A Framework for Collaborative and Distributed Web-based Design

Gangadhar Konduri, Anantha Chandrakasan
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge, MA 02139
{gangs,anantha}@mtl.mit.edu

Abstract

The increasing complexity and geographical separation of design data, tools and teams has created a need for a collaborative and distributed design environment. In this paper we present a framework that enables collaborative and distributed Web-based CAD, in which the designers can collaborate on a design and efficiently utilize existing design tools on the Internet. The framework includes a Java-based hierarchical collaborative schematic/block editor with interfaces to distributed Web tools and cell libraries, infrastructure to store and manipulate design objects, and protocols for tool communication, message passing and collaboration.

1 Introduction

The design of future high-performance VLSI systems will require a collaborative and distributed design methodology due to the diverse expertise required at various levels of abstraction. The emergence of *"systems-on-a-chip"*, with potentially more than 100 million transistors on a single chip, calls for a framework that facilitates fast and efficient information flow. Since designers are often separated geographically, a design environment that allows collaboration among the designers is also required. The high integration levels of future systems call for tools and generators that allow exploration of the design space irrespective of the geographical availability of the design tools. All of these requirements motivated us to build a collaborative and distributed framework where designers can cooperate in accessing and utilizing diverse resources from the desktop.

The advent of the Internet has opened new vistas in this area, and the World Wide Web has emerged as the most desirable platform for distributed access to information [1, 2], enabling the designer to access tools from any terminal on the Internet. The limitations of server side only computation on the Web have been alleviated with the emergence of Java, a platform independent programming language. The serialization mechanisms and the platform independence of Java support collaboration and allow the designers to use their favorite computing platform and browser to run the tools. All of these factors make the Web a universal platform for collaborative micro-system design.

With Web-based CAD, the user gets seamless access to the tools anywhere in the world without the need of a powerful client machine. However, the issues of limited internet bandwidth and secure communications still need to be addressed.

2 Related Work

The popularity of the internet and the Web has lead to several distributed design efforts, some of which are summarized below. Bentz *et al.* have proposed an information based design environment [3], in which the users collect and manage information in a uniform fashion, independent of the abstraction levels or implementation platforms. Lidsky and Rabaey presented a Web-based prototype tool, PowerPlay [4], which helps in system level exploration of power consumption. The WELD project at UC Berkeley [5], aims to construct a distributed CAD environment enabling Internet-wide IC design. The WELD group has developed a Java Object Database Server supporting persistent object class management. Boglilo *et al.* describe PPP - A Gate-Level Power Simulator [6], which provides a Web-based integrated environment for synthesis and simulation of low-power CMOS circuits. In addition, there have been efforts to provide Web-based Interfaces to executable CAD / CAM software.

A standalone utility, XMX [7], for sharing an X Window system session on multiple X displays has been developed at Brown University. XMX acts as an intermediary between XClient and XServer and though it leads to a collaborative environment, it works only on X-based systems. So, it is neither platform-indepedent nor Web-based. Some other collaborative systems include Xplexer [8], Xshare [9] and XTV [10]. Lavana *et al.* use executable directed hypergraphs to describe collaborative design activities on the internet [11].

Although there are numerous instances of design tools being available over the Web, there has been limited attention to designing a framework which would allow designers to utilize the different tools over the Web in a collaborative fashion. In this paper we demonstrate how a collaborative and distributed framework can be built over the Web. This framework utilizes the core Web technologies to support efficient com-

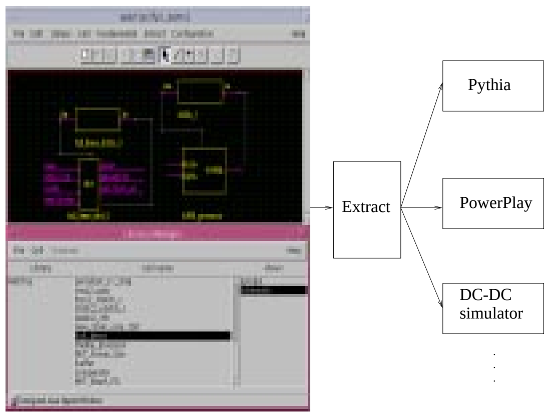

Figure 1: Distributed Framework, WebTop

munication and data exchange between different Web-based tools.

3 The Distributed Web-based CAD tool

A framework for distributed Web-based micro-system design, WebTop, has been developed with a hierarchical schematic/block editor, with a simple GUI, for the design entry. The designer can use the basic cells incorporated into WebTop in their design, or cells created by another designer can be downloaded and used. The tool is Java-based and can be embedded in HTML pages. The designer can extract the netlist from the design so that the output conforms to the input specification of various Web-tools. The tool uses the CGI (Common Gateway Interface) mechanisms to invoke remote tools. WebTop has interfaces with several tools including Pythia (a verilog RTL power estimator tool developed at MIT), PowerPlay (a system level power exploration tool developed at U.C. Berkeley), and WebSpice.

We have implemented a large application on WebTop in order to demonstrate the usefulness of the distributed tool. As a driver application, we entered the design of a single-chip media-processor that includes an embedded ARM core, a video compression module, with more than 160K transistors, and power supply circuits. The design has been validated using a distributed simulation strategy. The remote tools used in this process were Pythia, PowerPlay and a DC-DC converter simulator tool (developed at Stanford). The verilog netlist of the decompression module can be extracted from WebTop and submitted to Pythia to get a verilog power estimate. The PowerPlay netlist can be extracted from WebTop and the results of the behavioral level power estimate can be obtained by submitting the netlist to PowerPlay. The netlist to conform to the DC-DC simulator's input can be extracted and the simulation results can be seen after invoking the remote tool. Thus, the design of a single chip media processor has been validated using the distributed framework, a simple web browser, and three different, geographically separated, remote tools. A snapshot of WebTop with the library manager and the Web-tools used is shown in Figure 1.

4 Collaborative Framework

A collaborative environment is essential for coordinating the work of the designers working remotely on a project. The basis for such a framework is presented in this section.

We extended the distributed Web-based CAD tool, WebTop, into a collaborative framework, CollabTop, whose architecture is described below. The aim is to create an environment where all the designers can see a consistent view of the design in their edit windows.

In a typical session with WebTop, the designer creates a schematic by adding new components either from the primitive cells built into the tool or from the existing cells in the library. The designer proceeds to create another cell or to submit the extracted netlist from the cell to a Web-tool. During the design process, the designer can download cells from the Web and use them in the design. All these designer actions generate events to be sent to other designers.

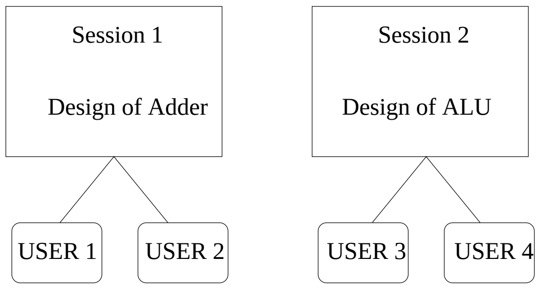

Figure 2: Collaboration with several sessions

In order to make the framework collaborative, the effect of all of the events caused by one designer's actions should be seen by all other designers in the view of their editor. For practical use of the tool, the designers should be able to join other designers in sessions, and several sessions should be able to run in parallel as shown in Figure 2. The events in one session should not affect the design in other sessions. Hereafter, we assume that the designers are in the same session, unless otherwise specified.

4.1 Client-Server Architecture

When a designer edits a schematic, several events are generated. These events can either be propagated to all other designers (peer-to-peer) or sent to a central server which broadcasts these events to all the designers (client-server). CollabTop has been implemented using the client-server model as it is easier to incorporate the higher level ideas such as synchronization into the system using this model. The client-server configuration is also less prone to errors caused by network congestion as it uses fewer channels than a peer-peer configuration. Moreover, a client-server system can be scaled later to support multicasting in the connections from the server to the clients.

Designers can also enter sessions as a read-only client. Events produced by such a client are not broadcast, but the client is able to see a consistent view of the copy being edited. This feature is very useful in the design review phase, where particular parts of the design can be highlighted and hierarchically edited by one or more clients while the read-only client views it.

4.2 Implementation Decisions

The distributed tool, WebTop, was developed as a Java applet, and uses the CGI mechanisms to submit the extracted netlist to various web-tools. The events that are generated because of the actions of a designer prompt the applet to take the necessary action. This architecture of the system has to be changed to the client-server model, shown in Figure 3.

Each client event gets passed on as a message to the server, a stand-alone Java program, which broadcasts them to all the clients, which then take the corresponding action. The server is a stand-alone Java program. Initially, each designer client notifies the server of the designer name and session name to join. The server can be thought of as being composed of two components: the NameServer and the CollabServer. The NameServer takes care of all the messages with the session names and the designer names in the sessions. It also provides the designer with the information on various designer clients present in a session. The CollabServer takes care of all other messages. The designer chooses a session to enter when entering the framework. Designs in different sessions are not affected by each other's events.

The messages that are passed between the clients and the server hold the key to the tool. These messages go from the the clients to the server and vice-versa. There are a variety of ways in which this message-passing can be implemented using the object-web technologies. Distributed object-oriented computing refers to computing environments where programs can make procedure calls to other address spaces, possibly on other machines. Two such technologies are the Common Object Request Broker Architecture(CORBA) and Java Remote Method Invocation (RMI). Another way of implementing this environment is to send messages as objects, using the Java serialization mechanism and the sockets to send and receive the messages. This approach was used to implement the message-passing. Our motivation for choosing this implementation was that it leads to a cleaner architecture and the resulting software follows the intuitive flow of messages. In addition, the implementation of several higher level issues, described in later sections, will be simpler and more intuitive.

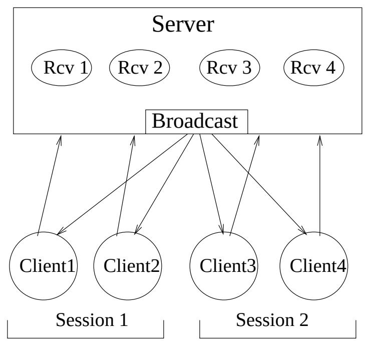

Figure 3: Client-Server Architecture

5 Higher Level Issues

For the simple framework described in the previous section to work, some higher level issues need to be addressed. This includes the design of the components of the messages that are passed, dealing with the reliability of clients and recovering from the failure of any of the clients. Also, issues such as the reordering of messages at the server end and locking some of the critical resources need to be addressed. The motivation and issues behind these concepts are described below.

5.1 Synchronization

The framework, as explained so far, follows the asynchronous model of computing; there is no restriction on the timing of a client event. This leads to the problem of the messages from individual clients being reordered either at the server or at the clients. It is crucial that the consistency of the view in the edit window of different clients in a collaborative session be maintained.

We model the channel between each client and server as a reliable, First In First Out (FIFO), directed communication channel. This matches with the implementation using the Java sockets. By the reliability of the channel we mean that all messages sent by the client are received at the server end of the channel.

At the server there is a thread dedicated to each of the clients. It is this thread that receives the messages from its client. The channel between the client and this thread is modeled as a FIFO channel. We model the channels from the server to the clients also as directed reliable FIFO channels. Hence, messages sent by the server to the clients are received in the same FIFO manner. However, when the server tries to broadcast the messages we need a way of timestamping the messages, to maintain order among the messages from different clients.

The concept of logical time cannot be applied here because it does not model the ordering of events at different nodes. Thus, we need real time to do the time-stamping. We use the Greenwich Mean Time on the corresponding machines to model this. Though, this is not accurate and it requires elaborate synchronization algorithms to get this time to synchronize, it stands as a good realistic implementation. The message packets received by the server and clients are tuples of type (msg, t) where msg is the message and t is the time-stamp associated with it. The ordering of message packets is:

$$(m_1, t_1) < (m_2, t_2) \text{ if } t_1 < t_2$$

The server receives the message packets and puts them into an ordered queue, based on their time-stamps The server first broadcasts the message that is the least in the ordering. The pseudo-code for the broadcast routine of the server is given below:

```
ServerRcvPacket(pkt) {
  PutPacketInOrderedQueue(pkt);
  if(QSize > MinQSize) {
    // Get the pkt with min time-stamp
    // from the ordered queue
    pkt = GetPktFromQueue();
    if(pkt != null) broadcast(pkt);
  }
}
```

Time-stamping also helps in the synchronization that occurs when a client designer joins a session which is already in progress. The screen shot of the editor and the status of the library manager need to be sent to the new client by the other clients. The new client sends a message ("$sync- < id >''$, t) to the server which is then broadcast to all clients in that session. The clients reply with $(sync - reply, t)$ to the server which then sends it to the new client. Thus, the new client gets replies to its synchronize request from each of the other clients in the same session. The following analysis shows that all of these replies have the same screen shot. So the new client uses the first such message to construct the starting view of its editor.

The server broadcasts from the queue in a critical section of the code and since the channels from client to server are modeled as FIFO channels it maintains that all clients receive the synchronize message in the same relative position in the queue. Also, all the replies are similar because of the consistency constraint we have due to the time-stamping. Hence, the new client can use any one of the replies of the synchronize request and then can start taking actions as usual.

Suppose two clients have sent two different snapshot views to the new client. Without loss of generality, we can say that client1 received at least one message before getting the sync-request from the new client and the client2 received the same message after receiving the sync-request from the new client. However, the action of broadcast is in the critical section in the server and the thread doesn't start broadcasting another message till the present message is broadcast to all clients. This implies that the messages were sent in proper order at the server end and they were rearranged in the channel. This contradicts our assumption that the channels are reliable FIFO channels. Hence, all clients should receive these messages in the same order and the new client should get the same snapshot from all other clients.

Note that the new client might have an event in its queue which was received before the $sync-reply$, but whose action has been taken by the client which sent the reply. In this case, the new client should not take the corresponding action. An example is shown in the Figure 4. The message for synchronization request will be received by the client 1 only after it has received the wireMode and mouseEvt messages. But the sync-reply given by the client1 is given after the corresponding actions on messages 1 and 2 but before message4. So the new client should queue up message4 and take the corresponding action after it takes the action on the $sync-reply$ message.

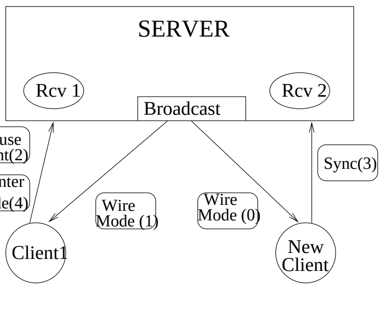

Figure 4: Synchronization of the System

The new client queues up all the events that it received before receiving its synchronization replies. It will take the corresponding action on the events with time-stamps greater than the synchronization-reply message in the start queue, after finishing synchronizing the view. The corresponding pseudo-code is given below:

```
ClientRcvPacket(pkt) {
  if(SessionInProgress) {
    if(pktIsSyncReplyMessage(pkt)) {
      TakeSyncReplyAction(pkt);
      TakeActionOnStartQueue(pkt.timeStamp);
      // Take action on the packets in
      // start queue with larger time-stamp.
    }
    else PutInStartQueue(pkt);
  }
  else TakeAction(pkt);
}
```

5.2 Failure Recovery

Recovery from the failure of any client is another important issue in the collaborative environment. We model the failure of the clients as *stopping* failures rather than *byzantine* failures because we are concerned with the sudden death of clients and have modeled the channels as reliable. If the client program gracefully exits the code, it sends an exit message to the server. The server then deletes all of the resources used for this client. If the client gets killed otherwise, a server vulture which keeps track of the time for which a client has not sent an event, sends a message to that client. If no result is received for that, it assumes that the client is killed and takes the required action. The assumption that the channel is reliable plays a crucial role here.

5.3 Resource Locking

The locking of critical resources is another behavior to be incorporated into the collaborative environment. One such resource is the mouse movements in wire mode. If this is not locked and if more than one client uses the mouse at the same time, they do not get the expected results. Each session has a corresponding wireModeLock in the server and it gets set automatically whenever a client sends a WireModeRequest message when it is not already set. The lock gets released when the same client sets to another mode. The mouse movements from other clients are blocked from being broadcast when the lock is held. The locks are also released when the vulture finds that the lock-holding client got killed.

On the whole, we are trying to simulate a shared-memory system out of an asynchronous network system. We try to achieve this using the high level constraints, the mutual exclusion programming paradigms, and the implementation of some higher level ideas like time-stamping and synchronization.

6 Implementation Details

The programming language used for the implementation is Java (Ver 1.1.6). The Java 1.1 observer-based event model is well suited for our event message passing needs. A snapshot of the tool with two users' editors who are in the same session is given in Figure 5. Multithreading in Java is utilized in making the server run several threads, one for each client to receive messages. The synchronized statement of Java is used for implementing the critical sections of the code for mutual exclusion. The several events that are passed as messages include the mouseEvents, the cellSelect events, the extract events, and all of the UI events on the boxes involved in the editor. The Java serialization mechanism is used after an event object is constructed in order to pass them over the network. All such message objects are derived from a base node, sendObject. The server just broadcasts these objects unless it is of the type ChangeOfMode or a MouseEvent. In that case it does the locking record maintenance and broadcasts the message if the lock is held by the sender and the wireMode is on or rejects the message otherwise. The clients on receiving the message put it in their queue. The message with the least time-stamp is taken up for action. Depending on the type of message, it is delegated to the corresponding collaborative message handler method. The vulture is implemented as a thread in the server which periodically checks on the inactive clients. The tool is available for use at *http://apsara.mit.edu/CollabTop*.

7 Example Collaborative Flow

An example flow of events in the use of the collaborative framework is given in this section. In this example, one designer joins the session when the session is in progress, to demonstrate the synchronization mechanism. The remote tool invoked in the example is Pythia.

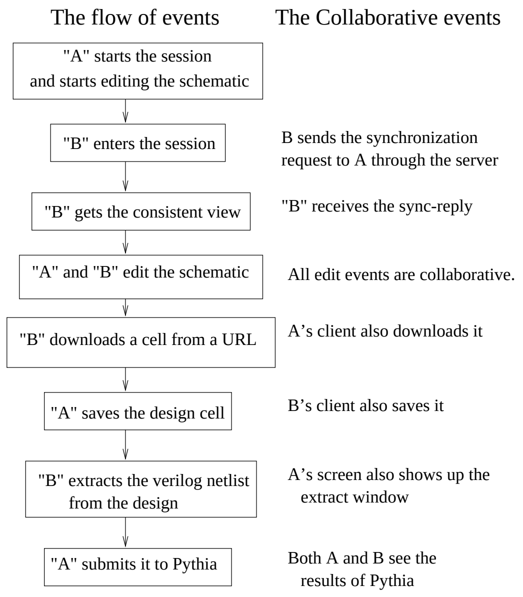
The flow of events
The Collaborative events
"A" starts the session and starts editing the schematic
"B" enters the session
B sends the synchronization request to A through the server
"B" gets the consistent view
"B" receives the sync-reply
"A" and "B" edit the schematic
All edit events are collaborative.
"B" downloads a cell from a URL
A's client also downloads it
"A" saves the design cell
B's client also saves it
"B" extracts the verilog netlist from the design
A's screen also shows up the extract window
"A" submits it to Pythia
Both A and B see the results of Pythia

Figure 6: Example flow in CollabTop

Designer A starts the session and starts creating and editing the schematic in the editor using the library cells in the tool. Designer B joins the session and the synchronization mechanisms bring the consistent view of the design and the saved cells in the library, if any, into B's editor. Then, A and B collaboratively edit the design. In the process, B initiates downloading some WebTop cells from a URL. This generates collaborative events and these cells appear in both A and B's library managers. After editing, a designer saves the cell in the library which generates collaborative events to save it on both sides. Then one designer extracts the verilog netlist from the schematic which makes the extract window with the netlist pop up on their screens. A designer submits it to Pythia and the results can be viewed by both. They then store the design in CellServer and quit the session. This flow is shown in Figure 6.

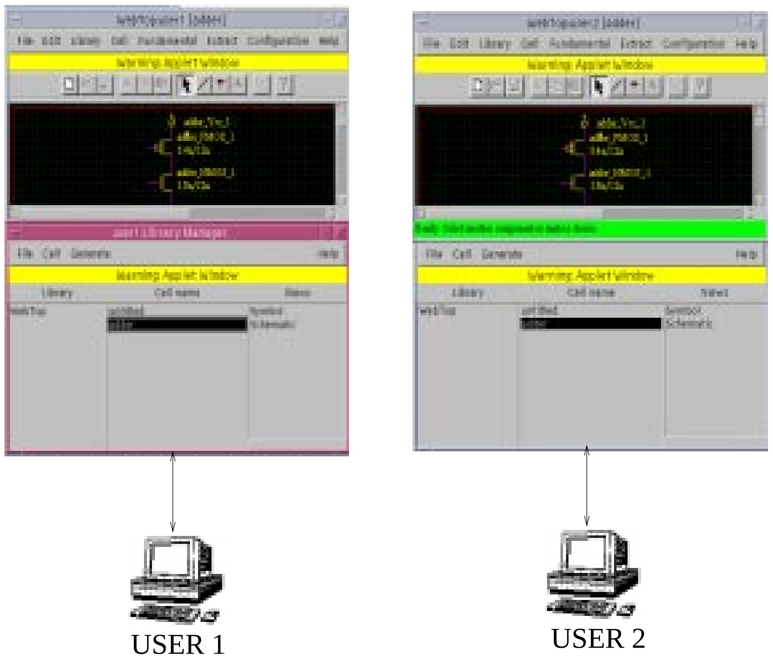

Figure 5: Snapshot of CollabTop

8 Conclusion

A collaborative environment has been developed from a distributed Web-based CAD tool. All of the principles and ideas discussed have been included in the implementation. Future work includes addressing the problem of single point of failure at the server and adding the functionality of client-side caching and saving of the files. For now, since the Java applets can only save at the host of origin, the designers must save the files at the CellServer run at MIT.

Acknowledgments

This work is sponsored by DARPA and an NSF Career Development Award MIP-9501995. The remote demonstration of Section3 was done in collaboration with D. Lidsky and Prof. J. Rabaey of U.C. Berkeley and Prof. T. Meng of Stanford.

References

[1] "What's ahead for design on the Web?", Panel Discussion, *IEEE Spectrum*, September 1998, pp. 53-63.

[2] "IC Design on the World Wide Web", *IEEE Spectrum*, June 1998.

[3] O. Bentz, D. Lidsky, J. M. Rabaey,"Information-based Design Environment", *IEEE VLSI Signal Processing VIII*, pp. 237-246, Nov 1995.

[4] D. Lidsky, J. M. Rabaey, "Early Power Exploration - a World Wide Web Application", *Proc. Design Automation Conf, Las Vegas, NV*, June 1996.

[5] The WELD Project, *http://www-cad.EECS.Berkeley.EDU/Respep/Research/weld*

[6] A. Boglio, L. Benini, G. De Micheli and B. Ricco, "PPP: A Gate-Level Power Estimator - A World Wide Web Application", *Stanford Technical Report No. CSL-TR-96-691, 1996.*

[7] XMX Home Page, *http://www.cs.brown.edu/software*

[8] Xplexer: The Application sharing technology, *http://andru.unx.com/DD/advisor/docs/jun95*

[9] XShare: Workstation conferencing, *http://www.eit.com/software/xshare*

[10] XTV: A Users Guide, *http://www.visc.vt.edu/succeed/xtv.html*

[11] Hemang Lavana, Amit Khetawat, Franc Brglez, Kyzysztof Kozminski, "Executable Workflows: A Paradigm for Collaborative Design on the Internet", *Proceedings of the Design Automation Conference, June 1997.*

[12] Debashis Saha, "Framework for distributed Web-based Microsystem design", *Masters thesis, MIT*, Jan 1998.