Distributed EDA tool integration: the PPP paradigm

L. Benini A. Bogliolo G. De Micheli
Computer Systems Laboratory
Stanford University, Stanford CA 94305 - 9030

Abstract

In this paper we describe a paradigm for integrating EDA tools running on distributed platforms under a common user interface. We focus on interactive remote execution more than on simple information retrieval. Minimum support is required on the user side to access the features offered by a large and diverse set of tools: the user's WWW browser is the interface for all interactions. Integrating new tools in the environment is a straightforward process that does not require any effort from the end users. The learning curve is extremely short because the well-known user interface provided by the WWW browser is exploited. PPP is a prototype we implemented to show the feasibility of WWW-based tool integration. We describe the features of PPP, its architecture and its implementation.

1 Introduction

Computer-aided design tools have become an essential part in the design flow of any complex VLSI system. At the same time, design teams are realizing that the number of tools needed to implement complex systems is ever increasing, and such tools are generally provided by many different suppliers. Integration of EDA tools into unified frameworks prompts for an increasing effort in the creation of standard interfaces [1, 2]. The definition of standard formats for design descriptions such as VHDL and EDIF has been an important milestone in this direction. However, the user interfaces provided by different EDA vendors still lack in uniformity and compatibility.

Moreover, as computer networking becomes pervasive, design teams will be geographically dispersed, and the need for reliable and straightforward communication paradigms will steadily grow. Computationally demanding tasks (optimization and/or simulation of large circuits) could be performed on high-performance servers remotely connected to the designers by Internet links. Again, a uniform user interface should be provided in this concurrent and distributed engineering environment.

The opportunities for commercial EDA vendors are promising as well. New tool usage paradigms may emerge: users could temporarily connect to a tool provider, perform a specific task and be billed on a usage-time basis. Similarly, demos can be performed across the network, a much more effective form of advertisement because it gives the perspective user direct hands-on experience. Necessary conditions for such new paradigms to be successful is again the existence of a general and friendly user interface for remote connections and execution control.

The explosive diffusion of an user-friendly and powerful interface such as the World Wide Web [5] (WWW) has prompted several attempts directed to exploit its features in the CAD-EDA area. Virtually every computer user is familiar with the WWW, and Web browsers provide a uniform interface to a number of different communication protocols. A Web-based interface to CAD tools is an important step toward standardization and distributed tool integration. Preliminary results in this direction have been reported in [3] and [4]. Both these works are focused on an *information-centric perspective*, that reflects the original purposes of the WWW development [5].

In the information-centric perspective, the designer's need for geographically disperse and heterogeneous information is addressed. The designer can retrieve background material (i.e. algorithms, bibliography, benchmarking information, etc.) and design libraries simply by activating a hyperlink 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.

In this work we extend the ideas proposed in [3, 4] by adopting an *application-centric perspective*, where the needs of EDA tool users are more directly addressed. The end-user of CAD tools not only necessitates to access information and data, but mainly he/she needs to process and modify the data, by executing the available CAD programs. We mainly focus on *execution* paradigms, in an effort to provide an effective interface not only for information retrieval

1063-6404/96 $5.00 © 1996 IEEE

and exchange but also for interactive execution control.

Simple protocols for remote execution have been described in [3] and [4]. Both approaches focused on batch execution: the user can request services to remote tools, that will perform the required tasks and return the results. The interaction between user and tools is sparse and loose. Effective execution of remote tools requires a much higher level of interaction: the user should be allowed to change the setting of a parameterized tool run, preview partial result, access visual information like waveforms and network schematics, modify networks and update design databases.

We have developed PPP, an integrated simulation and synthesis environment with a modular and highly interactive Web-based interface. In this paper we describe the architecture of PPP, its main features and the innovative aspects of its implementation. The main target of PPP is the synthesis and the simulation of low-power CMOS circuits [6, 7], but its modularity allows us to interface with general-purpose commercial and academic tools. PPP is work in progress, therefore we will describe a paradigm for distributed tool integration, more than a product.

The rest of the paper is organized as follows. In the next section we will give an overview of the key concepts and goals that lead to the design of PPP. In the third section the architecture of PPP is described in detail. The fourth section contains an example of a synthesis and simulation session on PPP. Starting from the example we will illustrate some of the features of our tool and relevant aspects of its implementation. Finally, in section five we will discuss the future developments of PPP.

2 An overview

Primary target of the PPP project is to develop a fully integrated synthesis and simulation environment with short learning curve and architecture-independent Web-based interface. PPP is a modular system composed of interacting tools that may run on different machines and be provided by different vendors or contributors. The user accesses PPP using his/her own Web browser, with no additional requirements on hardware or software installation. The tools can reside on remote servers or on the local users' machine. This is fully transparent to the user. The graphical user interface (GUI) of PPP is exactly the same used for Web navigation and no additional effort needs to be spent in familiarizing with a new GUI when the user first accesses the system.

In the design of PPP, we focused on two key ideas: plug-and-play modularity and architecture-independent interactive remote execution. Plug-and-play modularity allows new tools to be integrated in PPP with little effort and no modifications. The impact on other tools already embedded in the environment is negligible. Remote execution can be seen as an advanced caching strategy. Upon invocation, a resource

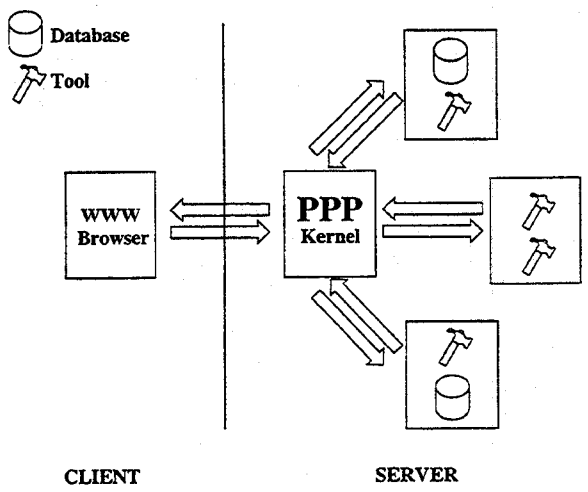

Figure 1. User interaction with PPP

in PPP can maintain its state throughout a sequence of interactions. The state information is not necessary saved on files to avoid the penalty of iterated disk accesses (or, worse, of sending data across the Internet), but it is kept in memory because the resource does not terminate its execution after every interaction with the user.

From the user's point of view, PPP appears as a remote HTTP server and each user can access it by simply inserting a pointer to the PPP URL in his/her bookmark file. This is the only setup operation required. Figure 1 depicts the user interaction with PPP. The rectangles represent different machines connected to the Internet. PPP is the core of a distributed architecture. It manages two different kinds of interface: connection to the client and connection to the resources (tools and databases). The arrows represent Internet protocols (FTP, HTTP, mail, etc).

Multiple users are allowed to concurrently access PPP, similarly to a traditional WWW server. Figure 1 depicts a *centralized* structure. The PPP *kernel* manages all interactions between the users and the tools. In the next section we will describe the organization of the PPP kernel and the overall architecture.

3 Architecture

PPP is build upon a layered architecture, following a well-known software engineering paradigm. The three layers are: *application, communication* and *interface*. The organization of PPP is shown in Figure 2. The top two layers are embedded in the kernel, while the application layer is distributed. Application is at the bottom of the hierarchy and manages the interaction with the tools and the information sources.

3.1. Application

The synthesis and simulation tools embedded in PPP represent the bulk of the application layer and are called *resources*. No constraint is posed on the target architecture and

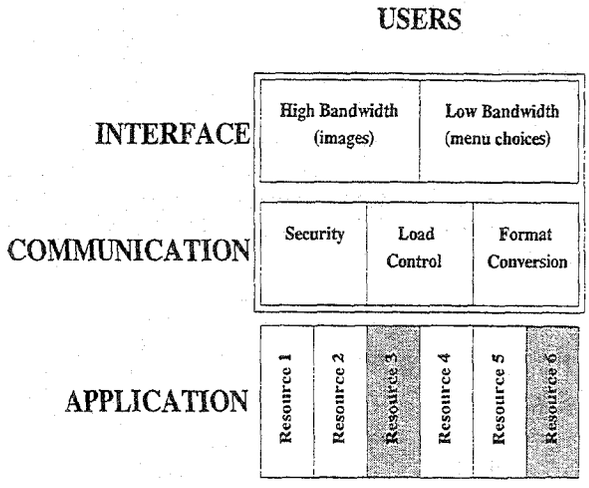

Figure 2. The layered architecture of PPP. Resources represented with shaded rectangles require an high degree of interactivity.

on the programming language used for the implementation of the resources. The resources may run on different machines. Two level of interaction between resources and the above layers are possible. The simplest interaction is *stateless connection*: the tool performs some manipulation on the input data, produces an output and terminates the execution. Information about previous invocations may be stored in files. Unfortunately, retrieving the state from files imposes heavy performance penalties when highly interactive operation is required (for example, during an user-controlled optimization session). Stateless connection resembles the batch execution features described in [3, 4].

For applications where higher interactivity is required, we provide a novel paradigm called *interactive remote execution*. Once started, the resource does not terminate its execution after every command, but it simply waits for another command to be issued. The resource becomes a *server* that awaits for service requests from the client (the user) keeping complete information about past requests and their results. Resources that provide interactive remote execution must satisfy some requirements on their external interface. In particular, it must be possible to interactively receive data from another process without terminating the execution. Interactive remote execution is required for running optimization and validation tools that perform user-assisted tasks.

Example 1 *An application requiring interactive remote execution is logic optimization, where the user can decide among different optimization strategies by observing the quality of intermediate results. In contrast, a long simulation is a typical example of batch execution. After setting up the simulation parameters, the user dispatches the run and does not need to interact with the tool until the run terminates.*

3.2. Communication

The communication layer represents the core of the PPP architecture. Its purposes are the following.

- Process users' requests and translate them in a format that is accepted by the resources.
- Collect the results produced by the resources and provide them to the users in a readable/downloadable format.
- Manage data files provided by the users and/or created by the resources and store them where retrieval will be faster.
- Provide security. Different users work on private space and should not be allowed to interact unless they explicitly require to work on a shared design. Access identification should be enforced.
- Manage server's resources. If PPP runs on multiple machines, the communication layer can direct the requests of the users toward unloaded machines.

All functionalities listed above are in some degree implemented in PPP. Each user works in a separate space, where input files and results are stored. User access authentication is enforced by a password mechanism. All format conversions are performed transparently. Since the resources run on different machines, the communication layer must be implemented as a distributed multiprocess application. In the current implementation, scheduling of resources, conflicts and mutual exclusion are managed using simple message files on a shared file system.

Example 2 *Format conversion is a typical function performed in the communication layer. For instance, in PPP part of the synthesis process is performed by SIS [11]. The output format provided by SIS is BLIF. The simulation tool embedded in PPP is a customized version of Verilog-XL [7], that accepts files in Verilog HDL. The conversion between BLIF and Verilog is performed by a process in the communication layer. The conversion is completely transparent to the user.*

3.3. Interface

Interface is the uppermost layer, with which the user interacts. The largest part of the interface layer is directly supported by the WEB browser. If the interaction is limited to simple menu-based communication, the form feature provided by HTML [9] is sufficient. The same holds for small images and graphs. This is however not sufficient in general. In many cases, the users wants to specify the input or examine the output in more complex ways.

We call *high-bandwidth* the kind of interaction for which the WEB browser does not provide satisfying performance. Examples of high-bandwidth interactions are transfers of large input files (i.e. the networks that the user wants to simulate/optimize) and interactive graphical output (such as waveform or network browsing). In these cases alternate mechanisms for interaction must be provided. For user-originated communication such as the transfer of input files, we currently use the FTP protocol.

For resource-originated communication, such as waveform display, more advanced mechanisms are needed. The resource generates output data files, the communication layer then takes care of transforming the data in a graphical output that is sent to the user. The user can view the graphical output using helper programs or directly through the limited image display capability of the WEB browser.

Example 3 *After simulation, the users specifies the signals he/she wants to view. This information is communicated to PPP (this is a typical low-bandwidth interaction) using a form. From the simulation output files, the relevant signal transitions are extracted and a waveform image is generated to be sent back to the user. This is a high bandwidth interaction.*

3.4. Bandwidth management

A fundamental issue in the design of all layers of PPP is *bandwidth management*. Although local area networks (LAN) can usually provide the bandwidth required for most data transfers, PPP is not limited to run on LANs. When some of the connections are supported by a wide area network (WAN), the bandwidth of the data transfer has to be reduced as much as possible. In the current implementation, the only critical connection is between the PPP kernel and the client. On this connection, bandwidth-critical applications are those involving massive data transfer and a high degree of interactivity. In this case, the level of interaction with the resource is so high that we may want to transfer the resource itself on the client's side.

An example of critical application is waveform display. In the current implementation of PPP, we try to minimize the bandwidth required by sending to the client compressed images of the waveforms. This solution is satisfying only if the user does not need to dynamically update the waveform view very frequently. The advantage of this approach is that the user does not need any dedicated support for waveform display. More aggressive bandwidth reduction can be achieved without requiring the user to install dedicated programs. To this purpose, it is necessary to enable the transmission of executable code across the network. Languages such as JAVA [8] and TCL [10] allow this kind of interaction.

4 Functionality

To give a more concrete understanding of the features provided by PPP, we now describe a simulation session.

User identification is enforced to access PPP. The access point is an HTML form asking for user's name, password and e-mail address. When the form is applied, the main menu of PPP appears and two directories are created on the server file-system to allow the user to work in a private space: an I/O directory that can be accessed from the client for explicit file transfers based on FTP, and a (hidden) work directory used by the system to store intermediate results.

Several interacting tools can be accessed from the main page of PPP: The tools are embedded into a unified framework that hides to the user the details of their interaction. A typical user session consists of running the tools, possibly iteratively, directly from the Web-browser. Partial results are always available and visible. In the following we describe step by step a simulation session.

4.1. Circuit specification and optimization

PPP contains (but is not limited to) a cell-based gate-level simulator [7]. The circuit to be simulated is first mapped onto a pre-characterized cell library. Three fields need to be specified: *i*) the name of the file containing the circuit description (*slif*, *blif* and *Verilog* are supported), *ii*) the name of a pre-characterized cell library, and *iii*) the optimization options that will be used during the mapping. The circuit is read from the user I/O directory, where the user can put his/her own circuits using a standard file transfer protocol.

Optimization options can also be specified for the mapping, and a batch run of logic optimization and library binding is run. A form-based menu is available to select the optimization options. When applying the form, SIS [11] is remotely run to map the circuit onto the library. Since PPP uses Verilog-XL as simulation platform, a Verilog description of the mapped circuit is also created and stored in the work directory.

Alternatively, the user can start an interactive optimization session. If this option is selected, a command prompt is displayed. The user issues optimization commands at the prompt, the synthesis tool executes them and reports to the user. Notice that the interactive optimization session is an example of a tighter interaction level than that allowed by batch run. Between two consecutive optimization commands, the synthesis tool *does not terminate execution*, thus the structure of the network is kept in the memory of the machine running the tool. For large networks, this kind of interaction is paramount to obtain acceptable latency between two successive user commands.

After specification and optimization, the simulation must be set up. The simulation setup can be either read from a file in the user's I/O directory or specified using the interactive

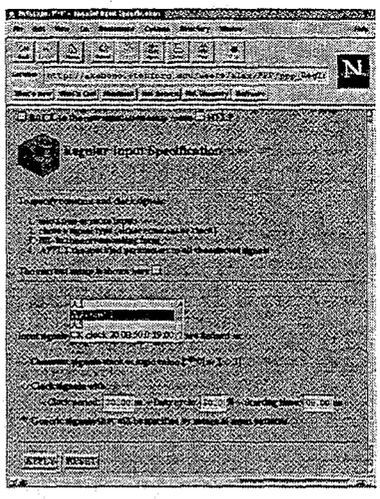

Figure 3. User interaction with PPP: simulation setup

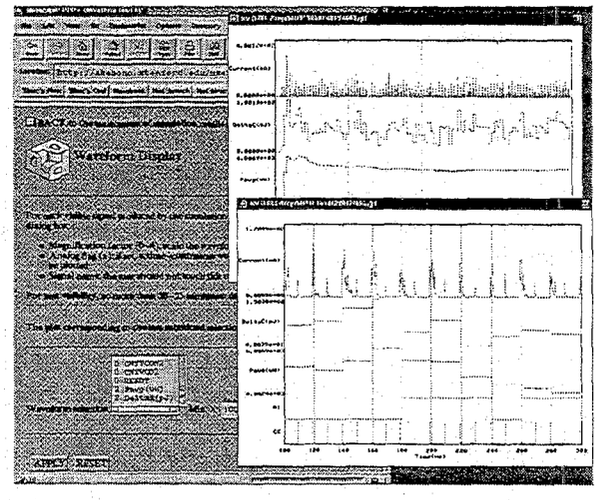

Figure 4. User interaction with PPP: waveform display

interface of PPP. In this case, three main HTML forms are available to specify: *i*) global parameters, *ii*) constant and clock signals and *iii*) generic inputs. Global parameters include the simulation style (either random or deterministic), the test size and the time step between input patterns. The constant and clock signals specification form (shown in Fig. 3) allows the user to specify some signals (selected among the list of current circuit inputs) to behave either as a periodic waveform or as a constant.

4.2. Power simulation

The PPP simulator is accessed by choosing the power simulation option of the main menu. From the first page of the simulation interface the user is allowed to: *i*) look at the circuit statistics, *ii*) look at the current simulation setup, *iii*) run the simulator by applying the current simulation setup to the current circuit, and *iv*) analyze the results of the last simulation run.

A simulation run consists of a remote execution of the PPP simulator and causes the updating of simulation results. CPU and memory consumptions are reported at the end of each simulation run.

There are two sets of simulation results: time-domain waveforms and statistical analysis. The Web-based waveform display is shown in Fig. 4. The waveform plot is saved in a GIF file and a hyperlink is created to access the GIF file from the Web-browser. Fig. 4 shows the result of two subsequent applications of the interface form on the same set of data (namely the results of a PPP simulation of sequential benchmark s344). Current, power and energy waveforms are displayed together with the clock and the input signal *A*1.

A similar interface is available for statistical results. In particular, statistical data are collected about the peak supply current per input vector, the local average power consumption (*i.e.*, the power consumption of each cell) and the total energy drawn by the circuit corresponding to each input transition.

4.3. Implementation

During a typical user session, PPP appears as a tree of HTML forms that can be navigated by the user. In reality, the HTML pages are *dynamically* created by PERL scripts controlled by the user's commands. The set of PERL scripts that generates the HTML forms for the user is the *interface layer* of PPP. Although the interaction allowed by HTML forms is limited and not very flexible, the GUI is straightforward and familiar to any user. As a consequence, a completely inexperienced user can obtain results after a single PPP session.

The *communication layer* is composed by another set of PERL scripts. The user does not directly access the communication layer, but its commands are parsed by the scripts in the interface layer, translated, and sent to the communication layer. The scripts in the communication layers perform mainly translation and setup tasks to prepare the environment needed for tool execution. Another important function performed in the communication layer is the choice of the machine on which the requested tool will run.

Finally, the environment is ready for the execution of the tools. Tool execution is managed by the application layer. If basic batch runs are requested, the tool is simply executed with the input information provided by the communication layer. Tools that execute in batch mode can be incorporated in PPP with minimal effort: the interface and communication layers are modified to allow the user to access the tool and to provide the files needed for the batch run.

When interactive remote execution is required, the process of linking the tool to PPP is less straightforward. The standard input and output channels must be re-directed to a

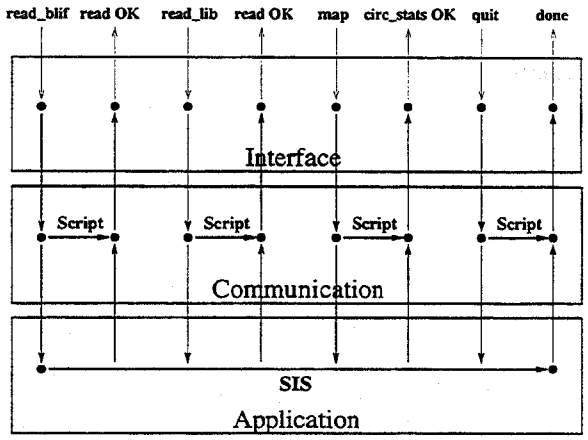

Figure 5. An example of interactive remote execution: SIS session

script that manages the interaction with the interface layer. The script is run when the user specifies a command, it translates the input format and it sends the command to the tool. The script then waits for the results of the command to be sent back by the tool. The results are saved on files or sent directly to the user after being translated to HTML. Finally the script terminates the execution, closing the contact with the tool and the interface layer. Notice that the tool *does not terminate the execution*, thus the state of the program remains in the main memory until the user explicitly asks to terminate the session. Whenever a new command is issued by the user, the interaction with the tool is managed by a new run of the communication script.

Setting a tool for interactive remote execution requires i) the ability of redirecting the standard input and output of the tool, ii) the implementation of the communication script with the characteristics described above. In the current implementation of PPP, interactive remote execution is available for SIS.

Example 4 *In Figure 5 we show a diagram representing multiple commands during a SIS session performed with the interactive remote execution paradigm. Time proceeds from left to right. Arrows represent the duration of processes. Dots represent their start and termination points.*

When the user issues a command a process in the interface layer is started to translate the command in the format required by SIS (the duration of this process is very short, therefore it is represented as a single dot). In the communication layer a process is started (with the label Script) to manage the interaction between SIS and the user. The command is then passed to SIS. When SIS returns the result the script will pass it to the interface layer that will translate it to HTML and serve it to the users. Then the communication process terminates, but the execution of SIS does not terminate until the user issues the quit command.

5 Conclusion

The implementation of PPP demonstrates the feasibility of one key concept: a uniform user interface for a heterogeneous set of EDA tools based on the WWW. We believe that PPP is an example of a new paradigm for distributed network-based applications, where the connectivity offered by Internet is exploited not only for retrieving information but also to dispatch and control execution. The second version of PPP that is under development, will exploit the features of the JAVA [8] programming environment for increasing the interactivity, the robustness and the efficiency of the current implementation.

We envision the birth of a new paradigm for the usage of EDA tools. The computational power required to run complex simulation and synthesis tasks can be concentrated in few dedicated servers, but all users will be able to access it through a familiar graphical interface that is efficiently supported by cheap personal communication devices optimized for Internet access.

6 Acknowledgement

This research is partially supported by NSF under contract MIP-9421129.

References

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

[2] A. Bededenfeld and R. Camposano, "Tool integration and construction using generated graph-based design representation," in *DAC*, pp. 94–99, 1995.

[3] M. J. Silva and R. H. Katz, "The case for design using the World Wide Web," in *DAC*, pp. 579–585, 1995.

[4] P. G. Ploger et al., "WWW Based structuring of codesigns," in *ISSS*, pp. 138–143, 1995.

[5] T. Berners-Lee et al., "The World-Wide Web," *Communications of the ACM*, vol. 37, no. 8, pp. 76–82, 1994.

[6] A. Bogliolo, L. Benini, and B. Riccò, "Power Estimation of Cell-Based CMOS Circuits," *DAC*, pp. 433–438, 1996.

[7] A. Bogliolo, L. Benini, B. Riccò and G. De Micheli, "Accurate logic-level power estimation," in *SLPE*, pp. 40–41, 1995.

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

[9] "HTML Working and background materials," http://www.w3.org/hypertext/WWW/MarkUp/MarkUp.html.

[10] J. K. Ousterhout, "Tcl and the Tk toolkit," *Addison-Wesley*, 1994.

[11] E. Sentovich et al., "Sequential circuit design using synthesis and optimization," in *ICCD* , pp. 328–333, 1992.