WWW-based Collaboration Environments with Distributed Tool Services

Gail E. Kaiser, Stephen E. Dossick, Wenyu Jiang,
Jack Jingshuang Yang, Sonny Xi Ye

Columbia University
Department of Computer Science
1214 Amsterdam Avenue, Mail Code 0401
New York, NY 10027, UNITED STATES
212-939-7081/fax:212-939-7084
kaiser@cs.columbia.edu

CUCS-003-97

February 28, 1997

Abstract

We have developed an architecture and realization of a framework for hypermedia collaboration environments that support purposeful work by orchestrated teams. The hypermedia represents all plausible multimedia artifacts concerned with the collaborative task(s) at hand that can be placed or generated on-line, from application-specific materials (e.g., source code, chip layouts, blueprints) to formal documentation to digital library resources to informal email and chat transcripts. The environment capabilities include both internal (hypertext) and external (link server) links among these artifacts, which can be added incrementally as useful connections are discovered; project-specific hypermedia search and browsing; automated construction of artifacts and hyperlinks according to the semantics of the group and individual tasks and the overall process workflow; application of tools to the artifacts; and collaborative work for geographically dispersed teams.

We present a general architecture for what we call hypermedia *subwebs*, and imposition of *groupspace* services operating on shared subwebs, based on World Wide Web technology — which could be applied over the Internet and/or within an organizational intranet. We describe our realization in **OzWeb**, which reuses object-oriented data management for application-specific subweb organization, and workflow enactment and cooperative transactions as built-in groupspace services, which were originally developed for the Oz process-centered software development environment framework. Further, we present a general architecture for a WWW-based distributed tool launching service. This service is implemented by the generic **Rivendell** component, which could be employed in a stand-alone manner, but has been integrated into **OzWeb** as an example "foreign" (i.e., add-on) groupspace service.

Keywords: Computer-supported cooperative work, Environment frameworks, Hypermedia, Process, Tools, Transactions, Workflow

1 INTRODUCTION

In general, *collaboration environments* seek to improve end-result quality and team productivity during both pre-arranged and *ad hoc* group activities and projects, ranging from industrial decision support and military command and control, to healthcare management and delivery, to software engineering and engineering design, to publishing and education, to crisis management and emergency response. Collaboration environments assist information workers, both individually and together in a group, via data organization and search, coordination of concurrent access, process workflow [1] modeling and enactment, application program launching, and other services. Collaboration environment *frameworks* provide generic facilities that can easily and rapidly be tailored to a specific effort.

Hypermedia collaboration environments intertwine support for planning and execution of purposeful work by assisting information workers in generating, discovering, retrieving, updating, cross-referencing and exploiting all on-line materials that may plausibly be relevant to an endeavor. The conventional application-specific artifacts, e.g., press releases and intelligence reports, formally produced and consulted during the conduct of a project may make up only a small proportion of the relevant hypermedia. There could also be arbitrary informal documents such as email and newsgroup archives, meeting minutes perhaps including video and audio components, even scanned-in diagrams sketched on napkins, as well as traces of the process that has been followed to date during the project, with various analyses and metrics regarding the progress of the effort. Without such broad support, *ad hoc* efforts may be slowed down, particularly during their initial stages, due to lack of organized information access. Further, the documentation available after project completion may not include all the nitty details that came up during meetings, email, discussion groups, etc.; apparent minutiae, some of which may later be quite valuable for desiderata and post-mortem analysis, has likely been lost.

Hypermedia materials need not reside in a single repository constructed specifically for that project or internal to the encompassing organization, but could potentially reference documents from arbitrary external sources and even add on links between external data of arbitrary types whose native repositories or formats provide no hyperlink capabilities. End-users and the environment's task engine should be able to introduce links on top of and orthogonal to any hyperlinks embedded in documents by the authors, to add references to materials the authors never thought of, did not know about or that did not exist when the documents were written. Note this permits hyperlinking materials generated and managed *independently* from the collaborative activity, such as digital library publications, technical reports from unrelated institutions that provided inspiration or insights, videos or transcripts of seminars or television presentations with some bearing on the work, public resources, etc. These features are particularly useful for activities whose participating organizations and/or information resources must be drawn together quickly with limited time to set up a project-specific database and reformat existing data as hypermedia.

This is very nearly what the World Wide Web (WWW) provides. Thus we choose Web technology, particularly HyperText Transfer Protocol (HTTP) and HyperText Markup Language (HTML), as the infrastructure on which to construct hypermedia collaboration environments. Web technology can be applied within an organizational intranet, with references to external resources tunneled through "firewalls"; obviously, proprietary or secret materials would not be made publicly available on the Internet. However, Web technology alone, or in tandem with other distributed computing infrastructures (e.g., DCE, OLE, CORBA), does not provide the *services* necessary for collaboration:

- There is little *organization* of Web documents, a great advantage for generic use, but consequently project-specific navigation and search are challenging. Although new Web pages can be created with

[1] We use the terms "process" and "workflow" interchangeably throughout this paper, ignoring the quibble that workflows are generally relatively short and a long-lived process may encompass many workflows.

links to documents elsewhere on the Web, there is no general way to add on application-specific types, attributes and links to arbitrary materials. Application-specific organization would provide a common context for collaborative information sharing.

- There is no support for *workflow* modeling and enactment regarding access to Web pages. A workflow engine can present to users, and guide and assist them in following, a medical care plan, design process, business practice, etc. whose information components are stored as Web hypermedia. Workflow would provide a structured means for coordinating the roles that participants take in collaborative tasks.
- Although the Web may soon support versioning and configuration management following the "checkout model", there is little concept of *transactions* [2], with either the classical atomicity, consistency, isolation and durability properties, or permitting relaxations proposed for long-duration, interactive, and/or cooperative work [22, 18]. The first author argues in [32] why the checkout model is less attractive for collaborative efforts than the emerging variety of cooperative transaction models. Concurrency control of some sort is a key requirement for managing concurrent access to shared information.

Both collaborative and solo work involve more than browsing of HTML documents; artifacts stored in hypermedia collaboration environments come in many different formats. There is no standard means of *using* these disparate forms of information once found, particularly as inputs to tools. Software tools and application programs are to information workers as the hammer and chisel are to the carpenter: essential components in accomplishing the tasks required by their work. Imagine telling a carpenter that some of her favorite tools could only be used on a specific style of workbench, and other tools she uses every day can only be used on a different style of workbench, and a third set of tools are only available on a particular workbench located in a neighboring city. Users of collaboration environments face an analogous problem. To treat the Web as a repository for hypermedia collaboration environments, it is necessary to apply both conventional tools (document editors, spreadsheets, virtual whiteboards, etc.) and emerging Web-aware tools (HTML editors, emailers, etc.) to Web entities, and place tool results back onto the Web. Many useful tools run only on particular computing platforms (e.g., Solaris *versus* Windows), and economic constraints may restrict licensing to a particular host (because either the hardware or software is very expensive, e.g., a Cray numerical analysis package), making it difficult to integrate the use of these tools with a user's task at hand — preferably performed sitting at that user's office workstation or on a mobile host in the field.

All these hurdles get in the users' way. They must remember not only where they can run certain tools, but also the specifics on starting them (including environment variables needed, command line options, etc.). A distributed tool service can solve many of these problems by handling the knowledge about how and where to run a tool. All the user need do is request a tool from the tool server. The tool server is then responsible for remembering and deciding the specifics on running the tool. Many existing systems can benefit from the addition of (or integration of) a tool services component. The problem of managing the use of software resources needed by an organization (making tools as easy to run and as ubiquitous as possible) is one which can be solved by incorporating a tool server into the use of the system. We are particularly concerned here, of course, with integration of distributed tool services into hypermedia collaboration environments.

We propose functionality in terms of "subwebs" and "groupspaces", and then present an innovative architecture that makes it relatively easy to add application-specific subweb (subset of Web) organization, navigation and search, and workflow, transactions, tool launching and related groupspace (multi-user workspace) services, to the World Wide Web infrastructure. Our approach does **not** require a special browser, *any* browser that supports the HTTP 1.0 standard [9] is sufficient. Our approach does **not** require use of a special server, *any* HTTP-compliant web server will do. The "trick" is to use HTTP proxy servers, which can mediate all traffic from/to any browser with respect to any servers and can be configured to apply to all WWW browsers throughout a site. HTTP proxies are generally used for implementing shared caches and tunneling WWW traffic through organizational firewalls, and but can be applied for any purpose where automatic filtering of all URLs (Web Universal Resource Locators) is desired.

[2] Note that we use the term "transaction" as it appears in the database literature, to mean a sequence of operations generally taking a collection of data from one semantically consistent state to another, as opposed to its common usage in reference to WWW, i.e., a single request from a browser to a website server and the corresponding response.

We have implemented a prototype hypermedia collaboration environment framework called **OzWeb**, and several working environment instances. This research is concerned with how to apply project-specific data management, workflow, and cooperative transactions to Web materials, **not** which specific data definition notation, process modeling language and enactment model, concurrency control mechanism, etc. might be best for a particular class of collaboration environment. Thus we adapted these facilities from the Oz process-centered environment [6] to our architecture, rather than attempting to invent new ones or arguing in favor of our old ones. However, it was necessary to devise a new approach to tool management amenable to deployment over WWW. Thus we also describe an architecture for a generic Web-based tool server, our prototype implementation of such a system, dubbed **Rivendell**, and its integration with **OzWeb**. Finally, WWW admits to a wide range of user interfaces. Our sample graphical user interface (GUI) assumes the HTML 2.0 standard [8] (e.g., frames), but the GUI is completely replaceable and a less sophisticated non-frames version or a more sophisticated Java-based GUI could easily be substituted. We are working the latter, where a Java window interacts with the user and browser windows only display HTML documents.

We first give two motivating scenarios, one a "real life" situation that we sought to contend with in developing subwebs and groupspaces, and the other a hypothetical but realistic example of when distributed tool services are needed. Then in Section 2 we propose functionality for subweb organization and groupspace services to achieve hypermedia collaboration environments. Section 3 describes our architecture on top of the World Wide Web infrastructure, followed by Section 4 with our implementation in **OzWeb**. Section 5 focuses on our distributed tool service, as a groupspace service added on the **OzWeb**, covering requirements and approach, architecture, realization and integration. We conclude with evaluation of our results, in Section 6, comparison to related work in Section 7, and contributions and future work ideas in Section 8.

1.1 Motivating Scenario 1

The Spring 1996 Introduction to Software Engineering course at Columbia involved a team project where groups designed, coded and tested a query processor for a given object-oriented database system (OODB). The objectbase implementation was supplied as C header files and object code libraries, and a functional specification for the ideal query processor was given. Each group developed a structured design for the query processor. The design documents were swapped with other groups, who performed design review and then coded this design. The groups inspected and revised their own code, and swapped with another group for testing. Since only two groups had complete implementations, the "lobster" group tested the code from the "whitespace" group and all other groups tested lobster's code. [3] Some groups submitted their homework in HTML or postscript format from their Web pages.

The students in the lobster group were invited to take the Undergraduate Projects in Computer Science course in Summer 1996, to extend their query processor. Several graduate students and a staff member were concurrently working on the same OODB, and on the Oz and **OzWeb** systems constructed on top; everyone was working against tight deadlines. This was **not** "homework": they were developing a significant piece of our system — but would soon graduate and disappear, so their code would ultimately be maintained by other people.

We had numerous potentially *useful* artifacts on-line: The OODB manual, source code, interface and library; class lecture notes, homework assignments, required readings, etc.; the functional specification for the assigned query processor and the student proposal for its extensions; design documents from several groups; baseline code (at semester end) and new code (in progress) from the lobster group; lobster's code inspection reports; the sample data model and objectbase supplied for testing; testing reports and journals from other groups on lobster's code; email between the students and instructor; and the class newsgroup archive. One could also imagine talk/chat transcripts, and video and/or audio meeting records.

Simply placing these materials on-line is inadequate. Manually creating and maintaining cross-references as the work continues and documents change would be an enormous undertaking. Instead, *useful* hyperlinks

[3] The student groups invent their own names, e.g., now during the Spring 1997 course one group calls themselves "three stooges".

should be automatically generated when possible, and be automatically added as the materials undergo further evolution. Environment users should be able to introduce hyperlinks incrementally as they discover *useful* relationships, either manually or through automated agents. We emphasize *useful*, because simply indexing or cross-referencing every appearance of a given word or phrase would result in a densely linked mass of little value. Instead, we advocate a "loose" schema categorizing and organizing entities of interest, process modeling to indicate semantic dependencies, and workflow automation to automatically add/remove hyperlinks as tasks are completed, as key value-adding services.

1.2 Motivating Scenario 2

Imagine a situation in which a company, XYZ Industries, follows a sprocket design process that involves mathematical simulations using a rather expensive software package running on a Cray supercomputer. Related CAD work is done on SGI UNIX workstations running AutoCad. XYZ has standardized on the UNIX version of Adobe's FrameMaker for all their internal documentation, and they use a mix of the UNIX and Windows versions of Adobe Photoshop for creating illustrations for the documentation. Microsoft Project is used for scheduling projects, including tasks like producing and updating Gantt charts of project milestones and deadlines. All the engineers at the company require access to all these tools in order to complete their work.

Unfortunately, XYZ Industries has let things get a little out of hand. Every engineer sits in front of either a SGI workstation or a Windows PC. Problems occur because all the users require access to all the tools, no matter what kind of computer they happen to be sitting in front of. Training is a nightmare, because Windows PC users do not want to learn UNIX commands, and UNIX users don't want to be forced to borrow a Windows PC whenever they need a tool from that environment. Giving each engineer both a UNIX workstation and a PC on their desks is prohibitively expensive.

In addition, complicating things even further, is the need to share tools among multiple sites on the XYZ corporate wide area network (WAN). Engineers in Palo Alto and New York must access the Cray numerical analysis tools, and the Cray happens to reside at the Chicago office. Everyone needs access to AutoCad running on the powerful SGI Onyx in the Palo Alto office. XYZ can afford the relatively inexpensive telecommunications links between these three cities, but they cannot afford to duplicate the extremely expensive Cray and associated software at every corporate site. Figure 1 illustrates this hypothetical WAN/LAN setup.

WWW technology provides a unique platform for distributed tool services, whether throughout the Internet or within an organizational intranet (i.e., WAN) separated from the Internet by firewalls. The HTTP protocol includes facilities that allow a client to request something from a server (the HTTP GET method). This request format can be generalized to allow a client to request a tool from a tool server "masquerading" as a Web server. The client, in this case, can be either a standard unmodified WWW browser (including Netscape Navigator, Microsoft Internet Explorer, or NCSA Mosaic) or any application capable of making an HTTP GET request.

2 APPROACH

2.1 Subwebs

A *subweb* organizes on-line materials of plausible interest to a project or organization, such as in our first and second motivating scenarios, respectively, over its life-time. A subweb supports structured associative and navigational queries, and unstructured information retrieval and hyperlink following. Some materials may be updated for unrelated purposes, so it is not always appropriate to copy everything to a project-specific repository: some documents should continue to reside at their original homes but be treated as part of the

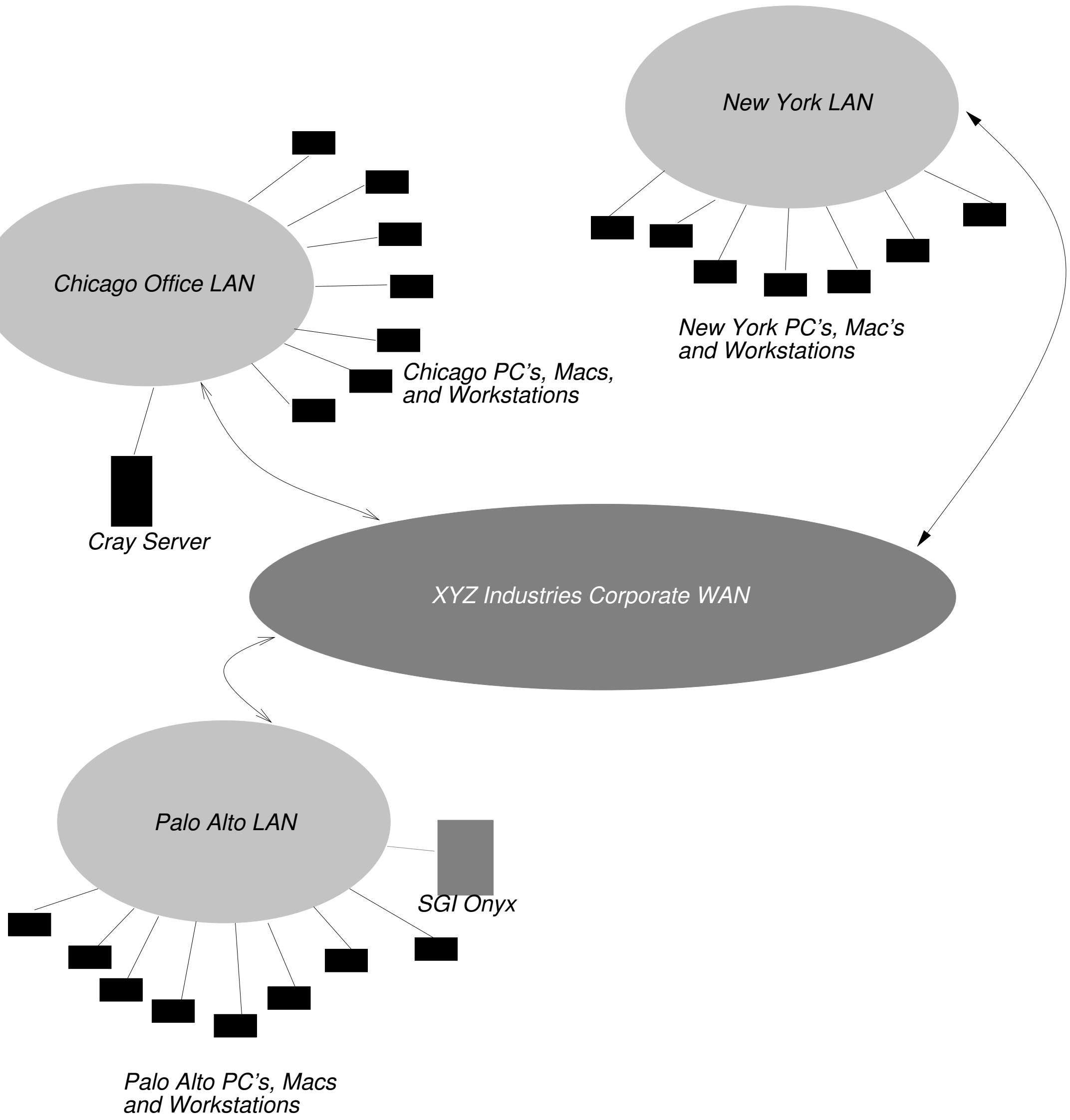

Figure 1: XYZ Corporate WAN

```
WebObj :: superclass ENTITY;
    URL : string;
    type : (Reference, Copy, Updatable)
        = Reference;
    content : binary = ".html";
    updateinfo : text = ".updinfo";
    my_links : set_of link ENTITY;
    my_children : set_of ENTITY
end
```

Figure 2: WebObj Superclass

project's information base, perhaps without the authors' knowledge (for publicly available materials), and possibly with external hyperlinks added on, constituting *two-level hypertext*.

Subweb organization follows what we call a "loose" schema because it specifies the existence, names and types of possible composition and referential links imposed on top of the materials, as well as other attributes, but does **not** define or affect any hyperlinks that may be embedded in their content. In particular, the schema may introduce *external* links among documents, images, etc. that are unknown (and perhaps irrelevant) to the authors of those materials. Thus a subweb server is a form of *link server* with an object-oriented database veneer, in which those objects that are instances of, say, the `WebObj` class correspond to WWW URLs. Note URLs need not refer to full pages or images, but may correspond to a point within a document, e.g., using the fragment name feature of HTML (`<A HREF="...#<name>"` and `<A NAME="...">`).

Figure 2 shows a sample `WebObj` class. Its attributes represent the WWW anchor, i.e., a fully-qualified canonical `URL`, as an ASCII string (following the HTTP standard to retain compatibility with arbitrary backend services); the access `type`, which defaults to `Reference`; cache the hypermedia entity `content` in an opaque binary file; and store related information such as timestamp (from website) and location (if moved) in the `updateinfo` text file. The `my_links` and `my_children` attributes allow arbitrary references and composition of objects, in addition to any typed link and composite attributes that might be defined by subclasses inheriting from `WebObj`.

The *Reference* access type refers in virtual form to whatever entity is found by accessing the anchor. The entity might be changed "out from under" the subweb, or become inaccessible at any time (although the `content` field is used as a cache). Repeated accesses (e.g., HTTP conditional GET) might not retrieve the identical document. All accesses via the subweb are read-only. *Updatable* is the same as reference, except that subweb users can modify the entity (e.g., using HTTP PUT). The subweb may "own" the website, for storing project-specific materials, or an agreement might be made between website and subweb administrators. Process constraints are employed, so not necessary *all* users of a given subweb can make updates, perhaps only specially privileged users, or user roles, and then only when particular prerequisites are satisfied.

Copy refers to a subweb-resident copy, made when the web object is first instantiated, or re-instantiated on demand later on (e.g., a Web crawler might periodically check for modifications and notify users). The copy can be modified by subweb users (although a particular environment instance might add a "read-only" attribute enforced by process constraints), but is **not** intended as a write-through cache: changes to and new versions of the copy do not in any way impact the original. If the process depends on controlling all updating (or deleting) of a particular web page, i.e., changes to that web page "out from under" the process engine as above are not acceptable, that page should be designated as `Copy` — or, alternatively, the website directory hierarchy should be protected so that it can be modified only via the subweb server.

In addition to schema-based queries, subwebs provide text-matching search analogous to Yahoo, Lycos, AltaVista, etc. But search is *restricted* to only those hypermedia documents represented by a selected set of web objects (perhaps the entire subweb), or a subset of the documents reachable via embedded links from the web objects (e.g., on the same website as the originating web object). Current Web search engines can restrict to or exclude a particular website and/or subdirectory hierarchy, but cannot consider an externally imposed application-specific collection of Web entities arbitrarily strewn over multiple websites. In practice, netsearch often presents many screenfuls of seemingly random materials, whereas subweb search will likely to result in a much smaller and more precise retrieval.

The user may write the entity, e.g., using an HTML editor or tool output, only if the target web object is `Copy` or `Updatable`. For `Copy`, the local copy in the web object `content` attribute is replaced (or a new version created); for `Updatable`, both the `content` cache and the original are modified (e.g., via HTTP PUT). If the URL points to a directory or otherwise results in generation of an HTML page, that page is treated as the `content`, but writing may not be meaningful.

WWW supports *forms* whereby the user enters input to arbitrary backend programs, to be transmitted and processed via the standard Common Gateway Interface (CGI) protocol [14] (i.e., HTTP POST). There are several user interaction models: The user might intend the web object to represent the blank form (to submit arbitrary queries later), or the filled-in form (effectively a particular query that might be resubmitted later with different results). Although it would correspond to a different URL, the user might have in mind the output of submitting the form with particular inputs (the result of a previous query). Each of these models incurs different read/write implications. Selection among models can be done in several ways, including hardwiring a particular choice into a given subweb server, defining a subclass of `WebObj` corresponding to each of the choices, adding an attribute whose value determines the choice for that particular web object instance, or prompting the user when the choice is to be made during execution.

2.2 Groupspaces

To construct fully functional collaboration environments on top of hypermedia subwebs, we add what we call *groupspace* services consisting of multi-participant process automation, concurrency control and failure recovery for collaborative work, and tool management technologies. We allow for addition of other services, such as knowledge-based search/classification agents, content-based security mechanisms and general access control, not explored here. Some or all groupspace services could be omitted in a given implementation, and subwebs would still be useful for organizing materials. We choose the term "groupspace" because we think of each affected Web browser as the user's personal workspace, and collaboration support through these services turns the relevant portion of the Web into a cooperative forum.

Any process/workflow paradigm (Petri nets, rules, task graphs, etc. [17, 24]) can be adapted to hypermedia. A process model includes partially ordered tasks, allowing for alternatives and iteration. Tasks usually have parameters involving process state and/or product artifacts, actions that may involve invocation of external tools, may form a hierarchy of subtasks, may have prerequisites and consequences, and may specify resource needs. Parallel work by human participants may be synchronized.

Once a subweb mechanism is in place, process modeling and analysis needs relatively little extension to work with hypermedia. We identify the three main concerns of such extension: treating hypermedia accesses as tasks on which process constraints may be applied, treating hypermedia entities as arguments to conventionally defined tasks, and reflecting dependencies among data items as hyperlinks.

In addition to whatever task definition mechanism is already provided, the process notation should support *read* (HTTP GET) and *write* (HTTP PUT) of a hypermedia entity, *follow* of an external hyperlink (*follow* might not be distinguished from *read* where the user enters the desired URL explicitly, since both correspond to HTTP GET), and *link/unlink* of an external hyperlink. These operations then could be used in the prerequisites and consequences of tasks, and their results serve as the arguments to task actions. Further, these operations should be treatable as primitive tasks in themselves, i.e., the operations are themselves the actions, with process-specific prerequisites and consequences — generally with different prerequisites and consequences for different subclasses of `WebObj`.

Then the normal task aggregation facility can be used to support composite hypermedia operations, such as reading or writing a compound document consisting of multiple entities. And partial ordering, synchronization, prerequisites and consequences, etc. can be applied to these operations in the same manner as for other tasks defined in the process system's native notation. For example, prerequisites must be satisfied before a browser may successfully retrieve (HTTP GET) or update (HTTP PUT) the WWW entity associated with a web object. No prerequisites are enforced when accessing URLs (e.g., a friend's home page or www.olympic.att.com) outside the subweb. This extension to process concepts may not involve significant changes, if the original language already includes *read* of a project datum, etc. In a hypermedia collaboration environment, these operations apply to web objects as objects, and to their underlying hypermedia entities, as well as to other data. An ideal subweb implementation would make this distinction completely transparent.

The process definition notation should also include facilities for specifying those dependencies among entities where composite and/or referential links are required to be explicitly placed under human supervision *versus* those links that could/should be inferred and implicitly placed via process automation. For instance, a task prerequisite may require that a particular link exist between two of its parameters, or an implicit parameter might be derived by following a specified link (whose absence might result in failure to satisfy the prerequisite, depending on the logic written by the process designer). Or a task consequence may introduce or remove a link between its inputs (parameters and derived parameters) and outputs, including parameters derived via associative queries or information retrieval.

Such prerequisites and consequences could be defined manually by the process designer, to carefully determine the cases where hyperlinks should be placed by a cognizant user *versus* automatically inserted. Or the process model might be mechanically transformed according to template(s) constructed by the process designer or provided by the collaboration environment framework, although freewheeling automation may result in a dense and useless mass of cross-links. This extension to process concepts may not involve any syntactic or semantic changes, if the original language already includes such support (operating on its native data repository). Again, the subweb should hide the distinction between web objects and native data from the process engine.

Enactment of the groupspace-specific process, and visualization of the process while it is in progress, requires that all Web browser requests be *intercepted* so that prerequisites, consequences, synchronization, etc. are enforced and automated, tasks and task segments operating on WWW entities are automatically initiated and controlled, users are notified when tasks they are supposed to do become enabled or tasks that affect them are completed, and visualization display(s) are updated as the process unfolds. Different process systems may not provide all these capabilities, e.g., some process systems attempt to automate satisfaction of prerequisites and assertion of consequences and thus might introduce appropriate hyperlinks without human intervention, whereas others support more limited workflow that only prompts the human to do the work. For all these functions, interception of all HTTP traffic emanating from a participating Web browser is mandatory; thus our choice of a proxy server-based approach.

If the process formalism is adapted to cover changes to hypermedia *content*, particularly `Reference` and `Updatable` web objects that change independently of the subweb, then a polling scheme is needed, e.g., a periodic Web crawler that notifies affected subwebs (we cannot assume a notification capability by the website itself, since subwebs and groupspaces must work with *all* HTTP servers). Such notification must trigger proper process enactment, perhaps without an attending human user. Some process systems may already handle external changes (outside their control) to their data repositories or other "off-line" invocation.

Process measurement and evolution do not seem particularly complicated by hypermedia — except that the underlying entities may be manipulated outside the subweb mechanism. Thus it is virtually impossible for groupspaces to maintain fully accurate statistics or guarantee that any process state embedded in WWW components change only through controlled evolution. This is more insidious that the inherent problem where a user becomes "root" and arbitrarily manipulate a project repository through operating system facilities, because WWW-based hypermedia has no central authority.

Groupspaces do not require any particular transaction model for collaborative work, but there must be some model — preferably one that takes process semantics into account to permit appropriate serializability conflicts (e.g., when using groupware tools) and avoid inappropriate rollback of partial changes interrupted by failures (e.g., when human labor *versus* regeneratable tool output is at stake). To enforce that two groupspace users are not updating the same Web entity or related entities at the same time, or over writing each others' changes when transaction properties disallow it, the system must again *intercept* all Web accesses.

Finally, any collaboration environment must support tools, including both commercial off-the-shelf (COTS) tools with no notion of hypermedia and emerging Web-aware tools (e.g., HTML editors and emailers). If Web-aware tools can be invoked directly from the operating system, the subweb implementation should still guarantee that the tool will receive the appropriate view, e.g., the subweb's local copy if the web object is of type `Copy`, as opposed to the original entity from its home website. Otherwise, tools can easily (and unintentionally) subvert groupspace services. If a non-Web tool is invoked outside the groupspace interface, say directly on the underlying file system, it is generally impossible to impose groupspace services or subweb discipline except through access control. Other requirements for a general distributed tool service are discussed in Section 5.

3 ARCHITECTURE

3.1 Subweb Discipline

Figure 3 illustrates our subweb architecture. The key component is the *subweb proxy*, through which all Web traffic is funneled, as for any HTTP proxy. Two distinct subwebs are shown. The subweb on the right contains six web objects, whose URL attributes point to Web pages or images located one at website A, two at website B, one at website C, and two at the subweb's own websites. Subweb users are depicted as WWW browsers. The browsers are configured to transmit all their traffic through the subweb's proxy server. These may be daisy-chained with other proxy servers before and after the subweb proxy, performing other functions, as in OreO's stream transducers [12]. The same subweb could be accessed through multiple subweb proxies. The browsers, proxy server(s), subweb server, subweb website(s) and any other relevant websites may reside on different machines dispersed across the Internet or an intranet. Normally, there would be a much larger number of objects in a populated subweb, perhaps thousands or more, and more users per subweb, both numbers depending on the application.

When one of these browsers sends an HTTP GET request for a URL, the subweb proxy asks its subweb server if that URL corresponds to any web object contained in that subweb. If not, the subweb proxy behaves like a conventional HTTP proxy and contacts the website server indicated in the URL to retrieve the entity, which is sent to the browser in the normal fashion. In our realization, described in Section 4, the proxy server also sends along a second HTML frame with its groupspace logo and some menu items that may be relevant to non-subweb documents, e.g., a command to add the current document to the subweb (and determine its canonical URL, since it might have been referenced relative to some other document). Note other user interface models are possible. The proxy server and subweb lookup add a small overhead to each non-subweb access, negligible compared to network delay, on the same order of 20 milliseconds as for a caching proxy such as Harvest [26].

If the requested URL matches a web object, then *subweb discipline* is imposed, meaning the proxy diverts the request to the subweb server. The subweb server is then responsible for providing the contents of the requested URL to the proxy and hence to the browser. The mechanics of subweb server retrieval depends on the access type: the subweb server contacts the originating website with an HTTP conditional GET for a `Reference` or `Updatable` web object — and updates its local cache in the `content` attribute, but supplies its private copy for `Copy` — also from the `content` attribute. The proxy server tacks on additional materials (groupspace icon, etc. as above) only for those browsers listed in the proxy server's configuration file, to be

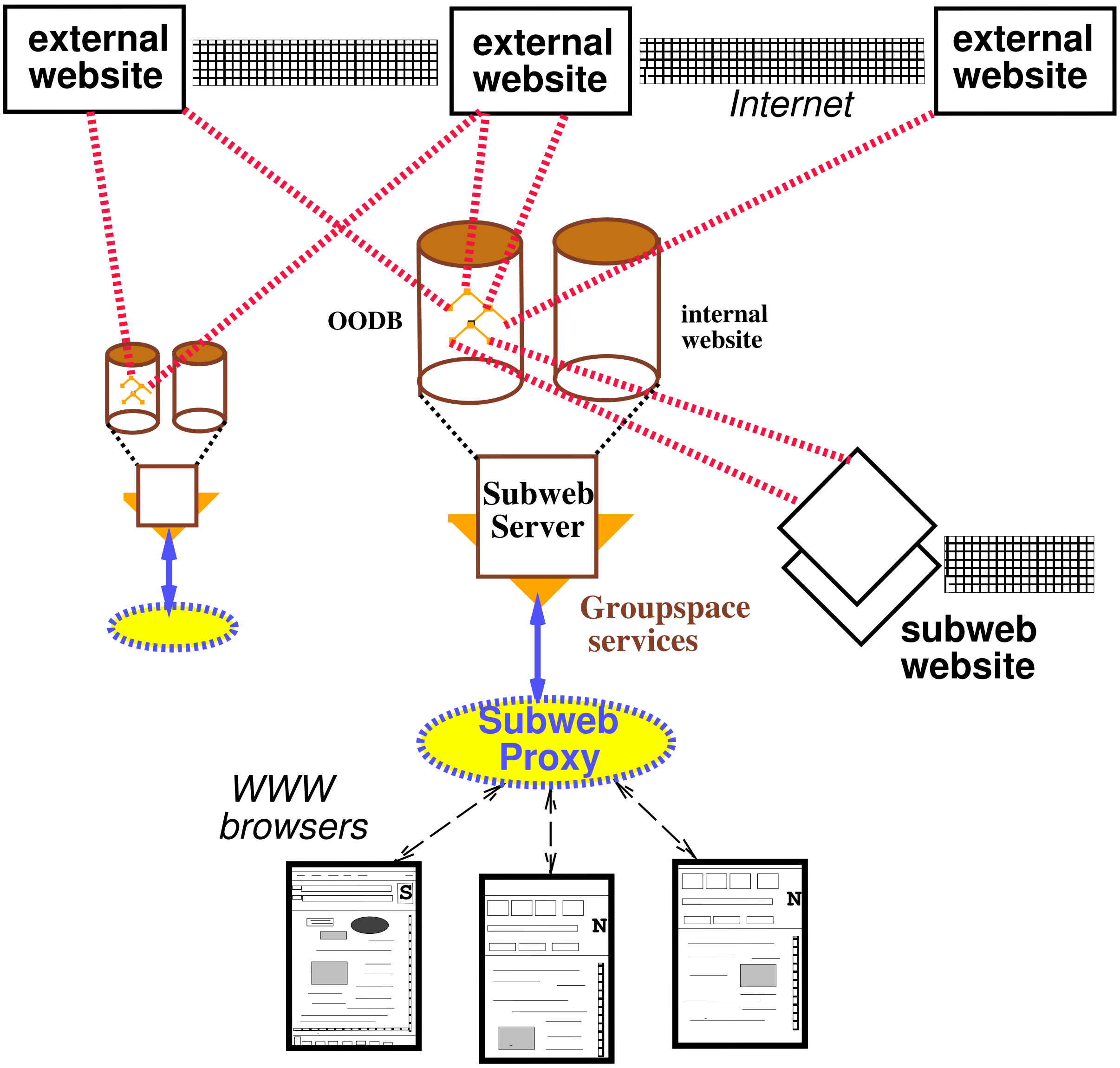

Figure 3: Subweb Architecture

```
Browser sends "GET <URL>" to proxy server.

Proxy server sends "GET /QueryAttr/<URL>"
to subweb server.

If the subweb contains the <URL>
    Then the subweb server sends "HTTP/1.0 200
    OK\nContent-type: text/html\n\n
    <content>" to proxy server.

    Proxy server sends the same thing on
    to browser.
Else if not
    Then the subweb server sends "HTTP/1.0 404
    Not Found" to proxy server.

    Proxy server performs standard "GET <URL>".
End if
```

Figure 4: Subweb Proxy Server Protocol

matched against the standard HTTP `User-Agent` field; the point is to support arbitrary Web-aware tools, which would not be listed unless they are capable of handling add-ons in a reasonable fashion, to operate within the subweb discipline. Ignoring our GUI's use of frames, which adds an HTTP Redirect exchange, the basic protocol is shown in Figure 4.

HTTP PUT works similarly. If the associated URL is not resident in the subweb server, the proxy attempts to PUT directly to the appropriate website server. That server may of course refuse the PUT, if the end-user, or the subweb server acting on his/her behalf, is not properly authorized, in which case an error message is returned to the browser. If the URL is in the subweb, then the PUT and corresponding content is sent to the subweb server. If the web object is of `Reference` access type, the PUT is rejected, since by definition the subweb treats the underlying Web entity as read-only. If the web object is `Updatable`, then the PUT is sent to the website server; if the PUT succeeds, then the local cache, i.e., the web object's `content` attribute, is also updated. Finally, if the web object is of type `Copy`, then only the local `content` is changed and there is no interaction with the website associated with the URL.

3.2 Imposition of Groupspace Services

At the subweb server, as an extension of subweb discipline, the retrieval (or update) is sandwiched between applicable groupspace services. For example, process and concurrency control constraints might be applied first to determine whether or not the `read` (or `write`) operation can succeed at this time and by this user in the current context. The operation might be rejected if the user does not have proper authorization with respect to the given web object, if the process prerequisites have not been satisfied (and cannot be automatically satisfied through process automation), or if there is a concurrency control conflict with another user who is concurrently accessing the same web object as part of a transaction.

In the case of HTTP GET, the subweb server performs process-specific operations indicating the consequences of the `read` immediately after sending the document to the subweb proxy (and hence to the user's browser). This may trigger process automation to fulfill the implications. Note this is different from the usual process enactment model, where the effects are asserted only *after* the entire task has been completed: It is impossible to determine in the Web browser case when the user is "done" with his/her reading. Although another icon could be added to the document display, which the user would click to indicate completion, this cannot be taken as proof that the user will not return to reading the document later on without notifying the subweb server since the document could be stored in the browser's own cache and access via the "back" button. There is a similar issue regarding concurrency control, since arbitrary browsers do not signal when reading is "done".

A related problem arises with HTTP PUT, because by definition the `write` task has been completed at the time of the PUT operation. If the PUT is denied for process or concurrency control reasons, the user's efforts may not have been totally wasted: generally the modified content is still available in the user's browser/editor and the PUT can be retried later. Of course, the PUT can be denied at the underlying website server because the end-user does not have proper authorization, independent of subweb discipline or groupspace services. The general problem of client/server transactions, and proposed solutions appropriate to process/workflow, is the main focus of other work, e.g., see [51], and not addressed further here.

The architecture and realization of our tool service are independent of (although complementary to) subwebs/groupspaces and their implementation in **OzWeb**, and are presented later in Section 5.

4 OZWEB REALIZATION

4.1 Oz Background

OzWeb reuses the process modeling language and enactment engine, the transaction manager, and the object-oriented database management system developed for the Oz process-centered environment [6, 48]. Oz provides a rule-based process notation in which a rule generally corresponds to a workflow step. Each rule specifies the step's name as it would appear in a user menu or agenda; typed parameters and bindings of local variables from the project objectbase; a condition to be satisfied before initiating the step's activity; the tool invocation script and arguments for the activity; and a set of effects, one of which asserts the actual results of completing the activity. The return value of the tool script selects the appropriate effect. Built-in operations like `add`, `delete`, etc. are modeled as standard rules, and can be overloaded based on the parameter type, e.g., to introduce type-specific conditions and effects on those operations; built-in operations can also be used in the effects of arbitrary rules.

Oz enforces that rule conditions are satisfied, and automates the process via forward and backward chaining. When a user requests to perform a step whose condition is not currently satisfied, the process engine backward chains to execute other rules whose effect may satisfy the condition; if all possibilities are exhausted, the user is informed that the chosen step cannot be enacted at this time. When a rule completes, its asserted effect may trigger automatic enactment of other rules whose condition has become satisfied. Users usually control the process by selecting rules representing entry points into composite tasks consisting of one main rule and a small number of auxiliary rules (reached via chaining), but it is possible to define complete workflows in either goal-driven (backward chaining) or event-driven (forward chaining) fashion.

Oz employs a client/server architecture. Clients provide one of several user interfaces (XView, Motif and tty) and invoke tool scripts, which in turn fork and manage external tools as described in [25, 55]. Servers context-switch among multiple clients, and include the process engine, transaction manager, and database. An Oz environment usually consists of several servers, each with its own data schema, objectbase, process model and tool scripts. Clients are always connected to one "local" server, and may also open and close connections to "remote" servers in the same configuration. Servers communicate among themselves to establish and operate *alliances* supporting process interoperability and limited data and task sharing across processes.

4.2 OzWeb Implementation

OzWeb extends the Oz server to operate as a subweb server — and also as a general-purpose HTTP server (ftp, gopher, etc. protocols are not supported). The subweb proxy is a client of the **OzWeb** server, communicating with it via HTTP. Native Oz clients continue to work with the **OzWeb** server, and are useful for debugging. The user browsers, proxy server(s), subweb server, subweb website(s) and any other websites represented in a subweb may all reside on different machines dispersed across the Internet, but could also reside on the same host. **OzWeb** subweb and proxy servers run on Solaris 2.5+. Of course, browser clients of the subweb proxy run on any platform where WWW browsers are supported.

When the proxy asks the subweb server whether or not a given URL is represented in the subweb, an objectbase query is formulated to search for an instance of `WebObj`, or one of its subclasses, with that URL. Although "built in" to the subweb server, the `WebObj` implementation is formulated as a separate layer from the OODB, so that another database system could potentially be substituted. (We are working on other such layers for CORBA objects [42], Chimera objects [2], etc., to apply subweb discipline, groupspace services and, eventually, alliances to backend servers other than website servers and the native Oz objectbase; we also plan to add frontend access protocols for CORBA clients, Chimera clients, etc.)

Oz's process engine was extended with *read* and *write* operations that can be overloaded by process-specific rules, thus adding prerequisites and consequences to these primitive tasks. The subweb mechanism transparently supplies or modifies the requested file attribute for all objectbase objects. In the case of a web object and its `content` attribute, the retrieval or update proceeds as in Section 3.1; only the objectbase's file attribute is affected in the case of native Oz objects or other file attributes of web objects. `read` and `write` could alternatively have been implemented purely as process-specific rules, rather than built-in operations, but this approach permits us to use them in the effects of arbitrary rules and is more efficient. Oz already provided built-in *link* and *unlink* operations that can similarly be overloaded as well as used in rule effects. There is no distinction between how *link* and *unlink* are applied to web objects compared to native Oz objects; in particular, the external links are represented in the subweb objectbase and do not in any way affect any HTML links embedded in WWW entity content.

The distinction between `WebObj` instances and native Oz objects is completely transparent to the process engine, transaction manager and underlying OODB. The process engine was modified only to support primitive `read` and `write` operations on all types of objects. No changes were made in either the transaction manager or OODB, which leads to a "loophole" in both concurrency control and failure recovery: locks to enforce serializability, and undo logs used to guarantee failure atomicity, apply <u>only</u> to the web object representation inside the subweb server's objectbase. There is nothing preventing other users, not employing the subweb's proxy, to arbitrarily access and modify a website directory hierarchy directly if permitted by that file system's protection mechanism. This issue is addressed further in Section 6.

The standard Oz commands were augmented to include a *search* command. When applied to the `contents` attribute of a web object, an implicit `read` operation is performed (perhaps updating the local cache in the case of `Reference` and `Updatable` types). If the underlying WWW entity consists of HTML or other ASCII text, an internal index is updated using the Glimpse utility [37]. Then the actual search is performed on the appropriate subset of the index corresponding to those web objects of interest, that is, more than one web object can be specified as arguments to the same search. The same search mechanism also applies to the `text` file attributes of all Oz objects.

When a registered browser is sent any WWW entity (whether in the subweb or not) by the proxy server, an additional HTML frame is sent to display (at the top of the browser window) an **OzWeb** panel with an icon to select the **OzWeb** control screen. When web object `content` or any object's file attribute is read from a subweb, the subweb server also sends to the proxy server this icon, plus the values of primitive attributes defined by the object's class and its superclasses; HTML links corresponding to each file, composite and reference attribute (multiple links for `set` values); and additional subweb-specific materials, e.g., there might be a "message of the day" to all subweb users plus user-specific notices.

Selection of the control screen icon brings up another browser window giving the full Action Menu and an objectbase display formatted in HTML. The objectbase is represented as a two-column table with a list of the names of objects at the "current" level on the left hand side and the names of the children of the currently selected object on the right hand side, all represented as HTML links. Selecting a child moves it (and its siblings) to the left, with the selected child now the "current" object, and shows its own children on the right; there is also a link for moving up in the objectbase composition hierarchy. The values of primitive attributes and links representing file, composite and reference attribute of the "current" object are shown below this table. Selecting any link in the Action Menu brings up browser windows for entering arguments to the corresponding command. When an argument may be an object, a browsing table similar to the objectbase display is shown; textual entry (e.g., for *search* patterns) is also supported. Several sample screenshots are shown in Section 6.1.

The objectbase display and browsing tables contain URLs referring to the **OzWeb** subweb server rather than some conventional website. For example, the current position in the objectbase is encoded in the URL transmitted to the browser in the form `"http://\-<website>/ObjectView/OID"`. So the URL that the browser sees may not be the same as the entity's home URL, i.e., the `URL` field of the web object with the given OID (Object IDentifier). Selecting an HTML link from an objectbase table thus works slightly differently than for access through standard WWW browser windows: the **OzWeb** server transforms the URL and uses HTTP Redirect to force the browser to re-request the entity from its proper location; subweb lookup is not needed, since objectbase links always refer to subweb materials.

5 TOOL SERVICE

5.1 Requirements

We identify a number of requirements for a distributed tool server system:

- Run tools on multiple platforms
  Users should be able to request tools that run on a range of machine architectures and operating systems. Users should not have to remember, for example, that the Cray software is in Chicago and the associated network commands to access it. They should be able to ask any one of a set of "well known" tool servers for the tool, and each tool server should be responsible for knowing either the details on how to start the tool itself, or to which of its peer tool servers to forward the request.

  Although it may seem that an organization might need a large number of tool servers running, say if there is one per host where tools reside, each uses relatively few system resources compared to many users logging into the system in order to run tools. In addition, if resources are scarce and the tool server is needed infrequently, it could be started automatically upon a user request by a lightweight system daemon that starts up when a user tries to connect to its TCP request port.

- No local tools *required* on client hosts
  It should be possible to request all tools from the tool server. The tool server may be able to exploit locally available applications (i.e., software installed on the user's computer), but it must be able to make remote tools available via display redirection facilities like those of X Windows. This minimizes user and system administrator burden, since it is then unnecessary for popular packages to be installed on every computer.

- Run tools on the user's own UNIX workstation, PC, or Macintosh *when possible*
  A tool server should be able to decide that a given tool can and should be run locally on the client user's machine. This is intended as an optimization, since it's often more efficient (in terms of network bandwidth) to run a tool locally than to run a tool remotely and redirect its GUI using a remote display facility like X Windows. However, in the case of data-intensive tools, it might be preferable to run the tool on the same host where that data resides (which might be a shared file server rather than the user's machine).

- Wrap tools and sequences of tools
It should be possible to "wrap" sequences of tools and have the wrapper appear to the user as a single activity that they are able to request from the tool server. This is useful for tools whose input and/or output must be converted between file formats. Wrappers can also handle startup configurations, including moving all relevant data files to a temporary directory, setting environment variables, etc.

- Easy to use
Users should be able to connect to a tool server to receive a list of available tools or activities, without the creation of a heavyweight user interface that would need to be ported to all computing platforms in use and learned by all users.

- Easy integration with existing systems
A tool service should be easy to integrate with existing systems that might benefit from a tool launching facility, such as CAD/CAM, software development environments, and workflow or medical care plan automation systems in addition to simple menu-based systems that present a list of potential tasks to the user. This requires the tool server to provide an Application Programming Interface (API) of some sort for program-to-program requests and responses without user intervention.

5.2 Component Architecture

We have devised an architecture that combines a collection of cooperating tool servers running on a variety of "server" computers with a "personal tool services" component that runs on each user's own machine. We use the phrase **tool server** to denote a remote tool server running on a server computer, and the phrase **personal tool server** (or PTS) to refer to a tool server that runs tools on a user's behalf directly on his/her local UNIX workstation, PC or Macintosh.

The tool server is modeled as an HTTP server, and uses HTTP as the communication method not only between itself and clients, but also between peer tool servers. HTTP (as opposed to, say, RPC or CORBA) was selected because it is a simple TCP/IP-based request/response protocol, making it easy to "graft" onto existing systems (possibly using toolkits like **libWWW** [16] from the World Wide Web Consortium or **ASHeS** [47], the Application Specific HTTP Services toolkit from our research group). The HTTP specification provides methods for a client to request a document (HTTP GET) and send data (HTTP PUT) to a server. In our architecture, the URL is treated as a request for a tool.

Users connect to tool servers via standard Web browsers. When a tool execution is desired, the user, or an existing system that needs to run a tool on a user's behalf, simply requests an appropriate URL from a tool server. Rather than returning a document, as would a standard website server, the tool server invokes the tool (or a tool wrapper), perhaps by employing a peer tool server or personal tool server. Any return code or output from a terminating tool is sent to the user's Web browser.

When a tool server starts up (e.g., during system boot of a server machine), it contacts other tool servers to exchange information. These peers may be specified in a configuration file or may be inquired from a directory service. By storing information on what tools each known remote tool server is capable of running, it is possible for tool servers to handle requests for tools it may not be able to run directly. When a request for tool execution arrives from a client, the tool server can simply relay the request to a peer tool server that knows how to perform the execution. Figure 5 depicts a user requesting a tool from a given tool server, where that tool server then forwards the request to a peer. Note this makes possible a "clearinghouse" approach to tool management. In this scheme, most tool servers are set up to know how to run a specific class of tools on a particular architecture, host, etc. In addition, one or more "clearinghouse" servers are configured to contact the other servers and retrieve their tool definitions. Users can then connect to a clearinghouse server to request tools that actually reside on any one of the other servers.

Implementing tool servers as WWW servers makes it easy to run tools directly on the user's local machine: We employ the Web's standard mechanism for capability augmentation, a new Multipurpose Internet Mail Extensions (MIME [11]) type and corresponding MIME helper application. Every response from an HTTP

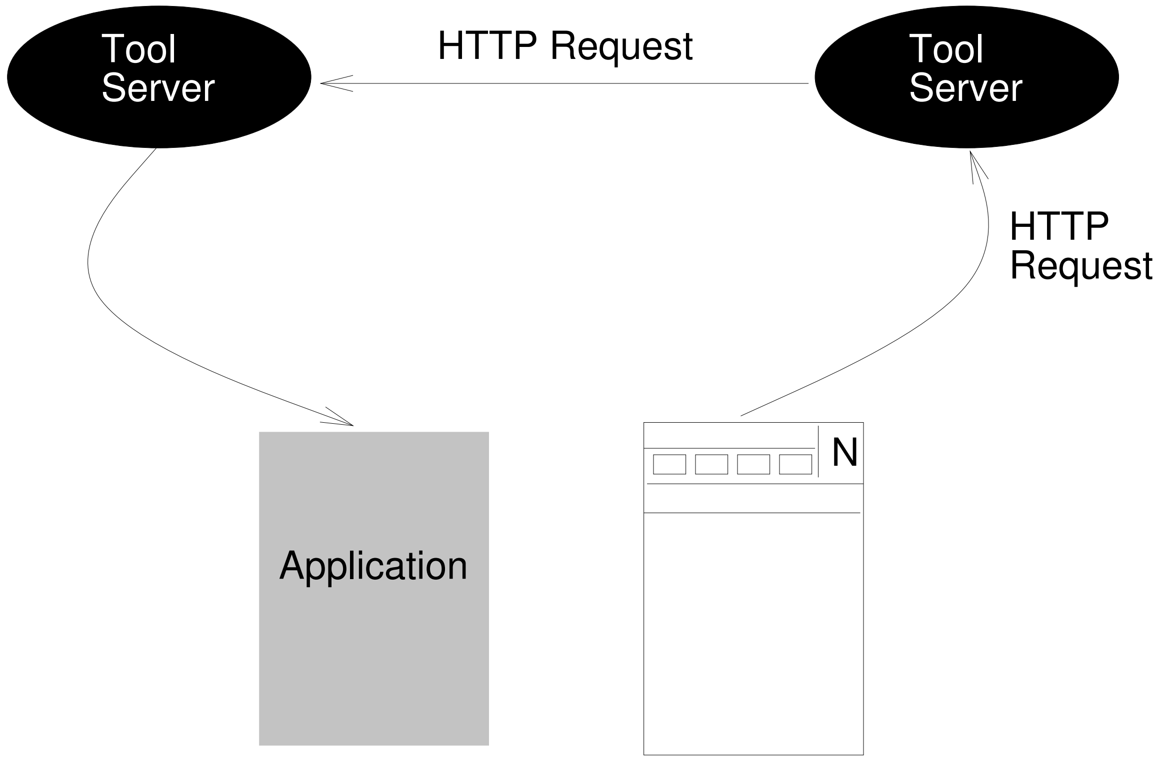

Figure 5: Peer Tool Servers

```
HTTP/1.0 200 OK
Date: Fri, 04 Oct 1996 05:10:34 GMT
Server: Apache/1.1.1
Content-type: text/html
Content-length: 5149
Last-modified: Wed, 14 Aug 1996 15:13:00 GMT

<more data...>
```

Figure 6: Response headers from an HTTP transaction

server is tagged with a MIME type tag. Figure 6 shows an example response and its associated MIME type field. Every Web browser can be configured to start what is known as a **Helper Application** (or app) when it receives data with a certain MIME type. If received data is destined for a helper app, the browser simply saves that data to a temporary file and starts the helper app, giving it the temporary file name as input.

For example, suppose a tool server receives a request from a PC user who wishes to run Microsoft Excel — which happens to be installed locally on that user's computer. The user's system administrator has configured her browser so that receipt of data with the MIME type "application/x-toolserv" causes a MIME helper application to be started on the computer. This helper app is the Personal Tool Server, which is responsible for actually starting the tool, as illustrated in Figure 7. The data from the HTTP server tells the PTS which local application is being requested and the pathname to the executable program. The pathname may point to either the application itself or a wrapper. When the user terminates the tool, the PTS closes down and exits.

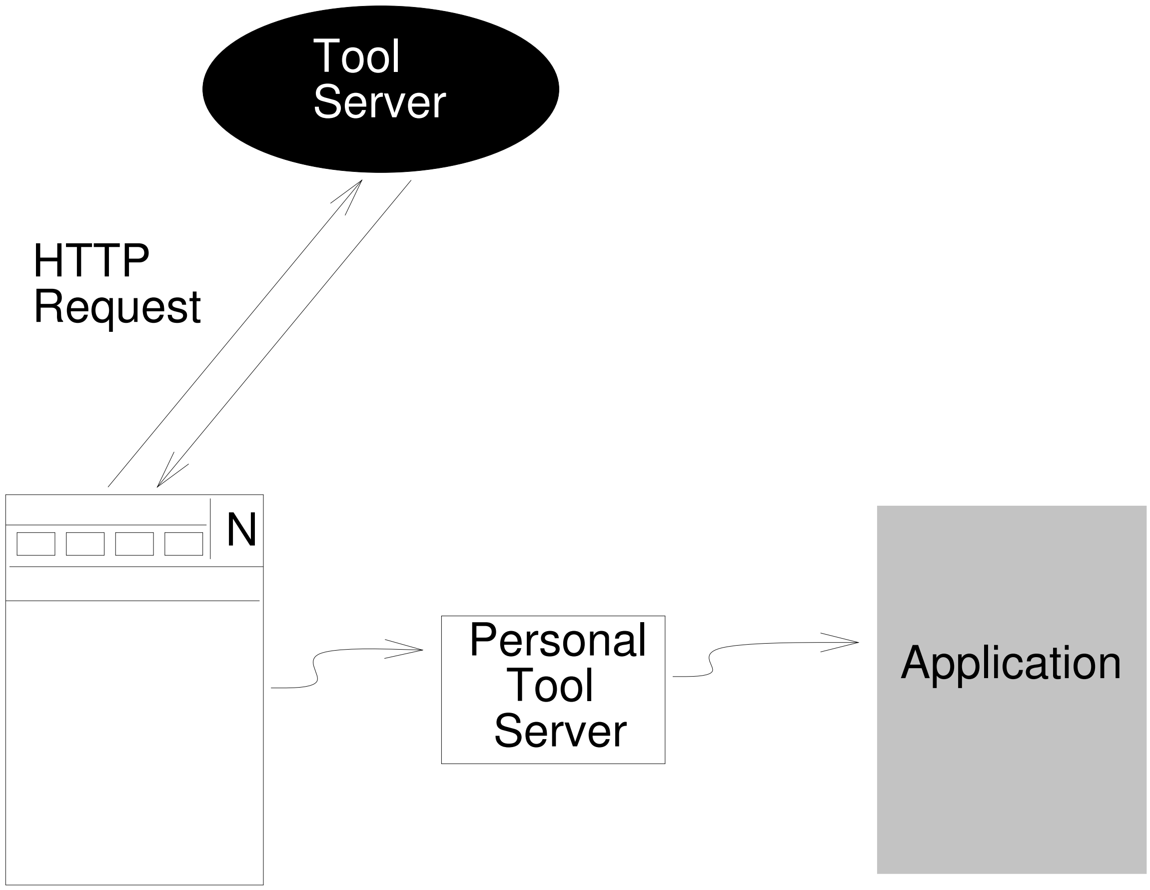

Figure 7: Personal Tool Server

5.3 Activities and Scenarios

For some classes of tools, there may be multiple individual tools (e.g., running on different platforms or hosts) which can accomplish the same task. Imagine Jane, an employee of XYZ Industries, wishes to edit a C source code file. If Jane uses a UNIX workstation, it may be appropriate for the tool server to run the Emacs text editor. If she happens to sit in front of a PC, it may be better to start a Windows tool. In both cases, Jane will accomplish the same task: editing a source file. The user requesting the tool may leave it up to the tool server to determine which tool is more appropriate in a given situation.

Our tool server models **Activities** and **Scenarios** to address such situations. Jane's **Activity** might be called "Edit". The possible **Scenarios** for Edit might be Emacs on Sparc/Solaris and WinEdit on Windows 95. When users request tools from the tool server (see below), they can specify a particular scenario name if they need to run a specific tool from the available choices; otherwise only the activity is specified.

The tool request sent by the client is a standard HTTP GET for a URL of the following form:
`GET /Activity_Name/Scenario_Name/Arch/Host/Parameter_1/.../Parameter_N`

- `Activity_Name` is the only required parameter. It tells the tool server which Activity the user is requesting.
- `Scenario_Name` chooses a particular Scenario. "*" indicates no preference.
- `Arch` allows the user to specify that this Activity must run on a particular computer platform, encoding both architecture and operating system (for instance, "Sparc/Solaris" or "Intel/Solaris"). This can be used to differentiate among Activities that use the same parameters as input but generate different side-effects when executed. For example, a "Compile" activity may have two possibilities defined, one for the HP/UX operating system and another for Solaris. One source code file may be used to generate object code on either platform, which the **Arch** parameter allows to be chosen. Again, "*" may be specified for tools where the platform does not matter.
- `Host` requests that a particular Activity be run on a specific host, overriding the tool server's choice of which host would be the best place to run the selected tool. "*" refers to this default.
- After the `Host` parameter, any other portions of the URL are considered parameters to be passed on the command line when starting the tool (or wrapper).

5.4 Rivendell Realization

Our preliminary implementation of a distributed tool server component is known as **Rivendell** and follows the architecture outlined above, running on Solaris 2.5+, Linux 2.0+ and Windows NT. **Rivendell** acts as a standard HTTP 1.0 server, receiving GET requests of the form specified above to launch tools. **Rivendell** is currently started during system boot, although it would be possible to employ a daemon to start **Rivendell** only when a user request is received.

During startup, a **Rivendell** tool server attempts to read its configuration files, which specify the Activities and Scenarios it knows how to run, as well as the peer tool servers it should contact, and then contacts those peer tool servers — running on other machines on a LAN, across a corporate WAN (or intranet), or even across the Internet. When a peer tool server is contacted, **Rivendell** attempts to download the list of activities and scenarios known by the foreign server. Once this has been accomplished, **Rivendell** performs an HTTP PUT to upload into the foreign server the list of activities and scenarios it has been configured to handle. Figure 8 shows an example **Rivendell** configuration file. The format strongly resembles the tool base information from MTP [55], a previous tool management effort in our lab, which allowed for (among other things) specifying particular hosts where tools had to run. MTP focused on shared groupware tools, one of the extensions we hope to make to **Rivendell** during our future work.

```
Peer: "pearl.psl.cs.columbia.edu:7777";
Peer: "marginal.psl.cs.columbia.edu:7777";

PTS_MIME: "application/x-toolserv";

Tool Editor
 [ Name  : "Emacs";
   Arch  : "Sparc/Solaris";
   Host  : "*";
   Exec  : "/usr/local/gnu/bin/emacs";
   Graphical : T;
 ];
 [ Name  : "WinEdit";
   Arch  : "Intel/Windows";
   Host  : "*";
   Exec  : "\edit\winedit.exe";
   Graphical : T;
 ];

Tool Compiler
 [ Name  : "GCC";
   Arch  : "Sparc/Solaris";
   Host  : "*";
   Exec  : "/usr/local/gnu/bin/gcc";
   Graphical : F;
 ];
 [ Name  : "SparcWorks";
   Arch  : "Sparc/Solaris";
   Host  : "pearl.psl.cs.columbia.edu";
   Exec  : "/opt/SUNWspro/bin/cc";
   Graphical : T;
 ];
```

Figure 8: Sample Rivendell Configuration

- The first portion of the configuration file, the `Peer:` lines, tell this particular **Rivendell** instance how to locate its peer tool servers. These tool servers will be contacted, and configuration information shared with them, at startup time. This **Rivendell** instance has two peers, running on "pearl.psl.cs.columbia.edu" and "marginal.psl.cs.columbia.edu", both on TCP port 7777.
- The `PTS_MIME:` line tells **Rivendell** what MIME type to send to the user's Web browser in order to tell the browser to start the PTS for local tool execution. The browser must have been configured beforehand to start the PTS upon receipt of this MIME type. This configuration is often handled for users by an organization's system administrators.
- Following the MIME type definition are the definitions for the Activities and Scenarios known to this particular **Rivendell** instance. Two Activities are defined, "Editor" and "Compiler", each of which has two scenarios. The Editor Activity defines a Scenario called "Emacs" on Sparc/Solaris, as well as one called "WinEdit" on Intel/Windows.
- The actual pathname to the executable tool, or wrapper, is given in the `Exec` parameter in each Scenario.
- The `Graphical` parameter tells **Rivendell** if the tool has a GUI or is a batch tool.

After the configuration file has been read, **Rivendell** contacts each of its peer tool servers, issuing the following HTTP GET request:
`GET /Rivendell_ACTIVITIES`
The remote tool server is expected to respond with its Activities and Scenarios. The following example illustrates this format, which is the same format as the Activity definitions from the configuration file read at startup.

```
        Tool Project
   [ Name          : "MS Project";
   Arch          : "Intel/Windows";
         Host          : "*";
Exec          : "\msoffice\project\project.exe";
```

```
                                        Graphical     : T;
      ];
```

This response indicates that the contacted tool server defines one local Activity, called "Project". **Rivendell** stores this information, and if a client requests the Project Activity, that request will be forwarded to this Peer Tool Server. Once **Rivendell** has downloaded the Activity and Scenario information from the remote server, it performs an HTTP PUT of the following form:

`PUT /Rivendell_ACTIVITIES [local activity information]`

Rivendell thus sends the Activity definitions from its configuration file to the remote peer. It contacts each peer tool server in this fashion, transferring information. Once this has been completed, **Rivendell** is ready to receive user requests.

A user's tool request is an HTTP GET where the URL specifies the Activity the user wishes to run. Again using an example from XYZ Industries, imagine that Bob needs to run the Emacs text editor. Bob's tool server defines an activity called "Editor", which includes an "Emacs" scenario. The closest tool server to Bob, who is in New York, is running on TCP port 7777 on a machine called "nyserv.xyz.com". In this case, Bob points his Netscape browser at the following URL:

`http://nyserv.xyz.com:7777/Editor/Emacs/*/*/`

When the nyserv tool server receives Bob's request, it first checks that there is an Activity named Editor and a Scenario named Emacs defined. The Activity can be one from nyserv's own configuration file, or one informed by a peer tool server. Once nyserv has determined that the request is valid, it checks whether Bob's UNIX workstation is able to run Emacs (via a personal tool server). The tool server on nyserv has no special knowledge of Bob's machine. Since the Editor Activity is locally defined (i.e., it was not downloaded from a foreign tool server), nyserv moves on to the architecture and hostname requirements, which can be set in an Activity definition. Since these are both "*", this Activity is free of restrictions. Thus, nyserv deems itself the best place to run Emacs for Bob, and does so. X11 display redirection is used to move the Emacs display to Bob's workstation, i.e., the DISPLAY environment variable is set to Bob's workstation before the tool is invoked.

If the Editor Activity from this example had been from a remote peer tool server, nyserv would have forwarded the request to that remote tool server. Bob would not need to know that his request had been forwarded, it would be the responsibility of the tool server on nyserv to do this transparently.

In addition to X11 Windows, **Rivendell** also interoperates with the WinDD software from Tektronix [53]. WinDD allows a user at a UNIX workstation to remotely execute tools on a Windows NT Server machine with the GUI display redirected to the user's machine. WinDD employs special software, running on the UNIX workstation, to contact the Windows NT server. When **Rivendell** runs a tool whose `Arch` configuration parameter is set to Intel/Windows, a WinDD client is started for the user, and WinDD is then instructed to execute the required tool. Figure 9 is a screen shot of WinDD running an application on a Windows NT server and displaying on a UNIX workstation. Similar software, notably NTrigue from Insignia Solutions [29] and WinFrame from Citrix [15] accomplishes the same function as WinDD and could be integrated as well.

5.5 Integration with OzWeb

Introduction of **Rivendell** as a new groupspace service within the **OzWeb** hypermedia collaboration environment framework described in Section 4 required small changes in the **OzWeb** server. In particular, we modified **OzWeb** to pass on all tool invocations to its affiliated **Rivendell** tool server, specified as part of the **OzWeb** instance's configuration. This was done by having **OzWeb** issue an HTTP GET to **Rivendell** each time tool execution was needed (e.g., for a workflow step). The connection to the **Rivendell** server is transparent to the **OzWeb** user — the tool is simply brought up on the user's behalf. In the case of a tool executed by a personal tool server, the only user intervention necessary is the one-time configuration of their Web browser to run the PTS upon receipt of the relevant MIME type; if this is done beforehand by a system administrator, no user intervention is required.

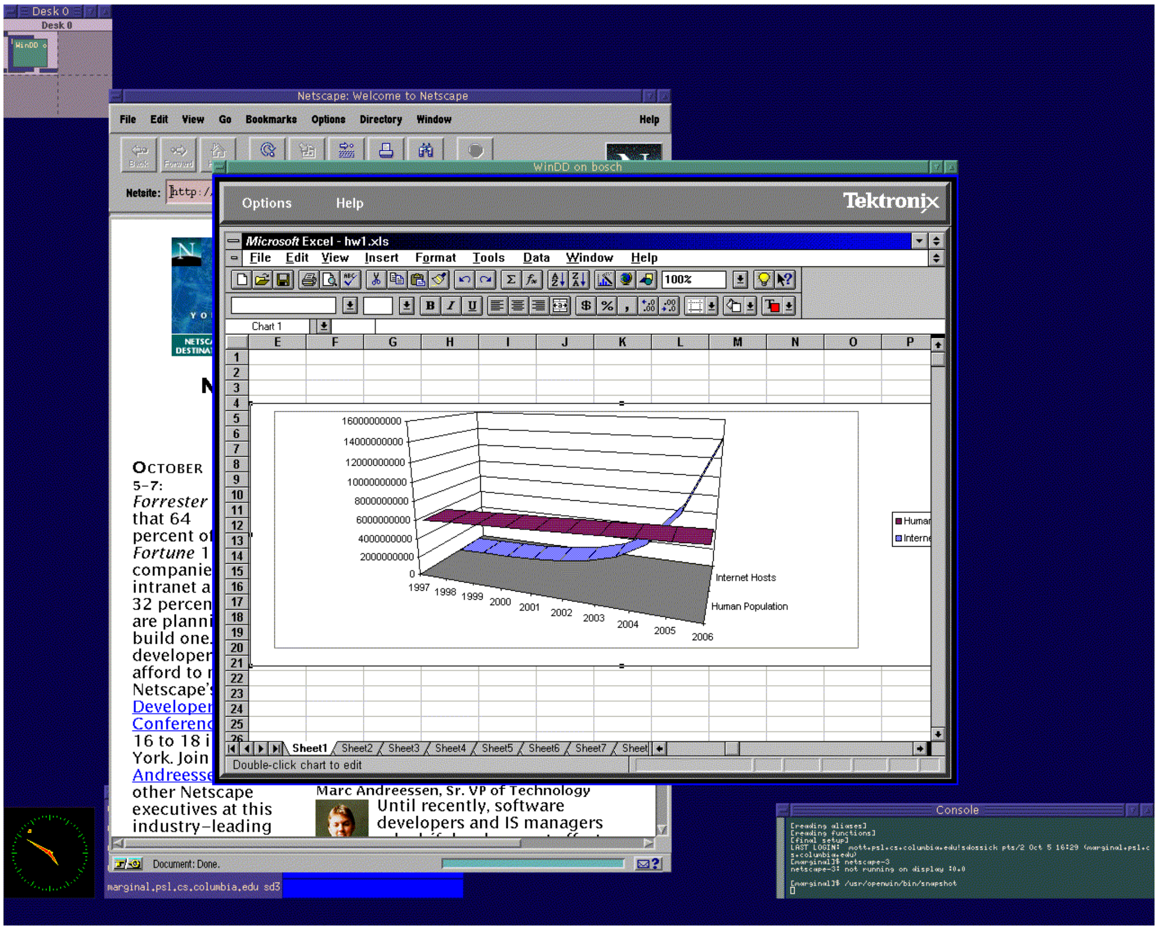

Figure 9: Using WinDD

6 EVALUATION

The implementation presented here is **OzWeb** 0.35, which combines an earlier **OzWeb** 0.2 and **Rivendell**. The most significant limitation of **OzWeb** 0.35 is we ignore any version and configuration management facilities that may be present in the underlying websites from which subweb materials are obtained [56]. A process designer may explicitly include version control of web objects in the process, as was often done in Oz instances, but this affects the web object content for only the `Copy` access type. We are working on an extension to **OzWeb**'s transaction management facilities, as described in [30], to take advantage of any locking, "check out", versioning, etc. facilities in backend website servers to more fully enforce transactional properties.

We previously developed another system called **OzWeb** (effectively version 0), described in [21]. It provided a general-purpose Web-based GUI to the unmodified Oz , analogous to the Web interfaces to some commercial workflow systems, e.g., Action Workflow Metro [1] or Lotus Notes Domino [35] (we separately developed a special-purpose Web-based GUI for a proof-of-concept Oz environment instance supporting medical care plans [34]). The original architecture involved an HTTP proxy server and would work with any HTTP browser, but that is where the similarity ends. There was no hypermedia: none of the entities were in HTML format or included embedded links. There were no subwebs, and thus no ability to incorporate external materials from WWW or elsewhere: all documents resided in the Oz's native objectbase. The browser had to explicitly request Oz functionality via a URL of the form "http://oz/command/argument1/argument2/..."; if the "oz" site was not included in the URL, nothing happened beyond the usual retrieval of the page from its website server. This is an important distinction: our subweb proxy supports automatic application of full groupspace services (process, transactions, tool invocation, etc.) to every URL represented in the corresponding subweb, whether explicitly requested or selected via link selection, whose actual Web page may reside anywhere on the Internet or intranet.

The functionality (beyond Oz itself) was implemented for version 0 in a heavyweight proxy server constructed by modifying an Oz tty client. It used HTTP to communicate with Web browsers, but directly used the peculiar Oz client/server protocol to communicate with the Oz server. In contrast, the main functionality in 0.35 resides in the **OzWeb** subweb server, which communicates with its proxy using HTTP. The **OzWeb** server was implemented by extensive modifications to the old Oz server. The 0.35 proxy alone is very lightweight, the main thing it does differently from a standard HTTP proxy (e.g., used for caching) is to query the **OzWeb** server to see whether or not the requested URL is in the subweb and (optionally) add on presentation of subweb object attributes; otherwise it just does everything the browsers and **OzWeb** server tell it to do via standard HTTP.

We also developed the experimental subweb server described in [59]. We used a rather convoluted implementation involving direct requests for invocation of CGI scripts, rather than a proxy. It was thus impossible to intercept *all* Web accesses as in our current system. No groupspace services were supported, this was purely an organizational facility. We explored "view objects", to describe application-specific reformatting of HTML materials, not yet implemented in **OzWeb**.

There are several limitations of **OzWeb** 0.35 regarding our goals for subwebs and groupspaces. For example, each subweb proxy is currently hardwired to a particular subweb. For an end-user to move to a different groupspace, he/she must designate a different proxy in his/her Web browser configuration. It is not possible for an end-user to be operating in multiple groupspaces at the same time from the same browser. This limitation is easy to remove by connecting each subweb proxy to a directory service, e.g., using the Lightweight Directory Access Protocol (LDAP [60]) — an industry standard that allows for simple queries to be made to X.500 compliant directory servers, through which it can interact with any number of registered subwebs. We have not changed the implementation, however, because of a conceptual complication regarding what should be done when a browser accesses a URL represented in two or more subwebs. This is part of our in-progress research on extending Oz alliances to **OzWeb**, to support interoperability among groupspaces with different process models, data schemas, etc.; see Section 8.

In **OzWeb** 0.2, all tools were invoked via scripts forked by the subweb proxy. A tool's GUI was redirected to

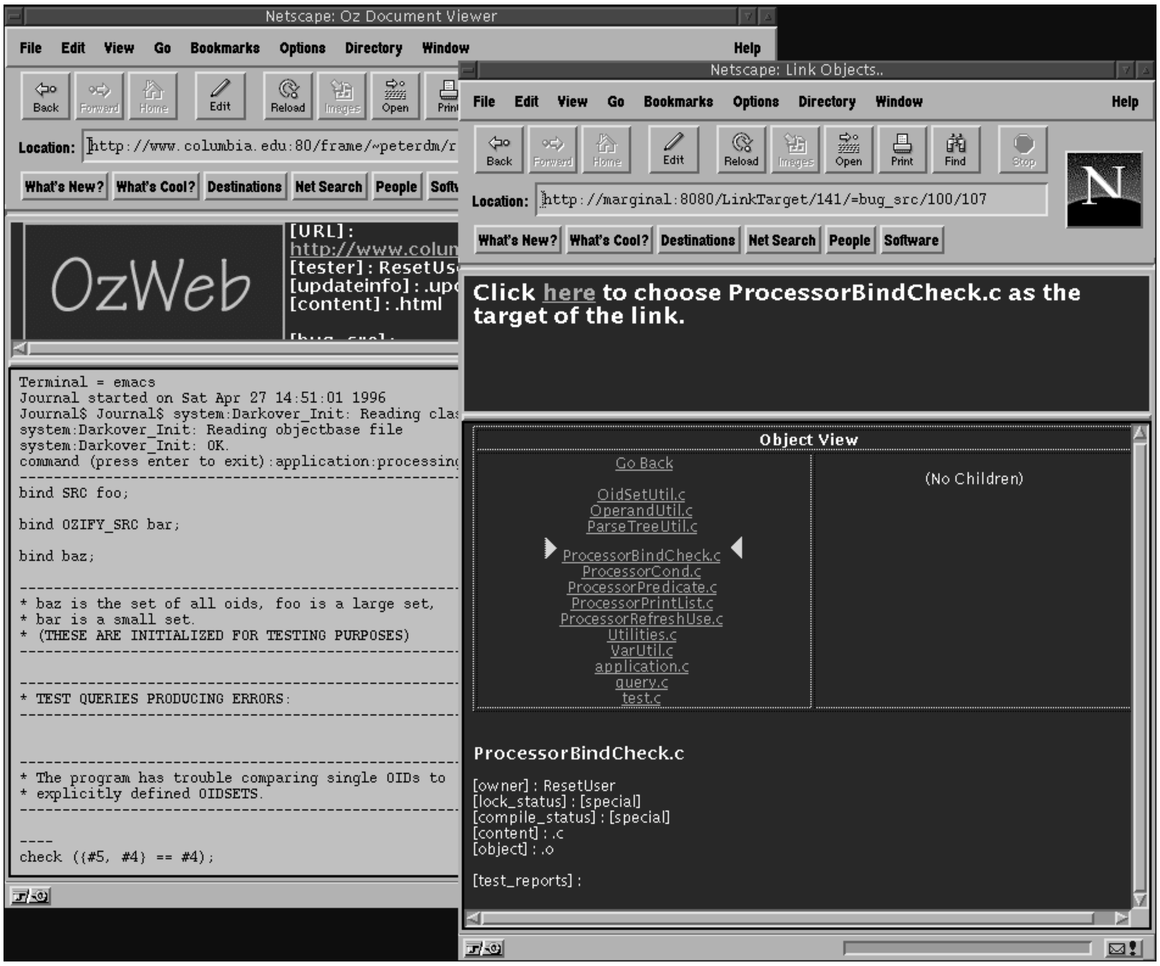

Figure 10: Linking QA Report and Code

the end-user's screen via X Windows capabilities, so hosts not running X Windows were limited to command line tools with no interactive user prompts; this has been alleviated somewhat by the extension to WinDD with personal tool servers running on Windows NT. Scripts and thus tools in **OzWeb** 0.2 were always run using the operating system userid under which the subweb proxy was invoked, say "oz", introducing a potential security concern. In the **OzWeb** 0.35 integration with **Rivendell**, tool servers continue to launch tools under a common userid like "oz", but personal tool servers always execute tools under the relevant user's own userid. The shared-userid problem could be solved by installing tool servers with "root" privileges, in which case they could fork child operating system processes under any local userid, but then each remote end-user would be required to have a userid on the host where the relevant tool server runs. Note there may be conflicts between userids from different administrative domains, e.g., multiple users of the same groupspace with userid "yang".

6.1 Revisit Motivating Scenario 1

We have developed a demonstration **OzWeb** environment supporting our first motivating scenario. Users edit, compile, test, etc. C code for the evolving OODB system and query processor, stored in the subweb along with other project materials. GUI-oriented activities like editing are performed on each user's own workstation, whereas CPU-intensive work such as compilation and system build are performed by tools on a shared server machine. Source code is automatically converted to cross-referenced hypertext (i.e., uses of

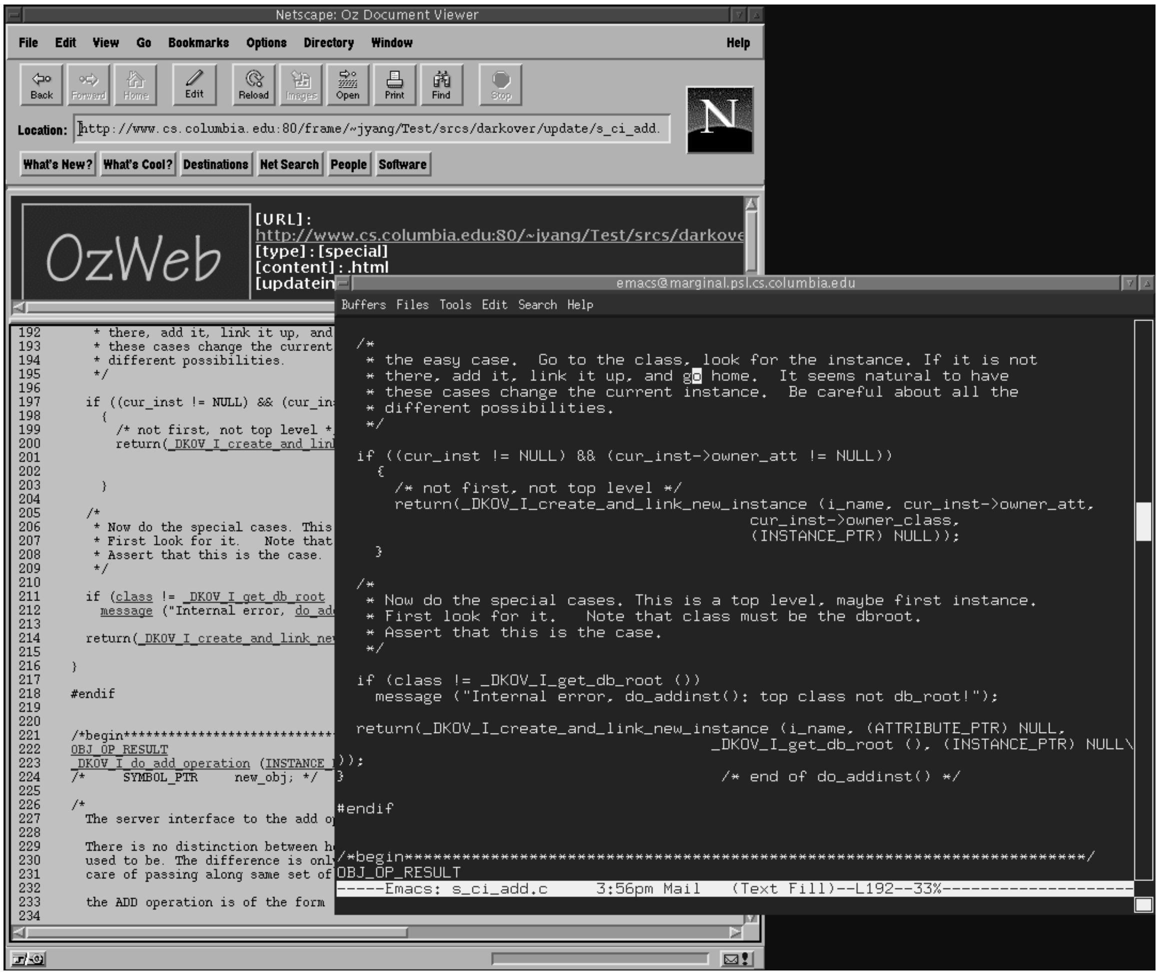

Figure 11: Cross-Linked Code and Tool

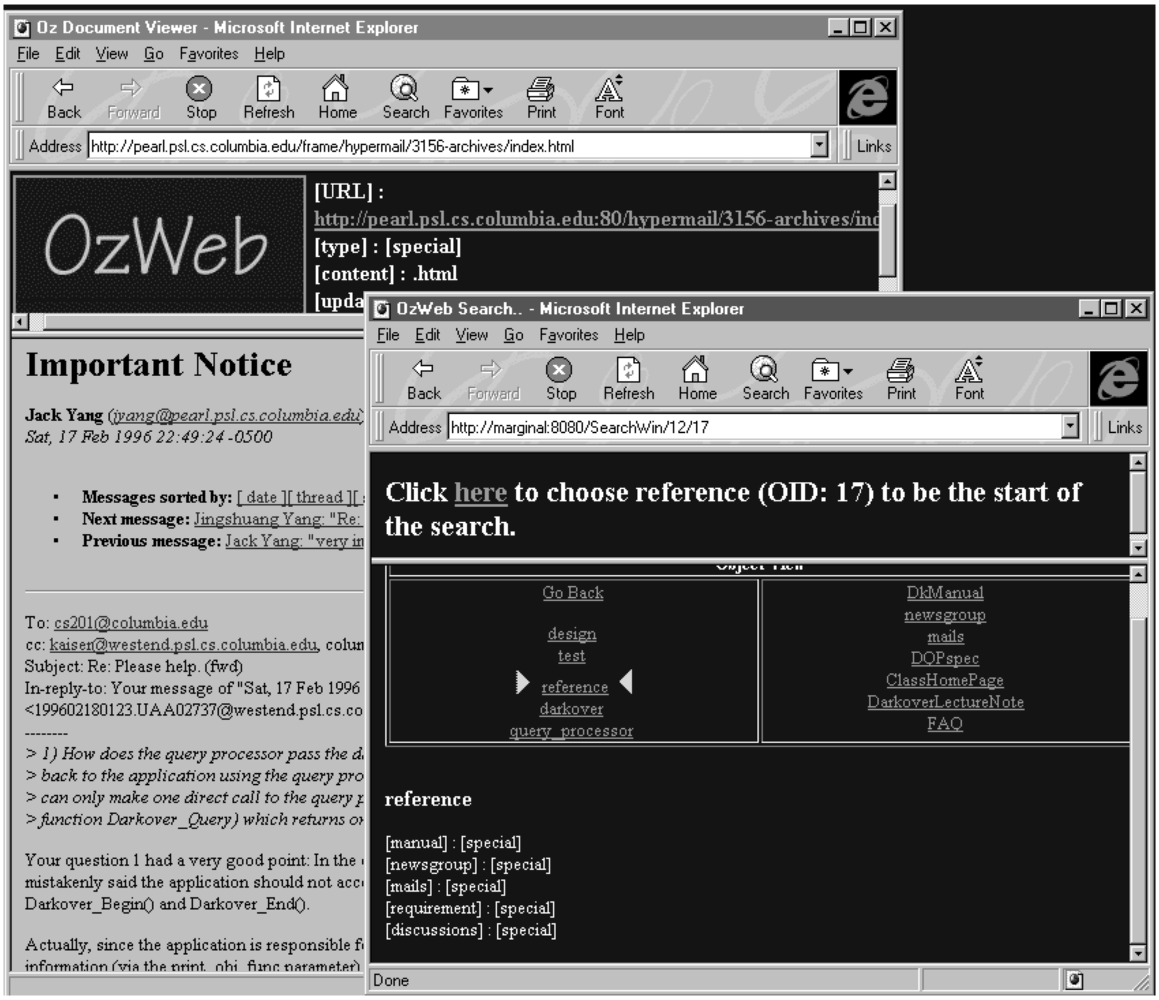

Figure 12: Subweb-specific Search

identifiers are linked to their definitions) via process automation that invokes a home-grown utility called Hi-C. Email archives were converted to HTML using Hypermail [28], and the subweb is populated with the code, design, testing and informal materials described previously, which are divided among three Columbia websites (two public, representing the course materials "as is" developed independently of this research, and one internal for the subweb).

To give a flavor regarding how this hypercode environment might be used, we go through two small process fragments. A user, say Laura, a member of the "lobster" group from our first motivating scenario, views some test reports regarding the baseline code through a standard Web browser. Then she requests the **OzWeb** GUI and adds an external link between one test report and a code file that she believes likely to contain the bug, as depicted in Figure 10. Note Laura is not the author of this test report, which was written by another student group in the spring 1996 class, so she cannot edit it to directly embed a hyperlink. A *link* operation in the rule effect (task consequences) automatically adds a reverse link.A *link* operation in the overloaded rule effect automatically adds a reverse link.

Now in Figure 11, Laura takes a look at the hypertext source code in her browser, and selects the `edit` task link from the Action Menu. In this process segment, a regular ASCII editor tool is invoked instead of an HTML editor (and the HTML is generated), but an alternative process could support a Web editor and strip off the HTML tags before presenting to the C compiler and other COTS C tools. The editor (`emacs`) is invoked by the subweb proxy and displays on Laura's workstation. The `edit` rule indicates to use the plain source code as the tool script argument, not the hypertext version. After the `edit` task completes, Hi-C incrementally updates the cross-references.

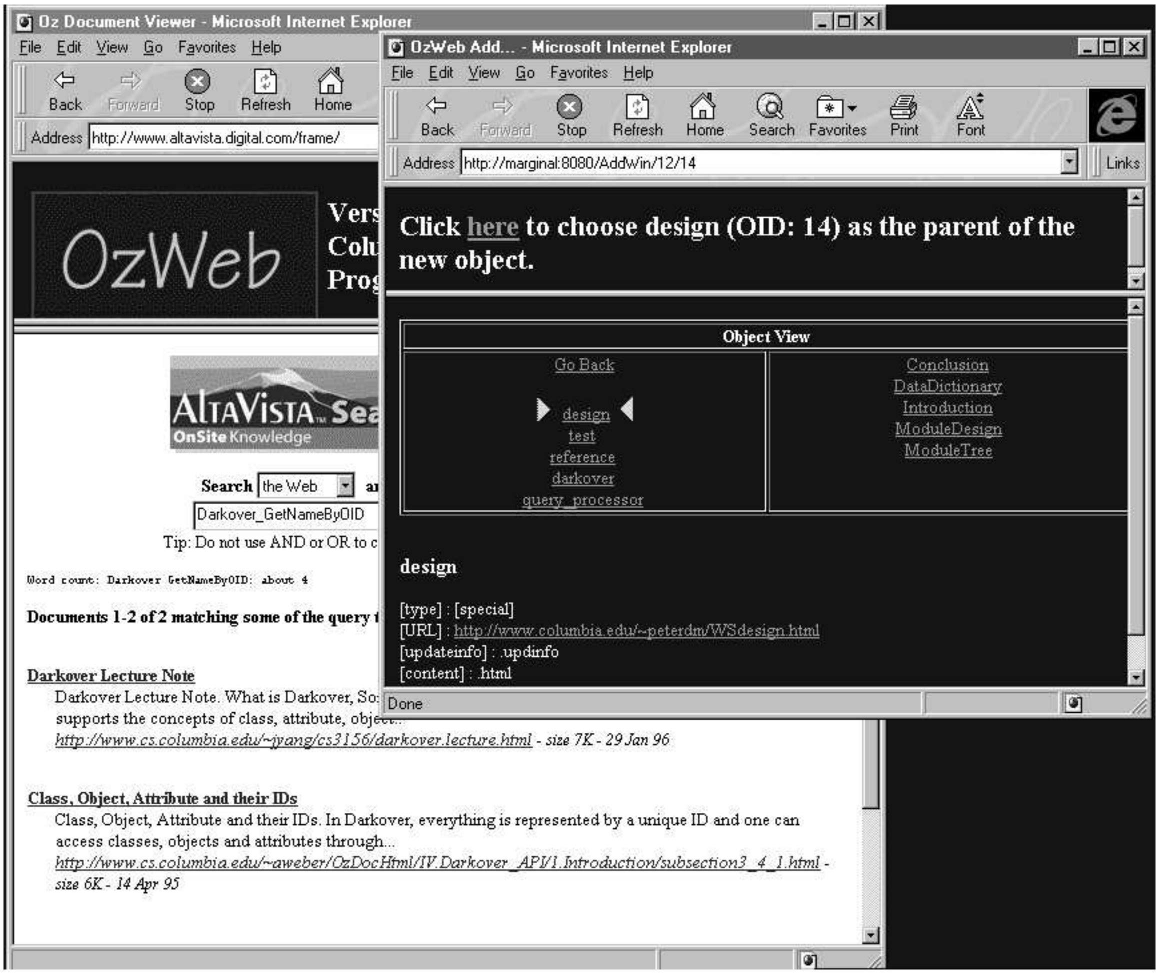

Figure 13: Adding to a Subweb

Meanwhile, say another lobster student Hugh searches through informal "proprietary" documents like email, newsgroup archives, lecture notes, etc. As shown in Figure 12, he uses **OzWeb** *search* to find subweb documents containing the string `''Darkover_GetNameByOID''`, because he is working on this function (Darkover is the name of the relevant OODB). Then Hugh decides to search the full WWW, not just the subweb, using a standard search engine. He finds a "public" page he deems relevant, and adds it to the subweb as depicted in Figure 13.

7 RELATED WORK

ComMentor [38] annotation servers are similar to our subweb servers, adding external application-specific super-structures on top of pre-existing hypermedia, but without the active capabilities of groupspace services. Annotation servers are realized via modified WWW browsers or "plug-ins" to standard browsers. WebMake [40] supports hierarchical structuring of Web files, checkout for editing, and invocation of file-based tools like `make`. A special Web client hides details from users; otherwise WebMake uses standard facilities: CGI and MIME types. Data may be temporarily transmitted to another locale for tool execution (e.g., to compile for a particular architecture) using XMosaic remote control, but the entirety of a structured document normally resides at the same website. There are no transactions operating across separate accesses, and no process. WAIBA [44] produced a suite of utilities useful to collaborative endeavors, such as shared annotations, search engines that find what's changed in categories of interest, what's changed in recently viewed pages, what's out there that's "similar" to what the user is currently viewing, displays link composition to a given depth, displays browsing history graphically, etc. over public and private websites. Meteor [52] is a transactional workflow engine that submits workflow tasks *to* the Web (and backend databases) using CGI, but does not support workflow *over* Web (i.e., URL) accesses.

Databases often support access *from* the Web *to* a database, but not vice versa (e.g., the O2 Web Gateway [36]). A few databases have been constructed on top of hypermedia, akin to our subwebs, although none are close enough to warrant attempted reuse. Most have no support for either workflow or transactions, and are centralized (the equivalent of a single website). Some exceptions: Hyperform [57] is a hypertext database with an extensible object-oriented schema. It allows choice between checkout of individual attributes or entire objects (dirty reads are allowed), and conventional transactions over multiple objects. An application must be implemented by a method of the relevant class in the hyperbase schema to use transactions, but methods written in Scheme can be added dynamically. DHT [41] overlays hypertext to add on transparent access and external organization to distributed, heterogeneous, autonomously-maintained repositories. Repository data is transformed to/from a common structured hypermedia format by a repository-specific gateway. Only navigational access is supported. Update requests are turned down if the object has changed since last retrieved, but there are no transactions, and no process.

WebCard [13] is one of several systems that supports representation of Web pages in the style of email/news folders. Lightweight Databases [19] extends HTML to map relational database schemas onto hypertext documents, to support database-style queries that rely on semantic knowledge of the structure and content of documents. Relationships among hypertext pages augment the usual links. However, data content must be modified to include the entity class, attributes, and relationships. Either client or server-side processing is supported. In contrast, Hyper-G [3] supports hierarchical structuring of aggregate collections of Web pages, where a collection may be a component of multiple parents and collections can be spread across websites. Links are represented externally to the hypermedia content. Scaling is supported by replication and caching (with weak consistency). Hyper-G uses its own SGML [4] format, but converts to HTML when serving a WWW client. It provides an interchange format and utilities for importing and exporting external collections. None of these systems supports process or transactions.

Field [50] uses a message bus to incorporate tools into an integrated software development architecture. This requires either source code availability (to modify the tool to understand Field messages) or an existing API through which the capability can be added. There is little support for integrating other types of tools. Sun's ToolTalk [31] protocol allows applications from differing vendors to share information via a message bus type architecture. ToolTalk could conceivably be employed as another communication option via which users could request tools from a tool server. Integration with Sun development tools would be made easier, since they support the integration of third-party software via the ToolTalk protocol. HP's SoftBench [54] framework, a tool-integration platform that provides an open, common set of communication and user interface services to all tools integrated with the SoftBench environment would also be a useful platform with which to integrate a tool services component. A tool server which connected to SoftBench would make it possible to more easily add non SoftBench-aware tools to the development system.

Matchmaker [39] is a distributed computing interface specification language which allows a programmer to define RPC interfaces between remote processes. A multi-targeted compiler then generates C, Pascal, Lisp, or Ada code which implements the interfaces defined. Polylith [49] extends this concept to allow executing distributed system components to be temporarily shut down and moved among hosts. These and other related models of tool integration are subsumed by the event-based framework presented in [5]. We do not intend to compete with such work, but will consider how to build on top as part of our future directions on groupspace tool servers.

Ockerbloom [43] proposes an alternative to MIME types, called Typed Object Model (TOM), that could conceivably be employed instead of a MIME extension to incorporate a Personal Tool Server into our architecture. Objects types exported from anywhere on the Internet can be registered in "type oracles", specialized servers that may communicate among themselves to uncover the definitions of types registered elsewhere. Web browsers that happen upon a type they do not understand can ask one of the type oracles how to convert it into a known supertype. In this way, the Web browser would not have to be set up to handle a new MIME type. They could simply query the type oracle, which could return information on how to run the tools.

8 CONTRIBUTIONS AND FUTURE WORK

We have designed a general approach to hypermedia collaboration environments centered on subweb repositories and groupspace services, investigated many of the technical issues — particularly distributed tool services — in depth, developed a feasible architecture and implementation techniques based on WWW technology, and realized a prototype environment framework and several sample **OzWeb** environments, including the "WebCity" environment we use for our own day-to-day team software development work.

OzWeb reuses Oz's process engine and transaction manager more-or-less "as is"; the same OODB is used in the subweb implementation. A new tool service component, **Rivendell**, replaced Oz's native tool management mechanism. In future work we would like to further exploit our recent componentization direction, where these components have been tugged apart, are in principle replaceable within Oz, and have been experimentally introduced into foreign systems, such as ProcessWEAVER [23], as described in [27, 46]. For example, a later version of **OzWeb** might employ an alternative process engine, or a synergized set of process engines, e.g., [33] explains how we integrated Oz's (and now **OzWeb**'s) process engine with TeamWare [10].

Our subweb architecture seems reasonably general, simple and elegant for adapting existing client/server process-centered environments into a hypermedia collaboration environment framework. The server for a process-centered environment would already provide basic groupspace functionality and could be augmented by the subweb mechanism, as we described for Oz. Native clients would be replaced by Web browsers and proxy servers. Peer/peer process-centered environments where the peers already share a common repository [45] may be adaptable to multiple groupspace servers sharing a common subweb server, but that investigation is outside our scope.

For fully distributed, "shared nothing" multi-server systems, for example to scale up to collaborative work among teams, as opposed to among individual members of one team, we plan to expand the "International Alliance" metaphor developed for Oz [7]. The gist is that groupspaces would register with one or more LDAP's to become available to their subweb proxy servers. Groupspaces affiliated with the same LDAP

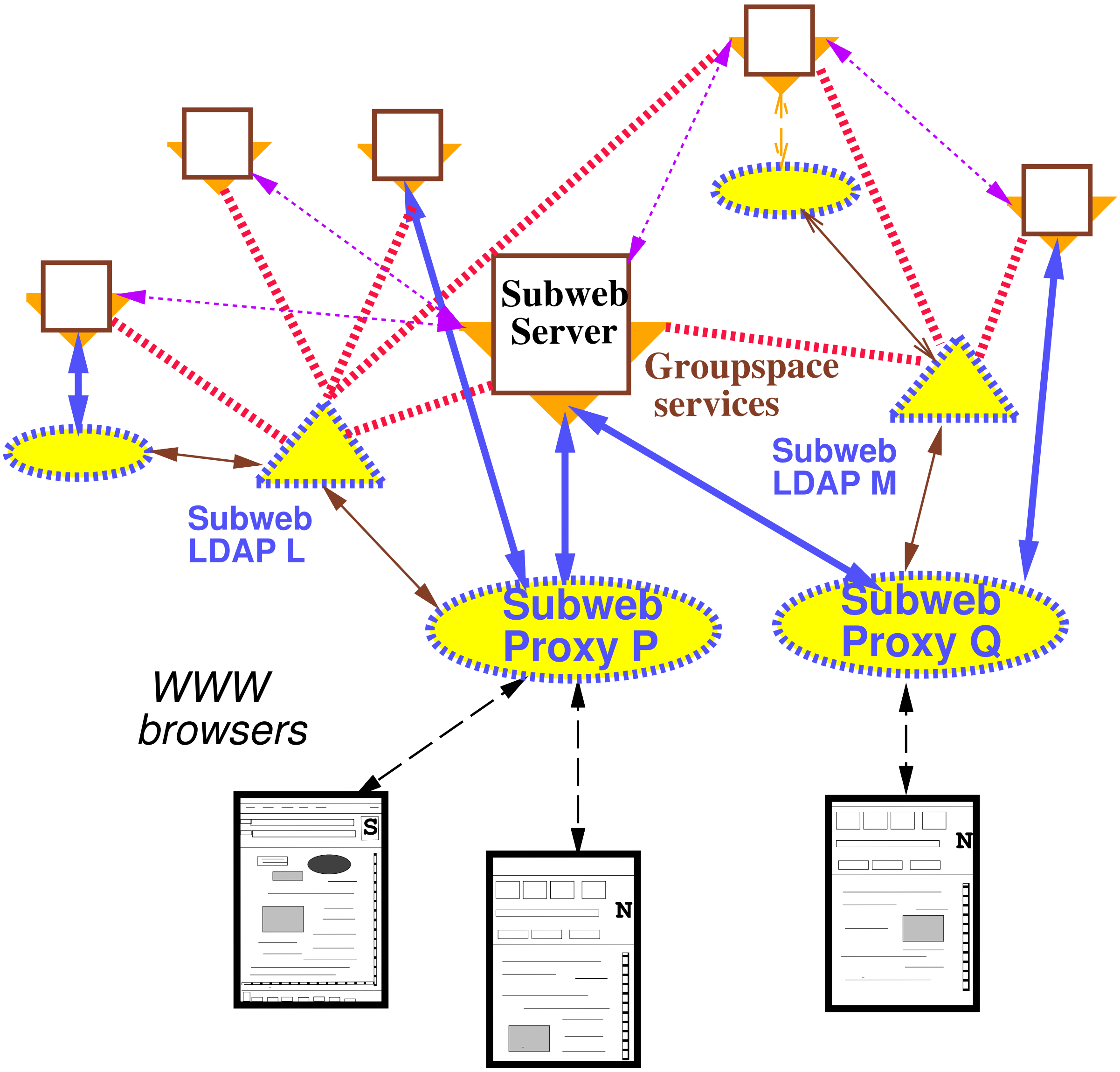

Figure 14: Alliances Architecture

could optionally form "Treaties" with one or more other such groupspaces to agree to share selected portions of their schemas, subwebs, process models and/or tools, and specify how to handle situations where the same ultimate hypermedia entity is represented by multiple web objects and/or is moved among subwebs. Then "Summits" would implement one groupspace performing a process task on behalf of another groupspace, users and tools from multiple groupspaces collaboratively performing process tasks, movement of users and data from one groupspace to another, etc. This model is illustrated in Figure 14.

For navigation and movement of users, data and tasks among groupspaces, we plan to develop a "meta-web" analogous to multi-user domains (MUDs, e.g., Lambda MOO [58]), where users can move from room to adjacent room (sometimes through trap-doors) and pick up an object in one room and carry it to another (but the object may transform from a trunk filled with gold to a monkey). Workflow-based MUDs have already been investigated in Promo [20]. However, in our case the same groupspace may participate in multiple alliances, each corresponding to a MUD, so exactly what a user can do and what objects he/she can access may depend on how that user arrived in the groupspace. This does not seem as challenging technically as from a user interface perspective, so we are working on what we call the "Twilight Zone Fifth Dimension" user model to address the end-user confusion that could naturally occur.

It is tempting to consider how other distributed computing infrastructures, besides WWW, might serve as the basis for subweb and groupspace implementation. We studied CORBA 2 as a candidate, but it is currently lacking the key ingredient: a standard component that supports complete interception and mediation in the style of HTTP proxy servers. This problem may be ameliorated by the various proposed integrations of HTTP and CORBA's Internet Inter-Orb Protocol, which we plan to explore as an alternative standard on which to base subwebs and groupspaces.

9 ACKNOWLEDGEMENTS

We thank George Heineman, Dick Taylor, Sankar Virdhagriswaran and Alex Wolf for useful technical discussions. Laura Xiaoyu Xu participated in the development of the first motivating scenario. **OzWeb** 0.35 has been released in beta form to Brown University and the University of California at Irvine, and is available for other external use.

Effort sponsored by the Defense Advanced Research Projects Agency, and Rome Laboratory, Air Force Materiel Command, USAF, under agreements number F30602-94-C-0197/F30602-97-2-0022, by NSF CDA-9529304, and by New York State Science and Technology Foundation, Center for Advanced Technology – High Performance Computing and Communications in Healthcare. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, the Air Force, the U.S. Government, NSF or NYSSTF.

References

[1] Action Workflow Metro World-Wide-Workflow, November 1995. www.actiontech.com/metrotour/resources/Metwp.htm.

[2] Kenneth M. Anderson, Richard N. Taylor, and E. James Whitehead, Jr. Chimera: Hypertext for heterogeneous software environments. In *1994 European Conference on Hypermedia Technology*, pages 94–107, Edinburgh, Scotland, September 1994.

[3] Keith Andrews, Frank Kappe, and Hermann Maurer. Serving information to the Web with Hyper-G. In *3rd International World-Wide Web Conference*, Darmstadt, Germany, April 1995. Elsevier Science B.V. www.igd.fhg.de/www/www95/proceedings/papers/105/hgw3.html.

[4] Standard for electronic manuscript preparation and markup, 1986.

[5] Daniel J. Barrett, Lori A. Clarke, Peri L. Tarr, and Alexander E. Wise. A framework for event-based software integration. *ACM Transactions on Software Engineering and Methodology*, 5(4):378–421, October 1996.

[6] Israel Ben-Shaul and Gail E. Kaiser. *A Paradigm for Decentralized Process Modeling.* Kluwer Academic Publishers, Boston, 1995.

[7] Israel Z. Ben-Shaul and Gail E. Kaiser. An interoperability model for process-centered software engineering environments and its implementation in Oz. Technical Report CUCS-034-95, Columbia University Department of Computer Science, December 1995. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-034-95.ps.Z.

[8] T. Berners-Lee and D. Connolly. Hypertext Markup Language – 2.0, November 1995. Network Working Group Request For Comments: 1866,.

[9] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0, May 1996. Network Working Group Request For Comments: 1945.

[10] Gregory Alan Bolcer and Richard N. Taylor. Endeavors: A process system integration infrastructure. In Wilhelm Schäfer, editor, *4th International Conference on the Software Process: Software Process – Improvement and Practice*, December 1996. In press.

[11] N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions) part one: Mechanisms for specifying and describing the format of Internet message bodies, September 1993. Network Working Group Request for Comments: 1521.

[12] Charles Brooks, Murray S. Mazer, Scott Meeks, and Jim Miller. Application-specific proxy servers as HTTP stream transducers. In *World Wide Web Journal: 4th International World Wide Web Conference*, pages 539–548, Boston MA, December 1995. O'Reilly & Associates.

[13] Marc H. Brown. WebCard: Integrated and uniform access to mail, news, and the Web. Technical Report 139a, DEC Systems Research Center, July 1996. ftp.digital.com/pub/DEC/SRC/research-reports/SRC-139a.html.

[14] cgi@ncsa.uiuc.edu. The Common Gateway Interface. hoohoo.ncsa.uiuc.edu/cgi/overview.html.

[15] Citrix, Inc. Citrix WinFrame. http://www.citrix.com/.

[16] World-Wide-Web Consortium. W3C Reference Library. http://www.w3.org/pub/WWW/Library/.

[17] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Communications of the ACM*, 35(9):75–90, September 1992.

[18] Umesh Dayal, Hector Garcia-Molina, Mei Hsu, Ben Kao, and Ming-Chien Shan. Third generation TP monitors: A database challenge. In *1993 SIGMOD International Conference on Management of Data*, pages 393–398, May 1993. Special issue of *SIGMOD Record*, 22(2), June 1993.

[19] S.A. Dobson and V.A. Burrill. Lightweight databases. In *3rd International World-Wide Web Conference*, Darmstadt, Germany, April 1995. Elsevier Science B.V. www.igd.fhg.de/www/www95/proceedings/papers/54/darm.html.

[20] John C. Doppke. Software processing modeling and execution within virtual environments. Technical Report CU-CS-805-96, University of Colorado Department of Computer Science, July 1996.

[21] Stephen E. Dossick and Gail E. Kaiser. WWW access to legacy client/server applications. In *5th International World Wide Web Conference*, pages 931–940, Paris, France, May 1996. Elsevier Science B.V. Special issue of *Computer Networks and ISDN Systems, The International Journal of Computer and Telecommunications Networking*, 28(7-11), May 1996. http://www.psl.cs.columbia.edu/papers/CUCS-003-96.html.

[22] Ahmed K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann, San Mateo CA, 1992.

[23] Christer Fernström. PROCESS WEAVER: Adding process support to UNIX. In *2nd International Conference on the Software Process: Continuous Software Process Improvement*, pages 12–26, Berlin, Germany, February 1993. IEEE Computer Society Press.

[24] Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modelling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–152, 1995.

[25] Mark A. Gisi and Gail E. Kaiser. Extending a tool integration language. In Mark Dowson, editor, *1st International Conference on the Software Process: Manufacturing Complex Systems*, pages 218–227, Redondo Beach CA, October 1991. IEEE Computer Society Press. ftp://ftp.psl.cs.columbia.edu/pub/psl/icsp91.ps.Z.

[26] The Harvest information discovery and access system, December 1995. harvest.cs.colorado.edu/.

[27] George T. Heineman. *A Transaction Manager Component for Cooperative Transaction Models.* PhD thesis, Columbia University Department of Computer Science, June 1996. CUCS-010-96. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-010-96.ps.gz.

[28] Kevin Hughes. Hypermail, 1996. www.eit.com/goodies/software/hypermail/.

[29] Insignia Solutions, Inc. What is NTrigue? http://www.insignia.com/marcom/DataSheets/NTrigue_DataSheet.html.

[30] Gail E. Kaiser Jack Jingshuang Yang and Stephen E. Dossick. An external transaction service for www. Technical Report CUCS-047-96, Columbia University Department of Computer Science, December 1996.

[31] Astrid M. Julienne and Brian Holtz. *ToolTalk & Open Protocols: Inter-Application Communication*. Prentice Hall, Englewood Cliffs NJ, 1994.

[32] Gail E. Kaiser. Cooperative transactions for multi-user environments. In Won Kim, editor, *Modern Database Systems: The Object Model, Interoperability, and Beyond*, chapter 20, pages 409–433. ACM Press, New York NY, 1994. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-006-93.ps.Z.

[33] Gail E. Kaiser, Israel Z. Ben-Shaul, Steven S. Popovich, and Stephen E. Dossick. A metalinguistic approach to process enactment extensibility. In Wilhelm Schäfer, editor, *4th International Conference on the Software Process: Improvement and Practice*, Brighton, UK, December 1996. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-016-96.ps.gz.

[34] Wenke Lee, Gail E. Kaiser, Paul D. Clayton, and Eric H. Sherman. OzCare: A workflow automation system for care plans. In James J. Cimino, editor, *1996 American Medical Informatics Association Annual Fall Symposium*, pages 577–581, Washington DC, October 1996.

[35] Lotus .domino. http://domino.lotus.com.

[36] Jean-Claude Mamou. OBDC and Mosaic, October 1995. www.w3.org/hypertext/WWW/Gateways/OQL.html.

[37] Udi Manber, Sun Wu, and Burra Gopal. GLIMPSE: A tool to search entire file systems, 1996. glimpse.cs.arizona.edu/.

[38] Christian Mogensen Martin Roscheisen and Terry Winograd. Beyond browsing: Shared commands, soaps, trails, and on-line communities. In *3rd International World-Wide Web Conference*, April 1995. www.igd.fhg.de/www/www95/proceedings/papers/88/TR/WWW95.html.

[39] Richard F. Rashid Michael B. Jones and Mary R. Thompson. Matchmaker: An interface specification language for distributed processing. In *12th Annual ACM Symposium on Principles of Programming Languages*, pages 225–235, New Orleans LA, January 1985.

[40] Michael Baentsch, Georg Molter and Peter Sturm. WebMake: Integrating distributed software development in a structure-enhanced Web. In *3rd International World-Wide Web Conference*, Darmstadt, Germany, April 1995. Elsevier Science B.V. www.igd.fhg.de/www/www95/proceedings/papers/51/WebMake/WebMake.html.

[41] John Noll and Walt Scacchi. A hypertext system for integrating heterogeneous autonomous software repositories. In *3rd Irvine Software Symposium*, pages 49–60, Irvine CA, April 1994.

[42] Object Management Group. *The Common Object Request Broker: Architecture Specification Revision 2.0*, July 1995. www.omg.org/corba2/cover.htm.

[43] John Ockerbloom. Introducing structured data types into Internet-scale information systems, May 1994. PhD Thesis Proposal. www.cs.cmu.edu/afs/cs.cmu.edu/user/spok/www/proposal.html.

[44] OSF Research Institute. *Intelligent Browsing Assistant for the World Wide Web and GroupWare for the Web*, October 1995. www.osf.org/www/waiba/index.html.

[45] Burkhard Peuschel and Stefan Wolf. Architectural support for distributed process centered software development environments. In Wilhelm Schäfer, editor, *8th International Software Process Workshop: State of the Practice in Process Technology*, pages 126–128, Wadern, Germany, March 1993. Position paper.

[46] Steven S. Popovich. *An Architecture for Extensible Workflow Process Servers*. PhD thesis, Columbia University Department of Computer Science, January 1997. CUCS-014-96. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-014-96.ps.gz.

[47] Programming Systems Laboratory. Application Specific Http Services (ASHeS). www.psl.cs.columbia.edu/software/ashes.html.

[48] Programming Systems Laboratory. *Oz 1.2 Manual Set*, July 1996. Columbia University Department of Computer Science. ftp.psl.cs.columbia.edu/pub/psl/oz.1.2.manuals.

[49] James M. Purtilo and Christine R. Hofmeister. Dynamic reconfiguration of distributed programs. In *11th International Conference on Distributed Computing Systems*, pages 560–571, Arlington TX, May 1991. IEEE Computer Society Press.

[50] Steven P. Reiss. Connecting tools using message passing in the Field environment. *IEEE Software*, 7(4):57–66, July 1990.

[51] Hans Schuster, Stefan Jablonski, and Christoph Buvler. Client/server qualities: A basis for reliable distributed workflow management systems. In *17th International Conference on Distributed Computing Systems*, Baltimore MD, May 1997. In press.

[52] Amit Sheth. Private communication, June 1996. See lsdis.cs.uga.edu/workflow/.

[53] Tektronix, Inc. WinDD Network Display.
http://www.tek.com/Network_Displays/Support/PAPERS/Web-5.doc.html.

[54] Gary Thunquest. Supporting task management & process automation in the softbench development environment. In Ian Thomas, editor, *7th International Software Process Workshop: Communication and Coordination in the Software Process*, pages 133–135, Yountville CA, October 1991. IEEE Computer Society Press.

[55] Giuseppe Valetto and Gail E. Kaiser. Enveloping sophisticated tools into process-centered environments. *Journal of Automated Software Engineering*, 3:309–345, 1996.
ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-022-95.ps.gz.

[56] Jim Whitehead. Working group on versioning and configuration management of World Wide Web content, June 1996. www.ics.uci.edu/~ejw/versioning/.

[57] Uffe K. Wiil. Hyperform: Rapid prototyping of hypermedia services. *Communications of the ACM*, 38(8):109–111, August 1995.

[58] Lambda MOO. telnet://lambda.parc.xerox.com:8888.

[59] Jack Jingshuang Yang and Gail E. Kaiser. An architecture for integrating OODBs with WWW. In *5th International World Wide Web Conference*, pages 1243–1254, Paris, France, May 1996. Elsevier Science B.V. Special issue of *Computer Networks and ISDN Systems, The International Journal of Computer and Telecommunications Networking*, 28(7-11), May 1996.
http://www.psl.cs.columbia.edu/papers/CUCS-004-96.html.

[60] W. Yeong, T. Howes, and S. Kille. Lightweight Directory Access Protocol, March 1995. Network Working Group Request For Comments: 1777, andrew2.andrew.cmu.edu/rfc/rfc1777.html.