A Virtual Environment Framework For Software Engineering

Stephen E. Dossick

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2001

Abstract

A Virtual Environment Framework for Software Engineering

Stephen E. Dossick

The field of Software Engineering is concerned with the investigation of new procedures and techniques which aid in the development of computer software. The holy grail of Software Engineering is the achievement of so-called “six-sigma” error rates (i.e. 99.999999% defect free), a rating pioneered in the Electrical Engineering field. The Software Engineering research community has long developed and experimented with new tools aimed at easing the problems faced in the process of building software products.

In this dissertation, we report on research into the problem of scaling software development to hundreds or thousands of simultaneous workers using thousands or hundreds of thousands of project artifacts during the course of development of a software product. We have developed a framework which enables the application of Virtual Environment techniques for the creation of Software Immersion Environments, a new form of virtual environment (targeted at Software Engineering) in which the project team members (developers, project managers, testers, etc.) walk among project artifacts in a computer-generated 3d space as though they were real objects. By increasing the project workers’ access to the artifacts associated with their develop-

ment tasks, we hope to make the process of developing large-scale software systems easier.

Our framework, known as CHIME (the Columbia Hypermedia IMmersion Environment), is built around a number of novel components. These include Xanth, an XML-based middleware information broker, the Groupspace Controller, an event bus which supports both pre- and post- event notifications and vetoes as well as service layering, the CHIME Virtual Environment Manager, responsible for mapping project artifacts into virtual environment furnishings, and the CHIME Theme Manager, a component which mediates between the virtual environment users and the information they are manipulating.

Table of Contents

List of Figures

CHAPTER 1 Introduction

‘Technology’ is ‘things invented after I was born’.
— Alan Kay

For decades, the field of Software Engineering has been dedicated to the investigation of techniques and tools to make building and maintaining software systems easier. The holy grail of Software Engineering is the achievement of quality levels like those achieved by our cousins in Electrical Engineering, where so-called Six Sigma error rates (i.e. 99.999999% defect-free) are possible.

As software systems grow older and larger, it becomes increasingly impossible for any one member of a development team to keep an accurate picture in his or her head regarding the entire system. The sheer number of artifacts (design and architecture documentation and rationale, testing plans, module interaction documentation, performance characteristics and requirements, bug reports and solutions, in addition to the source code itself) generated by a development team maintaining and enhancing a given software system makes the task of each developer grow more difficult every day. In addition, with development team members leaving the team and new people joining, the original developers of a system may no longer be involved with the project. This is the reason why, when given a choice, most developers (and graduate students) will choose to develop new functionality as an entirely new piece of software rather than graft new capabilities onto an existing, proven system. Most developers realize intuitively that it is often easier to build a whole system from scratch with identical functionality than to maintain and modify software with which they are unfamiliar. Clearly, it is often infeasible from a business standpoint to constantly throw away software.

Brooks' Law [Brooks, 1995] tells us that as we add more developers to a software project, the communications overhead of additional developers keeping up to date with modifications to other parts of the system begins to outweigh the benefits of adding more people to the project. Thus it is often the case that a 3-5 person development team may finish a project in less time and with fewer bugs than a 10-12 person team. There seems to be a corollary to Brooks' Law which tells us that as the number of artifacts relating to a given system grows, the harder it becomes for a development team to keep up. It certainly becomes more difficult for new project members to come up to speed on the roles and interactions of each component of the system. In addition, departing team members take even more of the "corporate memory" with them when they leave.

The Software Engineering research community has long investigated solutions to these problems. Software Development Environment (SDEs) research in particular has offered several compelling possibilities. SDEs can potentially organize all project artifacts and place them at a developers' fingertips for easy access and use. In addition, a subset of SDEs, known as Process-Centered Software Development Environments, can be programmed with the particular software process or workflow in use by a development team and can potentially guide developers with regards to which artifacts are to be used at a particular stage of development.

Rationale Capture systems have been built which attempt to build a knowledgebase of decisions made during system development and then exploit that knowledgebase to help project members find out why certain decisions were made and why the alternatives were dismissed. In this way, Rationale Capture tools can make it possible for new team members to gain insights into the thought processes of the original developers of a particular component which may go beyond the information they can learn from design documentation and other artifacts.

Software Visualization (and the related, somewhat overlapping area of Program Understanding) systems work to make it easier for developers to learn and understand how a particular module or algorithm works, or to see how given modules interact within a larger system. They attempt to accomplish this by using various graphical techniques (ranging from 2D-text displays all the way to more common 3D displays) to map the workings of a piece of source code to an onscreen display. It is worth noting, however, that research to date in this area have focused on extremely small, effectively “toy” systems; the problems inherent in building software visualizations of large scale software projects have not been solved. Nonetheless, this area of research seems quite promising.

In recent years, Collaborative Virtual Environments (CVEs) have been increasingly investigated by the CSCW community. CVEs are multi-user shared spaces in which users can communicate and work together on various objects and artifacts found within the shared space. Many CVEs trace their core concepts back to Multi-User Domains (MUDs) and chat rooms found on the Internet. While many CVEs are text-based, research and commercial groups are both beginning to work with 2D and 3D CVEs, in which the users inhabit a graphical space together. Studies of groups performing “real work” (as opposed to fun and games) inside CVEs have shown real productivity improvements for teams, especially when those teams are geographically or temporally dispersed [Benford et al., 1997].

1.1 A Virtual Environment Framework for Software Engineering

The research reported in this dissertation is motivated by a number of issues. First and foremost, how can we design a framework for building software development environments which makes it easy for developers to gain access to relevant artifacts of the software project *no matter where those artifacts come from*. This last point is extremely important. In traditional SDEs, external tools (word processors, design tools, etc.) may be used during development, but the SDE must eventually control and store the artifacts locally. While this may be feasible for certain development tools (word processing documents, for instance, are usually easily stored as a single file and thus lend themselves to control by the SDE), it is hard or impossible with others.

Rationale Capture systems, for instance, often store data in proprietary databases. Documentation for a given component or library may be accessible only on a World Wide Web (WWW) site, not in a form storable inside an SDE. Yet if the goal of an SDE is to make it easier for developers to perform their work, access to these kinds of data is imperative.

Second, we would like our framework to allow us to investigate the possibilities of CVEs as a medium for software development. As mentioned above, the CSCW community has reported that CVEs can indeed aid in productivity, especially in cases where some or all of the team members are geographically or temporally distributed. Models of collaboration have been developed to describe the desirable features of CVEs [Fitzpatrick, 1998], and our framework should exploit this work and apply it to software development.

This research meets these challenges by combining features of Software Development Environments, Collaborative Virtual Environments, and Software Visualization systems into a framework for building environments for “Software Immersion.” In a Software Immersion, the developers are immersed into a virtual environment made up entirely of artifacts (and the relationships among those artifacts) from the development project they are working on. This differs from Software Visualization in that we are not attempting to display the inner workings of a particular software component, but rather to illustrate the relationships among the artifacts of the development process. It is our hope that Software Immersion will aid developers in finding the information they need to carry out the daily tasks of software development.

The goal of this research is to create a framework for building Software Immersions which perform both the more traditional functions of SDEs as well as apply the unique strengths of CVEs to the discipline of Software Engineering. The primary concern is to make it possible for a development team to quickly and easily build a CVE which is tailored to their particular development methods and processes. Support for collaborations among geographically distributed workers is also an important concern of this work. In addition, a major goal of the work is to design and build a framework which can scale equally well down to small projects as well as up to an extremely large, multi-team project involving 30 year-old software and hundreds of developers during its lifespan.

The framework that we have created to fulfill these goals, known as CHIME (the Columbia Hypermedia IMmersion Environment), addresses these concerns primarily through *flexibility* and *extensibility*. With CHIME, developers of a Software Immersion are able to easily incorporate software project artifacts and tools into the resulting CVE, regardless of where those artifacts reside (i.e. regardless of what organization or software engineering tool retains control over the artifacts). CHIME utilizes an XML-based metadata architecture to describe artifacts and the mechanisms or protocols via which they can be accessed. The set of protocols can be extended at runtime to allow integration with essentially any other system or tool.

CHIME includes the most common building blocks we feel developers of Software Immersions will need to be successful. These include modules handling bi-directional, n-ary, typed hypertext linking among project artifacts, protocol modules for accessing artifacts stored in common Software Engineering tools (including modules handling WWW-based repositories, CVS configuration management repositories, and SQL Databases in addition to more mundane filesystem repositories), as well as capabilities for creating federated, cooperating Software Immersions whereby multiple organizations can collaborate. In addition, we have created several example Software Immersions which help illustrate the power of our framework.

This dissertation will discuss the design and implementation of the various components of CHIME, and how they fit together to satisfy our goal of a flexible and extensible framework for building Software Immersions. Examples of the use of CHIME's various components, and justifications for various design decisions, will be provided by presenting simple examples and discussing more complete CVEs built with the framework.

1.2 Research Contributions

The research described in this dissertation involves the creation of a framework which assists developers in the creation of Software Immersions. The contributions of this research are:

1. Software Immersion, the model for a novel combination of research work from a number of areas (including Software Development Environments, CVEs, and Software Visualization).
2. Groupspaces and Groupviews, models for persistent electronic collaboration services which underpin Software Immersions. A software component known as the Groupspace Controller implements these models through a unique event-based middleware system with support for pre- and post- notification of events as well as support for vetoable events.
3. The CHIME framework and implementation, which ties together several aspects of our work into a flexible and extensible toolkit for the creation and exploration of Software Immersions.

1.2.1 Software Immersion

In a Software Immersion, team members collaborate and perform individual tasks in a virtual space defined by the structure of the artifacts and tools making up the software system. This builds in some respects on previous work done in the Software Visualization community (see [De Pauw, et al., 1993]) in which visualizations of software module interactions and interrelations are created. The primary difference here is that a Software Immersion is intended to be built semi-automatically, while most software visualizations are generated by hand by human experts. When visualizations have been created by software, the generating software has been built to handle a certain small class of input (output from sorting algorithms for example [Stasko, et al., 1998]).

When applied properly, Software Immersion can speed the learning curve faced by new project members. The architecture and organization of the system they are learning is no longer an abstract concept, it is something they can walk around in and inhabit. Software Immersion is similar in concept to emerging technology in use in Civil Engineering. In [Hogan, 1998], the author shows quantitatively that new construction project members come up to speed faster and perform fewer mistakes when an immersive, virtual construction environment is built from the building design.

1.2.2 Groupspaces and Groupviews

We use the term Groupspace to describe a persistent collaborative virtual space in which participants work. The participants may be geographically or temporally distributed, and they may be from different organizations cooperating on a common project (subcontractors on a defense contract, for example). Contained within the groupspace are project artifacts as well as the tools used to create, modify, and maintain them. Artifacts may be organized and re-organized at will by project participants.

Central to the Groupspace concept is the idea that project artifacts continue to exist in their original form in their original repositories. This differs from traditional Software Development Environments (SDEs) (like Oz [Ben-Shaul and Kaiser, 1995], Desert [Reiss, 1998], Sun NSE [Sun Microsystems, 1988], Microsoft Visual C++ [Microsoft, 2000]), as well as most traditional Groupware systems (like eRoom [Instinctive, 2000], TeamRooms [TeamWave, 2000], and Orbit [Mansfield, et al., 1997]) in which artifacts are under the strict control of the environment. In these systems, users are expected to access artifacts only through the development environment's cadre of tools or via COTS tools specially “wrapped” to work with the environment. In a Groupspace, artifacts continue to exist in their legacy databases, configuration management systems, bug tracking systems, rationale capture tools, etc.

Additionally, Groupspaces may contain information generated within the space. A particular Groupspace may contain built-in tools and services to be used by participants, e.g. to add arbitrary annotations to particular artifacts, hold real-time chat sessions, add hypertext links on top of (and separate from) artifacts in the system, semi-automatically propagate knowledge among participants (in the manner of a recommender system [Sarwar, et al., 1998]), etc.

We use the term Groupviews to describe multiuser, scalable user interfaces used to navigate and work in a Groupspace. In addition to allowing Groupspace participants to find and access relevant information quickly (as they might in a single user system, or a system in which they had no knowledge of other users' actions), Groupviews keep users informed about work being performed by fellow users.

Groupviews build on research and commercial work in Multi-User Domains (MUDs) [Curtis, 1992], chat systems [MIRC, 2000], virtual environments [Electric Communities, 2000], and 3d immersive games like DOOM [Id Software, 1995]. In a Groupview, a set of virtual environment rooms containing project artifacts is generated from the organization of the artifacts in the Groupspace. Rather than placing artifacts into these rooms arbitrarily or according to some external mapping mechanism (as in Promo [Doppke, et al., 1998], where the mapping from artifacts to rooms is created from a software process definition and cannot be modified by users without corresponding modification to the process), a Groupview generates the rooms and connections between the rooms from the artifacts themselves. For example, a software module might become a room in the Groupview, and the source files making up the module might be furnishings inside the room. Corridors might link the modules' room with rooms containing design documentation, test reports, and other artifacts related to the code.

A core aspect of Groupviews is the ability to provide selective awareness of other users' actions. Participants' locations in the virtual environment, as well as their scope of interest (i.e. the project or projects they are currently involved in, portions of the system they are considered “expert” in, related documents they have recently read, written, or modified, etc.) are shared among other users. In the case of a Groupview involving multiple teams working on separate (but interrelated) portions of a project, it should be possible for users to “tune” awareness so they receive only information relevant to them and their work. In this respect, Groupviews are closely related to the Locales framework[Mansfield, et al., 1997], well-known research from the CSCW community which investigates awareness capabilities and other aspects of work performed in a virtual environment.

1.2.3 The CHIME Framework

The CHIME framework embodies and instantiates out models of Groupspaces, Groupviews, and Software Immersion. Its main goal is to allow software developers to generate useful Software Immersions for their projects quickly and easily. CHIME breaks down the process of creating Software Immersions into three main steps (and three main framework components). First, identification of the data (including source code, design documents, test plans, etc.) which is to be included as part of the eventual Software Immersion. CHIME's Xanth Data Server component is responsible for maintaining this collection of data, as well as any hypertextual links which have been layered on top of this data (for instance, linking source code to the design documentations or testing plans which deal with it). In addition, Xanth acts as lightweight, extensible middleware, allowing other CHIME components access to the back-end data.

The next task of the developer of a Software Immersion is deciding what roles the various pieces of data will play in the eventual virtual environment. CHIME's Virtual Environment Modeler (VEM) allows developers to "tag" each piece of data from the Xanth Data Server with one of an extensible set of Virtual Environment Types. Base types include 'Container,' 'Component,' and 'Connector,' which correspond vaguely with standard virtual environment concepts of 'Room,' 'Object in Room,' and

‘Link.’ Developers may easily add more VEM types which make particular sense for their application. The core idea of the VEM is to allow Software Immersion developers to add in metadata to each particular piece of data which will aid in the presentation of that data to the users of the system. A particular aspect of the VEM to note is that it does not define, in any way, how the data should be presented to the user; the metadata maintained by the VEM act simply as ‘hints’ for the next layer(s) of the framework to work with.

The final task in defining a Software Immersion with our framework is the creation of a Theme for displaying and accessing the data. In CHIME, the Theme is responsible for deciding how to display data (e.g. what a particular room looks like, what commands are available to a user for manipulating particular pieces of data, etc.) The Theme component acts as a sort of “late-binding” mechanism for deciding the look-and-feel of the resulting virtual environment. CHIME’s ThemeManager component is responsible for interfacing a particular Theme with the rest of the system, as well as handling collaborative aspects of the Software Immersion (including awareness of other users’ activities, modifications to the underlying data, chatting among users, etc.)

The final component of the CHIME framework is TreatyMaker, a lightweight toolkit for federation of network services. TreatyMaker is based loosely on the International Alliance metaphor described in [Ben Shaul, 1995] for federation of collaborating workflow systems. In CHIME, TreatyMaker makes possible the “alliance” of multiple organizations allowing seamless collaboration among project participants from all sites. Peer-to-peer federations as well as hierarchical federations are possible, with each site maintaining administrative control over their data and services.

CHAPTER 2 An Overview of CHIME

A successful technology creates problems only it can solve.
— Alan Kay

In this chapter, we will give an overview of CHIME, the framework built as part of this research, and describe the ideas encompassed within CHIME that form the core of this thesis. We will begin by briefly describing our previous work in this area, including our initial attempts at extending the functionality of a more traditional Software Development Environment (in our case, a Process-Centered SDE known as Oz [Ben-Shaul, 1995]). Next, we will describe our motivations in constructing a Virtual Environment framework for Software Engineering, detailing several requirements we formulated along the way. A discussion of related research areas follows. Finally, we conclude with an implementation overview of the CHIME framework and discuss some related areas of research.

2.1 Previous Work

Prior to the creation of the CHIME framework, our research into Software Development Environments (SDEs) focused on two main problems. First, how can we provide the user with access to project artifacts and other data which are not under the direct control of the SDE in use? Second, how can we provide lightweight access to the SDE, i.e. could we make it possible for the user to make use of SDE functionality no matter where she was located. In the context of this second problem, we later broadened our research goals to also look at ways of making it easier for developers to navigate among related project artifacts.

2.1.1 Oz and OzWeb

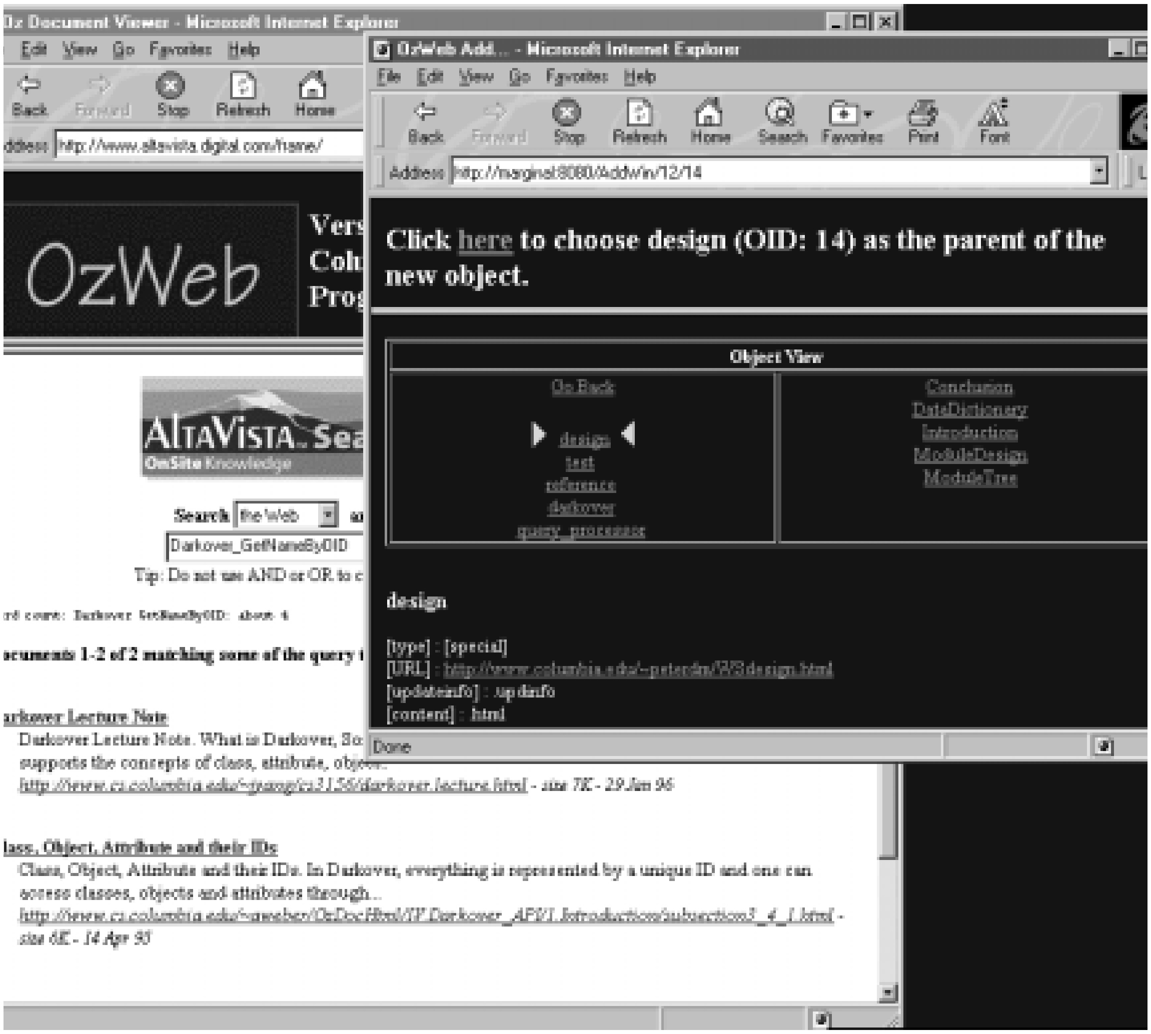
Oz [Ben-Shaul, 1995] is a Process-Centered SDE in which the software process in use is described through a set of rules governing the relationships among artifacts in the system. Users invoke rules as part of their daily work on the system, in order to edit code, interact with version control systems, write and update system documentation, etc. Oz rules are written in a high-level declarative style, in which the pre-conditions, parameters to the rule, and post-conditions (effects of the rule) are explicitly defined. Oz was a mature research system when this work began, supporting the daily programming and documentation tasks of our lab's mix of PhD students, MS thesis students, and varying numbers of undergraduate and graduate level project students.

In the fall of 1995, the WWW gained widespread acceptance. It had existed in one form or another for a few years (and of course could trace its roots back to the beginnings of the Internet), but by 1995, it became clear that the Web would soon play host to an immense amount of material which could be useful to Software Engineers. Software tool and library vendors began deploying all sorts of useful documentation and updates to their products online. Companies began to set up corporate intranets, essentially web sites inside their firewalls, as repositories of useful information for their employees. These sites became natural places to host project documentation, especially archival information on older projects, due in part to the ease of access afforded these documents through the use of a web browser.

Our initial efforts with the Oz system involved modifying its object oriented database layer (known as Darkover [Programming Systems Lab, 1995]) to allow it access to artifacts which were contained inside web servers. This was accomplished by creating a new built-in class type for the database, called WebObject. With all other data types, the database layer maintained the attributes associated with the object (e.g. its name in the database, who created it, etc.) as well as the content (e.g. the actual Microsoft Word document which held a testing plan). For WebObjects (and object types deriving from WebObject), the database maintained only the attributes and an associated URL; the system also provided built-in operations for reading and writing this content (analogous to HTTP’s GET and PUT operations) when needed. The system went one step further and provided caching mechanisms, with the aim of reducing the number of web requests made for objects which were frequently used.

Once this work was completed, we turned our attention to the question of lightweight access to the SDE. Our lab had done research in the past on low-bandwidth connections to the SDE [Skopp and Kaiser, 1993], but these efforts focused mainly on modifying the Oz system client to pre-cache data and objects the user might need to use while disconnected (or connected via slow-speed dialup links). While this was a useful line of research, it did not truly address the problem of lightweight access to the system. Users still had to have access to computers powerful enough to run our Oz system client (at that point, a Sun Workstation system running the Solaris operating system), as well as having to have the Oz system installed and properly configured on this computer.

In response to this, we focused on creating a Web-based interface to the Oz system, called OzWeb [Kaiser, et al., 1997]. At the time, we felt that a web interface to the system provided the utmost in flexibility. A user could log into the system and work with project artifacts from any web browser (subject, of course, to firewall constraints and such.) OzWeb was realized through the creation of an additional code module, embedded in the Oz system server, which acted as an HTTP server. Web browser users would connect to this server as though it were any other web server available on the internet, and would be shown an HTML-based user interface to the system. As part of the OzWeb work, we built a web-based tool invocation platform, called Rivendell [Valetto and Kaiser, 1995], which allowed web users to launch external tools on system data as part of their work (the original Oz system client launched tools on behalf of users, but in general this cannot be done from a web browser without special configuration). Figure 2- 1 has a screenshot of the OzWeb HTML-based user interface.


Oz Document Viewer - Microsoft Internet Explorer
Edit View Go Favorites Help
Back Forward Stop Refresh Home
Address
OzWeb
AltaVista Search
OnSite Knowledge
OzWeb Add... - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites Print Font
Address
Done

http://www.altavista.digital.com/home/

Vers
Colu
Prog

Search the Web

Darkover_GetNameByOID

Tip: Do not use AND or OR to c

rd count: Darkover GetNameByOID: about 4

ocuments 1-2 of 2 matching some of the query t

arkover Lecture Note
Darkover Lecture Note. What is Darkover, So
supports the concepts of class, attribute, object.
http://www.cs.columbia.edu/~jyang/cs3156/darkover.lecture.html - size 7K - 29 Jan 96

lass, Object, Attribute and their IDs
Class, Object, Attribute and their IDs. In Darkover, everything is represented by a unique ID and one can
access classes, objects and attributes through...
http://www.cs.columbia.edu/~aweber/OzDocHtml/IV.Darkover_API/1.Introduction/subsection3_4_1.html - size 6K - 14 Apr 95

http://marginal:8080/Addwin/12/14

Click here to choose design (OID: 14) as the parent of the new object.

Object View	
Go Back design test reference darkover query_processor	Conclusion DataDictionary Introduction ModuleDesign ModuleTree

design

[type] : [special]
[URL] : http://www.columbia.edu/~peterdm/WSdesign.html
[updateinfo] : updinfo
[content] : html

Figure 2-1: OzWeb placed a WWW browser front end on top of the Oz Software Development Environment to make access to Oz more lightweight.

2.1.2 Xanth 1.0 and the Groupspace Controller

With OzWeb, we had an SDE which could seamlessly include both local and Web-based artifacts (i.e. access to Web-based artifacts on the back end) as well as allow access to these artifacts from specialized client software as well as from generic Web browsers (i.e. access on the front-end). We soon realized, however, that this was a specific example of a more general solution. Wouldn't it be great, we thought, if we could access data and project artifacts living in any back-end information system, not just web servers? Further, wouldn’t it also be great if we could seamlessly allow access to (and provide workflow, transactions, and other collaboration services on top of) this information from clients speaking any protocol, not just our own Oz protocol or HTTP-speaking Web browsers?

Thus the Xanth and Groupspace Controller projects were born. Xanth was built in response to two main goals: first, to generalize the data access mechanisms from the OzWeb project to provide access to data stored in potentially any information system. Second, to provide that access to clients speaking potentially any protocol. Thus Xanth acted as a sort of switching mechanism, allowing clients written for a particular client-server information system to access data stored not only in their native server but also potentially in many other servers.

Xanth accomplished these goals through a modular architecture in which protocol-specific components could be configured into a Xanth system at runtime. Access to back-end data stored in myriad information systems was provided by a variety of Data Access Modules (DAMs). On the front-end, modules known as Protocol Access Modules (PAMs) allowed client software from many different client-server systems to connect to Xanth and access data provided by the DAMs. As part of the Xanth project, we built a variety of PAMs and DAMs which handled well-known internet protocols and information systems, including modules for HTTP, NNTP (the protocol used for connecting to Usenet News servers), FTP, CVS (the freeware software configuration management system), Chimera 1.0 [Anderson, et al., 2000] (a hypermedia system developed by a research group at U.C. Irvine), email (which allowed commands to be sent in to the server by email and responses mailed back), and others.

The Groupspace Controller was a related project which acts as an event bus and service stack. This allows us to layer services on top of data accesses passing through a Xanth server. The Groupspace Controller (GC) has at its core a novel event notification and service stack which goes beyond standard forms of event notification and allows both pre- and post-event request and notifications. This allows services connected to the event stream to propose and veto actions proposed by other services. This is accomplished by both pre- and post-event notifications. Figure 2- 2 shows the basic operation and architecture of the Groupspace Controller.

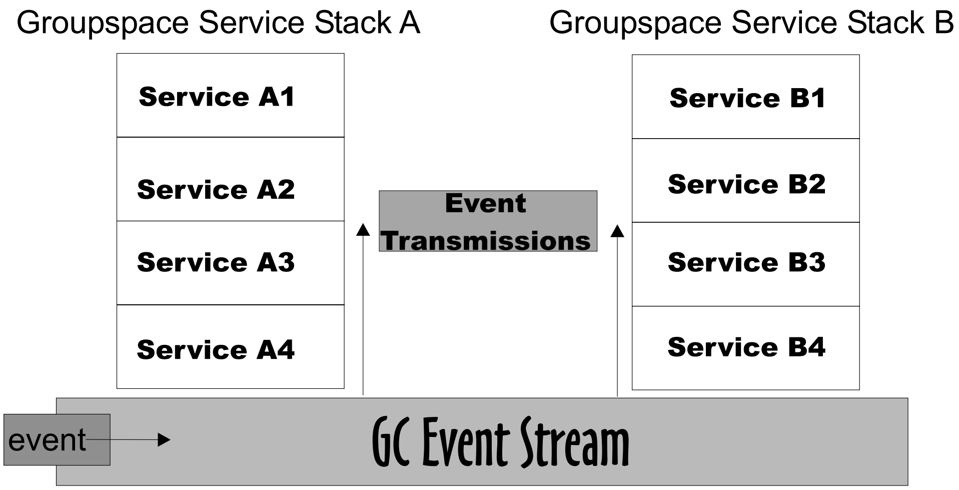


Figure 2-2: Groupspace Controller architecture. Events enter the system and are transmitted up the various service stacks. Higher level services may veto actions of services below them in the stack.

Groupspace Services are built as modules (loaded at runtime) which can connect to the event stream of the Groupspace Controller and subscribe to event types they are interested in receiving. Services are not necessarily peers as they would be in a standard event bus. Rather, they are configured into a service stack, with events travelling up the stack in a predefined ordering (the order is defined when the service modules are loaded at runtime). Multiple services may be configured at each level of the stack. Thus Groupspace Services are not peers as modules are in a more traditional event bus system, but instead wrap lower-level services. Higher-level services can veto events sent up from a lower level service; this allows a workflow or transaction service to disallow access to a particular piece of Xanth data if such access is not appropriate given the current transaction model or workflow in use.

Groupspace Services can communicate in two additional ways (besides transmitting events to the stack). First, all Groupspace Services are required to define and implement a standard Groupspace Role. This allows, for instance, a standard role for a Transaction Manager to be defined. No matter what specific Transaction Manager actually gets loaded at runtime, other services can communicate with it via the standard TM role. Second, services may publish a list of commands they accept, a sort of menu of possible messages they will respond to.

The combination of Xanth and Groupspace Controller make a powerful team. Specifically, they allow us to build services which layer on top of access to data via Xanth. This allows us to provide interesting services, including the previously mentioned Workflow and Transaction Management services, as well as potentially layering things like hypermedia, annotations, and other services on top of the Xanth data. To show off the capabilities of these two interrelated server-side components, we built a prototype collaboration and software development environment dubbed XanthWorks. XanthWorks included a client which connected to a particular configuration of services in a Groupspace Controller and an extensible set of DAMs in a Xanth instance and allowed distributed users to collaborate using data from a variety of back-end information systems. XanthWorks formed the core of a successful demonstration of both Xanth and the Groupspace Controller to our research group’s main funding source at the time.

2.1.3 SubReality

By this point in our work, we had long since discovered the problems inherent in the various 2D representations of the data and services accessible inside an SDE. Oz’s graphical tree structure proved unwieldy once the number of objects in the system exceeded a fairly small limit. OzWeb’s HTML interface was an improvement, but it was too easy to “get lost” among the huge amount of information in the SDE. XanthWorks used a Microsoft Windows Explorer style interface, with a collapsible and expandable tree containing all the data elements. While this did allow us to work with more objects, this interface had similar problems as OzWeb.

SubReality was our first foray into the use of 3D representations for the SDE. SubReality was a VRML-based virtual world in which each code module was represented as a subway station; the textures and graphics used were taken from scanned-in pictures of the New York City Subway System. To travel to other code modules, users would get on board a subway train (trains regularly arrived and departed each station) and select another station from the subway map (which could be found in every car)

. Users interacted through a number of means. Simple text chatting was available between all users in a particular room, and messages (known as graffiti) could be left for offline users by scrawling on a message board. Other users were represented in the virtual world as avatars (graphical representations of the users); as they moved through the subway station their avatars moved on other users' screens. Figure 2- 3 shows a screenshot from the SubReality project.

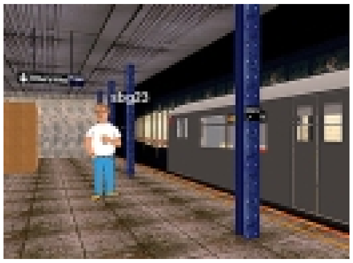


Figure 2-3: A screenshot from the SubReality project. Other users were represented as avatars inhabiting subway platforms. Users could take the trains between stations, each of which represented a different portion of the software project under development.

SubReality was a prototype in every sense of the word. The layout of the virtual world was entirely hardcoded to match a particular software project. The avatars were hardcoded into the world. The state of the virtual world was not saved across launches of the system; any changes made to the environment were lost as soon as the server was shut down. We quickly discovered the flaws of VRML (lack of a standard debugging environment, differences in implementation between web browsers, constant software crashes).

Despite its many flaws and limitations, SubReality allowed us to explore the possibilities 3D had to offer as the basis for an SDE. We quickly realized that the problems inherent in a 2D representation of a data hierarchy (those which we had encountered with Oz, OzWeb, and XanthWorks) could be at least partially addressed through the use of 3D techniques. In addition, we were pleased to discover the wealth of 3D user interface techniques which had been developed in the User Interface research community to that point. We felt that we were in a good position to make use of those techniques in an SDE.

2.2 Motivation

Based on our experiences with the SubReality prototype, we set out to create a flexible framework for creating SDEs which utilized 3D virtual environment techniques. This work was motivated in part by our desire to explore the use of what we later termed Software Immersion as an environment for work on SE projects.

We noted that large software projects tended to be poorly served by traditional Software Development Environments. With thousands (or hundreds of thousands) of artifacts relating to the project, finding needed information quickly becomes an ardurous task for the developers. Keeping up to speed with work going on in other (related) areas of the project becomes harder as the project grows. New development team members face a steep learning curve when coming up to speed on project history. In addition, non-programmers have not always been considered in the design of an SDE. Shouldn’t technical documentation writers, managers, and other users be given access to appropriate information on a project as it is being built? In addition, with literally hundreds of software publishers producing specialized tools which could be useful to a development effort, providing access to information and results stored inside these tools from within the development environment seemed potentially useful.

The concept of a Software Immersion (SI) was our initial answer to these issues. SubReality was our first experience in attempting to build a Software Immersion, and it helped us to crystallize the requirements both for SIs as well as for a toolkit to enable SIs to be created easily. We quickly realized that SIs were complex enough that one would not want to build them from scratch, but that having a toolkit which made it easy to build and customize new SIs to the needs of a particular SE project and development team’s working style would be a great help. We also realized that our goal of exploring SIs would be facilitated if we had a flexible platform on which to build. Further, the creation of such a framework seemed a good fit for our research groups’ typical system-building style of investigation, in which we built new systems to explore and experiment.

2.2.1 Requirements for our framework

The requirements we set forth for our SI framework reflect our research group's desire to quickly and easily build SIs for experimentation. We wanted to (as much as possible) factor out what we thought were the common building blocks of SIs, leaving it up the developer of a particular SI to customize the details of the resulting environment to the development project at hand. In addition to this level of flexibility, we intended our framework to be a platform for our research into SIs, so the ability to easily extend the capabilities provided by the framework was a must. Therefore, the following requirements must be met:

Heterogeneous access to project information. Based on our research experiences with Xanth and the XanthWorks prototype environment, we felt it was important that our framework make it easy for project information and artifacts stored in remote data repositories to be included in the resulting Software Immersion. With more and more relevant information stored in Web sites and other information systems, access to this information from the SI was essential.

Integration with 3d party development tools. Our SI framework had to make it easy for external tools to be integrated. This goes beyond the first requirement, which deals with accessing remote data. In this case, we wanted to at least open the framework to the possibility of white- and grey-box integration with external tools, where status information about what tasks the tools were working on, etc., could be communicated into the framework. Black-box integration, involving the framework simply launching an external tool, is of course straightforward and an easy requirement to satisfy.

Service layering. With our new framework, we hoped to continue the service layering model we had experimented with in our Groupspace Controller project. Specifically, our initial vision for the framework involved a set of services, starting at the lowest level with the information and artifacts making up the eventual SI, with higher level services layering on top of one another up to the final Virtual Environment layer. This allows each component to be self-contained; higher level components could use the services provided but were insulated from changes to the inner workings of a given piece of the framework.

Open to addition of new SI services. Our service layering requirement makes it easy for us to fulfill another of our requirements for the framework, namely that it be simple and straightforward to add in new services as their need became clear. We intended the framework to be used as a research platform, and as such it needed to be open as much as possible to the easy integration of new ideas. In addition, in the process of building SIs with the framework, a given development process might require the addition of services which had not been previously designed.

Separation of VE interface from SI services. With our new framework, we hoped to continue the service layering model we had experimented with in our Groupspace Controller project. Specifically, we felt that the Virtual Environments built with our framework should separate the specifics of the user experience (i.e. whether each component of the software project was eventually represented as a skyscraper or a cubicle in the resulting VE) from the other services and layers in the framework. This requirement allows designers to view the building of an SI as an assembly line; first, they focus on the data and artifacts to be used, next they focus on services layering above the data (including hypermedia services, collaboration services like group annotations, etc.), and finally focus on the specifics of the resulting VE itself (how the user interacts with the various artifacts, what tools are integrated with the environment, etc.)

Ability to explore non-3D SIs. This requirement was fairly important to us. Our ideas for Software Immersion borrow heavily from work done with MUDs (Multi-User Domains), many of which are entirely textual. Although our main focus was on designing a framework for SIs which exploited the power of 3D interaction techniques, we felt that nothing in our framework should prevent it from also being used to explore 2D graphical or purely textual representations for a Software Immersion.

Scalable from small to large projects. Although many of the initial goals we hoped to achieve with Software Immersions were applicable mainly to larger and long-lived SE efforts, we hoped that our framework could also prove useful in allowing us to investigate use of SIs for smaller projects. Further, we wanted the resulting SIs to be easy to deploy and maintain regardless of the number of developers being supported. It is not uncommon for hundreds or even thousands of developers in multiple organizations to be working on related projects (or components of a large project), and we wanted our framework to span the gamut of development team and project sizes.

Recognition of geographical and temporal distribution of users. One of the main benefits we hoped to achieve through the use of SIs involved smoothing the problems of geographically dispersed development. In addition, the design of the framework needed to recognize that because of the distribution of the users, some of them would be connected to the system via lower bandwidth or higher latency communications links. Further, we wanted to in some way address the problem of temporal distribution, to help a user catch up when he or she has been away from the project for a period of time.

Built on existing tools, techniques where applicable. Our final requirement was to build the framework around existing tools and techniques when possible. Our research group does not focus on 3D User Interfaces, thus we hoped to borrow as much as possible from the research results from that community. In addition, we did not want to reinvent the wheel while designing our framework. By using well-known tools and techniques inside the framework, we hoped to make it easier for a given development team to design an SI for their own use.

2.3 Implementation Overview

CHIME was implemented entirely with the Java programming language. A number of factors motivated this decision, although our chief concern was the eventual ability of the system to run on multiple platforms. In addition, we intended to build our framework partially on top of existing systems we had built as part of past research (including the Groupspace Controller and Xanth projects), and these components had already been built using Java. Finally, Columbia's Computer Science Department had recently switched entirely over to the use Java for nearly all the courses in its undergraduate curriculum. Because of this, we felt it would be easier to find and keep talented research project students to work on CHIME-related tasks if we used a programming language they were already familiar with and excited about.

For the 3D components of the system (the client which would display the Virtual Environment to the user), we went through a number of iterations. Our first attempt involved a commercial 3D game engine (the Quake II engine [Id Software, 1997]). This engine was built to be extended and modified at runtime by sophisticated end users, and extensive documentation was available both from the original developers as well as a diverse collection of online gaming sites. Since the engine was divided up into two main layers (a low-level graphics primitive layer responsible mainly for network communications and display of the virtual world, and a separate game logic layer which provides the 3D models and determines how the user interacts with the world), we felt we could simply replace the game logic layer with a module which provided the Software Immersion experience for the user. In addition, this engine was available on a number of different operating systems and hardware platforms, which satisfied our goal of supporting multiple platforms.

Unfortunately, it turned out that the Quake II engine was not flexible enough for our needs. Specifically, the map or layout of the virtual world needed to be determined ahead of time; for our purposes, a constantly changing virtual world layout which reflected the changing makeup of the underlying software systems was needed. As a result, we moved to using the OpenInventor 3D library, originally from SGI but now available for many operating systems (and with a programming interface for Java). OpenInventor proved to be a good choice on a number of levels. First, as an extremely flexible programming framework, we were able to quickly develop a system client for CHIME. Second, a number of books have been published on the use of OpenInventor, which makes the task of training a project student or other user of the system to build their own 3D environments simpler.

2.4 Related Research Areas

In this section, we will discuss a number of research areas generally related to the work described in this dissertation. More detailed descriptions of related work will be presented in the chapters to which they are directly relevant.

2.4.1 Software Development Environments

Software Development Environments have long been a focus of attention in the Software Engineering research community. In [Reiss, 1996], the author defines a Software Development Environment as “a collection of tools and capabilities designed to simplify programming and thus enhance programmer productivity.” SDE research has involved a number of different directions, including single-user environments, multi-user environments designed to support a team (or teams of teams) developing and designing software, process-centered environments designed to guide programmers along a well-defined development process. SDEs have emerged as popular commercial tools as well; millions of programmers use Microsoft's Visual C++ and Sun's SunWorkshop development environments to accomplish their work every day.

Nearly all recent SDEs have allowed for some form of integration of external tools into the environment. While early SDEs often attempted to provide all the tools needed for development, this approach proved be quite inflexible; if the development project requirements change in ways not originally imagined by the SDE developers, the SDE may no longer meet the needs of the developers. Tool integration in more recent environments has typically followed either a data integration or a control integration paradigm. With data integration, one assumes that it is possible for the constituent tools to share information. Project data is often stored within a central repository or database maintained by the SDE, which can then mediate access and modification to this data. Control integration involves individual tools sending messages to one another (often on some kind of Event Bus) to provide coordinated access to SDE functionality. PCTE [Boudier et al. 1989] and FIELD [Reiss 1995] are two examples of environments whose primary mechanism is control integration.

The CHIME framework utilizes both forms of integration. Control integration is accomplished via the Groupspace Controller component. By making it possible to layer services atop one another, the Groupspace Controller extends the event-based integration techniques pioneered in previous research into SDEs. On the data integration side, the Xanth Data Integration component makes it possible to access (and potentially modify) external tools' data in a uniform fashion.

2.4.2 Virtual Environment Systems

Virtual environment systems are computer-mediated spaces designed to facilitate human-human or human-information interaction, especially when one of the parties is geographically removed from the other(s). Virtual environments can be entirely text based, can involve 2d "talking heads" (as in a virtual conferencing system like PictureTel or Microsoft's NetMeeting), or may involve sophisticated 3d graphics. In all cases, the goal of the VE system is to give the user the immersive feeling of being in a virtual room or space defined entirely by the computer.

Multi-User Domains (MUDs) are text-based chat environments which usually include a number of interlinked rooms within a virtual world. Users in a particular room chat with one another by simply entering text; whatever the user Joe types is repeated to the entire room as 'Joe says xxxx.' The first MUD was introduced in North America in 1991 by Pavel Curtis at Xerox PARC [Curtis, 1992]. Despite the lack of "fancy" graphics techniques, text-based MUDs can be quite immersive; user studies (see, for instance [Darken 1995]) have shown that users of text-based virtual environments report thinking of the MUD as a physical space they can visit and meet up with “friends” they may have only met online.

Collaborative virtual environments are a significant subset of the body of research into VEs, and focus specifically on allowing participants to collaborate over a set of shared information to accomplish work tasks. Systems like MASSIVE [Greenhalgh and Benford, 1995] and DIVE [Carlsson and Hagsand, 1993] are representative of work in this area. Both systems allow multiple, geographically distributed participants to interact in a 3d virtual environment.

Most of the research done in the area of virtual environments has focused on the interaction techniques used by participants. As such, little of this work has involved more practical issues, including how to incorporate external tools and data into the virtual environment. At best, these environments allow the incorporation of external documents which are then stored in the virtual environments’ own repository or database. Thus the tool no longer has control over the data; a copy is now stored inside the environment. This is a key distinction from the CHIME framework; environments created with CHIME can easily include external data and services in the environment. While CHIME environments do not currently include all the rich interaction possibilities afforded by systems from this area (videoconferencing within the environment, for instance [Benford and Fahlen, 1994]), the CHIME framework provides a good baseline from which these features could be added easily.

Finally, in the arena of Virtual Environments, the Promo system [Doppke, et al. 1998] has overall goals which are extremely similar to the CHIME framework Promo attempts to automatically create a MUD-based virtual environment from a software process description. Users can then interact with one another inside the MUD; rooms in the MUD correspond directly with work tasks from the software process. Exit conditions must be satisfied for the current task before the user is allowed to exit the room to an adjoining room (work task). CHIME is not tied so closely with software process enactment, but the overall goal of applying MUD and Virtual Environment techniques to create a useful and immersive software development environment is similar to CHIME’s Software Immersion goals.

2.4.3 Distributed Groupware

A number of research toolkits and systems have been developed to enable the creation of distributed, multiuser groupware systems. Groupware systems typically focus on allowing users to share documents, easy communications within the environment, and on providing awareness to users of each others’ current work focuses. The sheer number of research systems developed in the Groupware space prevents us from giving a complete accounting of related systems. However, we have selected a small number of systems whose goals or areas of interest are similar to portions of the CHIME framework.

GroupKit [Roseman and Greenberg, 1996] is a toolkit for designing synchronous and asynchronous Groupware applications. GroupKit is similar in overall design to the CHIME framework; each major component of the system is responsible for a well-defined set of services, which are made available to users of the toolkit independently. In this way, GroupKit makes it easy for Groupware authors to use only the functionality they desire from the toolkit.

Orbit [Fitzpatrick et al., 1998] is a Groupware system based loosely on principles from Sociology describing how people best work together (their work is based most closely on the research of Anselm Strauss [Strauss, 1993]). The central concept of Orbit is that of a locale; each locale in a given collaborative space represents a single area of work. Multiple individuals may participate in each locale, and the locales contain shared documents and other project artifacts. Orbit tries to promote awareness of the activities of other users as a central concept, and uses a color coded set of interest criteria to allow each user to specify his or her level of interest in a particular locale. Orbit's goals are closely related to the CHIME framework concept of Groupspaces. However, a chief difference is that Groupspaces recognize the need for project artifacts and data to remain at their original locations inside whatever particular information system they came from. Orbit requires (for collaboration purposes) that documents be relocated inside the Orbit server.

TeamRooms [TeamWave Software, 2000] is a synchronous collaboration system (i.e. all users must be online at the same time) built around the concept of share project repositories. Each room in the project represents a different portion of the project or tasks to be accomplished. This is conceptually similar to the locales concept from Orbit. TeamRooms supports a unique collection of interaction styles, including “mingling,” in which users may use any documents and collaborate with each other in any way they please, and “lecture,” in which one user or a small group of users can control the displays of other participants. This is extremely useful as a setting for distance learning, etc.

2.4.4 Software Visualization

Research into Software Visualization (and the related area of Algorithm Animation) looks at the design and development of techniques to show program code, algorithms, and data structures by using typography, graphics, and animation. The Software Immersion in our conceptual model for CHIME can be seen as a form of Software Visualization, as we are displaying the organization of software artifacts through the design of a virtual environment. [Stasko et al., 1998] contains a good overview of research in this area.

Typical of Software Visualization is the requirement that the visualizations be hand built for each software project. When visualizations have been built automatically, they have been reflections of, for instance, memory usage of a program over time or call stacks representing internal function calls. While these kinds of visualizations can be useful in debugging or profiling a given software system, in order to provide more useful support to software engineers, the visualizations must be tailored by a human to the particulars of the project at hand. The POLKA system [Jerding et al., 1997] is representative of recent efforts in the Software Visualization space.

CHAPTER 3 Software Immersion

It’s never too late to learn, but sometimes it’s too early.
— Charlie Brown

Earlier in this dissertation, we touched upon the models which underlie the CHIME framework. In this chapter, we will give an overview of these models, Groupspaces, Groupviews, and Software Immersion, and discuss their relationships. As you will see, these models, when applied in combination, allow us to define and build powerful Software Development Environments (SDEs). This chapter proceeds as follows: first, we describe the interrelated concepts of Groupspaces and Groupviews which underpin Software Immersion. Next, we describe the Software Immersion model in detail. Following this, we describe a thought experiment involving the creation of a Software Immersion for a fictitious software engineering effort at a major engineering firm. This thought experiment is used to further describe the details of Software Immersion. Finally, we discuss some related work, mainly from the Software Development Environment research area.

3.1 Groupspaces

We use the term Groupspace to describe a persistent collaborative information space that provides access to all the information related to a project or set of interrelated projects. In defining Groupspaces, we recognized that projects involve information stored in many different tools, formats, and repositories; only the smallest project (whether it be a Software Engineering effort or any other kind of project) can be fully described and worked on using a single tool. It is not uncommon for a project member to use 5 or 10 different tools or systems to accomplish a single task (for Software Engineers, these tools might include compilers, debuggers, regression testers, development environments, software architecture description tools, etc.). Thus Groupspaces were designed around the concept that much of a project’s data can be accessed only via a variety of information systems protocols (in the case of client-server tools), and is stored in a variety of formats. Further, we realized that it would be a mistake for a Groupspace to attempt to recreate the tools used to access data from these back-end sources; although the result might potentially be a more integrated solution, it seems silly for someone setting up a Groupspace of project data to attempt to recreate fully the capabilities, for instance, of a well-written rationale capture tool. Because of this, the Groupspace model allows for (and encourages) continued use of existing tools to access and work with project artifacts, simply redirecting this access to travel through the Groupspace rather than directly connecting to the information system. In this case, the Groupspace acts as an Information Brokering or Middleware component, mediating access between a tool run by a user and the information repository containing the data of interest. This is a fundamental change from the models underlying most traditional project environments, including many Software Development Environments. With many of these systems, all project data must be owned by the development environment. While it is in many cases possible to import existing data in a variety of formats into a development environment, once done, this data falls under the exclusive control of the environment itself, not the information system which originally held the data.

In addition to organizing project data regardless of its true back-end location, Groupspaces were designed to add useful collaborative services transparently on top of this project data. Many tools which are used to accomplish project tasks have either limited or no collaboration support. Collaboration and coordination facilities including Workflow, Transaction support, Hypertext and Hypermedia support, Collaborative Annotation support, etc., can be layered on top of access to project data through the Groupspace via the inclusion of Groupspace Services.

In short, a Groupspace represents the entire universe of information available to project members. Figure 3- 1 shows the relationships among the components of the model. On the far left hand side, we see the various information services which contain relevant project data or which perform tasks for the project (i.e. back-end information services may be simple data providers or may be brokers -- potentially other Groupspaces -- which perform computations on data on behalf of users). In the middle, we see the Groupspace itself, which includes the Groupspace Service layers adding on functionality atop access to the data. These services may combine to provide higher levels of functionality. For example, the Transaction and Workflow layers may work in combination to provide Extended Transaction Model support to a particular running Workflow task which involves Groupspace data. Finally, on the far right side, we see the existing system clients and new Groupview (see below) clients which are used to access project artifacts in the Groupspace.

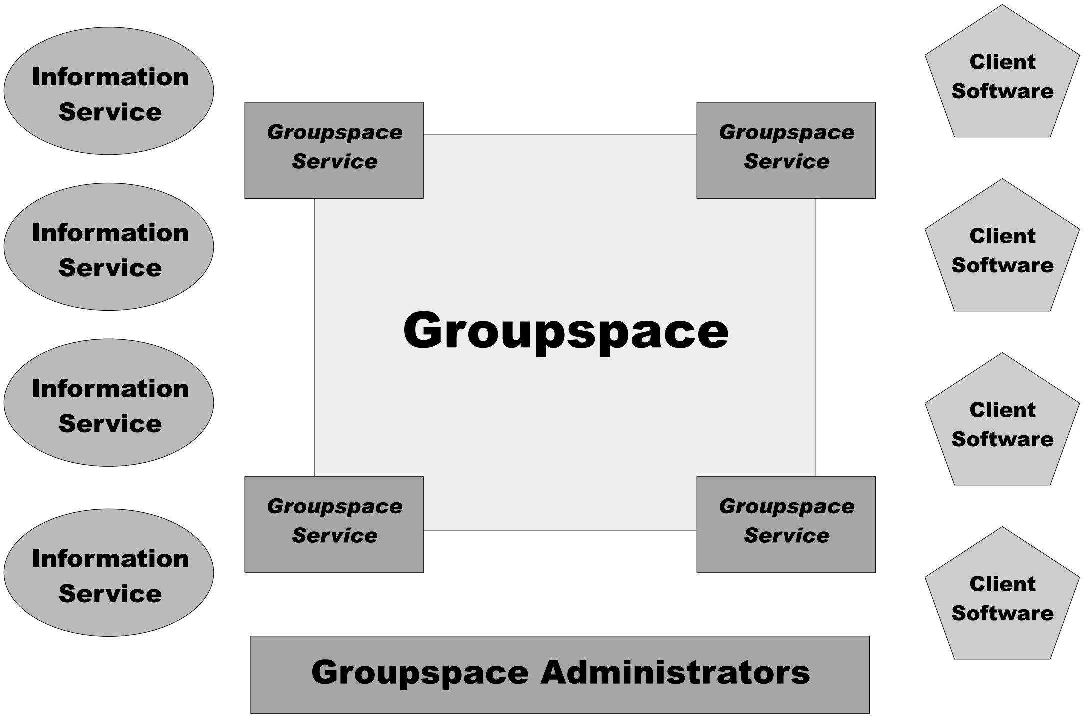


Figure 3-1: Components of a Groupspace.

Perhaps the most important element of the Groupspace model is shown at the bottom of Figure x. The Groupspace Administrator is the person (or group of people, for particularly large Groupspaces) responsible for getting all the relevant project data into the Groupspace initially as well as deciding on (and possibly building) the set of Groupspace Services which are to be available to project members. The role of the Groupspace Administrator is critical; without this person, the Groupspace does not exist. While it would be wonderful to claim that Groupspaces could be built automatically, this is unfortunately not the case.

As you can see, the key aims of the Groupspace model are to organize project data in any way useful to team members as they carry out their work, as well as to augment the capabilities of their existing tools with new collaborative services. The organizational aspect of Groupspaces is accomplished through the ability to operate seamlessly on data from multiple back-end information systems and repositories using existing tools. The capability augmentation aspect is handled via new Groupspace Services, whose benefits may be fully utilized through the use of new tools and access mechanisms (see Groupviews below). Thus the main benefits of the Groupspace model are that it collects all relevant project artifacts in a single place, allows users to continue using existing tools to access the artifacts, and allows for the addition of new services by layering them on top of project artifact access. Oftentimes these new services will require new user interface elements in order for project members to utilize them. The next section will describe the Groupview model, which is to a certain extent the sibling of the Groupspace model that allows these new elements to be displayed to the user.

3.2 Groupviews

As we mentioned above, a Groupspace organizes and manages access to all relevant project data. This makes possible the layering of additional services on top of access to this data (called Groupspace Services). In order to make the most use of these services, new user interfaces are often required. We collect these new user interfaces and tools, as well as a number of other user interface requirements, into the Groupviews model. Groupviews are designed around concepts pioneered in Collaborative Virtual Environments, sometimes called MUDs (Multi-User Domains) or MOOs (MUD Object Oriented). A small group of researchers has recently begun studying the use of MUD- and MOO-style environments for “real work,” [Poltrock and Engelbeck, 1997] and have reported a number of advantages through their use in situations where project members are geographically or temporally distributed.

Groupviews include new tools and user interface mechanisms used to make the facilities offered by Groupspace Services (see above) available to the users of the system. While Groupspaces do allow continued access to project data with the original tools used to access that data (e.g. web browsers to access web pages, requirements analysis tools accessing proprietary repositories, etc.), new tools are potentially needed to access the new features provided by Groupspace Services.

Groupviews provide the user with coordinated access to these capabilities through intuitive collaborative interfaces which emphasize the team aspects of the work being performed.

Groupviews give collaborative access to data contained within a Groupspace. As we mentioned above when discussing the Groupspace model, many of the tools and information systems which might potentially be used through a Groupspace do not contain full support for collaborative work. Groupviews add a number of collaboration features on top of this data. In addition, Groupviews can provide access to new capabilities provided by Groupspace Services.

One of the most difficult problems faced by team members working on a project is coordinating work among themselves. Thus an important facet of the Groupview model involves allowing users to keep up to speed with what other users are doing in the system. Of course, when users are working on multiple projects at a time (not an uncommon occurrence), they often want to focus more on a single project at any given point in time. So another requirement of the Groupviews model allows users to set a “volume level” on each subproject they’re working on; they will not be inundated with information on projects set to a lower volume level.

When a new team member joins an ongoing project, there is a time period in which they are less productive because they need to spend a lot of time coming up to speed with the projects’ history. The Groupviews model addresses this issue by including the ability for team members to play back the events which have occurred on the project. When a new member joins the project, or when a team member returns after an absence (vacations, etc.), the Groupviews model makes it possible for them to catch up.

Finally, the Groupviews model requires that it be possible for users to view Groupspace data from multiple levels. For example, managers might view projects from a “50,000 foot” view which shows only the major accomplishments and milestones of the project. Team members should be able to jump to and from different levels, zooming into and out of the details of a particular portion of the project.

3.3 Software Immersion

Software Immersion, then, is an outgrowth of the previous two models. Perhaps the easiest way to begin to explore Software Immersion is to understand it as the combination of both the Groupspace and Groupview models along with certain modifications designed to be applied specifically to Software Engineering projects.

Because of its relationship with the Groupspace and Groupview models, Software Immersion combines features of Software Development Environments, Middleware/Information Broker systems, Collaborative Virtual Environments, and to a lesser extent, Software Visualization systems. In a Software Immersion, software engineers are immersed into a virtual environment made up entirely of artifacts (and the relationships among those artifacts) from the development project they are working on. This differs from Software Visualization in that Software Immersion does not attempt to illustrate the details of a particular software component or algorithm, but rather to detail the relationships among the artifacts involved in a development project. Software Immersion attempts to aid developers in finding the information they need to carry out their daily development tasks.

Software Immersions are intended to be built semi-automatically. The layout of information from the underlying Groupview is influenced by factors specific to the Software Engineering domain. For instance, although regression test results and the source code they are intended to test may be accessible only through distinct repositories (source from a configuration management system and test results from a server side component of a testing tool), these two interrelated pieces of data might be located directly next to each other in the Software Immersion. Of course, it should always be possible for the users of the Software Immersion to customize the layout and move objects around in any way that they see fit.

As might be inferred from the previous example, in a Software Immersion, the architecture of the underlying system defines the layout of the virtual environment used by team members. Team members collaborate and perform individual tasks in this virtual space. Research in the Collaborative Virtual Environment community focusing on use of CVEs for “real” work [Benford, et al., 1997] has shown the advantages of laying out a virtual environment according to principles and locations users might know from the real world. By immersing developers and other project members (managers, testers, etc.) in a virtual world built from the architecture of the underlying system, we attempt to reify the system architecture. Through this, we attempt to apply as many of the benefits of spatial layout as possible to the daily development of the system. In addition, in a Software Immersion, the layout is dynamically changing as users work on project tasks. For instance, the addition of a new code module may trigger the addition of a brand new room in the CVE, accessible via other modules it interacts with.

3.4 A Thought Experiment

In order to make our discussion of these models a bit more concrete, we will take the reader through a thought experiment involving the creation of a Software Immersion at a major (fictitious) engineering firm, FooCo, Inc. FooCo is looking for any advantage it can get for its development staff, as competitor BarCo has recently been getting products to market faster and with fewer defects. In addition, FooCo has its hands full with continued development on its existing line of products, as well as on the new FooCo.COM e-commerce web site (which will be spinning out into a wholly-owned subsidiary via an IPO later this year).

FooCo’s development team has decided to try augmenting their existing software development environment (a patchwork of home-grown utilities and commercial tools) with a Software Immersion to pull their projects together. Like most development staffs, FooCo constantly faces “churn” among its development team members. (Churn, the rate at which technical employees leave and are replaced at a company, has reached 40% per annum in some parts of the country [Loomis, 2000]). Product development schedules and defect rates suffer as a result of this constant loss of corporate memory about current and past projects at the company. In addition, when new staff are hired, they face a steep learning curve in coming up to speed on the details of their new projects. Further, FooCo has recently opened a new office in Los Angeles, complementing the existing ones in Silicon Valley and New York. Coordinating work among these teams has become more of a problem.

As we mentioned earlier, FooCo’s current development environment consists of a number of different tools. Fortunately, most of these tools are client-server and use published protocols to communicate, so fitting them in to the new Software Immersion they’ll be creating shouldn’t be too hard. They are particularly reticent about replacing any of the component development tools they use; development staff as well as less technical team members like managers, documenters, and other workers have been trained to use these tools and do not have the time to relearn new systems to accomplish their work.

At FooCo, the developer tools in use include CVS (the public-domain configuration management system), which was used on a number of past development efforts. Developers still refer occasionally to code stored in CVS, and so making things stored in their CVS servers available in the Software Immersion is a must. Rational ClearCase is in use for newer projects' configuration management needs. A number of different internal and external Websites contain documentation on some of the code libraries built in house or licensed from third-party vendors. The GNATS bug tracking and auditing tool is used both by the technical support people at FooCo as well as by the internal developers. GNATS, an extremely powerful tool with a cumbersome interface that only a programmer could love, is a tool ripe for use through a Groupspace. FooCo has recently begun using Rational's Testmate and PureCoverage systems as part of the testing procedures for new code and modules being written. Both Testmate and PureCoverage have open APIs which make integration into a Software Immersion straightforward.

Like most organizations, FooCo uses a variety of standard productivity applications for documentation needs as well as project and time management. Documentation is written using Microsoft Word; the resulting files are stored in a Microsoft Office Extensions Server repository. Since Office Extensions is a WebDAV [Goland, et al., 1999]-compliant document repository, integration into the new Groupspace will be fairly easy. The team also uses Microsoft's Project tool to schedule various phases of development. These schedules are also stored on the Office Extensions server.

3.4.1 Steps to creating the Software Immersion

Now that we have a clearer picture of the universe of data which FooCo's development staff would want to include in a Software Immersion, we can discuss the process through which the Software Immersion would be created. Much of this is a manual process, but in a number of steps the modular nature of Software Immersions will allow the creators of the SI to easily reuse existing modules (Groupspace Services, etc.) created for other Software Immersions in defining their own SI.

The first step involved in creating FooCo's Software Immersion is identifying the data and tools which will make up the underlying Groupspace of the Software Immersion. The data to be included needs to be organized into two subcategories: information stored in simple files, and information accessible only through some kind of information repository. For things in a repository, the Groupspace Administrator (the person responsible for setting up and maintaining the Software Immersion) must determine if the network protocol used to communicate with that repository is "open" (i.e. if documentation is available for the protocol or not). Obviously, including simple files into a Groupspace is much more straightforward than including information from a back-end repository. Fortunately for FooCo, all of the products they use for development which involve back-end repositories have open protocols. A non-scientific survey of the leading commercial development tools (performed in February 2000) shows that the vast majority of them utilize documented protocols, as it makes it easier for customers to perform just the sorts of integrations FooCo will need to do to get its Software Immersion up and running.

Once the Groupspace Administrator has identified which protocols are needed to access the repository-based information resources, their next task is to go ahead and develop small code modules to plug into the Groupspace which “speak” these protocols and mediate the process of accessing the data. This is not as onerous a task as it sounds. Many COTS tool vendors make extensive libraries and documentation for integration available. In the case of FooCo, all the tools in use are either based on Internet-standard protocols (like HTTP [Fielding, et al., 1999], WebDAV extensions [Goland, et al., 1999], CVS, etc.) or on protocols for which vendor libraries will make the task of integration easier (Rational’s ClearCase). In addition, a library of code modules which can access a variety of back-end information systems has been developed as part of the research described in this dissertation. In some cases, it may be possible to simply reuse these modules in the construction of a new Software Immersion.

The next step in the creation of a new Software Immersion is to decide what, if any, Groupspace Services to incorporate into the environment. As we mentioned above, FooCo has recently opened a third office in Los Angeles, complementing the existing New York and Silicon Valley offices. The addition of this third office has strained the abilities of developers in all three cities to stay abreast of work taking place in other locations. In order to combat this problem, the Groupspace Administrators have decided to add in some Hypertext services to the new Software Immersion, as well as some Collaborative Annotation services.

The Hypertext services add in typed, named, n-ary linking among project data in the Groupspace. This allows, for instance, project test plans to be cross linked with source code they exercise and source code to be linked with bug reports from the team’s bug tracking software. In this way, it is easier for team members to discover the relationships among project artifacts that may not have been readily apparent to them initially.

Collaborative Annotation services allow users to annotate project artifacts externally, i.e. without changes to the underlying data. Annotations are maintained separately from the back-end data, allowing this facility to be layered on top of all underlying data types. Annotations make it easy for project members to attach pearls of wisdom to various pieces of the project, along with suggestions of related information to track down or people who may be able to answer certain kinds of questions about various project modules. This unstructured, free-form type of information makes it easy for project members to include snippets of information which might not be easily incorporated elsewhere.

Following the selection of Groupspace Services to use within the Software Immersion, the details of the Virtual Environment paradigm to be used must be designed. There are a number of activities involved in this task, and because it can have a profound effect on the success or failure of the eventual Software Immersion, the choices made must be carefully considered. Each selection must be made both according to the details of the underlying process, data, or mechanism it represents as well as in the context of all the other choices made about the VE, with the aim of creating a unified Virtual Environment experience for the users.

FooCo is in the aerospace services industry. Their main product lines are hardware and software systems which provide a number of non-critical services to airline passengers, including in-flight movie and entertainment systems, airphone services, etc. Because of their airline industry-centric focus, they've not surprisingly chosen an airplane based metaphor for their Software Immersion.

Each project will be housed in its own plane, with various sections of the plane used as metaphors for the different types of project artifacts to be incorporated into the Software Immersion. All projects are represented as planes next to each other in a large airline hangar, which allows developers to easily "walk" among projects.

Inside of each airplane (the outside of each plane is decorated with the project name, logos, etc. to distinguish it from others), developers find that the passenger seats have been broken up into sections according to subprojects of the system being built. Each section contains code, design documentation, and bug reports relating to the system; the seats in each section are colored uniquely according to subproject. By sitting in a seat, a developer is given access to the full range of artifacts relating to the subproject. Developers can talk among themselves inside the Virtual Environment, and see which projects or tasks others are accomplishing. In effect, the full range of project data has been made available right at their fingertips, and communicating with team members (regardless of their actual physical location) is made easier. Hyperlinks among related subproject artifacts allow developers to easily “switch seats” and examine artifacts from other sections of the plane easily.

By sitting in the cockpit of the aircraft, developers can get a look ahead at development schedules, test plans, and performance analyses of portions of the project. In addition, the “radar screen” of the aircraft might display the location of this project in relation to others being carried out in the development organization (accessible through other airplanes in the hangar). The first class section of the plane, with its roomy seats, can be used for meetings of the development staff. As we mentioned earlier, FooCo’s development is being carried out in three different cities in the U.S., and so they wanted their Software Immersion to create a realistic virtual space which transcended the physical location of each individual developer. The goal is for each developer to feel like he works with all of the other developers in a common space, despite the actual geographical distances involved.

The final aspect to planning FooCo's Software Immersion involves choosing the layout mechanisms and details for the artifacts to be represented in the Software Immersion. We've already touched upon some of these issues, as they are directly intertwined with the choice of Virtual Environment metaphor to use. We have already mentioned that each project currently being worked on will be represented by an airplane in a hangar. The hangar will stretch in size to accommodate as many projects as necessary (FooCo rarely has more than 8 projects in progress at any given time, so the hangar will never really become overwhelmingly large). When a new project is started, it appears semi-transparent in the hangar. As the project nears completion, the level of transparency lowers, until at completion time the plane is 100% opaque. This allows developers from other projects, as well as managers and other non-technical staff to easily see how far a particular project is from completion at a single glance.

Older projects can be found by accessing a plane storage field through a door at the rear of the hangar. Project artifacts from these older projects are cross-linked with relevant information from current projects, making it faster for the developers to find artifacts (i.e. they are not forced to walk out of their projects' plane, out the back door of the hangar, and into a previous projects' plane every time they need an artifact; rather, they click and are transported directly to it).

Inside each plane, as we've previously mentioned, project artifacts are broken up into various areas of the plane. Subprojects inhabit sections of the passenger seating area; rows of seats are colored according to the project they represent. Larger subprojects inhabit larger sections of seating than smaller ones. A similar transparency mechanism is in use for subprojects; this makes it easy to tell at a glance how far a particular subproject is from completion. In addition, the walls of the plane and hangar can be decorated with information displays drawn from project status information, making it easy for all project team members to find out more details of a particular subprojects' status, tasks each member is scheduled to focus on, etc.

From this simple overview, it should be clear that there are many details involved in the creation of a helpful Software Immersion. In addition, it should also be clear that by immersing developers into a Software Immersion, we cannot solve all the problems faced by Software Engineering projects. However, we can aim to make it easier for team members to communicate, to know what tasks others are working on, to find out the status of related subprojects, and to find artifacts as they are needed.

3.5 Related Work

Related work falls into three main categories. Hypermedia systems have pioneered many of the hypermedia concepts embodied in Software Immersion. Software Development Environments have always tried to provide a complete environment for their users. Groupware researchers have provided a variety of collaboration models which contain elements similar to those found in CHIME. We will describe each of these in turn.

3.5.1 Hypermedia Systems

Hypermedia systems predate the World Wide Web by a number of years. Research into hypertext and hypermedia can be traced back to 1988 [Campbell and Goodman, 1988]. Two major varieties of hypermedia systems have emerged; hyperbase systems store the documents and project artifacts inside the hyperbase repository while link server systems store only link information, assuming that the actual documents will be stored elsewhere. This gives link server systems much more flexibility in terms of dealing with a variety of information types.

A number of research systems have been developed (both link server and hyperbase systems) which have investigated the kinds of hypermedia we have included within the CHIME framework. Typed, n-ary links among documents (in which links are treated as first class objects connecting a variety of information on both ends) have long been the standard in hypermedia research systems. These systems (including Chimera [Anderson et al., 1994], Hyperform [Will and Leggett, 1992], and others) have explored the rich variety of hypermedia functionality which can be layered atop a variety of information systems.

3.5.2 Collaboration Models

The Computer-Supported Cooperative Work (CSCW) and Computer-Human Interaction (CHI) research communities have produced a variety of models of collaboration. A number of these models are related to the models embodied within the CHIME framework.

In [Vygotskij, 1978], the author describes an early model of how humans work together though the use of “electronic communications.” This model is of course grounded in the limited technology of the time, but it provides interesting insights. This model is based around the concept of a work queue, in which participants use electronic means to select from a variety of tasks which need to be accomplished, receive instructions on where they can find any needed documents or information, and can report success or failure on their tasks. There is an interesting correlation between this models’ use of the work queue as the central repository of all knowledge and the CHIME framework concept of the Groupspace as the collector of all relevant project artifacts.

A much later contribution to the field describes the Locales framework for [Fitzpatrick, 1998]. A locale is a place for support of social world interactions. Each locale represents the information and “civic structures” (relationships among participants and locales) to all users. Each participant may inhabit a number of locales simultaneously, a common situation in the real world (it is often the case that a particular employee is involved in a number of ongoing projects at any given time). Locales also represent mutuality, a term used to describe the presence and awareness aspects of the locales which allow participants to easily discover the tasks of focus of co-workers. The Locales framework is aimed more at general office work than at the particular discipline of Software Engineering, and as such does not make the assumptions that the Software Immersion model makes regarding the underlying type of work it is meant to support. In particular, the Groupview concept of laying out a virtual world based on relationships among components of a project could not be included in a more general framework.

CHAPTER 4 Groupspace Controller, Xanth, and VEM

Education is ‘hanging around until you’ve caught on’.
— Robert Frost

Earlier in this dissertation, we described the Groupspace and Groupview models. These models allow relevant project data to be pulled in from back-end information systems and data repositories, collected in a single environment, and at the same time allow the application of new collaborative services on top of this data. The CHIME framework includes a number of different components in its architecture that help to realize these models. In this chapter, we will discuss three of these pieces. First, we will describe the Groupspace Controller, a novel event bus mechanism which allows services to wrap one another’s access to the event stream. In addition, Groupspace Controller supports both pre- and post- event notifications, which are necessary for a number of interesting collaborative applications. Next, we describe the Xanth Data Service, which is a Groupspace Service responsible for providing access to data stored in back-end information systems. Following this, we will discuss a component of the CHIME architecture which extends Xanth’s capabilities into the domain of Virtual Environments, called the Virtual Environment Modeler (VEM) service. VEM is a Groupspace Service (layered on top of data in the Xanth component) which handles tagging of Xanth data with Virtual Environment information. Finally, we discuss some related work in both the fields of middleware and data brokering systems, event notification systems, and virtual environment frameworks which deal with external data.

4.1 Groupspace Controller

As described in a previous chapter, the Groupspace Controller sits at the center of the CHIME framework, and underpins all CHIME services. The Groupspace Controller is a novel event bus via which hierarchies of event transmitters (called Groupspace Services) communicate. Unlike many event bus systems, events in the Groupspace Controller are not simply notifications that something has occurred. Instead, GC breaks the event notification process into pre- and post-event notifications. This allows services to “wrap” one another and effectively veto operations attempted by lower-level services.

The vision behind the Groupspace Controller was to extend the standard event notification and event bus paradigm so that it could be used to support complex collaborative services like workflow and transaction management. Only the simplest kinds of workflow can be built on top of an event notification mechanism in which the workflow engine or service is notified of changes to data or process state only after the change occurs. Instead, the workflow engine needs to be notified both before an activity is attempted (so that it may disallow actions prohibited by the workflow or process in use) and after an activity is completed (so that it can record the success or failure of the activity and maintain a complete picture of the current process state). Similarly, a transaction manager simply cannot do its job of assuring controlled access to data if it is only notified of accesses and modifications after the fact.

In the Groupspace Controller, events are first-class objects which are passed among interested services. The Event base class used by the Groupspace Controller contains a number of fields which may be used by services to describe the event which has occurred. The Event class contains a handle which allows it to be associated with a particular object in the larger system, a timestamp describing when it was sent into the system, contains information on the proposed change (for pre-event notifications) or information on the change that has taken place (for post-event notifications). The Groupspace Controller guarantees that events are delivered in the order they were generated. This makes certain kind of bookkeeping by the Groupspace Services unnecessary, since they can assume that they are processing the event stream in the order it occurred.

The biggest difference between the Groupspace Controller and more traditional event bus systems comes from the hierarchies of services, called Groupspace Service Stacks. In a traditional event bus, listeners attach to the bus to hear event notifications. Every listener is a peer; there is no mechanism for one service to prevent another service from performing some action. The event notifications transmitted on the bus are just that -- notifications. In the Groupspace Controller, rather than a standard event bus, we have one or more Groupspace Service Stacks. As you can see from Figure 4- 1, listeners (Groupspace Services) do not connect directly to the bus but are rather configured into service stacks. When a service generates an event, it first travels up to higher levels of the stack before being transmitted onto the general bus.

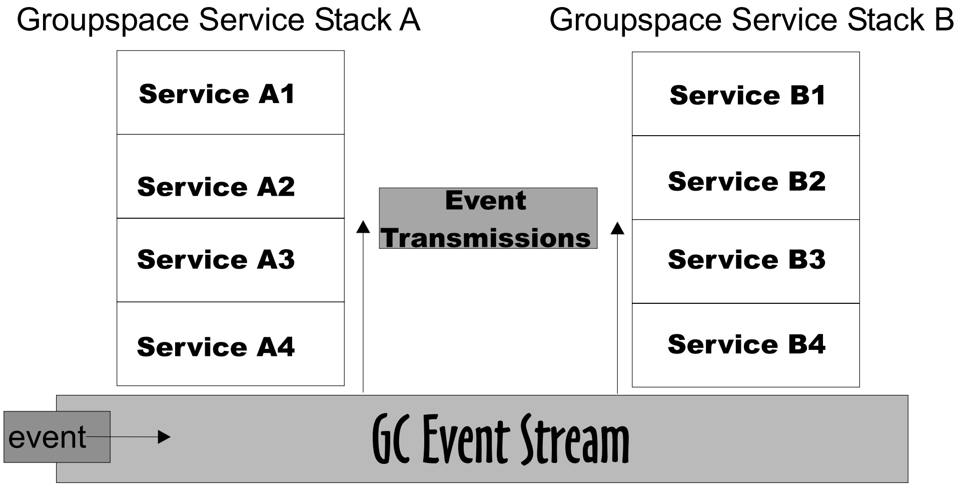


Figure 4-1: Groupspace Controller Architecture. Notifications travel up the service stacks; higher-level services can veto impending actions of lower level ones.

As implied above, the Groupspace Controller supports vetoable events. (Standard non-vetoable events are also supported for notification purposes). These events are broken into pre- and post- event notifications. After receiving a pre-event notification from a service below it in the stack, a higher-level Groupspace Service may throw a veto exception which halts processing of the event and notifies the origin service of the veto. The origin service is free to attempt the event at a later time; the event may then proceed successfully due to other changes in the system.

Services provide a list of event types they may generate and a list they respond to. These are separate from event subscriptions (which tell the Groupspace Controller what events a particular service is interested in hearing about at the current time), which may change depending on configuration of particular services, etc. Instead, this list is one of the ways the Groupspace Controller makes it possible for services to interact directly. In addition to this list of events, services may also provide direct access via a set of predefined functions or methods which other services may use directly. These direct-access mechanisms make it possible, for instance, for a given Groupspace Service to provide a list of the services it supplies to a user client, even if the client was not programmed with knowledge of this particular service. This comes in handy when we're attempting to layer services on top of legacy systems.

Every Groupspace Service must implement one or more Groupspace Roles. Each Groupspace Role describes a particular type of service which might be configured into a Groupspace Controller. The Groupspace Role defines the methods and data types exposed by a type service to the outside world (i.e. to other services). The underlying concept is that while there may be dozens (or hundreds) of implementations (or Groupspace Providers as we call them) of a particular service (say, a transaction management component), they all conform to a particular Groupspace Role. This makes it possible for other services to interact with the particular transaction manager in use inside a particular Groupspace Controller by programming to the TransactionManager Groupspace Role. This makes interactions among Groupspace Services extremely flexible. The Transaction Manager (or any other service) in use can be modified or replaced without the knowledge of other services which depend on it.

Services in the Groupspace Controller can be loaded and unloaded at any time. Runtime configuration and reconfiguration of Groupspace Service Stacks makes it possible to extend the capabilities of a Groupspace Service or a set of services at runtime with no impact on the running server. In addition, it is possible to reorganize the Stacks themselves at any time to change the hierarchical relationships among services.

4.1.1 Groupspace Service Example: A Collaborative Annotation Service

By way of explaining Groupspace Services in more depth, we will describe the inner workings of a Collaborative Annotation Service. The idea of this service is to allow multiple users of a collaboration system to comment individually about a particular piece of shared data. The service really does not need to know anything about the data the comments refer to. Instead, it simply needs to store the comment data (which could be anything from a piece of text, some HTML markup, or even a Microsoft Word document) along with a unique identifier for the data item the comment should be attached to. The idea of the annotation service is to provide a mechanism for free-form commenting and discussion relating to a particular piece of information.

What event types does our annotation service need to generate? For starters, it should generate vetoable events whenever a new comment is about to be added for a particular piece of data, as well as when an existing comment is to be deleted or modified in any way. Table x contains a more complete list of the events (vetoable and non-vetoable) an annotation service might generate. Services which might wrap the annotation service may want to restrict the ability for annotations to be changed based on external factors. For example, perhaps only particular users are allowed to comment on certain pieces of data. Perhaps deleting a comment is a task which is limited to certain administrators. Perhaps certain comments are only meant to be read by executives in an organization. This annotation service is meant to be simple and straightforward; it does not include any real concept of users or usernames (other than optionally storing a text string “username” along with each comment), or a particular permissions system which would allow it to determine how a given user was allowed to manipulate annotation data. By issuing vetoable events onto the Groupspace Stack whenever a particular event is about to occur, our annotation service makes it easy for this kind of functionality to be delegated to another Groupspace Service. An authentication service might be constructed for a particular application of the Groupspace Controller that would be responsible for determining which operations were allowed. A transaction service might disallow the operation because the underlying data item is somehow “locked” by another operation being performed.

Our annotation service might be interested in hearing other events as well. For instance, it might listen for an event stating that a particular piece of data has been deleted from some other component; when it received this notification, it would go through its list of annotations and remove any which were associated with the deleted content.

In addition to interacting with other Groupspace Services via event transmission and reception, our annotation service offers direct access to a number of its services as well. Through the “AnnotationService” Groupspace Role which it implements, other Groupspace Services connected via the Groupspace Controller can call methods in our annotation server. For example, these methods might be used to retrieve a list of the annotations which have been stored for a particular piece of data, or to retrieve a list of annotations stored by a particular user. Some of these methods may emit vetoable events onto the Groupspace Controller’s bus; the operation will only proceed if the event is not vetoed. This might be the case, for instance, with a method to delete a particular annotation.

As you can see, our annotation service is an extremely simple example of a Groupspace Service. It performs one task, and uses Groupspace Controller mechanisms to keep other interested services informed of its actions. Vetoable events are used to delegate functionality to other services which may or may not be configured into a particular Groupspace Controller instance, and it receives events to keep appraised of actions performed by other services. Our annotation service is an example of possibly the simplest useful Groupspace Service which can be constructed. It provides its services and has no real coupling to any other Groupspace Service.

4.2 Xanth Data Service

As we mentioned earlier, Xanth (named for the fictional Piers Anthony-created science fiction world) is a Groupspace Service which acts as a middleware layer to allow project artifacts to remain in their original location (i.e. stored and controlled by the information services which created them). The artifacts can then be accessed both via the original system client software and via new software which can take advantage of Xanth’s and the Groupspace Controller’s capabilities. As you can see from Figure 4- 2, Xanth sits in the middle and acts as an information broker, and knows how to retrieve the actual data from the origin server.

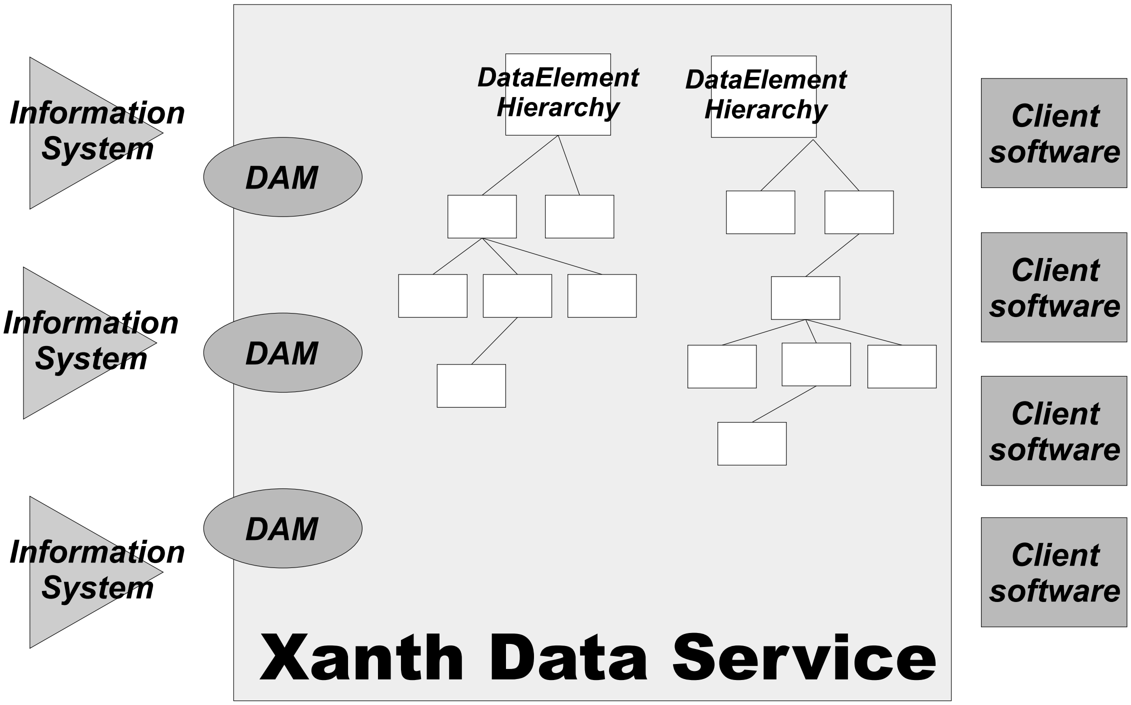
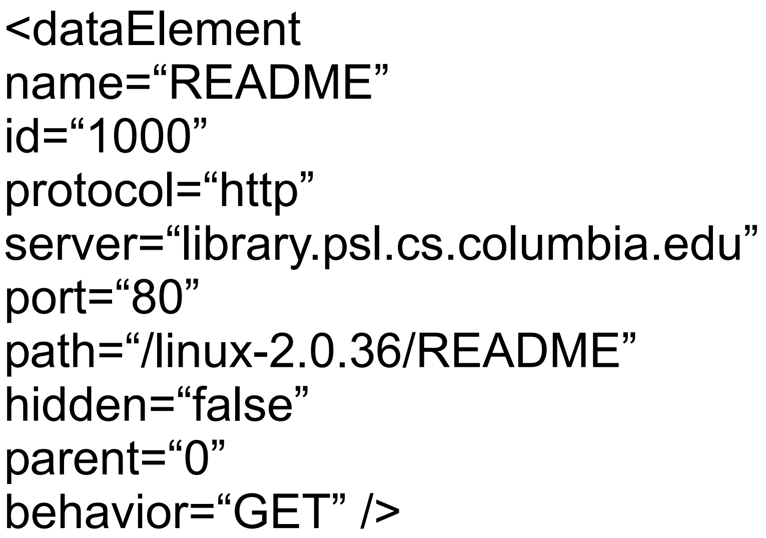


Figure 4-2: The Xanth Data Service acts as a middleware layer between client software and back-end information systems. Xanth's DataElement hierarchies impose an unique ordering ordering among project data.

Xanth acts as an organizer for project data. It maintains a multi-rooted tree hierarchy of stub references to the external data (these stubs are known as Xanth DataElements). Each DataElement contains a number of different fields which Xanth uses to retrieve the referenced data from the origin server. These include the protocol used to communicate with the server, any connection information necessary (server hostname and port, as well as any request name or path for the information in question, etc.), as well as a name and unique id number for the data. A parent field specifies the id number of this DataElement's parent DataElement in the hierarchy. Figure 4- 3 contains the definition of an example DataElement. Xanth maintains the

DataElement hierarchy as a set of XML elements, and uses XML as its persistent storage format. XML was chosen to make potential integrations with other systems easier.



```
<dataElement
name="README"
id="1000"
protocol="http"
server="library.psl.cs.columbia.edu"
port="80"
path="/linux-2.0.36/README"
hidden="false"
parent="0"
behavior="GET" />
```

Figure 4-3: XML representation of a Xanth DataElement.

Through its hierarchy of DataElements, Xanth maintains relationships among project artifacts which could not normally be determined from the data alone. For instance, in the data hierarchy, a DataElement describing test plans for a particular component may be located as a child element below the DataElement for the source code they refer to. Despite the fact that they are likely stored in separate information repositories (the source code may be located in a configuration management system while the test plans are stored in a document management system), inside Xanth there is a parent-child relationship among the DataElements which represent the artifacts.

Xanth was developed in reaction to the traditional Software Development Environment model of full control over project artifacts. Most SDEs (in fact, most project environments used in a number of disciplines) assume that all data which is relevant to a particular project is under the exclusive control of the development environment. While it is often easy to invoke external tools as part of the development process, the files generated or modified by these tools must be controlled by the environment. Unfortunately, a number of useful tools involve client-server access to an information repository; users run client software which communicates, via a network, with server software which "owns" the data. The World Wide Web provides a good example of this problem. It is not uncommon for development groups to publish useful information regarding their products or projects on a WWW server. Users who need this data access it via Web Browser software. There is no way for the web-based project data to be owned by the development environment. If a team utilizes any of the available WebDAV [Goland, et al., 1999] clients and servers to publish and maintain project documentation, the development environment cannot maintain any sort of control over the process.

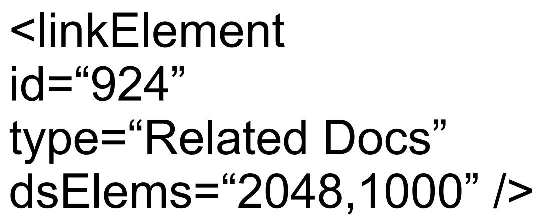
In order to provide useful SDE-type services to developers accessing and updating information living in external servers, a middleware system like Xanth is needed. Xanth sits between client software and data repositories, mediating access to the data. Xanth emits Groupspace Controller events for each data access, allowing other Groupspace Services to not only maintain knowledge of the actions being taken by users of the system, but also to allow services to veto access or otherwise modify the request. Thus Groupspace Services can be interposed above access and usage of project data.

How does Xanth accomplish this interposition? As you can see from Figure 4- 2, Xanth includes protocol components which allow it to mediate access to information systems. Protocol plugins known as PAMs (Protocol Access Modules) are responsible for communications with the back-end information server that contains the needed data. PAMs are lightweight code modules which can be loaded and unloaded as needed at runtime and can build on each other’s services.

Xanth uses its PAMs to implement the retrieval protocols specified in the DataElements. For example, an HTTP plugin might be configured into an instance of Xanth, allowing it to communicate with WWW servers. A basic protocol plugin is quite simple; all it can do is verify that the server, port, and path fields of a given DataElement are of the proper format for this protocol. To become more useful, each PAM may provide a set of “behaviors” for its DataElements. Without these behaviors, Xanth cannot perform any actions on the data. An HTTP plugin, for example, might provide behaviors for the basic HTTP methods, namely GET, POST, PUT, etc. If the protocol plugin provides behaviors, it is expected to add a “behavior” field to each of its DataElements' XML. This field contains a list of behaviors supported for this DataElement, and may be used by other components of the system to determine the actions the user can take with a given DataElement. In Figure 4- 3, we can see that an HTTP plugin has added the GET behavior to this DataElement, indicating it is able to fetch the document on behalf of the user if needed.

In the course of the research described in this dissertation, we have developed a number of PAMs for a variety of protocols. These have included CVS, the open source configuration management system, HTTP, the protocol used by WWW servers and browsers, NNTP, the protocol used by Usenet newsgroups, Chimera, a protocol used by the Chimera Open Hypermedia System [Anderson, et al., 2000], FTP, the file transfer protocol, and others. The variety of actions which can be performed by each protocol is varied, and the extensibility of the XML format used for DataElements comes in handy here. By listing the set of operations possible in the “behavior” field of each DataElement, other Groupspace Services configured into a Groupspace Controller along with a Xanth instance can perform detailed operations with a particular DataElement. Of course, Xanth handles the default cases of read access to DataElement information by default, so other services can be built with as little knowledge of the DataElement protocols as possible.

To address the hypertext features of the Groupspace model, Xanth includes a Link Service which provides typed, n-ary, bidirectional hypertext links among elements. The Xanth Link Service maintains its own XML document made up of



```
<linkElement
id=“924”
type=“Related Docs”
dsElems=“2048,1000” />
```

Figure 4-4: XML definition of a Xanth LinkElement. This link connects DataElements 2048 and 1000.

LinkElements (see Figure 4- 4). Each LinkElement has 3 fields: a unique id number, a descriptive type field, and a list of DataElement id's which are part of this link. The hypertext model followed by the Xanth Link Service is different from the hypertext model underlying the WWW -- in Xanth, hyperlinks are stored separately from the data they reference, while WWW pages embed link references inside their content. Xanth's model is richer, supporting more sophisticated hypertext among artifacts. It is conceptually similar to the hypertext provided by the various Open Hypermedia Systems [Halasz and Schwartz, 1994] which have been created by the hypertext research community.

4.2.1 An Example Xanth Setup

To better explain the use of Xanth, we will describe an describe a possible usage of its features in a fictitious setup.

A software development team is working on a project, called the X1 Satellite project. The project is broken up into two parts, the satellite ground system (the part which stays on earth) and the actual satellite itself. There are a number of reusable components which they will be using to develop the code for the projects, some of which have been licensed from third-party vendors and others of which are standard components built by their organization for past projects. As you might imagine, they are using a number of standard development tools to work on their code, including testing software, profiling tools, and bug tracking databases, in addition to web-based documentation repositories and configuration management systems.

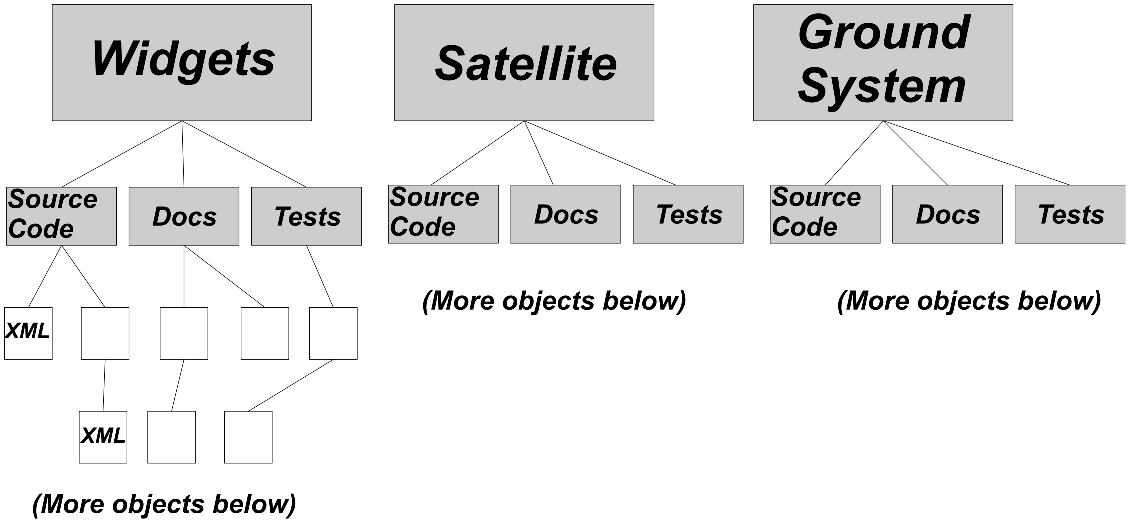
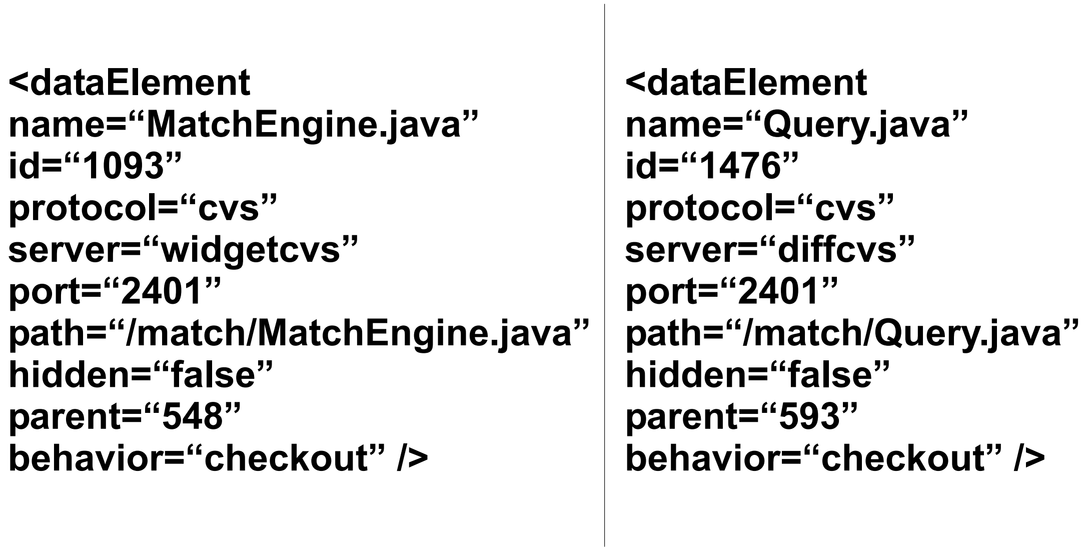


Figure 4-5: DataElement hierarchy for the X1 Satellite project.

Figure 4- 5 shows a snapshot of the DataElement hierarchy from their Xanth installation. For space purposes, the figure includes only selected portions of the actual hierarchy the project might include. As you can see from the diagram, DataElements in their Xanth installation have been organized into three main sub-categories: reusable components (labeled ‘Widgets’), deliverables for the satellite ground system, and deliverables for the satellite itself.

Figure 4- 6 contains the XML representation of the highlighted DataElements (marked ‘XML’) from Figure 4- 5. As you can see, despite the fact that different project artifacts reside in different repositories, they are organized into a single hierarchy of elements inside Xanth. This allows the developers to see a complete picture of all relevant artifacts, making it easier for them to find information they need.



```
<dataElement
name=“MatchEngine.java”
id=“1093”
protocol=“cvs”
server=“widgetcvs”
port=“2401”
path=“/match/MatchEngine.java”
hidden=“false”
parent=“548”
behavior=“checkout” />
```

```
<dataElement
name=“Query.java”
id=“1476”
protocol=“cvs”
server=“diffcvs”
port=“2401”
path=“/match/Query.java”
hidden=“false”
parent=“593”
behavior=“checkout” />
```

Figure 4-6: XML representation of selected DataElements.

In this Xanth installation, three primary PAMs are in use to fetch data represented by DataElements. These include CVS for source code configuration management, HTTP for fetching data from Web-based documentation repositories, and GNATS for fetching information from a GNATS defect tracking database.

Xanth was designed to work hand in hand with other CHIME framework components. The main feature it lacks is any real user interface components; in the

CHIME framework it is the responsibility of the ThemeManager and Theme components to provide a user interface to the DataElements and services provided by Xanth. However, Xanth could be plausibly used outside of CHIME. It includes a simple text based command line user interface which can be used to add and remove data, inspect DataElements, etc. Realistically, another Groupspace Service would need to be constructed to handle any user interface requirements, and client software would also likely need to be built to allow users access to Xanth. During the course of this research, we built such a user interface, known as “XanthWorks” to demonstrate the Xanth component during a funding meeting (it is notoriously hard to demonstrate server-side components without some sort of user interface). XanthWorks provided an extremely thin veneer over the Xanth Groupspace Service and showed that the component could be used outside of CHIME.

4.3 The Virtual Environment Modeler (VEM)

The Virtual Environment Modeler (VEM) is another CHIME framework component which is implemented as a Groupspace Service. In this section, we will describe the VEM, the relationships it has to other CHIME framework components, and the role it plays in the creation of CHIME worlds.

Conceptually, the VEM is the middle component in a three-step process of getting information and project artifacts into a CHIME world. Figure 4- 7 shows the three components, Xanth (described earlier), the VEM, and the CHIME ThemeManager component (which is described in the following chapter). After the project artifacts have been identified and the PAMs needed to access them have been built and configured into a Xanth instance, the next step is to begin the process of creating the Virtual Environment from the data.

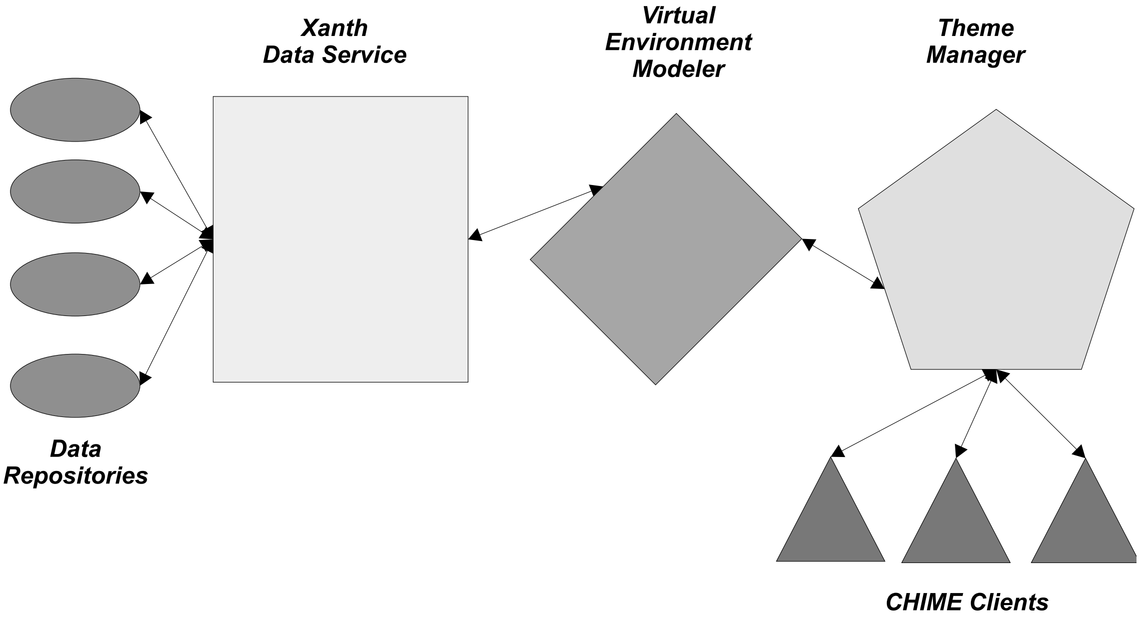


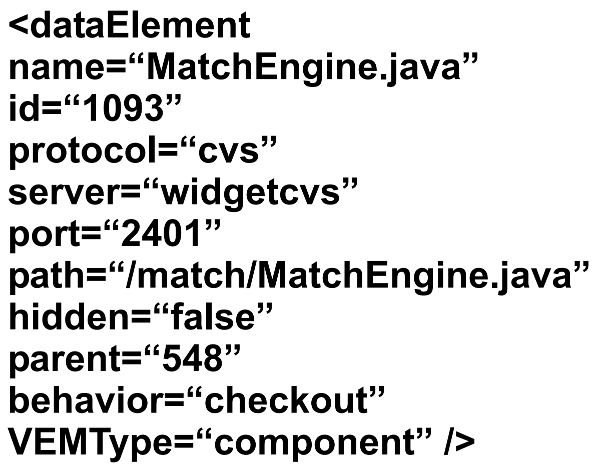
Figure 4-7: CHIME architecture.

VEM begins this process through “Virtual Environment Hinting.” Essentially, the person(s) setting up the CHIME world tag each DataElement in the Xanth hierarchy with VEM information identifying the role of the DataElement in the eventual virtual world. Is a particular DataElement meant to contain other elements (i.e. will it end up as a room of some sort in the virtual world)? Perhaps a particular Xanth link should be specially treated in the virtual world?

The VEM supports an extensible hierarchy of Virtual Environment types (VEM types) which are used to tag DataElements. The base set includes Components, Containers, and Connectors. Components are the “base class” from which all other types derive. A Component is just that, a part of the virtual world. It will sit inside a room in the virtual world, and can then be manipulated by users in the room. Containers are components which will contain other DataElements in the eventual virtual world. They will likely end up as rooms or subrooms, depending on the particulars defined by the ThemeManager component (see chapter x). Connectors are essentially links, and are treated as first class objects in the VEM system. A Connector communicates a relationship among two DataElements which is not obvious from their locations in the Xanth hierarchy.

VEM hints are written into the XML representation of the DataElement hierarchy in Xanth. This takes advantage of the extensible nature of XML: components which parse Xanth’s XML output which do not understand (or care about) VEM hints will silently ignore them. Figure 4- 8 contains a Xanth DataElement which has been tagged with a VEM hint.

VEM is implemented as a Groupspace Service which layers on top of the Xanth Groupspace Service. The VEM can be configured at runtime with a series of defaults regarding the hints to be applied to a variety of object types; for instance, in a given CHIME world, all source code might be tagged as a Component while third



```
<dataElement
name="MatchEngine.java"
id="1093"
protocol="cvs"
server="widgetcvs"
port="2401"
path="/match/MatchEngine.java"
hidden="false"
parent="548"
behavior="checkout"
VEMType="component" />
```

Figure 4-8: Example DataElement XML with VEM hint added.

party libraries become Containers. When Xanth emits an event onto the Groupspace Controller indicating a new DataElement has been added, VEM can apply these defaults and attempt an automatic categorization and hinting. Similarly, when the VEM is notified that a DataElement has been moved in the Xanth hierarchy, VEM attempts a recategorization (the relative location in the hierarchy may be a criteria used to determine the VEM hint applied to a DataElement).

An important aspect of the VEM is that it does not make any decisions or dictate any choices regarding the eventual display or layout of the CHIME world being created. The VEM hints are just that, hints to the ThemeManager component which is responsible for utilizing Xanth DataElement hierarchies and VEM hints to create the actual virtual world. The role of the VEM is simply to begin the process of converting the project’s artifacts into a CHIME world.

The roles of the three components of the CHIME framework (Xanth, VEM, and ThemeManager) will be more fully discussed in the next chapter. You might have noticed, however, a vague similarity between the Smalltalk-80 programming language's concept of Model-View-Controller (MVC) and the three-level separation among Xanth, VEM, and ThemeManager. In the MVC paradigm user input, the modeling of the external world, and the visual feedback to the user are explicitly separated and handled by three types of object, each specialized for its task. In the CHIME framework, Xanth acts as the Model, managing internal data for the application. VEM acts as the View component, utilizing its hinting mechanism to maintain a blueprint of the eventual virtual world. The CHIME ThemeManager is responsible for all things visual in the virtual environment being created.

4.4 Related Work

Research into Middleware systems is most closely related to the Xanth and Groupspace Controller components of the CHIME framework. Much of this work has come out of the Database community, which (quite reasonably) focuses on the ability to perform logical SQL-style queries against the back-end data. The VEM component is similar in some respects to aspects of a number of Virtual Environment Frameworks. We will discuss each in turn.

4.4.1 Middleware

Middleware systems typically synthesize access to a variety of back-end information systems into a single coherent whole presented to client systems. Clients may access data inside the middleware without knowledge of the mechanisms used to access the actual data; data continues to reside in its original information system.

The MIX project [Baru et al., 1999] utilizes XML metadata to create a digital library from a set of constituent document repositories. Documents and access rights are described in a proprietary dialect of XML which supports queries and view materializations of the kind normally only found in traditional database systems. MIX also includes a Query-by-example (QBE) system, allowing novice users the ability to query for documents without knowledge of MIXes XML query mechanism.

MIX is conceptually similar to Xanth in that both attempt to provide uniform access to data stored in remote repositories. To date, however, the MIX project has attempted only to mediate access to a variety of back-end database systems. While Xanth lacks the query mechanisms found in MIX, it has proven flexible enough to handle access to a wide variety of information repositories.

ARIADNE [Ambite et al., 1998] and TSIMMIS [Garcia-Molina et al., 1995] are research systems which utilize mediators (small plugin components) to provide extensible, customizable access to back-end information repositories. In these systems, the mediator is awre of the information present in different sources and retrieves information from them through a “wrapper” around each source. TSIMMIS has been used to integrate information from a variety of heterogeneous database systems. Ariadne provides similar kinds of access to a variety of Web-based sources.

CHAPTER 5 The ThemeManager And Themes

If you can dream it, you can do it.
— Walt Disney

In the previous chapter, we described the CHIME framework's Xanth and Virtual Environment Manager (VEM) components, and how they combine to organize project artifacts and begin the process of creating a virtual environment around them. In this chapter, we will describe CHIME's ThemeManager component, which is responsible for all facets of the user's experience in the CHIME world. First, we will describe in detail how the ThemeManager and its Themes decide the display aspects of the CHIME world. Next, we will describe how the ThemeManager interacts with the CHIME system client to allow users access to the virtual world, and to allow them to communicate with each other, including a detailed description of the protocol in use between clients and the ThemeManager and a description of ancillary Groupspace Services included as part of the CHIME framework. Following this, we will outline the process of building a CHIME Theme by defining some requirements for Themes and describing steps to follow in order to meet those requirements. Finally, we will discuss related work in the areas of Virtual Environments, Multi-User Domains (MUDs), use of VEs for real work, and Software Visualization.

5.1 ThemeManager

As you can see from Figure 5- 1, the CHIME ThemeManager is the third component in the process of taking project artifacts and creating a Virtual Environment from them. The ThemeManager is responsible for maintaining all aspects of the virtual world, including communications between users, acting as a conduit for user requests into Xanth and other Groupspace Services, and, in combination with a Theme plugin, making all final decisions regarding layout, look, and feel of the virtual world.

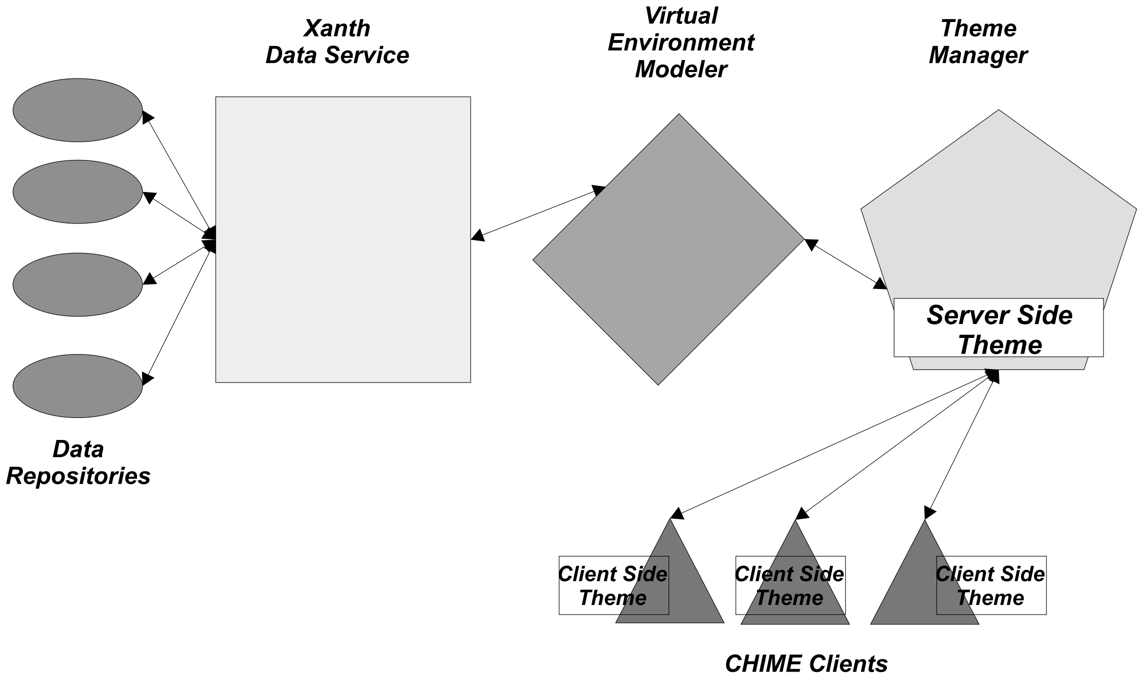


Figure 5-1: The process of creating a Virtual World with CHIME. Data flows through the Xanth Data Service, into the VEM, and into the Theme Manager for delivery to Virtual Environment Inhabitants.

The ThemeManager is a generic component which, like the Xanth Groupspace Service described in an earlier chapter, is an integral part of the CHIME framework. A ThemeManager is present in every CHIME world created. In order to handle customization of the CHIME world to the requirements of the organization implementing it, the ThemeManager works with a Theme plugin. The Theme plugin code, which is loaded at runtime by the ThemeManager, decides look and feel and other layout issues. The ThemeManager, which is responsible for interfacing with the rest of the CHIME framework, delegates all aspects of what the user sees and can do to the Theme Plugin code. This is the primary mechanism through which CHIME allows the creation of wildly different virtual worlds using the same basic framework.

Figure 5- 1 illustrates the ThemeManager's place in the CHIME framework's architecture. As you can see, the Theme plugin is broken up into two halves, a ServerSideTheme and a ClientTheme. The two work hand in hand to create the virtual world experience. As you might have guessed from the names, the ServerSideTheme runs on the CHIME server side of things, and the ClientTheme runs in the user's client software (in fact the CHIME client software, at startup, downloads the current ClientTheme from the server if it does not have a locally cached copy).

As we mentioned earlier, the ThemeManager is responsible for tracking users through the virtual world and providing a variety of ancillary information about each user. The ThemeManager is a generic component of the CHIME framework, and is used unmodified in every CHIME world. As a result, the tracking of users is done in an opaque fashion. The ClientTheme running on each users' client software periodically notifies the ThemeManager of its users' location in the virtual world it has created; the ThemeManager has no way of interpreting or understanding this data which is sent to it. It simply passes the information along to other ClientTheme instances (running in other users' software) as necessary.

In addition to tracking users detailed positions through the virtual world, the ThemeManager also keeps track of other information on each user. Information such as which avatar model a particular user is using as his or her representation in a 3D world is important for all ClientThemes to know so that they can accurately draw the shared virtual world. Related to the detailed positioning information, other ClientThemes might need to ask which room of the virtual world a particular user is located in. (To conserve network bandwidth, the ThemeManager only sends to each user detailed positioning information for other users inhabiting the same room. We will more fully discuss the network aspects of the ThemeManager/Theme plugin architecture later in this chapter).

An important role played by the ThemeManager and the Theme Plugin is to funnel user requests into Xanth and other Groupspace Services, and thus to make available the resources and capabilities of the local Groupspace Services to the CHIME world's users. For instance, one of the major capabilities of Xanth is to fetch data from a variety of information services; it is the ThemeManager and Theme Plugin's responsibility to make this capability available to CHIME world users. Xanth's hypermedia capabilities are another example of a service which the Theme Plugin code needs to make available to the end users. Thus the Theme Plugin code (both the ServerSideTheme and the ClientTheme code) can be viewed as the “glue” which connects the services of the CHIME framework to the specifics of the Theme. The Theme code is responsible for using these services in the construction of the virtual world.

The Theme Plugin code’s most important task is to create the virtual world users will inhabit. This includes processing all the DataElements from the Xanth hierarchy, utilizing the VEM hints which have been placed on each DataElement, and determining a layout and scheme for the virtual world. We use the term “Theme” to describe this component because it is responsible both for the layout of the virtual world as well as its entire look and feel. In the case of a 3D virtual world, this will include the choice of 3D models, colors, and other aesthetic choices which make the world immersive. For example, the Theme Plugin might be designed to use an airplane hangar metaphor (as mentioned in an earlier chapter). This would include models and color schemes both for the hangar and all the components within (reference manuals and parts along the side walls, airplanes inside the hangar, etc.) Thus the real job of the Theme Plugin is to pull together all of the component features and services found within the CHIME framework and lay out a complete, immersive virtual environment which users will inhabit and work within.

In addition to an initial layout of Xanth DataElements into the virtual world, a major task of the Theme Plugin is to determine how to evolve the world in response to changes in the organization of the Xanth DataElement hierarchy. Deciding how the display of the world should change based on an element being added or a whole branch of the hierarchy moving is a non-trivial task. The Theme Plugin is responsible for making these kinds of final decisions about the layout, look, and feel of the virtual world.

It should be noted that nothing in the design or architecture of the ThemeManager or the specifications for Theme Plugin code requires that the virtual environment being create use 3D models or even any sort of graphical user interface for the VE. While the CHIME framework was built specifically to support Software Immersions done with the help of 3D techniques, it does not preclude the creation of textual virtual environments. Indeed, research studies [Jerding and Stasko, 1995] have shown that textual virtual environments, while not as flashy or colorful as 3D environments, are quite able to capture the users' attention and become quite immersive. It is up to the Theme Plugin to decide whether or not to make use of 3D as an aid to creating an immersive experience. There are a number of situations, in fact, in which 3D could not be used, including those in which the users of the virtual environment do not have access to powerful enough hardware to support 3D graphics well. Low-bandwidth environments are another example of an arena in which an immersive virtual environment with access to project data might be useful, but the large graphics files needed for a 3D environment could not be handled.

5.1.1 Model-View-Controller

From Figure 5- 1, you can see that in the CHIME architecture, the process of constructing a virtual environment from project artifacts is broken up into three steps. Each major step is represented by a separate component. Xanth handles the organization of the project artifacts into a hierarchy of elements to include in the virtual environment. The Virtual Environment Modeler (VEM) component begins the process of categorizing artifacts for inclusion in the virtual environment. The ThemeManager (in conjunction with Theme Plugins), as we have discussed in this chapter, is responsible for the final layout and operation of the virtual environment.This is conceptually similar to the Smalltalk-80 programming language's implementation of all graphical and user interface components using a paradigm they termed "Model-View-Controller" (MVC).

In the MVC paradigm, user input, the modeling of the external world, and the visual feedback to the user are explicitly separated and handled by three types of object, each specialized for its task. Each architectural component is associated with three small objects, one each for the model, view, and controller. The model object represents the state of the component, while the view and controller handle display and working with the component respectively. This separation of responsibility makes for an extremely flexible architecture. The back-end information about a particular component (the model) need not know anything about how its information ends up being displayed or manipulated. Similarly, the display (view) object does not need to know any internal details about the data, or about how the manipulators work. Finally, the manipulator really doesn't need to know much about how the model or view work internally.

In the CHIME framework, Xanth acts as the Model, managing internal data for the application. VEM acts as the View component, utilizing its hinting mechanism to maintain a blueprint of role of various Xanth DataElements in the eventual virtual world. The VEM really does not need to understand or care about how Xanth might go about fetching the data behind the various DataElements, or about how those DataElements got placed in Xanth. The CHIME ThemeManager (and the Theme Plugin code) is responsible for allowing the user to interact with the data from the model (Xanth) and view (VEM), and for creating a usable whole from the components. By following the Model-View-Controller paradigm, we have attempted to make the CHIME framework's components extremely flexible and easy to extend for future needs.

5.2 Pulling it all together

The CHIME system client is the users' main entrypoint into a CHIME world. It provides the user interface through which the user will interact with all the project artifacts, see other users' avatars, and navigate around the virtual world to accomplish their work. The client is architected in a modular way, with much of its functionality provided by the client-side Theme Plugin code, which is downloaded from the ThemeManager at startup time. In this section, we will describe the architecture of the

CHIME client, as well as discussing the protocol it uses to communicate with the CHIME server. In addition, we will talk a bit about some of the other Groupspace Services (besides Xanth, the VEM, and the ThemeManager) which the client relies upon to perform its role.

As you can see from Figure 5- 2, the CHIME system client is divided internally into three major layers. The Communications Layer is responsible (as you might imagine) for all communications with the CHIME server. The Theme Services Layer provides a foundation for the downloaded Theme plugin code to build upon, offering access to functions inside the client. Finally, the User Interface Layer is responsible for display of all of the built in user interface components (a Room Browser, a User List, a Chat Window, etc. Figure 5- 3 shows a screenshot of the CHIME client running a Theme, describing each of the static components) as well as the 3d models specified by the Theme plugin code.

One of the major design goals for the CHIME client was to support its use in situations where there is only a low or medium bandwidth communications channel between the client and the CHIME server. The major way in which this is accomplished is through the use of an extremely lightweight communications layer, which handles all virtual world-related communications between the client and server.

In order to create a realistic, immersive virtual environment, it is necessary for the CHIME clients to be in constant communications with the server regarding which virtual world room they are located in, their positioning inside the room, the avatars

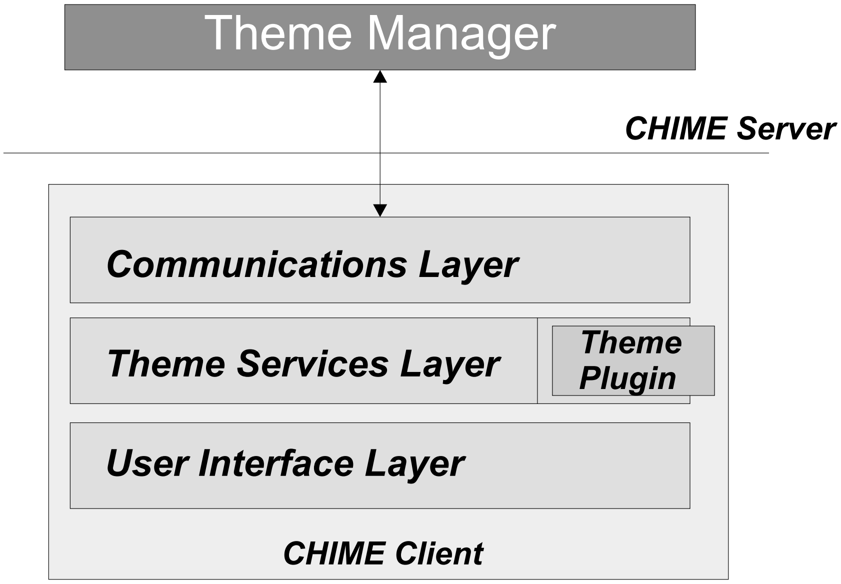


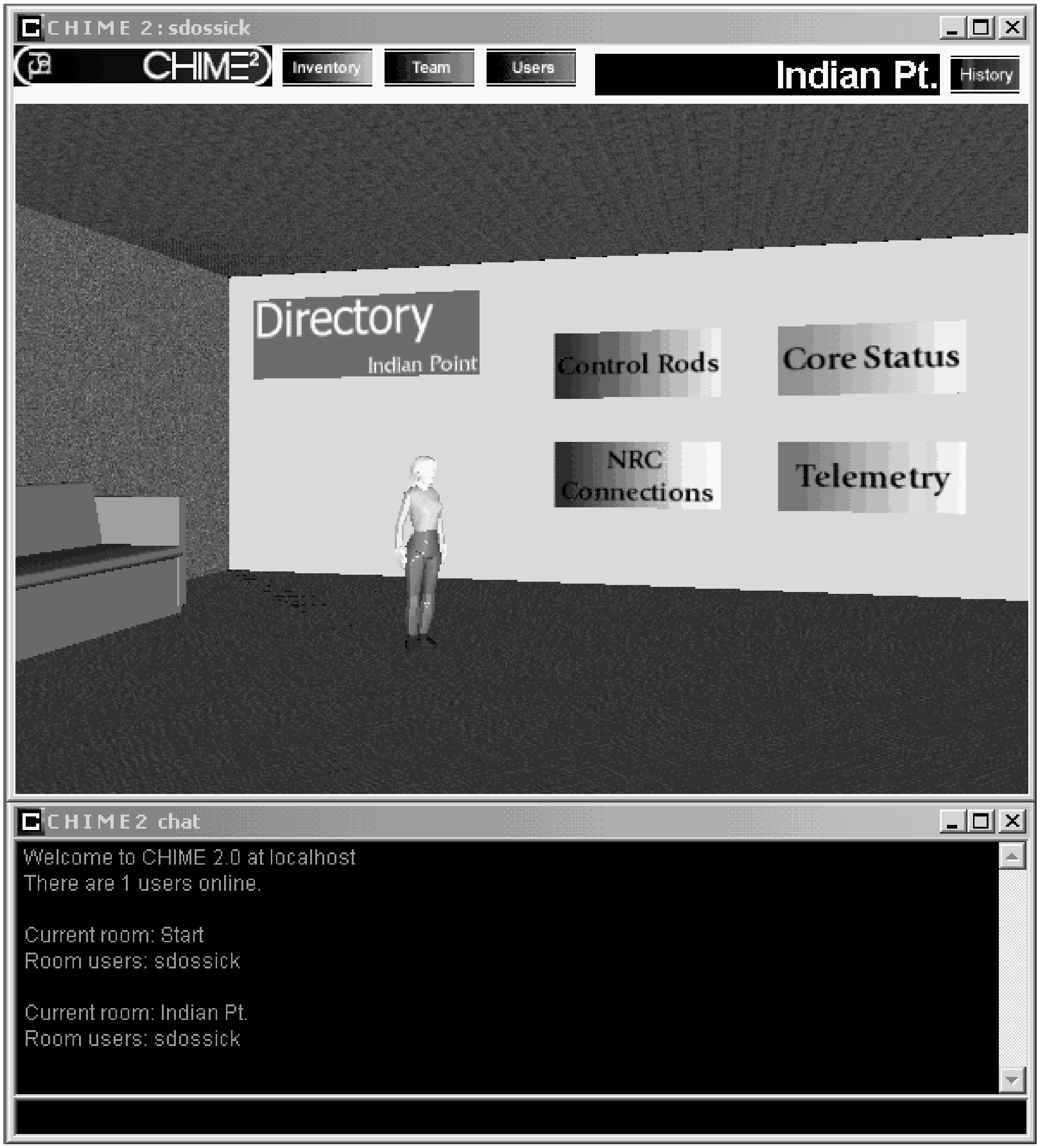
Figure 5-2: CHIME Client Architecture.

they are using to represent themselves in the virtual world, and the objects they are currently working with. Since we envisioned CHIME worlds being used to overcome geographical distribution hurdles (where users are located in separate physical locations, possibly far removed physically from one another), and since CHIME worlds may involve hundreds or even thousands of users, we needed a lightweight communications architecture for the data to be sent back and forth.

In the architecture we settled on, the CHIME server sits conceptually in the center, mediating communications between clients and sending down information packets tailored to the needs of each client. All communications are done via lightweight "ThemeManagerDatagram" packets, which are sent back and forth from the clients and server. Clients periodically encode their position and orientation, along with the id of the room they are presently in into a ThemeManagerDatagram and send it up to the ThemeManager. The ThemeManager then resends this information out to all other “interested” clients (we define “interested” clients as those who are in the same room as the client reporting its position information. Clearly there is no need for those outside of his or her room to receive position information, so they don’t receive it as a bandwidth limitation mechanism). ThemeManagerDatagrams are also used to support the chat capabilities of the CHIME client; a user can send a chat message to the entire room or send a private message to another user. In both cases, the ThemeManager receives the ThemeManagerDatagram packet and sends it out to other interested clients.

While we have not performed an exhaustive investigation into the performance and bandwidth requirements of the communications protocol in use by CHIME (as other Virtual Environment researchers have performed for their systems -- see, for instance, [Macedonia, et al., 1994]), qualitative results show us that its lightweight design is successful in supporting low- and medium- bandwidth connections between the clients and server. We have tested CHIME both across the Internet via high speed lines as well as over ISDN (128kbps) and standard analog modems (33.6kbps). Under all conditions, the client remains quite usable.

The Theme Services Layer provides an API for Theme plugin code (downloaded from the CHIME server at startup) to interface with the rest of the CHIME client code. This API allows the Theme to request information from the server (e.g. to determine what DataElements are to be found in a particular room, what their VEM



```
Welcome to CHIME 2.0 at localhost
There are 1 users online.

Current room: Start
Room users: sdossick

Current room: Indian Pt.
Room users: sdossick
```

Figure 5-3: CHIME client running a Theme.

hints are, what links have been set up among them, etc.) in a modular way, allowing Theme writers to concentrate on the specifics of their themes rather than on the particular details of the CHIME framework internals. The Theme Services Layer also gives themes access to any other Groupspace Services which are available on the CHIME server, so that they can make use of them or make their services available to the user.

One of the most important functions of the Theme Services Layer is to allow Theme plugins to specify the layout and arrangement of the rooms which will make up the CHIME virtual world. The job of the Theme plugin is to parse through the Xanth DataElement hierarchy and associated VEM hints to create the CHIME world. In order to do this, the Theme must decide which rooms to create and how they will interconnect. Once it has done this, it needs to communicate this information to the local CHIME client, which it does through the Theme Services Layer. In addition, the Theme specifies which 3d models are to be used for the rooms, including positioning and coloring information for each object. The Theme must do an initial layout of each room at startup time as well as respond to all DataElement additions, deletions, and moves within the hierarchy, and update the room layout accordingly.

The User Interface Layer in the CHIME client is responsible for drawing all the various components of the client user interface, including both static user interface elements (which are handled completely by the client) as well as user interface elements specified by the Theme Plugin downloaded at runtime. Referring back to Figure 5- 3, we see a screenshot of a running CHIME client. As you can see, a number of static, baseline windows are included with the CHIME client. These include a chat window, through which users can chat with one another in a given room as well as send private messages to other users, a list of all the users in the system, a list of all the rooms created by the current Theme, as well as the main viewport window which displays the current room’s 3d models.

The User Interface Layer is responsible for taking the 3d models specified by the Theme Plugin to the Theme Services Layer and actually displaying them. It is also responsible for allowing the user to navigate among the artifacts in a given room, by walking around them. When a user changes position, the User Interface Layer needs to redraw the current display based on the user's new viewpoint. In addition, when a user changes rooms, it is up to the User Interface Layer to redraw the screen with the new room's models.

The CHIME client is implemented entirely in Java, and utilizes the SGI Open-Inventor libraries [Wernecke, 1994] to handle all 3d graphics requirements. The Open-Inventor libraries are a well known 3d graphics paradigm which, while not the most powerful mechanism for building 3d software, are very straightforward for non-3d experts (like the author of this dissertation) to use. The provided an ideal platform for experimentation, and their use of the industry standard Scene Graph construct makes it easy for anyone who has ever written 2d windowing GUI code (with any toolkit ranging from Java's AWT to Microsoft's Win32 API to X Windows) to learn. All 3d models provided by a Theme Plugin for use in a CHIME world must be stored in either the native Inventor 2.1 format or in VRML 1.0 format, both of which are widely supported by a variety of 3d tools.

5.2.1 Auxiliary components

The CHIME framework includes a number of Groupspace Services which are used in the execution of CHIME worlds but which have not yet been discussed.

The most straightforward Groupspace Service included with the CHIME framework, and one which is potentially extremely useful in any kind of collaboration environment (including the Software Immersions we intend the CHIME framework to be used for) is the CHIME Annotation Service. This service allows CHIME users to attach arbitrary annotations to any DataElement in the Xanth hierarchy. The Annotation Service does not store the DataElements referenced in the annotations, rather it stores the unique DataElement id along with the annotation on that DataElement. This allows CHIME clients to quickly look up any annotations which have been made about the project artifacts inside a given room in the virtual world. Annotations can take any form; the Annotation Service simply stores the binary representation of each annotation, and passes it back to CHIME clients opaquely when asked. This allows the client to be responsible for all display and manipulation of these annotations; the Annotation Service really does not care about the formats of the data it stores.

Arguably the most important auxiliary Groupspace Service used within the CHIME framework is the UserManager service. This service stores usernames and passwords, as well as certain kinds of personal information (real name, email address, mailing address and phone number, etc.) for each user. Because it stores usernames and passwords, the UserManager plays a critical role in authenticating users as they attempt to login to a CHIME server. In addition, the UserManager provides access to the personal information it stores for each of the users in the system, which allows other users' CHIME clients to easily request this information.

The CHIME TeamManager is a Groupspace Service which allows virtual world users to be grouped into project teams. A single user may be found in multiple teams (as is the case in many companies and development organizations where a given employee may be involved in a number of projects at any given time). CHIME Theme Plugins may choose to make use of team information from the TeamManager if it makes sense to include such information in the particular virtual world being created. Team information can be used in combination with simple visual techniques (specific color shadings for different teams, etc.) to easily convey project information to users of the virtual world. CHIME Theme Plugins are free to make use of TeamManager information as they see fit.

The CHIME ConsoleService is a simple administrative component which allows a CHIME virtual world administrator to change server settings, see a list of which users are logged into the server at any given time, change user passwords, create new user accounts, change team membership information, as well as quickly add and delete elements from the Xanth DataElement hierarchy. The ConsoleService is implemented as a command-line user interface; this allows administrators to access it from virtually any networked computer, without needing so much as a web browser to access its services.

5.3 Building a CHIME Theme

To further illustrate the role that CHIME Theme Plugins play in the framework, in this section we will describe them in greater depth. First we will discuss some requirements of each CHIME Theme, paying attention to the particular aspects of virtual world creation which the Theme is responsible for. Next we will describe the steps involved in building a new theme, and show how these steps can be used to fulfill all the requirements set forth for the creation of a Theme Plugin.

5.3.1 Theme Requirements

Theme Metaphor: The Theme must be built around a metaphor which governs the mapping of project information and artifacts into the creation of the virtual world. For example, in a previous chapter we described a thought experiment in which a Software Immersion was created for a software development project in the aerospace industry. The metaphor of an airplane hangar was chosen there. The airplane hangar included airplanes which were the actual software projects being worked on; planes were co-located in the same hangar based on a relationship among the projects or sub-projects they represented. The metaphor chosen should be rich enough such that it can be extended for the particular project it is applied to, but simple enough that users inside the virtual world can intuitively guess what various aspects of the metaphor refer to in the underlying project. Going back to the thought experiment described previously, you may recall that transparency was used on each airplane in the hangar to represent the degree of completion of the project; as projects progressed they would become progressively less transparent until they were completely opaque.

Layout of DataElements at startup: A more concrete requirement is that the Theme must utilize the Xanth DataElement hierarchy at startup time, as well as the Virtual Environment hints found in the VEM service to define a layout for the virtual world. This layout must be deterministic; since each user of the virtual world will be using the same Theme code in their client, the layout produced by each one must be identical. By deterministic, we mean that the Theme cannot randomly scatter DataElements among rooms, but rather must produce the same virtual world each time it is run on the identical DataElement hierarchy. This is a key requirement if we are to create virtual worlds which can be inhabited by multiple people, as they each must be sharing the same virtual space in order to create the feelings of immersion we are looking for.

Continual update of virtual world based on DataElement changes: A related requirement is that the virtual world created by a Theme Plugin must constantly evolve as changes are made to DataElements in the Xanth hierarchy. As elements are added, deleted, or are moved within the hierarchy by users through the course of their work inside the virtual world, the Theme must accommodate these changes my modifying the virtual world accordingly. This brings up interesting questions with regards to how users are to be made aware of the changes which are occurring around them. A significant portion of the future work identified by this research involves mechanisms for making users of the virtual world aware of the actions taken by others without inundating them with update information or disorienting them by having the world constantly changing around them. (A more thorough discussion of future research possibilities can be found in the final chapter of this dissertation).

User interface mechanisms for artifact and user interaction: A final requirement for Theme Plugins is that they must provide mechanisms through which the virtual world inhabitants can interact with the project artifacts the DataElements represent as well as with each other. As we have discussed earlier in this chapter, the CHIME system client provides a certain baseline of interaction possibilities through its “static” user interface elements (we call them static because they are provided by the client itself, not by Themes, and are thus always available to the user no matter the Theme in use). It is up to the Theme Plugin to go beyond the capabilities afforded by the static user interface elements of the CHIME client and give users the ability to manipulate DataElements (perhaps by displaying the project artifact they represent, moving the artifact to a new location in the virtual world, etc.). These capabilities are what make the virtual world immersive. To a lesser extent, it is the job of the Theme Plugin to provide even more opportunities for user interaction. The CHIME client provides as one of its static elements a Chat window through which users in the same room can chat with each other (or send private messages to one another). Richer interaction techniques are certainly possible and should be provided by the Theme Plugin where appropriate.

5.3.2 Building a Theme

The first step in building a Theme Plugin is to pick a metaphor to use for the Theme. Although we described the role of the Theme metaphor when describing the requirements for a Theme, it cannot be stressed enough how important the metaphor is to the eventual success of the virtual world. If the metaphor is carefully chosen, it becomes easy to map project artifacts and various project tools into the metaphor to create a very immersive virtual world experience for the user. Poorly chosen metaphors are difficult for the user to understand.

Once the metaphor for the Theme has been chosen, the next step is to find or build some 3d models which will be used in the metaphor. These include models which will be used to represent project artifacts from the Xanth DataElement hierarchy as well as models which will be used to represent rooms the user walks around in. Selection of a good metaphor has a beneficial impact here as well; if the metaphor is well chosen, it is easy for the Theme builder to imagine what models he or she would like to use in the virtual world. As we mentioned above when describing the implementation details of the CHIME client, 3d models used by Theme plugins need to be in particular file formats, either the SGI Inventor 2.1 or VRML 1.0 formats. Fortunately, there are libraries full of freely-available 3d models in these formats easily found on the Web. These libraries make it easy for those without much artistic talent (including the author of this dissertation) to construct reasonably nice looking virtual worlds in keeping with the metaphors we have chosen.

Related to the choice of 3d models to use in the virtual world, the next step to the creation of a Theme is to decide what role color or other display attributes will play in the metaphor. For example, transparency can be used to represent visually the state of a particular project with respect to completeness. Color shading can be used with models to visually tag them as being part of a particular project. These kinds of visual communication of information can be extremely powerful. Exploiting this kind of visual information architecture [Tufte, 1997] has been shown to radically increase people's understanding of data [Tufte, 1990].

It is worth noting in our discussion of metaphors and 3d models that there is no particular reason the metaphor chosen for a particular CHIME Theme has to be rooted in the "real world." Rooms in a CHIME virtual world do not need to look at all like a room one might find here on earth. In fact, it can sometimes be advantageous to create a virtual space which bears no resemblance to anything the user might be familiar with. By ignoring certain physical laws (such as with transparency or color shading) we can sometimes quickly communicate information and ideas.

Finally, perhaps the most important part of creating a Theme Plugin for CHIME is to decide on an algorithm for mapping the Xanth DataElement hierarchy into virtual world elements. VEM hints can help to categorize each DataElement into a particular role in the virtual world, and Themes can be written to work with the extensibility found in the VEM; if the base VEM types of Component, Container, and Connector are augmented with application-specific types, the Theme Plugin can be written to exploit these. As we mentioned in the requirements for Themes, the algorithm decided upon must be able to generate the virtual world from the Xanth hierarchy deterministically, so that each user is sharing the same virtual world simultaneously. In addition, the algorithm must be able to dynamically update the virtual world based on changes in the DataElement hierarchy. Some algorithms may involve preprocessing the DataElement hierarchy on the server side to minimize processing time on each client. These kinds of tasks are the reason for the existence of ServerSideTheme Plugins. These are portions of the Theme which reside on the CHIME server and work with the Client-side Theme code to create the virtual world.

5.4 Related Work

Research into frameworks for building virtual environments as well as into the particulars of the network layer used by distributed virtual environment systems is related to the work described in this chapter. We will describe each of these areas in turn.

5.4.1 Virtual Environment Frameworks

A variety of researchers have built frameworks for the creation of virtual environments. One of the earliest of these was the LambdaMOO project [Curtis, 1992]. LambdaMOO included an object-oriented scripting language integrated with an underlying object-oriented database system to allow users to customize the MUD experience. Users could add a variety of new types of data into the system (by using standard object-oriented programming techniques as well as utilizing the extensibility of the

OODB paradigm). This extensibility is similar the malleability of the CHIME frameworks' underlying data representations, embodied within the Xanth and VEM framework components.

DIVE [Fahlen et al., 1993] is perhaps the oldest of the graphical virtual environment frameworks. DIVE includes mechanisms allowing a system administrator to easily define the component rooms of the virtual environment as well as the set of avatars which can be chosen by the eventual users of the system. As the DIVE research was performed a number of years ago, it focused quite a bit on providing a good experience even to users without powerful graphics hardware. This is no longer a limitation of today's computer graphics; even the graphics performance included in typical PCs today is a vast improvement over specialized workstations of just a few years ago. Like the CHIME ThemeManager, DIVE takes care of the underlying virtual environment functionality while allowing the framework user the freedom to customize the virtual environment as he or she sees fit.

Alice [Conway et al., 1994] is a 3D virtual environment framework aimed specifically at the problem of prototyping such environments. Alice is based on an object-oriented high-level language (Python) which makes it easy to customize in a variety of directions. In many ways, Alice represents a cross between LambdaMOO style customizablitiy and the 3D interaction techniques of systems like DIVE. Alice was developed by researchers from the CSCW community and as such is focused on exploration of effective interaction techniques among participants in a virtual environment. While

CHIME's Themes are not quite as flexible as the high-level customizability of the Alice environment, we feel they strike a good balance between scalability and flexibility. One possibility for future work with CHIME is to add in a standardized scripting language to be used within CHIME Themes. This would allow easy runtime customizations and exploratory programming from within the Theme environment, and could make the creation and customization of CHIME themes easier.

5.4.2 Virtual Environment Network Layers

A small group of researchers has focused on the problems inherent in scaling virtual environment systems (in particular, graphical or 3D virtual environment systems) to large numbers of potentially geographically distributed users. In particular, the NPSNET system [Macedonia et al., 1994] developed at the Naval Postgraduate School has focused specifically on these issues. NPSNET has designed an architecture which makes use of IP Multicast and other infrequently-used technologies to scale up to thousands of users sharing the same virtual space.

One problem with the NPSNET architecture from the standpoint of CHIME is that it requires the use of, in particular, IP Multicast. In order to truly support geographical distribution of participants, users most likely will connect with the virtual environment server over the public Internet. Unfortunately, IP Multicast is not yet in widespread use over the Internet backbone (due in part to difficulties in routing the connections appropriately). This limits NPSNET's appeal to closed environments in which the network can be deployed and maintained by a single group (the military, which sponsored the NPSNET work, is a notable user of such technologies). Nonetheless, the goals of NPSNET are similar to the goals outlined for CHIME. The lessons learned in the creation of the NPSNET and other scalable virtual environment infrastructures should be useful informants to future work on CHIME.

CHAPTER 6 Designing a Theme In Depth

The secret of being a bore is to tell everything.
— Voltaire

In the last chapter, we described the CHIME ThemeManager component and the various ways in which the CHIME framework interacts with Theme Plugins. In this chapter, we hope to make that discussion a bit more concrete by describing the process we followed in developing a number of Themes for the CHIME system. These range from small Themes developed to exercise and demonstrate various portions of the framework during its development to a large Theme for a software demonstration we developed for a major governmental research funding agency. This particular demonstration revolved around a nuclear power plant software malfunction, which is reported by the plant manager to the plant builder (Westinghouse, Corp.), who then assigns a software maintenance person to look into the malfunction. The Westinghouse engineer uses his CHIME environment to identify the malfunction and release a software patch to the power plant manager, who successfully applies it and saves the day. This chapter proceeds as follows: first, we discuss a number of smaller Themes we built, paying particular attention to the lessons we learned about the CHIME framework while doing so. Next, we discuss in more detail the scenario around which we built our large Nuclear

Power Plant Theme, known as the NuclearTheme. Finally, we outline the process we followed in creating the NuclearTheme, including the layout of the various room types used in the Theme, finding and modifying 3d models to be used in the demonstration, and the process of actually implementing the Theme, which included both Server- and Client Theme modules.

6.1 Initial Theme Experiences

During the course of the development of the CHIME framework, we needed to build a number of prototype themes in order to explore a number of the issues we encountered. In addition, we built some themes which were used solely as testbeds for further work on the framework itself. These included themes which allowed us to quickly test out various features of the framework, ranging from basic functionality (making sure connections between rooms were handled correctly, debugging support for avatars and their positioning inside CHIME worlds, debugging various Xanth-related problems, etc.) to more advanced components of the framework, including annotation and team support features. In this section, we will describe briefly some of the more notable themes which we built, and discuss lessons learned from the construction of each one.

6.1.1 Star Wars Theme

The initial CHIME Theme that we built was affectionately entitled the "Star Wars Theme." This Theme utilized 3d models of characters, spaceships, locations, and buildings from the original Star Wars movie to depict artifacts from the underlying project. The 3d models were chosen for their humorous value; most people who were shown initial demos of the CHIME framework using the Star Wars Theme enjoyed seeing project artifacts depicted as Darth Vader or Stormtrooper models. The Star Wars Theme was built entirely as a vehicle for testing basic CHIME functionality (after all, it was the first theme built in the course of our research).

As we mentioned above, the Star Wars theme used humorous 3d models as components of the Virtual Environment. While this did add to the enjoyment of all those with whom we shared early glimpses of the CHIME framework, it was not at all useful as a Theme which might actually be used for real work. In fact, even during the extremely early stage of CHIME development in which the Star Wars Theme was created, it became readily apparent to us that in order to be useful as collaborative environments, and in order to support the CHIME framework goal of providing an immersive experience within project artifacts, Themes built to work with CHIME would have to be well thought out. Simply throwing together a Theme built around 3d models we found on Star Wars fan sites did not create a useful environment, specifically because little thought was put into the layout of the world, the uses of each particular model (i.e. making sure particular models represented the same kinds of project artifacts in each room they were used in), etc. Despite its many faults, however, the Star Wars Theme played an important role in many early CHIME feature implementations, as it was used as a real testbed for many of the capabilities of the system. Figure 6- 1 contains a screenshot of the Star Wars Theme.

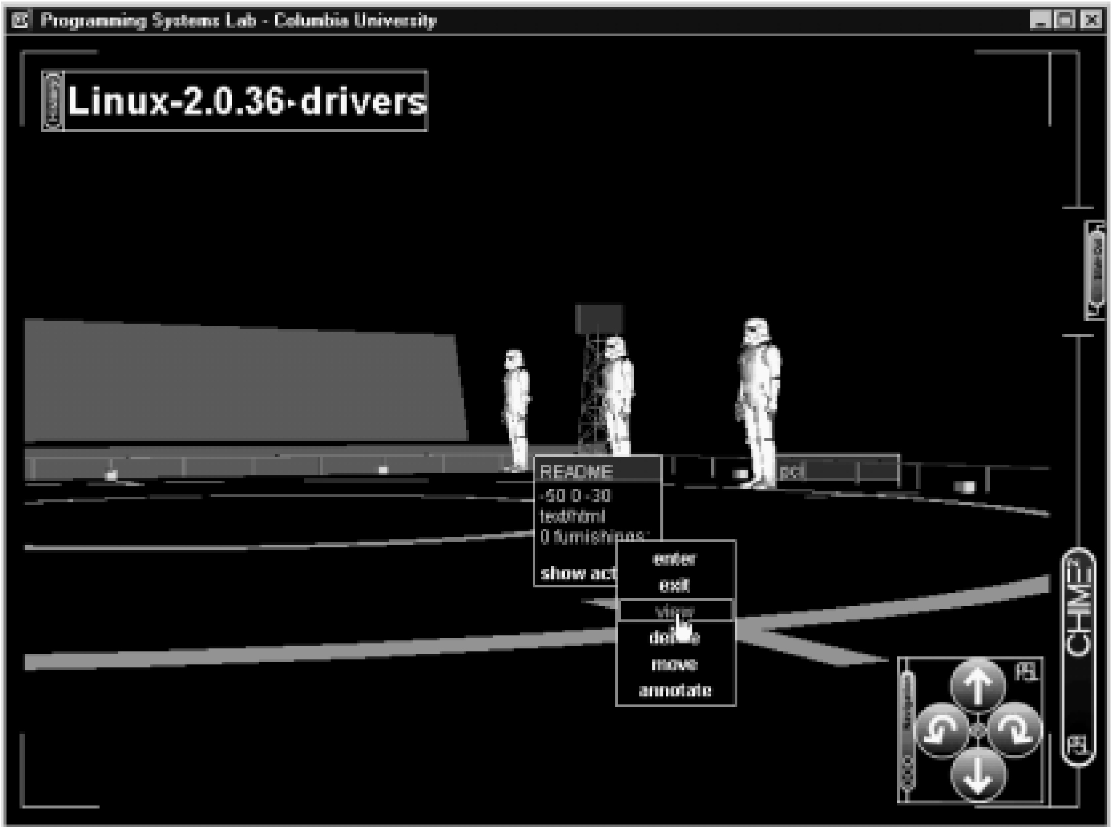


Figure 6-1: The original Star Wars Theme.

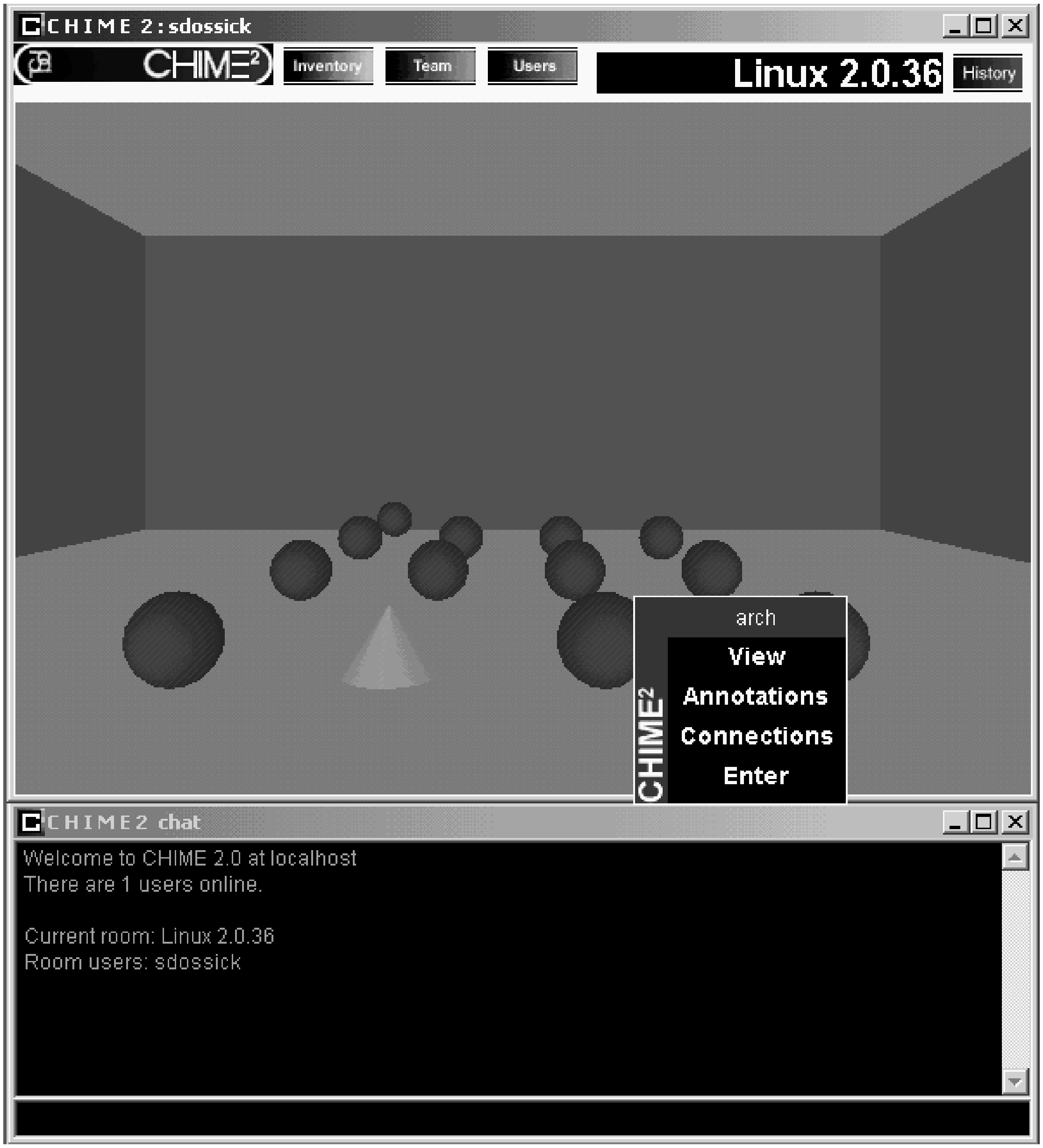
6.1.2 BlockTheme

The next important Theme we created during the development phase of CHIME was one we called “BlockTheme.” It was named BlockTheme because it used no real 3d models, only simple 3d components like spheres, cones, and cubes, to represent project artifacts in the virtual environment. These three particular types of objects are the basic building blocks of all 3d models and are often used in test applications because of their simplicity.

BlockTheme, like the Star Wars Theme before it, paid no attention whatsoever to the problems of creating an immersive virtual environment. It was solely used for feature testing and debugging. Rooms were essentially large boxes, with no decorations on walls or any variation in coloring or other appearance among the various rooms. Artifacts were laid out in neat rows of spheres and cones, with spheres representing other Rooms into which the user could move. Figure 6- 2 contains a screenshot of the BlockTheme.

Despite the primitive nature of BlockTheme, it did allow us to test a number of the capabilities of the CHIME framework. Most importantly, it allowed us to test the ability of the CHIME system client to scale the 3d models for rooms (provided by the Theme) to whatever size necessary to hold the potentially large number of artifacts contained within a given room. For example, a room containing 50 objects will need to stretch in at least one dimension to accommodate all of the artifacts inside itself.

Another important use of the BlockTheme was to prototype and test CHIME’s multiuser abilities. The creation of the BlockTheme was almost simultaneous with the implementation of CHIME’s avatar and chatting capabilities. Thus we used BlockTheme to test (and occasionally demonstrate) the use of avatars to represent other users in the virtual world, and to ensure that when a user moved through the world, his or her avatar moved accordingly on other users’ displays. We also used BlockTheme while we tested the CHIME clients’ chatting abilities, which allow users to send mes-



```
Welcome to CHIME 2.0 at localhost
There are 1 users online.

Current room: Linux 2.0.36
Room users: sdossick
```

Figure 6-2: CHIME client running the BlockTheme.

sages which are heard by all other users in their current room, or to send private messages to a single other user.

6.1.3 AnteRoom

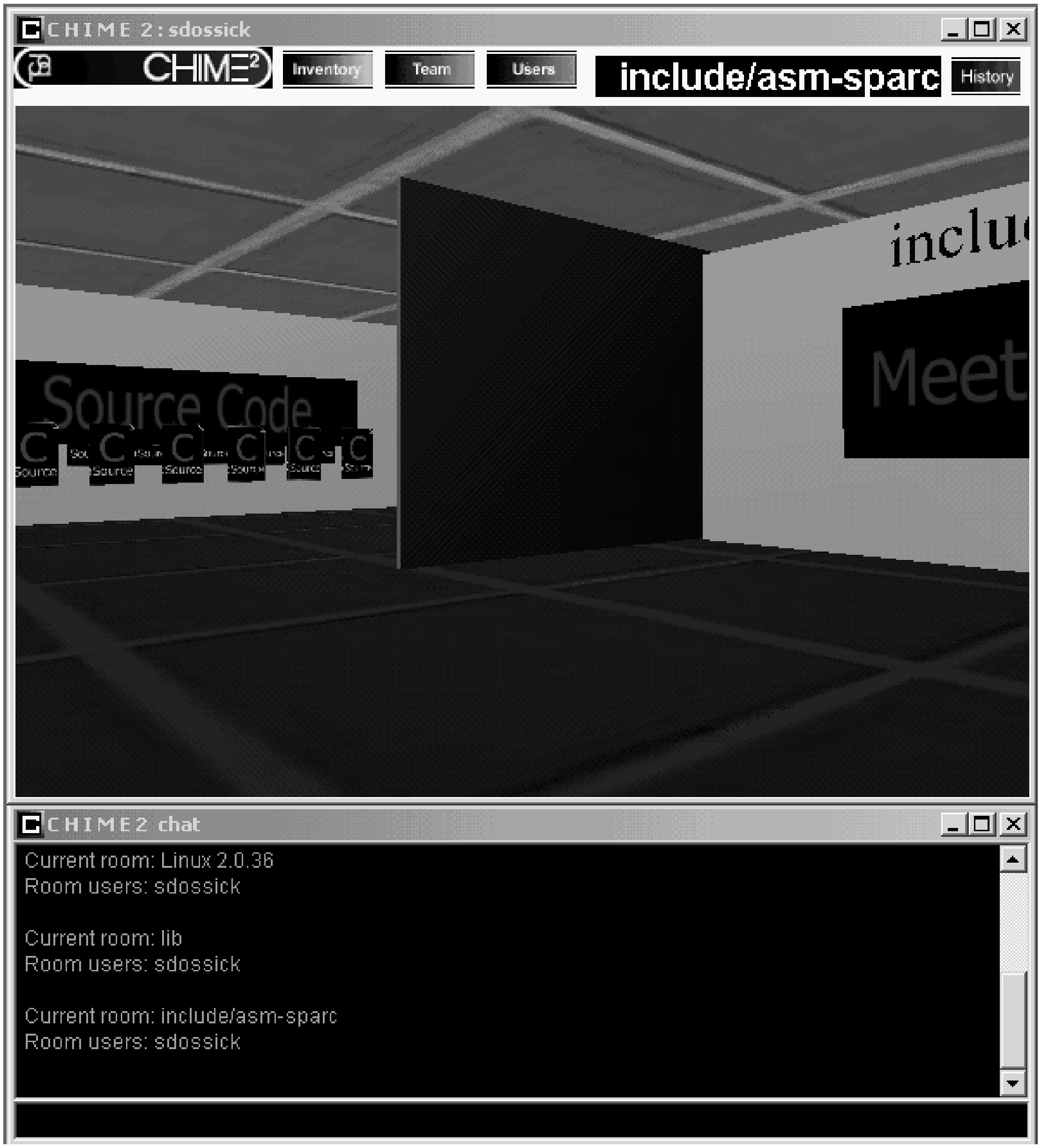
AnteRoom was a significant theme as it was built to test a number of different CHIME framework components for scalability in a variety of ways. In particular, we wanted to test the scalability of Xanth, the VEM, and our CHIME AnnotationService when handling tens of thousands of project artifacts. In order to ensure that the framework could support large scale software development efforts, a realistic scaling test was needed.

We settled on the publicly available Linux kernel source code and ancillary documentation as our test case for a medium- to large- software development effort. In particular, we loaded Xanth and the VEM with the source code from the Linux 2.0.36 kernel source tree, which involved over 1.4 million lines of source code stored inside approximately 8,000 total source code (both C source and header files) artifacts. In addition to the source code, we also stored ancillary project documentation, including information included with kernel releases, documentation from publicly available WWW sites which focused on particular parts of the kernel (e.g. the SCSI or TCP/IP subsystems), as well as messages from email archives of the various kernel- and system-level development mailing lists used to coordinate and discuss ongoing development efforts.

We named this theme “AnteRoom” because of its particular design. We realized that in addition to providing us with extremely useful scalability metrics for Xanth and other CHIME framework services, the creation of a CHIME world around such a large development project was a unique opportunity for us to begin to explore in more depth some of the nuances of Theme creation. Thus AnteRoom was the first theme in which we paid any sort of attention to the details of the virtual world, trying in part to at least build an environment which people might actually be able to perform real work inside. Figure 6- 3 contains a screenshot of the AnteRoom theme running with the Linux kernel source code project artifacts. As you can see, each room created by the AnteRoom theme has a number of different sub-areas, one each for documentation, source code, connections to related rooms, as well as an open space designed for avatars to meet and chat in.

As you might expect, this first attempt at designing a useful Theme was not a raging success. We made a critical error in not realizing the amount of effort involved in the creation of a useful Theme in which developers might immerse themselves. Thus we devoted only a small part of our development schedule to the creation of the theme; because of this, we barely scratched the surface in a number of important areas that we later realized were crucial to the success of any Theme. A tremendous amount of work must be put into the room layouts and other aspects of the Theme in order to make it easy for developers to use and work in.

The AnteRoom Theme did serve its initial purpose in letting us test scalability of Xanth and other framework components. (After a number of small code enhancements to address some performance problems with large artifact bases, Xanth was able to handle over 100,000 artifacts with no problems on standard server hardware). How-



```
Current room: Linux 2.0.36
Room users: sdossick

Current room: lib
Room users: sdossick

Current room: include/asm-sparc
Room users: sdossick
```

Figure 6-3: CHIME client running the AnteRoom Theme.

ever, based on our poor initial experiences with designing a more useful Theme, we developed the requirements for Theme creation discussed in the previous chapter. These requirements are quickly summarized here:

Theme Metaphor: The Theme must be built around a metaphor which governs the mapping of project information and artifacts into the creation of the virtual world. This requirement is the most direct result of our initial experiences with building simple Themes. Without a lot of thought put into the metaphor used by the Theme to build the virtual world, we created environments which could not possibly be used for real work. The metaphor represents the closest thing to “look and feel” of the virtual environment, and so quite a bit of attention must be paid to its design and planning.

Layout of DataElements at startup: A more concrete requirement is that the Theme must utilize the Xanth DataElement hierarchy at startup time, as well as the Virtual Environment hints found in the VEM service to define a layout for the virtual world. The Theme must make good use of the layout of the DataElements as an extension of the metaphor in use. The project artifacts must be located in places within the virtual world which make sense based on their role within the project.

Continual update of virtual world based on DataElement changes: A related requirement is that the virtual world created by a Theme Plugin must constantly evolve as changes are made to DataElements in the Xanth hierarchy. These updates to the virtual world must be in accordance with the Theme metaphor in use. Users must be able to predict fairly accurately where new DataElements will appear and how a particular change will be represented in the virtual world.

User interface mechanisms for artifact and user interaction: A final requirement for Theme Plugins is that they must provide mechanisms through which the virtual world inhabitants can interact with the project artifacts the DataElements represent as well as with each other. The Theme is responsible for making sure that users in the virtual world are able to use the artifacts in natural ways, for instance making hypermedia links between source code and related documentation. Depending on the metaphor chosen for a given Theme, this capability may be accessed differently (choices on a pop up menu in one Theme, or drawing a line between the objects in another Theme).

Our initial forays into Theme creation were fruitful in a number of ways. They allowed us to test and debug a variety of features and capabilities of the CHIME framework, ranging from basic functionality all the way to scalability testing. In addition, we learned that the amount of work which must be done to create a useful Theme is non-trivial. It would be wonderful to be able to claim that using the CHIME framework would make creation of useful project environments as easy as clicking a few buttons. Unfortunately, this is not the case. In order to reap the benefits from Software Immersion, much care must be put into the design and implementation of the Theme which the users will interact with. From our initial Theme work, we were able to define a number of useful requirements which can guide Theme designers to the creation of useful virtual environments.

6.2 NuclearTheme

As we mentioned briefly earlier in this chapter, the NuclearTheme was developed primarily as a demonstration vehicle for the CHIME framework. NuclearTheme was used in a successful demo of our research groups' work done for one of our major governmental agency sponsors. Because NuclearTheme was designed to show off the flexibility and capabilities of the CHIME framework, as well as to showcase the concept of Software Immersion, it was by far the largest Theme we had created to that point in our work on CHIME. In this section, we will detail the scenario used in the demonstration with NuclearTheme, as well as detail the development process. In particular, we will discuss how NuclearTheme meets all the requirements laid out for the creation of useful Themes, as well as discussing what lessons we learned from creating it.

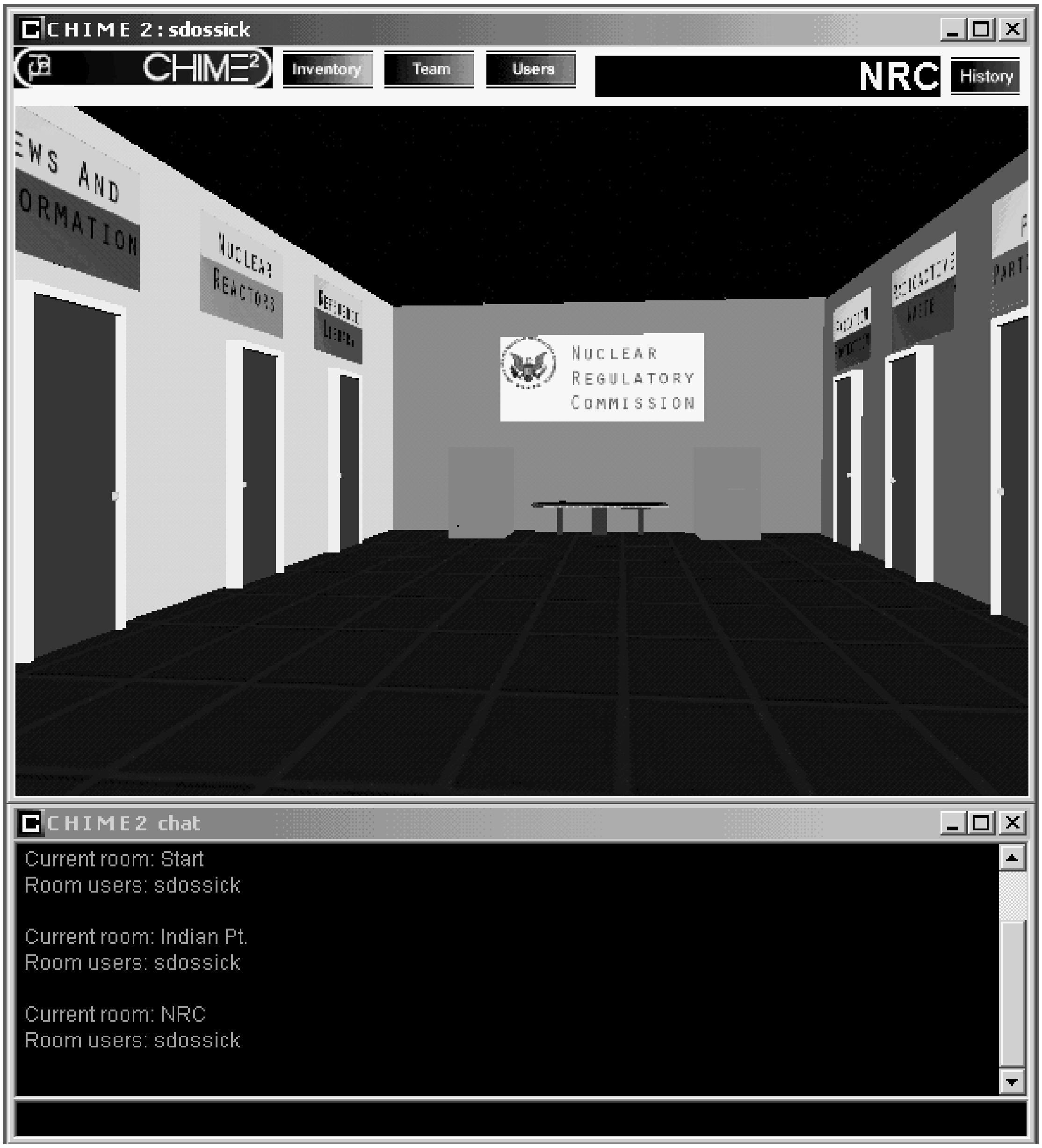
6.2.1 The Scenario

As we mentioned previously, NuclearTheme is centered around a software problem at a nuclear power plant (identified in the demonstrations as the Indian Point Nuclear Facility, nine miles north of New York City). By law, all nuclear plants have a live telemetry link with the Nuclear Regulatory Commission in Washington, D.C. If this telemetry link should fail for more than a period of 24 hours, the plant must shut down its reactor and suspend operations until the telemetry link can be restored. Clearly, this is something the power plant owners would like to avoid; when their plants are not capable of generating power for their customers, they must purchase power from other providers to resell to the public. In addition, starting a nuclear reaction requires more nuclear fuel than continuing an existing one, and as nuclear fuel is quite expensive plant owners would like to avoid stopping and starting the reaction process as much as possible.

For our demonstration scenario, we posited that software inside the telemetry link controller (an embedded device treated as a “black box” by plant workers) encounters a problem and the link to the NRC goes down. The plant manager discovers this problem on his normal round of system checks performed every morning when he begins work. These system checks are done inside of the CHIME virtual world; Xanth plugins have pulled information from a variety of plant systems and the Theme assembles the information into a coherent operational picture.

Concerned that the link has gone down, the plant manager decides to double check the NRC guidelines for the proper procedures in this type of situation. He is able to jump over to another CHIME server being run by the NRC, which contains a virtual world generated automatically from their web site. Figure 6- 4 shows a picture of this virtual environment. As you can see, various sections of their site are organized as doorways along a corridor. As new sections are added or older ones deleted or reorganized, the length of the hallway and the number of doorways will automatically change to compensate for modifications in the underlying information. Our plant manager decides to check the NRC Reference Library, a room entered through a doorway on the far left wall. Upon entering, the plant manager is presented with a room filled with cubicles (see Figure 6- 5 for a screenshot of this). This room is again automatically generated from the contents of the reference library section of the NRC's public web site. Each subsection is represented by its own cubicle; by entering a cubicle the plant manager is given access to the actual documents found on the NRC web site. The plant manager navigates to the emergency procedures cubicle and asks to see this portion of the web site. It is displayed in a web browser window alongside his CHIME client. Through this mechanism of a hallway and then a large library-style room, CHIME has made it easier for the plant manager to zero in on just the particular documentation he needs for his work; as you might imagine the NRC web site is large and complex, which makes it hard to find needed information quickly. The CHIME framework's ability to modify the virtual world based on changes in the underlying information in the Xanth DataElement hierarchy means that the virtual world through which the plant manager navigates will never be out of sync with the actual NRC web site.

Once the plant manager has checked the NRC regulations for the procedure to follow when the telemetry link fails, he jumps over to a third CHIME server run by Westinghouse Corp., the builder of the Indian Point facility. Once there, the plant manager logs a problem report regarding the telemetry link failure. The action in the demonstration then shifts over to a Westinghouse software engineer who is responsible for looking into the problem.



```
Current room: Start
Room users: sdossick

Current room: Indian Pt.
Room users: sdossick

Current room: NRC
Room users: sdossick
```

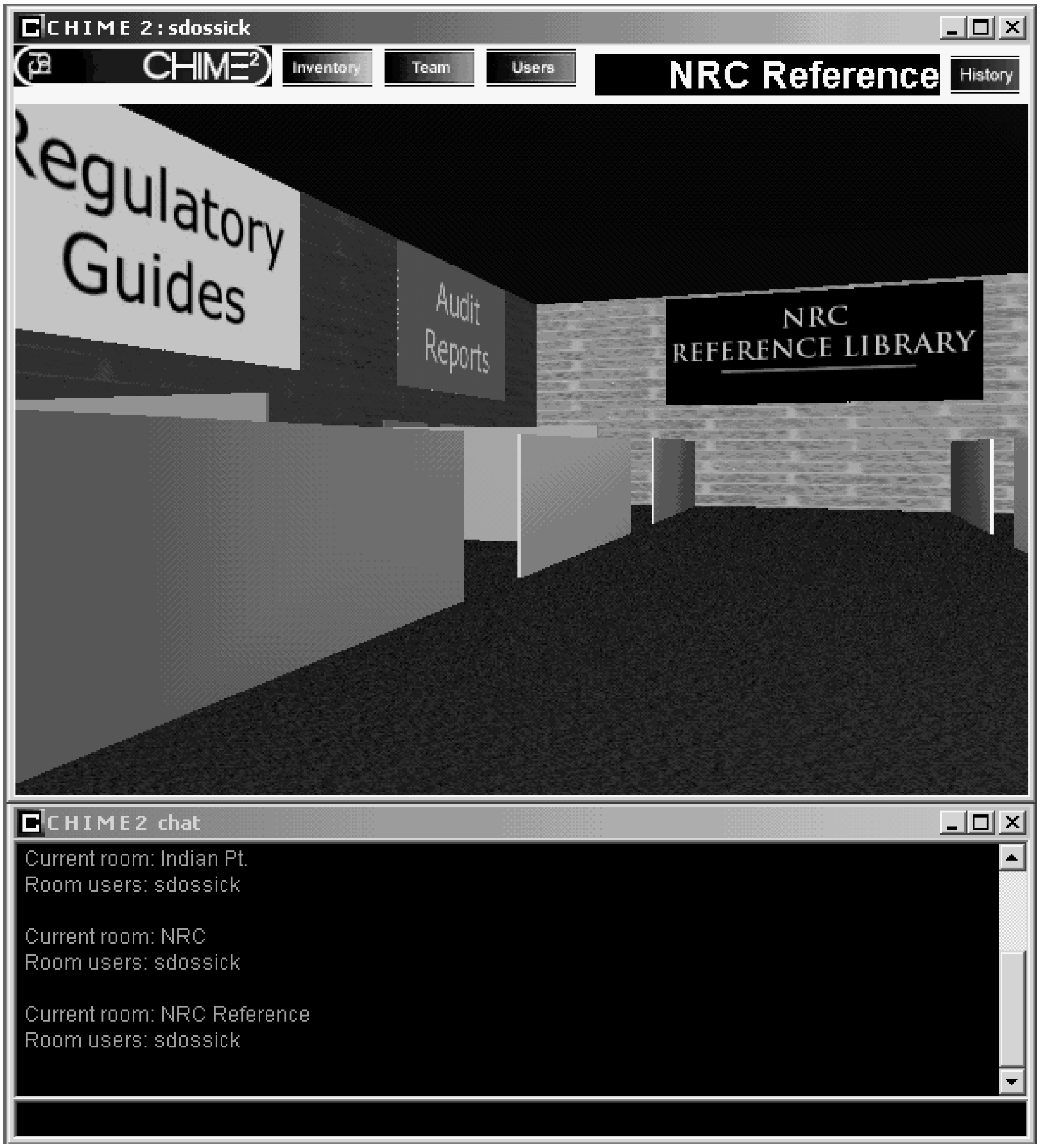
Figure 6-4: Hallway generated from NRC.gov web site by the Nuclear Theme.

The Westinghouse engineer proceeds to walk through the Westinghouse CHIME virtual world into a room containing all the project artifacts related to the software running inside the telemetry controller found at their nuclear power plants. At his fingertips he has access to source code, design documentation, problem reports and solutions archives, test plans, and numerous other kinds of artifacts which might be useful in helping him decipher and solve the software problem reported by the plant manager. In addition, he has direct access to the source code repository used to hold all the patch releases made over the past months. In the demonstration, the software engineer uses a variety of tools to analyze the problem, make a code change, test it, and release a patch to the Indian Point site. Once the patch has been installed, the plant manager checks his monitoring display and discovers that the telemetry link is now operational once again. CHIME and Westinghouse software engineering have saved the day.

6.2.2 Theme Requirements

In previous sections, we have outlined several requirements for CHIME Themes. These requirements were motivated by our initial experiences building several smaller Themes, and relate directly to the user experience inside the virtual world created by the Theme. In this section, we will discuss how the NuclearTheme meets all of these requirements.

Theme Metaphor: The Theme must be built around a metaphor which governs the mapping of project information and artifacts into the creation of the virtual world. The NuclearTheme takes place in a fairly large virtual world made up of rooms in three separate CHIME servers, interlinked via our alliance mechanism. The metaphor in use is that of a fairly standard office park-type setup (see Figure 6- 6 for a



Current room: Indian Pt.
Room users: sdossick

Current room: NRC
Room users: sdossick

Current room: NRC Reference
Room users: sdossick

Figure 6-5: NRC Reference Library, generated from another portion of the NRC.gov web site.

screenshot which makes the relationship among the various servers clearer), where tools, artifacts, and procedural reference documentation are all available inside various rooms. The worlds are segmented along organizational boundaries; the NRC information is contained within an NRC CHIME server, while the Westinghouse work takes place on their own server. For the goals of this particular Theme, which was to showcase the major features of the CHIME framework (as well as a variety of other tools developed by our research group which were used by the Westinghouse engineer), it was important that the Theme be extremely simple for a demonstration viewer unfamiliar with the project to grasp. Thus we chose to stick with a fairly normal world environment which would be readily accessible and understandable to our target audience.

Layout of DataElements at startup: In the NuclearTheme, we lay out a variety of information inside the virtual world in order to provide easier access to it for the characters in the demonstration. In particular, in the NRC rooms we automatically lay out a variety of information from the public NRC web site in real time when the virtual world is created. In the Westinghouse rooms, we add artifacts from the Westinghouse code and document repositories.

Continual update of virtual world based on DataElement changes: As we mentioned above in our description of the scenario behind the NuclearTheme demonstration, changes occurring in the back-end information systems from which we draw a number of the project artifacts in the virtual world are reflected immediately. In particular, if the NRC was to change the structure of its web site, our hallway representing their top-level categories would stretch or shrink, adding or deleting doorways to the subsections of their site. In addition, the room used by the Westinghouse software engineer to diagnose and solve the problem with the telemetry module code is popu-

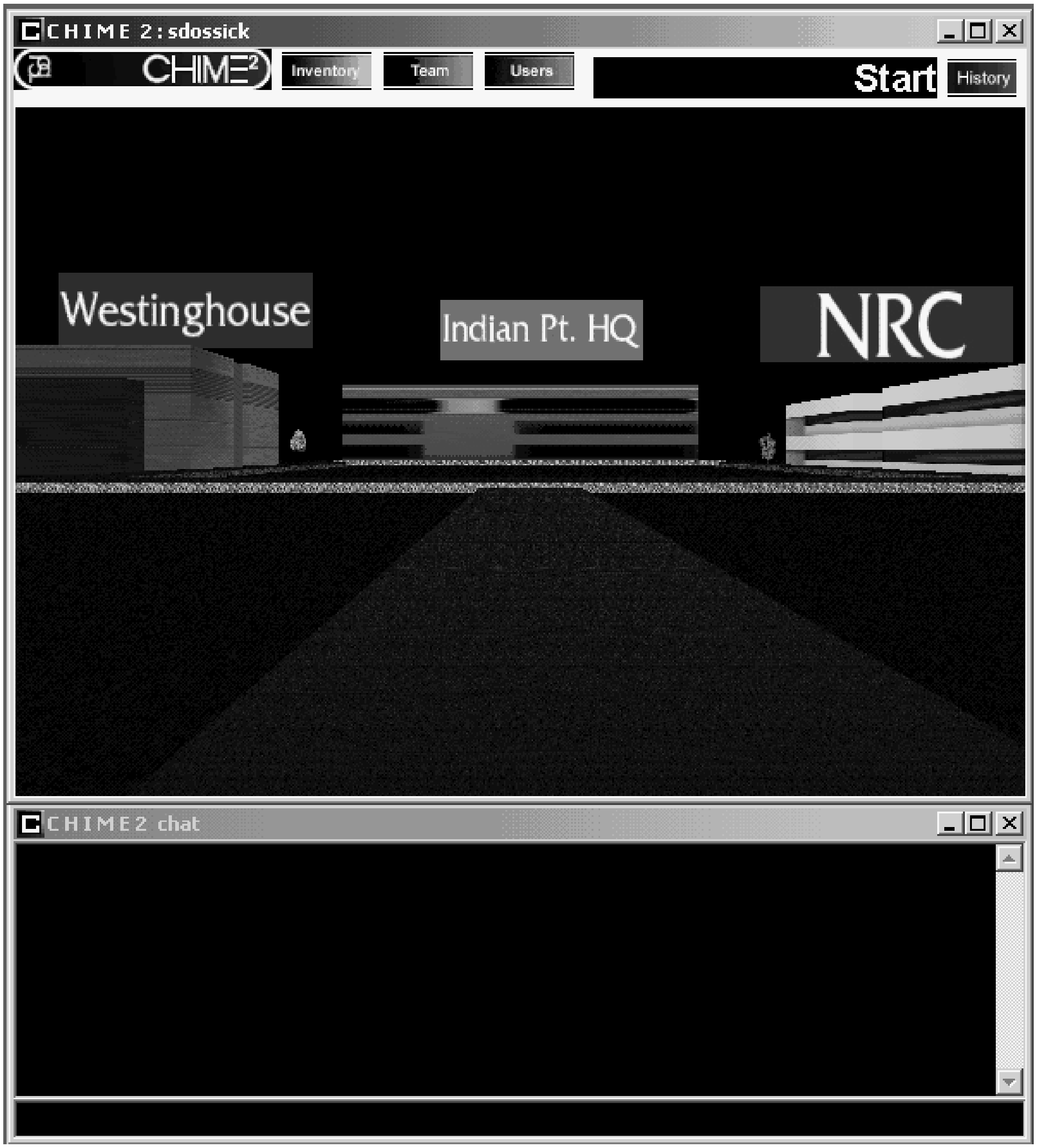


Figure 6-6: The office park metaphor chosen for the NuclearTheme.

lated with artifacts from the underlying code and document repositories. When the engineer modifies code and generates a new software patch in the configuration management system, and when he writes documentation describing the new patch, all the new artifacts are immediately reflected into the room.

User interface mechanisms for artifact and user interaction: As we mentioned earlier, the purpose of the NuclearTheme was to introduce the concepts of the CHIME framework in a very small amount of time to people totally unfamiliar with it. As such, many of the choices we made when designing the Theme tended toward standard sorts of metaphors which people would readily grasp. Thus the user interface mechanisms chosen for the NuclearTheme are standard sorts of menus and other elements which users are comfortable with.

6.2.3 Creating the NuclearTheme

We followed a fairly straightforward process in the creation of the NuclearTheme. First and foremost, we identified the goals of the Theme, which were mainly to demonstrate the capabilities and flexibility of the CHIME framework. In addition, we needed to be able to capture a demonstration viewers' attention quickly, and be able to give them a broad overview of the research we had undertaken in a very small time frame (typically, viewers spent 5-10 minutes looking at our work). A third goal was to demonstrate to viewers the benefits of using virtual environments for real work. Although the NuclearTheme did not involve a large number of users working together, we felt that by placing so much information within the easy grasp of the characters in the demonstration, our viewers would at least glimpse the potential usefulness of these kinds of collaboration environments.

Once we had identified the goals of the Theme, we began sketching on paper possible layouts for the various rooms we wanted for our virtual world, and started brainstorming ideas regarding look and feel for the metaphor embodied in the Theme. Our original concepts for the virtual world involved much more futuristic concepts than the simple office park we ultimately chose as our metaphor. We settled for something extremely down to earth both to make it easier for a viewer unfamiliar with our concepts to grasp CHIME's capabilities and also to remove some of the stigma of "playing video games while working." A question we were repeatedly asked during our demonstrations was whether "there are guns in this, and can we shoot our code and bugs?", referring to popular 3D video games in which players run around with weapons and shoot at other players. Clearly, people associate the use of 3D technology with game applications; by placing our virtual world in a simple office park setting we focus the demonstration viewers' attention more on the real-work aspects of the system.

Once the layout had been decided, the next task was to find 3D models to use for the various components of the virtual world. A number of publicly available collections of 3D models made it easy for us to find components to use. Simple 3D model editors were then used to modify the public domain elements for our use (changing colors, sizes, etc.) Since our office park resembled the real world rather closely, it was easy to find ready-made objects to use. In addition, our limited graphical abilities were called on to design 3D models for objects we could not borrow from others.

Following the location of 3D models to use, the next step was implementing the various Xanth Protocol Access Modules (PAMs) we needed to access the back-end information repositories and other software we would be utilizing within the demonstration. As expected, we ran into no real problems with this task. For the demonstration, we were disconnected from the Internet, and so had to make local copies of a number of artifacts which are normally publicly accessible, most importantly the Nuclear Regulatory Commission web site, which contains all the procedures and regulations utilized inside a running power plant, and which our plant manager refers to during the demonstration. In addition, we used a number of back-end information systems with which we were already familiar, including CVS, the open-source configuration management system, a public domain HTTP-based document repository, and a simple relational database for storing bug reports and other bug tracking information.

Once we had implemented the PAMs we needed, the final step was actually implementing the Theme plugin code itself. This too was a straightforward task, as we were able to rely on CHIME framework services (Xanth, VEM, and ThemeManager) for all of the “hard” work of creating a virtual environment and accessing the back-end data.

6.2.4 Lessons learned from NuclearTheme

The NuclearTheme was the largest CHIME Theme we had created to that date in our research. It taught us a number of lessons regarding the creation of Themes in general, as well as about the CHIME framework itself.

Creating the NuclearTheme validated the requirements we had set forth for Themes based on our previous experiences building smaller, more research-oriented Themes. By focusing on the metaphor in use first and foremost, we were able to generate a clear picture of the various components of the virtual world we were creating before we had begun drawing layouts, finding or designing models to be used in the world, or implementing PAMs for Xanth. This forced us to really think about the goals and motivations for the various parts of the Theme ahead of time, before making any real low-level design choices.

In addition, the NuclearTheme was an excellent example of the process which would be undertaken in the creation of a medium or large sized Theme. In this respect, it acted as an extremely good test case for the CHIME framework. Since we were using it for an important demonstration for a funding agency, building the NuclearTheme allowed us to think about the services contained within the framework from the perspective of someone building a Theme to handle mission-critical work. Had there been any major flaws or holes in the capabilities of the framework, attempting to build something like the NuclearTheme demonstration would have brought them to light quickly.

CHAPTER 7 Conclusions and Future Work

In this dissertation, we have examined the motivations, design, and implementation of CHIME, a framework for the creation and exploration of Software Immersions. Software Immersions are a novel combination of Software Development Environments, Collaborative Virtual Environments, and Software Visualization systems. In a Software Immersion, software engineers and other project team members are immersed into a virtual environment made up entirely of artifacts (and the relationships among those artifacts) from the development project they are working on. The goal here is to illustrate the relationships among the artifacts of the development process. Software Immersions can aid developers in finding the information they need to accomplish their work easily.

CHIME makes it possible for a software development organization to quickly and easily build a collaboration environment tailored to their particular development methods and processes. The resulting environments built with the CHIME framework are able to perform both the more traditional functions of Software Development Environments as well as utilize the unique strengths of CVEs for the discipline of Software Engineering. CVEs are a natural for supporting work among geographically and temporally distributed work teams, and the CHIME framework has a number of capabilities which make this possible as well.

As we had hoped from the beginning, the CHIME framework scales well between use on fairly small projects up to those involving thousands of artifacts and long development cycles. As we have described in several sections of this dissertation, we have performed a number of small experiments with the framework, involving the creation of CHIME Themes to support a variety of project sizes. While we have not performed true “usability studies” of the ease of creation of virtual worlds with CHIME, anecdotal evidence supports our feeling that CHIME makes it easy to build simple, straightforward virtual worlds, as well as making it possible to build larger, more complex environments as well.

The overriding design principle we followed in the creation of the CHIME framework is that of *flexibility* and *extensibility.* With CHIME, developers of a Software Immersion are able to easily incorporate software project artifacts and tools into the resulting CVE, regardless of where those artifacts reside (i.e. regardless of what organization or software engineering tool retains control over the artifacts). CHIME’s XML-based metadata architecture for describing artifacts and the mechanisms or protocols via which they can be accessed makes it easy to integrate with a variety of information repositories as well as external tools.

CHIME includes the most common building blocks we feel developers of Software Immersions will need to be successful. These include modules handling bi-directional, n-ary, typed hypertext linking among project artifacts, protocol modules for accessing artifacts stored in common Software Engineering tools (including modules handling WWW-based repositories, CVS configuration management repositories, and SQL Databases in addition to more mundane filesystem repositories), as well as capabilities for creating federated, cooperating Software Immersions whereby multiple organizations can collaborate. In addition, we have described in detail several example CHIME Themes which help illustrate the power of our framework.

One of the most important research contributions of the work reported in this dissertation is the concept of Software Immersion. As we have mentioned throughout this dissertation, Software Immersions are based on concepts refined in collaborative virtual environments; the goal is to create an immersive virtual environment in which team members collaborate and perform individual tasks in a virtual space defined by the structure of the software project they are working on. Software Immersions can be built semi-automatically with the CHIME framework, through the combination of framework components and runtime customizations possible with Theme Plugin code.

In addition to Software Immersion, the CHIME framework embodies and instantiates two other models defined during the course of this research, known as Groupspaces and Groupviews. The main goal of the framework is to allow software developers to generate useful Software Immersions for their projects quickly and easily. CHIME breaks down the process of creating Software Immersions into three main steps (and three main framework components). First, identification of the data (including source code, design documents, test plans, etc.) which is to be included as part of the eventual Software Immersion. CHIME's Xanth Data Server component is responsible for maintaining this collection of data, as well as any hypertextual links which have been layered on top of this data (for instance, linking source code to the design documentations or testing plans which deal with it). In addition, Xanth acts as lightweight, extensible middleware, allowing other CHIME components access to the back-end data.

The next task of the developer of a Software Immersion is deciding what roles the various pieces of data will play in the eventual virtual environment. CHIME's Virtual Environment Modeler (VEM) allows developers to "tag" each piece of data from the Xanth Data Server with one of an extensible set of Virtual Environment Types. Base types include 'Container,' 'Component,' and 'Connector,' which correspond vaguely with standard virtual environment concepts of 'Room,' 'Object in Room,' and 'Link.' Developers may easily add more VEM types which make particular sense for their application. The core idea of the VEM is to allow Software Immersion developers to add in metadata to each particular piece of data which will aid in the presentation of that data to the users of the system. A particular aspect of the VEM to note is that it does not define, in any way, how the data should be presented to the user; the metadata maintained by the VEM act simply as 'hints' for the next layer(s) of the framework to work with.

The final task in defining a Software Immersion with our framework is the creation of a Theme for displaying and accessing the data. In CHIME, the Theme is responsible for deciding how to display data (e.g. what a particular room looks like, what commands are available to a user for manipulating particular pieces of data, etc.) The Theme component acts as a sort of “late-binding” mechanism for deciding the look-and-feel of the resulting virtual environment. CHIME’s ThemeManager component is responsible for interfacing a particular Theme with the rest of the system, as well as handling collaborative aspects of the Software Immersion (including awareness of other users’ activities, modifications to the underlying data, chatting among users, etc.)

The final component of the CHIME framework is TreatyMaker, a lightweight toolkit for federation of network services. TreatyMaker is based loosely on the International Alliance metaphor described in [Ben-Shaul, 1995] for federation of collaborating workflow systems. In CHIME, TreatyMaker makes possible the “alliance” of multiple organizations allowing seamless collaboration among project participants from all sites. Peer-to-peer federations as well as hierarchical federations are possible, with each site maintaining administrative control over their data and services.

7.1 Future Work

In this dissertation, we have developed a framework that is extremely well suited to fast development of Software Immersions as well as (possibly more importantly from a research standpoint) a good foundation for future explorations and research on the topic. As we mentioned earlier, and which is hopefully obvious from the descriptions of the components contained in this dissertation, an overriding goal of this work was the creation of a flexible and extensible framework.

The work described in this dissertation is comprised of our initial forays into the area of Software Immersions. There are a number of areas which we feel would be fruitful for further exploration in the future, in order to more fully understand the implications of Software Immersions as well as to extend their capabilities in a number of interesting ways.

7.1.1 Transaction Support

An important component missing from the CHIME framework is an integration in some way with a transaction management component. More than likely, this would be a component which allowed for some level of customization in the transaction model to be used, including a number of research transaction systems supporting programmable extended transaction models (our research group has produced a variety of possible candidates over the past few years, including [Yang, 2000] and [Heineman, 1996].

The real research question that we will face when attempting this kind of integration is the exploration of what transactions “mean” inside a virtual world. When the data underlying an object in the virtual world is divorced from the representation, it is hard to describe a policy for handling of transactions. For instance, how does a locking conflict with another virtual world user get represented? What happens when a long-lived transaction involving a series of operations fails to commit? How does the virtual world reflect rollbacks which will occur? Clearly, no one set policy will work for all possible virtual worlds, so a large amount of flexibility and investigation will be required.

Also falling under the general category of transactional support inside the virtual world is support for what we have termed “work units.” There are many cases in which it would be advantageous if the virtual world shared by all users was not 100% synchronized at all times. For example, a wholesale rearchitecting of a particular sub project may affect any number of other team members’ work. It would be great if it were possible for a single team member to queue up changes to the virtual world and then be able to “commit” them, and have them distributed to other users. Only upon a commit would other users’ virtual worlds change to reflect the new work. A feature such as work units allows for a change in the way users’ interact with each other in the virtual space, allowing for both fine grained (supported in the current CHIME framework) and enabling large grained changes to the system. Work units would also allow for an undo facility which is sorely lacking from the existing system.

Once we have integrated a transaction management component into the CHIME framework, a large area for further study will be the investigation of suggested transaction policies for various types of virtual environments. For instance, CHIME worlds used primarily for referencing older project documentation and experiences will have a different set of requirements for the types of transaction models needed from an actively pursued development project. Projects involving small teams will likely benefit from transaction policies tailored for such a situation, and these will likely be different from policies targeted at work done by larger teams or by a number of small teams working closely together.

Finally, adding support for transactions into a virtual world will require real thought and experimentation with regards to the user interfaces supplied to allow virtual world users to lock and unlock objects, commit and abort transactions, rollback the state of objects in the world, etc. While it will be difficult if not impossible to formulate a user interface applicable to all Themes, it would be beneficial to compile a compendium of possible ideas, perhaps as a set of reusable software components which could be easily included Themes.

7.1.2 Workflow and Software Process Support

Another potential area for future work is the integration of a workflow or software process enactment component into the CHIME framework. Workflow automation of mundane tasks based on particular actions inside the virtual environment could be particularly powerful, as we could take advantage of the semantics provided by the Theme to map user actions onto workflow tasks. Instead of users selecting particular workflow tasks from a list or menu as they would in a traditional workflow system, they could interact with the workflow as an artifact found inside the virtual world. This would allow, for example, the workflow to become more tangible to them as it could be used to affect the layout of rooms or artifacts in the virtual world.

Particular rooms could be dedicated explicitly to the performance of particular tasks in the workflow. This would be quite similar to the work reported in [Doppke, et. al, 1998]. In the Promo work, a text-based virtual environment was generated solely from a software process description. Each room in the virtual environment corresponded explicitly to a single portion of the process in use. Users could only enter a particular room if they had accomplished certain tasks previously, and exiting a room could be governed by the successful completion of a particular task (generated from an exit condition in the underlying software process). The Promo work did not account for differences in the virtual environment in the way that the CHIME frameworks' Themes do, and so an open area of research with such an integration would be the way in which workflow or process requirements might be reflected into the virtual world. Here again, similar to the problems of defining a blanket policy for the reflection of transaction model information into a virtual world, it will not be possible to define an explicit mapping between software processes and workflows and their representations in a virtual world. As with the integration of transaction support into the CHIME framework, the appropriate course of action is to look at a variety of possible uses for workflow and software process inside virtual worlds and describe a range of possibilities for use of these components inside a virtual world.

In addition to a Promo-style integration, there are other possibilities for making use of workflow support within virtual environments. Rather than explicitly mapping particular locations of the virtual world onto subportions of the workflow or process in use, the workflow state and other information could be reflected inside the virtual world by using similar sorts of user interface mechanisms as we have described for illustrating project artifact state in previous chapters of this dissertation. For instance, artifacts which are useful for the current task being worked on by a particular user might be displayed with a particular color tint applied to them (another users' display might have artifacts useful for her current task displayed with such a tint). In this way, the work process is integrated directly into the fabric of the world but does not affect layout. This and many other possibilities for integration deserve to be explored more fully.

7.1.3 Collaboration and Team Support Features

An area ripe for future work inside the CHIME framework is the addition of richer collaboration and team support. This can come in a variety of areas. An important aspect which could help the use of CHIME virtual worlds on large development projects would be the ability to mask out the activities and even the artifacts in use by other teams working in the same world. The idea here would be to remove clutter in the virtual world display by removing items which are not of immediate interest to the user.

This would allow a CHIME world user to focus only on users he or she works directly with and on artifacts which are directly relevant to their work.

Related to the possibility of ignoring irrelevant work being performed inside the virtual world is the addition of a capability through which project members could state their level of interest in a particular subproject, and receive more or less information about the current status of other projects based on this setting. Even in situations in which a particular user is working on a number of different tasks simultaneously, at any one moment in time they are likely focusing on one task more than others. Allowing virtual world inhabitants the ability to screen out distractions in this manner seems to be a compelling possibility for the CHIME framework. In many respects, this particular piece of future research is related to the wOrlds projects' Locales framework[Fitzpatrick, 1998]. In the Locales framework, artifacts related to different project tasks are stored in separate "locales"; users can set their level of interest in a particular project to "high bandwidth," "medium bandwidth," or "low bandwidth" depending on how interested they are in receiving updates about a particular subproject. A low bandwidth setting sends only major project updates to their screen; medium and high send more detailed information (with high reserved for tasks the user is currently focused on). A number of user studies have been performed in conjunction with the wOrlds project [Fitzpatrick, et al., 1998], and have successfully validated the usefulness of this approach.

A further capability which would be extremely useful inside the CHIME framework would be the ability to generate overviews of work being performed or of the artifacts and locations in the virtual world most related to a particular subtask. These sorts of “50000 foot views” of the work being (or intended to be) performed inside the virtual world would make it easier for new project team members to come up to speed on the various aspects of the project as well as for members of other teams to keep up with changes and progress in subprojects related or dependent on their own. Allowing the user to see the big picture of the work being performed seems like it would be a fairly straightforward addition to the responsibilities of the Theme Plugin code.

The generalization of this ability would be to make it easy for users to view the virtual world in a number of different ways. The standard virtual world display shows project artifacts separated and organized into distinct rooms, with hypertext links possibly connecting related artifacts. There is no reason why this needs to be the only display mechanism available to the user. Why not, for instance, allow users to organize the virtual world according to their own criteria? Consider for example a virtual world used for a small software project. While it may make sense for developers to have a code-centric view of the virtual world, in which source code is the central component and documentation, test cases, and other related artifacts are organized around the code modules they refer to, this may not be the most efficient organization for the testers. Perhaps for testers, the virtual world needs to be organized with test cases as the central artifact type, and the code and documentation linked to from the main set of test cases. An important potential area for future study is modifications to the CHIME framework allowing these kinds of “views” of the virtual world to be integrated.

When users leave projects for periods of time (for vacations or because they change project teams), an important capability which could be provided by the CHIME framework would be the ability to generate a summarization of the changes made to artifacts inside the virtual world while the user was gone. This sort of summarization would be invaluable both for re-integrating an existing team member after an absence as well as making it easier for new team members to get a sense of the projects’ work history. Summarization is not an easy task for computers to accomplish because in general they do not have much semantic knowledge about the actions being taken by their users, and thus cannot determine “important” changes from less relevant ones. However, Themes provide an important level of semantic knowledge about the particular project a given virtual world is supporting, and so perhaps the correct approach is to allow Theme Plugin code to specify a set of rules detailing what actions are unimportant. A new CHIME framework summarization component could be built to organize changes made to the virtual world according to the rules as set by the Theme code.

A final set of enhancements which could be made to the CHIME framework in the area of collaboration support would be to integrate a number of different mechanisms to allow users to communicate efficiently. Currently, the major form of communications supported is a text-based chat which allows all users sharing a common room to communicate by typing messages to the entire room; it is also possible to send a private message addressed to a single user. In addition, in all of our Themes we have supported the use of external video and audioconferencing software applications to let users communicate. A number of simple enhancements would make the virtual worlds created with CHIME become even more immersive. For instance, the support of a simple buddy-list style instant messaging application, to let a particular user find out which rooms and artifacts co-workers were focused on currently would enhance team support. The ability to let a single user guide a “tour” of other users, in essence taking them on a walkthrough of parts of the virtual world might also be a powerful addition to the capabilities of the framework. Users should perhaps be able to, where appropriate, send pointers to one anothers’ screens regarding important or urgent messages. The set of communications possibilities afforded by the current CHIME implementation must be revisited.

7.1.4 Tool Management

An important aspect of sharing a virtual world which is used for real work is the ability to launch a common set of tools in order to work with project artifacts. Our research group has, in the past, studied the problem of management of external tools in great depth (for instance, [Valetto, 1994], [Valetto and Kaiser, 1995], and [Valetto and Kaiser, 1996]). Previous work in the arena of process-centered software development environments involved the creation of a separate software component charged only with the management of external tools to be used by environment users known as Rivendell. Rivendell attempted to make tools which had particular hardware or other requirements accessible regardless of the computer type or location of the user. For instance, Rivendell could take care of starting up a mainframe tool which could only be run on a particular server computer and redirecting the display of this tool over to the users' desktop. Rivendell accomplished this through a variety of means, and was very customizable in terms of the addition of new tools to its cadre of supported applications. The integration of a Rivendell-like system for managing third-party tools which could be accessed through the virtual environment would be a powerful addition to the CHIME framework.

7.1.5 Automatic Linking of Related Project Artifacts

In the Rationale Capture and Reverse Engineering research areas of Software Engineering, a number of research projects have attempted to tackle the problem of automating the process of drawing connections between related project artifacts. In many cases, it is necessary to reconstruct the relationships among artifacts after the end of a project's life, in particular because none of the original developers are available to help. A variety of research tools, including [Erdem, et al., 1998] as well as a growing stable of commercial tools (including [Autonomy Inc, 2000]) are aimed at solving this problem. While it is not yet possible to achieve 100% success with tools such as these, they do often manage a reasonable categorization of project artifacts. Adding the ability to the CHIME framework to integrate with one or more of this variety of tools seems like it could add a powerful new capability to the CHIME framework. One of the hardest problems faced by the creators of a Theme is the organization of project artifacts, particularly the creation of hyperlinks among artifacts which may not at first glance appear to be related but which in reality are. The use of an automated tool to produce these kinds of linkages, organizations, and categorizations seems like it could go a long way toward alleviating this problem.

7.1.6 Software Visualization

As we mentioned briefly earlier in this dissertation, the young field of Software Visualization attempts to provide graphical representations of a variety of aspects of a software project. Obviously, this is very related to the goals of Software Immersion, which immerse developers and project team members into a graphical environment made up exclusively of project artifacts. To date, the most successful software visualizations have focused on graphically illustrating the workings of algorithms, displaying graphically the execution path of a particular piece of code, or visualizing memory and cpu usage of various subcomponents of a software system. Software visualizations tend to be hand generated by experts in the field; when they have been automatically generated it has been by specifying an algorithms' specifics to a visualization system using a language explicitly designed for this task. A number of research systems [Stasko, et al., 1998] have been created in this area which attempt to make it easy to create software visualizations from simple specification languages.

A powerful addition to the CHIME framework would be an integration with components like these. It would be great if software visualizations could be imported wholesale into CHIME virtual worlds, for inclusion as yet another form of project artifact. In this way, users would be given easy access to visualization tools, which seem to fit naturally into the immersive virtual world environments created with CHIME.

CHAPTER 8 References

Ambite, J., Ashish, N., Barish, G., Knoblock, C., Minton, S., Modi, P., Muslea, I., Philpot, A., and Tejada, S. (1998). ARIADNE: A System for Constructing Mediators for Internet Sources. In *Proc. 1998 ACM SIGMOD International Conference on Management of Data,* Seattle, WA.

Anderson, K., Taylor, R., and Whitehead, E. (1994). Chimera: Hypertext for Heterogeneous Software Environments. In *Proc. 1994 ACM Conference on Hypertext*, Edinburgh, Scotland.

Anderson, K., Taylor, R., and Whitehead, E. (2000). Chimera: Hypermedia for Heterogeneous Software Development Environments. *ACM Transactions on Information Systems,* vol. 18, no. 3, July 2000.

Autonomy, Inc. (2000). *The Technology Behind Autonomy.* Information available at http://www.autonomy.com/.

Baru, C., Gupta, A., Ludascher, B., Marciano, R., Papakonstantinou, P., Velikhov, P., and Chu, V. (1999). XML-Based Information Mediation With MIX. In *Proc. 1999 ACM SIGMOD International Conference on Management of Data*, Philadelphia, PA.

Benford, S., and Fahlen L. (1994). Supporting Co-operative Work in Virtual Environments. *The Computer Journal*, vol. 37, no. 8, Oxford, UK.

Benford, S., Colebourne, A., O'Brien, J., Rodden, T., and Snowdon, D. (1997). Informing the Design of Collaborative Virtual Environments. In *Proc. ACM Group '97,* Phoenix, AZ.

Ben-Shaul, I. (1995). *A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment.* PhD thesis, Columbia University.

Ben-Shaul, I. and Kaiser, G. (1995). *A Paradigm for Decentralized Process Modeling.* Kluwer, Amsterdam.

Boudier, G., Gallo, F., Minot, R., and Thomas, I. (1988). An Overview of PCTE and PCTE+. In Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Boston, MA.

Brooks, F. (1995). *The Mythical Man Month*. Addison Wesley, Reading, MA, USA, second edition.

Campbell, B., and Goodman, J. (1988). HAM: A General Purpose Hypertext Abstract Machine. *Communications of the ACM,* vol. 31, no. 7, July 1988, pp. 856-861.

Carlsson, C., and Hagsand, O. (1993). DIVE - A Platform for Multi-User Virtual Environments. *Computers and Graphics*, vol. 17, no. 6, June 1993.

Conway, M., Pausch, R., Gossweiler, R., and Burnette, T. (1994). Alice: A Rapid Prototyping System for Building Virtual Environments. In *Proc. 1994 ACM Symposium on Computer-Human Interaction,* Boston, MA.

Curtis, P. (1992). MUDs Grow Up: Social Virtual Reality in the Real World. In *Proc. 1992 conference on Directions and Implications of Advanced Computing*, Palo Alto, CA.

Darken, R. (1995). Wayfinding in Large-Scale Virtual Worlds. In *Proc. 1995 ACM Symposium on Computer-Human Interaction,* Denver, CO.

De Pauw, W., Helm, R., Kimelman, D., and Vlissides, J. (1993) Visualizing the Behavior of Object-Oriented Systems. In *Proc. ACM OOPSLA 1993*, Washington, D.C.

Doppke, J., Heimbigner, D., and Wolf, A. (1998). Software Process Modeling and Execution within Virtual Environments. *ACM Transactions on Software Engineering and Methodology*, vol. 7, no. 1, January 1998, pp. 1-40.

Electric Communities, Inc. (2000). The Palace. Information available at http://www.thepalace.com/.

Erdem, A., Johnson, W., and Marsella, S. (1998). Task Oriented Software Understanding. In *Proc. 1998 International Conference on Automated Software Engineering, Honolulu, HI*

Fahlen, L., Brown, O., and Carlsson, C. (1993). A Space Based Model for User Interaction in Shared Synthetic Environments. In *Proc. 1993 ACM INTERCHI Conference on Human Factors in Computing Systems,* Amsterdam, The Netherlands.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). *Hypertext Transfer Protocol -- HTTP/1.1*. IETF RFC 2616. Information available at http://www.w3c.org/.

Fitzpatrick, G. (1998). *The Locales Framework: Understanding and Designing for Cooperative Work*. PhD thesis, University of Queensland.

Fitzpatrick, G., Kaplan, S., and Mansfield, T. (1998). Applying the Locales Framework to Understanding and Designing. In *Proc. 1998 Conference on Computer Supported Cooperative Work*, Seattle, WA.

Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., and Widom, J. (1995). Integrating and Accessing Heterogeneous Information Sources in TSIMMIS. In *Proc. 1995 AAAI Symposium on Information Gathering*, Stanford, CA.

Greenhalgh, C., and Benford, S. (1995). MASSIVE: A Collaborative Virtual Environment for Teleconferencing. *ACM Transactions on Computer-Human Interaction*, vol. 2, no. 3, September 1995, pp. 239-261.

Goland, Y., Whitehead, E., Faizi, A., Carter, S., and Jensen, D. (1999). *HTTP Extensions for Distributed Authoring: WEBDAV*. IETF RFC 2518. Information available at http://www.webdav.org/.

Halasz, F., and Schwartz, M. (1994). The Dexter Hypertext Reference Model. *Communications of the ACM*, vol. 37, no. 2, February 1994, pp. 30-39.

Heineman, G. (1996). *A Transaction Manager Component for Cooperative Transaction Models.* PhD thesis, Columbia University.

Hogan, D. (1998). *Modeling Construction Cost Performance: A Comprehensive Approach Using Statistical Art.* PhD thesis, Columbia University.

Id Software, Inc. (1995). DOOM. Information available at http://www.idsoftware.com/.

Id Software, Inc. (1997). Quake II. Information available at http://www.idsoftware.com/.

Instinctive Corp. (2000). eRoom. Information available at http://www.instinctive.com/.

Jerding, D., and Stasko, J. (1995). Using Information Murals in Visualization Applications. In *Proc. 1991 Human Factors in Computing Systems Conference*, New Orleans, LA, USA.

Jerding, D., Stasko, J., and Ball, T. (1997). Visualizing Interactions in Program Executions. In *Proc. 1997 International Conference on Software Engineering: Pulling Together*, Boston, MA.

Kaiser, G., Dossick, S., Jiang, W., and Yang, J. (1997). An Architecture for WWW-based Hypercode Environments. In *Proc. 1997 International Conference on Software Engineering: Pulling Together*, Boston, MA.

Loomis, C. (2000). Employees Are Churning Up All Over The Place. *Fortune*, vol. 142, no. 2, February 2000, pp. 127-129.

Macedonia, M., Zyda, M., Pratt, D., Barham, P., and Zeswitz, S. (1994). NPSNET: A Network Software Architecture For Large Scale Virtual Environments. *Presence,* vol. 3, no. 4, Fall 1994.

Mansfield, T., Kaplan, S., Fitzpatrick, G., Phelps, T., Fitzpatrick, M., and Taylor, R. (1997). Evolving Orbit: A Progress Report On Building Locales. In *Proc. ACM Group '97*, Phoenix, AZ.

Microsoft Corp. (2000). Microsoft Visual C++. Information available at http://www.microsoft.com/visualc.

MIRC, Inc. (2000). Introduction to Internet Relay Chat. Information available at http://www.mirc.com/irc.html.

Poltrock, S., and Engelbeck, G. (1997). Requirements for a Virtual Collocation Environment. In *Proc. ACM Group '97,* Phoenix, AZ.

Programming Systems Lab. (1995). Darkover 1.0 Manual. Technical Report CUCS-023-95e, Columbia University, New York, NY.

Reiss, S. (1995). *The FIELD Programming Environment: A Friendly Integrated Environment for Learning and Development.* Kluwer Academic Publishers, Boston, MA.

Reiss, S. (1996). Simplifying Data Integration: the Design of the Desert Software Development Environment. In *Proc. 18th International Conference on Software Engineering*, Berlin, Germany.

Reiss, S. (1998). Software Visualization in the Desert Environment. *ACM SIGPLAN Notices*, vol. 33, no. 7, pp. 59-66.

Roseman, M., and Greenberg, S. (1996). Building Real Time Groupware with GroupKit, A Groupware Toolkit. *ACM Transactions on Computer-Human Interaction*, vol. 3, no. 1, March 1996, pp. 66-106.

Sarwar, B., Konstan, J., Borchers, J., Herlocker, B., Miller, B., and Reidl, J. (1998). Using Filtering Agents to Improve Prediction Quality in the GroupLens Research Collaborative Filtering System. In *Proc. 1998 Conference on Computer Supported Cooperative Work*, Seattle, WA.

Skopp, P. and Kaiser, G. (1993). Disconnected Operation in a Multi-User Software Development Environment. In *Proc. 1993 IEEE Workshop on Advances in Parallel and Distributed Systems*, Princeton, NJ.

Stasko, J., Price, B., and Brown, M. (1998). *Software Visualization*. MIT Press, Cambridge, MA, USA.

Strauss, A. (1993). *Continual Permutations of Action*. Aldine de Gruyter, New York, NY, USA.

Sun Microsystems, Inc. (1988). *Introduction to the Networked Software Environment*.

TeamWave Software, Ltd. (2000). Teamwave Workplace. Information available at http://www.teamwave.com/

Tufte, E. (1990). *Envisioning Information*. Graphics Press, Framingham, MA, USA.

Tufte, E. (1997). *Visual Explanations: Images and Quantities, Evidence and Narrative*. Graphics Press, Framingham, MA, USA.

Valetto, G. (1994). *Expanding the Repertoire of Process-based Tool Integration*. MS Thesis, Columbia University.

Valetto, G., and Kaiser, G. (1995). Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments. In *Proc. IEEE 7th International Workshop on Computer-Aided Software Engineering*, Toronto, Canada.

Valetto, G., and Kaiser, G. (1996). Enveloping Sophisticated Tools into Process-Centered Environments. *Automated Software Engineering*, vol. 3, March 1996, pp. 309-345.

Vygotskij, L. (1978). *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, Cambridge, MA, USA.

Wernecke, J. (1994). *The Inventor Mentor: Programming Object-Oriented 3d Graphics With Open Inventor, Release 2*. Addison Wesley, Reading, MA, USA.

Will, U., and Leggett, J. (1992). Hyperform: Using Extensibility to Develop Dynamic, Open and Distributed Hypertext Systems. In *Proc. 1992 European Conference on Hypertext (ECHT ‘92)*, Milan, Italy.

Yang, J. (2000). *An Approach to Cooperative Transaction Services on the World Wide Web*. PhD thesis, Columbia University.