Open Distributed EDA Environments on the Web

A. Bogliolo[†] L. Benini[†] D. Guan[*] D. Ku[*] G. De Micheli[†]

† CSL - Stanford University
Stanford, CA 94305-9030

* ESCALADE Corporation
Santa Clara, CA 95054

Abstract

Team design, distributed tool integration, design flow management, resource sharing are challenging needs of the electronic design automation (EDA) area. We propose the concept of open distributed design environment as a unified answer. Designers may access the environment through a network (i.e., the Internet) and use local and remote resources as part of the same design flow. All details of resource retrieval, data transfer and communication protocols are hidden to the user. New resources (tools, libraries, design data) may be transparently added to the environment at any time, thus becoming automatically accessible to any user. We describe the dynamic resource encapsulation mechanism that grants these features to the EDA environment. We then propose a Web-based implementation that allows designers to access the environment using standard Web browsers.

1 Introduction

The number of EDA tools involved in the design flow of a complex VLSI system is ever increasing. These tools are usually provided by many different suppliers, with their own user interfaces and data representations. Nevertheless, to make the design process more productive, designers should work in a unified EDA environment in which all tools are accessed through a common interface and may work on the same design. Several attempts have been made over the years to integrate EDA tools into unified frameworks. The definition of standard formats for design descriptions such as VHDL and EDIF has been an important milestone in this direction [1, 2]. However, tools provided by different vendors still lack uniformity and compatibility.

Moreover, as the complexity of design tasks increases, resource sharing and team design become critical issues as well. CPU-extensive EDA tools may be installed on high-performance servers accessed by several designers, possibly (but not necessarily) working on the same project. Thanks to the pervasive diffusion of computer networking, both tools and design teams may be geographically dispersed and connected through a network (in particular, the Internet). Reliable and effective communication paradigms are then required.

On the other hand, EDA tools have a relatively short life-time: they become obsolete in a few years and new versions are released every few months. As a consequence, user interfaces should have short learning curves and design frameworks should provide efficient and straightforward mechanisms to update/replace the tools they contain.

All the above observations prompt for moving from traditional EDA frameworks to an open, distributed environment, where designers can use local and remote resources (tools, design data, libraries) as part of complex design flows. Users may access resources by requesting specific services (regardless of how they are provided and where they are located) and paying for them on a usage basis. In the open environment, resources can be dynamically and transparently added or updated to improve capabilities and performances without affecting the user front-end and the design flow.

1.1 Toward an open, distributed design environment

The most critical step toward an open, distributed EDA environment consists of defining an effective mechanism for the *dynamic encapsulation of distributed resources.*

A successful mechanism for dynamic encapsulation has been behind the explosive growth of the World Wide Web (WWW) [3]. Web browsers provide a uniform interface to any kind of information distributed all over the world, while *uniform resource locators* (URLs) hide all the details of information retrieval [4]. New data can be added at any time becoming automatically accessible to everyone.

Several attempts have been made to exploit the features of the WWW in the CAD-EDA area. In particular, Web-based interfaces to CAD tools have been proposed in [5, 6, 7]. All these works are mainly focused on an *information-centric* perspective, that reflects the original purposes of the WWW. In the information-centric perspective, the designer's need for geographically disperse and heterogeneous information is addressed by means of distributed *data encapsulation*. The designer can retrieve background material (i.e. algorithms, bibliography, benchmarking information, etc.) and design libraries simply by activating hyperlinks on a WWW page. Multiple data formats can be transferred and viewed without the need of complex interactions. Security issues are addressed and privacy is guaranteed by encrypted transactions. The complexity of the communication among remote sites is hidden, and the exchange of data among users becomes straightforward.

However, the end-user of CAD tools not only necessitates to access information and data, but he/she mainly needs to process and modify data by executing the available programs. Simple protocols for remote batch execution have been described in [5, 6, 7]. To address more directly the needs of EDA users for distributed *resource encapsulation*, an *application-centric* perspective was adopted in [8]. Execution paradigms were proposed to provide an effective interface not only for information retrieval and remote job launching, but also for interactive execution control. As a prototype, a highly interactive Web-based interface was developed for PPP, an integrated environment for low-power design and simulation [9].

PPP contains several tools distributed on a *local area network* (LAN). Designers may access PPP through the Internet using standard Web browsers, without having any tool installed on their own machines. However, the encapsulation strategy proposed in [8] is *static* and PPP still lacks in flexibility and modularity: adding (or updating) a new tool into the environment is not straightforward and no interactions are allowed with other tools across the Internet.

A systematic approach to the *dynamic* encapsulation of distributed applications has been followed in the area of object oriented programming languages. In particular, the Object Management Group (OMG) defined a Common Object Request Broker Architecture (CORBA) [10] in which heterogeneous and dispersed applications may communicate with one another, no matter where they are located or who designed them. New applications can be made available at any time by updating a common repository, while existing applications can be transparently updated without changing their interfaces.

In this paper, we follow the same principle outlined above to propose a paradigm for dynamic encapsulation of distributed EDA resources. The overall architecture of an open, distributed design environment and the main encapsulation mechanisms are described in the next section. A Web-based implementation is proposed in Section 3.

2 Dynamic encapsulation of distributed EDA resources

We refer to a fully distributed environment in which designers, tools and data may be located everywhere on the network. Different tools with different features are available to perform design tasks. Designers have access to all these tools, and many designers may work simultaneously on projects that may be either independent or cooperative.

The key features we want to achieve are uniformity and modularity. Designers issue requests and tools provide services. In a uniform environment, designers do not need to know where data and tools are located, how tools communicate and how they are implemented. On the other hand, tools must be able to provide the same functionalities no matter where the user is located.

The environment can be dynamically and transparently updated, thus becoming potentially unlimited. New users and tools may be added at any time. Whenever a new user becomes part of the

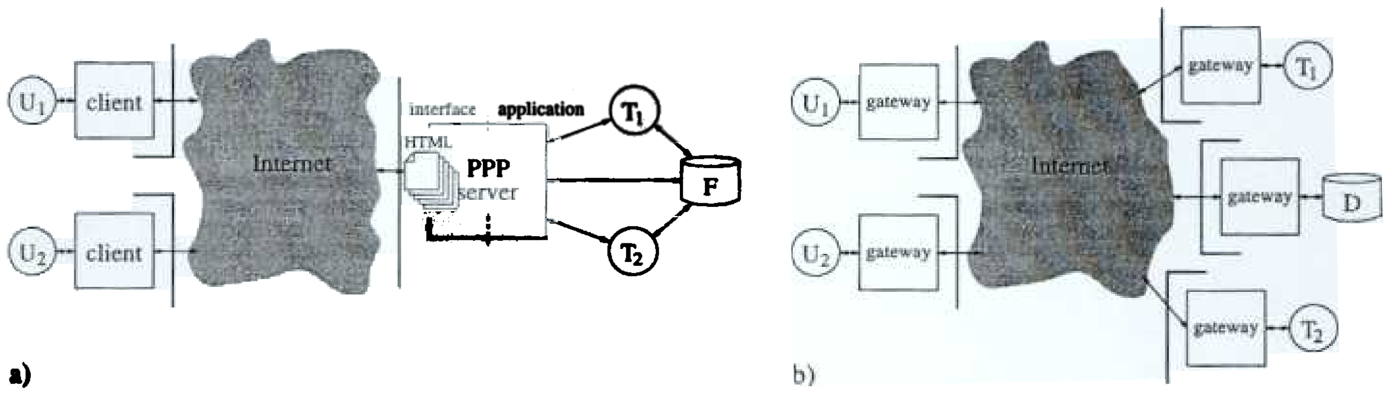

Figure 1: Comparison between a) the asymmetric client-server architecture of PPP and b) a generic distributed architecture in which users (U), tools (T) and databases (D) use symmetric gateways to interface to the network.

design environment, he/she gets access to all tools available on it. Whenever a new tool is added, it becomes automatically available to all designers.

Uniformity and modularity can be achieved in two main steps:

- making the tool interface independent from the implementation,
- making the whole architecture symmetric.

The solution proposed in [8] met the first requirement by means of a uniform Web-based interface, but it was completely asymmetric (see Fig. 1.a): the server of PPP was centralized, and ad-hoc mechanisms (based on a shared file system) were used to address tool-communication issues. As a consequence, adding a new tool into PPP was not straightforward and interoperability across the Internet was not supported. Moreover, the user had to do explicit file transfer to put his/her own design data on the PPP's file system.

A schematic representation of a symmetric distributed architecture is shown in Fig. 1.b. Any *agent* (user (U), tool (T), database (D)) needs a *gateway* to access the EDA environment, that is nothing but a set of agents using a common, well defined protocol to communicate to each other. Gateways translate between the standardized protocol and the agent-specific interfaces. In this way they wrap around tools and data, hiding all the details of data transfers, remote execution protocols and tool implementations.

Each user interfaces with the local gateway as with a unique tool providing many different services and working on a virtual file system. Each tool interfaces with its own gateway as with local users. The hidden mechanism in between retrieves resources and data, transfers files, delivers service requests and returns outputs back.

In principle, this is all we need to grant dynamic encapsulation of distributed resources: adding an agent to the environment is a local task that entails only creating a new gateway. Once a new resource has its own gateway to the common protocol, it becomes automatically accessible to any user.

2.1 Communication protocol

The design process can be viewed as a sequence of steps (tasks) involving interactions between pairs of agents: a *client* agent that requests a service and a *server* agent that provides it. In most of the design steps, the agent acting as a client is the designer. However, according to the symmetric scheme of Fig. 1.b, any agent may be viewed either as a client or as a server in the context of different tasks.

We refer to a generic situation in which an agent A requests a service to an agent B. Let us consider A and B as a designer and a tool, respectively. In a distributed environment, A and B may be on different machines and a communication channel is to be opened between them to allow A to provide the input data, control the execution of B and have the results back. We assume a standard hand-shaking protocol (providing *request* and *grant* primitives) to be used to create the channel. To ensure full connectivity, any agent may issue requests and grant connections.

Notice that, when A requests a connection to B, it knows which service B provides but it does not know how to interface to this service. However, since the hand-shaking primitives provide a standard way of creating a connection between A and B, the communication channel can be used to send to the client the whole interface to the service provided by B. This can be done either by using standard message passing protocols, or by exploiting the characteristics of languages designed for distributed execution on the Internet (such as JAVA [11]). In this case, the interface is sent as a machine independent executable program that may run on A while controlling B.

The entire communication mechanism is made transparent by the gateways. Referring to Fig. 2.a, we call A_g and B_g the gateways of A and B respectively. A_g makes the tools of the environment visible to A as if they were on the local machine. When A asks A_g to run tool B, A_g requests a connection to the corresponding gateway (B_g). If the resource is available, B_g launches B and sends to A_g a grant message containing the interfacing guidelines. A_g interprets the message and follows the guidelines to create a local interface to the remote tool (possibly using standard helpers). At this point A may use the interface to get direct control of the remote tool as if it were local (performance issues will be discussed later).

A message representation protocol is necessary to allow A_g to interpret the interface guidelines provided by B_g. To keep generality, we assume that the grant message starts with a standard header describing the message body and the helpers that can be used for interpretation. A message protocol following this principle is, for instance, the *multipurpose internet mail extension* (MIME) [12].

Both licensing and security issues can be transparently addressed. The request message sent by A_g to B_g may contain standard information allowing B to identify the client and check for its permissions, while encryption can be used to grant security.

Files containing design data and results are also to be transferred between A and B. However, we don't need to define a unique file transfer protocol. Once the connection is established, data transfers can be decided in accordance by the two agents and controlled either by the server or by client (through the tool interface). Nevertheless, we need to define what's the workspace where files can be stored and retrieved

2.2 Data management

To achieve data location transparency, we use the *database* abstraction to make file-systems accessible from the EDA environment. A database is nothing but an agent (with its own gateway) that provides an interface to a set of data (see Fig. 2.a).

This abstraction provides a general way of publishing design data and libraries, possibly allowing team design by means of *revision control system* (RCS) facilities. However, any agent also needs to access its local file-system to store temporary data.

We assume that every agent (user, tool or database) has a *local workspace* (represented by dashed cylinders in Fig. 2.a) that contains the *current design data*. When a user loads a design from a database, the corresponding files are stored in his/her local workspace. In the same way, if a remote tool has to run on the current design, (some of) the design files need to be transferred to the tool's workspace.

Notice that the local workspace is *not* the same as the local file system, although it may be implemented as a directory tree on the file system. The separation between the user's file system and the local workspace is strictly enforced. Access to the workspace is only allowed through the access primitives provided within the EDA architecture and implemented by the local gateway. If the users wants to obtain a copy of the design files, he/she needs to transfer them from the local workspace to a database agent representing his/her local file system. Security reasons are behind this architectural choice: direct interaction with the local workspace from outside the distributed environment may compromise the data integrity and the design correctness.

Each gateway can manage the local workspace according to the needs of the corresponding agent. For instance, if a tool allows multiple access, the local workspace should be partitioned to provide an independent space to each job. On the other hand, a designer may want to launch more than one tool simultaneously. In this case, check-out and check-in primitives can be implemented by the local gateway to grant consistence to the design files simultaneously used by different tools. Caching strategies

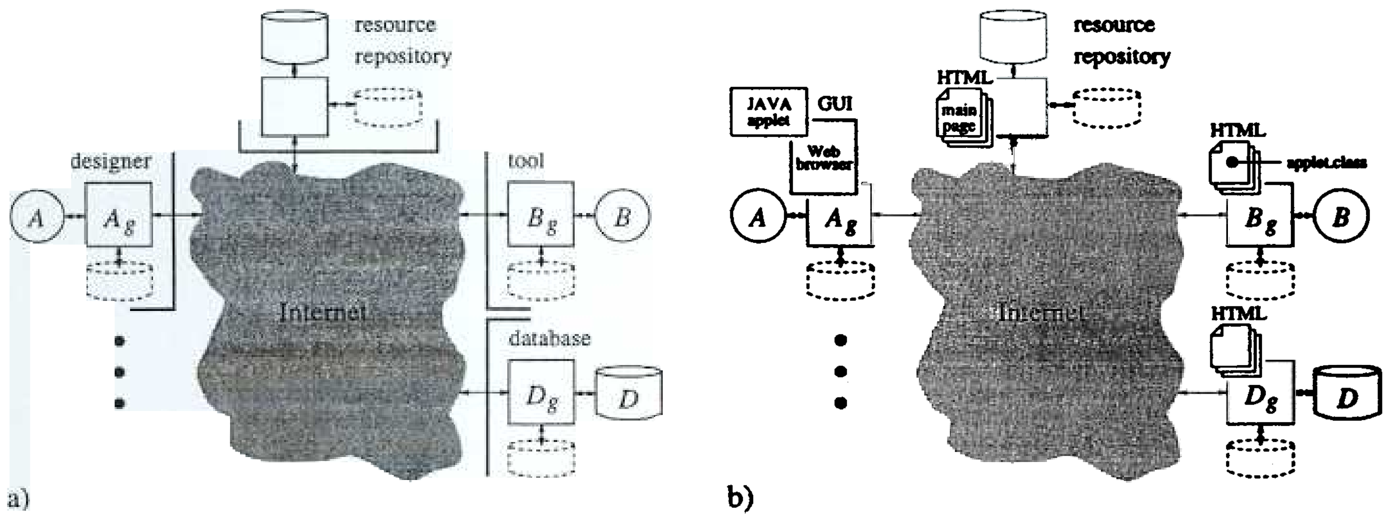

Figure 2: Schematic representation of a) an open, distributed EDA environment, and b) a possible Web-based implementation.

may also be implemented to improve performances.

2.3 Optimizing performance

The performance of Internet connection is widely variable depending on the degree of congestion. As an ever increasing number of users access the Internet with high-bandwidth requests, the performance of the links between users, tools and design databases may fluctuate from acceptable to extremely slow.

While for long batch runs with low interactivity the speed of the link between user and server may not be an issue (the network latency is hidden by the length of the batch run), this is not the case for highly interactive applications. Moreover, even long batch runs benefit from fast access to files used in input and output. We propose a cache-based solution that takes advantage of the presence of local workspaces associated with each agent. Access to the local workspace is fast because it is physically located on local resources (disk, main memory).

When the user accesses a design database and checks out a design, no file transfer is initiated. The user receives in his/her workspace only content information about current design data. When he/she specifies part of the design as target of tool runs, the corresponding files are automatically transferred to his/her local workspace and forwarded to the tool's local workspace (possibly using a different Internet link). A similar mechanism is used for the outputs of tools runs: initially only directory information is transferred to the user, while the actual results are transferred to the user's local workspace (and possibly to design databases or tools used to perform subsequent design tasks) only upon request.

This *lazy transfer* mechanism exploits the well-known principle of locality of access. Data is transferred when needed, although the directory information is completely available. More advanced caching schemes can be envisioned. One interesting alternative is to pre-fetch not-yet-requested data during the idle user time.

2.4 Resource repository

The mechanisms described so far allow transparent interactions between remote agents that do not necessarily know each other in advance. The last thing we need is to make agents visible and reachable within the environment.

To this purpose, each agent may be registered under a unique name into a common *resource repository*. The repository is a database that associates with the agent's name its location, its features and its permissions. When a designer access the environment, his/her gateway automatically connects to the repository to construct a local image of the environment, containing all the available resources. In particular, resource names and features are shown to the user, while resource locations are used by the gateway to ask connections upon user's requests. Global licensing policies can also be conceived. For instance, a license key can be associated with each agent. When it accesses the repository, its license key can be used as a filter to hide resources it cannot access.

Notice that the resource repository is a resource itself. However, its location needs to be known in advance by all the agents' gateways. This is the only asymmetry in the whole architecture of Fig. 2.a. The presence of a center node such as the repository to which all users must refer to access tools may rise concerns about the scalability of our architecture. We do not expect that traffic from/to the repository will become a severe limitation, because the connections with the repository have very low bandwidth requirements. The amount of data transferred is small: it consists only of access information for the required tools.

Fully symmetric solutions can also be adopted, such as those developed in the area of distributed data bases [13].

3 A Web-based implementation

The open EDA environment we have described is completely general. We have mainly focused on a dynamic tool integration paradigm and analyzed the necessary conditions to make it working. Many different protocols can be specified and different mechanisms conceived to meet these requirements. However, most of the features we have described are similar to those of the World Wide Web. Hence, we propose in this section a Web-based implementation for the architecture of Fig. 2.a. We use HTTP servers as gateways and we associate each agent with a URL. In addition, we use Web browsers as standard front-ends to access the environment.

The interface to the resource repository is a (tree of) HTML pages containing hyperlinks to the available resources [14]. Links can be classified and organized to make it easier to find data and services, and the HTML pages can be dynamically generated by scripts (based on the Common Gateway Interface, CGI [15]) in order to show to each user only the links he/she is allowed to follow. To connect to the environment, a designer uses a Web browser to access the URL of the resource repository. The corresponding HTML page can be conceived as the main-page of the EDA environment. From this page the designer may access design-data, run remote tools, contact other designers. This is always done by clicking the corresponding links.

Referring to Fig. 2.b, consider designer A accessing the environment. The entry point can be a local page (provided by A_g) containing a link to the URL of the resource repository. Following this link, user identification data are automatically posted by A_g to the HTTP server of the resource repository. If agent A is a registered user, the resource repository generates the HTML page containing the hyperlinks to all the resources A can access (including, for example, a design database D and a design tool B). From the user stand point, this page looks like the main page of a local EDA framework: resources can be represented by their functionality, or just be associated with icons to be clicked to access the corresponding resources.

Suppose that A opens a project p stored in D, and runs B to synthesize part of the design specified in p. When A clicks the hyperlink of D, a request is implicitly sent to D_g, that checks for permissions and provides the HTML pages that allow A to browse (part of) the contents of the database. The first of this pages can be viewed as the grant message sent by D_g to A_g. A_g interprets the message and shows the interface up. Actually, this is automatically done by the Web browser.

When A selects project p from the database, the corresponding files (or at least their directory information, if caching mechanisms are implemented) are to be transparently checked-out and transferred from D to the local workspace of A. However, the workspace of A is hidden to the Web browser. Files cannot be transferred through the communication channel opened between the browser and the HTTP server that implements the gateway of D. On the other hand, using standard File Transfer Protocol (FTP) will violate both transparency and security requirements. What we need is a new connection between A_g and D_g.

We assume that each gateway has a demon waiting on a given port for file transfer requests. The preexisting Internet connection can be used to provide D_g with all information required to connect to the A_g's file transfer demon: IP address, port number, file transfer protocol and password to be used. Upon user's selection, D_g requests a connection to the A_g's demon that makes the local workspace accessible for file transfers. Notice that, if caching mechanisms are implemented, the file-transfer connection will not be closed when A exits from D. The caching mechanism will be transparently managed by A_g and D_g, that will close the connection only when all the data have been transferred.

At this point of the user session, p has become the current project. A goes back to the main page (*i.e.*, to the resource repository) and clicks the icon associated with B to access the synthesis tool. The double hand-shaking procedure is then repeated to establish the connection between A and B: A_g sends to B_g a connection request containing the information about its file-transfer demon, B_g checks permissions and grants the connection by sending its Web-based interface. At the same time, B_g requests connection to the file transfer demon of A and automatically downloads the directory information about the current design. This grants consistence between the local workspaces of A and B and allows the user to work on B as on a local tool. In particular, the current design can be accessed through the Web-based interface of B. The user can then select part of the design to be the target for the tool run.

The double connection between two agents, closely resembles that defined in the standard file transfer protocol. A *message channel* is first created upon client's request. A *data channel* is then opened and controlled by the server to allow the client to upload and download files. In our architecture, the message channel is replaced by a highly interactive and flexible user interface, while the data channel is hidden to the user to make file transfers transparent.

So far we implicitly referred to tool interfaces implemented as HTML pages dynamically generated by the tool's gateway (as those used in PPP). HTML forms are used to issue commands on the remote server, while nothing is processed locally. In general, this provides an effective way of controlling tasks dominated by the execution time of batch processes (*i.e.*, optimization, simulation), but becomes inefficient when dealing with tools that entail frequent graphic interactions (*i.e.*, waveform displays or schematic editors).

In these cases, we can exploit the key features of languages designed for distributed execution on the Internet, such as JAVA. The Internet connection can be used not only to transfer data, but also to transfer executable programs in a platform independent fashion. Agent A can then receive a program (*e.g.*, a JAVA *applet*) implementing a full *graphic user interface* (GUI) to B. The interface is transferred once for all upon connection and runs locally, improving efficiency. Interactions through the network are involved only to launch remote jobs on B. In practice, the use of JAVA applets can be viewed as an advanced caching mechanism.

To further improve performance, local applications are to be used whenever available.

4 An experimental application: low-power design

Realizing a working prototype of a Web-based open EAD environment is the target of our current work. As a starting point, we have restricted the scope of our project to the area of CAD for low-power, where most of our research efforts are concentrated. We are following a three step process consisting of: *i*) definition of specifications, *ii*) feasibility study and *iii*) prototype implementation. This paper is based on the results of the first two steps.

Two sets of experiments were performed to verify feasibility. First, we used PPP to experiment remote execution strategies and to analyze the features of Web-based interfaces [8]. Simulation, synthesis and optimization tools for low-power CMOS circuits are integrated in PPP. A common Web-based GUI hides the boundaries among different tools and makes them accessible through the Internet. Both batch and interactive remote execution protocol are implemented. PPP is available on our Web server for direct evaluation (*http://akebono.stanford.edu/users/PPP*). Second, we developed a distributed RCS application to experiment the dynamic encapsulation mechanism based on a common hand-shaking protocol. In particular, we used a JAVA applet as platform independent interface to the database, and JAVA servers as file-transfer demons. Sockets were used to establish connections between them.

The implementation of a full fledged open EDA environment for low power is work in progress. The synthesis and simulation tools embedded in PPP will become the first "agents" in the environment.

5 Conclusions

We have proposed a new paradigm for the dynamic integration of EDA tools in an open, distributed environment. The architecture of the environment is symmetric and completely decentralized. Any agent (designer, tool or database) uses a local gateway to become part of the environment. A simple hand-shaking protocol provides a standard way of establishing connections between gateways of different agents. Gateways wrap around agents and hide all details of resource retrieval, remote connection and data transfer.

Each agent may act either as a server or as a client in the context of a given transaction. The Internet connection is used by the server to send to the client all information necessary to interface to the service it provides. This grants complete modularity to the environment: the client agent doesn't need to know in advance how to interface to the server. New tools can be added at any time becoming automatically available to any user without affecting the preexisting ones.

The agent abstraction makes the environment open and flexible. For instance, no restrictions on data format are necessary, since agents can be added to perform any kind of data conversions. Flow-manager agents may be also conceived: instead of containing tools or databases, flow-manager agents steer a task-specific design flow possibly involving many other agents. Designers are agents as well. They may not only access resources, but also provide services. For instance, they may provide information to other designers either by publishing design data or by accepting talk-like connections. Any kind of licensing policies and security protections may also be implemented.

We proposed a Web-based implementation for the dynamic encapsulation mechanisms. Preliminary experiments have been performed that demonstrate the viability of the approach. Our current work is mainly focused on realizing a working prototype of the open, distributed EDA environment.

Acknowledgements

This work was partially supported by NSF (under contract MIP-9421129), by AEI (under grant De Castro) and by a Stanford CIS grant.

References

[1] T. J. Barnes *et al.*, *Electronic CAD frameworks*. Kluwer Academic Publishers, 1992.

[2] R. Camposano and W. Wolf, *Trends in High-Level Synthesis*. Kluwer Academic Publishers, 1991.

[3] T. Berners-Lee *et al.*, "The world-wide web," *Communications of the ACM*, vol. 37, no. 8, pp. 76 - 82, 1994.

[4] Network Working Group, "Uniform Resource Locators (URL)," *RFC 1738*, `http://www.w3.-org/pub/WWW/Addressing/rfc1738.txt`, December 1994.

[5] M. J. Silva and R. H. Katz, "The case for design using the World Wide Web," in *Proc. of Design Automation Conf.*, pp. 579–585, 1995.

[6] P. G. Ploger, J. Wilberg, M. Langevin, and R. Camposano, "WWW Based structuring of codesigns," in *Proc. of Int.l Symposium on System Synthesis*, pp. 138–143, 1995.

[7] D. Lindsky and J. M. Rabaey, "Early Power Exploration - A World Wide Web Application," in *Proc. of Design Automation Conf.*, pp. 27 - 32, 1996.

[8] L. Benini, A. Bogliolo, and G. De Micheli, "Distributed EDA tool integration: the PPP paradigm," *to appear in Proceedings of ICCD*, 1996.

[9] A. Bogliolo, L. Benini, and B. Riccò, "Power Estimation of Cell-Based CMOS Circuits," in *Proc. of Design Automation Conf.*, pp. 433 - 438, 1996.

[10] S. Vinoski, "Distributed Object Computing with CORBA," *C++ Report Magazine*, July/August 1993.

[11] Sun Microsystems, "The JAVA language environment: a white paper," `http://java.sun.com/-whitePaper/java-whitepaper-1.html`.

[12] "MIME (Multipurpose Internet Mail Extensions)," *RFC 1521/1522*, `http://www.oac.uci.-edu/indiv/ehood/MIME/MIME.html`.

[13] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*. Benjamin/Cummings, 1994.

[14] Network Working Group, "HyperText Markup Language (HTML)," *Working and background materials*, `http://www.w3.org/hypertext/WWW/-MarkUp/MarkUp.html`.

[15] "The Common Gateway Interface," *Working and background materials*, `http://hoohoo.ncsa.-uiuc.edu/cgi`.