ARCHITECTURE AND INFRASTRUCTURE FOR A DISTRIBUTED DESIGN ENVIRONMENT A CLIENT PERSPECTIVE

by

Francis L. Chan

Memorandum No. UCB/ERL M98/10

17 March 1998

ARCHITECTURE AND INFRASTRUCTURE FOR A DISTRIBUTED DESIGN ENVIRONMENT A CLIENT PERSPECTIVE

by

Francis L. Chan

Memorandum No. UCB/ERL M98/10

17 March 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Acknowledgments

I would like to thank Professor A. Richard Newton, my research advisor, for his invaluable support during my years at Berkeley as a graduate student. He provided constant academic and research guidance and inspired many of the ideas presented in this report. A great communicator and man with rich academic, industry and life experience, Professor Newton has shared with me not only much academic and technical knowledge, but also many lessons of life. I also have to thank him for trusting me and giving me the freedom and flexibility in my research work.

I would like to thank Professor Jan Rabaey for being the reader of my report and giving me much constructive feedback and encouragement.

Mark Spiller, my counterpart in the server area of the client/server sub-group, was someone whom I worked very closely with throughout my graduate career and contributed to many designs decisions documented in this research report. We spent many good and bad times together, at and outside of school. I thank him for his work and the support he has given me in many different aspects of life.

I thank Michael Shilman for introducing me to this research group and for the talks we had in many different areas and discussions of research ideas from high-level system architecture to the font-type of my presentations.

I also wish to acknowledge Wendell Baker, Serena Leung and Jim Young who has helped me in many ways with my work here at Berkeley.

This work was supported in part by DARPA under contract DABT63-95-C-0074-NEWTON-06/96 and Digital Equipment Corporation. Their support is gratefully acknowledged.

Last but certainly not least, I thank my family and friends for supporting me and bearing with me throughout my college and graduate career. I am deeply grateful for their patience and love.

1. Contents

2. Introduction

The continuing advances in computer processing power, storage and network transmission capacity, as well as an ever-increasing amount of information and services available on the Internet indicate that there are great potential advantages for future CAD environments [1, 2] to leverage wide-area networks. Moreover, as design complexity continues to grow exponentially, so do the needs for and opportunities provided by network collaboration and design environments.

In the future, advances in computer and networking technologies will enable the rise of ubiquitous computing [36], applications that dynamically adapt to different hardware capabilities [37, 38], and an increase in the number of network tools and services. Moreover, emerging technologies will enable data migration [39], agent/application migration [40-42], and network data access[43]. In this environment, it will be imperative that distributed applications be scaleable and extensible. In addition, the rise of such distributed design systems will require advanced server features [44, 46, 50] like consistency, fault tolerance, security, and intelligent resource location mechanisms.

On the client side, users of the system should be able to access and execute tools with minimal hardware and software in place (e.g. Java-enabled network browser). Moreover, the working environment should allow remote tool invocation according to users' access criteria and

permission. Flexible open frameworks could be achieved by customizable configuration of data formats and tools from different sources.

This report documents the research work carried out in the high-level design and architecture of a distributed design environment. It also highlights and describes different pieces of client software infrastructure and applications that have been developed to test the technical and usage feasibility of a network design environment. The issues and challenges involved in building such a distributed environment will be discussed and the experience and findings with developing, integrating and deploying the client and server infrastructure will be analyzed.

3. WELD Architecture

3.1 Introduction

The goal of the WELD infrastructure team [97, 99] is to provide a high-level system architecture as well as the software infrastructure that enable and facilitate a distributed design environment. This environment should allow application developers to easily incorporate their tools into the environment and should allow network users to access and flexibly configure the tools and services available.

Throughout the design, implementation, and testing process, not only have we considered the technical challenges, such as communications, connectivity, data consistency and availability, etc., involved in building and deploying a distributed environment, we have also looked into ways of improving the usability, data transmission efficiency and hence the overall performance of a network-based design environment.

3.2 Comparison to Previous Work

The work of providing the architecture and infrastructure of a distributed design environment can be compared closely to that of developing CAD frameworks, which is an area of active research[1-6]. Our system provides many of the features that a CAD framework provides[2], such as:

- design database – provision of a data manager and client object management package,

- design data manager – versioning, security and meta-data[1] handling capabilities and
- design process manager – distributed tool flow manager.

However, our approach differs from others' work in a number of fundamental ways. Most of the past efforts in the area of CAD frameworks is involved in introducing new systems, techniques or extensions in specific areas of design data management [7, 8], design meta data management [9-12] and flow, process and tool management [13-25]. WELD, however, is concerned with providing the connection and communication mechanisms among distributed users, tools and services. While we are also involved in the development EDA applications [94], our main goal is to provide the enabling and enhancing[2] mechanisms that leverages most, if not all, systems and toolkits in place[3]. We deliberately engineered our infrastructure in a way such that no restrictions or assumptions are placed on data representation, design methodologies or data and tool usage. Instead, we allow application users and developers to retain their existing methodologies and/or build on top of our infrastructure.

Past frameworks and systems were also tightly-coupled with their particular operating environment; in many recent cases, it has been the UNIX and NFS (Network File System) environment. Such a relationship made it difficult, if not impossible, to extend the frameworks/systems to an Internet environment, which consists of heterogeneous hardware platforms and operating systems. On the other hand, WELD infrastructure is based on platform-independent standards, such as Java on the client side, socket implementation for network

[1] Meta data refers to information regarding the data, e.g. version, user and permission information.

[2] Enhancing mechanisms include manipulation and processing of design information and tool dependency, etc.

connectivity, and open and generic string-based communication protocols. Complete platform independence on both the client[4] and server[5] sides is another feature which distinguishes our environment with other frameworks.

3.3 Motivation

Rapid technological development is taking place in the area of Internet, networking and data processing technologies, such as HTTP [63, 64], Java [65], object-oriented database [47-49] and distributed systems [44]. It is important that any design technology infrastructure take account of, is compatible with, and leverage these technologies. The increase in complexity of the data and process of electronic design, large and distributed teams of engineers, also provides many needs and opportunities [23, 35] for wide-area collaboration in the design of complex electronic systems. These trends enable and necessitate the development of a distributed design system that is composed of cross-platform, network-enabled tools and a platform independent and collaborative user environment.

At the system level, there are many attributes and benefits that pertain to an open distributed design environment [44-46]. The environment and infrastructure should be *scaleable and adaptive*, i.e. services and performance of the system would not be sacrificed with the addition of tools, users and different technologies. The WELD infrastructure consists of components that are flexible and extensible so that users can easily extend existing features (on the client, server

[3] New or legacy tools can be integrated into our environment by placing a server wrapper on top of it.
[4] A Java-enabled web browser, such as Netscape or Internet Explorer, is all that a user needs to access the WELD system and available resources.
[5]Tool encapsulation mechanism is provided by a generic C++ server wrapper.

sides and communication and network service capabilities) and that the system could easily adapt to new, as well as legacy, tools and technologies.

The distributed environment should be *platform independent* and place a minimum requirement on end-users' hardware and software environment and capability. Anyone who has access to a Java-enabled web browser can access the WELD system, regardless of their hardware and software computing environment.

In addition, the system should have the ability to *support a wide range of computing resources.* Server communication technology allows local computing power to be coupled with applications and data processing of network servers. The *Distributed Data Manager* (explained later in detail) enables users to store and retrieve information at a network location. These capabilities allow the WELD environment to support of a wide range of computing platforms from high-end workstations to mobile PDAs.

Users and developers of a distributed (design) environment should be able to *leverage existing software, systems and toolkits.* Java, network protocols and server technology enable the encapsulation of existing tools and software resources regardless of their programming language, operating system base and development environment (such as commercial or academia) and allow them to be integrated into the WELD environment.

A distributed environment should also facilitate *parallel processing*. Parallel processing can be achieved implicitly and at a low cost in a web-based environment (like WELD) since operations can be easily distributed over the network to different servers.

There are also a set of characteristics, at the user-level, that are important to a distributed design environment. The distributed environment should be able to reach and be accessible by a *large user base*. The large number of Internet users both creates needs and provides opportunities for innovation and wide-area collaboration.

Applications and services in a distributed environment should manifest *high availability*. Networked tools in the WELD environment provide users with on-demand access regardless of time of request or geographical location.

Finally, a distributed design environment should provide *consistent and intuitive user interfaces* and allow *flexible tool configuration* to help reduce engineers' start-up and training time. Platform independence of Java enable client applications to provide a consistent user interface across different computing platforms. The *Distributed Tool Flow Manager* (explained later in detail) allows users to custom design and configure workflow of "networked" services to meet application-specific needs.

3.4 System Architecture

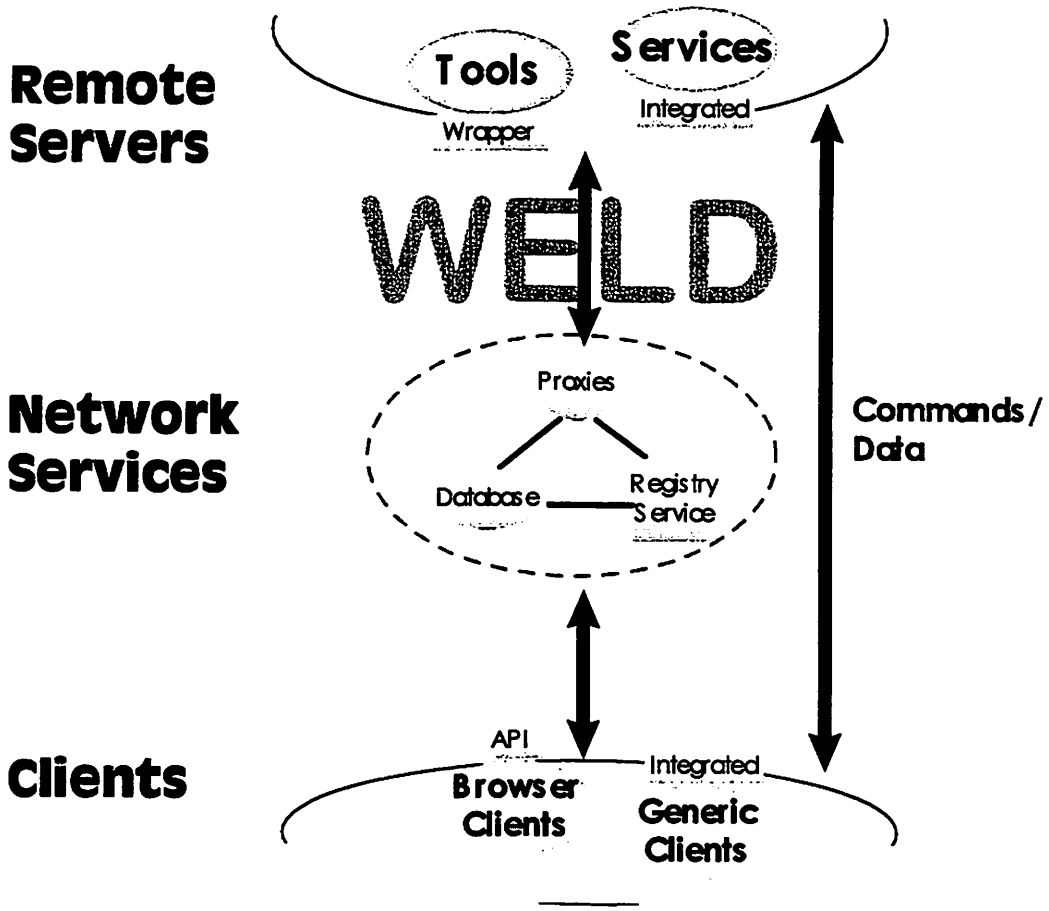

Figure 3-1 Weld System Architecture.

The WELD architecture is a three-tier architecture consisting of, *Clients* – users/programs who access the network resources of the system, *Remote Servers* – tools or services that are made available for network access and *Network Services* – services existing in the network that assist various client/server activities. The mechanisms that enable these network entities to communicate with each other are the *Client-Server Communication Protocol* and *Client-Database Communication Protocol.*

3.4.1 Components Description

This section provides a description and detailed explanation of the functionality of each WELD component.

3.4.1.1 Clients

Clients are (end-user) applications that make use of WELD network infrastructure. They can be *Java Network Clients* – Java client programs ran via a Java-enabled browser, such as Netscape or Internet Explorer, which utilize the capabilities of the *Java-Client Package* to gain access to the system (database and remote servers). They can also be *Stand-Alone Clients* – applications that are developed in Java, C, C++ or PERL, that support network socket operations and the *client protocol*(s) can also connect with tools and services in the system.

3.4.1.2 Remote Servers

Remote servers are tools that can be invoked by WELD clients across the Internet. They can be *Legacy Tool Servers* – legacy (and new) tools wrapped with *server wrappers* can be integrated into the distributed system and executed by network clients. They can also be *Integrated Servers* – servers, developed in Java, C, C++, etc., that have built-in support for socket connections and the *client-server protocol* can be seamlessly integrated into the WELD system. Integrated network capabilities may also enable *Integrated Servers* to play a larger role (beyond simple program execution and data processing) in the system.

3.4.1.3 Network Services

Network services are components that provide features and functionality that are useful to most applications. These include the *Distributed Data Manager*, *Proxies* and *Registry Service.*

3.4.1.3.1 Distributed Data Manager

A database system that manages data residing in the network, as opposed to on client machines or servers.

Functionality of the data manager include:

- Saving and loading of data.
- Query and search.
- Versioning of data/objects/programs [50].
- Incremental update mechanism [50].
- Fine-grained granularity of object/data locking [51].
- Transaction mechanism [52].
- Directory/Registry service.
- HTTP support (may be used also as a web server).

3.4.1.3.2 Proxies

In the WELD architecture, we refer to a *proxy* as a software program residing in the network that acts "intelligently" and transparently as an agent[42] between clients, network servers and other proxies.

Functionality of proxies [37] include:

- When co-located with Web servers, proxies extend the Netscape-Java security model and enable Java applets/clients to communicate with available network resources.
- Caching of (frequently accessed) data.
- Automatic translation between data formats as needed for tools.

- Data "distillation" [38] according to application needs and network links.
- Security/Access control [54-58].
- Dynamic search for servers (in case of failures).
- Queuing, batch processing and scheduling/locking for servers.
- Plug-in architecture that allows the extension of proxy capability to meet application-specific needs.

3.4.1.3.3 **Registry Service**

A table (dynamic storage), which may be co-located with a data server, that maintains information on the availability of network resources.

- Tools and services register with the system to inform the registry of their existence and availability, as well as any restriction that is placed upon access of the resources.
- User applications can query the registry for availability and possible selection of tools.

3.4.1.4 Client-Server Communication Protocol

The client/server communication protocol enables communication between client-server and proxy-server. It was designed to:

- Flexibly handle server parameter type and values.
- Allow for stand-alone, as well as recursive, command structure.
- Be extensible by users and application developers.

3.4.1.5 Client-Database Communication Protocol

The client/database communication protocol enables clients/client objects to communication with a network database system. This protocol:

- Allows client objects to be translated and mirrored in the database.
- Allows the linking of objects in a completely arbitrary manner.
- Enables clients to take advantage of built-in database capabilities, such as save, load, query, versioning, etc.

4. Client Infrastructure and Applications

4.1 Introduction

The success of a distributed environment hinges upon both the quality and quantity of services available as well as the number of users that can access the system. To enable a large user base, Java [65], with its applets and applications being platform-independent, was chosen as the language for the development of infrastructure and client applications for our distributed design environment.

The focus of this research is placed on client infrastructure and applications. These packages and applications cover a wide spectrum of functionality, ranging from end-user applications to a Java object-level management package.

A brief description of the software developed are as followed:

- **Java Client Persistent Object Management Package** – a software package that allows any Java object to be mirrored in and manipulated by a remote data server.
- **Java-based OCT - A CAD Data Manager** – Java-based version of the OCT system [26] that helps manage CAD data model.
- **Web-based Project Management Application** – an application that facilitates managing and collaborating on projects over the Internet.

- **Distributed Tool Flow Manager** – an application that allows clients to custom design and configure the flow of networked servers and data processing.

4.2 Java Client Persistent Object Management Package

4.2.1 Objective

Java has evolved [66, 67] from a Internet programming language for constructing applets for fancy display on home pages to a programming language for robust, full-blown applications. The first wave of applets were client-side applications that were downloaded as complete applications. With the use of Java network and socket infrastructure [68], then came a new class of applications, such as on-line games, chat rooms and whiteboards, that provide limited interactivity and data transaction to users. Those boundaries were once again extended as the technology and need for persistent (Java) objects arose. To meet this demand, we have developed a *Java Client Persistent Object Management Package* that utilizes a data-backend to support persistent objects. The ability to manage objects across the network (WAN or Internet) is of paramount importance especially in applications such as distributed electronic design [34, 35].

4.2.2 Introduction

The Java Client Persistent Object Management Package allows Java (client) objects to be managed and manipulated by a network data server. In addition to object storage and retrieval, it also allows capabilities that the remote data backend provides to be extended to client users.

The Persistent Object package allows fields of objects to be stored in internal data structures. For remote object management, these information are then extracted to compose a string according to a client-server communication protocol which is then sent over to the data server. The data server then translates the messages received and recreates the object at the data backend. (Object processing in terms of connections/attachments can then take place at the backend. These "connection" relationships form the basis of representing object fields, such as vectors.) The client applications can also utilize the capabilities that the data server provides, such as querying and versioning.

The API provided for the Persistent Object package is simple and intuitive so that application developers can easily utilize the features provided. It is also flexible and extendible so that it could be tailored towards application-specific needs. Its scalability and ability to work in a distributed manner make it an attractive means for networked object processing.

4.2.3 Package Description

4.2.3.1 Simple

One of the main goals of building the Persistent Object package was to provide a simple, extendible and flexible tool to empower application developers and users to add value to a networked environment. The overhead for using the package is minimal, and exists mainly in object definition and instantiation. The simple API, which consists of commands such as load, save, connect, etc., allows the complexity of the underlying operations (object persistence,

network implementation, and workings and interactions of the base classes) to be completely abstracted away from the user.

4.2.3.2 Flexible

The current implementation of the package stores and retrieves only the pre-designated fields of the objects. Other fields in the object will be automatically skipped when performing network operations. A similar mechanism has been implemented in the Object Serialization [69] portion of the Java Remote Method Invocation [70] system provided by Sun Microsystems where `static` and `private` fields that are declared transient will not be processed for remote operations.

The package is flexible in handling policies[6] such as caching and security. While the infrastructure is present for implementing these policies (such as a `isDirty` field and a `dbId` field in the objects), no policy is enforced in the current implementation. Application developers are free to use the structures provided to implement the policies that suit their specific needs.

4.2.3.3 Extensible

Due to the object-oriented nature of Java, it is very easy to extend the functionalities of the base package. For instance, developers are free to create additional objects/methods that inherit from `PersistentObject` or to simply override existing methods.

[6] Policies are rules and behaviors that make use of primitive mechanisms of a system, as defined by users or application developers.

The classes are organized in a functionally-modular way so that any component can be changed if certain parameters need to be altered, such as the client-server communication protocol (`PO_CommandConstructor.java`) or the network connection (`PO_NetClient.java`) mechanism.

4.2.3.4 Distributed

The location where objects are to be loaded from and saved to can be changed at the individual-object level (by calling `setServer`, `setPort`). This ability allows parallel and distributing processing across the network.

4.2.3.5 Object Manipulation Mechanisms

The network operations and object manipulation mechanisms supported by the package include:

Command	Description
`save`	Saves a copy of the object to the data server.
`versionSave`	Saves a new version of the object to the server.
`load`	Loads the object with the corresponding ID.
`versionLoad`	Loads the object with the corresponding ID and version.
`deleteObj`	Deletes the object in the data backend (all attached objects will also be automatically deleted).
`connect`	Connects 2 objects in the data backend.
`disconnect`	Disconnects the objects in the data backend (if they are attached).
`initGenContents`	Loads the objects that are attached to the current object.
`initGenContainers`	Loads the objects that the current object is attached to

4.2.3.6 Versioning

The support of versioning is of great importance, especially to applications and processes such as electronic circuit design [1, 2] and software engineering [49]. Versioning capabilities of this package are built on-top of the basic save/load mechanisms. They are enabled by backend mechanisms and a version field (string) in each persistent object on the front end. The client-end versioning capabilities, which includes, save, retrieve, browse, etc., are described below.

4.2.3.6.1 Save

Any (Persistent) object can be saved as a new version using `versionSave`

- Versioning a Point - If it is the first time the object is saved, a `VersionObject` will be created at the data backend with the original object and new object attached to it, otherwise a copy of the object to be saved will be attached to the already existing `VersionObject` in the data backend.
- Versioning a Tree - All and any attached objects, starting from the point of the object to be saved will be recursively saved and attached on the backend server.

4.2.3.6.2 Retrieval

Three types of versioned-object retrieval mechanisms are provided:

1. Retrieving a specific version (e.g. version A.B)
2. Retrieving the latest release (with version flag set as "R")
3. Retrieving the latest version (with version flag set as "L")

4.2.3.6.3 Mechanisms

Although versioning capabilities are built-in, objects that do not require versioning need not deal with the implementation at all. The basic versioning mechanisms provided with the package also allows more sophisticated versioning policies, that applications may need, to be built on top of them.

4.2.4 Extensions in the Package

The core of the Persistent Object package has itself been used to implement some of the more advanced features that were deemed important and useful to general applications.

4.2.4.1 Directory Object

In order to facilitate a directory structure in the OODB server, a special class of `PersistentObject-DirObject` was implemented. In addition to the `PersistentObject` functionality, `DirObjects` also provide special connection-related commands such as `detach` and `getContext`.

A Database Browser (graphical user interface) (see Figure 4-6) has been implemented in Java to allow users of the system to peruse the database and directory structure.

4.2.4.2 Query Object

`QueryObject` was implemented in order to provide a more flexible and effective retrieving mechanism and take advantage of backend intelligence. Currently, the *keyField* (first field) of the object can be used to perform a pattern matching retrieval. Plans have been made to expand the protocol and package to allow:

- Query object field value.
- Query field of object by unique id.
- Query field value by field value.

4.2.4.3 Version Object

`VersionObject` was implemented to represent a version object node in the data backend. It acts as the point where the traversal of versioned objects is performed.

4.2.4.4 Data Object

As a basic extension of the package, `DataObjects` allow users to easily take advantage of a network data server. Any files, data or objects, that can be translated to a string representation can be saved to and retrieved from the data server with the use of this class.

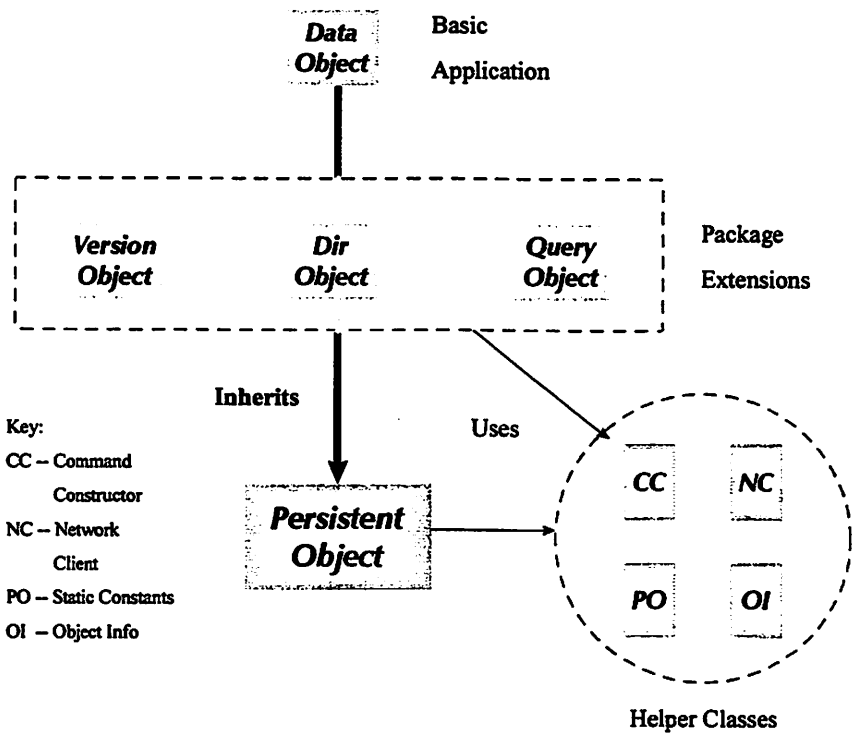

Figure 4-2 Module-level organization of the PersistentObject Package

4.2.5 Future Extensions and Considerations

Although the `PersistentObject` class is functional at the present stage, there are many possible directions of extending its functionality and improving its robustness. Some of the possibilities are described below.

4.2.5.1 Preprocessor of Source Class Files

Currently, use of the `PersistentObject` class requires the object developer to absorb an initial code implementation cost. Such an overhead, though fairly small already, can be further reduced by the development and incorporation of a preprocessor that parses the object files (`.java`) and automatically generate and compile the code for network object management. However, at this stage, this would be a task that provides little value-added to the research of distributed Java object management.

4.2.5.2 Client/Server Load Distribution

With a data management backend, applications can intelligently make use of server processing power by "shifting" some of the computation/analysis to the backend. Load balancing [11] and distribution between a Java application front-end and server backend serves as great future extension of this project or an independent project on its own.

4.2.5.3 Resource Location

While the package allows for retrieval of individual objects from different servers and ports, the mechanisms for locating an object or a service by providing user-specified properties has not been implemented. Even though resource location [53] is an important aspect of a distributed

environment, the discussion and implementation of resource location on top of the persistent object management package is outside the scope of this document.

4.2.5.4 Object Management Policies

The current Persistent Object package has provided many enabling mechanisms in the areas of caching, security and versioning. Application-specific or generic policies can be implemented using the underlying structures and mechanisms.

4.2.5.5 Security

Different methods and measures can be taken to improve the authenticity, integrity and privacy of the data transmitted to and from the backend server.

1. Authorized access [57, 58] of objects on the backend (login/password), which is tightly-linked to session control and efficient multi-user support.
2. Check-length/checksum of protocol message to provide greater security measures for verifying that message received is in fact message sent.
3. Other encryption schemes [55] (e.g. digital signatures) can be added to the transmission and receipt of message streams to ensure privacy of network session.

4.2.6 Overview and Analysis of Java Remote Method Invocation

4.2.6.1 Introduction

The Java Remote Method Invocation (JRMI) system [70] allows distributed Java applications that run on different network hosts to communicate with one another as if only local calls were made. Its main feature and restriction is that the system was specifically designed to operate in the Java environment. It assumes the homogeneous environment of the Java Virtual Machine[7] [62, 71], allowing it to seamlessly follow the Java object model.

Since the JRMI system/package is a part of the Java API [68], it is very convenient for application developers to take advantage of the networking and inter-object communication capabilities. However, it being a complete Java client-server/peer-peer system make it an unattractive/unusable option in a distributed system consisting of applications developed in multiple languages.

4.2.6.2 Technical Description

4.2.6.2.1 Definition

Remote Object - an object whose methods can be invoked from another Java Virtual Machine. This type of object has to be declared by one or more (Java) remote interfaces.

[7] The Java Virtual Machine is the software implementation of a CPU designed to run compiled Java code, including stand-alone Java applications as well as applets that are downloaded and run in Java-enabled Web browsers.

Remote Method Invocation - invoking a method of a remote interface on a remote object.

4.2.6.2.2 Technical Merits and Features

There are a many technical merits in using JRMI as the communication mechanism among distributed Java objects. The reference of a remote object can be passed as an argument or returned as a result in any method invocation, as if only a local operation was performed. The JRMI system takes advantage of and extends the language and security features of Java, such as security managers, class loaders and distributed garbage collection of remote objects. It also transparently handles server replication, multiple object invocation and the loading of a class dynamically if it is not readily available locally. The `java.rmi.Naming` interface provides URL-based methods to lookup, bind, rebind, unbind and list the name and object pairings maintained on a particular host and port which make it convenient to access remote objects.

4.2.6.3 Comparison with Other Object Communication Models

Model	Advantages	Disadvantages
JRMI	• Seamless incorporation of distributed objects and object manipulation.	• Only supports Java-Java environment.
RPC [59, 60]	• System is built-in and supported by the remote host.	• Ability to manage and manipulate objects is very restricted. • Calls are platform dependent.
CORBA [78-80]	• Pre-defined interface that provides platform-independence. • Ability to take advantage of the CORBA resources already in place.	• Requires a language-neutral object model for all parties involved (in order to handle a heterogeneous, multi-language environment).
Java/Network Sockets	• Protocols can be tailored to suit application-specific needs (speed, memory requirement, etc.). • Remote calls are platform independent.	• Requires that the client and server engage in a pre-defined application-level protocol. • Requires the packaging of and decoding/parsing of messages by clients and servers.

4.2.6.4 Comparison with WELD Client/Server Infrastructure

JRMI Characteristic	WELD Infrastructure Characteristic
Possible to access a remote object by reference.	Object referencing is not supported. • However, dynamic referencing is only meaningful in a transient environment. Remote persistent object management has to be done by copying object.
The Object Serialization protocol was designed and implemented to achieve the translation and re-construction of objects. • There are no natural extension to furthering the built-in capabilities.	The communication protocols allows for querying, versioning, as well as translations of objects. • Developers can easily extend the client, server and/or client-server protocol for application-specific needs.
The `writeObject` method serializes the specified object and traverses its references to other objects in the object graph recursively to create a complete serialized representation of the graph. Object Serialization produces just one stream format that encodes and stores the contained objects. • Minimizes number of network	Each object is encoded into a separate message, object relationships are represented by an explicit connection message/command. • Finer object granularity allows for

operations.	flexible data transfer, especially for loading.
Users only need to import the JRMI package, which is available with the Java API, for the objects to utilize the functionality.	Users need to explicitly download the package for usage. • Users may use the database server provided by the WELD group or implement their own data server.
Versioning is done with respect to new/updated object definition.	Versioning is done with respect to updated data/field values of existing objects.

4.2.7 Conclusion

The Java Remote Method Invocation system serves as a useful network package for developing distributed Java-based applications. Such applications could easily take advantage of the built-in networking and communication functionality, such as object serialization, remote object method invocation and referencing, etc. However, its language-dependence makes the JRMI an unsuitable candidate for the communication backbone of a distributed environment, such as WELD, which consists of not only Java-based clients, but also various network entities, such as database manager, proxies, application servers, that are developed in various programming languages.

4.3 Java-Based OCT - A CAD Data Manager

4.3.1 Objective

OCT is a data manager for VLSI/CAD applications [26]. It has been a major component of the Berkeley CAD framework and has been used in many CAD applications and projects [25-33] since the 1980s.

The development of the Java-based OCT package marked WELD group's first attempt to gain experience in evaluating efficient techniques and data structures for representing CAD data formats and models and managing data in a network environment, with the use of an object-oriented database backend.

There were a number of reasons why we chose to port the OCT package as the first-cut network data model. Structurally, the use of generic attachments among `OctObjects` to represent object linkage/relationship translated well into an object-oriented model. While architecturally simple, the OCT package could be used to build and traverse arbitrarily large and complex objects, using simple mechanisms [26] such as `Attach`, `InitGenContents` and `InitGenContainers`. Developed originally at UC Berkeley, OCT is a data model known and understood well within the research (UC Berkeley CAD) group [95]. Modifying some of the OCT-compliant legacy tools to work in a network environment with the Java-based OCT and Java front-ends was one of the considerations for future extensions and experiments. The resulting package would be a good base-object structure for the class assignments and project for

a graduate class in design technology at UC Berkeley. This translates to a solid user base for implementation and usage testing.

4.3.2 Technical Introduction

Because of OCT's architectural elegance and simplicity, Java-based OCT has retained many of original OCT's features, properties and structural representation mechanism. The basic unit in a design is the *cell*. A cell is the portion of a design that the user wishes to consider as a unit. A cell may have many *views*, such as "schematic", "symbolic", "physical", "simulation", etc. Each view has a *facet* named "contents" which contains the actual definition of the view and various application-dependent "interface" facets. Facets, which exist beneath the level of views, contain instances of other cells, which may in turn contain instances of other cells.

Facets consist of a collection of objects that are related by attaching one to another. The basic OCT system allows `OctObjects` such as bags, boxes, terminals, etc. to be arbitrarily attached to each other. The main mechanism of traversing these relationships is by `InitGenContents` (for generating the `contents vector`[8] of objects attached) and `InitGenContainers` (for generating the `containers vector` of objects which the object is attached to).

OCT imposes no restriction on the interpretation of data and few restrictions on the organization

[8] `Vector` is a data structure similar to a linked-list. It is provided as a part of the Java API.

of `OctObjects`. It is left to the application developer to implement policies that assign meanings to the organization and structure of data represented using OCT.

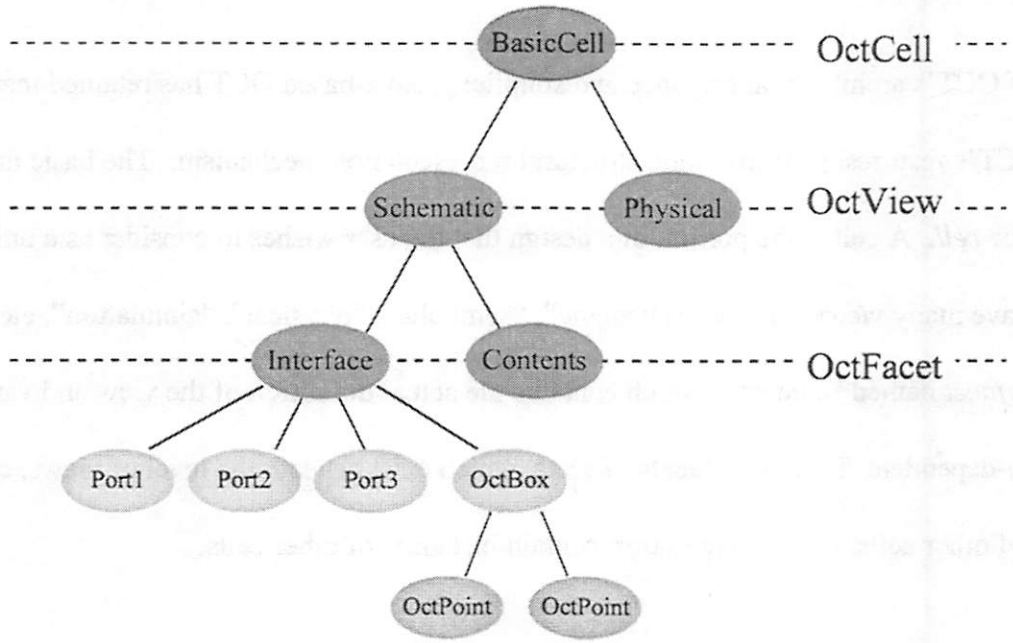

Figure 4-3 OCT Data Structure Example.

The figure shows an example of an OCT structure (that was used as the basic cell of a class assignment (explained later in detail)). The ovals represent `OctObjects` and the links represent `Attachment` relationships among `OctObjects`.

4.3.3 Implementation Description

While we tried to retain as much of the original flavor of OCT as possible, this implementation differs in many ways from the original OCT system. New concepts, structures and organizations have been introduced and, at the same time, unsuitable features have been discarded.

The three main implementational differences between Java-based OCT and the original OCT are:

1. The client/user code is written it in Java, an object-oriented programming language, as opposed to C, which is a procedural language. Object-oriented design and organization can and has been utilized.
2. An object-oriented database is used as a data back-end (for load, save, queries, etc.), whereas the original OCT made use of UNIX files.
3. Since the Java-based version of OCT was designed to be able to work in both a Internet and LAN environment, a lot of emphasis has been placed on protocol efficiency and minimization of (network) transactions for network storage and retrieval.

4.3.4 Functional Description

The (Oct)object manipulation mechanisms are all supported in this Java-based implementation and are embedded in the `OctObject` implementation which all other `OctObjects` inherit. These mechanisms include:

- `Attach`
- `AttachOnce`
- `Delete`
- `DeleteCommit`
- `Detach`
- `DetachCommit`
- `InitGenContainers`
- `InitGenContents`

`DeleteCommit` and `DetachCommit` were added to the list of manipulation mechanisms due to the need to differentiate between local and remote object processing. Objects that exists only dynamically on the client side or operations that need not be committed can make local calls

(`Delete`, `Detach`); operations which require the persistent object/data server to record the updates need to explicitly call the methods (`DeleteCommit`, `DetachCommit`) that invoke remote calls over the network so that the respective operations can be executed by the data server.

4.3.5 Applications

The Java-based OCT package was used in two assignments of the class *Design Technology for Integrated Electronic Systems* (EE 244, UC Berkeley, Fall 96) [96]. The assignments involved the place-and-route and display of a netlist (consisting of 10X10 cells) and the subsequent use of partitioning algorithms to optimize the placement of the cells (see Figure 4-4). The purpose of the assignment, in addition to introducing CAD concepts to the students, was to test and demonstrate the usability and ease of programming of Java by regular students/engineers, as well as the acceptable performance of Java applets/applications in a computing- and user-interface-intensive setting.

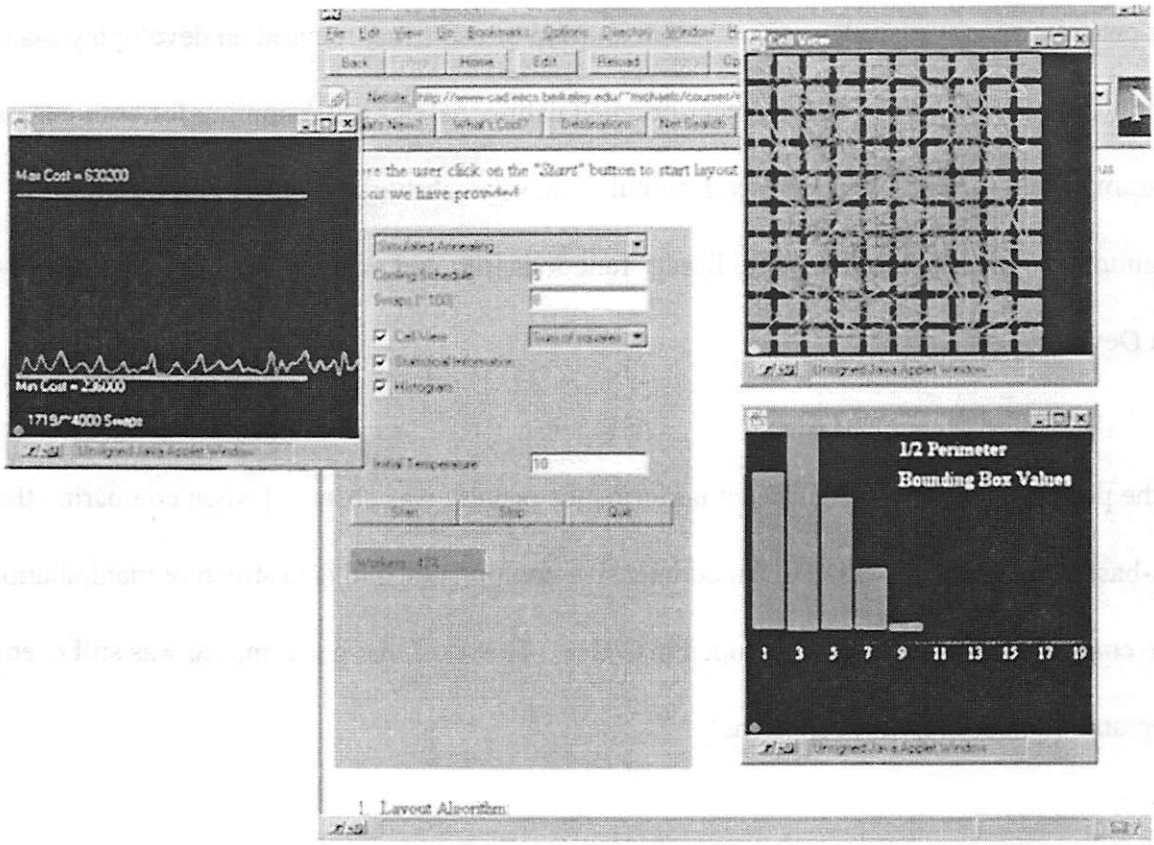

Figure 4-4 Snapshot of one of the homework examples that used Java-based Oct as the base object data structure.

The figure shows a Java applet that allows users to choose multiple partitioning algorithms and visualize the placement of the cells/results. During execution, various costs of the placement are also displayed (by the line graph and bar chart) to the user.

4.3.6 Conclusion

The experience gained from this assignment/experiment provided a number of interesting and encouraging insights with regards to using Java as a programming language for computing-intensive (EDA) applications and general user interface.

Java programming was well-received within a group of users that focused on developing mainly C applications. This is due to the object-oriented nature, ease of programming for both general programming logic and user interface manipulation, good abstraction of I/O and network communication, as well as the useful library functions [68] and documentation provided with the Java Development Kit [72].

On the performance side, a significant performance penalty was observed when comparing the Java-based applications (which included intensive computation and data structure manipulation) with comparable applications developed in C/C++. However, the performance was still deemed acceptable for full-blown applications.

The development of Java-dedicated hardware [73] and compilers [74] promises to provide a great performance boost to Java applets and applications. The performance improvement, together with the inherent advantages of Java as a programming language [65], and it being Internet-compliant (supported by most Web browsers), will make Java an attractive candidate for developing both user interface for existing EDA tools, as well as new applications.

4.4 Web-Based Project Management Application

4.4.1 Introduction

Evolving into more than just a communications medium for static publishing, the Internet is emerging as the platform for wide-area collaboration [20, 35]. However, many challenges and obstacles [44-46] factor in into the viability of such a platform. These include a rich set of user applications, a "secure" transaction and storage data model, efficient communications protocols and supporting infrastructure applications.

The development of *WebProj*, a Java-based project management application [81, 82], contributes to a number of the aforementioned aspects. *WebProj* provides a common working environment and helps manage the flow, collaboration and administration of projects, with an emphasis on electronic design, that are performed by people over the Internet.

4.4.2 Functional Description

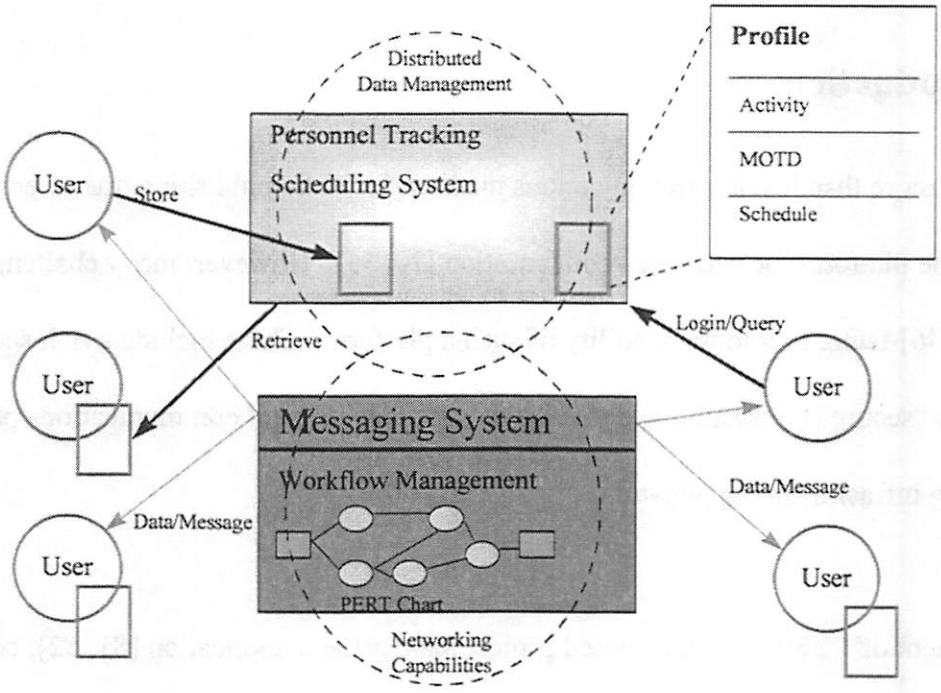

Figure 4-5 High-level functional description of the different components of the WebProj system.

Users can access the WebProj system either as a Java application or as an applet running via a Java-enabled network browser. A registered or new user can log on to the application environment via a Profile Window, which is analogous to an electronic time-sheet or the finger information in a UNIX environment. After filling in the fields of the Profile window, a Password Window will prompt the user for a password. The password is compared with the corresponding Member object in the back-end data server. (If it is a new user, the information will be used to create a new Member object.) After the verification process, the profile is saved and an e-mail is sent to the administrator indicating that a login has occurred. The main window of the application will then appear, allowing the user to create new projects (and edit them in the

form of a PERT Chart [83-85]), update existing projects, etc.. Through a Database Dialog window, the user can access desired additional information (such as members, projects, profiles) as allowed by his/her permissions.

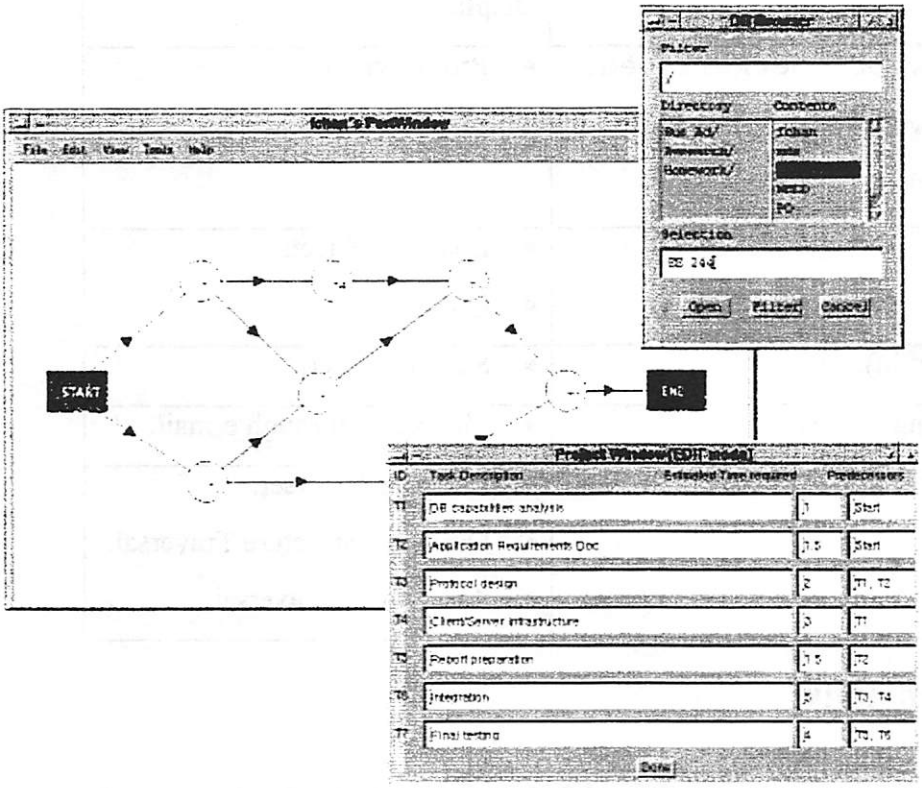

Figure 4-6 Snapshot of WebProj.

The figure shows the main window of WebProj, which has in display a PERT chart that corresponds to the project as indicated by the Project Window. The Database Browser is also shown.

4.4.2.1 Persistent Object Package Usage

In addition to it being a groupware/project management application [89], *WebProj* also served as a driver and user of the features provided by the Persistent Object package.

A table that pinpoints the demonstrated infrastructure capabilities is followed.

Activity	Infrastructure Capability Displayed
Loading from `www-cad.eecs.berkeley.edu` (web server `theseus`) with data server at `yoyodyne.eecs.berkeley.edu`.	• Proxy Server.
Password Verification.	• Loading Objects. • Query.
Saving Profile (during log in).	• Saving Object.
E-mail Notification (during log in).	• Messaging through e-mail.
Opening Project.	• Database Browser. • Directory Structure Traversal. • Attachment Traversal.

4.4.3 Future Extensions

The extension of Java client object versioning capabilities will enable the development of a distributed software configuration management system [86, 87]. This is especially important in managing network files or data, especially those that are frequently updated of have versions scattered across the net in different data servers.

Policies and visualization techniques that allow dependencies among data and members in the environment to be easily traced and managed could be developed to enhance the wide-area collaboration aspect of this application.

Database [61] capabilities such as indexing, storing, table look-up and queries can be used for user profile operations. Inherent database features, such as replication, for distributed data management can also be explored.

Scheduling functionality can be incorporated into personal profiles through integration with other Java packages. These may include calendar and personal organizer functionality as well as meeting proposals.

4.4.4 Summary

The development of *WebProj* provided an excellent opportunity for testing, extending and showcasing the features of the Java Persistent Object Package and Distributed Data Manager. The design and implementation of such a groupware shed light onto the needs of engineers in a distributed collaborative environment, areas of user interface, concurrent engineering and network data access.

It also acted as a springboard to the development of an important piece of infrastructure/application in a Web-based design environment – The Distributed Tool Flow Manager, presented in the following section.

4.5 Distributed Tool Flow Manager

4.5.1 Introduction

One of the visions of a distributed design environment was the ability to pool together as many tools as possible and enable any user to leverage the networked resources. There is a lot of room for innovation and creativity in such an environment as users can choose best-of-breed applications easily and at a low cost[9].

The Distributed Tool Flow Manager is a Java-based application that allows users to flexibly choose network tools, design workflow and configure servers to meet application-specific needs. This capability is of great value, especially to CAD applications and processes [13-16, 20, 24].

At the system level, the Distributed Tool Flow Manager ties together the whole environment by utilizing many pieces of the network infrastructure developed. These include:

- **Java Client Package, Client-Server Communication Protocol, Client-Database Communication Protocol** – Enables any Java/browser client to access the network tools and services.
- **Proxy** – Enables any browser client to access network resources.
- **Registry Service** – Provides network clients with information on the availability of network resources.

[9] Users can access tools without following the traditional purchase-install model.

- **Server Wrapper** – Enables any new or legacy tool to connect to the WELD system by allowing network clients to communicate with and invoke the tool.
- **Data Manger** – Provides network storage for client.

4.5.2 Functional Description

Networked tools that are encapsulated by the Server Wrapper register with the Registry Service to inform potential users of its availability, network location and other information, such as parameter types. When a client invokes the Distributed Tool Flow Manager, a command is sent to the registry to query the tools that are available. The response from the registry (available tools) is then used to dynamically configure the menu bar entries of the tool flow manager. A user can then pick any of the tools (such as data processing servers, translation tools, database entries, etc.) to be part of a flow. The tool will be represented as an object on the tool panel. The user can then "connect" the tools chosen and complete a flow. When the user "executes" the flow, the tool flow manager automatically traverses the graph and configures the tools (according to user input parameters) so that the actual network flow can be carried out and executed by the tool flow manager or a backend workflow server. During the flow, the Tool Flow Manager continuously queries the servers and updates the tool panel so as to allow users to visually track progress, including intermediate results and failure information of the flow.

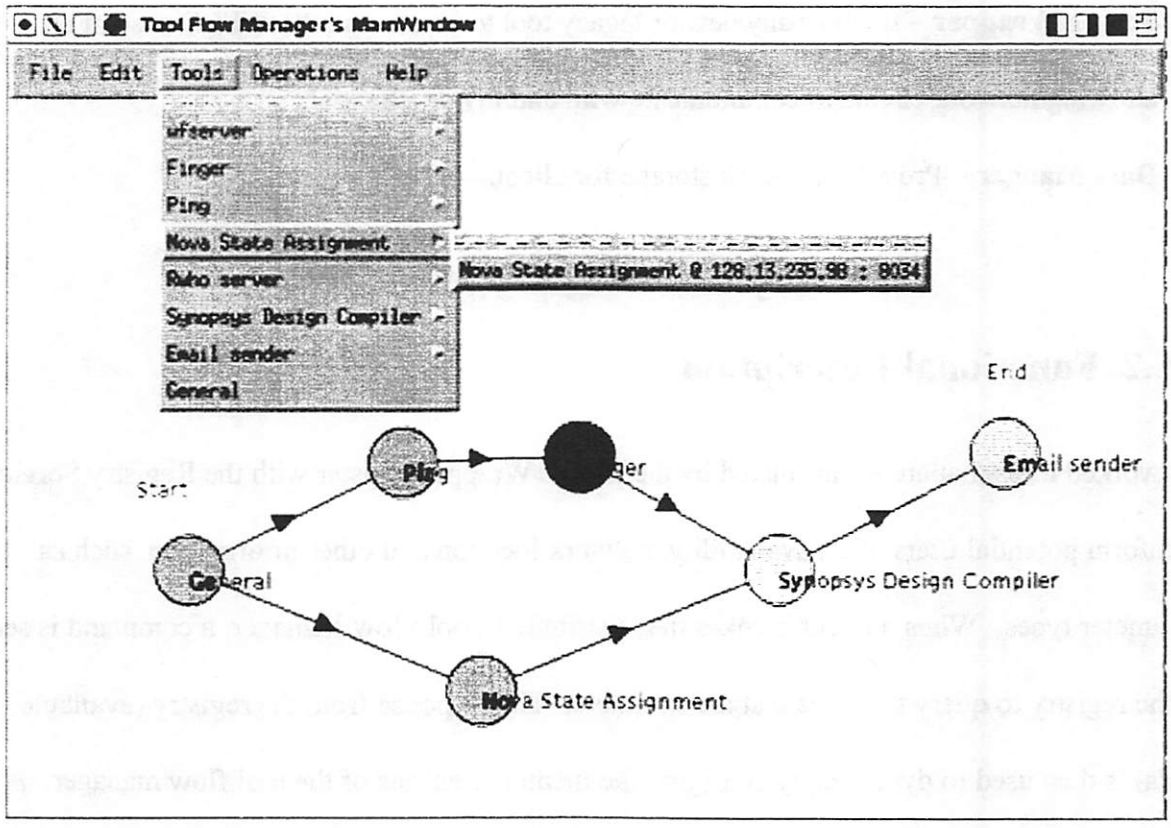

Figure 4-7 Snapshot of the Distributed Tool Flow Manger.

The figure shows a workflow that is constructed with the help of the Distributed Tool Flow Manager. The circles represent the tools chosen and the connections represent data flow between them. The colors of the tools indicate the execution status of the current workflow (e.g. red for failed, green for finished, blue for not yet run).

4.5.3 Conclusion

Many challenges and issues were encountered with the development of the tool flow manager, at both the server[10] and client side. At the client side, challenges included how best to represent and edit a workflow, using different user interface metaphors, e.g. different graphs, colors and shapes. Furthermore, there were also interactivity issues as to how much, how frequent and what kind of feedback about the (execution(s) of) workflow should be relayed back to and what kind of information to seek from the user.

The Distributed Tool Flow Manager will be used to demonstrate a significant CAD workflow at the 1997 Design Automation Conference[94]. Experience from development and deployment, as well as feedback gained from potential users will be used to improve the features of the application.

[10] The focus of implementation related to this report is on the client side. Details of the issues at the systems level can be found at [97].

4.6 Results and Experience

Through the design and development of various infrastructure components and client applications, we have gained much knowledge and experience with regards to the needs, issues and challenges of building a distributed environment.

These experience and insight include:

- Choice and tradeoffs among different types of data servers in a distributed object environment.
- Types of applications that best fit a distributed client-server model.
- Current and expected performance of Java as applets and applications.
- Performance and different modes of networked object operations.

4.6.1 Server Requirements

Currently, an object-oriented database [47-48] acts the data manager of the environment. However, the underlying implementation that makes use of sockets and the transmission of strings make the components flexible enough to interface with any backend proxy or data server, such as relational database or file server.

The table below describes the tradeoffs of using different data backends:

Server Type	Capability/Advantage	Disadvantage
Object-Oriented Database	• Object characteristics and properties can be preserved. • Can provide built-in database capabilities (e.g. query, attribute processing).	• The whole database needs to be recompiled/re-linked after the manual entry of any new schema (object definition), which requires database administrator access.
Relational Database	• Fairly flexible in object addition, deletion and manipulation. • Built-in database capabilities.	• Requires mapping of object from object-oriented structure to a relational representation.
File Server	• Minimal extra software required.	• Complex object-mapping techniques are required. • An (intelligent) transaction manager in the native environment has to be implemented.

While the use of an object-oriented data backend currently suffers from slight deficiencies such as inflexible object additions, it was adopted due to its important ability to seamlessly preserve object properties and thus provide object manipulation.

4.6.2 Applications

The WELD group has been involved in developing various types of network applications:

- Full-blown EDA applications written in Java – SpecChart Editor[75].
- Addition of Java user interface to existing tools – WebSpice [76].
- Web-sizing and integration of legacy (Nova) and commercial tools (Synopsys Design Compiler) with our server technology [77].

A summary of the effort required and application capabilities of the aforementioned development routes is listed below:

Type	Development Efforts	Suitable Applications	Remarks
Developing/porting full-blown Java applications	High	Stand-alone user applications that cannot utilize network processing. User-interface-intensive[11].	Performance hit since Java is still slower than C/C++.
Addition of Java front-end to existing tools	Moderate	Applications that can leverage both flexible user input/response and network server processing.	Suitable for most applications
Integrating with server wrapper	Low	Data processing/computing-intensive applications.	Cannot support UI-intensive applications

[11] UI Programming with Java is easier (to do and learn) than Xtoolkit, but less powerful and harder than Tck/Tk [93].

4.6.3 Performance

The running of Java applets, as opposed to C/C++ programs in a native environment, introduces various performance issues.

Our experience and expectation of their current and future impacts are summarized below:

	Affecting Factors	Experience and Expectation
Loading of Applet	Size of applet Network bandwidth	• Acceptable [75] • Bandwidth is highly dependent on the client and server. Will get better in the future.
Program Execution	Complexity of applet	• Acceptable, but will improve upon the introduction and optimization of Just In Time compilers [74] and Java chips [73] (dedicated hardware).

4.6.3.1 Data Transmission Overhead

Another performance issue is network latency caused by (frequent) network operations. This kind of network overhead is introduced with the setting up of connection, opening and closing sockets. While making use and taking advantage of object-oriented representation of data, we needed to design and implement the policies of how (to what extent) objects would be saved and retrieved.

We made the decision that during save, the complete object, including everything below it would be saved. This mode is the most intuitive and reasonable since users need only to save the

highest level object and can expect the necessary updates to be saved and passed over the network, as opposed to having to explicitly state which particular object(s) to save.

As for loading object, we decided to follow the mechanism of the OCT system – a load only retrieves an object at a single level. The user has to explicitly call `initGenContents` to retrieved the data/objects that are attached to the original object. This policy could reduce the amount of data or objects (at a lower level) that is unnecessarily-transferred.

While observing object-oriented properties, this approach introduced complexity and (object-oriented) features that were not utilized by many legacy tools. Therefore, we implemented `DataObject`, a Java class which provided the abstraction and mechanism that allows users to easily save to and retrieve a string of arbitrary length and representation from a network data server. This mechanism allows the simple file-saving mechanism of many existing tools to be preserved and reduces the integration time of these tools to the WELD environment.

Moreover, during stress-testing and application integration, we realized the network overhead of saving and loading complex objects to be fairly substantial. Therefore, we have looked at bundling the commands that saving and retrieving a complex object involves to reduce the network overhead involved.

4 different approaches to handling network data transmission are described in the 2 tables below:

Data/Object Transfer Mechanism	Performance		Availability	Example
	Single/Small Object Set	Large Set		(--- socket set-up)
Entire Objects	Lowest	Moderate	Low - Whole object structure has to be locked	--- Save Box --- Save Point1 1 1 --- Connect Box Point1 --- Save Point2 2 2 --- Connect Box Point2
Block Files	Low	Highest	Low - Whole file has to be locked	--- Save {Box {Point1(1,1) Point2(2,2) } }
Separate Network Commands	High	Low	High - Object-level locking can be deployed	--- Save Box --- Save Point1 1 1 --- Connect Box Point1 --- Save Point2 2 2 --- Connect Box Point2
Complete Command in 1 String	High	Moderate	High - Object-level locking can be deployed	--- Save Box Save Point1 1 1 Connect Box Point1 Save Point2 2 2 Connect Box Point2

The pros and cons of each type of object/data transfer mechanism is listed below:

Data/Object Transfer Mechanism	**Pros**	**Cons**
Entire Objects	• User receives the whole object with a single load.	• Performance hit due to unnecessarily-transferred data.
Block Files	• High network performance when transferring whole files.	• Does not take advantage of object-oriented properties. • Low Availability. • Clients need an internal representation and a parser for objects.
Separate Network Commands	• Observes object-oriented properties.	• May suffer performance hits when transferring complicated objects.
Complete Command in 1 String	• Gives user highest flexibility.	• Complicated message format. • Complexity introduced at both the client and server end in preparing and parsing commands.

4.6.4 Conclusion

Many tradeoffs exist in the choice of infrastructure (such as different kinds of data servers and network requirements) and application characteristics when building distributed applications and systems. Our experience has shown that there is not always a clear-cut favorable strategy. It is important that the system architects have the right vision and understanding of technology in order to choose an implementation strategy that best meets system and user capabilities and demands. They should also be able to evolve their thinking as system requirements and technologies evolve.

5. Conclusions and Future Directions

This report has described our vision and motivation of building, as well as the high-level architecture of a distributed design environment. Our experience and results in developing, testing and using various applications in this environment have also been documented. While we have taken a substantial step in realizing the goal of delivering a distributed design environment, there still exists many architectural, feature and client challenges that we need to overcome in order for the WELD system to be considered successful.

5.1 Architectural Challenges

Architecturally, it is of paramount importance to design the system and its protocols to be scaleable, flexible and extensible. With the increasing popularity and prevalence of Internet applications and users, it is imperative that the system be scaleable and able to accommodate the addition of tools and users to the system without sacrificing the performance and availability of resources. The architecture and components should be flexible and extensible so that it is easy for clients or external developers, in both the industry and academia, to add-value to the system by connecting legacy tools, incorporating emerging technologies or providing custom features for our environment, etc.

5.2 Feature and Service Provision Challenges

The development and provision of applications at different user levels:

Object-Level Management Package – Java Client Persistent Object Management Package

Data Manager – Java-Based OCT

Engineering Application – Distributed Tool Flow Manager

General-User Application – Web-Based Project Management Application,

to the system have enabled us to realize that these applications not only satisfy specific client needs and improve the richness of the environment, they also provide a means of measuring and evaluating the performance of the infrastructure and (network) environment, which could help reveal potential challenges and problematic areas that needs to be addressed and improved. Furthermore, they also serve as drivers for the research and development of new and useful technical features of the system.

5.2.1 Future Development

In addition to continuously improving the features of the different packages and infrastructure in place, effort could be put into investigating traditional and innovative ways to enhance collaboration, as well as tackle the issues and challenges in building and deploying a distributed environment.

5.2.1.1 Collaboration Over Space

Besides encouraging and making it easy for geographically-dispersed clients to use the environment and application developers and vendors to link in their tools and services, more work can be done in the direction of facilitating the management of flows and dependencies of network tools, data and users through the display of information using various visualization techniques [98].

5.2.1.2 Collaboration Over Time

Design is a highly collaborative and iterative process which often includes rounds of refinements and exploration using different (combinations of) data, techniques and tools in search for the optimal or an acceptable solution. Therefore, features which enhance concurrent engineering and version management [86-88] are of a high value to a design environment.

- **Locking and Transaction Model** – We should investigate ways of data synchronization[43] to achieve:
    - Incremental update and real-time sharing of data.
    - A transaction mechanism that ensures data consistency and facilitates collaboration.
    - Fine-grained locking and access that enable parts of a design or data to be queried and updated.
    - Balance in database availability and performance.
- **Versioning** – More elaborate versioning techniques, such as branching, merging, etc. (techniques used by many software configuration management products [86-88]), could be investigated and implemented to augment the current versioning features.

5.2.1.3 Distributed Environment

While we have overcome many challenges in building a distributed design environment, such as a high-level architecture, remote data and process management, communication mechanisms and tool and user connectivity, there remains a number of system and client areas where we can improve upon.

- **Interactivity** – Work can be done in the area of researching and prototyping mechanisms for delivering and displaying information that are of value to users, as well as finding out situations where seeking user input or feedback may be necessary and/or useful.
- **Reliability** – Ways of improving the fault tolerance, durability, availability of network resources and handling of network errors should also be considered.

5.3 User and Developer Participation Challenges

The expectations and goals of developing an environment is somewhat different from regular research and development, where success is measured by the elegance of solution to a problem, efficiency or a protocol, features and robustness of a piece of software, etc. The success of a system is measured by, in addition to the technical soundness of the environment and components, the number of users and the benefits that are generated from using the environment, in terms of parameters such as improved productivity, higher application profile, better user education about available tools, collaboration, etc.

Through interfacing with potential users and application providers of the system, we have found out that users/clients are generally concerned and skeptical in the areas of security and performance (of Java and the Internet). Therefore, besides making the system scaleable, easy to use and feature-rich, we need to make the system secure and robust, by providing fault tolerance, data replication, encryption, access control capabilities, etc., so as to ensure data consistency and help manage and protect intellectual property, in terms of designs and data. Only will such a secure environment gain the trust and interest of a critical mass of users and developers to make the system truly rich and useful. We also need to continuously look into ways of improving the performance of our infrastructure components through better design and making use of advanced technologies. Last but not least, we need to inform users of the system and promote its ease-of-use and benefits so that more people can use, provide feedback and input on feature extensions for, contribute and add value to the system.

While much of the design and research is either done or under way, a lot of engineering and development is left to improve and extend the various features of the system. Sound and feature-rich infrastructure, outside participation in both using and providing applications, and improvement in networking and Java capabilities will determine the eventual success of the WELD environment.

6. References

[1] D. S. Harrison, A. R. Newton, R. L. Spickelmier and T. J. Barnes, "Electronic CAD Frameworks," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 393-417, Feb. 1990.

[2] P. van der Wolf, *CAD Frameworks Principles and Arcitecture*, Kluwer Publishers, 1994.

[3] J. B. Brockman et al., "The Odyssey CAD Framework," *DATC Newsletter on Design Automation*, Spring 1992.

[4] Network Computing System (NCS) Reference, Apollo Computer Inc., 1987.

[5] B. Johnson, "A distributed computing environment framework: an OSF perspective," Technical Report DEV-DCE-TP6-1, OSF, January 1992.

[6] H. Sarmento and P. R. dos Santos, "Pace - a framework for electronic design automation," *Proceedings of the IFIP WG 10.2 Workshop on Electronic Design Automation Frameworks*, pp. 85-97, Nov 1990.

[7] F. R. Wagner, L. Golendziner, and M. R. Fornari, "A tightly coupled approach to design and data management," *European Design Automation Conference with EURO-VHDL*, pp. 194-199, September 1994.

[8] W. Schettler, S. Heymann, "Towards support for design description languages in EDA frameworks," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 762-767, November 1994.

[9] S. T. Frezza, S. Levitan and P. Chrysanthis, "Requirements-based design evaluation," *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, pp. 76-81, June 1995.

[10] E. W. Johnson and J. B. Brockman. "Incorporating design schedule management into a flow management system," *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, pp. 88-93, June 1995.

[11] J. Schubert, A Kunzmann and W. Rosentiel, "Reduced design time by load distribution with CAD framework and methodology information," *European Design Automation Conference with EURO-VHDL*, pp. 314-319, September 1995.

[12] A. Bredenfeld, "Cooperative concurrency control for design environment," *European Design Automation Conference with EURO-VHDL*, pp. 308-313, September 1995.

[13] S. Kleinfeldt, M. Guiney, J. K. Miller and M. Barnes, "Design methodology management," *Proceedings of the IEEE*, vol. 82, no. 2, pp. 231-250, Feb. 1994.

[14] P. R. Sutton, J. B. Brockman and S. W. Director, "Design management using dynamically defined flows," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 648-653, June 1993.

[15] K. O. ten Bosch, P. Bingley and P. van der Wolf, "Design Flow Management in the NELSIS CAD Framework," *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pp. 711-716, June 1991.

[16] P. van den Hamer and M. A. Treffers, "A data flow based architecture for CAD Frameworks," *Proceedings of the 1990 IEEE International Conference on Computer-Aided Design*, pp. 482-485, Nov. 1990.

[17] J. Daniel and S. W. Director, "An object oriented approach to CAD tool control within a design framework," *Proceedings of the IEEE*, vol.78, no. 2, pp. 1062-1081, February 1990.

[18] N. Filer, M. Brown and Z. Moosa, "Integrating CAD tools into a framework environment using a flexible and adaptable procedural interface," *European Design Automation Conference with EURO-VHDL*, pp. 200-205, September 1994.

[19] A. Hoeven, O. Bosch, R. Leuken, and P. Wolf, "A flexible access control mechanism for CAD frameworks," *European Design Automation Conference with EURO-VHDL*, pp. 188-193, September 1994.

[20] A. Khetawat, H. Lavana and F. Brglez, "Collaborative workflows: a paradigm for distributed benchmarking and design on the Internet," Technical Report 1997-TR@CBL-02, Collaborative Benchmarking Laboratory, North Carolina State University, February 1997.

[21] P. R. Sutton and S. W. Director, "A description language for design process management," *Proceedings of the 33rd ACM/IEEE Design Automation Conference*, pp. 175-180, June 1996.

[22] J. W. Hagerman and S. W. Director, "Improved tool and data selection in task management," *Proceedings of the 33rd ACM/IEEE Design Automation Conference*, pp. 181-184, June 1996.

[23] I. Videira, P. Verissimo and H. Sarmento, "Efficient communication in a design environment," *Proceedings of the 33rd ACM/IEEE Design Automation Conference*, pp. 169-174, June 1996.

[24] A. Casotto, A.R. Newton and A. Sangiovanni-Vincentelli, "Design management based on design trace," *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 136-141, 1990.

[25] A. Casotto and T. Roessel, "Real-World Application of Run-Time Design Tracing," *EDA Integration and Interoperability Conference*, May 1994.

[26] *OCTTOOLS-5.2 Reference Manual*, University of California at Berkeley, 1993.

[27] M. Silva, D. Gedye, R. H. Katz, and A. R. Newton, "Protection and versioning for Oct," *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pp. 264-269, June, 1989.

[28] D. S. Harrison, P. Moore, R. L. Spickelmier and A. R. Newton, "Data management and graphics editing in the Berkeley design environment," *Proceedings of the IEEE ICCAD-86*, pp. 24-27, 1986.

[29] J. M. Rabaey, C. Chu, P. Hoang and M. Potkonjak, "Fast Prototyping of Datapath-Intensive Architectures," *IEEE Design and Test of Computers*, pp. 40-51, June 1991.

[30] J. T. Buck, S. Ha, E. A. Lee and D. G. Messerschmitt, "Ptolemy: A Mixed Paradigm Simulation/Prototyping Platform in C++," *Proceedings of the C++ At Work Conference*, Santa Clara, CA, November, 1991.

[31] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," *Proceedings IEEE International Conference on Computer Design, pp. 328-333*, October 1992.

[32] R. W. Broderson, ed., *Anatomy of a Silicon Compiler*, Kluwer Academic Publishers, Boston, 1992.

[33] The VIS Group, "VIS: A system for verification and synthesis," *Proceedings of the 8th International Conference on Computer Aided Verification*, p428-432, 1996.

[34] M. J. Silva, "Active documentation for VLSI design," Ph.D. dissertation, University of California, Berkeley, 1994.

[35] M. J. Silva, R. H. Katz, "The case for design using the World Wide Web," *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, pp. 579-585, June 1995.

[36] M. Weiser, "Some computer science issues in ubiquitous computing," *Communications of the ACM*, vol. 24, no. 7, pp. 412-418, July 1981.

[37] A. Fox, S. D. Gribble, E. A. Brewer and Elan Amir, "Adapting to network and client variation via on-demand dynamic transcoding," *Proceedings of the ASPLOS-VII*, Boston, October 1996.

[38] A. Fox and E. A. Brewer, "Reducing WWW latency and bandwidth requirements by real-time distillation," *Proceedings of the Fifth International World Wide Web Conference*, Paris, France, 1996.

[39] T. P. Ng, "Optimal data migration policies in distributed systems", Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign, 1991.

[40] T. von Eicken, D. E. Culler, S. C. Goldstein and K. E. Schauser, "Active messages: a mechanism for integrated communication and computation," *Proceedings of the 19th Annual Symposium on Computer Architecture*, pp. 256-266, May 1992.

[41] F. Douglis and J. Ousterhout, "Transparent process migration: design alternatives and the Sprite implementation," *Software Practice and Experience*, Vol. 21, No. 7, pp. 157-183, August 1991.

[42] D. B. Lange and D. T. Chang, "Programming Mobile Agents in Java," A White Paper Draft, IBM Corporation, September 1996.

[43] M. Stonebraker, P. Aoki, R. Devine, W. Litwin and M. Olson, "Mariposa: a new architecture for distributed data," *Proceedings of the 10th International Conference on Data Engineering*, pp. 54-65, February 1994.

[44] R. Bagrodia, W. W. Chu, L. Kleinrock, and G. Popek, "Vision, issues, and architecture for nomadic computing," *IEEE Personal Communications*, December 1995.

[45] C. A. R. Hoare, The Emperor's Old Clothes, Communications of the ACM, Vol. 24, No. 2, February 1981, pp. 75-83

[46] B. W. Lampson, "Hints for computer system design," *Proceedings of the 9th Symposium on Operating Systems Principles*, pp. 33-48, October 1983.

[47] Objectivity Inc., Objectivity/DB Technical Review, 1995.

[48] Objectivity Inc., Objectivity/DB Programmer's Reference, 1995.

[49] L. C. Liu and E. Horowitz, "Object database support for a software project management environment," *Proceedings of the 3rd ACM Symposium on Software Development Environments*, pp. 85-96, 1988

[50] H. T. Chou and W. Kim, "Versions and change notification in an object-oriented database system," *Proceedings of the 25th ACM/IEEE Design Automation Conference*, pages 275-281, Anaheim, June 1988.

[51] J. N. Gray, R. A. Lorie, G. R. Putzolu and I. L. Traiger, "Granularity of locks and degrees of consistency in a shared data base," *IFIP Working Conference on Modelling of Data Base Management Systems*, pp. 1-29, 1976.

[52] C. Mohan, B. Lindsay,and R. Obermarck, "Transaction management in the R* distributed database management system," *ACM Transactions on Database Systems*, vol. 11, no. 4, December 1986.

[53] D. C. Oppen and Y.K. Dalal, "The Clearinghouse: A decentralized agent for locating named objects in a distributed environment," *ACM Transactions on Office Information Systems*, July 1983.

[54] J. H. Saltzer, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp.1278-1308, September 1975.

[55] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, Vol. 21, No. 12, pp. 993-999, December 1978.

[56] D. E. Denning and P. J. Denning, "Data security," *Computing Surveys*, vol. 11, no. 3, pp. 227-249, September 1979.

[57] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: theory and practice," *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, October 1991.

[58] D. Brown, "Techniques for privacy and authentication in personal communication systems," *IEEE Personal Communications*, August 1995.

[59] Sun Microsystems, "RPC: remote procedure call specification," *Internet Network Working Group Request for Comments*, NIC, 1988.

[60] B. N. Bershad, T. E. Anderson, E. D. Lazowska, and H. M. Levy, "Lightweight remote procedure call," *ACM Transactions on Computer Systems*, vol. 8, no. 1, pp. 37-55, February 1990.

[61] C. J. Date, *An Introduction to Database Systems*, Fourth Edition, volume I. Addison-Wesley Systems Programming Series, 1986.

[62] R. P. Goldberg, "Survey of virtual machine research," *IEEE Computer*, vol. 7, no. 6, pp. 34-45, June 1974.

[63] R. Fielding, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.0," Network Working Group, May 1996.

[64] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," Network Working Group, January 1997.

[65] J. Gosling, B. Joy and G. Steele, "The Java Language Specification," http://www.javasoft.com:80/docs/language_specification/index.html.

[66] Gamelan home page, http://www.gamelan.com.

[67] Java Applet Rating Service home page, http://www.jars.com.

[68] J. Gosling and F. Yellin, The Java Team, Java API Documentation Version 1.02, http://www.javasoft.com:80/products/jdk/1.0.2/api/.

[69] Sun Microsystems, Inc., "Object Serialization Specification," http://www.javasoft.com/products/jdk/1.1/docs/guide/serialization/spec/serialTOC.doc.html.

[70] Sun Microsystems, Inc., "Remote Method Invocation Specification," http://www.javasoft.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html.

[71] T. Lindholm and F. Yellin, *The Java Virtual Machine Specification*, Addison-Wesley Publishing Company, Inc., 1997.

[72] Sun Microsystems, Inc., The Java Developers Kit Version 1.02, http://java.sun.com/products/jdk/1.0.2/index.html.

[73] T. R. Halfhill, "Java Chips Boost Applet Speed," *BYTE Magazine*, April 1996.

[74] Symantec Corporation, Just In Time Compiler Performance Analysis, http://www.symantec.com/jit/jit_pa.html.

[75] S. Leung, SpecChart Editor, http://www-cad.eecs.berkeley.edu/~wleung/specchart.

[76] J. C. Chen, WebSpice, http://radon.eecs.berkeley.edu/~jamesc/classes/ee244/project/project.html.

[77] WELD Group DAC 96 Demo, http://www-cad.eecs.berkeley.edu/Respep/Research/weld/dac96/demo.html.

[78] Z. Yang and K. Duddy, "Distributed object computing with CORBA," DSTC Technical Report 23, CRC for Distributed Systems Technology Level 7 Gehrmann Laboratories, June 1995.

[79] The Common Object Request Broker: Architecture and Specification, OMG and X/Open, December 1991.

[80] Object Management Group home page, http://www.omg.org/.

[81] F. P. Brooks, Jr., *The Mythical Man-Month*, 1995 Edition, Addison-Wesley Publishing Company, 1995.

[82] Microsoft Corporation, *Microsoft Project User's Reference*, Microsoft Corporation, 1992.

[83] A. W. Shogan, *Management Science*, Prentice-Hall, 1988.

[84] P. Edwards, *Systems Analysis & Design*, McGraw-Hill, 1993.

[85] Q. W. Fleming, J. W. Bronn, and G. C. Humphreys, *Product and Production Scheduling*, Probus Publishing Company, 1987.

[86] ClearCase Product Information, http://www.pureatria.com/products/clearcase/index.html.

[87] Continuus Software Corporation home page, http://www.continuus.com/.

[88] M. Cagan, "Change Management for Software Development," Continuus Software Corporation, 1995.

[89] Lotus Development Corporation, "Lotus Notes: An Overview", http://www.lotus.com/notesr4/over2d.htm.

[90] B. Reinwald, C. Mohan, "Structured Workflow Management with Lotus Notes Release 4," IEEE Computer Society International Conference, February 1996.

[91] C. Mohan, G. Alonso, R, Guenthoer and M. Kamath, "Exotica: A Research Perspective on Workflow Management Systems," *Data Engineering Bulletin (Special Issue on Infrastructure for Business Process Management)*, vol. 18, no. 1, pp. 19-26, March 1995.

[92] Workflow Management Coalition home page, http://www.aiai.ed.ac.uk:80/project/wfmc/.

[93] M. D. Spiller, personal communication, March 1997.

[94] WELD Group home page, http://www-cad.eecs.berkeley.edu/weld.

[95] UC Berkeley CAD Group home page, http://www-cad.eecs.berkeley.edu/.

[96] A. R. Newton and J. Rabaey, EE 244 Fall 1996 home page, http://www-cad.eecs.berkeley.edu/~newton/courses/.

[97] M. D. Spiller, Research in Systems and Server Technology for a Distributed Design Environment, http://www-cad.eecs.berkeley.edu/~mds/research.

[98] M. Shilman, Research in Interactive Visualization for CAD, http://www-cad.eecs.berkeley.edu/~michaels/research.

[99] F. Chan, WELD Group Research Materials, http://www-cad.EECS.Berkeley.EDU/~fchan/research/index.html.

7. Appendices

7.1 WELD Client-Server Communication Protocol

7.2 WELD Client-Database Communication Protocol

WELD Client-Server Communication Protocol

Interface Syntax

(Each line is terminated by "\n\r", including one additional line after the last line given).

Called when a tool informs of the registry/data server of its availability.

- Resource Register

  (Resource) Client:

  REG
  <DB Identifier String>
  <ResourceName>
  <ResourceMachineName>
  <ResourcePortNumber>

  Server:

  <XXX> (Three digit status code)

Called when a tool informs of the registry/data server that it is no longer available.

- Resource De-Register

  (Resource) Client:

  DEREG
  <DB Identifier String> (Is this necessary/reasonable???)
  <ResourceName>
  <ResourceMachineName>
  <ResourcePortNumber>

  Server:

  <XXX> (Three digit status code)

Assumption: Tools will not stay down for an extended period of time without de-registering.

Tools can re-register, e.g. after a crash.

The triplet (name, server, port) "should" be unique.

Called when a client starts a tool flow manager and queries about the availability of network resources (for

dynamic UI (menu bar) construction) from the registry/data server.

- Query Registry

  Client:

  QUERYREG
  <DB Identifier String>

  Server:

  <XXX> (Three digit status code)
  <NumContents>
  <ResourceName> (the triplet of each available server will be listed)
  <ResourceServerName>
  <ResourcePortNumber>
  ...

(*Rationale: The tool/service name, server and port are the only essential information needed for a client to make a connection.)

Called when a client wants to execute a program at a "networked" server.

Execute:

Client:

EXECUTE
<DB Identifier String>
<ToolName>
<ParameterString>
<Data>

Server:

<XXX> (Three digit status code)
<Results/DataString>

The first-cut implementation will be to have the clients send and receive the data that is to be transmitted. However, we will investigate different mechanisms where the data field can be replaced by a "pointer" to the data location, such as DataObject in the database, URL, etc. (for efficient data transfer).

Francis Chan, Mark D. Spiller

Last modified: April 3, 1997

WELD Client-Database Communication Protocol

Interface Syntax

(Each line is terminated by "\n\r", including one additional line after the last line given).

Put:

Client:

PUT
<DB Identifier String>
<ClassName>
<ClassSpecificArguements * #ClassSpecificArguements> (each on its own line)

Server:

<XXX> (Three digit status code)
<UniqueObjectInteger (=0 if status failed)>

Modify:

Client:

MODIFY
<DB Identifier String>
<ClassName>
<UniqueObjectInteger>
<ClassSpecificArguements * #ClassSpecificArguements> (each on its own line)

Server:

<XXX> (Three digit status code)

Get by String: (Only Cells?)

Client:

GETSTR
<DB Identifier String>
<ClassName>
<Location Identifier String>

Server:

<XXX> (Three digit status code)
<ClassName>
<ClassSpecificArguements * #ClassSpecificArguements> (each on its own line)
<UniqueObjectInteger>
<Int String # of Content Objects >

<Int String # of Container Objects >

Get by Id:

Client:

GETID
<DB Identifier String>
<ClassName>
<IDString> (Integer String)

Server:

<XXX> (Three digit status code)
<ClassName>
<ClassSpecificArguements * #ClassSpecificArguements> (each on its own line)
<UniqueObjectInteger>
<Int String # of Content Objects >
<Int String # of Container Objects >

Get by Attachment:

Client:

GETATT
<DB Identifier String>
<ClassName>
<UniqueObjectInteger>
<CONTAINER || CONTENTS>
<Int String of Attach # in list>

Server:

<XXX> (Three digit status code)
<ClassName>
<ClassSpecificArguements * #ClassSpecificArguements> (each on its own line)
<UniqueObjectInteger>
<Int String # of Content Objects >
<Int String # of Container Objects >

Get Attachment Information:

Client:

GETATTINFO
<DB Identifier String>
<ClassName>
<UniqueObjectInteger>
<CONTAINER || CONTENTS>
<Int String of Attach # in list>

Server:

<XXX> (Three digit status code)
<ClassName>
<UniqueObjectInteger>

Delete by Id:

Client:

DELETE
<DB Identifier String>
<ClassName>
<IDString> (Integer String)

Server:

<XXX> (Three digit status code)

*Tentatively: All children are deleted?

Connect Two Objects by Id: (A -> B, A contains B)

Client:

CONNECT
<DB Identifier String>
<ID For A> (Integer String)
<ID For B> (Integer String)

Server:

<XXX> (Three digit status code)

Connect Two Objects, parent by Id child by info: (A -> B, A contains B, B being created)

Client:

CONNECTNEW
<DB Identifier String>
<ID For A> (Integer String)
<ClassName for B>
<ClassSpecificArguements * #ClassSpecificArguements> (each on its own line)

Server:

<XXX> (Three digit status code)
<UniqueObjectInteger (for B)>

Disconnect Two Objects by Id: (A -> B, A contains B)

Client:

DISCONNECT
<DB Identifier String>

<ID For A> (Integer String)
<ID For B> (Integer String)

Server:

<XXX> (Three digit status code)

GetRootDir

Client:

GETROOTDIR
<DB Identifier String>

Server:

<XXX> (Three digit status code)
<ClassName>
<ClassSpecificArguements * #ClassSpecificArguements> (each on its own line)
<UniqueObjectInteger>
<Int String # of Content Objects >
<Int String of "0" >

GetContext

(returns first field of each content)

Client:

GETCONTEXT
<DB Identifier String>
<ClassName>
<IDString> (Integer String)

Server:

<XXX> (Three digit status code)
<ClassName>
<NumContents>
<ContentClassName> (set of information transmitted for each entity)
<Each Contents' first identifier> (each on its own line)
<Each Contents' UniqueObjectInteger> (each on its own line)

Versioning

Version strings are of the form A.B, where A&B are 32 bit unsigned integers.

Get Version (by Id):

Client:

GETVERID
<DB Identifier String>
<ClassName>
<IDString> (Integer String, ID of version object)
<VersionsString> (form A.B, "R", or "L")

Server:

<XXX> (Three digit status code)
<ClassName>
<ClassSpecificArguements * #ClassSpecificArguements> (each on its own line)
<UniqueObjectInteger>
<Int String # of Content Objects > (not valid/used)
<Int String # of Container Objects > (not valid/used)

Version Put:

Client:

PUTVER
<DB Identifier String>
<ClassName>
<IDString> (Integer String)
<Version Flag> ("P" for point (this single object) or "T" for tree rooted at this point. "R" appended to create a new release (incr A))
<ClassSpecificArguements * #ClassSpecificArguements> (each on its own line)

Server:

<XXX> (Three digit status code)
<UniqueObjectInteger (=0 if status failed)>
(if Flag is P - returns id of VO, if Flag is T returns the id of the top of the copy tree (now under a VO)).

Version GetContext

(returns first field of each content)

Client:

VERGETCONTEXT
<DB Identifier String>
<ClassName>
<IDString> (Integer String)

Server:

<XXX> (Three digit status code)
<ClassName>
<NumContents>
<ContentClassName> (set of information transmitted for each entity)
<Each Contents' version string> (each on its own line)
<Each Contents' UniqueObjectInteger> (each on its own line)

QUERY

(only a simple start, much more needs to be done here...)

Client:

QUERY
<DB Identifier String> (= "global" for global search)
<ClassName>
<Query Scope> ("G" for global, "C" for class, "D" for container/db)
<Search Index Type ("F" for fieldname, "I" for integer)
<Field Name/Index Number>
<IDString> (search String)

Server:

<XXX> (Three digit status code)
<NumContents>
<Each Found Object's Class Type> (set of information transmitted for each entity)
<Each Found Object's UniqueObjectInteger> (each on its own line)

Generators:

Generators are tracked by the client, so that the server can stay as stateless as possible (in terms of connections).

Details

- Multiple attachments between the same object are not allowed.
- The power of prevention is in the hands of the application developer. The database will allow you to make an object its own parent/child, as well as allow (potentially fatal) cycles (for now).
- Connections (attachments) must be unique - multiple connections are currently undefined.

Francis Chan, Mark D. Spiller

Last modified: Wed Feb 12 10:43:46 PST