Contents

1 Introduction

The adoption of new technology and computing infrastructure in the Electronic Design Automation (EDA) industry has played a key role in determining overall chip design methodology, and thus has played a major role in establishing the resulting tool architectures and algorithms used to implement design systems. Advances in areas such as software methodology and environments, operating systems, storage systems, and programming languages have often had an enormous effect on EDA. The explosive growth in the development of wide-area network infrastructure over the past few years indicates an opportunity for the industry and the field of computer science in general to make a leap to a new generation of capabilities. Specifically, we envision the entire EDA community organized as a single, integrated, distributed environment that offers the user the ability to create a continually evolvable and adaptable "virtual" design system that can couple tools, libraries, design, and validation services. Beyond that, the system could also provide manufacturing, consulting, component acquisition, and product distribution services, encompassing the developments of companies, universities, and individuals throughout the world. Users world-wide would be able to collaborate on complex design tasks in a completely customizable network environment. Example usage could also include companies seeking to utilize a processing tool that is too expensive or updated too frequently to have installed on-site, a processor company that wants to allow users to run simulations of its next-generation chip but does not want to release the actual software for security and espionage reasons, or a company with large multi-site data sets that would prefer the dynamic movement of modular tools instead of data relocation. This report describes network (server) infrastructure developed for such a distributed design system.

2 The WELD Architecture – Motivation and Description

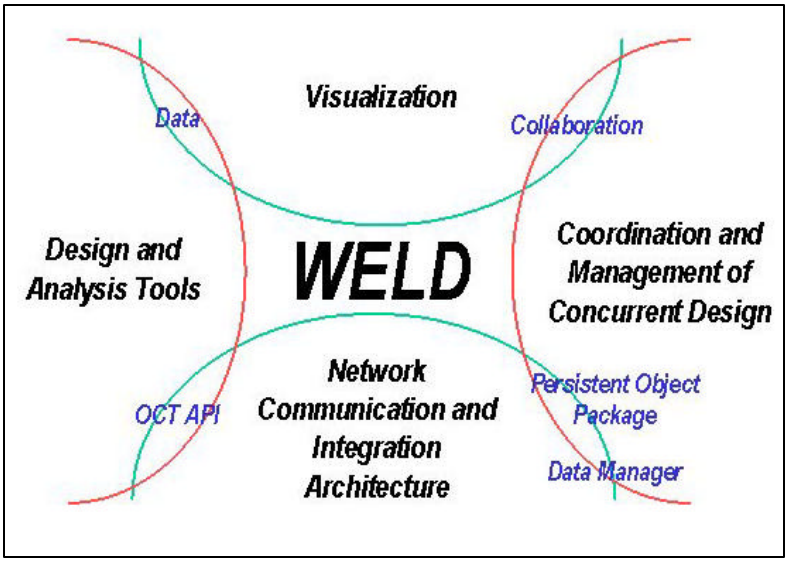

Figure 1: The WELD Project

The goal of the WELD project [WEL96] is to pull together emerging technologies, such as visualization and alternative interfaces, network communications, and Bayesian search, to provide a next-generation CAD design system, as shown in Figure 1. Such a system will need to be scalable, flexible, secure, and cost effective, in order to handle the working sets of modern VLSI designs, which are approaching two gigabytes of data today and will be considerably larger in the future. The growth of the working sets, in conjunction with advances in semiconductor technology, is forcing an increase in design team size as well [SEM97]. One can envision that as design teams grow and partitioning methods improve, designers will be distributed across multiple sites, and will need a collaborative, distributed design system to be able to work together.

2.1 The Need for WELD

Many of the basic ideas behind network computing were introduced in the Multics project [CV65], but were hampered by the lack of high-speed network infrastructure. Even today, networks rarely guarantee any degree of quality of service, often ranging widely in response time, availability, connectivity, and load. The World Wide Web [WWW97] can be viewed as a "bad multiprocessor", which has all of the possibilities for parallelization of a normal multiprocessor, but a great deal more uncertainty in the time, performance, and consistency guarantees that are offered [NEW96]. The inherent variability of this situation requires that tools seeking to utilize it

must themselves be very flexible. Key characteristics that must be considered in designing systems for such an environment include:

- **Scalability**: Network services must be able to incrementally support exponential growth in both the number of users and the number of services. Additional consideration will need to be given to the ways of distinguishing and finding services, data interoperability, and wizard-like graphical interfaces to link services together coherently. Finally, scalable wide-area collaboration will become a necessity as virtual project teams become more popular.
- **Accessibility**: The rise in mobile computing will require ubiquitous access, driving a need for applications that are either platform independent or dynamically adaptable, as well as consistent, intuitive user interfaces (such as the internet browser). Services and applications must be available on-demand and be able to adapt to ranges in resources such as CPU, memory, bandwidth, and storage, perhaps using technologies such as data migration, agents, and proxies.
- **Availability & Robustness**: Before users will trust a distributed design system, they must be confident that access will be uninterrupted, service degradation will be graceful, recoverable, and infrequent, and that the information travelling through the network will be secure and private. In addition, suitable means of managing and protecting intellectual property must be provided, and there must be at least some form of guarantees for time-critical applications.
- **Cost effectiveness**: All of the above attributes must be provided in a cost-effective manner that allows for the exploitation of economies of scale in the network. In addition, there must be ways to leverage existing software, systems, and toolkits, to save the cost of re-implementing large quantities of software.

These above goals are, admittedly, very ambitious. The WELD project attempts to take an initial step in these directions, with a focus on prototyping and developing useful and flexible infrastructure leading towards the realizations of these goals. Although existing technology has been leveraged where possible, at times it has been necessary to build infrastructure in areas where there was insufficient existing technology, including user interface toolkits, protocols, integration interfaces, data management back-ends, and transaction models. This report begins with a high-level view of both the client and server network infrastructure provided, and then goes into detail on servers and proxies. It should be noted that, although this system is being demonstrated and prototyped using CAD applications, the architecture and infrastructure is general and thus applicable to other applications as well.

2.1.1 ACID/BASE

As mentioned above, the "bad multiprocessor" view of the Web requires a different set of assumptions than more traditional applications. While the traditional transactional database world is primarily concerned with the ACID properties (**A**tomicity, **C**onsistency, **I**solation, and **D**urability)[GRA81], the strong semantics provided by this model come at a high cost and implementation complexity. The ACID model does not make any guarantees regarding availability and would rather have a service be unavailable than relax the semantics [FGB+97].

It has been argued that at times it is more important to maintain high availability and low latency access to information than to ensure strong consistency or durability. Quickly delivered approximate answers, based on stale or incomplete data, may be more valuable than the correct answer, which may require a great deal more data or be delivered in an unsuitable timeframe. A new set of semantics referred to as BASE [FGB+97] (**B**asically **A**vailable, **S**oft state, **E**ventual consistency) has been proposed. The BASE semantics are very applicable when one views the Web as a "bad multiprocessor", where remote access latency and bandwidth can vary by orders of magnitude in comparison to a traditional computer system.

Any system attempting to build structure upon the potentially unreliable and unpredictable Internet will require a mix of both the ACID and BASE properties. Currently, the majority of Web applications tend to fall in the camp of BASE, but a large-scale EDA system would require some of the ACID guarantees as well. Part of the design challenge facing the WELD group is finding a general architecture that features properties of both and addresses as broad a range of EDA requirements as possible.

2.2 System Architecture

Part of the trade-off between ACID and BASE discussed above involves the balancing of services between the desktop (including both locally installed tools and Java applets loaded over the network) and jobs sent via the network to the outside world for processing or for parallelization[1]. The WELD group has at this point prototyped several applications testing the feasibility and performance of several different approaches. This section gives a high-level overview of the client, server, and proxy (middleware) infrastructure that has been built.

[1] It should be noted that although there is a good case for distributed tools on the Web, there are designers who would be happy with a good web interface to a system that would allow them to parallelize their applications to speed up processes that require multiple iterations [PER97]. Although this is possible right now using scripts, it is far from convenient. The WELD system should be able to accommodate this as well.

2.2.1 Client Infrastructure

Clients are user programs run on the local desktop or mobile client, either as standalone applications or as a front-end into the network design system. For instance, the SpecCharts Editor [LEU97] is primarily standalone, using the network only for file save and load, while the Project Management application [CHA97] depends upon the network for object structure and organization. While Java applications run through network browsers (such as Netscape Navigator [NET97_2]) are used extensively for WELD clients, any programming language that supports sockets, such as Java, C, C++, or PERL, can be used to build client applications that can connect to the network back-ends using the generic Client-Server, Client-Database, and Workflow Communication Protocols (see Appendices). Since the WELD client infrastructure, including the Java client extensions, sample applications, and detailed information on object usage, is covered in detail elsewhere [CHA97, LEU97], it will not be discussed here.

2.2.2 Server Infrastructure

Servers in the WELD system are components found in the network that provide a variety of useful functionality, including data management, information retrieval, registration services, and workflow services that invoke tools remotely. Server infrastructure built for the WELD project is covered in detail in Section 4.

2.2.3 Proxies

Proxies are intelligent network applications that reside between the clients and servers, and have the potential to offer additional services that may not be practical, reasonable, or desirable at the endpoints. We began our work on proxies while looking for a simple way to bypass the limitations of the Netscape Java security model, which restricts applets to only be allowed to open socket connections to the machine from which they were downloaded. While this is a reasonable restriction that is intended to prevent applets running behind a firewall from making arbitrary connections to the outside world and potentially sending out unauthorized information, it severely restricts the client-server possibilities of Java applets. This security model severely restricts the distributive properties, and thus the scalability, of a network design system based on Java, since it makes it impossible for a single socket server to be accessible to applets loaded from different

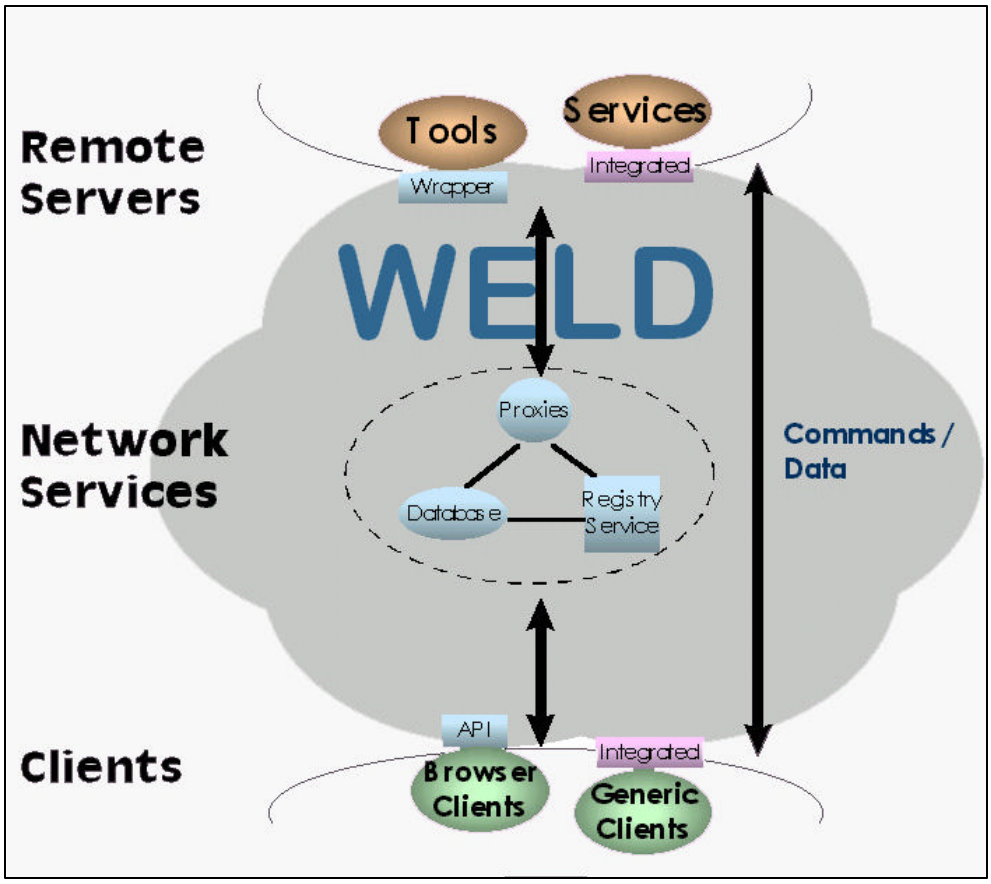

Figure 2: The WELD System Architecture

sources[2]. A short-term prototyped solution used a proxy run on the same machine as the HTTP server to detect and forward messages to other machines[3]. While this is a security violation, it made it possible to prototype a more realistic configuration that may be possible in future security models.

While working on this proxy, a wide range of other possibilities for proxies between the client and server became apparent, including the automatic translation of data into different formats, the filtering and redirection of data flow between applications, the use of centralized proxy servers to track and monitor network tool use, remote system administration, collaboration, caching, and security. Proxy-based system architectures offer significant advantages to a distributed design system, and will be heavily leveraged in the future of the WELD Project. Additional information on proxies and proxy architectures are presented in both the Related Work

[2] One result of this security model was that due to the platform/licensing dependencies of some of the server software (such as the Objectivity OODBMS and the Synopsys design compiler, both of which were licensed and only able to run on one machine), the heaviest loaded server had to also carry the additional weight of being a HTTP server if applets were to access the database or the design compiler!

[3] Other systems have been developed recently, such as the Agent Router found in the Java Agent Template [JAT97], that provide similar functionality.

and Future Directions sections below. The workflow server described in Section 4.3 could also be viewed as a highly intelligent proxy.

3 Related Work

In the two years since the beginning of the WELD project, many other research and industrial organizations have also worked in the areas of objects, proxies, and workflow. Indeed, we are now collaborating with a group of other universities in the VELA project [VEL97] in an effort to build momentum towards a standard design environment for the Web. Important background research as well as current research into similar- or sub-areas is discussed in this section. It should be noted that while an effort is made to present as many of the important technologies under development as possible, there are currently so many Web-related research projects underway that it is impossible to keep track of them all.

3.1 Computer-Aided Design

In the early part of this decade, EDA systems were built upon framework models [HRS+90, WOL94, BRO92, NCS87, JOH92, SS90], which were organized as shown in Figure 3 [HRS+90]. Operating system services were abstracted as supporting user interaction, process services, local storage, or network services. These facilities were then grouped into those that supported process management and those that supported physical data management, including distributed data management on a network. This model was developed to fit the multitasking process/communication model of the UNIX operating system [BEL78], and is still commonly used by EDA systems today. Specialized services, including user interfaces, design versioning and configuration management, and EDA data representation semantics, were stacked upon this layer. These, in turn, were used by the specific applications, whether design tools, project management applications, or users themselves, to implement their particular task. Systems such as OCT [HMS86, OCT93] were built following this framework approach.

Although the frameworks have helped make EDA a multi-billion dollar industry, a large

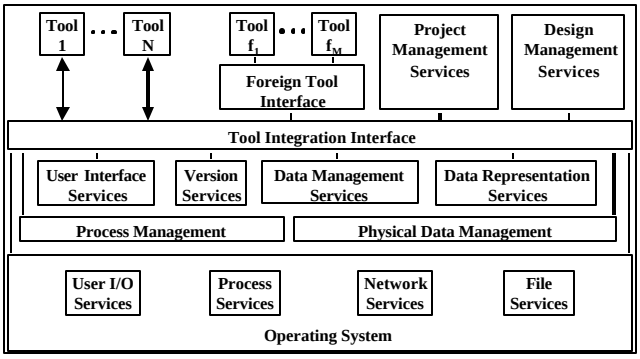

Figure 3: The CAD Framework Model

quantity of the recent research has involved incremental improvements, for instance in the specific areas of design data management [SH94, WGF94], design meta data management [BRE95, FLC95, JB95, SK94, SKR95], and flow, process, and tool management [BBW91, CNS90, DD91, FBM94, HBL+94, HD96, HT90, KLB97, SBD93, SD96, SGM+94, VVS96]. The goal of the WELD project is to take advantage of currently under-utilized, new technology available to the industry to enable not an incremental but a generational advance in EDA technology. Although traditional CAD tools are also being developed by WELD, a primary goal is to build a standard infrastructure in which no restrictions are placed on data representation, design methodology, or usage. It is our goal to provide through a set of general services an abstraction through which CAD developers can easily incorporate their contributions into a globally compatible system.

The network in the WELD model eliminates the local distinction between process, network, and secondary storage services present in the frameworks. Instead, all of these services can be considered simply as different aspects of the abstraction of services provided via the network. This model makes it possible to leverage all of the recent client-oriented services (some of which are similar to the WELD infrastructure discussed in Section 4), such as data manipulation and querying [ASA+95, BDF+97, BEH95, DIS97, GS87, LID96, SOC97, SSU95, VDA96]), visualization for design and collaboration [CRE97, DAR97], workflow management (e.g.[CHA97, CS93, LKB+97, RTD97]), and library and part management (e.g. [ASP97, EDT97, SYN97]).

In the past, much of the distributed tool flow and collaboration work was implemented under the X-Windows architecture or, more recently, using Tcl/Tk [OUS94]. Some specific points of interest include:

- The Design Agent system [BEN97], which presents an information-centric paradigm for CAD systems and could make use of WELD Infrastructure.
- The REUBEN system ([KLB97, LKB+97]), developed by the Collaborative Benchmarking Laboratory at North Carolina State University (a partner in the VELA project), which features user-defined and reconfigurable execution sequences by creating dependency edges between program nodes (application icons) and file nodes (data icons), data-dependent execution sequences by dynamic scheduling of path as well as loop executions, and host-transparency as to the location of applications and data (both can reside on any host with a unique IP address). The system is presently written in Tcl and makes use of popular protocols such as telnet and ftp.[4]

[4] The REUBEN system also makes use of some WELD tools, including the FSM Editor [Leu97].

- Active Documents [SIL94], which presents a Web-based system that embeds compound data for CAD tools in documents using MIME and hyperlinks.
- There are a number of commercial systems available that tackle specific aspects of the distributed workflow management task for EDA (e.g.[ARF90,91, ASP97, EDT97, RTD97, SYN97).

At the present, many companies and universities are evaluating the possibilities of networks, Java [BEA97 BLU97, CCI+97, HAL96, JAV97_1,2, JAT97, MAR97], and advanced object models [BEN95, NII95, ODM97].

3.2 Proxies

There has been significant work done on proxies at UC Berkeley by Professor Eric Brewer's research group. Originally seen as a means of mitigating client and network variation and as a means to cache data [BKK85], proxies can be placed in varying locations (such as the client end, server end, or middle) in a network connection to offer variations in the ACID/BASE models [FGB+97]. As described by Brewer [BF97, FGB+97], proxies have the ability to:

- Support new features without having to add them directly into the servers, as a means to build services before economies of scale exist and add services that the servers are not interested in providing.
- Add services without having to stop the existing servers, which would be detrimental to fault tolerance.
- Perform aggregation, characterization, measurement, and collaboration services over multiple sites.
- Offer increased flexibility (i.e. on where to place the service) and incremental scalability.
- Perform added functionality with low overhead – the pass-through delay of a base proxy, written in PERL, is 3-6% [BMM+96].

Although the proxy architecture developed by this group is very extensive, it does not fully meet the requirements of CAD applications. Currently, the proxies work best when dealing with no or soft state, which is clearly not sufficient for CAD. In addition, their proxies usually have to be trusted in some way. Currently this is not an issue for them, because they are providing services based upon insecure, public data (for instance in the TranSend system [FGB+97], which distills Web data)[FG97]. For a distributed design system there would be numerous additions needed to support both design security as well as some of the necessary ACID semantics. It is our plan to leverage their infrastructure, making the additions necessary to enable it for CAD.

3.3 Collaboration

There are numerous projects underway investigating the possibilities of collaborative aspects of the Web, which are of critical importance to interactive, distributed design environments. CREW, The Collaboratory for Research on Electronic Work at the University of Michigan, focuses on the design of new organizations and the technologies of voice, data, and video communication that make them possible. The CREW Web site [CRE97] contains a comprehensive set of on-line references on collaboration at many different levels.

Another major project that involves the application of visualization technologies to collaboration is DARPA's Intelligent Collaboration & Visualization (IC&V) program, which seeks to enable both teams and teams of teams to collaborate more effectively through distributed information systems that encode relevant knowledge, expertise, and semantics and that permit multiple, shareable views of common collaborative information spaces. Their Web site [DAR97] points to a number of efforts that could eventually be very useful in the EDA world.

4 Server Infrastructure

This section focuses on the server infrastructure built for the WELD project over the past two years in the effort to understand the issues involved in the design and implementation of a distributed design system. Soon after the project began to use Java, it became apparent that it would be necessary to build additional infrastructure, such as server wrappers and network data servers. Such infrastructure was investigated, built, and then successfully demonstrated at several conferences [DAC96, 97], with applications such as the Distributed Workflow System. All of these are discussed in detail in the following sections, along with some performance data in the final section.

4.1 Server Wrapper

4.1.1 Introduction

When we began to explore the uses of Java in the WELD project, it quickly became clear that additional infrastructure would be required in the network. The Java security model prevents users from saving and loading data to the local hard drive, leaving the network as the only means of accessing stored data without running the Java code as an application instead of an applet. The Netscape Java implementation also restricts the set of machines to which an applet can connect to the machine from which the applet was downloaded, but for a feasibility test this was sufficient. The first server written was a program watching a socket interface that allowed Serena Leung's FSM Editor [LEU97] to save and load files over the network.

It became apparent during the development of the save/load server that such a server could also be very useful to access legacy batch UNIX tools through Java. A few modifications to the save/load server transformed it into a server that would allow users to run the state assignment tool Nova [OCT93] over the network through the FSM Editor. These two servers, in conjunction with a similar server that wrapped around the Synopsys Design Compiler [DES96] provided the basis for the 1996 WELD demonstration (see Figure 4) at the Design Automation Conference in Las Vegas [DAC96].

The experience with these servers and the abundance of possibilities that such servers could offer to a distributed system motivated the creation of a generic piece of code that would allow potential service providers to quickly and easily make their own services available on the network. The result of this effort was the server wrapper [SPI97].[5]

[5] It should be noted that the encapsulation performed here is at a much simpler level than some standards, such as the CFI encapsulation model [CFI91] or CORBA [OMG97].

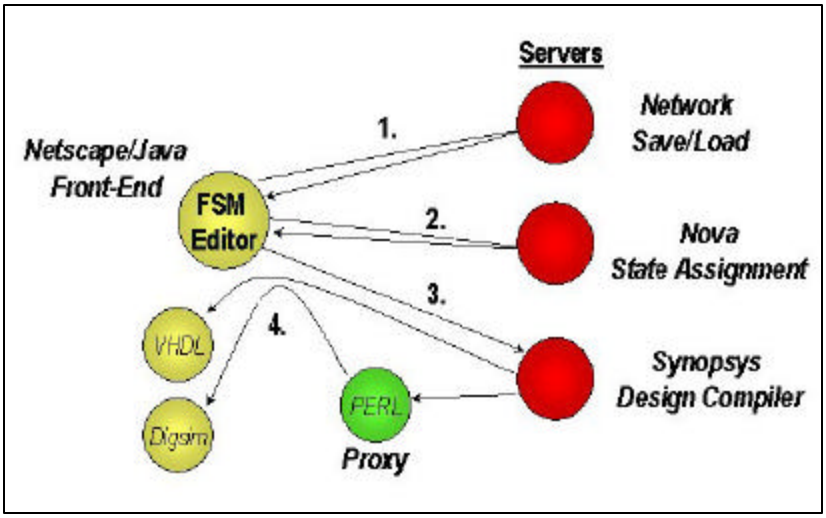

Figure 4: DAC '96 Demonstration Flow

4.1.2 Description

The server wrapper provides a layer of abstraction on top of UNIX sockets [STE90], allowing users to add tools to the WELD system without having to deal with most of the details of sockets and calls to the UNIX exec system call. Originally, the server wrapper was written in C, but over time it evolved to C++ as the group began to use object-oriented techniques and object databases (see Section 4.2). At this point, the server wrapper contains:

- A network package (the NetChannel class, built to use the SocketClient and SocketServer classes, both of which inherit from the SocketHandler class) that allows the program to switch between the roles of a client, server, and proxy in the WELD system.
- The capability (along with examples) to send heartbeats and periodic updates to calling clients and servers, providing status and failure information to the other side of the connection.

4.1.3 Conclusions and Future Directions

The server wrapper proved to be instrumental in the WELD demonstration for the 1997 DAC [DAC97]. The server wrapper release was used to quickly integrate into the demonstration various other Berkeley CAD tools, including SIS [SSL+92], POLIS [POL96], VIS [VIS96], and Tycho [TYC97]. Other research groups have also used both the wrapper as well as the underlying object-oriented socket code as a basis for additional functionality in their own systems.

It is likely that another version of the server wrapper will be necessary in the future, as support for collaboration (perhaps interaction and multicast) and interactive design sessions are added into the WELD system. The current wrapper is geared more towards batch processing, and would need to be expanded to efficiently handle state and data over multiple sessions (i.e. debugging a design being simulated on a remote server). Finally, it would also be useful to be able to wrap tools that run on non-UNIX platforms, especially Windows NT, which is being used increasingly for CAD [PER97].

4.2 Data Server

4.2.1 Introduction

The location, accessibility, durability, security[6], and consistency of data storage are of crucial importance to a distributed design system. In an effort to utilize existing technology, an object-oriented database [OBJ97_2] was chosen to handle the back-end data. It should be noted that there is no underlying restriction on data back-end implementations, and that it should be quite feasible to build similar support on top of other platforms such as traditional file systems, hierarchical databases, and relational databases. The intention in using the database was to try and leverage the built-in services, including versioning, transactions, replication, search and query capabilities, fault-tolerance, and migration, to:

- Make it easy to create service using the network.
- Promote the creation of flexible object-based data models on the web.
- Simplify the transfer and translation of data between different tools.
- Build infrastructure that would allow us to create and test a prototype distributed design system on the Web for CAD.

An object database was chosen because it was readily available and seemed to match the object nature of Java.

The end result of this research phase became persistent Java object management package [CHA97] that was used to develop a web-based version of the Berkeley OCT Tools [OCT93], a web-based project manager [CHA97], and a web-based workflow system (see Section 4.3 below).

[6] It should be noted that although security is also a primary concern for a distributed design system, our system is a proof of concept prototype, and does not currently protect data from hostile or malicious attacks. Although we continue to monitor efforts in the area of web security, at this point we are not actively engaged in such research.

4.2.2 Description

The data server watches a port on the network, providing data services to remote applications upon request. Since it accepts generic network socket connections, it can accommodate any client language that can open a socket, including Java, C/C++ and PERL, thus allowing the server to support both next-generation and legacy tools.

Messages are sent to the server in text, following the WELD client/database communication protocol (Appendix 2, Section 7.2). Although fairly short and straightforward, this protocol can be used to create and link existing objects in a completely arbitrary manner. The persistent objects used in the front-end clients are mirrored onto the server over the network using a translation mechanism that the protocol enables. A "root directory" into the object system is provided via the protocol, from which the client can negotiate through the object graph using unique object ids or connection information provided by the protocol. Although a "root" is provided, no structure is imposed onto the application programmer, and there are many alternate ways in which objects can be ordered and stored, in order to leave as much flexibility and power as possible in the hands of the application developer.

Upon message reception, the data server goes through a dispatch loop in which actions are taken based upon the message. The object-oriented nature of the dispatch makes it easy to add new commands by simply creating a new class to handle the actual message using the existing messages as templates. Current messages exist in the protocol to handle the save, modification, browsing, connection, deletion, versioning, query, and retrieval of objects. In addition, a set of messages based upon the HTTP protocol [HTT97] has been implemented that allows users to peruse the data contained in the database via a web browser such as Netscape Navigator [NET97_2].

4.2.3 A Web Version of OCT

The first application of the data server and the persistent object management package was a web-based version of the Berkeley OCT Tools. OCT acts as the data manager for a large number of VLSI/CAD applications following the Berkeley CAD framework. Although the OCT Tool system was designed before the advent of object-oriented programming, the attachment structure between OCT Objects can be very nicely represented using modern object technology. This, in conjunction with the fact that OCT remained well-known and used at Berkeley, made it a perfect candidate for a web-capable object-oriented rewrite using the data server. The class EECS 244, taught by Professors Newton and Rabaey in Fall Spring 1997, presented an ideal opportunity to subject the system to users, providing both feedback on the usability of the server and package as

well as a means to test functionality and limitations. A schema similar to the OCT schema was developed in the OODB, with the cell as the basic design unit. A cell may have many views, such as schematic, symbolic, physical, simulation, etc… Facets exist beneath the level of views, and can contain instances of other cells (which may in turn contain instances of other cells) as well as other collections of attached objects, such as bags, boxes, terminals, and properties. The data server follows the OCT convention that objects can be arbitrarily attached to each other, providing a general mechanism for the organization of data upon which specific policies can be imposed by the application developer. The role of the Java persistent package is to provide a layer of abstraction on top of the data server that presents an interface similar to that of the OCT generator interface.

The three main implementational differences between (Java) 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 has been utilized.

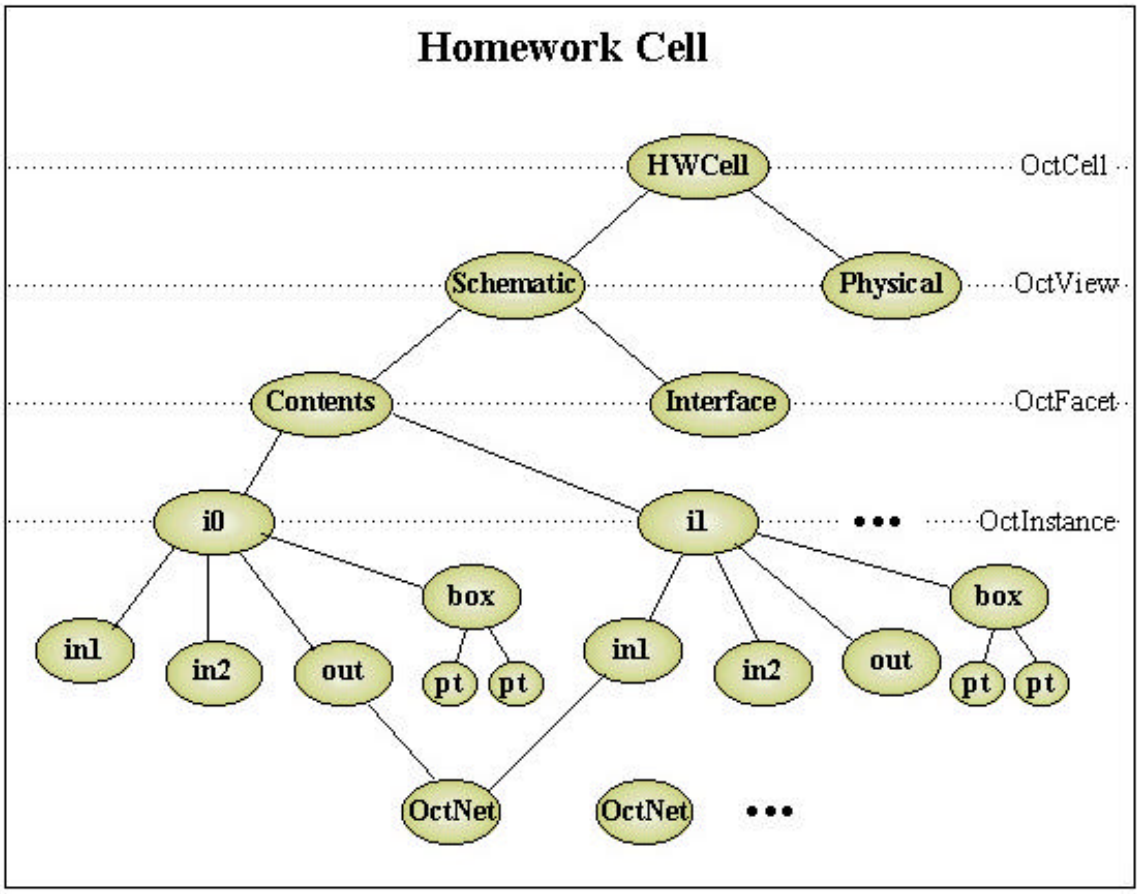

Figure 5: A Sample Design in the Java OCT System (From EE244)

2. An object-oriented database is used as a back-end (for load, save, queries, etc.), as opposed to a UNIX file server.
3. Operations are envisioned to occur over a wide area network (WAN), requiring a greater emphasis upon protocol efficiency and minimization of network transactions than in OCT, which was primarily local and used RPC [CHA97, FBG+96, OCT93].

4.2.4 Conclusions and Future Directions

Tool providers have realized the possibilities of combining database technology with the Web. An example of this is Netscape, which is using part of the ObjectStore database as a tightly integrated part of its browser, as an alternate means to cookies of storing persistent user data and to preserve applet data beyond the end of a session [NET97_3]. Since work began on the persistent object management package there have been numerous industry efforts to fill the needs of the tool providers. Sun Microsystems has released the Java Database Connectivity (JDBC) API [JDB97], and many object database companies [INF97_2, NET97_1, OBJ97_2,3] have committed to creating Java persistent object systems using the Open Database Connectivity (ODBC) and the Object Database Management Group (ODMG) 2.0 [ODM97] standards. These standards provide a model for persistence, class naming, and protocol specifications that allow objects created using the Java language binding to be accessible from C++, Smalltalk, and SQL, and give designers the flexibility to choose the languages most appropriate for their applications. Although our system, which was under development during the same period, does not completely follow the specifics of these models, it is in many ways similar. For instance, in the use of named roots the WELD system prototype currently uses only one, while ODMG provides for multiple named roots, and goes into greater depth for federation management, transaction state integrity, and threading [OBJ97_1]). Similarly, while JDBC offers the options of communicating with the database by installing ODBC model binary code on the client, using Java to send a generic protocol which is translated to a database protocol by the server, or direct connection using an all-Java driver, the WELD system only offers the last option.

In the process, several limitations were discovered in the use of object databases. While the object model is certainly very useful when creating persistent objects for an object-based language, it also has weaknesses. Generally, while one can use compiler and language options to expand the message types and functionality, changes to the object model require a wipe and recompile of the database, making it very difficult to handle schema evolution [CS286]. Additionally, the data is not as portable as that of the tables of a relational database, which can be an issue in a Web-based system that would benefit from data migration outside of the database.

Finally, in some systems the numerous C++ additions, extensions, and macros make the code and database system very complex, and often require a great deal of finesse to compile, debug, and run successfully.

In general, however, the WELD persistent object management system fulfilled its original design goal of providing an infrastructure upon which a prototype distributed design system could be built. This system is described in Section 4.3 below. In the future, this infrastructure will need to be extended to support additional users and applications, but by that point, the industry may have evolved to the point where new off-the-shelf technology could be used.

4.3 The Distributed Workflow System[7]

The distributed workflow system pulls together all of the infrastructure and components that were developed previously, including the server wrapper, proxy, and data manager. The result is a cohesive, distributed design system that both makes use of network tools and addresses many of the criteria discussed in Sections 1 and 2, including fault tolerance, flexibility, customization, and ease of use. This application was designed to test the limits of the infrastructure, as well as generate feedback in the use of and performance of the architecture. An online demonstration (available at [DAC97]) is available, as well.

4.3.1 Workflow Background

In recent years, there has been increasing demand for workflow systems. Numerous commercial products are currently offered, including big names such as Lotus Notes [LOT97], IBM's FlowMark [FLO97], and Digital's Linkworks [LIN97]. The most common use for workflow systems is to track the steps (called activities) of a business processes, i.e. loan approvals, claims processing, or travel reimbursement. Examples of activities for a loan approval process might include processing an application, obtaining a credit report, or getting a signature. A business process expressed as a machine-readable group of activities is called a workflow. Workflow systems are often necessarily distributed because their activities are usually not all available at a single site. In addition, depending on the application the number of business processes can scale to the order of hundreds of thousands. The distributed nature and great number of processes presents a clear need for automated systems that track and report on the

[7] This section is based upon both a class paper that I co-wrote with Kristin Wright for Computer Science 286, Spring 1997], as well as the 1997 Design Automation Conference WELD demonstration, which I organized in conjunction with Francis Chan.

progress of business processes, and as a result there has been a great deal of research and development focused on the area of workflow systems.

Despite the amount of research done on workflow systems, the state-of-the-art still has less functionality, reliability, and robustness than is desired. In general, they have limited fault tolerance, do not scale well, have poor availability due to lack of replication, do not address mobile computing needs and lack in adequate transaction models [AAA+97]. Crucial areas of work include fault tolerance, continuous availability, and replication [MAG+95].

4.3.2 Components

Our distributed workflow system for CAD has five main components that can be distributed and replicated as desired for fault tolerance.[8] The components are:

1. The **workflow monitor** is a platform-independent Java front-end GUI to the workflow system that is used to create a workflow or track an existing workflow. The monitor (Shown in Figure 6) is loaded from a well-known URL using a standard browser such as Netscape. The browser interface provides a simple, familiar point and click UI, and fosters both ease-of-

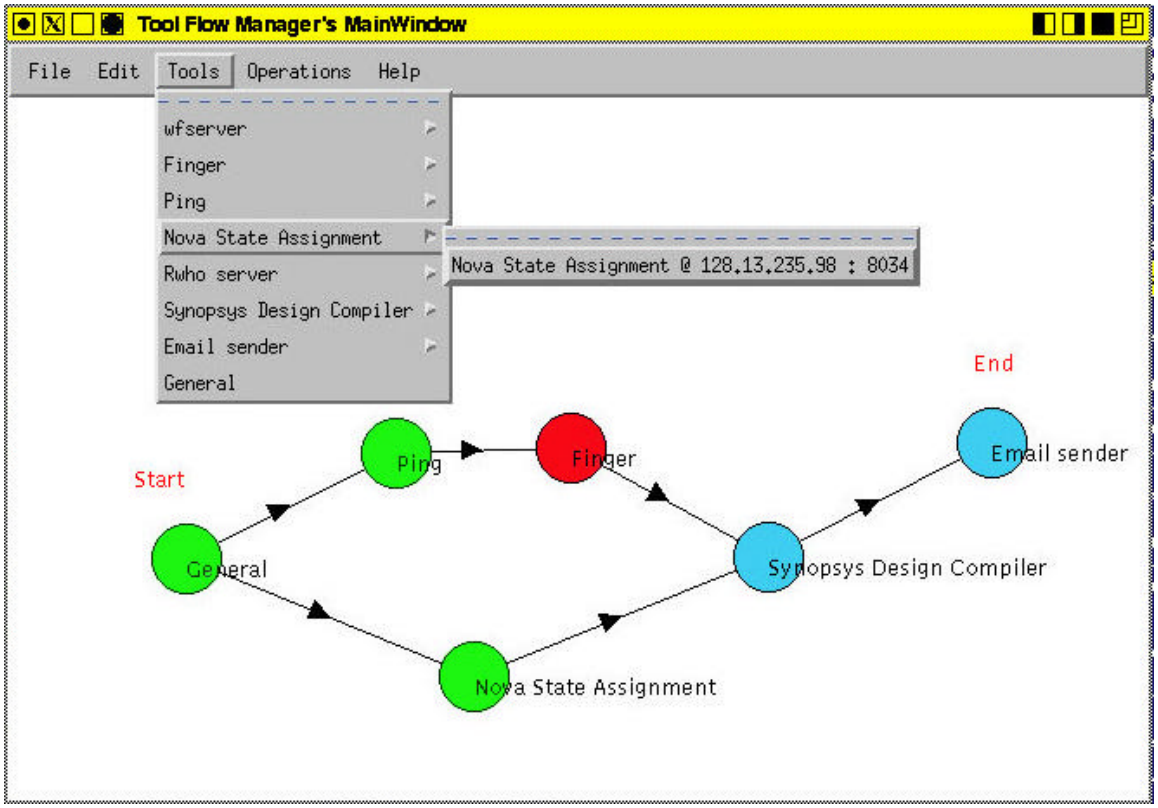

Figure 6: The Workflow Monitor

[8] However, during the development and testing phases, most of the services tended to be run on a single machine for simplified debugging.

use and mobility. Workflow progress is indicated by coloring the tool servers according to whether their job is in progress, complete, or not started. It should be noted that the workflow monitor is primarily a front-end, and that most of the processing, except for graph management, is handled on the workflow server.

2. The **tool server** is the component of the system that offers the tool service and is equivalent to the traditional workflow activity. Tool servers can be easily created using the server wrapper described above (4.1). Upon startup, each tool server registers with the registry and specifies its capabilities, input formats, and output formats. Upon detection of a process request, the tool begins to execute and send status messages until completion, at which point it sends the output data to the specified location.
3. The r**egistry** keeps track of the whereabouts of every available tool server and workflow server in the system. This data can be backed up in a database server, as well.
4. The **database server** is where the final and intermediate results of the workflow are stored and is described in Section 4.2.
5. The **workflow server** is the central point of control for the system. The workflow server is a proxy that sits between the monitor and the rest of the components in the system. It communicates with the registry to obtain information about the availability of tools, with the database server to store and retrieve information about the workflow and results, with the tool servers to actually handle execution, and with the monitor to get the initial design and to return status.

In time, if such a system became popular and widely used, a means of handling resource location would be necessary. While it is realistic to assume that the user can remember or bookmark a well-known URL, the tool servers and workflow servers will need to be able to find registries and database servers (In the current prototype, the locations of many of these components are hard-coded). We have discussed the possibility of using technology such as multicast (which is also used in the TranSend system [FGB+97]) or Marimba [MAR97] channels to broadcast the necessary location data, but have at this time not yet formally designed the functionality. The existence of such systems as Marimba and the Domain Name Servers (DNS) indicate that this is a problem that can be solved using off-the-shelf technology.

4.3.2.1 Component Interaction

Figure 7 illustrates the component interactions that occur in the system when a workflow is created. Since this example is meant to give a high-level understanding of the interaction, many details are omitted.

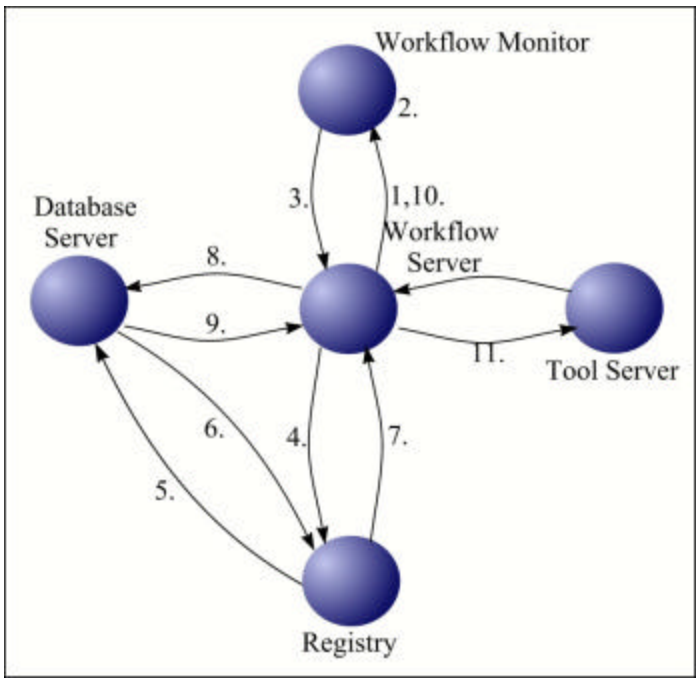

Figure 7: A Sample Workflow Interaction

- The user accesses a URL to download the tool-flow monitor, which connects to a workflow server (1).
- Using the monitor, the user builds a workflow by choosing tool servers from a menu, dropping them onto the page, and connecting them with edges. In addition to this graph, the user must also specify an email contact point to which status messages should be sent, which intermediate results should be stored permanently, and the location of any necessary input files (2).
- Upon execution of the workflow, the applet uploads the completed workflow specification to the workflow server (3).
- Upon receipt of the workflow, the workflow server queries the registry to verify that the activity list has a valid ordering (4,7). The registry does so, perhaps contacting the database server for updated information (5,6). The workflow server then performs some load balance checks; it can offload the responsibility for the workflow to a less loaded workflow server if necessary.
- The workflow server sends the updated completed workflow to a database server, where it is safely stored and can be replicated on any other existing database servers (8). The information saved includes the order of tool servers, email contact point, intermediate result commands, and the location of the workflow server that "owns" the workflow.

- After the workflow data has been stored, the necessary access information that the workflow server needs to access the workflow's information is sent (9). This data is also returned to the monitor (10), so that the user may perform status queries or reconnect later following some types of failure.
- Finally, the workflow server begins the execution of the workflow by sending the input file(s) to the first tool server(s) in the graph (11).

4.3.3 Transaction Model

Workflow systems are usually characterized by long-lived transactions, and thus require advanced transaction models [SSU95, MAG+95]. The CAD tools offered by the tool servers could take hours or days to complete. Basic functionality requires that the end result of the workflow be stored persistently, and fault tolerance and efficiency require that intermediate results be at least temporarily stored so that a design can be resumed from the last successful point rather than restarted in case of a tool server failure. Although there is a cost involved in writing the intermediate data, in the case of long-lived or failure-prone workflow the storage costs could prevent even more expensive redo operations.

To meet the distributed workflow system needs the traditional transaction model is relaxed to include groups of what would normally be considered individual transactions rather than each individual transaction. The level of resolution allowed for commits is that of the activity, since (1) the main system does not necessarily have access to the data at the resolution of the tool server internals and (2) any larger granularity will not provide the sufficient fault tolerance. Intermediate results, if not specifically saved, are deleted once all subsequent activities depending upon them have committed their results.

In the case of a tool server failure caused by a temporary network partition, it is possible for the failed node to come back up after the workflow server has detected failure and restarted the activity at a duplicate tool server (see Fault Tolerance below). In this case multiple tool servers may attempt to write results and could cause possible inconsistencies. This problem could be avoided by imposing the rule that only the tool server recognized by the workflow server is allowed to write results. However, in a case where payment was involved, some special conditions might be required to prevent double charging (i.e. perhaps payment is rendered for jobs completed only[9]).

[9]Even that might not suffice, though – for instance, the tool provider is not to blame for network partitions. Internet payment is a very open topic, but not the focus of this paper, and so is not discussed further here.

The following paragraphs summarize how the relaxed transaction model handles the ACID transaction model properties.

- Atomicity is guaranteed at the level of an activity. The data is forced to disk by the data server before the activity commits. This requirement does not introduce any perceivable performance degradation when transactions are long.
- Transactions are consistent and no intermediate states are observable because either a tool server's result is visible (committed) or not. Two-phase commit is used so that no data server shows a transaction committed until all do. Only currently running tool servers authorized by the workflow Server are allowed to write results.
- Transactions are isolated because a workflow defines a dependency list between activities. Since one activity cannot start before all previous activities commit the transactions are effectively isolated.
- Durability is guaranteed by forcing the data to disk before the transaction commits. This is not as much of a problem as it would be in high-transaction systems, since jobs are generally long-lived.

This discussion of the ACID properties is, unfortunately, fairly trivial, primarily because the workflow system does not at this time support true collaboration. Currently servers could be set up to provide checkout and locking similar to that of the RCS and CVS [MP97] programs, with a server (or perhaps a proxy) handling the merge activity. Adding functionality to support features such as interactive sessions that cache data in tool servers or maintaining a consistent view of shared data for multiple users present more challenges and would make the ACID description more interesting for the system. We intend to look into these areas as part of our future work.

4.3.4 Fault Tolerance

Failure in this system is defined as a lack of progress in a workflow. This lack of progress can be either caused by a failure of the node on which a component resides, a failure within the component itself, or a network failure. There are two primary components to our fault tolerance scheme: (1) *heartbeats*, which are sent from component to component to convey the fact that they are still running, and (2) a redundancy system, through which workflow servers recognize when another workflow server has died and the responsibility for its workflows must be redistributed. Not all of the features described have been implemented at this point.

4.3.4.1 Failure Detection: Heartbeats

For any workflow to execute, at least one workflow server and one database server must be alive and reachable from the user's physical location. Although the system will run with this minimum configuration, these two servers would be single points of failure, and would not provide the same level of robustness that multiple servers would. Different components are kept aware of the continued existence of other running components in the system by sending heartbeats from (1) database servers to workflow servers, (2) active tool servers to their responsible workflow server, and (3) each workflow server to its redundant workflow server(s).

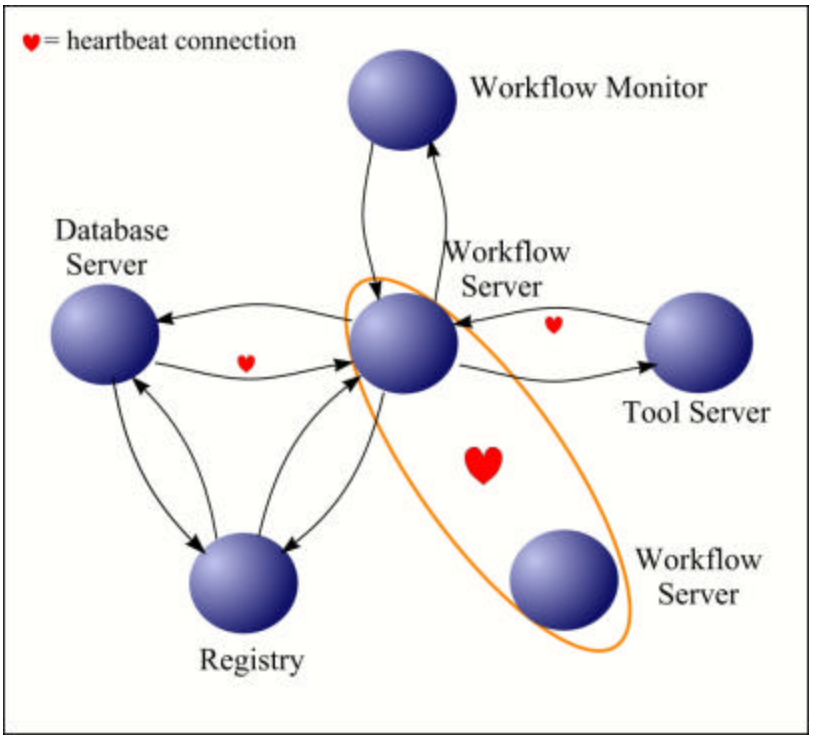

Figure 8: Fault Tolerance Mechanisms

The workflow server must be able to assess that at least one database server is alive and reachable. To accomplish this, the database servers can send a heartbeat out at a system-specified interval to the workflow servers' multicast address. In the case of redundant database servers, the database servers also listen to this channel, so that they can suppress their own messages within a certain interval to avoid message implosion. The registries can also listen in on this channel to update their information about database server resources. It is key to note that although redundant messages should be kept at a minimum, every database server must send out its own message at some minimum interval, to make failure apparent.

Active tool servers send heartbeats back to the workflow server that initiated their jobs. If a workflow server does not receive the heartbeat within a certain interval of time, it assumes that

the tool server has failed and will attempt to find another, duplicate tool server to execute that activity. If it finds one, it restarts the activity and notifies the database server. If it does not find one, it will notify the user either directly through the monitor or by using the email contact point saved with the workflow state in the database.[10] The workflow server also removes the tool server from the registry and sends an update to the monitor, if it is running.

4.3.4.2 Workflow Failure Recovery

Ideally, each workflow server would have a backup workflow server willing to take responsibility for progress should some failure occur. The user could choose this redundant server, or the registry could use its history and status information to automatically assign a suitable backup server from its list of all the workflow servers. Workflow servers would, of course, be updated if there were any changes in the backup system.

Each workflow server sends a heartbeat to its backup at constant intervals. If the backup does not receive a heartbeat for a specified interval, it attempts to make contact. If no contact can be made, the backup assumes that the server is unavailable, queries the Database Server to find which workflows the server had responsibility for, and then attempts to do load balancing of those workflows between the remaining workflow servers. Email is sent to the contact point notifying the user of the failure and the URL of the new workflow server. The current activity, if it is still running, can be updated to send heartbeats and status messages to the replacement server. If the tool server is also no longer available (i.e. due to a network partition) a new tool server must be found through the registry.

It is possible that a network upon which the system resides is partitioned in such a way that each half of the network has all of the components necessary to carry on and the workflow and its backup are separated. Although the execution of the workflow could continue in tandem, for cost reasons it might make more sense to use some sort of quorum scheme in conjunction with the system to prevent needless redundant computation. This is a complex situation, however, and the prototype system has not reached the point where it is beneficial to address these issues, although it is recognized that these will become issues for scalability.

4.3.5 Conclusions and Future Directions

Although the full fault tolerance system is still incomplete, the current system prototype includes all five components, works with basic design flows, detects failures, and allows a user to

[10] Of course, this is highly dependent on commercial issues. Currently, all of our tools are available free of charge. In the future, with commercial tools and associated fees, this policy may need to be revised.

track a workflow monitor. Although there are many loose ends to be tied off and further explored, the existing server wrapper and data manager infrastructure made it very easy to quickly incorporate new tools into the system. Current tools include both simple programs, such as rwho, finger, email [MAN92], VIS [VIS96], SIS [SSL+92], POLIS [POL96], and the Nova state optimization tool [OCT93], and several more advanced CAD Tool Servers, such as the Synopsys Design Compiler [DES96]. In addition, a number of tools were rapidly developed and integrated into the system by outside developers at Berkeley, including an interactive SIS session tool and Tycho [TYC97].

The development of this system has uncovered many possibilities for distributed resource organization, management, and selection. The distributed nature of the tools and the placement of the workflow server as a proxy makes it very easy to track tool information, including function, failure rates, help information, and sample designs. In addition, the role of the workflow server as a proxy simplifies the task of both the client as well as the actual tool servers while offering a convenient, scalable middle-point at which additional functionality can be added. For instance, the proxy offers, transparent to the user, fault tolerance (using heartbeats and the backup workflow server), replication, and the ability to re-start failed designs at checkpoints.

Current limitations of the system and possible future areas of study include interaction with existing workflow systems, security, collaboration, and fine-grained transactions. While the first is not really a research topic and the second is beyond our scope, the third and fourth points are relevant and of interest to us for further research. Smaller-grained transactions and interactive tool sessions are of particular importance in CAD, where some tools generate 1500 or more files over the course of the tool's execution [CAS97]. If failure occurs in the middle of such an activity, it would be desirable to pick up at the failure point. This functionality would require that the tool be capable of handling the state of its checkpoints, perhaps writing out internal, intermediate data to the database servers or some other persistent storage. This might require some modifications to the transaction model.

4.4 Experimental Results

Our prototyping efforts provided us with a wide range of experience ranging from performance to system organization and distribution. For instance, the prototype workflow system was developed and tested on a high-speed local area network, and thus does not represent accurately a widely distributed design environment (although it is very representative of a typical company's Intranet). Several measurements were made to gauge the effects of wide area networks on the system using a machine at North Carolina State University (NCSU). It should be

noted that these measurements are far from exhaustive or complete – they are meant to provide an estimate of the orders of magnitude involved, in order to gain feedback on the prototype and pinpoint areas for future work. Tests were run at least several times, and each individual value was derived over an iteration of 10-20 transfers with the maximum and minimum value dropped[11].

4.4.1 Java

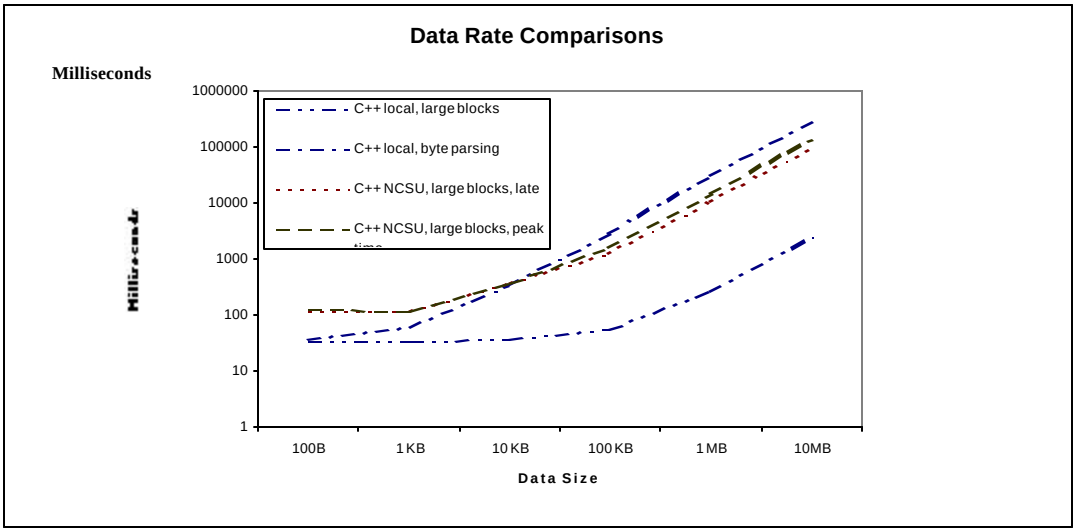

Figure 9: Data Rate Comparisons

Figure 9 shows a set of data rate comparisons for both local and network data transfer. Naturally, the C++ server on the local area network operated much faster than the remote server or the local server that parsed byte by byte (as is required by the current variable-length message structure). Similarly, it is reasonable that the peak time transfers to NCSU were slower than the late-night transfers. What was very surprising and revealing was the fact that the byte parsing, even in C++, delayed the system more than the most expensive wide-area network. Modifications to the message protocol to reflect these costs could provide significant performance improvements. Table 1 presents the values of one of the test runs, along with comparisons between the transfer rates. For the smaller data sizes, there is a clear minimum time interval required, as well as odd intervals, possibly because of the underlying network (For the local FDDI network, for instance, there are minimum guaranteed data rates for each workstation, which skew the results somewhat). However, for the larger data transfers, the measurements scale reasonably.

[11] For some of the larger data values, especially 10MB, the number of runs was reduced, out of politeness.

Figures 10 and 11 compare the network performance of C++ and various Java implementations. These results indicate that the basic network capabilities (Figure 10) of C++, the Sun Java Developer's Kit (JDK) 1.02 [JAV97_1], and Netscape 3.01 (Sparc) are very similar, which is reasonable, since all three were running on the same architecture, on top of the same kernel networking libraries. The Netscape 4.01 (PC) implementation results were radically different, but this may also be explained by the fact that the PC was on a switched ethernet instead of FDDI.

Table 1: Data Rate Comparisons

Data Size	C++, large buffer (ms)			C++, byte parsing (ms)			NCSU, large buffer (ms)			Ratios		
	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	local	parse	NCSU
100B	36	24	51	37	27	49	114	101	130	1	1	3
1 KB	35	24	46	61	50	71	114	101	130	1	2	3
10 KB	36	26	48	332	323	345	359	347	374	1	9	10
100 KB	56	44	66	2917	2895	2951	1243	1209	1286	1	52	22
1 MB	270	238	359	30510	29736	31789	10482	9959	11718	1	113	39
10MB	2446	2303	2717	284207	282755	286085	97506	97332	97745	1	116	40

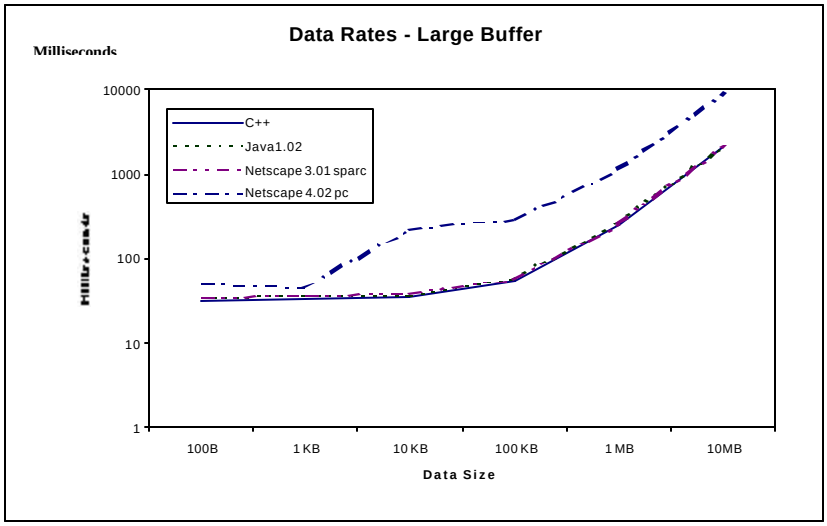

Figure 10: Large buffer data rates for C++ vs. Java

When the measurements were changed to take into account parsing (Figure 11), the Java JDK 1.02 performed nearly as well as C++, but the Netscape implementations performed more poorly.

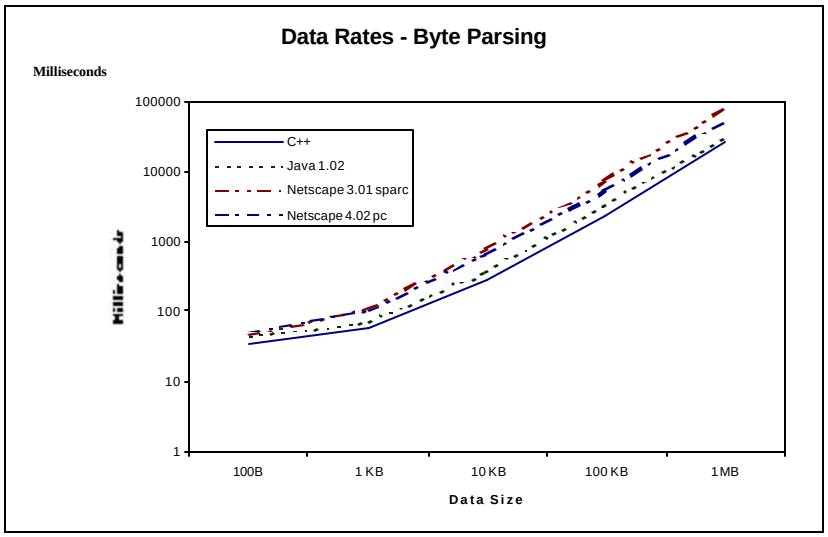

Figure 11: Byte parsing data rates for C++ vs. Java

The preceding measurements mirror a variety of problems that were encountered while using the various Java implementations. Much of the development of the Java side of the system was done using the JDK. This environment offered very favorable performance and promised a very portable Java front-end. However, this was not the case, as was discovered when it was necessary to run the 1997 DAC demonstrations in the Netscape Navigator or Internet Explorer browsers. Variations and inconsistencies in every version (many were tried) of both Java browser implementations required numerous rewrites and fixes, leaving us somewhat frustrated with the language, although still loyal to its ideals.

Another interesting trade-off was that between object resolution and Java performance, the latter of which was feared to be a system bottleneck. Java applet performance, however, was quite reasonable, and will probably only get better, due to the anticipated performance improvements that future optimizations such as just-in-time compilers will offer. In addition, the focus on a proxy-based architecture (See Section 5), which would off-load much of the work onto proxies, would mean that the Java client would primarily serve as a lightweight communications and user interface tool. Object granularity turned out to be more of an issue than the Java performance.

4.4.2 Database

Java tools developed in the WELD group are divided into two groups: (1) applications such as the SpecCharts Editor, which runs purely in Java and used the network primarily for save and load, and (2) applications such as the distributed workflow system and the Project Manager, which have a heavier dependence upon the network back-ends. In the latter case, and when

attempting to upgrade the SpecCharts Editor to use the data manager, the latency and connection cost of transporting the resulting large number of persistent Java objects over the network made system performance unacceptable. Therefore, several schema and protocol additions were made. For instance, the option to use the protocol in a connection-oriented manner was added (instead of the previous connectionless-only model) along with Data Objects, which remove object structure and hierarchy (serving the role of files instead of actual Java classes) but allow for larger and more efficient data transfers. Even with Data Objects, however, there were problems in Java with the time required to process large amounts of data received over the network. In this case, it might prove to be more optimal to make use of or develop new technology to instead move the programs to the data, especially in CAD applications where the data size can often push the limits of machine main memory.

The data transfer and organization questions and the appropriateness of the object database back-end issues were raised at several points during the implementation of the prototype. While the object-oriented nature and built-in capabilities made the database ideal for the storage of Java objects, the difficulty of schema evolution and overhead in object transfers made the choice sometimes rather frustrating and inefficient. The performance of the object database also turned out to be less than ideal. The results shown in Table 2 indicate that the data manager had nearly the same performance taking its input from a file as opposed to from a Java network connection for small objects (the experiment involved 23 small transactions), which leads us to believe that the data manager, not the Java client, was the bottleneck in the system. We intend to explore the new off-the-shelf persistent object packages to see if they have similar problems and to gain further insight into this performance problem.

Table 2: Java vs. file input to the data manager (ms)

Java, DB debugs off	Java, DB debugs on	File, DB debugs off	File, DB debugs on
10505.23333	10873.54933	10660.05875	11071.29325

A data service built upon a file system would have provided a solution very portable to other platforms, but would have required a great deal of additional infrastructure to handle many of the built-in capabilities of the object database, including versioning, transactions, object-linking, ids, and multiple hierarchy. A relational database provides middle ground between the file system and the object database, offering many of the database capabilities of the object database, but still requiring additional modifications to add new data types and object hierarchies (see table in [CHA97]). Since the beginning of the project, object-relational databases [INF97_1] have become widely popular, and may offer improved trade-offs. Perhaps the most interesting

new architecture, following our current research focus on proxies, would be a generic, simple data back-end and a set of proxies to manipulate and store data to it depending on the dynamic needs of the user and application. The general yet adaptable capabilities of such a model suggest that it would be the best all-around solution to the network-data management problem.

5 Conclusions and Future Directions

Computing infrastructure will continue to play a large role in determining overall EDA tool integration strategies, approaches to data visualization and collaboration, and even the most effective choice of algorithms for tools. Developments in the network service layer offer the potential to enable transparent distributed EDA systems that are reliable and secure and that make distributed collaborative design possible. Associated developments in visualization technologies, both for data and to support collaboration, are likely to make such environments both efficient and effective for the design and transfer to manufacturing of electronic systems. The WELD project will continue in the effort to use emerging technologies, such as distributed computing infrastructure, for electronic system design. The prototyping efforts and infrastructure made to date include a data manager, a distributed workflow system, a server wrapper, and a Java client package [CHA97], all built with scalability, flexibility, and extensibility in mind. Performance tests indicate that such a system, based upon Java front-ends in conjunction with native proxies and back-end servers found in the network, is indeed feasible, although there are still issues that need to be addressed.

Although there is a great deal of core functionality in the prototype system, it currently only supports one designer per project, and will require improved support for both geographic and temporal collaboration. In addition, Java still has problems that need to be overcome, including performance and true platform independence (in the implementations). Success depends both upon such technical factors as well as upon social factors. For instance, a new framework must provide sufficient features, be easy to use and extend, and address security concerns, before the EDA user population will completely embrace it. Our experiences at the Design Automation Conferences indicate that there is both a great demand and great research value in such a system, and thus we plan to work with other universities to continue to build the infrastructure necessary for a distributed design environment.

5.1 Future Directions for Collaboration

The key to successful collaboration will be state management. It will be necessary to build an architecture in which the desired ACID semantics can be dynamically chosen, providing soft state, locking, transactions, consistency, availability, and quality of service on demand. Areas of interest include:

- Generic data models, such as the one under development at IBM for the Silicon Integration Initiative (SI2, formerly the CAD Framework Initiative, CFI) [SI297], and data-centric design [BEN97]. These are currently being evaluated.

- The use of distributed caches in a potentially unreliable network.
- The use of multicast technology [MBO97] for resource location and as a means to relax the ACID semantics [FGB+97]
- Optimizations necessary for efficient design iteration over the network [CLE97, CON97].
- The possibility of dynamically creating and distributing interactive Java applets for use both in the visualization of data, tools, and the current status of the system, as well as to move the applications to large data sets that should not be sent over the network.

5.2 Future Directions for Proxies

As mentioned previously in Section 3.2, we see proxies as an ideal way to support the collaborative and state management issues mentioned above. We envision a general architecture (similar to that of [FGB+97]) in which all the proxies in the EDA network are relatively simple and architecturally identical, with a set of API's providing a means to extend the interfaces, as shown in Figure 12. In this model, the client side could easily be extended to customize and support human, tool, data/processor cluster, or service use. The network API's, meanwhile, could be used to implement the various criteria (i.e. reliability, availability, and security) outlined earlier. The proxy would be able to incorporate abstract datatype handlers as plug-ins, store client-specific information locally in a profile, and either implement or make use of caches. We plan to use proxies to enable an adaptable network fabric that automatically adjusts to computer and network reliability issues.

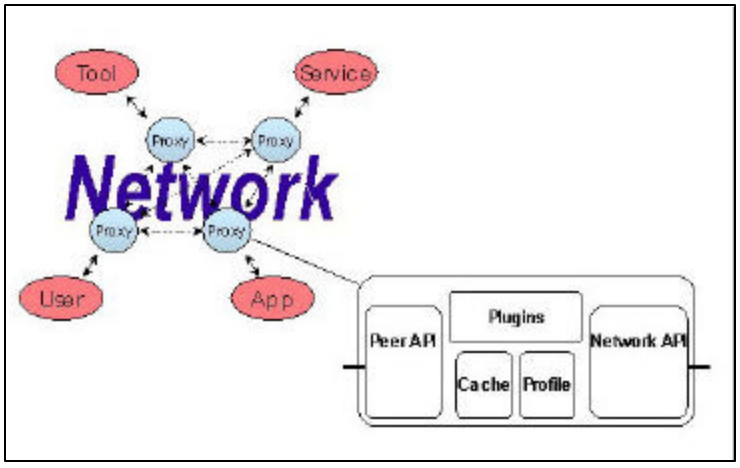

Figure 12: A Proxy Architecture

6 References

[AAA+97] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan, "Functionality and Limitations of Current Workflow Management Systems," *IEEE Expert* (Special Issue on Cooperative Information Systems), 1997.

[ARF90] W. Allen, D. Rosenthal, K. Fiduk, "Distributed Methodology Management for Design-in-the-Large," *IEEE/ACM International Conference on Computer Aided Design*, pages 346-349, 1990.

[ARF91] W. Allen, D. Rosenthal, K. Fiduk, "The MCC CAD Framework Methodology Management System," *Proceedings of the 28th ACM/IEEE-CS Design Automation Conference*, pages 694-698, 1991.

[ASA+95] M. Abrams, et al., "Caching Proxies: Limitations and Potentials," *Proceedings of the 4th International World Wide Web Conference*, Boston, MA, Dec. 1995.

[ASP97] Aspect Development, Inc., http://www.aspectonline.com.

[BBW91] 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.

[BDF+97] W. J. Bolosky, et al., "Operating System Directions for the Next Millennium," Position Paper, available at http://www.research.microsoft.com/research/os/Millennium/mgoals.html .

[BEA97] A number of excellent references for JavaBeans can be found at http://splash.javasoft.com/beans/.

[BEH95] B. Behlendorf, "A Proposal for Non-Intrusive Community Content Control Using Proxy Servers," http://www.organic.com/Staff/brian/community-filters.html.

[BEL78] The Bell System Technical Journal, Special Issue on UNIX, Vol.57, No.6, Jul-Aug., 1978, contains a collection of papers describing UNIX.

[BEN95] R. Ben-Natan., "CORBA : A Guide to Common Object Request Broker Architecture," McGraw-Hill, New York, 1995.

[BEN97] O. Bentz, "An Information-centric Design Exploration and Implementation Server", Ph.D. dissertation, UC Berkeley, 1997, http://infopad.EECS.Berkeley.EDU/research/tools/InfoBasedDesign/dissertation/.

[BF97] E. Brewer and A. Fox, "Internet Services," Computer Science 294-6, offered Fall 1997 at UC Berkeley.

[BKK85] F. Bancilhon, W. Kim, and H. Korth, "A Model of CAD Transactions," *Proceedings of the 11th International Conference on VLDB*, 1985.

[BLU97] Bluestone Sapphire/Web, http://www.bluestone.com.

[BMM+96] C. Brooks, M. Mazer et al., "Application-Specific Proxy Servers as HTTP Stream Transducers," *Fourth International World Wide Web Conference Proceedings*, Volume 1, Issue 1 (Winter 1996), http://www.w3j.com/1/brooks.056/paper/056.html.

[BRE95] A. Bredenfeld, "Cooperative concurrency control for design environment," *European Design Automation Conference with EURO-VHDL*, pp. 308-313, September 1995.

[BRO92] J. B. Brockman et al., "The Odyssey CAD Framework," *DATC Newsletter on Design Automation*, Spring 1992.

[CAS97] Personal communication with A. Casotto, April 1997.

[CCI+97] B. Christiansen, P. Cappello, M. F. Ionescu, M. O. Neary, K. E. Schauser, and D. Wu, "Javelin: Internet-Based Parallel Computing Using Java," *1997 ACM Workshop on Java for Science and Engineering Computation*, June 1997.

[CFI91] CAD Framework Initiative, Inc., "Tool Encapsulation Specification Standard," CFI Doc. DMM-91-G-1, 1991.

[CHA97] F. Chan, "Architecture and Infrastructure for a Distributed Design Environment- A Client Perspective," MS Report, University of California at Berkeley, May, 1997.

[CLE97] ClearCase, http://www.pureatria.com/products/clearcase/index.html.

[CNS90] A. Casotto, A. R. Newton, and A. Sangiovanni-Vincentelli, “Design Management Based on Design Traces,” *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 136-141, 1990.

[CON97] Continuus Software Corporation, http://www.continuus.com/ .

[CRE97] CREW, The Collaboratory for Research on Electronic Work, www.crew.umich.edu.

[CS93] A. Casotto and A. L. Sangiovanni-Vincentelli, "Automated Design Management Using Traces," *IEEE Trans on Computer-Aided Design*, Vol.12, No.8, pp.1077-1095, Aug. 1993.

[CS286] CS286 class discussion, UC Berkeley, Spring 1997.

[CV65] F. J. Corbato and V. A. Vyssotsky, "Introduction and Overview of the Multics System," *AFIPS Conference Proceedings*, No. 27, pp.185-195, Fall Joint Computer Conference, 1965 (At http://www.lilli.com/fjcc1.html.).

[DAC96] Article: P. Clarke, “WWW primed for EDA,” *Electronic Engineering Times*, June 10, 1996, issue 905. Demonstration materials: http://www-cad.eecs.berkeley.edu/weld/dac96/ .

[DAC97] Article: P. Clarke and R. Goering, “Web-based design hoists new sail,” *Electronic Engineering Times*, June 16, 1997, issue 958. Demonstration materials: http://www-cad.eecs.berkeley.edu/weld/dac97/ .

[DAR97] The DARPA Intelligent Collaboration & Visualization program (IC&V), http://www.ito.darpa.mil/ ResearchAreas96/IntellCollabVisual.html.

[DD91] J. Daniell and S. W. Director, "An Object Oriented Approach to CAD Tool Control Within a Design Framework," *IEEE Transactions on Computer-Aided* Design, Vol.10, No.6, pp.98-713, Jun. 1991.

[DES96] The Design Compiler Family Datasheet, http://www.synopsys.com/products/logic/design_comp_ds.html .

[DIS97] The Distributed Clients Project, http://www.osf.org/www/dist_client.

[EDT97] Electronic Design and Technology Network (EDTN), http://www.edtn.com.

[FBM94] 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.

[FG97] Personal communication with A. Fox and S. Gribble, Jan. 1997.

[FGB+96] A. Fox, et al., "Extensible Cluster-Based Scalable Network Services,” *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, St. Malo, France, October 1997.

[FLC95] 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.

[FLO97] IBM Corporation, “IBM FlowMark,” http://www.software.ibm.com/ad/flowmark/ .

[GRA81] J. Gray, "The Transaction Concept: Virtues and Limitations, " *Proceedings of VLDB*, pp.144-154, Cannes, France, Sept. 1981.

[GS87] H. Garcia-Molina and K. Salem, “Sagas,” *Proceedings ACM SIGMOD Conference*, 1987.

[HAL96] T. R. Halfhill, “Inside the Web PC,” *Byte Magazine*, pp. 22-36, March 1996.

[HBL+94] 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.

[HD96] 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.

[HMS86] 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 1986 IEEE International Conference on Computer-Aided Design*, pages 24-27, 1986.

[HRS+90] 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.

[HT90] 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.

[HTT97] The HTTP specification, http://www.w3.org/Protocols.

[INF97_1] The Informix Universal Web Architecture, http://www.informix.com/informix/bussol/uwa/uwa.htm.

[INF97_2] Infoscape, www.infoscape.com.

[JAT97] The Java Agent Template, http://java.stanford.edu/java_agent/html/.

[JAV97_1] The Java specification, http://java.sun.com.

[JAV97_2] Javaworld online magazine contains many useful Java references at http://www.javaworld.com.

[JB95] 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.

[JDB97] The JDBC page, http://java.sun.com/products/jdbc/index.html.

[JOH92] B. Johnson, "A distributed computing environment framework: an OSF perspective," Technical Report DEV-DCE-TP6-1, OSF, January 1992.

[KLB97] A. Khetawat, H. Lavana, and F. Brglez, "Collaborative Workflow: A Paradigm for Distributed Benchmarking and Design on the Internet," Technical Report 1997-TR@CBL-02, North Carolina State University, 1997.

[LC96] D. B. Lange and D. T. Chang, "Programming Mobile Agents in Java, A White Paper Draft," IBM Corporation, September 1996.

[LEU97] S. Leung, "A Java-based SpecChart Design System," Masters report, MS Report, University of California at Berkeley, May, 1997.

[LID96] Power Management via the WWW, http://infopad.eecs.berkeley.edu/lidsky/POWER/Power.

[LIN97] Digital Equipment Corporation, "Digital LinkWorks – Delivering Solutions to MIS and End-Users", http://www.aberdeen.com/secure/profiles/declink/declink1/declink.htm.

[LKB+97] H. Lavana, et al., "Executable Workflows, "A Paradigm for Collaborative Design on the Internet," *Proceedings of the 34th ACM/IEEE Design Automation Conference*, June 1997. Also available at http://www.cbl.ncsu.edu/publications/.

[LOT97] Lotus Development Corporation, "Lotus Notes: An Overview", http://www.lotus.com/notesr4/over2d.htm.

[MAG+95] C. Mohan, G. Alonso, R. Günthör, M.Kamath, and B. Reinwald, "An Overview of the Exotica Research Project on Workflow Management Systems," *Proceedings of the 6th International Workshop on High Performance Transaction Systems*, 1995.

[MAN92] Descriptions of finger, mail, and who are available in section one of the online UNIX Programmer's Manual on most UNIX systems, 1992.

[MAR97] Marimba, Inc., http://www.marimba.com.

[MBO97] An excellent introduction to the MBONE, "MBONE (Multicast Backbone)" by Jean Bunn, Geneva University, is available at http://www.unige.ch/seinf/mbone.html.

[MP97] T. Mikkelsen and S. Pherigo, "Practical Software Configuration Management," Allyn & Bacon Publishing, Inc, 1997.

[NCS87] Network Computing System (NCS) Reference, Apollo Computer Inc., 1987.

[NET97_1] NetDynamics, www.netdynamics.com

[NET97_2] The Netscape browser, http://www.netscape.com.

[NET97_3] Netscape using ObjectStore, http://www.odi.com/news/MSandNS.html.

[NEW96] Personal communication with Professor A. Richard Newton, 1996.

[NII95] The National Industrial Information Infrastructure Protocols (NIIIP), http://www.niiip.org.

[OBJ97_1] "Objectivity for Java," *The Insider : News and Views for Objectivity Users*, July 199.

[OBJ97_2] Objectivity, http://www.objectivity.com

[OBJ97_3] ObjectStore, http://www.odi.com.

[OCT93] OCTTOOLS-5.2 Reference Manual, University of California at Berkeley, 1993.

[ODM97] ODMG 2.0 Standard, which includes Java, published July 28, 1997. Press release at www.odmg.org/pressroom/pressreleases/odmg20.htm

[OUS94] J. K. Ousterhout, Tcl and the Tk Toolkit, Addison-Wesley, 1994.

[PER97] Personal communication with various designers at and after the 1997 Design Automation Conference.

[POL96] The Polis home page, http://www-cad.eecs.berkeley.edu/Respep/Research/hsc.

[RTD97] Runtime Design Automation, http://www.rtda.com.

[SBD93] 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.

[SD96] 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.

[SEM97] SEMATECH, http://www.sematech.org/member/division/dsgn/ecad/home.htm.

[SH94] 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.

[SI297] The Silicon Integration Initiative, www.si2.org.

[SIL94] M. J. Silva, "Active Documentation for VLSI Design," Ph.D. dissertation, available as Technical Report UCB/CSD-94-843, UC Berkeley, Dec. 1994.

[SK94] M. J. Silva and Randy H. Katz, "The case for design using the World Wide Web," Technical Report UCB/CSD-94-837, University of California at Berkeley, 1994.

[SKR95] 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.

[SOC97] NEC Socks, a access-control method that sets up circuit-level gateways using proxies, http://www.socks.nec.com/.

[SPI97] The WELD server wrapper, http://www-cad.eecs.berkeley.edu/~mds/wrapper.

[SS90] 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.

[SSL+92] E. M. Sentovich et al, "SIS: A System for Sequential Circuit Synthesis," Technical Report UCB/ERL M92/41, May 1992.

[SSU95] A. Silberschatz, M. Stonebraker, and J. Ullman, "Database Research: Achievements and opportunities into the 21st Century," Report of a NSF Workshop on the Future of Database Systems Research, May 1995.

[STE90] W. R. Stevens, "Unix Network Programming," chapter 6, Prentice-Hall, Inc., 1990.

[SYN97] Synchronicity, http://www.syncinc.com.

[TYC97] The Tycho home page, http://ptolemy.eecs.berkeley.edu/tycho/Tycho.html.

[VDA96] Amin Vahdat, et al. "Turning the Web Into a Computer," available at http://now.cs.berkeley.edu/WebOS/webos.ps, 1996.

[VEL97] The Vela project, http://www.cbl.ncsu.edu/vela/.

[VIS96] The VIS Group, "VIS: A system for verification and synthesis," *Proceedings of the 8th International Conference on Computer Aided Verifications*, p428-432, 1996.

[VVS96] 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.

[WWW97] The WWW consortium, http://www.w3.org/WWW/.

[WEL96] WELD: Web Based Electronic Design, http://www-cad.eecs.Berkeley.edu/Respep/Research/-weld, March, 1996.

[WGF94] 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.

[WOL94] P. van der Wolf, CAD Frameworks: Principles and Architecture, Kluwer Academic Publishers, Boston, 1994.

7 Appendices

7.1 WELD Client/Server Communication Protocol

7.2 WELD Client/Database Communication Protocol

7.3 Workflow protocol