servers. This configuration is presented in Figure 6.

This approach provides the greatest flexibility in configuration. Multiple servers mixed with local data stores make it potentially much more robust. Further, it provides greater scalability, potentially supporting large scale

[1] Note that this raises key issues of interoperability that have been explored in depth by other authors. See, for example, [1, 11, 14, 17]

networks of clients and servers. Objects can be cached locally and clients disconnected to work independently. Multiple servers support work across group or organizational boundaries. Servers can be configured to control access and coordinate process execution. For example a server owned by one organizational unit may provide access to its objects to a server or client from another unit, but limit or prevent object modification.

## 6 Distributed Tool Integration

Our approach to distribution also extends the ability to integrate external tools. We briefly summarize the methods of tool integration Endeavors supports and discuss how our approach adds to these capabilities.

### 6.1 Tool Integration Styles

*Non-communicative* tools are launched by triggered handler code. These tools do not provide external APIs, nor are they aware of Endeavors APIs. A handler may either wait for the tool to complete, so that, for example, its output may be examined, or continue to execute concurrently.

Tools with *externally visible controls* are similarly launched by handlers, but provide open APIs that may be accessed from within the executing process.

*Egocentric* tools are aware of and access Endeavors interfaces, but provide no comparable API for executing processes to access. Control from within the Endeavors process is limited to, at most, launching the tool, with further interaction directed by the tool itself. Access by the tool to process state is, of course, governed by controls on the underlying persistent object store, either file system or remote server.

*Cooperative* tools access Endeavors interfaces and also provide open APIs to manipulate the tool. Endeavors and the external tool act in concert to accomplish process activities.

### 6.2 Further Integration Approaches Using Distribution

Our distribution model expands the capabilities for tool integration. Handlers, launched by the foundation layer in response to events, may be executed on either the client or the server, providing added flexibility to the location where a tool is launched and where it interacts with the Endeavors system.

Distribution also provides an additional interface to Endeavors. In addition to the open APIs provided by each layer of the system (see Figure 2), the extended server provides HTTP access to foundation level services. All that is required is that an application make HTTP requests using simple URLs. This provides not only an additional interface mechanism, but extends the ability of remotely executing applications to integrate with the Endeavors system. A simple web-browser can access and modify the object store (again, subject to access controls).

In a tighter integration, cooperative tools can be included in processes through specialized agents, dedicated clients composed of Endeavors services closely integrated with the external tool, allowing it rich participation in process execution as a client. One example of this kind of application is a flight reservation agent integrated using a dedicated set of Endeavors client services to allow automated interaction with a reservation system. Interactions between the server and agents are the same as with clients serving human process participants.

Finally, extending an HTTP server provides for the integration of an additional kind of *server-aware* application. In a manner similar to the server extensions constructed for Endeavors, tools may be launched and integrated within the HTTP server to provide their own services, accessible by Endeavors clients, or by other HTTP-aware applications.

## 7 Related Work

Workflow process execution on the WWW takes a bipolar approach. One approach tightly constrains the actions that can be performed by a process stakeholder (such as a participant, end user, or designer) or software agent, but guarantees the internal data consistency of the workflow model and execution data. The other allows stakeholders to perform arbitrary actions to accomplish their activities but places minimal constraints on the structure of the intermediate and final data artifacts, providing few guarantees of consistency [5]. This section gauges several approaches exhibited by systems along this spectrum.

### 7.1 The Database Model

Traditional workflow systems are built on top of database systems. Activities are modeled as schema which include values for who (or what role) is responsible for the activity, what tools are needed to accomplish it, the relative priority of the activity at hand, and data dependencies, including precedence and document handoff. This approach limits the types of actions that an end-user can perform on each activity, as well as the changes that a workflow designer can make to the activities. Because the interactions are limited to pre-specified tools at pre-specified steps in the workflow, data consistency is guaranteed. Database systems also allow transaction support and rollback of processes to a previous state. This most often happens when the workflow model becomes out of sync with the real activities taking place. Each of the workflow process steps are data-driven. Specification of the control flow is minimal and often hard-coded into the database schema. Fixing the workflow steps and schema in this way requires the process to be stringently adhered to, thus impacting the dynamic evolution capabilities of the system, and also strongly constraining the work practices of the performing organization. By restricting the tools and actions allowed to guarantee data consistency, database systems tend to limit their flexibility. End-users cannot use arbitrary tools to accomplish their tasks. In addition, it is difficult to integrate new tools into the process or maintain local process state external to the database. Database systems typically have a high cost of adoption and require a critical mass [9] of users to participate in using the system before many of the workflow benefits are seen. There is no incremental adoption path for easily mixing human performed activities not stored in the database and those of the process model.

### 7.2 Ad Hoc Systems

An opposite approach to the database model is the ad hoc workflow model. This stems out of the CSCW field and is typified by such commercial systems as Netscape Collabra[16] and Microsoft Exchange[6]. Users are allowed to perform their work in an "ad hoc" way where solutions and goals are accomplished with whatever tools and data are immediately available. These systems are easily adopted and accessible through the user's desktop or browser. Their primary purpose is to inform users of the current problems, issues, and data needed to accomplish their particular tasks. While this approach tends to be appealing in that it mimics more closely the fast, furious, and unstructured pace of some projects, it provides minimal automation and guidance support. Workflow models lack a coherent, semantically rich, and precise description of the work being performed. Data and control dependencies, as well as assignments, are not immediately viewable, and measurable status of the project is difficult to ascertain. The ad hoc nature of these systems may cause the data and artifacts being created and shared to be inconsistent. However, because the changing data and rationale is recorded over time and is both viewable and shared by all relevant participants, any data inconsistencies are managed by the participants and not the system. This distinction allows for a broad degree of flexibility in accomplishing work, but increases the difficulty for managing it.

### 7.3 Hybrid Approaches

#### *7.3.1 Lotus Domino*

Lotus Domino[15] (the WWW extension to Lotus Notes) takes a hybrid approach. Domino mixes an underlying messaging and database system with mechanisms to allow stakeholders to view and change data and artifacts through a WWW browser. Data is stored in a Lotus Notes server, and distributed sites use a replication strategy. Domino includes development tools for workflow development and tool integration, and workflows are actually deployed as applications. Domino's support for Java and Java components [10] allows for some post workflow deployment evolution by allowing the presentation or view of the workflow to be changed. However, the underlying workflow model typically does not evolve once the workflow application has been deployed. Domino provides some WWW interfaces for manipulating and extracting information out of the underlying relational database, but typically this is controlled by the workflow application developer and not the end-user. Domino represents a slightly more open approach than the traditional database workflow systems in that custom views into the workflow can be created via the WWW through a standard set of APIs. These views, however, are usually fixed before deployment to maintain data consistency. The tools integrated into the workflow process, as well as the control flow and data dependencies, rarely (if ever) change once the workflow process begins execution. While this may be necessary to guarantee data consistency, it represents an inhibitor to the usefulness and accuracy of the system.

#### *7.3.2 OzWeb*

OzWeb [12, 13] allies itself much more closely with the WWW-based workflow systems. While it too has an underlying database, it provides a much more semantically rich process representation. The OzWeb system is highly componentized and has specific components for dealing with process descriptions, transactions, rules and constraints, tool integration and management, and inter-site coordination. Data or events can trigger rules and constraints to both proactively and reactively drive the workflow process. OzWeb supports evolution in that control flow and data constraint rules can be dynamically added, evaluated, and triggered over HTTP. In order to maintain distributed data consistency, OzWeb employs a summit and treaty negotiation model. When two distributed sites encounter a condition where the data is inconsistent, the system initiates a resolution mode to reconcile, either automatically or with the involvement of human participants, the inconsistent data. In this way, OzWeb allows the distributed components of a workflow process to proceed with local execution while providing coordination and control support. Because OzWeb processes are, at its center, rules, modification or evolution of a running process often requires a high degree of technical expertise. Those participating in a running process are not always the same stakeholders that have the ability to change the process to fit their work context or habits. This creates a mismatch between those developing or defining the workflows and those who use or benefit from it. This limits the types of end-user customizations that can be made to an executing workflow system.

### 7.4 Relationship to Endeavors

The Endeavors philosophy is derived directly from the WWW-based model. In a similar manner to an end-user managing the exception when encountering a "page not found" error on the WWW, Endeavors places the burden of resolution initially with the end-user. The end-user then has the option to employ automated or manual workarounds to the exception or inconsistency. Endeavors, however, differs from ad hoc workflow systems in that it includes a semantically rich process description language [22]. This allows highly structured process modeling and execution capabilities to be integrated into the workflow while not being so constraining that ad hoc work is inhibited. Endeavors processes allow multiple, customizable views. The default process view allows both non-technical and technical users to visually edit the workflow. Endeavors is also a highly componentized system. In addition to being able to reference and embed various Endeavors components into WWW pages and e-mail, various process fragments including their data and tools can be sent and executed using HTTP. Endeavors is flexible in its approach. Depending on how the system is customized, the level of data consistency enforcement allows Endeavors to be used to either primarily inform the users of their data and activities as in an ad hoc system or automate certain activities based on their inputs as in a standard workflow system. Similarly, Endeavors' policies can be customized to enforce the workflow process execution, by limiting the user's interactions thus keeping the data consistent, or as a support infrastructure to enhance the coordination capabilities between users by loosening these constraints.

## 8 Conclusions and Future Work

These approaches demonstrate the feasibility of using an extended HTTP server to provide distributed process workflow functionality. These techniques build on Endeavors' open extensible model to provide a mechanism for integrating with the increasing proliferation of Internet technologies. We have described our progression from single-user application to increasingly more powerful and flexible distribution approaches. Our goal is to provide a dynamically configurable system to support a wide range of organizational contexts and tasks. We believe this approach has broad application and may be usefully adopted by others.

Infrastructure that requires additional cost and effort to acquire and maintain can present a barrier to the adoption of new technologies and approaches. By utilizing extensible HTTP servers to provide a distribution mechanism, we take advantage of technology already available and familiar at many locations. We have examined and implemented a number of approaches to this integration and found it to be practical in supporting distributed process participants at minimal additional cost.

Flexibility, including tolerance for discontinuity and a dynamically changing environment are key emphases of this research. Process participants will likely find it necessary to disconnect from the network for a period of time, for example to travel with a laptop computer, and continue their individual activities. An effective distribution approach will need to support this sort of coordination and the ability to synchronize a process participant's state with that of others.

## Acknowledgments

The authors would like to recognize the hard work and effort in the design and implementation of Endeavors by Patrick Young, Peyman Oreizy, and Clay Cover. In addition we would like to acknowledge the members of the C2, Chimera, and WebSoft projects at UCI for their exchange of ideas during the development of this system.

## References

1. I. Ben-Shaul and S. Ifergan. WebRule: an event-based framework for active collaboration among web servers. In *Proceedings of The Sixth International World Wide Web Conference*, pages 521-531, Santa Clara, Calif., USA, April 1997.
2. T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol -- HTTP/1.0. Internet Informational RFC 1945, MIT/LCS, UC Irvine, May 1996. http://www.ics.uci.edu/pub/ietf/http/rfc1945.txt
3. T. Berners-Lee, L. Masinter, and M. McCahill. Uniform resource locators (URL). RFC 1738, CERN, Xerox, Univ. of Minnesota, December 1994. ftp://ds.internic.net/rfc/rfc1738.txt
4. G. A. Bolcer and R. N. Taylor. Endeavors: A process system integration infrastructure. In *4th International Software Process Workshop*, pages 76-85, Brighton, U.K., IEEE Computer Society Press, December 1996.
5. G. Cugola, E. Di Nitto, A. Fuggetta, and C. Ghezzi. A framework for formalizing inconsistencies and deviations in human-centered systems. *ACM Transactions on Software Engineering Methodology*, 5(3): 191-230, (July 1996).
6. D. Fay and T. Rizzo. Active messaging and the Microsoft exchange server. Whitepaper, *Microsoft Interactive Developer*, June, 1997.
7. R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, and T. Berners-Lee. Hypertext transfer protocol -- HTTP/1.1. RFC 2068, UC Irvine, DEC, MIT/LCS, January 1997. http://www.ics.uci.edu/pub/ietf/http/rfc2068.txt
8. J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison-Wesley. August 1996. http://java.sun.com/docs/books/jls/html/
9. J. Grudin and L. Palen. Why groupware succeeds: discretion or mandate? In *Proceedings of the 4th European Conference on Computer-Supported Cooperative Work*, pages 263-278, Kluwer Academic Publishers.
10. G. Hamilton, Ed. JavaBeans, revision 1.01. Sun Microsystems, July 1997. http://java.sun.com/beans/spec.html
11. R. Kadia. Issues encountered in building a flexible software development environment: lessons from the Arcadia project. In *Proceedings of ACM SIGSOFT Fifth Symposium on Software Development Environments,* pages 169-180, Tyson's Corner, VA, December 1992.
12. G. Kaiser, S. E. Dossick, W. Jiang, and J. J. Yang. An Architecture for WWW-based hypercode environments. In *Proceedings of the 19th International Conference on Software Engineering*, pages 3-13, Boston, Mass. USA, May 1997.
13. G. Kaiser, S. E. Dossick, W. Jiang, and J. J. Yang. WWW-based collaboration environments with distributed tool services. *World Wide Web Journal* (in press)
14. M. J. Maybee, D H. Heimbigner, and Leon J. Osterweil. Multilanguage interoperability in distributed systems: experience report. In *Proceedings of the 18th International Conference on Software Engineering*, March 1996.
15. Lotus Development Corp. Domino.Doc Whitepaper. Lotus Development Corp, April 1997.
16. Netscape Communications. Netscape collabra server 3.0, open discussion server for enterprise collaboration, Datasheet. Netscape Communications Inc., 1997. http://home.netscape.com/comprod/server_central/product/news/collabra3_data.html
17. Object Management Group. *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, July 1996. http://www.omg.org/corba/corbiiop.htm

18. J.A. Slein, F. Vitali, E. J. Whitehead Jr., and D.G. Durand. Requirements for distributed authoring and versioning on the world wide web, Internet-draft, work-in-progress. July 1997. http://www.ics.uci.edu/pub/ietf/webdav/requirements/draft-ietf-webdav-requirements-03.html

19. Sun Microsystems. Java remote method invocation specification. Sun MicroSystems, Inc., 1997. http://www.javasoft.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html

20. Sun Microsystems. The Java servlet api. Whitepaper, Sun Microsystems, Inc.,1997.

21. E. J. Whitehead. World wide web distributed authoring and versioning (WebDAV): An Introduction. *ACM StandardView*, 5(1): 3-8, (March 1997).

22. P. Young. *Customizable Process Specification for Technical and Non-technical Users*. Ph.D. thesis, University of California, Irvine. August, 1991.

**Relevant URLs**

CGI: http://hoohoo.ncsa.uiuc.edu/cgi/

Endeavors: http://www.ics.uci.edu/pub/endeavors/

HTTP: http://www.ics.uci.edu/pub/ietf/http/

Java: http://java.sun.com/

JavaBeans: http://java.sun.com/beans/

Java RMI: http://java.sun.com/products/jdk/rmi/

Java Serialization: http://java.sun.com/products/jdk/rmi/serial/

Java Web Server: http://jserv.javasoft.com/products/java-server/webserver/

Lotus-Domino: http://www3.lotus.com/products/domino.nsf

Microsoft Exchange: http://www.microsoft.com/exchange/

Netscape Collabra: http://home.netscape.com/comprod/server_central/product/news/

OMG and CORBA: http://www.omg.org/

OzWeb: http://www.psl.cs.columbia.edu/ozweb.html

Servlets: http://jserv.javasoft.com/products/java-server/servlets/

URIs/URLs: http://www.ics.uci.edu/pub/ietf/uri/

WebDAV: http://www.ics.uci.edu/pub/ietf/webdav/