

Active Documentation for VLSI Design

by Mário Jorge Silva

B.S. (Technical University of Lisbon) 1983

M.S. (Technical University of Lisbon) 1987

A dissertation submitted in partial satisfaction of the  
requirements for the degree of  
Doctor of Philosophy

in

Engineering — Electrical Engineering  
and Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Randy H. Katz, Chair  
Professor A. Richard Newton  
Professor Alice M. Agogino

# **Active Documentation for VLSI Design**

Copyright © 1994

by

Mário J. Silva

Abstract

## **Active Documentation for VLSI Design**

by

Mário J. Silva

Doctor of Philosophy in Engineering — Electrical Engineering and Computer Science

University of California at Berkeley

Professor Randy H. Katz, Chair

The main proposal of this dissertation is the integration of design and documentation in a VLSI design system using hypermedia technologies. We introduce and demonstrate the use of active documents in VLSI design. These are multimedia presentations that incorporate invocations to the tools to display and modify the design data.

The combination of design and documentation systems offers a new way of creating integrated environments for designers in general, and introduces a new paradigm for VLSI design. We show that it is possible to develop effective design methodologies that enable creating design and documentation in a single thread without intruding in the design process.

Active documentation is also used as a new paradigm for creating a common interface to heterogeneous tools and data used in system design environments. We add a new integration layer that hides heterogeneity by enabling designers to control the flow of information between independent systems through a document manipulation paradigm. We also show how information-based services available through electronic commerce could be integrated into the design environment using this paradigm. Active documents become a vehicle for transporting design data and operations between frameworks in independent organizations, enabling the creation of virtual enterprises for development of electronic systems.

We develop a realistic model for creating a system supporting the integration of design and documentation based on a new infrastructure that attempts to re-use existing framework services and design and documentation tools with minimal modification. From the data point of view, it intro-

duces a new layer within the design database. This new layer contains descriptions of how the design data is organized and presented. It is a data structure with references to the design data, configuration and history data. It also includes mechanisms to tool invocations to present the data. The new presentation layer is organized as a set of active of documents. Designers manipulate them just like the documents produced by documentation processing systems.

---

Randy H. Katz

## Acknowledgments

I thank Professor Randy Katz for his invaluable support during my years at Berkeley. He provided constant academic guidance and inspired most of the ideas presented here. Randy is a great communicator and an excellent conductor of research projects. Through his example, he taught me more than anybody else. I was very fortunate in having the chance to work with him as my research advisor. I look forward to further develop our relationship both as a colleague and a friend.

Professors Richard Newton and Alice Agogino were in the dissertation committee and reviewed this thesis. I also wish to thank Richard for helping me come to Berkeley as a graduate student. Prof. Larry Rowe reviewed the dissertation proposal and guided me on the development of a strategy to evaluate the Henry System. Prof. Jan Rabaey coined the term information-centric design and reviewed many of the architectural ideas we developed for the Henry System.

Prof. Andy Neureuther gave me the opportunity to conduct Henry's usability experiment with his class. I also thank the Fall'93 EE141 students that volunteered to participate. Wayne Yeung deserves an added note of thanks for the intensive use of the initial prototype of Henry and for the many suggestions for improvement.

Prof. Bob Brayton and Alberto Sangiovanni-Vincentelli taught me how to develop CAD algorithms. Alberto was also my initial graduate studies' advisor. I thank him for the words of encouragement in the first years.

At Berkeley, we learn as much from our professors as from the continuous interaction with other students. I wish to acknowledge in particular my mates during the past years, Elan Amir, Hari Balakrishnan, Ann Chervenak, Tzi-cker Chiueh, Mike Dahlin, Ethan Miller and Srinivasan Seshan. In particular, Tzi-cker participated in the initial brainstorming that led to Henry and Srin

reviewed all my writings. Many thanks also to Wendell Baker, Ole Bentz and Arlindo Oliveira for the input on the design of Henry, and Hamid Savoj for the exciting work we did together while developing Boolean Matching algorithms.

Theresa Lessard-Smith and Bob Miller performed the administrative work required for this research. Terry was also my advisor for university and American social life related matters.

For my graduate education I received support from the following institutions: FLAD, the Fulbright Program Committee, IST, JNICT and the NSF.

Professor Luís Vidigal was my research advisor at the Technical University of Lisbon. He has been my source of inspiration and encouragement since I was his undergraduate student.

I thank my parents for their lifetime commitment to provide me with the best possible education. My wife Paula and my children Joana and Miguel suffered much during these years. I am grateful for their love and patience.

*To my wife, Paula.*

# Contents

## Chapter 1

<b>Introduction.....</b>	<b>1</b>
1.1 A New Approach to Address Design Complexity .....	4
1.2 Combining Design and Documentation.....	6
1.2.1 Active Documentation .....	9
1.2.2 Tool Ensembles in VLSI Design.....	12
1.3 Support of multi-organizational design methodologies .....	13
1.4 Dissertation Overview .....	14

## Chapter 2

<b>Related and Previous Work .....</b>	<b>18</b>
2.1 Introduction.....	18
2.2 Operating Systems Services and Application Frameworks .....	22
2.2.1 COSE and CDE.....	24
2.2.2 CFI .....	25
2.2.3 Design Process Management Systems.....	26
2.2.3.1 What is a Design Tool? .....	28
2.3 Documentation Processing Systems .....	29
2.3.1 SGML .....	30



2.4	Hypertext and Hypermedia Systems.....	31
2.4.1	Open Hypermedia Systems.....	33
2.4.2	Compound Document Manipulation Architectures .....	35
2.4.3	Notebooks .....	36
2.5	Design Rationale Capture Systems.....	38
2.6	Computer Supported Cooperative Work.....	40
2.6.1	Active Mail .....	41
2.7	Summary .....	42

### **Chapter 3**

	<b>Internet System Software and Applications .....</b>	<b>43</b>
3.1	Introduction.....	43
3.2	MIME — Multipurpose Internet Mail Extensions.....	45
3.3	Enabled Mail.....	46
3.3.1	Enabled Mail and Security .....	48
3.4	World Wide Web.....	50
3.5	Electronic Commerce.....	51
3.6	Summary .....	53

### **Chapter 4**

	<b>The Next Generation Design Environment .....</b>	<b>55</b>
4.1	Introduction.....	55
4.2	Requirements .....	57
4.2.1	Multiple Heterogeneous Sets of Tools.....	58
4.2.2	Interaction with Electronic Commerce Services.....	59

4.2.3 New Business Models: The Virtual Corporation .....	60
4.2.4 More Design-Related Information Needs to be Integrated.....	62
4.3 Architectural implications for new design environments .....	63
4.3.1 Common Messaging Services.....	64
4.3.2 Common Data and Methodology Management Services .....	67
4.3.3 Uniform Methods to Manipulate Design Information .....	69
4.4 Summary .....	69

## **Chapter 5**

<b>The Infrastructure for Integrated Design and Documentation.....</b>	<b>71</b>
5.1 Introduction.....	71
5.2 Communications in Henry .....	73
5.2.1 Conceptual Model for Message Handling .....	77
5.2.2 Operation of the HUB.....	78
5.2.2.1 The Message Transport Layer.....	80
5.2.2.2 The Message Handling Layer .....	85
5.3 Henry as an Open Hypermedia System .....	87
5.3.1 The Henry Linking Mechanism.....	88
5.3.2 Making Design Tools Hypertext-aware .....	89
5.3.3 Link Configuration.....	91
5.3.4 User Interaction with HUBs.....	93
5.4 Henry as an Infrastructure for Building Tool Ensembles .....	94
5.4.1 Symmetric Tool Ensembles .....	96
5.4.2 The Role of the Extension Language.....	97
5.4.2.1 Extension Languages and Concurrency Control.....	97

5.4.2.2 Extension Languages and Inter-operability .....	99
5.5 Summary .....	100

## Chapter 6

<b>Tools For Integrated Design and Documentation.....</b>	<b>101</b>
6.1 Introduction.....	101
6.2 Integrated Design and Documentation with Electronic Notebooks.....	104
6.2.1 Organization of Documentation in Electronic Notebooks.....	104
6.2.2 Authoring Active Documents with an Electronic Notebook .....	107
6.2.2.1 Document Building Toolkit .....	107
6.2.2.2 Active Clip-Art .....	108
6.2.2.3 Agents .....	110
6.3 Document Representation and Manipulation.....	111
6.3.1 Basic Conceptual Model for Document Representation.....	112
6.3.2 Integrated Design and Documentation Concepts.....	114
6.3.3 DocScript versus SGML .....	116
6.4 The Navigator .....	118
6.4.1 The Navigator Versus Other Documentation Tools .....	121
6.5 Summary and Conclusions .....	122

## Chapter 7

<b>The Henry System Implementation .....</b>	<b>124</b>
7.1 Introduction.....	124
7.2 System Services Implemented by the HUBs .....	126
7.2.1 Session Management .....	126
7.2.2 Services Registry .....	127

7.3	Interfaces with Design Tools and Frameworks.....	128
7.3.1	Interface to Spice3 .....	131
7.3.2	Interface to Magic .....	132
7.3.3	Interface to VEM .....	132
7.4	Navigator Implementation .....	133
7.5	Interface with the WWW .....	135
7.5.1	Active Messages Transported by Electronic Mail .....	136
7.5.2	Active Messages Transported by the WWW Protocol .....	138
7.6	Exploring Internet-based Design Scenarios.....	139
7.6.1	The Remote Circuit Simulation Service .....	141
7.6.2	The On-line Component Selection and Ordering System.....	144
7.7	How the Henry System Evolved.....	148
7.8	Summary and Conclusions .....	151
 <b>Chapter 8</b>		
	<b>Active Documentation Experiment .....</b>	<b>153</b>
8.1	Introduction.....	154
8.2	Setting the Goals .....	155
8.3	Adapting the Prototype to the Users Environment .....	156
8.4	The Documents Produced.....	158
8.4.1	Software Lab Reports .....	159
8.4.2	Projects.....	164
8.5	Observation Data .....	164
8.5.1	Automatic Usage Statistics Collection.....	167

8.5.2 Questionnaires.....	168
8.5.3 Informal Contacts.....	171
8.6 Hypotheses for an Industrial Strength Experiment.....	172
8.7 Summary and Conclusions .....	174

## **Chapter 9**

<b>Conclusion and Directions for Future Work.....</b>	<b>176</b>
9.1 Research Contributions.....	176
9.2 Notes and Recommendations for a Future Experiment .....	178
9.3 New Directions .....	180
9.3.1 Message Oriented Design Management Systems .....	180
9.3.2 Design Space Exploration Tools .....	182
9.3.3 Use of New Devices to Manipulate the Design Information.....	183
9.3.4 Use of Video in VLSI Design .....	184
9.4 Final Words .....	185

# List of Figures

Figure 1.1	The New Viewport into the Design Environment.....	3
Figure 1.2	The Spiral of VLSI Design .....	5
Figure 1.3	How Complexity Has Been Addressed.....	6
Figure 1.4	Dimensions of Electronic Systems Design Complexity .....	7
Figure 1.5	Design and Documentation in Single Thread .....	8
Figure 1.6	A VLSI Active Design Document .....	10
Figure 1.7	New Layer.....	11
Figure 1.8	An Active Document Describing a VLSI Design.....	15
Figure 1.9	Methodology of the Dissertation .....	16
Figure 2.1	Framework Reference Model .....	23
Figure 4.1	The New Design and Documentation Environment .....	65
Figure 5.1	The Information Centric Design Environment Organization .....	75
Figure 5.2	Message Handling in Henry.....	78
Figure 5.3	Architecture the Henry System HUB. ....	81
Figure 5.4	The Sub-Layers of the HUB Message Transport Layer .....	82
Figure 5.5	The HUB Message Handling Layer.....	86
Figure 5.6	The Design Environment from the Users' Perspective.....	90
Figure 5.7	Magic as a Hypermedia Tool .....	92
Figure 5.8	The HUB's Session Manager Dialog Window .....	94
Figure 5.9	The Extension Language in Henry and CFI environments.....	98
Figure 6.1	The Information flow as seen from Henry's Electronic Notebook .....	105

Figure 6.2	Domain-specific Accelerators for Creating Active Documents.....	109
Figure 6.3	The Role of the Document Description Language in Henry .....	111
Figure 6.4	Code for Describing a Minimal DocScript Document .....	113
Figure 6.5	The Effect of the Cursor Position in the Design Database State .....	117
Figure 6.6	The Elements of the User Interface of the Navigator .....	120
Figure 7.1	The Components of the Henry Prototype .....	125
Figure 7.2	Extensions Loaded by the HUB Tcl Interpreter.....	130
Figure 7.3	Implementation of Henry's Interface to the WWW. ....	137
Figure 7.4	Flow Diagram of the Simulation Server .....	142
Figure 7.5	The User Interface of the Electronic Component Library .....	146
Figure 7.6	Information Flow in Transactions with the Component Library .....	147
Figure 8.1	The Henry Help System as an Active Document .....	160
Figure 8.2	An Active Document Containing the Report of Software Lab 3.....	162
Figure 8.3	An Active Document Describing the Detection of a Layout Error .....	163
Figure 8.4	An Active Document Illustrating the Design of a Logic Gate.....	165
Figure 8.5	An Active Document for the Design of a Long Interconnect.....	166
Figure 9.1	Project Summary Sheets .....	181

# List of Tables

Table 3.1	Format of a MIME message .....	47
Table 3.2	Format of an Enabled Mail message.....	49
Table 5.1	Summary of CFI's Framework Architecture Reference Views .....	72
Table 5.2	Common Operations Supported by the Tools Integrated with the HUB .....	84
Table 6.1	Design versus documentation concepts .....	102
Table 6.2	Summary of the Model for Representing Active Documents.....	114
Table 6.3	Feature Comparison Between FrameMaker, Mosaic and the Navigator .....	122
Table 8.1	Summary of the Documents Produced During the Experiment.....	159
Table 8.2	Navigator Usage Statistics .....	168
Table 8.3	The First Questionnaire.....	170
Table 9.1	New Design Data Representation Formats Spawn New Design Tools .....	182
Table 9.2	Electronic Notebooks versus Paper-based Notebooks.....	183



“I am speaking of an intelligent writing which ... can defend itself, and knows when to speak and when to be silent.”

— Socrates

## Chapter 1

# Introduction

In current VLSI design environments, designers are well supplied with tools to help them create and analyze their designs. More recent work has concentrated on the development of a set of integrated services to help cope with the complexity management of the design process and its data. These are collectively known as CAD frameworks [Harr90]. Despite these efforts, little has been done to help designers understand their designs.

In intensive design environments, we believe that *understanding* is synonymous with documentation. Inevitably, documentation becomes central to the design process itself. The best definition of design is recursive: it is the process of writing its documentation.

We believe that a quantitative improvement in documentation would be possible if today's electronic-based media could be exploited. Hypermedia is the ability to link together text, graphics, audio, and video into a coherent, navigable document. Our approach uses a hypermedia system as a front-end to the CAD system. This kind of front-end provides a radical new means for documenting system designs via multimedia presentations. For

example, segments of a videotaped design review could be used as annotations to the files specifying a design artifact.

This dissertation discusses the integration of hypermedia technology within CAD environments for three purposes:

1. Combination of design and documentation activities.
2. Providing a common user interface to heterogeneous design and documentation tools.
3. Supporting the exchange of design-related information between designers in independent groups and remote design information services via electronic documents.

These are discussed in terms of our experience with:

1. Developing a system model that supports this integration.
2. Prototyping a system that implements the model.
3. Using and evaluating the system.

A common user interface to the design tools has long been identified as one of the critical sub-systems of a CAD framework. Unlike more traditional CAD interfaces, the one we propose is founded on the metaphor of manipulating design documents. It uses a hyper-linking mechanism for relating pieces of design data and for coordinating design tool executions. This makes possible the construction of several forms of multi-media design documentation, as well as a new way to access design history. As a result, we have added a new viewport to the design environment, in which tools, component libraries and design files can be accessed and organized in a simple, yet powerful way (see Figure 1.1).

We also have exploited the use of the documentation interface for other purposes. Documentation handling processes can be used as a metaphor for exchanging information between CAD systems. These can be used to define protocols for passing design data as multi-media documents, offering a new way to provide interoperability.

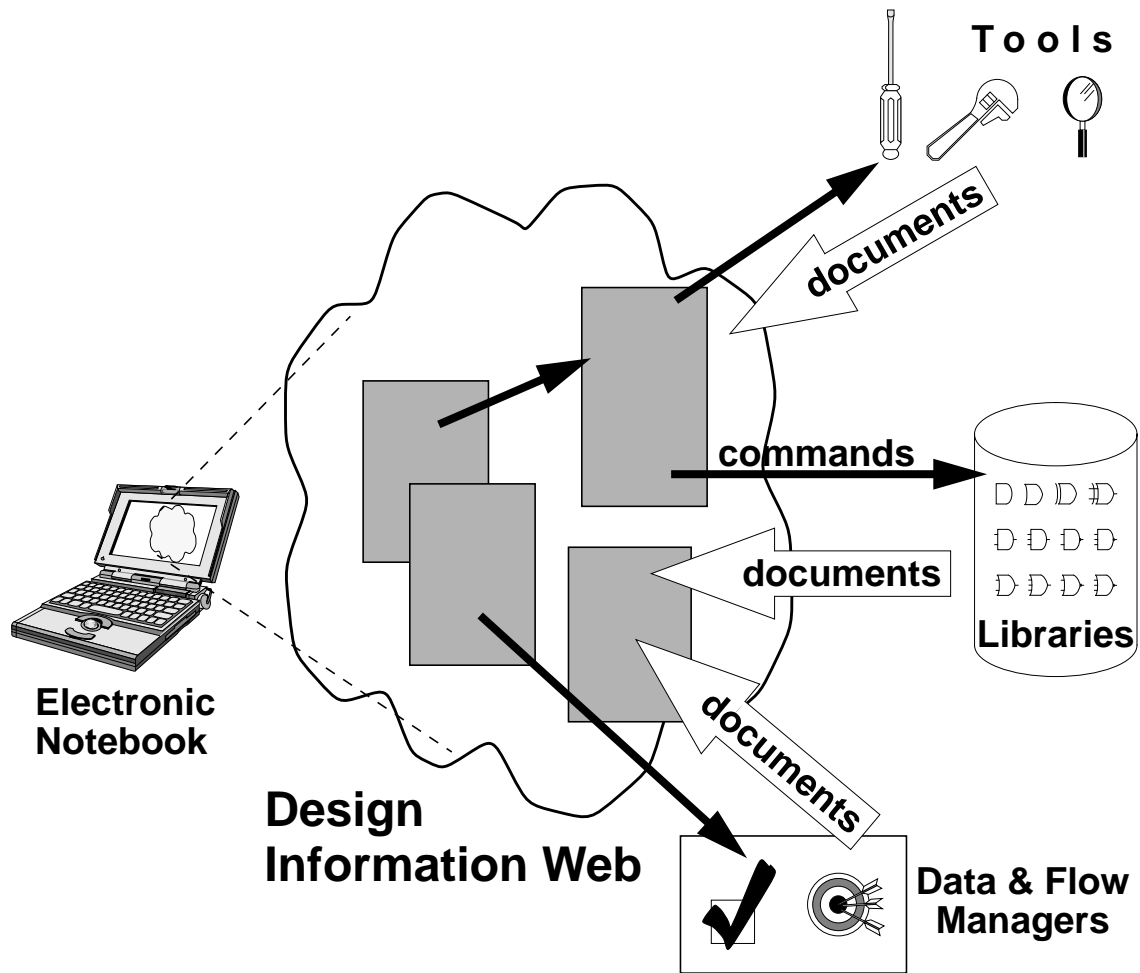


Figure 1.1 The New Viewport into the Design Environment

Our goal is to give designers an integrated view of all the data, documentation, tools, and framework services available to them. We use a common metaphor based on documentation. All design information is organized as a hypermedia document, which can be browsed by an electronic notebook. The data is still manipulated by the tools, but viewed as part of the design documents.

Documentation-based interfaces to assist designers in accessing services distributed across the Internet have become popular with new kinds of information systems like the World Wide Web [BL94b]. These access the data through information servers as if they were part of a network-based hypermedia system. Electronic design-specific commercial services, such as component information or remote simulators, could be made available to designers via this infrastructure. In this dissertation, we describe the documentation based

interface we developed to support interaction and retrieval of information from design documentation.

In the remainder of this chapter we start by discussing how design complexity has been addressed in the past and propose a new paradigm based on documentation. Section 1.2 presents the case for VLSI design methodologies that integrate design and documentation activities into a single thread. In Section 1.3 we review new information services and forms of collaboration available to designers over the Internet, and advocate the use of a documentation manipulation metaphor for exchange of design related information between these independent design environments. Finally, in Section 1.4 we summarize our research goals and give an overview of the dissertation.

## **1.1 A New Approach to Address Design Complexity**

In an early paper, Newton and his collaborators advocated the use of computer aids for managing the design process [Newt81]. Gajski and Kuhn described the VLSI design process as a spiral, where designers use the tools to create successively less abstract representations in different domains [Gajs83] (Figure 1.2). At about the same time, Séquin discussed how complexity inevitably becomes the crucial problem in VLSI design and the ways to address it [Sé83]. These include the creation of new levels of abstraction, hierarchical structures, methodologies, partitioning and separation of design and implementation. Their vision has guided research in VLSI system design since then. This has resulted in the creation of successively higher levels of abstraction for capturing design descriptions and for handling the design process (see Figure 1.3).

However, new levels of abstraction never completely hide the lower levels. To complete a design, it is usually necessary to interact with design tools that work at the lower levels of representation. For instance, designers may synthesize automatically a large CPU block from a VHDL model, but they will always have to verify the connections between the modules and from modules to the pads of the chip. If timing issues are important, it is likely that a simulation at the transistor level will be performed involving the circuitry in

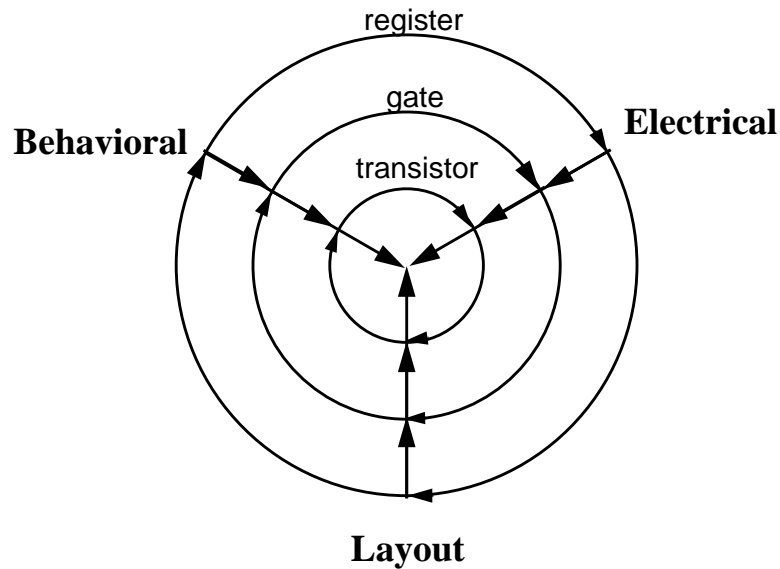


Figure 1.2 The Spiral of VLSI Design

The VLSI design process evolves from more abstract to more detailed representations in multiple domains. Designers use the tools to create or synthesize them and verify their accuracy.

critical paths. More important, to make changes into the design, designers will likely have to correlate descriptions at different levels, in a process called back-annotation. The addition to the design environment of new levels of abstraction and new tools for handling them has the benefit of enabling designers to create much larger systems, but the liability of making them more difficult to design and maintain.

In our view, this new kind of complexity needs to be addressed by a new generation of systems that integrate design and documentation tools. These include aids for producing and organizing design related documentation. These documents then operate as the interface for accessing the multiple views of the design space, illustrating alternative aspects of the design.

Certain aspects of system design are not well supported by today's design environments. Modern synthesis and verification tools address the problem of creating and interconnecting modules at the chip and printed circuit board levels (see Figure 1.4). However, this level of automation covers a rather restricted subset of the complete system design pro-

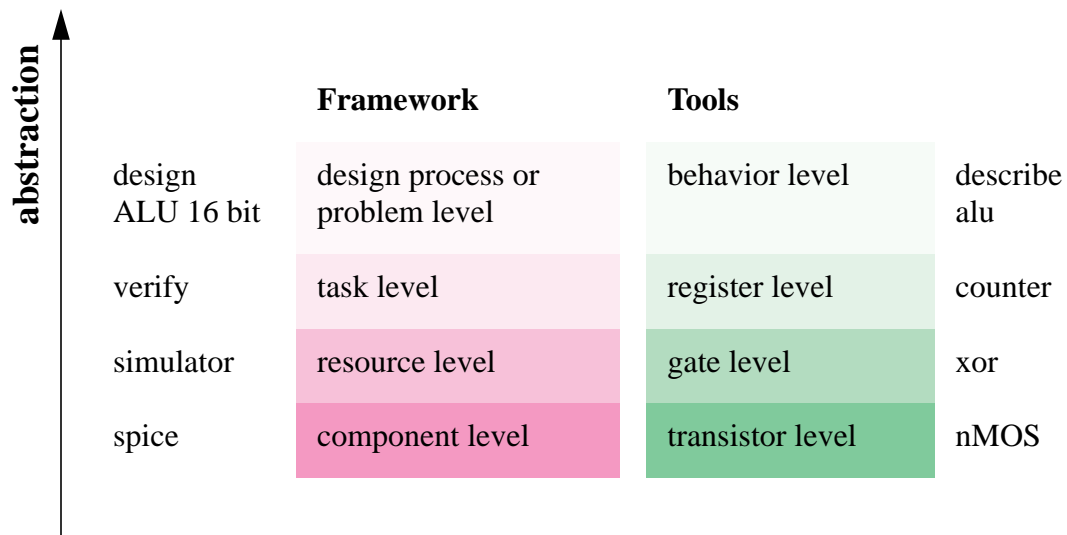


Figure 1.3 How Complexity Has Been Addressed

We have been coping with complexity by introducing successfully higher levels of abstraction both in the design representations and the tools that manipulate them. However, since new layers of abstraction do not completely hide the lower levels, designs are becoming more difficult to understand and designers must become familiar with increasing numbers of representations.

cess. In almost every case, system design also involves the procurement of modules and components that must be integrated. It also encompasses interaction with service providers, such as manufacturers, and other groups working in related activities, such as product marketing and definition. Instead of starting systems design from a pre-defined library of modules and manufacturing processes, we observe that designers also work intensively as information hunters. Design systems, should assist designers to locate and retrieve information and document the decisions made.

## 1.2 Combining Design and Documentation

A project's internal documentation is of crucial importance for design teams, where interfaces have to be shared and high personnel turnover is unavoidable. But the text processing system is not well integrated with the design system. As a result, designing and documenting VLSI artifacts are largely non-integrated processes. In many cases, the

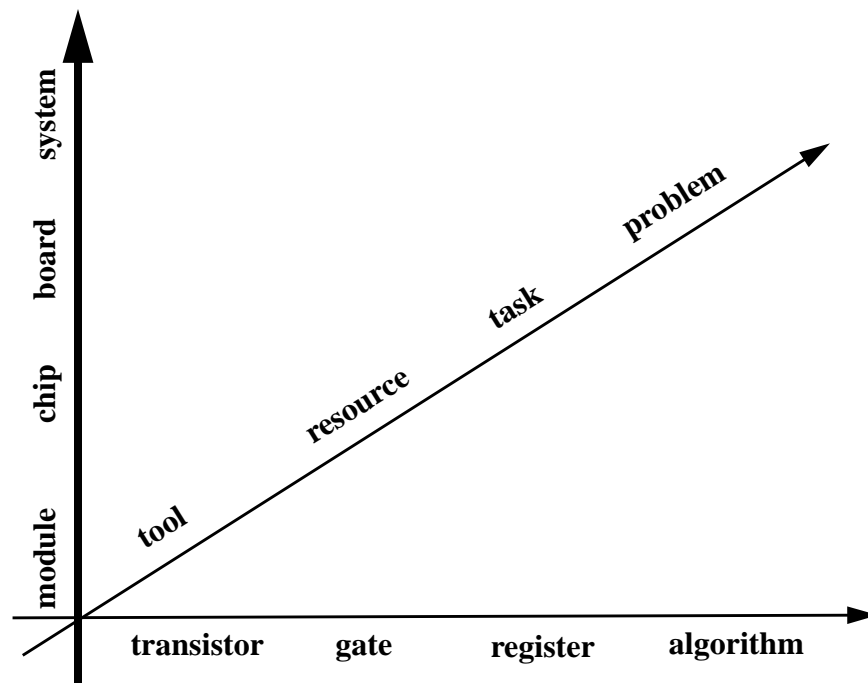


Figure 1.4 Dimensions of Electronic Systems Design Complexity

VLSI design automation has enabled the creation of increasingly complex modules through higher levels of abstraction for their specification and for the methodologies used. However, as we move up in the vertical axis we observe that the level of support available to designers decreases. We can produce very complex modules, but electronic systems still have relatively few components. This happens because, at the system level, design methodologies have to be substantially different: instead of using tools to synthesize the artifacts, designers need to hunt for information about modules that could be used, and interact with design and manufacturing services. This information is currently available mostly in the form of documents. The new generation of design environments must provide support for searching and organizing this kind of design information. New tools could then help automate the process of making design decisions at the system level.

design documentation is created only after the implementation is complete (see Figure 1.5).

Commercially available CAD frameworks, such as those from Cadence or Mentor Graphics, partially address this problem by providing “hooks” to text processing systems such as Frame Technology’s FrameMaker. Since the primary focus of these facilities is to make

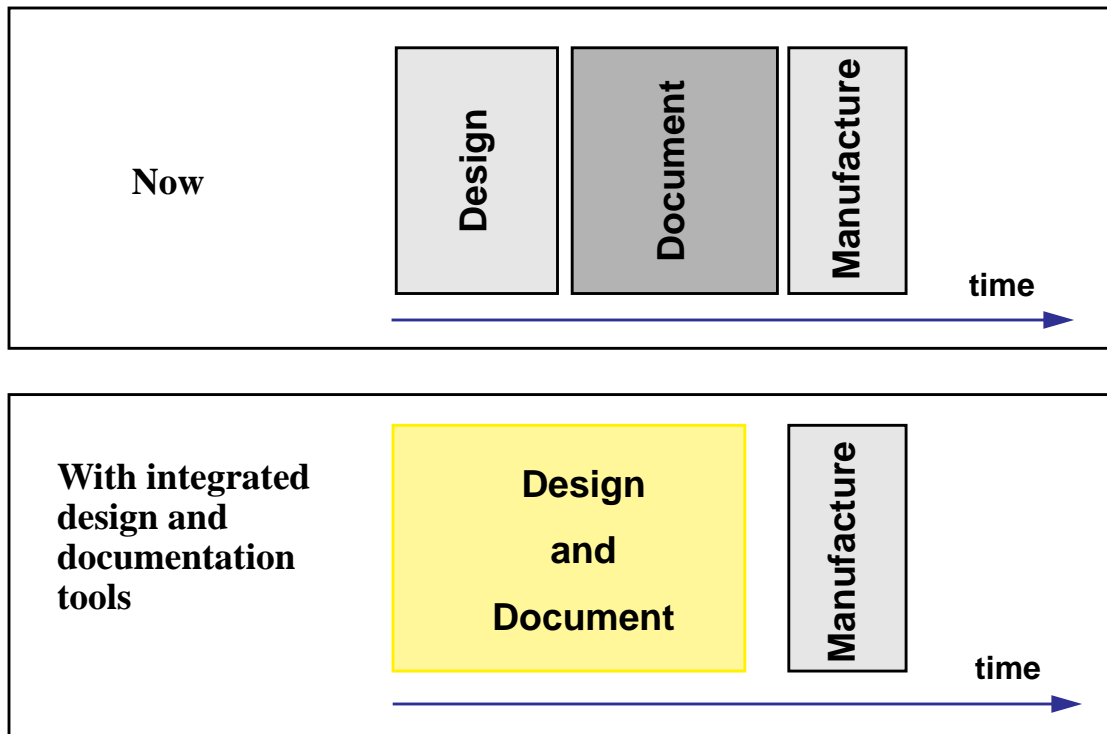


Figure 1.5 Design and Documentation in Single Thread

Currently design and documentation activities run independently during the design process. Our goal is to combine them into a single thread. To achieve this, we need to integrate design and documentation tools into a common framework, providing support for ensuring consistency between design and documentation data and a common interface to manipulate them.

datasheets and other documentation available on-line to end-users, they provide little assistance in documenting a VLSI system as it is being created.

We believe that the understandability of designs would be significantly improved if the processes of design and documentation could be better integrated. Both types of information should be made available and manipulated concurrently from the same user interface. Design specifications should be easily accessible while browsing the design models. Object derivations and design histories represent important information needed to understand a design. Today this design information cannot be accessed as understandable documents.

To create the design data and its documentation in a single thread, designers will need:



- *support for maintaining consistency between related pieces of information.* A modification to a component's behavior, will require the designer to find what parts of the system's documentation will need to be re-written. He or she should be able to navigate from one to the other and quickly identify the portions of the design and documentation affected by a change.
- *A single interface to design and documentation data.* To easily reach any piece of design data and documentation, designers should have views where both are interpreted as information related to the task at hand that can be manipulated uniformly.

However, these should be provided without enforcing intrusive methodologies. Even with better coupling between design and documentation tools, it is essential that the design flow does not force designers to write documentation or when to write it. We view integration between design and documentation as providing a set of mechanisms for merging design and documentation data into presentations that enable designers to quickly find and modify the all the design related information.

Our approach for combining design and documentation is to create a common support infrastructure, where all of the tools that manipulate design information are well integrated. We use the term *tool ensembles* to describe these sets of tightly integrated tools, which can be synchronized to help a designer perform a complex design task as well as if he or she were using single tool customized to assist him in the task. *Active documents* are the interactive presentation of a design produced by using a tool ensemble. We elaborate these two concepts in the context of VLSI design in the remainder of this section.

### 1.2.1 Active Documentation

Hypertext techniques have been used in the VLSI design domain, primarily for on-line help [Lee87]. A new opportunity was made possible by the technology of active documentation [Terr90]. Active documents contain objects, beyond static text or figures, with which readers can interact. A traditional figure is a black-box to the document processing

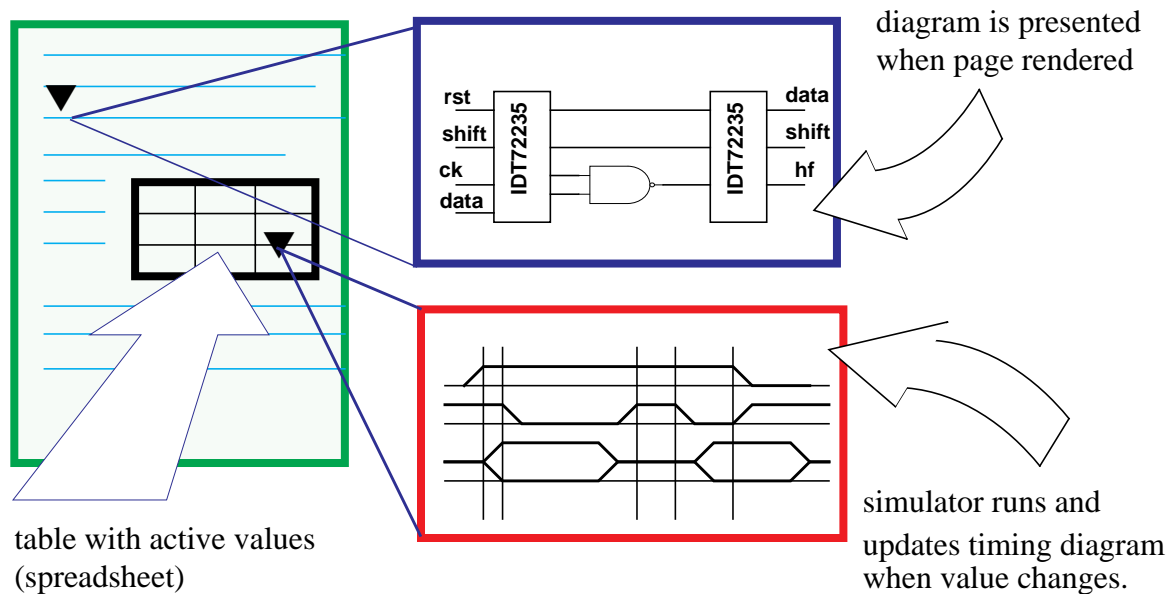


Figure 1.6 A VLSI Active Design Document

Active documents have their contents filled-in as they are rendered. Designers' interactions with an active document may cause its contents to be modified or other portions of the document to be displayed. The figure illustrates how active documents could be used as a better replacement for traditional paper documents in VLSI Design. Instead of static pictures, we could have the design tools display insets with the actual design data. Changing parametric data in a document's table would cause the circuit being described to be re-simulated and the update of the inset showing the waveforms.

system. An *active figure* could be the main window of a running design tool, to which the user sends commands and observes changes to the document. Adding interactive objects to a document begins to blend some of the notions of document processing with a direct interface to a CAD system. For instance, in the production of a document, a logic simulator could be used to illustrate the timing protocol for interfacing with a library module, or a schematics editor could be invoked to display circuit diagrams, allowing readers to examine details at their will (see Figure 1.6).

In VLSI design, we observe a general trend towards executable specifications. The development of VHDL, the VHSIC Hardware Description Language, resulted from the users' need to verify specifications via execution [IEE87]. Companies constructing VLSI components are increasingly including VHDL models with their documentation. VLSI compo-

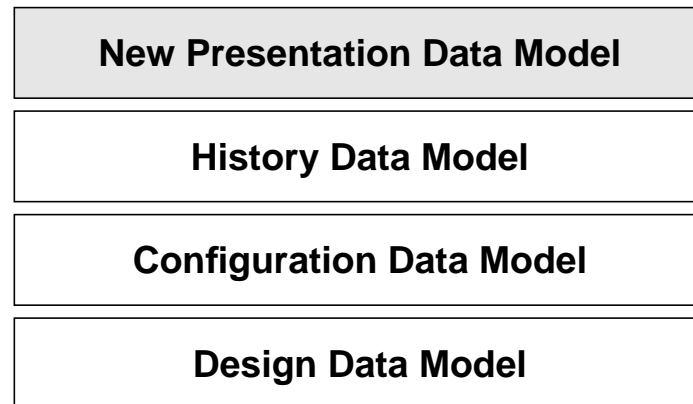


Figure 1.7 New Layer

Historically, VLSI design data bases started by defining common data models to represent the design artifacts. These have later been extended to support multiple configurations of the data. More recent research defines conceptual models for the design flow, which are based on representations of the design process history. We propose a new *data presentation* layer. This describes how data is presented to the users and how to link pieces of more loosely related pieces of information, such as a design's layout and the associated documentation. In our model, the data in this layer is represented as active documents.

nents would be much easier to understand if these specifications could be executed and observed directly within the documents that describe them. As a result, active documents will appear to designers as the next evolutionary step.

The current generation of design frameworks has focused on providing common services to the tools: intertool communication, common databases and the same user interface look and feel. More recent work on process management services for VLSI design enable the creation of sequences of tool executions in the time domain, by finding dependencies between data and running sequences of tools to optimize the data [Klei94]. A need also exists to present the results produced by the tools on the user's screen. To display some design details or to perform some user level tasks, several tools have to be running concurrently with some graphical arrangement between them. From the design data organization point of view, this corresponds to adding a new *presentation* layer to the conceptual model of the design environment (see Figure 1.7). We propose the a definition of this new layer, based on documentation.

### 1.2.2 Tool Ensembles in VLSI Design

The idea of combining tools to produce ensembles is not new. We have seen the same idea exploited in different ways as new generations of platforms for running CAD systems were introduced. It is instructive to discuss how it has been applied in successive generations of software produced at Berkeley. The Unix environment, where most of the initial CAD systems were built, popularized the notion of combining tools by the *filters and pipes* paradigm. In this environment, design tasks are defined as sequences of tool executions that use as input the output of the previous tool and run as a batch job. A good example of such an ensemble is the *Mosaico* layout system. This is a script to route the macro-cells in a chip following the conventional steps: channel definition, global routing, detailed routing and compaction. In the script, each step corresponds to a tool invocation.

Nonetheless, this paradigm has become obsolete. The capability to *interactively* exploit design options has become more important. An example of a new generation of tool ensembles in VLSI is the MIS logic synthesis system [Bray87]. MIS is a shell and a set of independent tools, all combined into a single program. Users normally start the shell and then invoke the tools interactively to operate on a logic-level representation of a circuit. Another advantage of this architecture is that it does not require loading and storing the circuit representation into a file and re-initializing the system between operations. This becomes especially important as circuit complexities increase to millions of gates. Many tool re-initializations would have a significant impact on performance.

In spite of that, MIS is a single program. In VLSI design, many situations arise when a set of independent tools are needed for the same task. For instance, consider the situation of a designer adjusting the timing performance of a circuit. In general, this is accomplished with a circuit editor, a simulator and a waveform displayer. When the circuit is modified, the user needs to re-simulate it and look at the new waveforms. Frameworks, such as the *Octtools*, were designed to make it possible to create these tool ensembles as set of independent programs [Harr86].

However, existing frameworks still do not satisfy our requirement for integrating design and documentation tools and access them through a common interface. Design frameworks do not provide interfaces to send and receive commands from documentation tools.

There are also several examples of application frameworks based on documentation systems, such as FrameMaker and Interleaf. These provide hooks for integrating external tools and retrieving information from databases. However, the limitations we find are of the same type: interfaces to design tools are very limited. Tools view themselves as the center of the universe. There is not a general mechanism for invoking other tools as a peer.

In both domains frameworks are designed so that there is a specific tool that works as the center of the environment. In the Octtools, all applications have to be started from VEM, the design database browser. With Framemaker, it is possible to modify a design tool to display its contents inside an inset, but it would be impossible to have it running in a sub-window of a design tool for displaying a related piece of documentation data.

Our goal is to create a common framework, where tools in both domains could be integrated in a symmetric way. No specific tool or design domain should have a special role that would make it the center of gravity of the design environment.

### **1.3 Support of multi-organizational design methodologies**

The expansion of the Internet in recent years created the opportunity to develop new models for collaboration between organizations. Groups of independent companies, called Virtual corporations, will work together to market, design and manufacture products [Davi92]. We believe that, to support this, concurrent engineering environments currently supported by a single framework, will evolve into heterogeneous, wide-area distributed, multi-framework environments, independently managed at different locations.

The Internet is also opening a new market, where information based services can be transacted electronically. Many new services for hire could be made available to support electronic designers, such as remote simulators, emulators or on-line component information.

Information systems like the World Wide Web popularized an interface to heterogeneous information services based on documentation. We favor the application of this information manipulation model to electronic design related electronic commerce and exchange of information between organizations participating in virtual enterprises. This model has the advantage of being well integrated with the information centric view of the design environment supported by the new integrated design and documentation interface.

## 1.4 Dissertation Overview

In this dissertation, we present the design, implementation and usage experience of a new CAD System for VLSI design. This is called the Henry System, or more simply Henry<sup>1</sup>. Figure 1.8 shows the windows of an active document produced with some of the design and documentation tools integrated with the existing implementation.

Figure 1.9 shows the methodology we used for developing the Henry System. We built it as vehicle for studying the research goals outlined in previous sections. These can be summarized as:

- Development of a framework and tools supporting integrated design and documentation methodologies.
- Providing a new common interface to the design environment, based on documentation. This creates a new inter-operability layer, unifying the interactions between VLSI designers and the multiple heterogeneous systems in use.

---

1. The name is given after Henry, the Navigator, the first creator of a research enterprise, founded on scientific innovation and systematic collection of information.

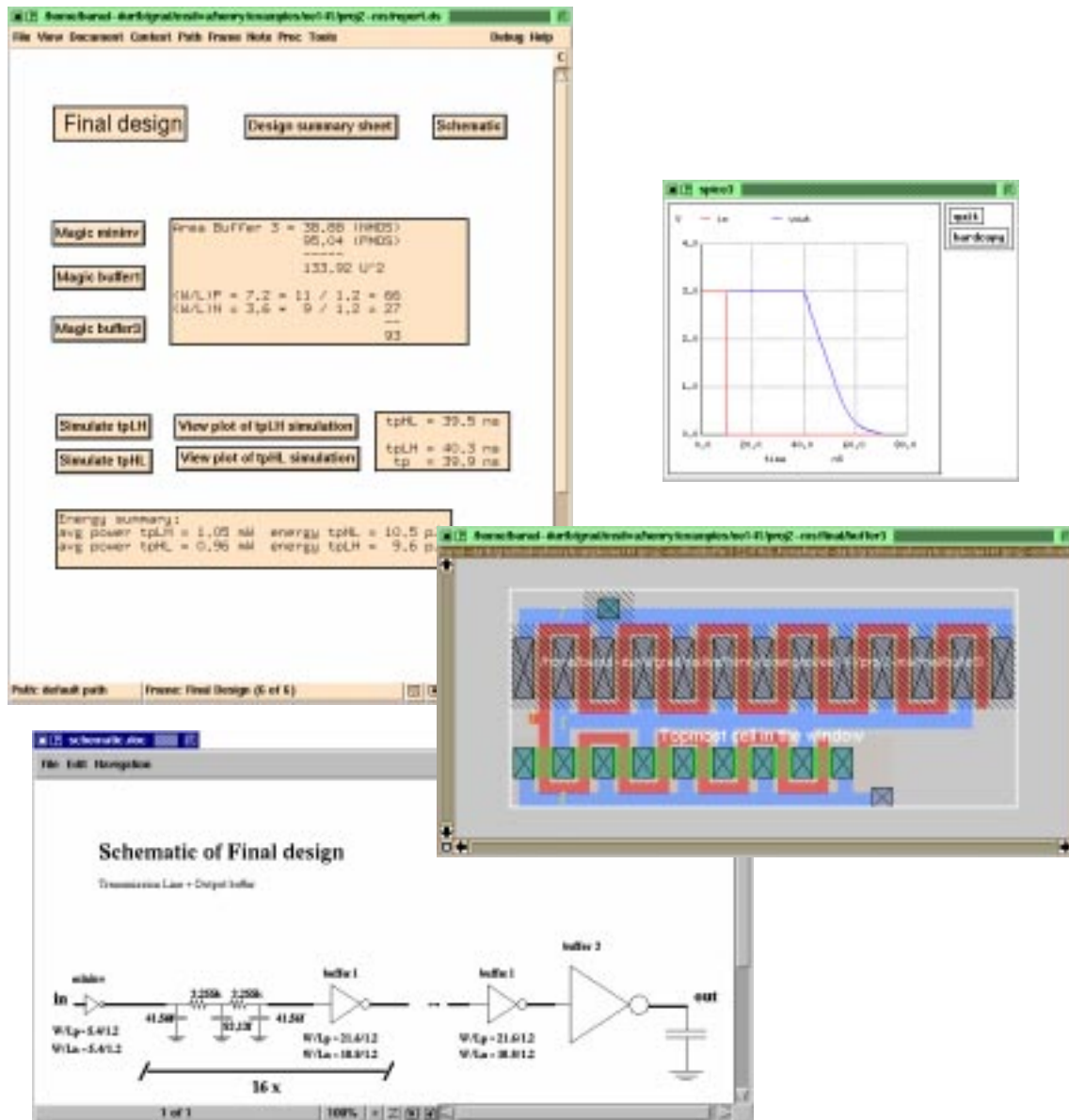


Figure 1.8 An Active Document Describing a VLSI Design

The figure shows an active document describing the design of a transmission line and output buffer for a signal in a VLSI chip. The active document was produced with the Henry System, a prototype integrated design and documentation system whose design and implementation is presented in this dissertation. Going clockwise from the top-left corner, we can see windows of the Navigator, an electronic notebook design for the system, Spice3, a circuit simulator, Magic, a layout editor, and FrameMaker, a documentation processing system. All the windows were created as the result of pressing "buttons" in the Navigator's window. All the design tools shown have been modified to have the same capability of sending commands to other tools to display windows with related information.

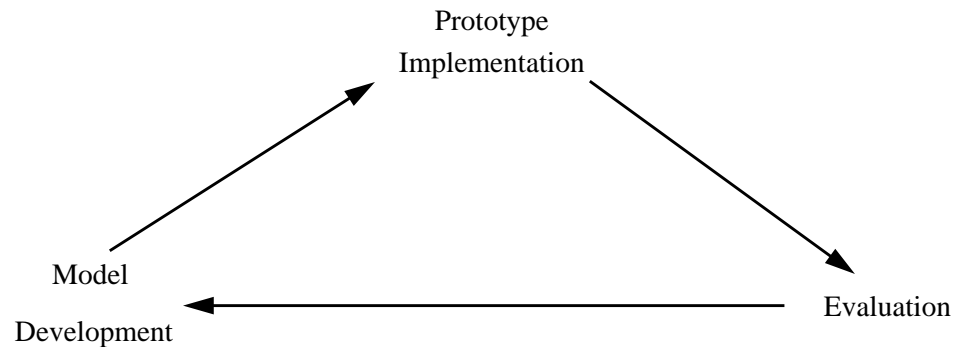


Figure 1.9 Methodology of the Dissertation

For the development of the Henry System we adopted an iterative methodology. We created an initial model for the system and used it in the implementation of a first prototype. This was used by a group of users. We then modified the model based on the observation and evaluation of the use of the prototype.

- Integration of information-based services available through electronic commerce into the design environment.

In Chapter 2, we survey previous research works and current initiatives that complement our work. We cover on-line document representation formats, hypermedia systems, previous approaches for integrating design and documentation, computer aids for supporting collaborative work and research in electronic design frameworks.

Chapter 3 presents protocols, message formats and information systems used for communications in the Internet. We also review extensions for sending active documents through electronic mail systems, the World Wide Web, and new information services we anticipate it will offer to designers.

In Chapter 4, we describe the conceptual model for the next generation of design environments. These will be composed of multiple heterogeneous frameworks operated by cooperating independent groups. In the new environment, design and documentation are seamlessly integrated. We start by examining the assumptions used for building existing design frameworks that, in our view, are no longer valid. From these new requirements,



we derive the implications for the next generation. We propose the documentation paradigm for organizing and manipulating design and documentation data. This is also used to provide a unified method to integrate heterogeneous tools and frameworks.

Chapter 5 presents the architecture of the Henry System. This implements the main blocks of new design environment organization introduced in Chapter 4. We describe the communications infrastructure for supporting integrated design and documentation and exchanging design information as active documents. Then we present Henry as seen by its primary users: the designers and system integrators. The former see it as a collection of aids for manipulating all information as documentation. The latter view it as an infrastructure for building ensembles of design and documentation tools.

In Chapter 6, we present tools and authoring techniques that enable the use of integrated design and documentation methodologies within the new design environment. We describe the conceptual model of the Navigator, an electronic design notebook that uses the services of the Henry System presented in Chapter 5.

Chapter 7 describes the implementation of the various components of the existing prototype. This includes Henry's support services, the interfaces offered to design and documentation tools, and the Navigator. We also present design scenarios involving interactions with electronic commerce services using active documents. These were prototyped using Henry's software.

In Chapter 8, we describe a usability test involving the use of Henry in laboratory sessions and class projects by the students of a VLSI design course. We present the goals set for the experiment, how it run, the documents produced, the collected data, and our conclusions.

Chapter 9 summarizes the main contributions of this thesis, recapitulates the lessons learned from this project and gives directions for further research.

We use [logic] to simplify and summarize our thoughts. We use it to explain arguments to other people ... We use it to reformulate our own ideas. But I doubt that we often use logic actually to solve problems or to “get” new ideas. Instead, we formulate our arguments and conclusions in logical terms *after* we have constructed or discovered them in other ways; only then do we use ... formal reasoning to “clean things up.”

— Marvin Minsky, The Society of Mind

## Chapter 2

# Related and Previous Work

In this chapter, we review existing software supporting the production of documents and the design process in various domains related to our application. We cover operating environments for electronic design systems, documentation processing tools, hypermedia, design rationale capture systems, and tools supporting collaboration.

### 2.1 Introduction

The integration of documentation and design is not a new idea. For example, this has been used with some success in the Mathematica system [Wolf91]. Mathematica notebooks allow the designer to sprinkle documentation among the Mathematica expressions and their outputs. The text can be expanded or hidden on demand. At least one Mathematica text book has been written as nothing more than notebooks [Gray91]. One of our goals is to combine VLSI design and documentation in an identical way. However, the same goal is much more difficult to achieve in our domain. A Mathematica document is handled in a single-user environment, by a single tool, and contains only a limited set of data types. In

a typical VLSI design environment, a project is the product of multiple users running many Unix-based tools that manipulate many complex data types.

To handle this type of complexity, multiple approaches have been proposed in the past. In general, these rely on capturing the design history into data structures of different forms and combining it with the design data in different ways, including:

- *Design information management systems.* These organize the design history and data into hypermedia presentations, and provide means of quickly locating information related to the design. Systems in this class are organized around a single hypermedia tool that can display various non design-specific data types, such as text and pictures. Design data needs to be converted into the supported data types before being used by these systems. The problems of this approach is that the semantics of the CAD data is lost in the conversion process (for instance, when a VLSI layout is converted into a bitmap), and severely limits the capability of the searching and indexing tools. This also makes the update of this information very difficult. To perform an engineering change, once the desired design information is located with the system, one needs to start the appropriate design tool, perform the modifications, convert the design data into text and images, and, finally, re-index it. Nevertheless, the advantages of capturing the design history and data into these systems have been demonstrated in domains other than VLSI, such as mechanical design [Brad91]. Our goal is to create a similar system where design information can be searched and indexed using the design tools, hence overcome this limitation. This requires the development of an architecture for integrating the information manipulation tools with the design tools, so that the former can send commands to coordinate the operation of the latter. This would enable VLSI designers to look into layouts using layout editors directly, instead of bitmap viewers, and also make it possible for the information system to highlight a specific circuit in a layout as a result of a query.

- *On-line help and documentation systems.* We include in this category the developments of the major electronic design automation companies, which are incorporating text processing systems such as FrameMaker within their frameworks. This is meant to provide hypertext documents for on-line help, and better documentation for library components. Additionally, designers use the system to write their own designs documents, mainly datasheets. Our project also offers designers support for documentation, but its main purpose is something else. Our system does not aim to prepare manuals. Instead, it is conceived as a design team notebook that doubles as a front-end to the design tools. It implements a conceptual model for documentation that is intended to capture aspects common to all types of documents manipulated in electronic design.
- *Design process management systems.* These systems store definitions of the various operations that can be performed in a design environment and capture the design history to find dependences between the various pieces of data [Klei94]. Our research does not attempt to create another of these systems. The goal is to offer mechanisms in the design support infrastructure that make design process management an integrated part of the design information management system. Such an integrated system would simultaneously provide an operational view to the design data and help maintaining the design documentation.
- *Design rationale capture systems.* These systems create a new data structure with data not available in the design representations or history: the explanation of why the design decisions have been made [Carr91]. This type of information needs to be given explicitly by the designers during the design process. We believe this to be too intrusive for at least the initial phases of typical VLSI design projects. As in any other creative activity, designers use other types of reasoning other than logic reasoning when creating their designs [Mins86]. These include the exploration of analogies and trial-and-error starting from “ball-park estimates.” Our interest is in capturing of the design data, history and annotations into a spaghetti-like structure that could later be refined by the designers into documentation understandable by other designers. For this reason, the body of

research on design rationale is largely complementary to our work on tools for integrated design and documentation. However, we still see design rationale capture tools as a system component that could be useful for some types of electronic design, if integrated with an electronic design system like Henry.

- *Design collaboration support systems.* These systems create applications that build on the metaphor of an engineering notebook to create shared workspaces to which designers can add design information and also any other type of design-related data. The resulting electronic notebook is then seen as a medium for sharing, locating and reasoning about design information [Toye93, Gorr91]. These systems also use hypermedia to create presentations of the design data and its documentation, and are conceptually very similar to ours. Henry also includes an electronic notebook, designed as an editor of hypermedia documents. However, our research goals are not the same. Henry's notebook is the first tool in this class conceived to specifically support VLSI designers. We focus our research on active documents on the interactions between the notebook and various types of design tools and process management services used in VLSI design environments.

In the remainder of this chapter we analyze in more detail previous research works in the fields enumerated above and existing software systems related to the theme of our dissertation. We organize the discussion as follows:

<b>Section</b>	<b>Describes</b>	<b>Page</b>
2.2	Operating environments and design frameworks used by electronic systems designers.	22
2.3	Documentation Processing Systems.	29
2.4	Hypertext and Hypermedia Systems.	31
2.5	Design Rationale Capture Systems.	38
2.6	Computer Supported Cooperative Work.	40
2.7	Summary of related research and technologies.	42

## 2.2 Operating Systems Services and Application Frameworks

In a computer system, an applications framework stands between the applications and the operating system and the set of standard programs and libraries bundled with it. It provides common services required by the tools of an application domain not provided by the operating system, such as a common database supporting domain-specific data types.

Of special interest to us are Electronic CAD Frameworks, or simply CAD Frameworks as they are known to electronic systems designers. Recent surveys of the most important concepts associated with electronic CAD Frameworks have been published recently [Barn92, Harr90].

Figure 2.1 shows the common reference model adopted by software engineers for application frameworks [Nati90]. A complete design framework offers several common services, including:

- *User interface libraries*, to be used by the all the framework applications to ensure a common look and feel.
- *Message services*<sup>1</sup>, for communication between the tools and framework services.
- *Design data management and representation*, to support the domain-specific data types and control concurrent accesses to the design data.
- *Design process management*, for controlling the execution of design-related activities.

These services are targeted to multiple classes of users: designers, tool programmers and framework system administrators.

There are various commercial frameworks attempting to implement this architectural model. Examples include PowerFrame from DEC, DesignFramework from Cadence, and Falcon from Mentor Graphics. Several organizations work on reference implementations

---

1. sometimes also called inter-tool communication.

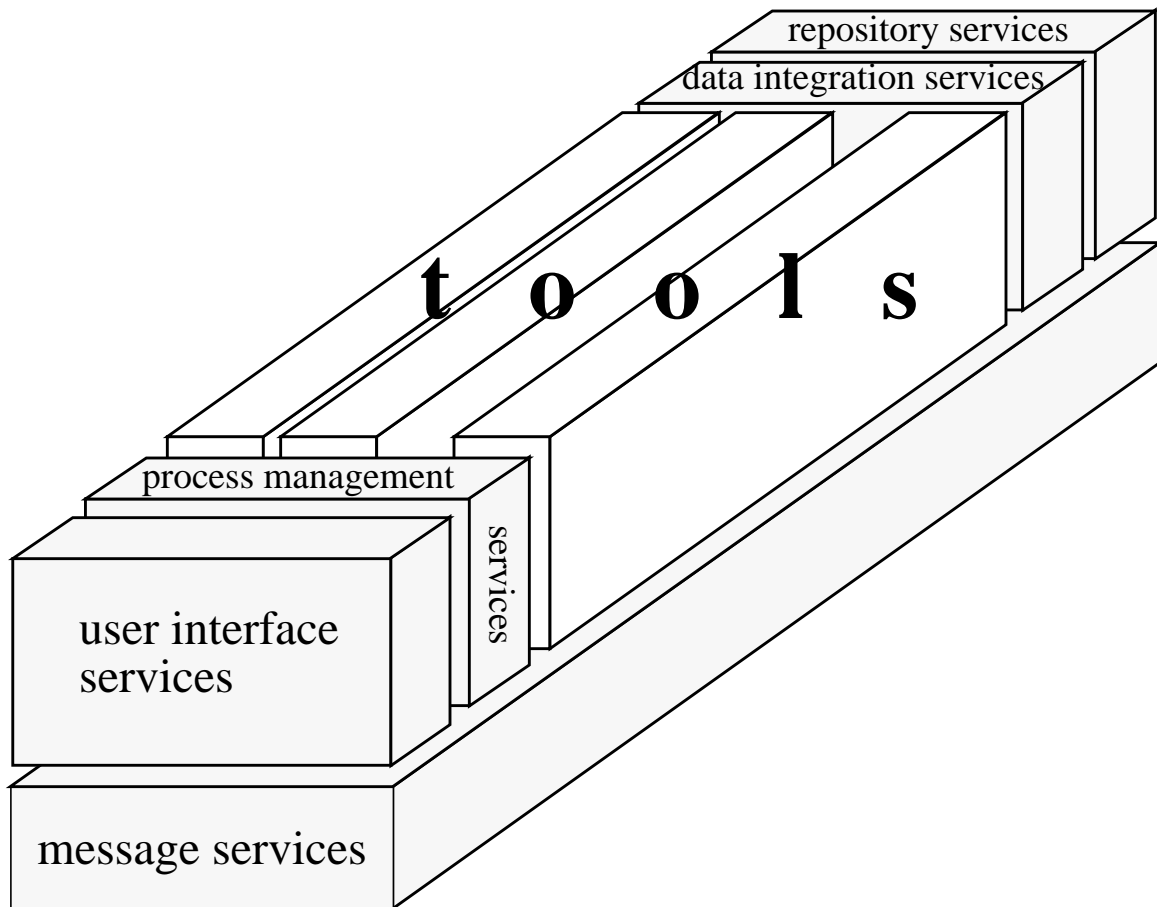


Figure 2.1 Framework Reference Model

This is the reference model for an applications framework. It is accepted by several standards organizations, including CFI. It has been applied in various domains, including computer-aided software engineering and computer-aided design of electronic systems. The model emphasizes the use of interchangeable “plug-and-play” tools that use a common set of services.

or the standardization of the interfaces between the tools and the multiple framework components, the most important being the CAD Framework Initiative (CFI) in the U.S., and the JESSI Framework in Europe.

Frameworks have also been proposed in other design-related domains. Of particular relevance is the research in software engineering environments and its application to CASE<sup>1</sup>,

1. CASE — Computer Aided Software Engineering.

because of its similarity with electronic CAD. The common framework reference model presented above was initially developed for software engineering environments. However, despite many similarities, there have been no records of successful adoptions of CASE frameworks for VLSI design, nor of notebook-based interfaces to a CASE environment. To some extent, this is because software is often *self-documenting*, with extensive text documentation sprinkled throughout the code. In fact, this idea of embedding code and documentation has been explored long ago in this domain, with environments like Knuth's WEB system [Knut84], which he used to develop the TEX text formatting program. What we intend to provide with active documents is a similar capability, to add documentation to the data describing the artifacts and manipulate them with an integrated set of tools.

There is not a static interface between a computer's common software environment and an applications framework. Over time, the common software environment of every computer evolves to support new standardized services previously covered by frameworks. Services like inter-tool communication, which are provided by individual CAD frameworks today, are now part of the next generation of commercial operating systems and will be common to all frameworks.

The rest of this section describes reviews the standard software environments and design frameworks adopted by the generality of VLSI designers. Then, we describe design process management systems in more detail.

### **2.2.1 COSE and CDE**

COSE<sup>1</sup> is a consortium of UNIX vendors that attempts to define a common standard for this operating system. COSE is addressing multiple areas in computer systems, including networking, operating systems and computer interfaces. UNIX is clearly the dominant platform for VLSI CAD software today. COSE standards have been endorsed by the CFI, the association of CAD software vendors and users.

---

1. COSE — Common Open Software Environment.



Of the technologies under development by COSE, CDE<sup>1</sup> is the one that will have most impact on our application and is closest to reaching the commercialization stage. CDE is a set of many diverse components, including several closely related to our research on active documents:

- *A common user interface look and feel.* This is essential for our active documents, which are intended to integrate many applications under a common manipulation paradigm.
- *Message services* for the communication between the tools in the desktop.
- *Session management and remote application invocation*, to start design and documentation tools from active documents when required.
- An hypertext-based *help system*.
- *A visual scripting language*, to quickly create programs using all the above software.

For the development of the Henry System we used either software components providing similar capability or developed our own. Some of the components, such as the session manager, could be simply replaced once CDE is finally released. Other services could make use of the new COSE services, but would still require the use of some of our extensions. For instance, the COSE message services are intended for communication between desktop applications. The Henry System supports a much more general inter-application communications model, from messages between the tools running in a designer's machine to client-server communications between independent organizations on a wide-area network.

### 2.2.2 CFI

The CAD Framework Initiative has been focusing on the definition of a common architecture and interface standards for the various elements of an electronic design applications

---

1. CDE — Common Desktop Environment.

framework [CAD93, CAD94]. This includes, among other, methods for inter-tool communication, design representation and design process management. There is also one subgroup interested in standardizing component information representation. All these efforts complement our research. Integrated design and documentation would never be conceivable without common interfaces to access design data and control tools executions.

Currently, it is very hard to have tools from different CAD frameworks working together, as there are only very poor interfaces between them. Users perceive this as the strongest limitation of these systems, because they often choose to adopt tools from different vendors for handling specific design steps. In a typical design environment one can easily find a VHDL simulator, a logic synthesis system, or a circuit layout system, each supplied from a different vendor. As a result, the common reference model adopted by CFI does not correspond to the observed reality: design environments often have multiple incompatible instances of the same design support services.

The Henry System supports an environment based on a framework with an architecture which is derived from the CFI reference model. This is extended into an environment where design objects, documentation, and the tools that manipulate them are all first-class objects handled seamlessly from the same interface. In addition, Henry also extends the model of a design environment based on a single framework, into one composed of multiple heterogeneous frameworks that communicate information through active documents.

### **2.2.3 Design Process Management Systems**

The design process, also known as design flow or design methodology is the procedure or set of procedures used to complete a design. Kleinfeldt et al., recently surveyed the state of the art in this domain and compared the existing systems [Klei94]. We adopt their terminology throughout this dissertation.

Design process management refers to the computer aids for helping designers define, execute and control the methodology. It is one of the main components of a modern CAD

framework. The basic entities in the conceptual model of a design process management system are *tools*, *tasks* and *flows*:

- A *tool* is a program execution. It may be the execution of a shell command, that triggers activation of a script or an application program, or a command to a running program, through a remote procedure call.
- A *task* corresponds to a definition of a sequence of tool invocations. It contains definitions of what tools to use, what data is given as input and produced as the output. Tasks are used to encapsulate design activities without the designers having to indicate the invocation details of the tools.
- A *flow* is a definition of a sequence of tasks. This is usually specified through data dependencies, as in a UNIX makefile. But it can also be temporal, when obtained from a log of tool invocations.

There are various examples of design flow management systems that let designers annotate the captured design history and use it as documentation. A prototype system named VOV at U. C. Berkeley used a record/playback approach to manage the design process. The flow dependencies are collected into a design traces, which can then be used to illustrate a given design methodology [Caso91]. This can also be seen as a constrained, but automated way of building design documentation. The user interacts with the example design, changing the files and re-running the tools to visualize the effects of the change. Moreover, this is documentation that can be re-used: designers use the design files as templates for the specific project at hand, modifying them as necessary.

In Papyrus, another process management system from Berkeley, there is a facility to produce design meta-data from the design process history [Chiu92]. For example, a logic minimizer, such as *espresso*, can output either tables or algebraic representations. From the analysis of the options given to invoke a the tool, it is possible to derive the types of the produced objects. In addition, if the environment contains design estimating tools to compute the power, area or delay, the analysis of the history data can be used to find what

modules are instantiated and automatically compute these parameters at the chip level. We believe that the same technique could also be used to produce human-readable documents, illustrating to designers important characteristics of a project's data.

At CMU, research has focused on the elaboration of models for representing the design process, and development of task management and process planning tools [Bush89, Dani89, Jaco92]. Although this will be helpful for designers, it also implies adding another layer of abstraction and further complexity. Process management tools do not offer the necessary aids to make design details more understandable. A system like Henry would be the necessary complement. On the other hand, we would like to be able to use these tools in a more information centric design system where they could be seen by the designers as agents for assisting in the generation and maintenance of the design documentation [Riec94].

### **2.2.3.1 What is a Design Tool?**

Unfortunately, the term tool becomes overloaded with different but related semantics, when we put together the multiple research areas related to active design documents. For a electronic designer, a tool represents any CAD program in general. This does not always mean the same to researchers in design process management. In the early days, these started by assuming that only non-interactive programs executions would have to be managed. However, the new generation of interactive CAD programs can receive commands from remote procedure calls. Methodology management systems now need to interpret each of the commands as what was formerly understood as a tool.

Interactive programs can also have several operating modes associated with the mouse commands. For user interface designers, each of these modes is also called a tool. Even some CAD programs use the term tool to designate these modes. For instance, the Magic layout editor [Oust85] defines several modes, such as a *box tool*, to operate on the rectangles defining the various layers of the layout, and a *irsim tool*, to interact with the data pro-

duced by the logic simulator that it runs as a sub-process. In each of these modes, the user then has several commands available.

In summary, a designer invokes application programs to operate on the data. Interactive applications may have a single or multiple operating modes. In each mode, several commands are available for execution. Due to overloading, the term tool can designate an application program, an operating mode, or a command. In this dissertation, we usually employ the term in the designer's assertion, as a CAD application that he can control to operate on design data. However, when we discuss interactions between design process managers and the tools that manipulate active documents, the term should be understood as viewed in design flow management.

### **2.3 Documentation Processing Systems**

Our concept of active documents is a generalization of the concept of on-line structured documents. The book edited by André et al. [Andr89] reviews the history of systems for processing on-line structured documents, their conceptual models and existing standards.

On-line structured document processing systems are designed primarily to produce documentation on paper. Active documents can have buttons, to jump to different pages in other documents, and insets displaying the windows of independently running programs. However, they inherit from structured documents all the fundamental concepts, such as the tree-like data structures representing the documents' hierarchies and a page rendering model.

The latest generation of documentation processing systems is based on direct manipulation user interfaces [Schn82]. These are characterized by 1) continuous representation of the objects of interest and 2) physical actions on the objects with immediate visual feedback. State of the art documentation processing systems in this category are FrameMaker, Interleaf, Microsoft Word and Word Perfect.

The conceptual models used for representing documents in these systems are all very similar. A good evidence of this assertion is that all these systems now support or have announced to support the capability to import and export documents in the SGML format, is the ISO standard for on-line documentation representation. We review this format in more detail in the rest of this section.

### 2.3.1 SGML

SGML (Standard Generalized Markup Language) is a relatively old standard format for electronic documents [fS86]. The book by Goldfarb provides an extended review of this standard [Gold90]. SGML is a language that can describe all the structuring elements of a document represented electronically, such as heading types, character types and paragraph formats, and mathematical expressions. A SGML document contains two parts:

1. a DTD<sup>1</sup>, which defines the style of the document. This contains the definitions of the structuring elements used in the document, or *tags* in SGML terminology.
2. The tagged text of the document.

SGML seemed to be doomed to become a standard supported only by a few specialized applications until very recently, when HTML<sup>2</sup>, a format derived from it, was adopted as the primary on-line document format for the World Wide Web [BL94b]. HTML is an extended subset of SGML [BL93b]. It adopts a pre-defined SGML DTD and extends it with new tags to be interpreted by the document processing system as hypertext links. Recently, the popular WWW system introduced an HTML extension, where new tags have been defined to provide a simple electronic form representation capability [Andr93]. With these extensions, some of our concepts for active design documents could be represented. However, more would be required to support it completely. Our model involves

---

1. DTD — Document Type Definition — the document structure definition part of a SGML document.

2. HTML — Hypertext Markup Language

interactions between multiple tools to create document presentations. For instance, we would like to include in an active document a simulation deck and commands to produce an animation. When displaying the document, the simulator would be called to process the commands, read the deck and display the waveforms in a figure. As HTML only supports text and images, this type of interaction cannot be represented in the document.

One of the early applications intended for SGML was its use as a system-independent document representation format that could be supported for many years. In the electronic design domain, these characteristics made it well suited for producing documentation of designs of military type products that would have to be maintained for long periods. The electronics industry also used it as a way of distributing components documentation in much cheaper ways than on paper, using CD-ROM and on-line databases. There is a proposed standard for a SGML DTD designed to represent the structure and data types included in electronic data sheets. This is known as the Pinnacles DTD [ATL94] and is being considered for adoption by the CFI.

## 2.4 Hypertext and Hypermedia Systems

A hypertext system allows authors or groups of authors to link information together. The term was coined in the early 1960s by Theodore Nelson to describe the idea of *non-sequential writing*. The mechanisms available to authors to create these connections include annotations to existing texts, notes directing readers to other documents and paths through bodies of related information. Recent tutorials and research surveys on hypertext technologies include the book *Hypertext and Hypermedia* by Nielsen [Niel90] and an article by Conklin [Conk87].

With a computer system, the process of activating explicit connections, or *links*, between portions of documents can be automated. An hypertext is not a simple a collection of information. It also contains an implicit navigation process defined by its authors. This contains in general many possible paths, from which one is chosen by the user while browsing the information. In addition, hypertext systems that support multiple users can

be used to communicate and collaborate through annotations to the collection of shared information. Hypermedia is an extension of hypertext, where the body of information can be any media type, such as audio, video or static and animated graphics.

Hypertext techniques are not exclusive to hypermedia systems. In recent years, hypertext is a technology that has progressively penetrated virtually all desktop computing operating systems and applications. All major direct manipulation-based text processing systems mentioned in the section above support hypertext links, either directly or through some other mechanism that gives authors the capability to create the same effect.

The concept of an Hypermedia Framework, an environment supporting hypertext links between media of many types manipulated by distinct tools was pioneered by systems like Intermedia, developed at Brown University [Meyr86, Yank88, Haan92]. If we see the data types available in the design environment as new media types, we could look to this data as a body of information that could be annotated using the above described hypermedia techniques. A CAD framework extended to support them could then be also seen as an hypermedia framework with integrated support for electronic design specific data types. One of the main goals of our research is precisely that of creating this extension.

There are multiple reports on early experiments combining hypertext and CAD applications, such as the work of Delisle and Schwartz [Deli86]. However, all the early hypertext systems suffer from many limitations, due to the almost unlimited amount of information that can be accessed from any link. In a 1988 paper, Halasz enumerated the main unresolved issues that the next generation of hypermedia systems should address [Hala88]. These include support for search and query of hypermedia information, *virtual structures*<sup>1</sup> for dealing with changing information, and collaborative work. Malcolm et al. describe in more detail the reasons why hypermedia systems do not support the needs of engineering enterprises. According to them, “current hypermedia tools do not support the needs of col-

---

1. This concept of a virtual hypertext structure, which is dynamically built for the user as he or she browses the information corresponds directly to our interpretation of active document concept.



laborative work groups in distributed heterogeneous environments and cannot be integrated into the existing and planned computing environments” [Malc91]. The limitations enumerated are 1) poor inter-operability of existing hypermedia systems, 2) absence of support for a shared workspace model, and 3) a distinction between *authors* and *readers* of design documents. In our view, these still remain unsolved today.

Our research on the integration of design and documentation data is an attempt to address these problems from the electronic design perspective. In the rest of this section we describe in more detail a few related hypermedia systems that attempt to provide answers to some of these limitations.

### 2.4.1 Open Hypermedia Systems

A key contribution to the integration of pieces of information produced by independently developed systems through hypermedia is the concept of *open hypermedia*, pioneered by the Sun Link Server [Pear89]. This research product eventually gave origin to the Tooltalk Message Server [SunS92], which is now part of the standard environment defined by COSE/CDE and CFI. The Link Server enables the creation of links between objects in independent applications using a common protocol. It provides a storage mechanism for link information and communicates with the applications. Each application is responsible for unique object identification, storage and provide the communications interface with the link server. We illustrate its operation through an example. Consider a situation where an application displays circuit layout and another shows documents of the specification of each of its components. In an hypertext system using both tools to show documentation associated with each piece of layout, we would have to add a remote procedural interface to them to accept the commands of the Link Server protocol. Each circuit in the layout would be annotated with the name of the associated document. Activation of the link would consist in a command given at the layout tool, which would be translated into a message to the Link Server containing the identification of the document. This would in turn send a command to the text processing tool to open the document.

The major limitation of this approach is that it requires the modification of the application's code to support the link server's communication protocol. If an application already has a remote procedure call interface, a new one must be built complying with the message server protocol. However, in most cases system integrators do not have access to the source code of the tools. As a result a solution like this would be practical only if all application developers decide to support it.

Another approach, which does not require applications to have a remote command interface, or any other modification to their code, consists in intercepting some of the mouse commands sent to the applications and have them interpreted by the link server. This has been attempted in systems like Microcosm, developed at the University of Southampton, UK [Foun90, Davi92]. We describe their operation using the same scenario as above, involving a circuit layout and a document editor. An hypertext system including both tools would start the layout editor and set it up to intercept any mouse clicks on its window. The link database would contain a table for translating each mouse location into layout coordinates in the first application and from these into the document names of the second. A mouse click on the layout editor would be intercepted by the Microcosm link server, which would translate into a file name which would be given as argument in an open command to the document editor.

This has the additional advantage of offering a consistent user interface for activating links, common to all tools. Another significant advantage is that all the information relative to the hyperlinks is stored in a common database, independent of the tools.

However, this approach has major drawbacks that make it unusable in our application domain, which demands support for simultaneous design and documentation. In the above example, a simple modification to the circuit layout would render the information in the link database obsolete and very hard to update. Another limitation is that building the links database on the link server is much more complicated. In the example above, a com-

plex application-specific program, would be required to generate and maintain the link translation tables.

Our approach stands in the middle. Unlike Microcosmos, the Henry System requires applications to have a remote procedure call interface to invoke its commands. From the designer's point of view its operation looks identical to the Sun Link Server. Information about links is also partially kept in the form of annotations or markers in the design tools data or documentation files. The architectural difference between the Henry communications server and the Sun Link Server is significant to tool integrators: the former is designed as an extensible server, capable of handling multiple inter-tool communication protocols. This makes it possible to integrate tools defining their own remote procedure call interface without modifying their code.

## **2.4.2 Compound Document Manipulation Architectures**

A link server like Tooltalk is only one of the necessary operating system components required to support the creation of hypermedia systems involving any of the applications available in a computing environment. To create active documents built of a tool ensemble of GUI<sup>1</sup> applications, it is also necessary to have a mechanism to compose them. The applications should appear to the user as embedded insets in the pages of an on-line compound document.

Support for this type of documents is already available on personal computers and is slowly being added to UNIX workstations. An example of a complete set of services is the second version of OLE<sup>2</sup>, available in the Windows operating system [Mic93]. A similar system is also available for Apple personal computers. There is a recent initiative to make this architecture an operating system independent standard [App93]. The latest version of the X Window System (release 6) adds new extensions that would enable the creation of

---

1. GUI — Graphic User Interface.

2. OLE — Object Linking and Embedding, a Microsoft, Inc. product.

applications that contain other applications running in its sub-windows. These, in conjunction with the Tooltalk message server and other new UNIX services could be used to create on-line compound documents.

For our research, as we were aware since the beginning that these undergoing development efforts, we decided not to develop a new model for creating composite embedded documents. Instead, our research focused on design-specific aspects of the integration of heterogeneous information, such as the integration of design and documentation activities through active documents. Our model for presentation of active documents assumes that one of the above mechanisms for embedding applications into hypermedia documents is available, although this is not supported by our prototype system.

### **2.4.3 Notebooks**

Support for on-line compound documents, when available in the computer platforms used by VLSI designers, will find a direct application in electronic design notebooks. An electronic notebook is a replacement for the notebooks used by scientists and engineers in many domains. We developed a prototype of VLSI design notebook as part of the Henry System.

Electronic notebooks are hypermedia systems customized to a specific application, supporting viewing and annotation of the domain-specific data formats. These offer dramatic new possibilities, when compared to their paper-based ancestors. They are not restricted to containing data of relatively small sizes; massive amounts of information, from data associated with the design (e.g., product information, data sets) to the design objects themselves can be accessed from within the notebook. Once information is stored in a computer-based notebook, it becomes much more available for sharing and ultimately for reuse. However, there are some limitations of the electronic media that make paper notebooks still attractive, such as their portability and ease of use.

The following are examples of projects in other design domains that also involved the development of an electronic notebooks:

- Virtual Notebook System (VNS) originally developed at Southern Methodist University for medical researchers [Gorr91]. This is the earliest documented attempt to design and use an electronic notebook. VNS exploits the notebook metaphor for its user interface, allowing users to organize heterogeneous data as if they were writing into a paper notebook. However, the single user notebook is extended to a distributed, multimedia hypertext system. VNS was conceived as a set of tools for information acquisition, sharing, and integration within collaborative research groups.
- Electronic Design Notebook (EDN), was developed at GE Laboratories within the DARPA Initiative in Concurrent Engineering (DICE) [Ueji90]. The purpose is to capture and preserve the design intent of an engineering project and to support cooperative work with computers.
- SHARE is another project, in the mechanical engineering domain, developed under the same initiative [Glic93, Toye93]. The electronic notebook explores the idea of a paper notebook metaphor to support sharing of design-related information and communication within the project team.

The notebook we developed for the Henry system, called the Navigator has many similarities with the above designs. However, the research goals we set for the Navigator are substantially different. We do not see the electronic notebook as the main front-end of the design environment, from which information is entered and searched. From Henry's perspective, the Navigator is simply another tool, specialized for browsing the web of information related to the design and creating annotations to that information. We were interested in studying its operation in conjunction with existing VLSI design and documentation tools, and designed the Navigator to make full use of Henry's infrastructure. Another novel aspect of the Navigator is its new user interface, which enables users to use the notebook as readers and authors simultaneously.

On the other hand, the Navigator does not provide any special support for user communication using the documents as the medium, or for concurrency control mechanisms for simultaneous authoring of the same active document by elements of a design group. This aspect is in part covered by the above notebooks and by the large existing body of research on engineering design data management.

## 2.5 Design Rationale Capture Systems

Design rationale (DR) designates the reasoning supporting the design. The goal of DR capture systems, also called *argumentation systems*, is to create explicit representations of the decisions made by a group of designers. Design rationale is a form of description of a design artifact that augments what is available in the normal documentation. In addition to the structure and function of the artifacts, it attempts to describe the reasons *why* the design choices have been made. An up-to-date survey of the research in this domain is available in a recent special publication coordinated by Carrol and Moran [Carr91]. The motivations indicated by these authors for constructing explicit design rationale are: 1) to support reasoning processes in design, 2) to facilitate communication among the various participants in the design process, and 3) to further the accumulation and development of design knowledge across design projects and products.

Argumentation systems, use some form of extension of the IBIS<sup>1</sup> model for capturing the deliberations of a group of designers, developed by Rittel in the 1970s [Ritt70]. The data structures of a DR capture system can be seen as a way of producing structured documentation. Most of these systems also have a hypermedia-based user interface, similar to that supported by Henry. The following are examples of argumentation systems developed for other domains:

---

1. IBIS — Issue-based Information System

- PHIDIAS is a system for environmental design, developed at the University of Colorado [McCa90]. It combines features of a CAD system with elements of an argumentation system, i.e., a system that tracks design alternatives and the rationale for their resolution. PHIDIAS makes it possible to structure the discussions about a design. These are stored in a hyperdocument, along with graphics and text nodes.
- gIBIS is an hypertext system based in the IBIS model, developed at MCC [Conk88]. This has been used at NCR in a software development experiment [Yake90]. The authors' analysis of the experiment indicate that reviews of the captured rationale made a number of problems evident relatively early in the design process. This more than paid for the cost of capturing and organizing the rationale.

Despite the many similarities, our goals for developing Henry are substantially different from these previous efforts. We do not intend to capture the rationale explicitly, but only to provide designers with a system that helps them organize the information related to the design. In our view, the interest in forcing a structured design methodology upon designers for capturing the rationale is arguable: the developers of PHIDIAS report that documenting design decisions as somehow separate from their realization can be disruptive. It can also prevent the smooth flow of reflective exploration [Fisc91].

Recently, work in this area has been exploring new directions motivated by these that depart from the IBIS model of organizing the design decisions process. Conklin and Yakevovich, based on their past experience, recently proposed “a new approach to design rationale that emphasizes supporting the design process in such a way that a trace of the rationale is captured with little disruption of the normal process” [Conc91]. This corresponds directly to our main goal of combining design and documentation in VLSI design, for capturing the rationale nonintrusively.

Another direction was pursued by Garcia, which uses an expert-system based approach applied to heat and ventilation systems design [Garc92]. In this system, an interface based on *active design documents* is also used. The basic motivations to employ active docu-

ments in her application (design) to a design-rationale capture approach are similar to ours: the need to have 1) low documentation overhead and 2) easy access to relevant information. However, her research is on developing a model of the design from which documentation can be generated and checked for consistency. She calls these documents Active Design Documents (ADD), because they evolve dynamically with the design specification, as ours do. However, this notion of active documents has little in common with our concept, which is derived from hypermedia. Her documents are generated automatically from the designer's interactions with a single tool, while ours result from a user controlled authoring process involving many tools.

## 2.6 Computer Supported Cooperative Work

Computer Supported Cooperative Work (CSCW) is an umbrella for a group of related technologies that are used to support collaboration. Several recent surveys give an in-depth introduction to research in the field [Grud94, Grud91]. Another term used in conjunction with CSCW is *groupware*. The distinction between both is controversial. We understand groupware as a designation for the multi-user computer applications used in CSCW.

The most successful groupware application is electronic mail. Other applications include shared calendars and hypermedia systems for group communication. Lotus Notes is the most widely used groupware application.

In the design of the Henry System we did not explore the use of active documents primarily as a communication media. We use active documents as structured descriptions of the design information. These can be exchanged in communications, but *are not* seen as the communication tool. Hence, Henry is not currently a system supporting all the needs of a group VLSI designers. We assumed that user-to-user communication in a production design environment would be supported by groupware applications such as Lotus Notes, shared white boards, and video-conferencing. These could then be integrated when available in the UNIX machines that support today's CAD tools.



Nevertheless, we have adopted the communications models and the software of active mail systems in Henry for exchanging active documents between CAD systems. We describe these groupware applications in more detail in the rest of this section.

### 2.6.1 Active Mail

Active mail systems (also known as *computational mail* systems) are based on a very simple idea: instead of sending plain text in a mail message, what is sent is a computer program. At the recipients end, the mail handling software recognizes that a message is to be executed rather than read, and evaluates it. Messages containing these programs are called *active messages*. An example of common groupware application built upon active mail is a calendar system. A user sends a list of possible dates for a meeting to a group in a program embedded in a mail message. When the recipients activate the program, their calendars are consulted and a list of dates when the recipient is available is returned. After collecting all the responses, the meeting organized can then decide on the date and send another active mail message to automatically mark the calendars when received.

A discussion of the basic mechanisms and applications of these systems is given by Borenstein [Bore92]. The main problems related to the implementation of these systems are security and handling of heterogeneity. Security is problematic because these systems must handle unsolicited messages that may contain harmful programs. Supporting heterogeneity is essential, given the diversity of computer platforms where mail is read.

One way to address both problems is to use an interpreted programming language for the active messages. This is the approach followed by ATOMICMAIL, a computational mail system based in the LISP language [Bore92]. Recently, ATOMICMAIL evolved into a new version [Bore94], based on the Tcl language and the Tk graphic user interface toolkit [Oust94].

Goldberg et al. propose an alternative system for active mail [Gold92]. This is based in an entirely different mechanism: instead of programs, this system passes special messages

which contain communication ports. When the recipient calls a special program to process the message, it opens a connection to the indicated port and communication is initiated. As a result their system is a framework for *rendez vous* between groupware applications through electronic mail then a system for exchanging active messages.

In the communications architecture of the Henry system, we adopted the active mail model based on active messages for exchanging documents between users through electronic mail. Active mail is more general, since it does not require establishment of direct connections, and handles the problems of security and heterogeneity.

## 2.7 Summary

The Henry System attempts to unify two essential parts of the VLSI design environment: the design system and the documentation processing system. Until now, these have rarely been considered together. Our research is unique in that it is the first to consider the complete integration of the two, so that design and documentation activities can be combined into a single thread.

The design of a new integrated system combining the features of both domains would be an impossible task, given the complexity. However, from the evaluation of the related systems presented in this chapter, we conclude that many of the needed components could be adapted without modification. What is required is a set of mechanisms for facilitating integration and the creation of a new model for developing design methodologies supporting simultaneous use of existing design and documentation tools. We provide these mechanisms and methodologies in Henry, via its common interfaces based on documentation. The architecture of its support services and tools, and its implementation and evaluation, are the subject of the following chapters.

“The universe is looking less and less like a great machine and more and more like a great thought.”

— Ortega y Gasset

## **Chapter 3**

# **Internet System Software and Applications**

The Henry System exploits several large software systems for handling communications in the Internet. This chapter introduces the protocols and applications we adopted and discusses their utility for designers.

### **3.1 Introduction**

Conventional CAD Frameworks support a group of users in a local area network. The design process involves manipulation of information that is located in a single database. Henry was designed to support design processes involving the use of multiple Frameworks and on-line information services by independent groups. These exchange control and data in the form of active documents.

In a heterogeneous distributed design environment, adoption of commonly accepted communication standards is of fundamental importance. For an information service provider, the selection of a protocol is synonymous to finding the best way to make its data available to the largest group of consumers. Given the kinds of services and interactions we are

interested in supporting with active design documents, this can be implemented with client-server communications. To satisfy these requirements rapidly, we must adopt the protocols of the Internet, which is now the largest and fastest growing computer network. Virtually all computer platforms in use by systems designers today are delivered with customized versions of this software already installed.

The Internet also has the advantage of being supported by a vast library of publicly available support software. This has simplified dramatically the implementation effort of the Henry System prototype, to be presented in Chapter 7. A significant part of Henry's communication software was either directly installed from public domain software or slightly adapted to work with our environment.

In this chapter, we do not describe the protocols for sending electronic mail or creating virtual connections between processes running in different hosts. Good introductory books are readily available [Come91, Come93]. Instead, we discuss how message formats and the higher level protocols that are built upon the basic communications infrastructure can be used to exchange active documents, perform business transactions and make design information easily accessible to end-users.

The remaining of the chapter is organized as follows. In Section 3.2, we introduce MIME, the standard Internet format for transporting multimedia data. Next, we present Enabled Mail, a conceptual model for handling MIME messages containing programs that are directly activated by the system or the reader upon reception. In Section 3.4 we summarize the World Wide Web, a hypermedia system for accessing all the information available on the Internet. Section 3.5 enumerates services based on electronic commerce that we anticipate that will be available to electronic systems designers in the near future. Section 3.6 summarizes the main ideas.

## 3.2 MIME — Multipurpose Internet Mail Extensions

In the Henry system, we adopted the common format for messages exchanged between sites on the Internet. This consists of a set of RFC822-type [Croc82] message headers followed by a MIME<sup>1</sup> [Bore93] formatted message body. Headers define how to route messages to the destination and the data format used in message bodies. They were first developed for use in electronic mail with the SMTP<sup>2</sup> protocol [Post82], but have been adopted for NNTP<sup>3</sup>, the network news protocol [Kant86]. More recently, the same basic formatting convention was used in the design of HTTP<sup>4</sup>, a client-server protocol for navigating hypermedia documents distributed across the Internet [BL94].

MIME defines how multi-media data is encoded in the messages transported by these protocols. MIME defines a set of conventions that communications applications use for encapsulating different media into a single message and defining the data types being transmitted.

By extending these conventions, it is possible to transport design and documentation-specific files and pass them automatically to the appropriate tools for processing. For instance, to transfer a layout in the format used by the Magic editor [Oust85], the sender and recipient would agree in defining a new application-specific name for that format. Following the established convention, this type would be called *application/x-magic*. With a properly configured electronic mail software, a message containing a MIME encapsulated layout would cause the automatic invocation of the Magic editor when the message was read by a designer.

- 
1. MIME — Multi-purpose Internet Mail Extensions, the extensible Internet standard for formatting electronic messages containing not only text but other types of data.
  2. SMTP: Simple Mail Transfer Protocol
  3. NNTP — Network News Transfer Protocol
  4. HTTP — Hypertext Transfer Protocol, currently in the approval process as an Internet Draft.

MIME also supports the encapsulation of multiple data-types in different sections of a single message. These are called *multipart* messages. For instance, for transferring a Frame-Maker document and a Magic layout, a MIME multipart message would be constructed containing two new content types, defined as *application/x-frameset* and *application/x-magic*. A MIME message containing these two types is shown in Table 3.1 .

### 3.3 Enabled Mail

The MIME extensions to the Internet message format alone are not sufficient to support the needs of designers. MIME assumes that only one operation can be associated with each content type. This works in most situations, where the operation that is defined is an invocation of the tool used as the *previewer* used for that type. In a VLSI design environment, there are many possible tools and operations that a designer may want to perform on any specific data type. For example, a netlist can be loaded into a simulator, a schematics editor or a design rules checker.

Enabled Mail (EM) is an extension of the traditional message handling model used in electronic mail [Bore94]. It proposes a new conceptual model of the message handling process, from end to end. EM defines mechanisms for sending MIME documents and programs for remote evaluation, and specifies conditions for delivery and time of execution.

To control which operations will be applied to a data type, we need such an extension. We have adapted an initial version of Enabled Mail (EM) for use in the Henry System. At this point, only a preliminary version of the Enabled Mail software and documentation has been distributed. The Henry System is one of the first applications testing the usability of this model and accompanying software. We adopt EM's conceptual model and adapted its support software for handling inter-tool communications. We expect that in the future EM will be submitted as an Internet RFC<sup>1</sup> and become a standard.

---

1. RFC — Request For Comments, the name used to designate Internet standards.

Description	Example MIME Message	
<p>RFC822 Headers</p> <p>MIME-defined RFC822 headers specify the content type of the message.</p> <p>Empty lines separate message headers from contents</p> <p>Prologue, any text can go here.</p> <p>The FrameMaker document, is between the <code>xxxxx</code> delimiters.</p> <p>It is sent in a similar structure:</p> <p>(1) headers, defining the content-type and encoding of data, (2) one empty line, and (3) the encoded document.</p> <p>The Magic layout is sent in a similar way.</p> <p>As magic represents the layout in text format, there is no need to encode it.</p> <p>end of <i>multipart/mixed</i> data</p> <p>Epilogue, any text can go here.</p>	<pre> From: Mario &lt;msilva@CS.Berkeley.EDU&gt; To: Randy &lt;randy@CS.Berkeley.EDU&gt; Subject: layout of the pseudo-NMOS gate  MIME-Version: 1.0 Content-Type: multipart/mixed;           boundary="xxxxx"  -----  Randy, here's the Magic layout and the FrameMaker document explaining the design.  --xxxxx Content-Type: application/x-framesmaker Content-Transfer-Encoding: base64 -----  HsGKasdaJUdsfjUYTdfkZKJddHwieueKJH Kd kJLKJHLsdfsjdkdILkjhLKJdHLdfgKbyHL KJH KJHadjhgsdJssJdLddHGsdPUJHgkjhG ksUYTkjhgUIYJHTERWjhgOIUkjhPOIUjkG LJuweKJHGklJHGklOIUkjhkPOPIkljkPIOP  --xxxxx Content-Type: application/x-magic;           charset=US-ASCII -----  tech scmos timestamp 719814828 ...etc  --xxxxx-- Bye, Mario. </pre>	<p>— Headers —</p> <p>— Message Body —</p>

Table 3.1 Format of a MIME message

MIME extends RFC822 electronic mail messages to transport any data type in such a way that it can be processed automatically by mail readers. A user reading the mail with a properly configured MIME-capable mail reader would see the contents of the FrameMaker and Magic data displayed automatically by the tools.

EM defines a new message format also based on MIME. A new message contents type, called *multipart/enabled-mail*, is used for active messages. A *multipart/enabled-mail* message contains by definition two parts. The first can be of any MIME data type. The second part, contains a program that, when invoked, has access to the data in the first part and can perform operations on that data.

The language proposed for use in the second part is Safe-Tcl, a system-independent interpreted language derived from Tcl [Oust94], a generic interpreted extensible language developed for being embedded in interactive tools. Safe-Tcl is an “extended subset” of the Tcl language. All the features of the Tcl language that could compromise the security of a system that automatically evaluates messages received from indiscriminate origins is removed. Safe-Tcl also defines new commands for manipulating and storing the data in the first part of *multipart/enabled-mail* messages.

The MIME type defined for scripts in Safe-Tcl language is *application/Safe-Tcl*. This type can also accept parameters specifying when evaluation of the active message will take place. In most cases, we use *evaluation-time=delivery*, meaning that the Safe-Tcl script will be run as soon as the message is received. Table 3.1 shows how the same data used in the example of the previous section could be sent in an EM message.

### 3.3.1 Enabled Mail and Security

Of special interest is Enabled Mail’s support for secure automatic activation of messages upon delivery. By using the Enabled Mail extensions to standard Internet mail, we can submit commands and programs for remote evaluation by electronic mail.

This is achieved by having two language interpreters in the Safe-Tcl program evaluation environment. The first is an untrusted interpreter of the Safe-Tcl language. The second, is a trusted interpreter that runs the full Tcl language. A secure environment for passing active messages is obtained by running the scripts included in the messages in the untrusted interpreter. When access to the design environment’s data or tools is needed,



Description	Example Enabled Mail Message	
<p>RFC822 headers.</p> <p>MIME-defined RFC822 headers.</p> <p>Empty lines separate message headers from contents.</p> <p>Prologue.</p> <p>The FrameMaker document, and the Magic layout are sent in a multipart/mixed message as before, but now encapsulated inside the first part of the top-level message</p> <p>The 2nd part of the EM message contains a script of commands in the Safe-Tcl language.</p> <p>End of EM message. Epilogue.</p>	<pre> From: Mario &lt;msilva@CS.Berkeley.EDU&gt; To: Randy &lt;randy@CS.Berkeley.EDU&gt; Subject: active msg with pseudo-NMOS gate  MIME-Version: 1.0 Content-Type: multipart/enabled-mail;             boundary="yyyyy"  -----  Randy, here's an active message that installs the design data you requested into your database  --yyyyy Content-Type: multipart/mixed; boundary="xx" -----  --xx FrameMaker document --xx Magic layout --xx--  --yyyyy Content-Type: application/Safe-Tcl;             evaluation-time=activation -----  Safe-Tcl script to install the data in the recipient's database  --yyyyy-- Bye, Mario. </pre>	<p>— Headers —</p> <p>— Message Body —</p>

Table 3.2 Format of an Enabled Mail message

Enabled-Mail messages include MIME data and a script with commands in an interpreted language. A user reading the mail with a properly configured MIME-capable mail reader would have the script to execute automatically upon reading the message. The script could include for instance the commands to install the received data into the design database after user inspection.

control is passed to the trusted command interpreter. The trusted interpreter performs authentication and control of system resources given to the script running in the untrusted interpreter.

### **3.4 World Wide Web**

The World Wide Web (WWW or W3) is a set of protocols, an addressing scheme and data formats for accessing all data available on the Internet [BL94b].

The WWW defines a collection of pieces of information in which all items have a reference by which they can be activated, stored or retrieved, called a URL<sup>1</sup> [BL93a]. Items could be any file on any host of the Internet, a user's electronic mail address or a program invocation. This information is handled by a collection of clients and servers. The former are the computers (or, more exactly, the programs) used to access or activate the information.

When a client starts, it displays a hypertext document, with links to other pieces of information, including other documents. The primary document format of the WWW is HTML<sup>2</sup>. This is derived from SGML<sup>3</sup>, the ISO standard for representing on-line documentation [fS86]. Clicking in highlighted areas of the document causes the client to retrieve another information object from another computer, the server. The retrieved data may be another hypertext document, which enables the execution of a navigation process.

In the WWW, servers and clients can be of many types. The most common types of servers available on the Internet, such as those that use electronic mail, electronic news, and network file access protocols can be accessed from WWW clients.

---

1. URL: Universal Resource Locator

2. HTML: Hypertext Markup Language

3. SGML: Standard Generalized Markup Language.

In addition, the WWW defines HTTP, a specific communication protocol optimized for transferring documents as the result of the activation of hyperlinks. Clients can access all information available from different servers using a graphical user interface. Retrieving a document or a design file can be done with a few mouse clicks.

The HTTP protocol, the HTML document format and the URL addressing mechanism are currently being considered by the Internet organizations for approval as new standards. The WWW uses the MIME conventions to tag data formats passed between documents. As a result, the techniques above described for extending MIME to exchange design data can be used also to transfer it using the WWW protocols.

### **3.5 Electronic Commerce**

Electronic Commerce is a new business infrastructure that enables the creation of a marketplace on the Internet. Electronic commerce uses:

- The World Wide Web, for making the information easily available to end-users.
- Electronic Data Interchange, to perform business transactions [oS91a].
- Encryption, to authenticate buyers and sellers and to protect the information exchange from being accessed by third parties.

Example initiatives that are developing electronic commerce services are:

- FAST, a research project at Information Sciences Institute [Nech93].
- EINET Galaxy, developed by MCC<sup>1</sup>.
- CommerceNet, a Silicon Valley initiative grouping the electronics industries in the area<sup>2</sup>.

---

1. The URL for EINET is <http://galaxy.einet.net/>

2. The URL for CommerceNet is <http://www.commerce.net/>

With the infrastructures for electronic commerce in place, new design methodologies based on the outsourcing of design and manufacturing services will become possible. We anticipate that electronic commerce will make it possible to offer multiple new services to electronic design and manufacturing organizations:

- *CAD outsourcing.* There will be available specialized CAD systems for specific design tasks. For instance, a company may sell the use of dedicated hardware and software for performing large and expensive simulations. The advantages of using this service are those associated with outsourcing in general: 1) economies of scale<sup>1</sup>, 2) focus on core competencies<sup>2</sup>, and 3) the flexibility of usage based costs<sup>3</sup>.
- *Collaborative Design and Design/Manufacturing Integration.* This involves adopting standards for conferencing, shared editing and exchange of design information. These enable much closer interactions between contractors and sub-contractors, speeding and increasing the quality of the artifacts produced.
- *On-line Component Information Services.* This will offer the ability to quickly retrieve datasheets and select components for a specific purpose. A great deal of component related information, mostly digitized from paper databooks, is already distributed by CD-ROM. As the Internet becomes more common, and mechanisms for selling this information are developed, it will eventually move to on-line services. This information contains design information that designers want to incorporate into their design databases, such as schematic symbols, simulation models, and application notes. On-

- 
1. Economies of scale — An organization with more experienced and specialized designers in one specific domain can in general provide services on that domain to other organizations at a smaller cost than that of supporting an in-house department offering the same service.
  2. Focus on core competencies — By not investing in resources that can be provided by external partners, organizations can invest more on the areas where they can develop sustainable competitive advantages.
  3. Usage based costs — Organizations will not have to support large up-front costs in CAD hardware, software and services. Their costs will scale uniformly as their needs of design related resources evolve over time.

line component information services will have more up-to-date information that will be easier to retrieve and install. New billing methods, based on the actual information retrieved, will be possible, making this information affordable to smaller organizations.

- *Broker Services.* New brokerage services will support the search for specialized information about design, prototyping and manufacturing services, or about electronic components. They can also evaluate it and give recommendations and endorsements of special providers. For instance, these recommendations could be provided as active documents containing a methodology for designing the client's product. The proposed design methodology included in the active document could have links to cell libraries, simulation services and other internet resources that could be used for the design.
- *Business Services.* These will be the non-design specific services that will form the backbone of electronic commerce infrastructures. They will include 1) electronic Yellow Pages containing indexed catalogs of design services, 2) electronic White Pages with catalogs of people, 3) electronic payment services using Electronic Data Interchange (EDI) standards and 4) electronic Notaries (also called certification authorities) for authentication of the participants in business transactions.

### 3.6 Summary

MIME is used to format multi-media data in the Internet protocols used for electronic mail, network news and client-server communications. As MIME is extensible, it can easily be used to encode design data. Interchange formats such as EDIF and VHDL are likely candidates to become universally recognized MIME types.

However, when transporting documents and data it is also necessary to have a mechanism for indicating which operations to perform on the received data. This can be achieved by extending MIME to also support active messages. These can also contain programs to be evaluated upon delivery, in addition to the data. As we describe in the next chapters, active messages are also the vehicle used by the Henry System to exchange active documents.

Given the ubiquity of the WWW and new services supported in electronic commerce, we can anticipate that these will also spawn many new services dedicated to designers of electronic systems. The next generation of CAD systems for VLSI design will have to provide seamless interfaces to access the services offered by electronic commerce. We believe that design systems like Henry, which use a documentation metaphor to interact with design tools and data, will also be specially well suited to give an integrated interface to access electronic commerce services.

“Be liberal in what you receive, but conservative in what you send.”

— John Postel

## **Chapter 4**

# **The Next Generation Design Environment**

As discussed in previous chapters, integration of design and documentation depends critically on a seamless interface that enables access to all design related information and tools. We observe that existing framework technology is not providing the necessary support and is not making progress in that direction. To successfully integrate all the information, we need to examine the existing design environment and find ways of interacting with it in a uniform way. This chapter presents our vision for a next generation design environment, providing this type of integration. We discuss the assumptions made by the designers of existing environments that are no longer valid, make new assumptions, derive new requirements and evaluate new support technologies. These will be used to support the architectural design of the Henry system, described in the next chapter.

### **4.1 Introduction**

The requirements for a CAD system for electronic design evolve continuously, as design complexity increases and new design automation tools and techniques are introduced. As the design complexity increases, computer and network technologies improve, and as our

understanding of design matures, new generations of design environments progressively replace the existing ones.

Historically, we have moved from an initial situation where designers run a set of independent tools, to design systems where tools shared a common data base, to design frameworks that attempt to integrate the entire design process (see Section 2.3 on page 29).

In the ideal implementation of a design framework-based environment, all tools have been specially designed for and integrated into the framework. There is one only instance of each of the framework services, shared by all. Tools use the framework services and give notification to the framework of any actions they perform on the design data.

However, the common design framework approach is extremely difficult to implement in the real design world, because there is no such thing as the best framework with the best tools for every conceivable design task. As a result, current design environments typically span multiple independent frameworks. There is little or no support for global design information and methodology management in such an environment.

In addition, with the advent of the Internet into the business world, we are beginning to observe a significant difference in the way electronic systems are designed. The Internet is quickly becoming available to virtually every organization in the electronic systems design industry. We anticipate that most information and services used by designers today will soon become available as part of a large distributed hypermedia system, interconnected via Internet protocols. We are moving from a tool-centered design environment built around a set of common services into a new *information-centric* environment, where designers focus on the creation of an information web. This contains not only the information describing the artifacts being designed but also references to component and design-service suppliers, manufacturing and prototyping services, and product marketing data,.

In this chapter, we present the conceptual model for a new generation of design environments that are well suited for these new realities. We propose means to manage design



data and processes that enable simultaneous use of multiple frameworks, based on the new paradigm for combining design and documentation through an open hypermedia approach. Currently, it is typical to have a VLSI design start in a schematics and simulation framework, then move to another framework for logic synthesis, then to another for the circuit layout. The process of moving the design data between these frameworks is not automated. Our goal is to make it possible to create design methodologies that could control the transfer of the design data between the various systems and designers involved as a process of exchanging multimedia documents containing the data.

The remainder of this chapter is organized as follows. We start by discussing in Section 4.2 the assumptions made by the designers of the current generation of design frameworks that we believe are no longer valid, and new requirements. Next, in Section 4.3, we present the implications of these new requirements for the architecture of the next generation design environments and the available technologies that can be used in their implementation. Section 4.4 closes the chapter with a summary of the main ideas for the conceptual model of the new environment.

## **4.2 Requirements**

The current generation of framework-based design environments makes the following assumptions:

1. A single framework controls the entire design process;
2. All design related information and services are available locally to the design team;
3. The design is the product of a single organization;
4. The framework is essentially used to manage design specific tools and artifact representation data.

In our view, these assumptions are no longer valid. The next generation of information-based design environments will have four new basic requirements:

1. Support for use of multiple, independently managed, heterogeneous frameworks;
2. Capable of accessing on-line services available through electronic commerce;
3. Flexible structure, adapted to new business models;
4. Capture more design-related information and earlier in the design process.

We present and discuss these requirements in the remaining of this section.

#### **4.2.1 Multiple Heterogeneous Sets of Tools**

Design frameworks have been designed under the assumption that the design environment would consist of a single framework capable of integrating all of the tools needed by the design team within the organization. All the information related to a given product's development and the critical tools will eventually be managed by *their* framework.

However, a recent innovation in product development technology is the introduction of concurrent engineering methodologies [Cart92]. Concurrent product engineering calls for the assembly of multi-disciplinary teams, including marketing specialists, designers, and manufacturing engineers, working together in the development of a new product.

In our view, this assumption has failed to address satisfactorily the needs of multi-disciplinary design teams. For example, consider electronic systems design. A complete electronic system involves integrated circuit, printed circuit board and mechanical designs. Integrated circuit design is typically divided into independent steps: architectural design, logic design, physical design. Given the variety and the complexity of the problems that have to be addressed at the various levels, it is not possible for a single framework supplier to offer a complete set of best-in-class tools covering the entire system design process. On the other hand, given this diversity, we are also unlikely to see the emergence of a dominant framework design that imposes a standard interface used by all design domains. In addition, porting tools not specifically designed for a chosen framework

requires an extraordinary effort, never completely successful, on adapting the new tools to different interfaces.

This has lead design system integrators to search for *best-in-breed* tools in each specific domain. Existing CAD systems are an assembly of the design frameworks required to run the tools selected for use in the design environment. In general, a separate framework is used for each design domain. Inside each domain, there is tight integration between the individual tools and the use of automated design methodologies. Transfer of information between these domains requires export/import of design information, using industrial standard formats for design data interchange (EDIF, Calma, IGES, VHDL).

The new generation of design environments should support integrated design methodologies involving uniform access methods to the data and tools managed by independent frameworks.

#### **4.2.2 Interaction with Electronic Commerce Services**

Another important assumption in the design of current Framework technology is that design will take place within a single organization and that all information required in the design process will be available within it.

However, we observe multiple electronic commerce initiatives whose primary objective is to create a global electronic marketplace built upon the Internet (see Section 3.5 on page 51). A significant part of the support for these initiatives comes directly from the computer and electronics industries. As a result, we can expect that an important mass of information organized as web of documents with critical importance to designers will soon be available interactively or via automated search methods.

The new electronic commerce services targeted to designers that we anticipated in Section 3.5 are not distant from reality. Cadence, the largest CAD Software vendor today, is considering providing CAD outsourcing services. These are similar to those offered for Management Information Systems (MIS) by companies like EDS. Designers not only rent

the hardware, but also the design tools. In addition, consulting services might be offered, both in the form of assistance by expert designers and of pre-design process flows for completing specific design tasks. Services of this kind would be appealing to small organizations, and could easily be provided through the electronic commerce infrastructure.

The requirement for the next generation design systems is that they should be able to make efficient use of electronic commerce services, supporting integration of remote information, tools and machinery, accessed via electronic data interchange transactions.

### **4.2.3 New Business Models: The Virtual Corporation**

Another important development is that system design organizations are adopting new business models for design, manufacture and commercialization of electronic systems. These models are based on networks of small and medium independent corporations and/or autonomous divisions within large corporations. This new business model has been called the *virtual corporation* [Davi92]. In the virtual corporation, products and services appear to the customer as marked, designed, manufactured and supported by a single entity, but are in reality the product of a consortium of independent business units working in tight cooperation.

Although it is likely that virtual corporations will be founded on the infrastructures developed for electronic commerce, they will also go far beyond the electronic data interchange model of electronic commerce. Electronic commerce is dedicated to faster trade of commodities. The virtual corporation implies development of long-term, but still dynamic, relationships. These are based on information sharing, common product design/development methodologies, and complementary core competencies of the organizations involved. Homa Bahrami characterized the emerging organizational system for high-technology firms as a “*federation*” or a “*constellation*” of business units that are typically *interdependent; relying on one another for critical expertise and know how* [Bahr92].

For designers and their organizations this implies a tremendous change. An organization embarking in a virtual corporation business model needs to pay much more attention to what information is given and obtained. It will be an even greater challenge for design environment integrators. In more traditional concurrent engineering, the goal is to integrate marketing/design/manufacture activities within a single corporation, by standardizing on a common design framework. In the virtual corporation, concurrent engineering must take place in an environment composed of highly heterogeneous frameworks, configured and managed independently by multiple business units.

To achieve this purpose in full, two critical aspects must be considered: the time required to exchange products between organizations and the trustworthiness of the organizations exchanging it.

Time is significant, even when the product consists exclusively of electronically transferred information and the existing Internet infrastructure and software could be used. Order processing and verification can still take days in many existing organizations. To make it faster, common standards for conferencing, shared editing and exchange of design information must be adopted. It also implies that mechanisms to automate the transfer of information between design environments must be deployed and used.

Ensuring trustworthiness of design services is a hard problem. Although good privacy and authentication can be easily provided with current software, these are only the initial security issues that need to be addressed by a virtual corporation. When an organization gives away the information relative to its knowledge in its domain of expertise, it needs to have some guarantee that those who receive it will not use it in ways that may cause damage to their own interests. For instance, if an organization has to rent the simulation hardware of a service provider to prototype a new design, the simulation files could easily be duplicated and made accessible to a third party without knowledge from the client. Virtual corporations need a combination of legal and technical mechanisms to prevent this. When a chip is sent to fabrication in a silicon foundry, some level of trust already exists between

the design and manufacturing organizations, which is also enforced with Intellectual Property law. However, we are assuming collaborations implying exchange of much more critical information. There is incomparably more knowledge in a VHDL simulation model of a system than in a set of fabrication masks. In the virtual corporation business model, trust is achieved primarily by locking-in customers and suppliers through the simultaneous creation of symbiotic relationships and critical dependencies. This means giving quick access to more information, but also creating an increased dependency on continued access to that information for those who are using it. In the example above, the survival of the simulation service provider would depend on the construction of a reputation for not selling or using the simulation models received. On the other hand, the survival of the design company would also depend on the quality of service provided to the simulation service provider.

What is required by design organizations operating using the virtual corporation model is an automated process of supporting collaborative relationships. For design environments, this is achieved by introducing flexible methodologies that enable progressive introduction of automation in the information exchange processes between collaborative business units. From the designers' perspective, the design environment becomes a network of information and service providers relying on each other. The network needs to provide quick access to information to trusted partners, assuring:

- secure access to critical resources;
- establishment of proper accounting methods for the use of information;

#### **4.2.4 More Design-Related Information Needs to be Integrated**

We believe that the model of a common CAD framework is inadequate to support the increased demand for addition of new tools and data types within the environment. As discussed above, electronic commerce will make the quantity and diversity of information available grow by orders of magnitude. In addition, the demand for larger and much more

heterogeneous information databases does not end here, as manufacture and documentation data become more deeply inter-related with the design data.

Better integration is necessary with the more general business tools that also manipulate design information, such as the FAX and electronic mail processing tools available on personal computers and workstations. System design involves many non design-specific tools, not integrated within any design framework, but which have a crucial role in the design process. Over time, portable mobile computers, liveboards and video conferencing hardware will become common to designers. Information captured by these new devices will also be able to be used as reference and documentation material for the design process. Designers will want to access corporate design information and will have to produce documents that include hypertext references to design objects in the team's common repository.

A key requirement is to support continuous expansion of the design environment. New data types, tools and information manipulation devices useful to designers are being introduced every day. The approach for integrating these into the common design framework may be too inflexible, requiring prohibitively expensive system integration costs and taking too much time. An alternative method is to have a mechanism to support quick integration of whole new information processing systems, running in separate frameworks, into the design environment.

### **4.3 Architectural implications for new design environments**

As discussed in the previous section, there are currently no efficient ways to access uniformly all the information available to designers. The capability to automate design methodologies spanning the entire product design and development cycle is even more remote. This requires synchronizing the execution of the tools and framework services in the multiple design frameworks used.

We believe that any feasible solution for addressing this problem, given the current status of design automation, needs to depart from the existing organization of the design environment. Typically this is independently developed and managed by different frameworks. In this organization, each framework is adapted to a specific design domain and tailored to the organizational demands of the design team using it. However, we believe that it is possible to create a new set of mechanisms to create and manage inter-dependencies between the data, tools and users of each set of tools. To achieve this, the development of the following common services is needed:

- Messaging mechanisms for passing data and control information between frameworks within a design group and inter-groups.
- Data and methodology management systems capable of tracking the design data and process flow at the global level. This is accomplished through monitoring, synchronizing and controlling the execution of exchanged messages within the environment.
- Uniform methods for accessing and manipulating information, to cope with the challenges of heterogeneity.

The relationship between the above components and how they fit together to extend the existing design environment is illustrated on Figure 4.1. In the remainder of this section, we propose an initial conceptual model, and discuss technologies that could be used in their implementation.

### **4.3.1 Common Messaging Services**

Electronic CAD tools are just now beginning to support the Tooltalk message services, the standard for inter-tool communications adopted by COSE and CFI (see Section 2.2 on page 22). In Tooltalk, a set of messages (or commands) that a tool accepts is defined for each class of tool. Tools conforming to the conventions can then become interchangeable. However, many more design tools exist based on other proprietary protocols. It is also



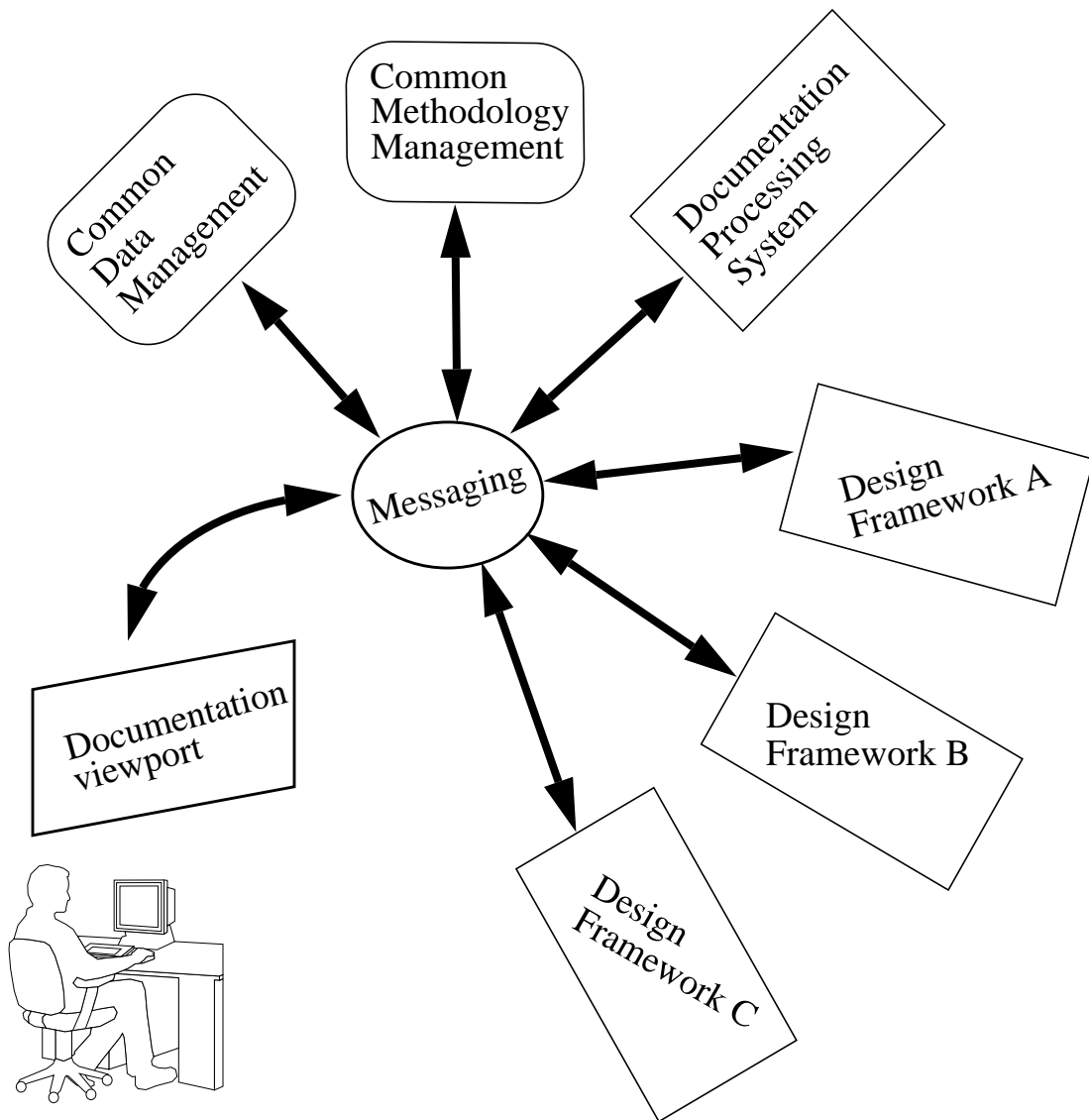


Figure 4.1 The New Design and Documentation Environment

The figure pictures our global vision of the new design environment as seen from the documentation point of view. Full integration of design and documentation is supported by the introduction of new components into the existing environment. Central to the system is a common message exchange system, capable of handling human and inter-framework communications. New common data and methodology management systems based on the common messaging system will be used to support complex design flows involving the tools, data and process management services of multiple design and documentation frameworks. Finally, a new viewport into the design environment, based on documentation, is required to provide uniform access to the heterogeneous data and tools of this environment. The arrows indicate how control information flows between the components of the system.

unlikely that the COSE message services will be standard on the more ubiquitous non-UNIX platforms.

It is then likely that we will tend to a situation where several inter-tool communication protocols will be common, and will co-exist with many other custom protocols. As a result, it is critical to develop *message transport systems* to address this situation. Within each framework, tools still intercommunicate using the existing inter-tool communications facility. However, when communication with an external framework is required, the message needs to be routed through an inter-framework messaging system that can translate between the used protocols.

This communications model is identical to that adopted for the Internet in the 1970s and 1980s. Instead of forcing every network to use the same set of protocols, the Internet created a set of communication protocols that would enable independent networks built on incompatible communications systems to exchange messages easily.

We believe that the existing Internet messaging protocols can be extended to support the type of inter-framework and inter-tool communications we envision for the next generation design environments. As discussed in the previous chapter, we have extended its communications software for use in the Henry System. We build on data transfer protocols developed on top of TCP/IP, such as SMTP for electronic mail, and HTTP used by the World Wide Web for client-server communications.

Adoption of these Internet protocols makes possible communications between design teams using incompatible environments, as messages containing design data could be processed manually or with the aid of widely available software. At the same time, it enables electronic commerce transactions and supports cooperation among organizations ready to automate information exchange between their environments.

In summary, we are now in a world of interconnected heterogeneous design environments for which a common communications model and protocols for automatic exchange of data

is needed. Internet protocols address the basic requirements for inter-organizational communications and could be used for inter-framework communication.

### **4.3.2 Common Data and Methodology Management Services**

A basic requirement for an integrated design and documentation system is its need to synchronize the execution of multiple tools to ensure that they display correlated data. Another important requirement is the ability to re-play the execution of the tools, to update the design and the documentation when an object is changed.

In a fully integrated environment, with multiple frameworks, this makes it necessary to add two new components:

- support for managing data configurations containing data manipulated by the tools running in multiple frameworks;
- availability of design process management systems capable of controlling execution of sequences of tools running within distinct frameworks.

In the typical industrial environments where designers must use multiple frameworks, it is already common practice to have an in-house developed or supported common set of process and data management utilities, independent of any framework [Rusu94]. Data is checked-in and checked-out between one vendor's framework database and another when the design flow moves from one tool to the other.

However, tools currently do not exist to support the design flow at this level. We believe that this could be achieved by designing data and methodology managers that can be composed to work with the framework's management systems, creating an hierarchy of management systems for design and data. Existing data managers would be in charge of data within their domains, while the high-level managers coordinate configurations composed of the data configurations in each environment. As an example, consider a design environment where three independent frameworks are used: one for schematics and simulation, another for printed circuit board layout and another for chip layout. Data in each design

domain would be managed by the corresponding framework, as it is done today. However, when users interact with the new global data manager, they can see the entire data, define global configurations involving data in the three frameworks, and pass control of individual objects from one framework to another. Similarly, existing methodology managers would be in charge of the design process management within their domain, while the high-level methodology manager would compose design flows managed by the domain specific methodology management systems.

Scallan has proposed a similar idea, based on the concept of a higher-level engineering framework, which manages the tools and data at the global level, integrated with multiple CFI-compliant application-specific frameworks [Scal94]. To make this possible, he proposes a new Inter-Framework Protocol, defined as a new set of CFI inter-tool communication messages. These would have to be supported by every framework in use.

We envision an identical hierarchy, but adapted to the virtual corporation model. It must be capable of being incrementally introduced into today's heterogeneous environments, where the inter-framework protocol would be based on common Internet protocols, complemented with message transport services for translating between tool-specific formats.

The new global management systems need to have special characteristics, not generally present in the management sub-systems of current frameworks:

- *Non-intrusive.* Individual data and process management systems should be able to work independently of global management systems. A designer working with a set of VLSI layout tools tightly integrated in a common framework with its own data and process management tools should perform all the layout related optimization without the need of running the global management services.
- *Cooperative with other management systems.* Framework-specific management tools should pass dependency constraints up to and down from the global management services, to support system level design and documentation tools that process information from multiple domains.

- *Support design transactions between frameworks and independent multi-framework environments.* There is a need not only to pass design information between different databases, but also to control exchange of information and synchronize execution of tools and user commands to different frameworks.

### 4.3.3 Uniform Methods to Manipulate Design Information

The third major element of the new design environment is a common method to access and manipulate information in its various formats. A common user interface will be an essential part of these new distributed heterogeneous environments. In our view, a common user interface for a design environment composed of many tools requires more than the same interface look and feel. The capability to create compositions of tools to support a given task, behaving as a single application as far as the user is concerned, is the critical need. We use the term *tool ensembles* to describe these sets of tightly integrated tools.

We believe that a documentation paradigm for the user interfaces of tool ensembles will become common practice. This idea of accessing and manipulating heterogeneous information as documentation has been extensively used in WWW navigation systems like Mosaic [Andr93]. With the WWW, multimedia active documents became the dominant paradigm for accessing remote services through the Internet. They offer a very powerful metaphor for manipulation of distributed information, and are effective in encapsulating the underlying heterogeneity.

## 4.4 Summary

We view the next generation design environment as a constellation of information systems, each with its own support services. The systems are integrated in such a way that they can cooperate to assist users in the system design process at the highest level.

Cooperation and uniform access methods are provided by new components of the design environment:

- Common message services.
- Common data and process management services.
- A new viewport into the design space, based on the manipulation of active documents.

Building the next generation design environment is a challenging engineering problem. Such an environment will be implemented on top of existing information systems: multiple CAD frameworks optimized for different design domains, documentation systems, software development environments, and groupware.

In the next two chapters, we describe the conceptual models, architectures and implementations of tools and services developed for the Henry System, an initial prototype environment embodying some of the ideas exposed here. This environment combines tools and frameworks from two independent domains, design and documentation, into a open hypermedia system. In Chapter 5, we describe the architecture of Henry, its support services and their implementation. Chapter 6 describes the design, implementation and integration of the Navigator and other tools and frameworks into this environment and develops some design scenarios involving the use of the prototype environment.

“The two most important tools an architect has are the eraser in the drawing room and the sledge hammer on the construction site.”

— Frank Loyd Wright

## **Chapter 5**

# **The Infrastructure for Integrated Design and Documentation**

We have built a prototype for integrated design and documentation, called the Henry System. Henry adopts a new architecture for the design environment, based on common messaging services. Henry’s primary design goal was to explore mechanisms for combining design and documentation activities. This is achieved by seeing the design process as the creation of multimedia presentations involving the tools and design data. In this chapter, we discuss the architecture and implementation of the infrastructure for supporting active documents offered by the Henry System.

### **5.1 Introduction**

We present the Henry design environment architecture by showing it from multiple views. The same approach was used by CFI in its Framework Architecture Reference (FAR) document [CAD93]. The Views defined in that document are summarized in Table 5.1.

In the design of the Henry System we have not addressed all the views, but only those where we had to introduce new perspectives. We are interested in how the tools, data and

<b>View</b>	<b>Describes</b>
User View	How the design framework is seen from its users: designers, administrators, tool developers.
Tool Integration View	How tools are encapsulated and integrated.
Framework Services View	System environment, system extension language, data and methodology management services, session management.
Communications View	The flow of events, data and control information between the components of the framework.
Data Management View	Conceptual model for management data.
Design Information View	Conceptual model for design data.

Table 5.1 Summary of CFI's Framework Architecture Reference Views

The Framework Architecture Reference (FAR) specification consists on the definition of the indicated views. In the Henry System, we address the top four views differently. This chapter discusses our perspective of these views.

framework services could be seen by users and system integrators as part of a hypermedia system. This implies that we need to re-visit the first four views of Table 5.1 and analyze them from this viewpoint. However, the study of the design environment from this perspective is orthogonal to how the design specific information is modelled and how it is managed in the database. We are interested in using the design tools in a different way, not in changing the way they represent or access the information.

The Henry System architecture is presented in the next three sections. These are followed by Section 5.5 on page 100, which contains a summary of the main ideas discussed. In Section 5.2, we present the communications infrastructure. In Section 5.3, we present the Henry architecture from the user's view, as a system that enables designers to access all the information as documentation. In Section 5.4, we discuss the system integrator's perspective of the system, as an infrastructure that offers services for building ensembles of design and documentation tools.



There is not a one-to-one mapping between the next three sections and corresponding Views in the CFI Framework Architecture Reference. Section 5.2, “Communications in Henry,” on page 73, discusses aspects of the “Communications View” and of the “Framework Services View.” Section 5.3, “Henry as an Open Hypermedia System,” on page 87, addresses the “User View” and “Tool Integration View”. Finally, Section 5.4, “Henry as an Infrastructure for Building Tool Ensembles,” on page 94, discusses the “Framework Services View and also the “Tool Integration View.”

## 5.2 Communications in Henry

As discussed in the previous chapter, given the enormous quantity and heterogeneity of information available to designers, existing design environment architectures will evolve into a new generation. In the new architecture, we envision design data distributed across a wide-area network, organized as web of related pieces of information. In this environment, inter-framework communication will be based on the exchange of messages containing documents.

The Henry System uses a new protocol for communication suited for the new active document-based environment. Active documents have the capability to send and receive commands and data. We call these commands active messages, as they resemble the messages used in active mail systems discussed in Section 2.6.1 on page 41.

Active messages contain data and commands to be performed on that data upon delivery. Active messages in Henry can be transported via SMTP<sup>1</sup> [rfc821] and handled by conventional mail readers, as in active mail systems. However, we use them not only for communication between end-users, but fundamentally for intertool communication. For instance, a user browsing an active design document may generate an active message requesting a database to return the layout of a circuit being described. The layout could come in the

---

1. SMTP — Simple Mail Transfer Protocol, the Internet standard for exchanging electronic mail messages between hosts.

form of another active message addressed to the layout editor, that would in turn display the data contained in the message.

Intertool communication is based on RPC<sup>1</sup> protocols [Birr84]. Intertool communication takes place either between the various tools run at the designer's workstation or between a tool and a service at a remote location. The protocols used in local area networks, such as Tooltalk [SunS92], are optimized for activating remote commands with small latencies. Data is assumed to be available via a common file system using NFS<sup>2</sup> [Grou88]. This combination of protocols does not scale well when we consider larger networks consisting of multiple organizations exchanging commands and data: they were not oriented to support the transaction-oriented paradigm for accessing information required by our application.

A protocol better adapted for the exchange of active messages is HTTP<sup>3</sup> [BL94a], the client-server communications protocol used in the WWW<sup>4</sup>. In Henry, we use HTTP to transport active messages. HTTP also uses MIME<sup>5</sup> [Bore93] as the encoding mechanism to pack information into messages.

The global view of the integrated design and documentation environment from the communications point of view is depicted on Figure 5.1. An information web is formed by a network of active documents. These documents are exchanged and modified through a network of inter-commutating processes that exchange active messages, which we call

- 
1. RPC — Remote Procedure Call, a protocol for activating procedures in programs running in different address spaces
  2. NFS — Network File System, a protocol and associated client and server software transparent access to files in remote computers.
  3. HTTP — HyperText Transfer Protocol, the basic communications protocol of the WWW.
  4. WWW — World Wide Web, the software system that views the Internet as a big hypermedia system. We introduced the WWW in Section 3.4 on page 50.
  5. MIME — Multi-purpose Internet Mail Extensions, the extensible Internet standard for formatting electronic messages containing not only text but other types of data. MIME was introduced in Section 3.2 on page 45.

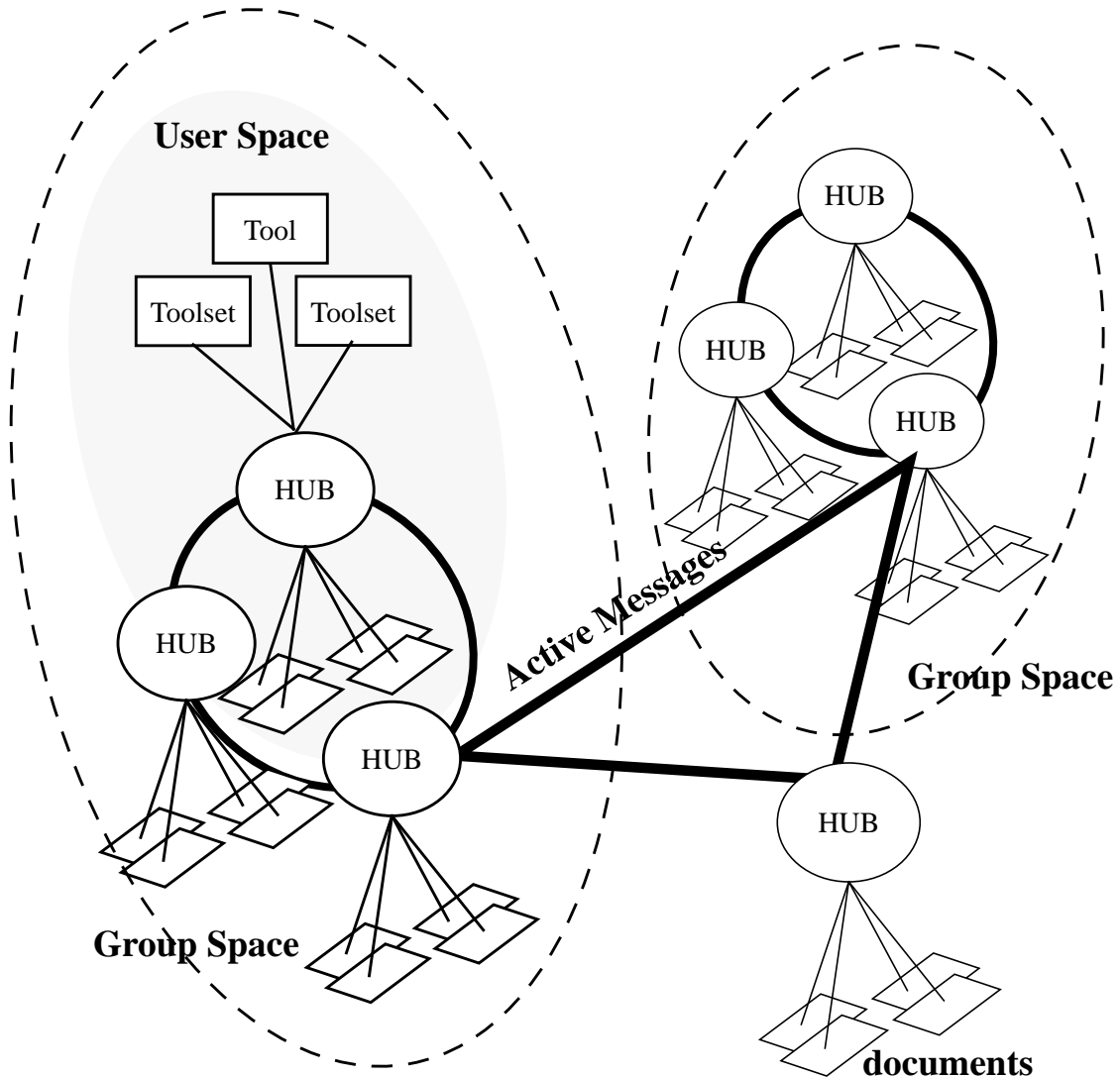


Figure 5.1 The Information Centric Design Environment Organization

All design information is viewed as a web of active documents, including design files and scripts of commands. Tools are used to manipulate documents. Documents are packed into *active messages*, containing data and operations to be performed on the data by the receiver of the message. Active messages are exchanged via specialized message servers, called HUBs. HUBs stand between document manipulation tools and the information web. HUBs can be setup to manage the exchange of information between the ensemble of tools run by a user or between groups of users. The format of active messages and the message exchange protocols used are defined by Internet standards. We use MIME for active message representation and HTTP as the message transport protocol.

HUBs. HUBs also provide additional functionality, as a side effect of the evaluation. In general, the result of processing a message includes

- forwarding the message to other HUBs, users or specific tools.
- modification of the information web, by converting messages into new documents.

The organization of the design environment based on a web of HUBs has the flexibility required to adapt to the dynamic constellations of business units that characterize the virtual corporation. In the Henry System, each user has an associated HUB running on his workstation. Users' HUBs manage the activation and inter-tool communication between the tools run by each user. In addition, groups of users can set up a HUB for handling messages for which the dispatching procedure requires knowledge of the group organization, such as broadcasts of messages addressed to team members assigned to a specific task. In an electronic system design team, group HUBs resolve message addresses like "logic designers" or the "PCB design manager." In a similar way, deeper hierarchies can be established to support larger groups with multiple teams.

As HUBs use the Internet message exchange protocols and formats, it is possible to create design environments with heterogeneous frameworks and many levels of integration. The possibility of exchanging design objects and commands to remote design systems via electronic mail makes it possible to create multi-organizational design environments operating at various levels of integration. At one site, processing of a given active message could consist in forwarding the embedded commands for execution by a running tool. In another less automated environment, the same message could be placed into a user's mailbox to be handled manually. To complete processing, the designer at the receiving site would have to examine the contents of the message, retrieve its contents, call the appropriate tools and return the resulting data formatted according to the conventions in use.

In our presentation of the communications architecture of the Henry System, we first present the conceptual model for message handling. Then, we describe the general opera-

tion of the HUB by following the path of a message from its generation by a user command to its delivery to a design tool.

### **5.2.1 Conceptual Model for Message Handling**

The common format adopted for active messages exchanged between HUBs is Enabled Mail (EM), the proposed standard format for active mail. However, the paradigm used to transport active messages in Henry is different from active mail systems. The former uses HTTP, the protocol used in the WWW, while the later uses SMTP, the Internet mail transfer protocol. Henry uses a RPC-based protocol to “pull” information from information servers, while electronic mail is designed to “push” information to receivers of information.

Enabled Mail assumes an environment for activating messages consisting of two interpreters of the Tcl language. One runs Safe-Tcl, a restricted subset of the commands of the Tcl language, while the other fully supports it. The former runs as an untrusted interpreter that evaluates the commands embedded in incoming active messages; it has no access to any system resources. The later evaluates the commands received from the untrusted interpreter, after verifying that the evaluation would not compromise the integrity of the system. Programs run by the untrusted interpreter may only access the data in the active message and send commands for evaluation to the trusted interpreter.

We have extended Enabled-Mail by adding new commands to the Safe-Tcl language. The new commands define an interface to access a library of message handling functions for sending commands to design and documentation tools. In the Henry System, the HUB also runs the two interpreters required in the Enabled Mail model. The message handling functions that communicate with the tools run in the trusted Tcl interpreter. On the other hand, the scripts embedded in active messages run in the untrusted interpreter. When part of the contents of a message needs to be saved into a file or a command has to be sent to a tool, a Tcl command is sent for evaluation in the trusted interpreter (See Figure 5.2.).

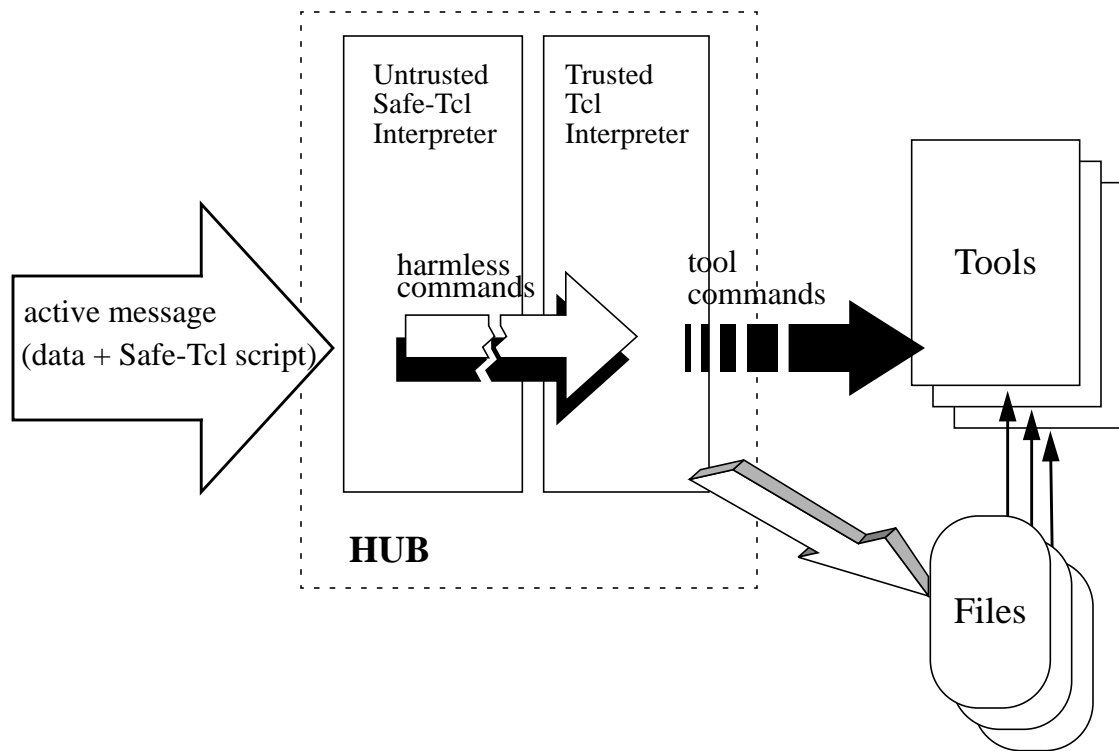


Figure 5.2 Message Handling in Henry

We use the Enabled Mail conceptual model for Message Handling. The Safe-Tcl commands embedded in active messages are evaluated in an untrusted interpreter. This interpreter cannot access any system resources, only the information contained in the message. To pass the data to the design tools, the unsafe environment has to use the commands available in the trusted Tcl interpreter. This interpreter checks that the commands received will not affect the integrity of the system before saving any data into files or invoking any operations on the tools.

### 5.2.2 Operation of the HUB

Receiving and processing a message in the HUB is a fairly complex operation. It is easiest to understand by following the step-by-step process from when a message is generated, sent to a HUB and finally delivered to another tool. To illustrate these concepts, we develop a scenario where a user requests a SPICE circuit simulation. To integrate SPICE into our architectural model, we developed a modified version of the simulator that

accepts commands from a TELNET protocol port concurrently with commands typed in at the user interface.

Assume that a user while browsing an active document, presses a button to request a simulation. A procedure associated with the button activation is invoked to generate the active message. The generated message is formatted using the MIME-compliant conventions described above, and is sent to the HUB for processing.

The generated simulation request is a *multipart/enabled-mail* message that contains the netlist, simulation models, simulation commands, and a Safe-Tcl script for their evaluation. The Safe-Tcl script included in the message is evaluated once received at the HUB. This results in the generation of the files required by the simulation and sending the appropriate commands to SPICE to perform the simulation. The files retrieved from the message are:

1. A simulation deck with the netlist and models.
2. A script of SPICE commands to read the deck, perform the simulation, and display the waveforms needed to illustrate the design detail presented in the active document.

After parsing the message, the Safe-Tcl script invokes a HUB procedure to connect to the SPICE simulator via the TELNET protocol. Finally, the script of SPICE commands is sent to the simulator for evaluation.

There are two major steps involved in the process of dispatching the simulation request message to a tool such as SPICE:

1. Transform the *multipart/enabled-mail* message into one or more files to be processed and the commands to be sent to the tool.
2. Establish a connection with the tool and send it the command(s) to evaluate.

The description of the processing of this relatively simple request illustrates the enormous possibilities that are opened to system integrators by using an integration mechanism

based on active messages. The Safe-Tcl scripts embedded within the messages could be made much more complex or new Tcl scripts could be added to the HUB to be executed as pre- and post-conditions for message activation. Database check-in and check-out operations, various types of design space name resolution functions, and process history logging are examples of handling functions that could be attached to every processed message for evaluation upon activation.

The HUB is structured in software layers, as is common in communications systems. There are two main layers:

1. The Message Transport Layer (MTL).
2. The Message Handling Layer (MHL).

The generic architecture of the Henry HUB is shown in Figure 5.3. The functions that support the operations of converting MIME messages to commands and data objects, as well as those for evaluating the active part of messages and associated handlers, constitute the Message Handling Layer. The Message Transport Layer, consists of the functions that perform the low level interface to start the tools and send them the commands and data. Both are described in more detail below.

### **5.2.2.1 The Message Transport Layer**

The Message Transport Layer provides support for interfacing the clients and servers integrated in the Henry System, encapsulating the different protocols they use under a common interface. We identify three sub-layers in the Message Transport Layer, as shown on Figure 5.4. The lower level layer, provides an interface to communicate with each protocol integrated within the HUB. In our implementation, we support, among others: (1) TCP/IP streams using the UNIX socket interface, (2) Sun RPC [Grou88, Corb91], and (3) communication through a common X Window System server [oS90, Sche92]. Each of these interfaces is supported via a library of C or C++ routines that can be invoked as Tcl commands. On top of the basic communication primitives, we have a layer consisting of



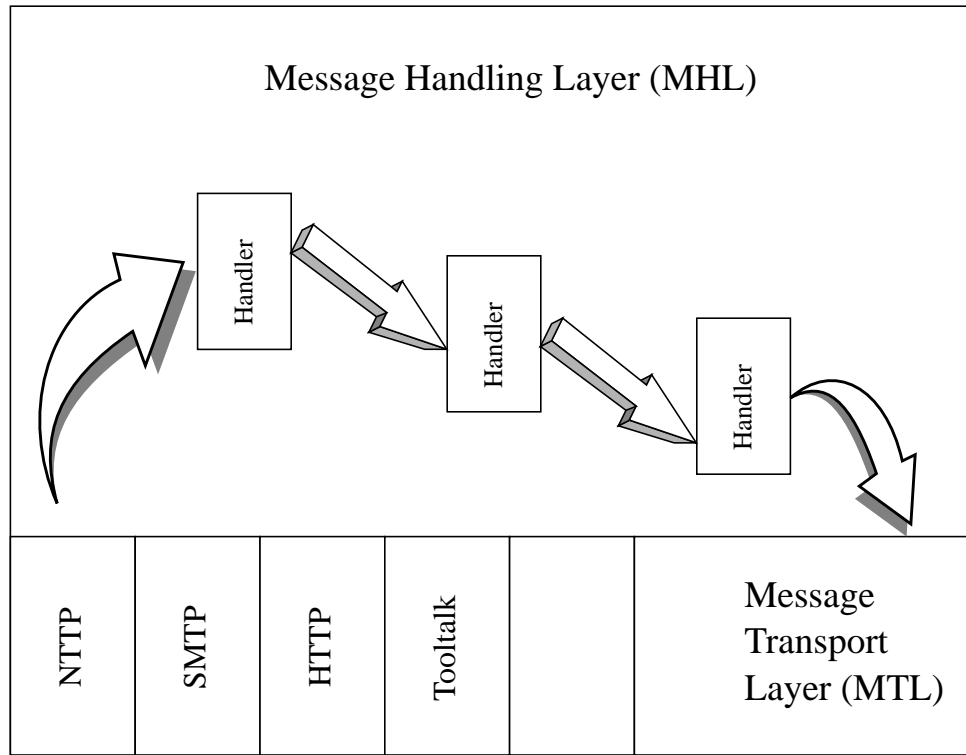


Figure 5.3 Architecture the Henry System HUB.

The Message Transport Layer, converts messages exchanged in various protocols into a common format. We adopted the Internet generic message format, consisting in RFC822 style headers and MIME formatted body. We support active messages through an extension to MIME that defines a new contents type for representing a script in an extension of the Tcl language. These scripts are evaluated in the Message Handling Layer. Each command in the scripting language is processed by a dedicated handler. Handlers can be added and modified to make HUBs perform new or modified functions as a side effect of dispatching active messages.

several modules, one for each tool or framework integrated in the Henry System. Each module defines a set of new Tcl commands for interfacing with the tool, based on the communications primitives defined by the lower protocol-specific layer. On top of this, we have a layer that abstracts the tools and attempts to provide a uniform interface to the Message Handling Layer above.

The top-level sub-layer defines the interface between the Message Handling Layer and the Message Transport Layer. This interface consists of a new Tcl command, *hmessage*, used

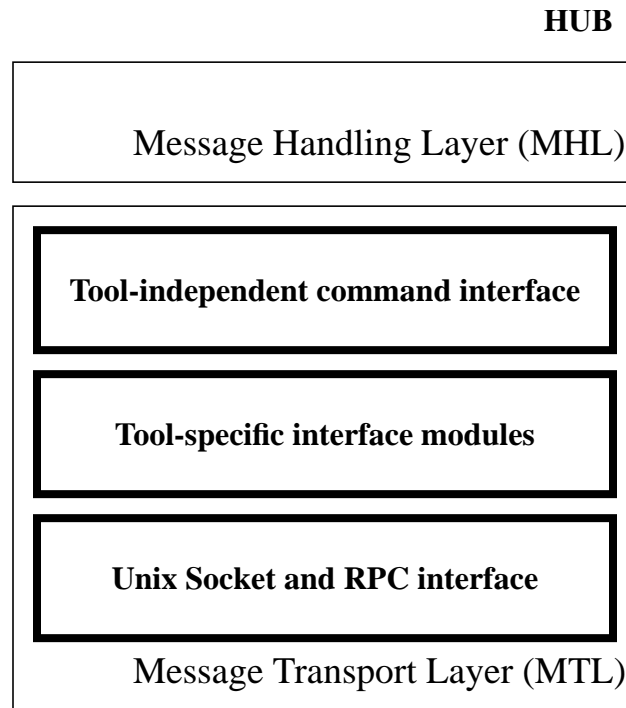


Figure 5.4 The Sub-Layers of the HUB Message Transport Layer

At the bottom, we have a command interface to access existing remote procedure call (RPC) and inter-process communication mechanisms. At the next layer, we have tool-specific interface modules. Each module defines the commands that can be sent to a tool, while abstracting the lower layer. The top layer offers a tool-independent single command interface for sending commands to each tool. These three layer of modules define the interface between the two main layers of the HUB.

to call the operations that can be performed on every tool. The general form of the *hmessage* command is

*hmessage application-address operation operation-parameters*

where the *tool-address* field is a three element list containing

1. The name of the application to which the message is directed.
2. The internet address of the user running the tool.
3. The host name and display where the tool should run.

An example of a completely specified tool address is:

*{Magic msilva@CS.Berkeley.EDU mercenary:0}*

This represents a message to the tool called *Magic*, run by the user with electronic mail address *msilva@CS.Berkeley.EDU* on host *mercenary*, X display 0.

The *hmessage* command defines an essential interface in the HUB architecture. It has two major roles:

1. Defines the point of transition between the tool independent message handling software and tool-specific message processing.
2. Defines a point of transition between the untrusted execution environment for active messages and the trusted environment; *hmessage* is declared safe for invocation from the Safe-Tcl untrusted interpreters used to evaluate active messages<sup>1</sup>.

An essential aspect of the Henry architecture is the design of a common interface to abstract the tools at this level. In a system comprising heterogeneous tools, a common framework is critical for supporting the different command syntaxes used by the tools. For instance, to read a file into an application's address space, we observe that SPICE3 uses the command *source*, whereas Magic uses *load* and FrameMaker *open*. All these prefer the syntax

*command [parameters],*

but VEM uses the inverse, i. e., polish notation.

We have identified the common operations supported by the tools to which we interface. These are listed in Table 5.2. By creating a uniform syntax to invoke these common operations, we make the tool interface uniform to higher layers of software. This uniformity also makes it easier for integrators to create hyperlinks to tools that have different com-

---

1. *declared harmless* in Safe-Tcl terminology.

Command	Function
ping	check if a tool is running
start	send the ping message to a tool and start it if no answer is received
open <object>	send the start message and open, source or load the object given as argument
do <command>	perform the command in the tool's command language syntax. This provides the "escape" function to execute any tool specific command not offered by this interface.
quit	terminate execution of the tool

Table 5.2 Common Operations Supported by the Tools Integrated with the HUB  
mand language syntaxes and terminologies, as they do not have to remember the specific command names used for the most common operations.

Another important aspect of the design of the Message Transport Layer is that its operation is based on stateless communications. The HUB does not maintain any connections with communicating peers. In Henry, message delivery always implies establishing a new connection with the receiver, exchanging the information and closing the connection.

We also designed the semantics of the MTL interface so that when delivering a message to a tool, the sender needs not worry about the tool's state. We recreate all the pre-conditions for successful execution of the message being delivered. For instance, when a client sends a command change the contents of a file, it does not have to send commands to start the tool or open the file in advance. These are implicit on the request and are invoked automatically by the HUB if required.

### 5.2.2.2 The Message Handling Layer

The HUB's top software layer is the Message Handling Layer. It provides the utilities for parsing the MIME messages used in the standard format and the environment for evaluation of the Safe-Tcl scripts contained within active messages.

The MHL is organized as a framework where multiple customizable operations can be applied to messages as they are processed (see Figure 5.5). In the MHL, operations on messages are performed by *handlers*. These can be of two types: *header handlers* and *command handlers*. The former operate on the message headers while the later are the Safe-Tcl routines that evaluate the scripts embedded in active messages.

Handlers can be written to perform many distinct functions. One of the most important is name resolution. Messages in general do not contain fully specified names of destination addresses, object versions to operate on, or the tools to be activated by the messages. For instance, the *hmessage* command could be written as follows when received by the HUB:

```
hmessage [MHL_tool LAYOUT_EDITOR] open [MHL_check-out NAND2],
```

In the Tcl language, this means that the handlers for the layout editor used at the HUB and the check-out of the current version of the 2-input nand gate would be invoked before execution of the *hmessage* command. After execution of the handlers, the command would read as

```
hmessage {Magic msilva localhost:0} open /users/msilva/design/version3/nand2.mag
```

Another important function of the HUBs handlers is the support for activating and managing hypermedia-type links between the integrated design and documentation tools. This will be discussed in more detail on the next section, in the context of the user's view of the design environment.

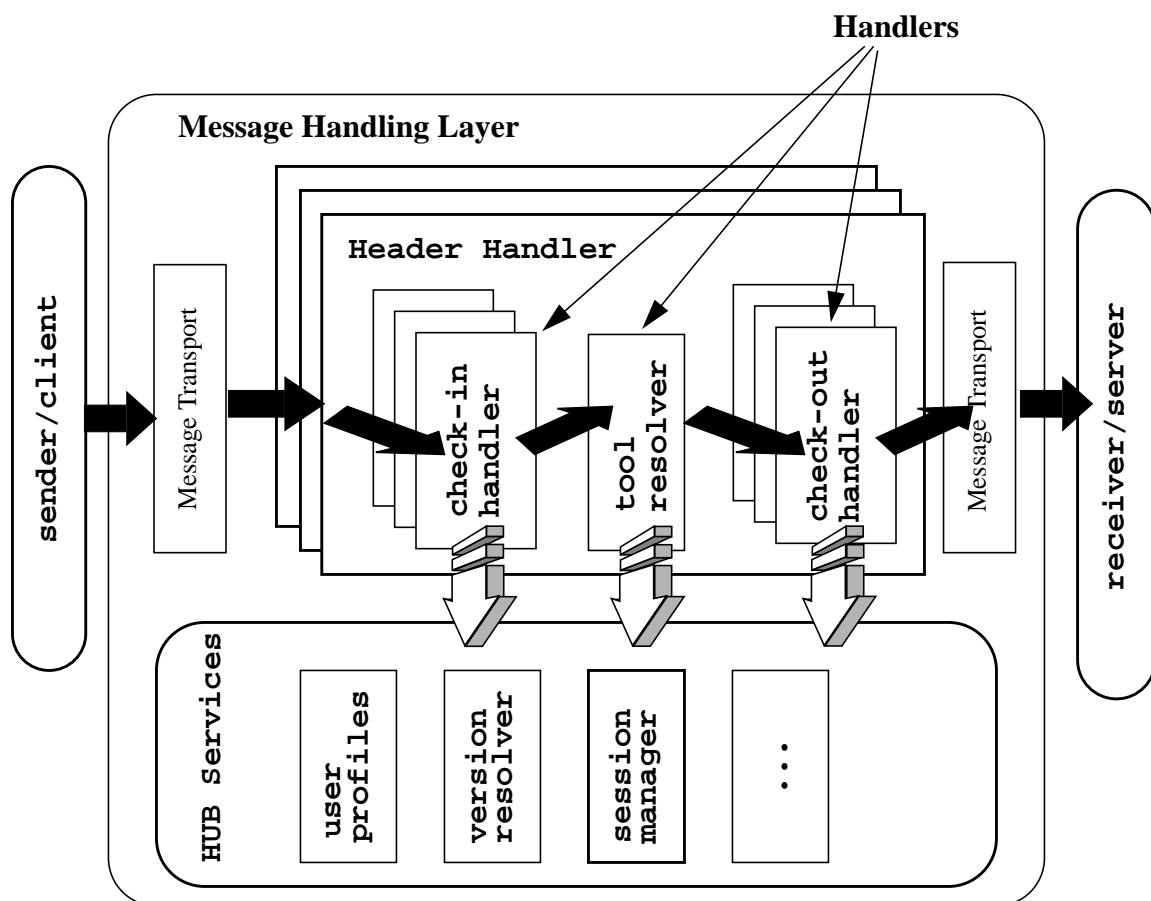


Figure 5.5 The HUB Message Handling Layer

Messages are translated into a common format in the HUB, and then processed by a configurable sequence of *message handlers*. There are two types of handlers: *header handlers* and *command handlers*. Header handlers operate on the message as a whole, without parsing its contents. For example, they can be used to resolve addresses or to keep a history of message activations. Command handlers are the procedures executed when the commands embedded in the active message scripts are evaluated. Both handlers make use of a library of utilities, called the *HUB services*.

### 5.3 Henry as an Open Hypermedia System

From the user's point of view, one of the main requirements for an integrated design information processing system is that it should provide quick, uniform and easily understandable mechanisms for accessing and manipulating all the information related to the design process. As discussed in Chapter 1, to satisfy this requirement we use hypermedia documents as the common metaphor for the interface with the design process, tools and data.

Using the hypermedia interface, designers gain access to a web of design information. From his or her perspective, the system operates as follows:

1. The designer selects a piece of design related-information.
2. When the designer activates the selection, he or she sees a list of descriptors for other pieces of information. These are related to the object upon which the operations are being performed.
3. Activation of one of the operations, launches the invocation of another tool. The new tool fetches and/or generates other pieces of information.

A similar type of interaction is already used with some combinations of tools by VLSI designers. For instance, there are commercial versions of integrated simulation systems containing a schematics editor, waveform displayer and circuit simulator. In these systems, a user can select a net on the editor and then request the waveform displayer to show the last simulated signal associated with the net. Our goal is to generalize this interaction, so that users can define and associate multiple actions with any design object, select one and invoke it.

In Hypermedia terminology, we call these operations *live link* activations. We use this term because they do not simply cause the display of other information, but rather send an arbitrary command to a running tool.

In a traditional Hypertext system, a link is an object containing information about two *anchors*. An anchor is a portion of text that is sensitive to user's activation. In general,

anchors can be in distinct documents. When the user activates the first anchor, the resulting action is the display of the second anchor. In a hypermedia system, the link concept is generalized. A link anchor may be any type of object, and activation may cause a pop-up menu to be activated from which the user selects one of several possible operations.

The remaining of this section is organized as follows. In the first sub-section we describe the general operation of the linking mechanism. Next, we describe how tools are adapted to become part of the Henry System. We then describe how links can be customized and the system extended to perform new operations via the activation of links. Finally, we describe how users interact with HUBs in the system.

### 5.3.1 The Henry Linking Mechanism

The linking mechanism in a system like Henry needs to comply with two major requirements:

1. A facility for creating links quickly and easily needs to be in place.
2. The linking mechanism needs to be specifically adaptable to work with a diverse spectrum of interactive tools with different hypertext support capabilities.

As in the Intermedia system [Haan92], our goal for the Henry environment is to create a framework where hyperlinks are as easy to do as cut and paste within personal computer software. However, we need to achieve this in a heterogeneous environment where each application is developed using a different set of user interface and inter-tool communication libraries. This has been called an *open* hypermedia system [Pear89]. To integrate all design and documentation tools in use, the linking system must support a wide spectrum of tools with different hypertext capabilities. On one end we have modern documentation processing systems, such as FrameMaker, that already have built-in hypertext and remote command invocation support. On the opposite side, we have interactive tools with command based interfaces and no inter-tool communication capabilities. Yet we would still like to integrate these with minimum effort.



In Henry, live links exploit active messages to define the actions and the link anchors. These are sent between applications using the HUB services. This interpretation of open hypermedia merges well with the concept of a information-based design environment described above. Figure 5.6 contains a diagram of the Henry design environment, as seen from the perspective of its users. The HUB is seen as a common background process, used by different sets of tools.

Henry requires that individual applications provide the support for defining and activating anchors. The only support it provides is that of sending the commands using the message system. If an object displayed by an application is to be used as the end-point to a traditional hypermedia link, then the application must provide a way for that object to be re-displayed when a command is received by the tool. If the object is to be used as the starting-point of a link, then the application must provide the support for loading and sending the active message associated with the link to the HUB.

### **5.3.2 Making Design Tools Hypertext-aware**

The work required for integrating an existing tool into the system depends on the capabilities it already offers for defining anchors and for inter-tool communication. In documentation tools that already support hyperlinks, the existing anchors can be re-used. This is how we integrate FrameMaker into our system. We use its built-in hypertext markers as anchors and associate with them new commands that send active messages to the HUB.

In design systems, where design data representations associate annotations (or property lists) with design objects, good integration is relatively simple to achieve. In VLSI editors, support for entering and displaying of annotations in schematic and layout representations is almost universal. A straightforward approach would consist of storing shell commands in the annotations. Then we use or add a new user command to execute the contents of any object's property. Henry has a simple shell script, also called *hmessage*, that formats the arguments into an active message in the HUB MTL format and sends it to the user's HUB. This shows the simplicity of the mechanism for defining anchors in interactive design

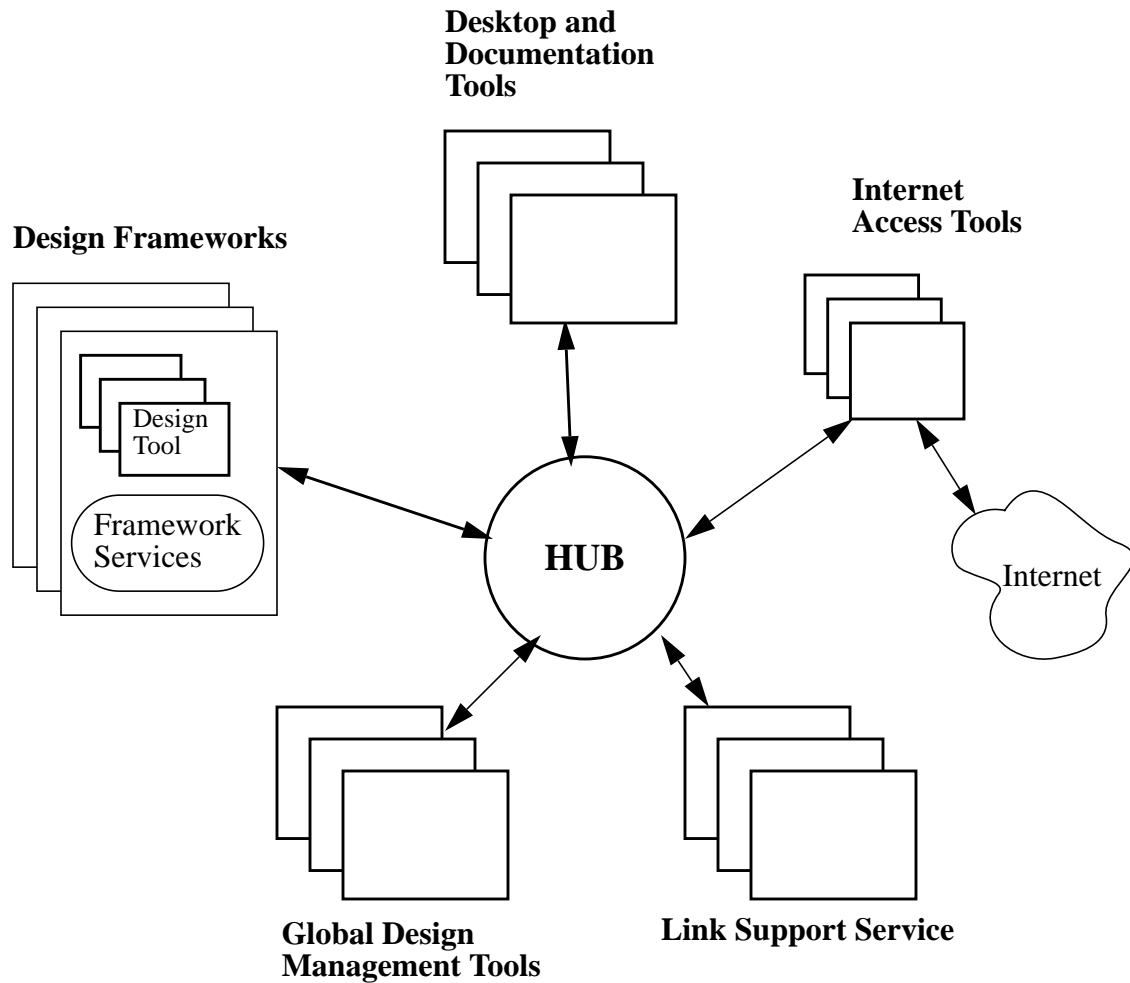


Figure 5.6 The Design Environment from the Users' Perspective

The Henry System user sees the design environment as a set of multiple design frameworks, desktop and documentation tools, and tools to access Internet services. All the tools are connected the user's HUB. The HUB is seen as a service that provides support for creating and managing links between pieces of information. In addition to the design and documentation tools, users may also need to interact with new design support services that make generate and produce useful information related to links and their activation.

tools. In general, the tool integrator must provide a more sophisticated user interface to activate links, hiding the details of defining and storing active messages in design object properties. Their contents can be executed as shell commands.

For those VLSI layout tools which do not support annotations but do allow for the placement of labels, an alternative is to use a special layer where labels are interpreted as hyper-text links. To create a link at a given point in a layout, the user places a label with the corresponding active message invocation command. Link activation is made as described above, by “executing” the label as a shell command. We used this approach to integrate the Magic layout editor into the Henry system (see Figure 5.7).

To integrate some tools, simpler mechanisms to define live links are necessary. For instance, in command interpreter-based interactive tools it is not possible to define anchors. However, it is still possible to define mechanisms for activating live links. To integrate the SPICE3 simulator, we added a new macro to the simulator’s command interface. To activate a link, users type in the macro command. This macro executes the same shell command used by Magic to send a *hmessage* to the HUB. The command then generates the active message that is sent to the HUB.

### 5.3.3 Link Configuration

In our description of the mechanisms for following and creating links, we have omitted many information details that are needed to completely characterize a live link. So far, we simply defined *what* information is displayed when a link is activated. A link representation also needs to include other information, which is generally ignored given the assumptions upon which hypermedia systems are built:

- *Where*: this information indicates the geometric area where the object will be presented upon activation of the link. It may be an application geometry specification on a X server display, or it could be a page or an inset in an application displaying multimedia documents.

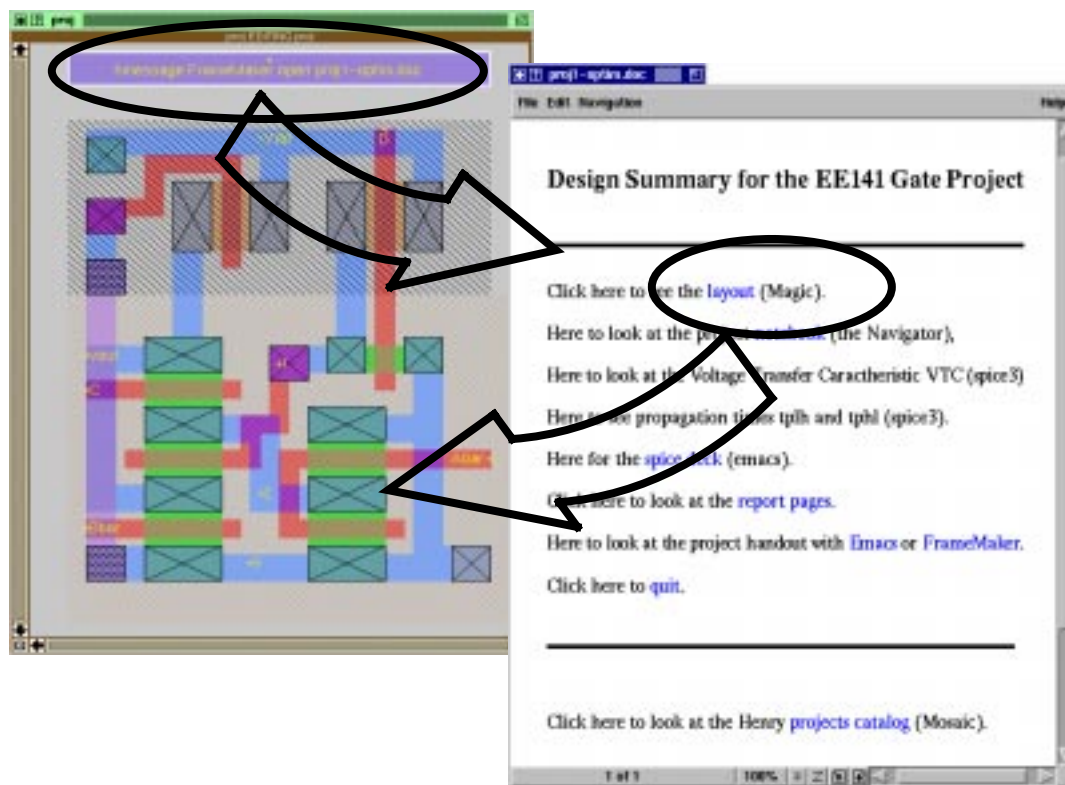


Figure 5.7 Magic as a Hypermedia Tool

To integrate the Magic layout editor into the Henry system, we implemented two extensions to the editor's source code:

- 1) a new command, *leval*, was created to evaluate the contents of a "label" as a shell command. In the example shown here, the designer of a logic gate placed the label with text "*hmessage...*" on the layout shown on the window on the left. This corresponds to a MTL message accepted by the HUB. When *leval* is invoked with the label selected, the label text is passed to a shell command that sends it to the HUB. The HUB in turn directs FrameMaker to open the document on the right. This document has hypertext links to various sections of the documentation and to other tools.

- 2) The built-in hypertext facilities in FrameMaker can be used to send commands to other running programs. We modified Magic to accept commands both from its command window or from a TCP/IP socket. When the document on the right references the layout, the designer has placed a hypertext command that sends a message to the HUB. That message is then converted into a command to open the layout file which is passed to Magic through the TCP/IP connection.

- *When*: In general, links are to be activated immediately. However, in a wide-area design environment such as the Henry System, a live link could involve access to resources not immediately available.
- *How*: this information indicates what tool will be used to display the information. It could be a generic name to be resolved by the HUB, such as a VHDL simulator, or the specific path name of the executable of a running program.

This shows that there is more information that needs to be considered when activating a link and that an extensible mechanism for specifying links is required. Henry's active messages, based on extensible formats and a communications language, offer that mechanism.

#### **5.3.4 User Interaction with HUBs**

A HUB acts as a link server that relays users' requests for accessing information to the tools that manipulate it. HUBs are extensible: new support services can be implemented as handlers in the MHL. Independent processes can also be invoked from the handlers as a side effect of link activation. One obvious extension would be to keep a history of all the links followed by the designers in a given project. This history could be browsed and used for multiple purposes useful to designers, such as producing a graph of dependencies between pieces of information.

HUBs are designed to run in the background. They do not have a top-level window from which users interact. However, we designed HUBs assuming that a human operator is assigned to them. For the HUBs that control the communication with users' tools, the operator is the user. A HUB requests its operator's assistance when it does not have all the knowledge required to dispatch an active message. User's HUBs frequently pop-up dialogs requesting information on how to proceed. One of its handlers is the *session manager*, a special handler that asserts that tools are started before active messages addressed to them are sent. If a tool is not running, the session manager starts the tool and delays deliv-

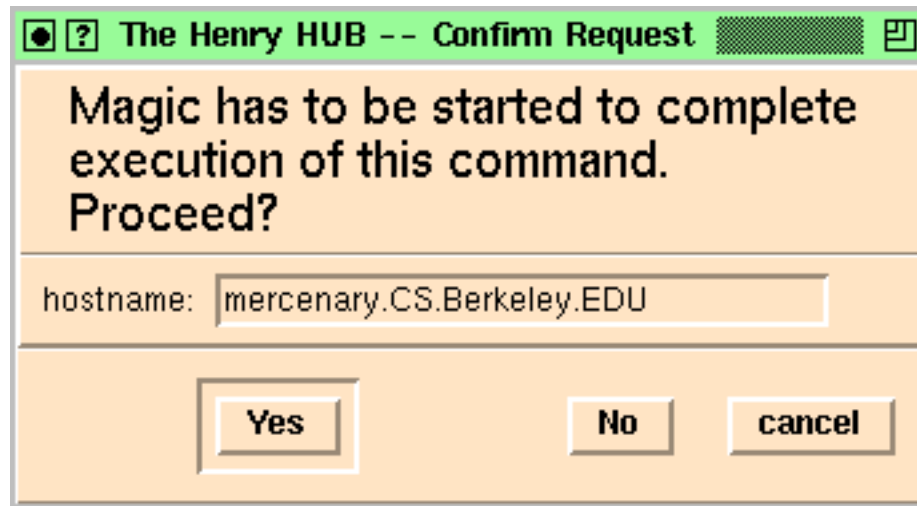


Figure 5.8 The HUB's Session Manager Dialog Window

HUBs run in the background, but can pop-up dialogs requesting assistance when messages require exceptional processing. When the HUB attempts to send a command to a tool (Magic in the figure) and the tool is not running, a dialog pops up, requesting the designer to confirm that he wants to start the tool and select the host where it will run.

ery of the message until is ready to process it. In the current implementation of the Henry System, this always causes the dialog shown on Figure 5.8 to be displayed. From the dialog, the user can cancel the activation of the link or select the machine in the network where the tool displaying the requested anchor will run.

## 5.4 Henry as an Infrastructure for Building Tool Ensembles

In Chapter 1, we discussed the concept of a *tool ensemble* in the context of VLSI design. We view a tool ensemble as a set of multiple tools that cooperate and synchronize their executions to provide assistance to a user in a single task.

In today's environments, the existing mechanisms for creating ensembles only allow for specification of sequences of tools. Some process management systems capture tool executions that may run in parallel, but there is no model for defining how the tools synchronize their executions. Another important need is a way to specify how a tool's main

window is embedded into the windows of other applications to create a compound active document.

In our view, a combination of services and conventions is necessary to support the requirements for creating tool ensembles composed of multiple graphic interactive applications:

- *Inter-tool communications.* In Henry, these are HUB-based for locating tools and sending commands between the tools.
- *Tool Embedding.* To create compound documents, a tool should be able to display its window within another tool's main window. For example, this makes it possible for a schematic editor to open an inset within a document rendered by another tool. Support for embedding requires an extensive set of conventions and support software adhered to by the tools. As described in Section 2.4.2 on page 35, all the major operating systems support (or soon will support) this capability.
- *A common user interface paradigm.* All tools should adhere to the same interaction style. We adopted the documentation paradigm. We view every window as a viewport to a piece of documentation in a multimedia information system.
- *A glue mechanism* to bind tools together. A common model and specification method is required for describing tools' executions synchronization commands exchanged and how tools windows are embedded within each other. In this model, it should be possible to specify any constraints required to make a set of tools behave to the user as a single application. In the Henry System, we also use an extension of the Tcl language for gluing the components of a tool ensemble.

In the context of the Henry System, active documents and tool ensembles are synonymous. Active documents are multimedia presentations written as scripts in this gluing language. The language aspects related with the representation and presentation of active documents will be detailed in the next chapter.

The remaining of this section is organized as follows. In the first subsection, we discuss the attributes of tool ensembles and the importance of having ensembles where no tool has a special role. Next, we discuss our view of the role of extension languages as a fundamental component for building tool ensembles.

### 5.4.1 Symmetric Tool Ensembles

From the HUB point of view, there is no central tool coordinating any aspect of the other tools. It is up to the system integrator to create the necessary composite tools and to define the constraints on them. We call these *symmetric tool ensembles*, since no tool has a pre-established role of container, communications broker or special service provider.

This contrasts with typical VLSI design and generic hypermedia systems, which are organized around a common front-end that can call external tools to extend the basic tasks supported by the main tool. In our application domain, at different times designers work on design, documentation, or both simultaneously. In this environment, no design or documentation specific tool has a special role.

Tools need to respect common conventions to be proper components of a symmetric tool ensemble. The most important consideration with regard to intertool communication is that tools have to be designed to operate both as clients and servers.

This is not typical in current VLSI CAD environments. Tools that have built-in intertool communications generally work either as clients or servers, but not both. A good example is the Octtools framework [Harr86]. VEM, the front-end to this system, only accepts commands from applications started from its console window. In computer networks terminology, it does not register its listening sockets and does not listen in a *well-known* socket. As a result, software systems that may need to re-use the graphic editing facilities of VEM are forced to run as applications invoked from VEM after it is started.

With respect to containment, it is also unusual to observe tools whose main window can work both as a container and contents to the windows of other tools. For example, Frame-



Maker, has a built-in capability to invoke tools to change the contents of figures displayed within FrameMaker documents. However, it is not possible to have FrameMaker running as an inset within other applications.

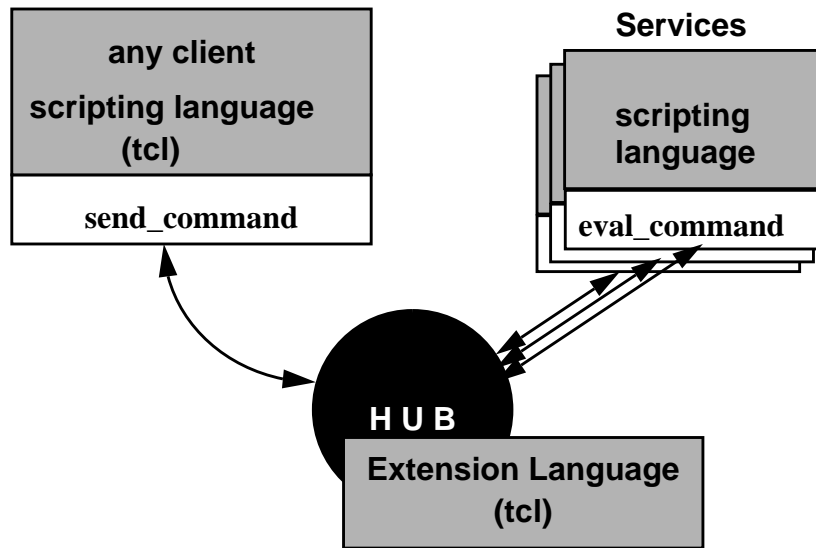
## **5.4.2 The Role of the Extension Language**

Henry's support for building tool ensembles is far from complete. Tool embedding is a particular need that must be further developed. However, the existing system already shows the critical need for an Extension Language (EL) for building tool ensembles of interactive tools and how the language could evolve to support the missing features. The key feature in Henry's support for tool integration is the availability of a powerful EL that can be embedded within the tools, and the communications service for sending the command language scripts to the tools.

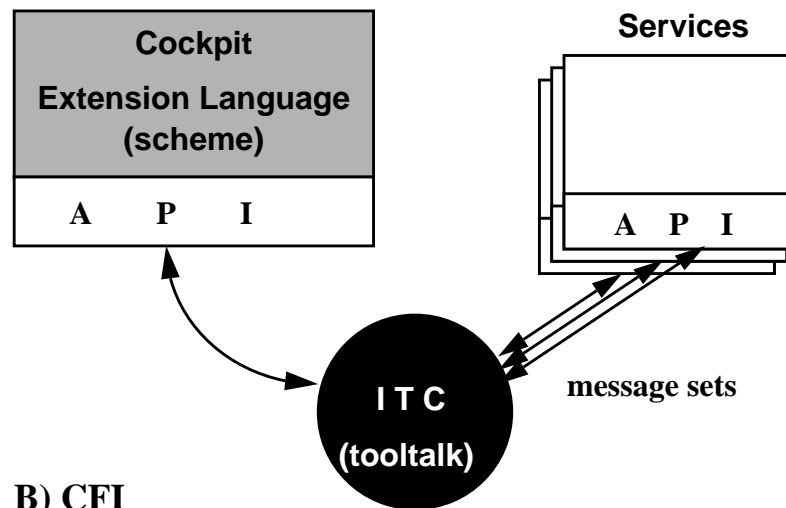
Our view of an extension language differs from CFI's perspective (see Figure 5.9). The CFI Framework Architecture Reference Model defines an EL as a component of a central *cockpit* that includes the session management component of the Framework. However, in the CFI architecture there is no reference to an EL for the intertool communication service or its availability in the tools. We advocate having the extensibility based on command language interpreters available not only in a specific service, but in all tools and services in the design environment.

### **5.4.2.1 Extension Languages and Concurrency Control**

The Henry HUB is founded on stateless communications between clients and servers. However, this makes it necessary to provide an additional concurrency control mechanism. Interactive tools accept commands from the user interface concurrently with messages received from the remote command execution interface. As a result, sequences of commands sent to an interactive tool for remote execution may be disrupted by the execution of commands directly typed by a designer.



A) Henry



B) CFI

Figure 5.9 The Extension Language in Henry and CFI environments

The figure contrasts Henry's architecture with CFI Architecture regarding the role of the extension language (EL). Henry (A) supports a (preferably extensible) scripting language interpreter in every component of the environment. It provides access to the tools' command interpreters. The Henry EL is based on Tcl. The CFI architecture (B) considers a single EL interpreter running at the framework's *cockpit*. The EL is based on Scheme, a LISP dialect. The language is extended with commands to invoke the functions of the various Application Program Interfaces (API) offered by the tools.

There are several ways to deal with this problem. All solutions can be reduced to a capability for executing an uninterrupted sequence of commands at the receiving end. In some instances, tools support receiving multiple commands in a single command line. We could translate a sender's message into a message to the receiver containing a sequence of commands at the HUB. Some tools support macros. In this case, we could translate a message into two commands, one for loading a pre-defined macro and another for executing it. For tools that do not have an embedded command interpreter or a macro facility, implementing support for concurrency control requires architectural changes to the applications.

#### **5.4.2.2 Extension Languages and Inter-operability**

Our experience with a common extension language to program the environment's inter-tool communications service indicates that it is a better mechanism for controlling system interoperation in CAD environments. Here, the crucial problem is the need to exchange control information between independently developed systems, as data interchange formats become increasingly accepted.

Henry's approach dramatically reduces the effort necessary to create an interface for inter-tool communication. To interface foreign tools in Henry, all that is required at the procedural interface level is the capability to send a command for execution at a remote tool. On the other hand, Application Program Interfaces (API), such as those used by CFI, require a complex process of development of long specifications of remote procedure calls. The capability to send commands to remote tools does not provide the same level of interoperability as would a completely specified API supported by all tools. On the other hand, it is much simpler to implement. The CFI has been attempting to develop a common interface for the last years without visible success. An equivalent effort would be required at the EL level. However, the low-level of interoperability provided by the capability to send commands to remote tools serves our purpose of being able to exchange control commands and synchronize executions of independent tools.

## 5.5 Summary

This chapter reviewed the architecture of the Henry System and how we have adapted existing design and documentation tools to become part of an integrated environment. There are three key aspects in the Henry architecture: its support for heterogeneity, its approach to make tools become part of a hypermedia system, and its extensibility.

Henry supports heterogeneity in many ways. From the communications perspective, the use of MIME-based messages and Internet message protocols facilitates cooperation between organizations with very different design environments. At the local level, Henry defines a common command interface to all the tools. Active messages enable inter-tool communication by encapsulating the data and the commands to the tools in MIME messages that can be sent across organizations.

In the Henry environment, hyperlinks' actions are defined by active messages. Link anchors are commands that can be stored as markers or executable properties in the tools. This requires minimal modifications to existing tools, while offering a powerful linking mechanism and the capability to make modifications to data and documents without affecting the link information.

Extension languages are an essential component of the Henry architecture. They are used in active messages, as the interface for communication with the tools, as the mechanism for extending and customizing the environment, and as the glue language for describing active documents.

In the Henry System, active documents are ensembles of tools used to create and display multi-media presentations involving design and documentation tools. So far we have only discussed the infrastructure we have built for supporting active documents. The topic of the next chapter is the tools developed for handling these documents.

“When Prince Henry’s mariners went farther south than Europeans had ever gone before, they faced new problems... The cosmopolitan community at Sagres helped to make the quadrant, the new mathematical tables and other novel instruments, which became part of Prince Henry’s exploring equipment.”

— Daniel J. Boorstin, “The Discoverers”

## Chapter 6

# Tools For Integrated Design and Documentation

This chapter presents the tools and authoring techniques developed for the Henry System. We describe the conceptual model of the Navigator, an electronic design notebook conceived as a test-bed for exploring the integrated VLSI design and documentation paradigm.

### 6.1 Introduction

In the previous chapter, we presented various tool integration mechanisms available in the Henry System, such as *executable annotations* as a method to anchor hyperlinks to objects displayed by CAD tools. In this chapter we focus on authoring techniques and tools that fully exploit the idea of integrating the design and documentation of VLSI systems.

Our model for integrated design and documentation is founded on the observation that there is a one-to-one correspondence between entities in the conceptual models used in documentation processing and electronic engineering systems. For instance, in word processors, non-text objects in diagrams can be drawn using a palette of tools. Each tool

draws a specific graphic object. This is similar to what VLSI designers do: to add or transform a piece of design data, they invoke a tool to perform the operation. Table 6.1 shows the correspondence between the most important concepts in both domains.

<b>design concept</b>	<b>document representation</b>
private workspace, group workspace, library/archive	notebook, project binder, manual
design alternative	conditional text
design configuration	document version
design methodology	document template
context (current configuration, activity, tool, view,...)	context (cursor, default font for paragraph,...)
hierarchy	sectioning, imported documents
instantiation	import by reference
tool space	drawing tools palette
activity history	session log
object attribute	annotation, link, footnote

Table 6.1 Design versus documentation concepts

We observe a one-to-one mapping between every fundamental design concept and a related concept used in documentation processing systems.

Of special interest to us is the similarity of the techniques used to handle complexity in the two domains. Both extensively explore the use of contexts. When a user types a character in a word processor's window, the position in the document where the character will be introduced and the font type that will be used are inferred from the defaults already set. Similarly, when a designer checks-out an object from a database, he does not have to indicate the version or view he wants to access. The designer's location in the design space is inferred from the previous commands given to the database. In our view, a document's

visual representations for context and structure could be extended to indicate the designers location in the design space. For instance, a design with multiple configurations could be associated with a document in which each page represents a configuration. Moving to a new page in this document has the side-effect of switching to a new default configuration in the design.

Given the existing trend towards offering user interfaces inspired by a documentation metaphor and the structural similarities among electronic design and documentation, we believe that documentation should be the common metaphor for manipulating design information. The direct conceptual correspondence between design and documentation gives us a new opportunity to create better interfaces to manipulate the information in an extremely complex environment.

To explore these ideas, we developed a prototype electronic book which makes use of the infrastructure provided by the Henry System. The Henry notebook, called the Navigator, is conceived as an editor for documents that can contain active messages anchored to selected objects. The Navigator offers an user interface that was specially designed to facilitate browsing and creation of active messages inside a document.

The Navigator has another distinctive feature. Existing documentation processing systems support authoring and browsing as two independent operating modes. In the Navigator there is no such separation. Active messages can be added, displayed and modified directly while browsing a document. This is an essential feature in our application domain, since the document authors — the VLSI designers — also constitute the audience of the documents produced with the electronic notebook.

The remaining of this chapter is organized as follows. Section 6.2 presents our operational model for a design environment founded on an electronic notebook paradigm for browsing a project's information web. Section 6.3 introduces a new conceptual model for representing and handling active documents, developed for the Henry System. Section 6.4 discusses the design of the Navigator. Section 6.5 closes the chapter with a summary of

the main ideas. In the next chapter, we describe how the tools developed for the Henry System have been used.

## 6.2 Integrated Design and Documentation with Electronic Notebooks

Figure 6.1 shows the flow of information in the Henry environment. Design data and documentation are organized as an information web. The Navigator is an electronic counterpart to the engineering notebooks used by designers today. It is used to annotate the design as it evolves and doubles as a browser for this complex web of information. From the documents, users access the tools, design libraries and design management aids. Interaction with the tools produces even more design data and documentation ready for incorporation into the design information web.

In this section, we begin by discussing the principles we follow for producing active design documents with electronic notebooks. Then, we present the kinds of authoring aids that can be provided in general, and conclude with a description of those we developed for the Henry system in particular.

### 6.2.1 Organization of Documentation in Electronic Notebooks

In our view, the documentation produced with electronic notebooks should follow the minimalist principles advocated by John Carrrol for creating instructional documentation [Carr90]. The objective, according to its proponent, is *to minimize the obstrusiveness to the learner of training material*.

Marc Rettig proposed the adoption of this principle for producing documentation of software systems [Rett91]. There are three aspects that must be considered in the design of minimalist documentation: 1) allow learners to start immediately on meaningful realistic tasks, 2) reduce the amount of reading and other passive activity, and 3) help to make errors and error recovery less traumatic and more pedagogically productive.



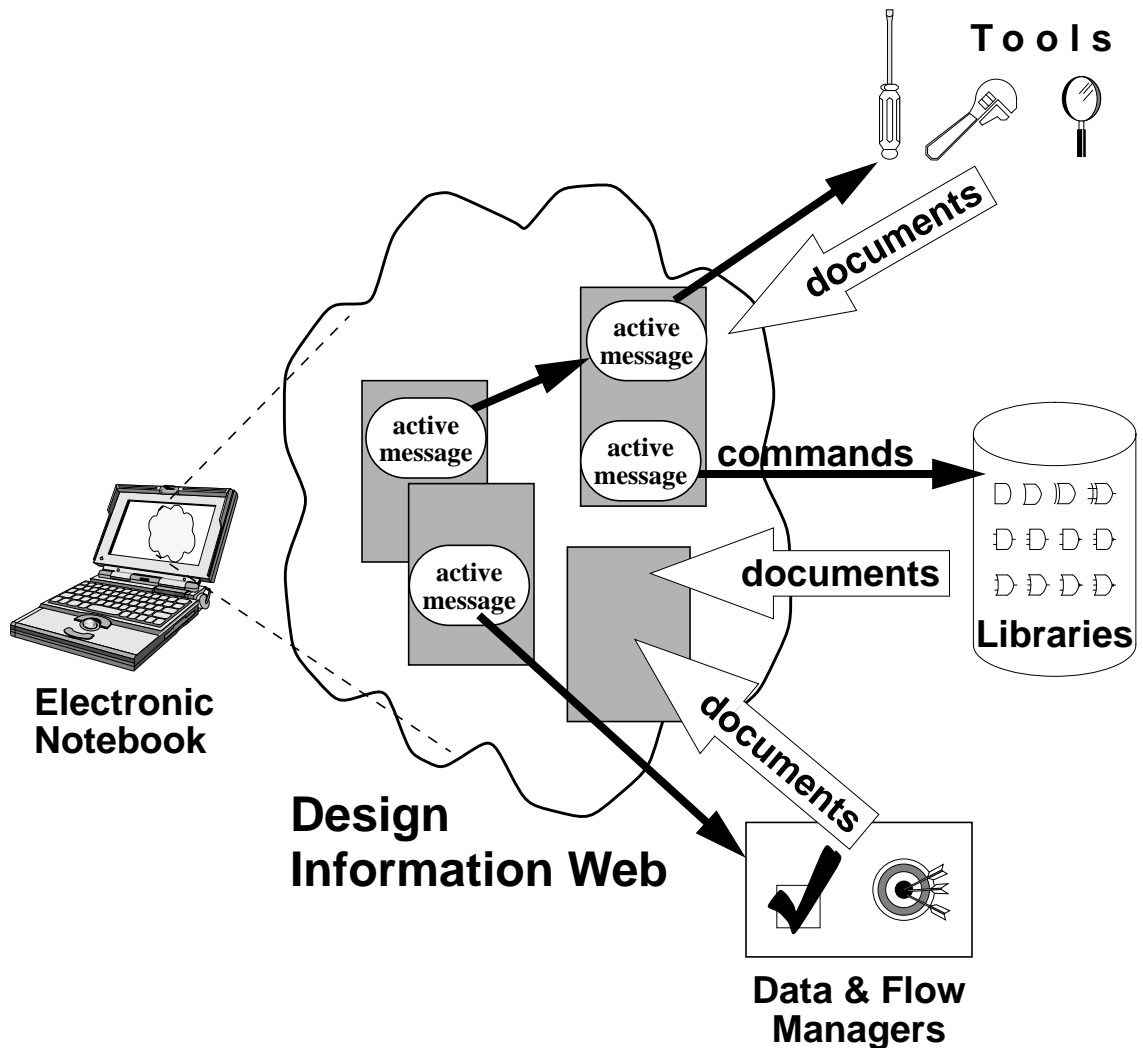


Figure 6.1 The Information flow as seen from Henry's Electronic Notebook

In the information-centered environment of the Henry System, designers access tools, libraries and design data, and flow managers via documents. These are connected into an information web through active messages stored within the documents. Active messages fire the commands to the tools. Command executions create additional data, which is integrated into the web as new documents. Henry's electronic notebook, the Navigator, is a program that runs on the designers' workstations. The Navigator is a tool specialized in the interactive edition of active documents and browsing of the information web of the Henry System.

We adopt similar goals for the organization of design documentation produced with notebooks:

1. Designers should have quick access to design tools and data. The command used to apply an operation to a piece of design data should be easy to invoke from the notebook. The resulting logging information should also be easily incorporated into the notebook. For instance, to capture a sequence of design operations, it should be possible to create with a single command a window running a shell that has its execution log automatically captured into a file. The file would be editable from the notebook to create annotations to activated programs and links to opened and created files.
2. Annotations should be small and made self-describing by incorporating links to the data and tool operations they are intended to document. For instance, instead of creating a table in a document with a list of the parameters of each device model used in a simulation deck, it is better to create a note with a link to the point in the simulation deck where the models are described. This requires less work, enforces consistency between data and documentation, and makes it easier for the designer to change one of the parameters if required.
3. The notebook should provide assistance for the construction of active documents and the debugging of errors in the authoring of these documents. For instance, if a link activation to send a command to a layout editor from the notebook fails, the notebook should provide detailed explanation of what failed in each of the processes involved and the steps to fix it. In Henry, these would be the Navigator, the HUB and Magic. The link activation could fail for many reasons, such as the HUB not running or the file containing the layout being inaccessible. A diagnostic message saying “cannot activate link” is not as informative as a message saying “Cannot open layout. File /user/adder.mag is protected.”

## 6.2.2 Authoring Active Documents with an Electronic Notebook

Creating the information web for a design has many similarities with authoring in a generic hypermedia system. For hypertext-based documentation systems, no unique paradigm exists for writing documents. Bottom-up and top-down processes are used equally [Niel90]. Generic hypertext systems have only limited aids for document builders, such as document consistency checkers to verify that there are no dead-end nodes, or browsers for the document structure.

In Henry, we provide authoring aids that complement those available in generic hypermedia systems. In addition to the electronic book for interactive edition and navigation on the design information web, we envision three types of aids for addressing designers' specific needs:

- *Document building toolkits*. These are sets of tools that automate the process of creation of design documents.
- *Active clip-art*. These are libraries of active document templates that can be re-used to document similar designs.
- *Documentation agents*. These are programs that assist designers in the production of documents and organization of the information web.

We discuss these different types of aids in the remainder of this section.

### 6.2.2.1 Document Building Toolkit

The idea of a toolkit for building hypermedia documents and applications has been proposed for general purpose hypermedia authoring [Sher90, Putt90]. We use the same approach, but focus on the development of aids particular to our specific application, the documentation of hardware designs. There are three types of tools in the Henry toolkit for generating VLSI specific documentation:

- *Document generators.* These are simple tools to produce documents that summarize information about a design or generate documents that explain the design's structure. As an example, in a top down process for writing documentation, we could easily write a tool that automatically generates a document, having a page summarizing the data about each component in a design and *links* reflecting the hierarchy. VLSI tools that compute statistical information about a design, such as *chipstats* from the Octtools framework [Harr86], are natural components of these toolkits.
- *Modified design tools to help navigation within documents.* An example of a VLSI specific aid is a layout browser adapted to display the documentation about a component when a user presses a button over the area defined by its protection frame. In the Henry System, this is supported by our modified version of Magic. In this tool, we can annotate objects with references to active messages for displaying related documentation. Messages can then invoked at the request of the designers<sup>1</sup>.
- *Tools and macros to automate the creation process.* These are accelerators for automating the process of generating links. For instance, in the Henry System we have automated the process of adding links to layout files and simulation plots. A user can create a button to display a layout file in the current page with a single command, both in the Navigator and in FrameMaker (see Figure 6.2.).

#### 6.2.2.2 Active Clip-Art

Other aids are libraries of pre-defined sections of documents and document templates, that designers can cut-and-paste into their specific documentation. This is analogous to the clip-art libraries distributed with word processors, with the difference that active clip-art libraries include active documents.

Active clip-art can be used by designers for several purposes. Document templates can be pre-defined for generic tasks. In Henry, pre-defined notebook pages offer a simple way to

---

1. This mechanism has been described in more detail in Section 5.3.2 on page 89.

FrameMaker



Navigator

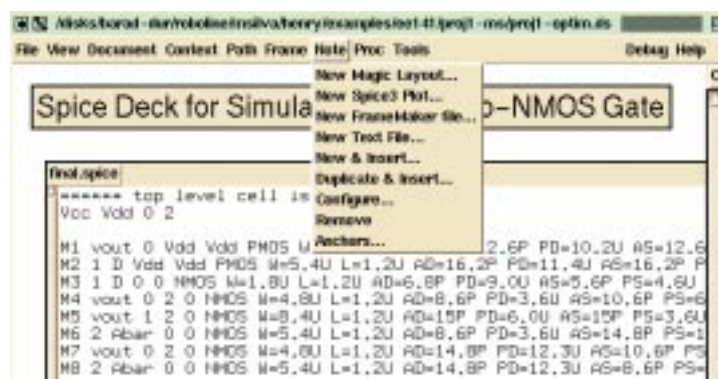


Figure 6.2 Domain-specific Accelerators for Creating Active Documents

In the Henry System, we have extended FrameMaker with new macros for automating the process of creating links. These can be invoked from a new pull-down menu. The same technique has been used in the Navigator. With a single mouse click, a user can generate an active message to create the windows from the design tools that display a given file. The commands also associate each generated active message with a “button” or an “inset” that is automatically mapped into the current page.

include points of access to data and tools from the Navigator. In effect, this converts design into the process of changing or filling-in the blanks of an electronic document template. Templates can also be used to enforce or guide the use of a given design methodology within a project team.

A good example for active clip-art is the application notes that manufacturers publish to describe how to integrate the components they produce into systems. Application notes could be made available as active documents including the data files and tool commands. This way, designers could efficiently reuse the documentation and methodologies embodied by the demonstrated application.

In our experiment with Henry as an integrated design and documentation system for teaching VLSI design, described in the next chapter, we developed active templates for the laboratory sessions reports. We used pre-defined active documents as a means to describe, automate and document the interactions that students had to perform with design tools (see Section 8.4.1 on page 159 for a detailed discussion).

### **6.2.2.3 Agents**

We envision the use of agent software [Maes94, Etzi94] for helping in the creation of the information web of VLSI designs. However, the adaptation and use of agents software in our domain goes beyond the scope of this dissertation. Here, we simply make the case for using agents as a potentially valuable aid for helping designers in finding and generating design documentation.

Agent programs could be set up to work in the background, monitoring designers actions, suggesting and performing operations on the design, and finally producing human-readable documentation of the design decisions.

Agents could be also employed to locate and retrieve information necessary for the design, such as components, their models and representations, or manufacturing services. We believe that the Henry System, with its interfaces for viewing heterogeneous design information as documentation distributed on the WWW<sup>1</sup>, provides an initial framework for developing this class of agents for the electronic design domain.

---

1. WWW — the World Wide Web [BL94b]

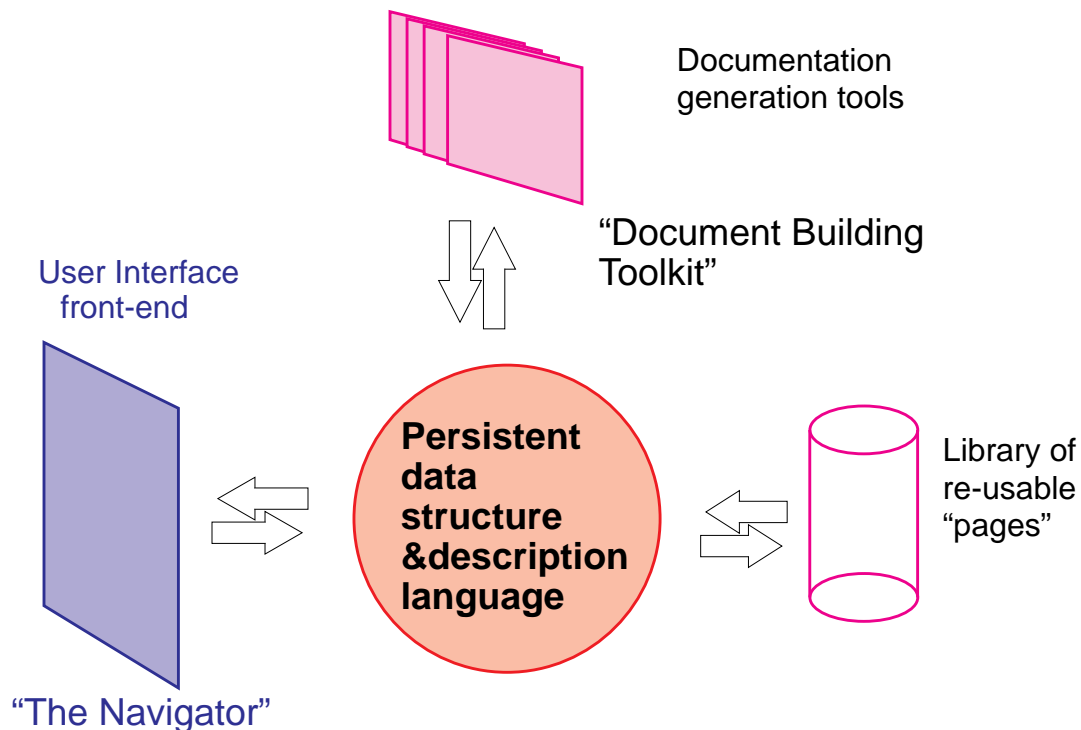


Figure 6.3 The Role of the Document Description Language in Henry

Henry's electronic notebook, the Navigator, uses DocScript, an extension to Tcl/Tk we developed for describing active document. The language is supported by a library of routines to parse and write the descriptions from/into files and to manipulate the data structures generated by the language constructs. The same language is used by the other tools of Henry's Document Building Toolkit.

### 6.3 Document Representation and Manipulation

We developed a new conceptual model for active documents for the Henry System. Documents in this model are described in a programming language. The language is supported by a procedure library that offers functions for 1) loading a document description from a file into a data structure, 2) interactively change this data structure using the language commands and 3) save the data structure back into a file. The Navigator and other tools we developed for Henry to create and manipulate active documents make use of this common representation and support library (see Figure 6.3).

The Henry active documentation description language, called DocScript, is an extension of Tcl/Tk [Oust94]. We use a programming language for describing active documents

because we wanted to exploit the full generality offered by embedding complete programs within a document. The same approach was used in Ness, an extension to the Andrew Toolkit to support active documents [Hans90]. Our approach for developing a document description language follows the strategy of the designers of PostScript [Taft90], an extension of the language Forth [Kell86] with new primitives for describing the contents of documents to be sent to printers. We viewed Tcl/Tk as an user interface description language and extended it to describe active documents containing dialogs and program invocations.

In the remaining of this section, we start by presenting a basic conceptual model for describing active documents. This model is based on hypermedia and only contains non-design specific primitives for describing active documents. We then present higher-level concepts that we added to this model for ensuring the consistency of the documentation and design data. Finally, we compare DocScript with SGML, the ISO standard for on-line documentation representation.

### 6.3.1 Basic Conceptual Model for Document Representation

The basic conceptual model for representing active documents used by the Navigator is based on hypertext. The model is inspired by the HIP hypermedia system developed at U. C. Berkeley [BSB90].

Each concept in this model has an associated DocScript command which, when invoked, creates a named instance of an object representing the concept. The concepts of the basic model and the associated DocScript commands are shown graphically in Figure 6.4.

A *frame* corresponds to a page in a traditional document. In the same way that a traditional page is partitioned into paragraphs and figures, a frame contains a set of *active notes*, or simply *notes*, that are organized spatially within the frame through a *binding*. A binding is an instantiation of a note within a frame, using the binding's geometry specification for the



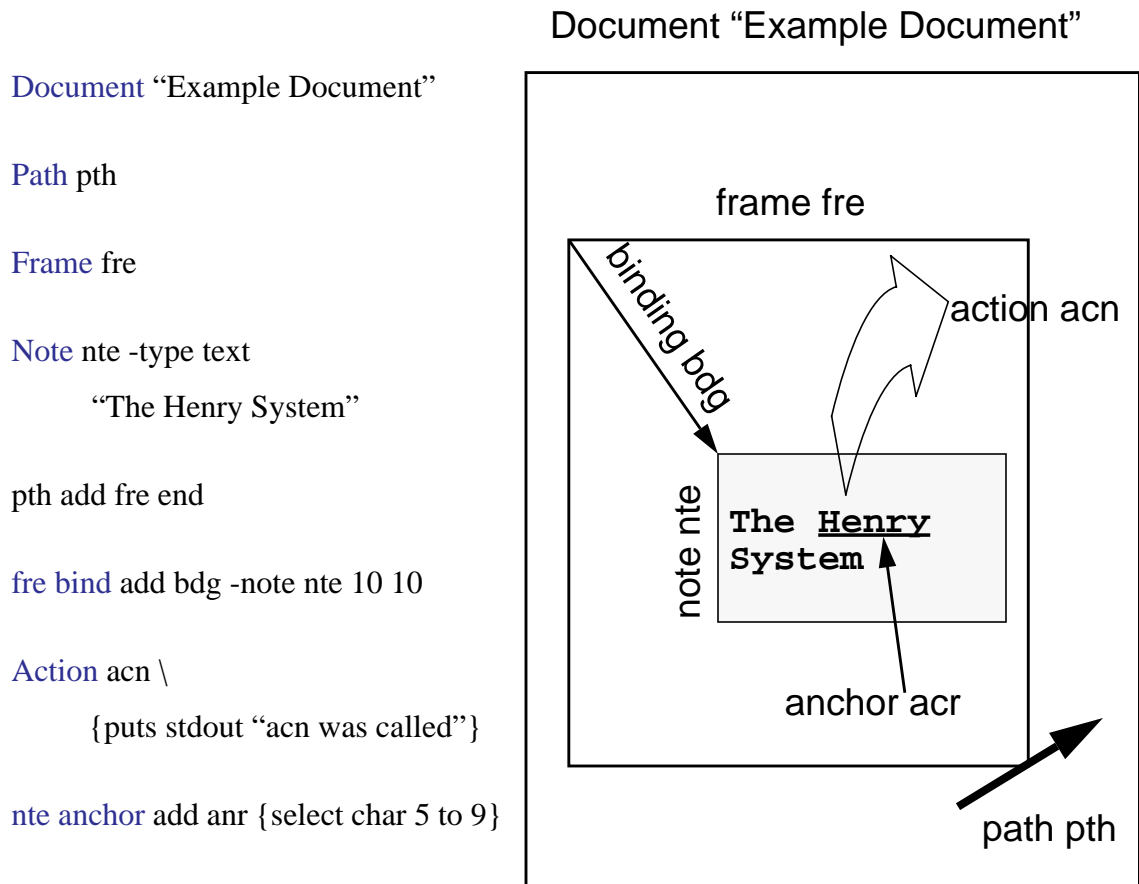


Figure 6.4 Code for Describing a Minimal DocScript Document

The figure shows the DocScript description of a very simple active document and the visual representation of the objects created by this description. The description reads as follows: *Document “Example Document”* contains a *path* named *pth*, a *frame* named *fre* and a *note* named *nte*. This is of type *text* and contains the string “The Henry System.” The *path* contains *frame fre*. The *frame* displays an instance of *note nte* at point 10 10 in the document’s coordinate system, referenced by *binding bdg*. The document also defines *action acn*, which prints string “acn was called.” This *action* is invoked from *anchor anr*. This is defined as characters 5 to 9 of the string in *note nte* (the sub-string “Henry”).

note. A note is an encapsulation of a documentation object: it could be a graphic user interface object, such as a button or a control panel, or a running tool.

DocScript supports non-linear views of the documents through the concept of a *path*. A path is an ordered set of references to frames, used to reference an order for traversing a subset of the frames that compose a *document*.

The synchronization of the contents of windows running in separate address spaces is made through *anchors*. An anchor defines a region or a piece of data in the context of a note. An anchor can be a range of text or some generic object identifier. It is possible to associate *actions* (scripts or commands) that are run by *Henry* in response to pre-specified *events*, such as a mouse button click or a frame being flipped.

Table 6.2. summarizes the relationships between the concepts of the basic model.

Object	Definition
Document	{Frame} + {Note} + {Path}
Path	[*Frame]
Frame	{Binding}
Binding	*Note + geometry
Note	documentation data + {Anchor}
Anchor	{Event -> Action}
<b>Key:</b> {} = set Of, [] = list Of, -> = Map, * = pointer To	

Table 6.2 Summary of the Model for Representing Active Documents

The table summarizes the relationships between the various concepts used to represent active documents in the Navigator.

### 6.3.2 Integrated Design and Documentation Concepts

We extended the basic model for describing active documents presented above with new concepts with richer semantic contents. These are used to describe the operations that have to be executed for synchronizing the design data and its documentation. For instance, we would like to be able to have the design database switch its context information automatically. For example, when a user navigates into a page describing a past release of a given project, the related information should be displayed in the design tools that are open. The

problem is better introduced in the context of a simple design scenario. Assume a notebook containing the description of the design of an ALU, with two configurations, *FASTConfig* and *SMALLConfig*. The document is organized around two frames, *FrameFAST*, describing design configuration *FASTConfig*, and *FrameSMALL* describing *SMALLConfig*. When a designer scrolls the notebook from *FrameFAST* to *FrameSMALL*, it is necessary to send a command to the database to change the default configuration. This makes it possible to maintain consistency between the contexts in the database and what is seen through the notebook interface. When a design object is accessed for display or editing by a design tool in any part of the notebook, the correct version in the database is always accessed.

Design contexts are defined as extended concepts in the active documentation model. For their representation, we use reserved property names and procedures that are associated with the objects of the basic model. All DocScript objects have the following defined attributes:

- A *variable table*. Each table contains property names and values that are associated with the documentation object.
- An *activateHook*. This is a special attribute, also common to all DocScript concepts, that can define an arbitrary Tcl procedure to be executed when the object is activated. In DocScript, activation means the instant of time when an object is about to be rendered. For instance, a Document's *activateHook* is called when the document is opened, and a Frame's *activate hook* is invoked when the Frame is about to be rendered.
- A *deactivateHook*. This is another special attribute in all identical to *activateHook*, but with deactivation occurring at the instant symmetric to that of the activation of the object.

In addition, DocScript supports *persistent* procedures. These are defined like regular Tcl procedures, but are saved with the document on which they were defined when it is closed.

With this combination of extensions we were able to support the synchronization requirements between data and documentation described in the above example. To have the contents of an active document synchronized with the data in a design database, we could write a script with the database initialization operations and define it as a persistent procedure associated with the document's *activateHook*. Similarly, we could define a procedure to change the default database view and invoke it with a different view name as argument from each Frame's *activateHook*.

However, synchronization between design data and documentation becomes more complex when we consider interactions between an electronic notebook and a database. For example, we must be able to synchronize the actions of a designer looking into different configurations at the same time from a document and passing information between them. Context information from the documentation processing tool, such as the location of the insertion cursor, also becomes important. Consider the situation illustrated on Figure 6.5, where a designer is calling an editor on an element of the ALU displayed on *FrameSmall* of the example above, modifies it, and saves it at the current location of the insertion cursor. As the cursor is left pointing to some position in *FrameSMALL*, the consistent operation is to read the object from *FASTConfig* and append it to *SMALLConfig*. For this reason, the database resolution mechanism needs to be re-configured automatically from the documentation front-end, based on the parts of the document being displayed and the current location of the cursor. Similar situations arise in other scenarios, such as when a designer wants to create or visualize two alternatives of a design in different frames.

### 6.3.3 DocScript versus SGML

We chose to develop our own language for describing active documents because at the time this project started we did not have access to software to parse and manipulate SGML. For DocScript, we could reuse the Tcl language parser with relatively few extensions.

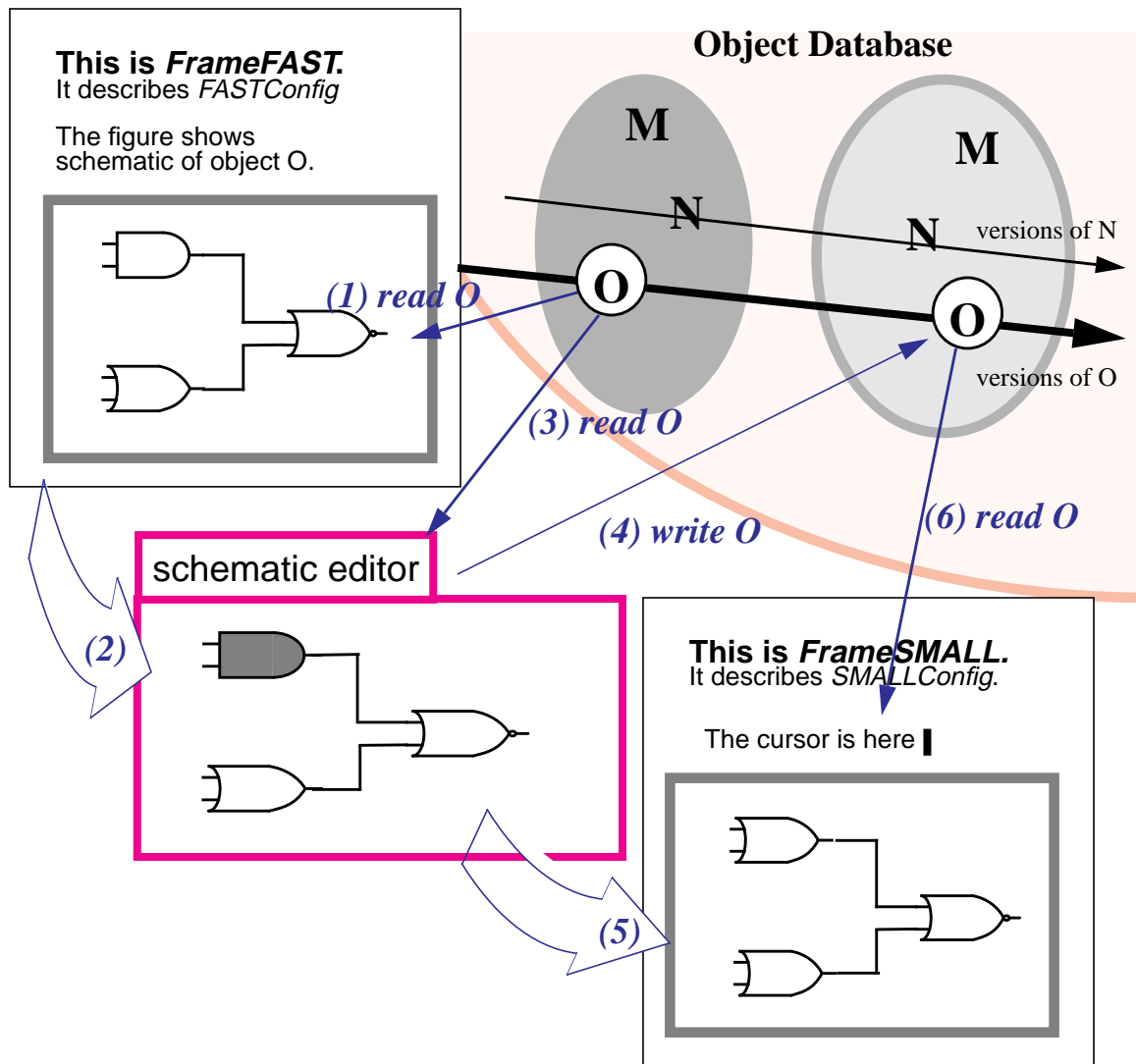


Figure 6.5 The Effect of the Cursor Position in the Design Database State

Names are resolved based on their location within the document upon reading, and on the location of the cursor upon writing. Assume a document describing a hypothetical hierarchical design with objects M, N and O organized in 2 configurations, *FastConfig* and *SmallConfig*. Consider the sequence of operations of the figure. When the cursor is left in *FrameSMALL*, clicking on the figure in *FrameFAST* will start a tool for editing the version of object O in Configuration *FASTConfig*. When the session with the editor is closed, a new version is appended to *SMALLConfig* in the database, and shown in *FrameSMALL* after the cursor.

DocScript documents can embed arbitrary user interface dialogs defined in Tcl/Tk. This makes our language very powerful for creating active documents with complex user interfaces. Creating an active document with a complex user interface in SGML would be much harder and would require the development of a complex extension to this representation format. And of course, the resulting description for the active document would no longer be standard.

However, given the popularity achieved by SGML in the WWW<sup>1</sup> and its adoption by the CAD industry to describe electronic components, we would now choose to sacrifice some flexibility and represent our active documents in this language. As a document description language, DocScript also has limitations that would require the development of large extensions to make it practical for describing full VLSI documentation. For instance, there is no support in DocScript for typesetting mathematical expressions.

## 6.4 The Navigator

The Navigator is a prototype documentation tool we developed for the Henry System. It operates as an electronic notebook for browsing design information webs and as an interactive editor for DocScript documents. We conceived it as a tool to operate in the earliest stage of documentation, for entering the initial annotations to a design during the exploratory phases.

The Navigator re-uses most of the visual elements of a typical multimedia documentation processing system, while adding extensions to support direct manipulation of the objects of our conceptual model for active documentation. In this section, we discuss the user interface design philosophy, oriented towards supporting simultaneous design and documentation.

---

1. the WWW uses HTML (Hypertext Markup Language) [BL93b], a SGML document type, as the primary document representation format.

In the Navigator, we emulate the user interface of FrameMaker as closely as possible. There were several reasons for this. First, FrameMaker already provides a user interface that has demonstrated high usability. Secondly, FrameMaker is currently the *de facto* standard for presenting on-line documentation in the electronic CAD industry. Most designers are familiar with this tool. Third, designers are likely to use FrameMaker in conjunction with the Navigator to display other documentation, such as on-line data-sheets of off-the-shelf components and tools' documentation.

The elements of the user interface of the Navigator are shown on Figure 6.6. The most important modifications to the FrameMaker user interface include:

- *Merging the authoring and browsing modes.* FrameMaker has a default authoring mode and a "hypertext" mode. In the first, it is not possible to activate hyperlinks. In the latter, modification of the contents of a document is disabled. Users switch between them by typing an escape sequence<sup>1</sup>. This makes modification of an active document also being used to navigate in the design information web very cumbersome. In the Navigator, there is no such modal behavior. A normal mouse click has the effect a user would expect from an active object, while a mouse click modified by a control key pressed simultaneously will pop-up a dialog for configuring the behavior of the object. For instance, in a *note* containing a "button," a mouse click on the note would result in the invocation of the DocScript procedure associated with that event. On the other hand, a mouse click in simultaneous with the control key, would activate the control panel for changing the label and the DocScript procedure.
- *Controls for handling the activation of active documentation objects.* In FrameMaker and other documentation tools in general, active objects are restricted to hypertext buttons that have to be explicitly fired by a user mouse command. In the Navigator, a frame can also be active. When it is mapped into the screen, a command can be invoked

---

1. <esc> F1 k

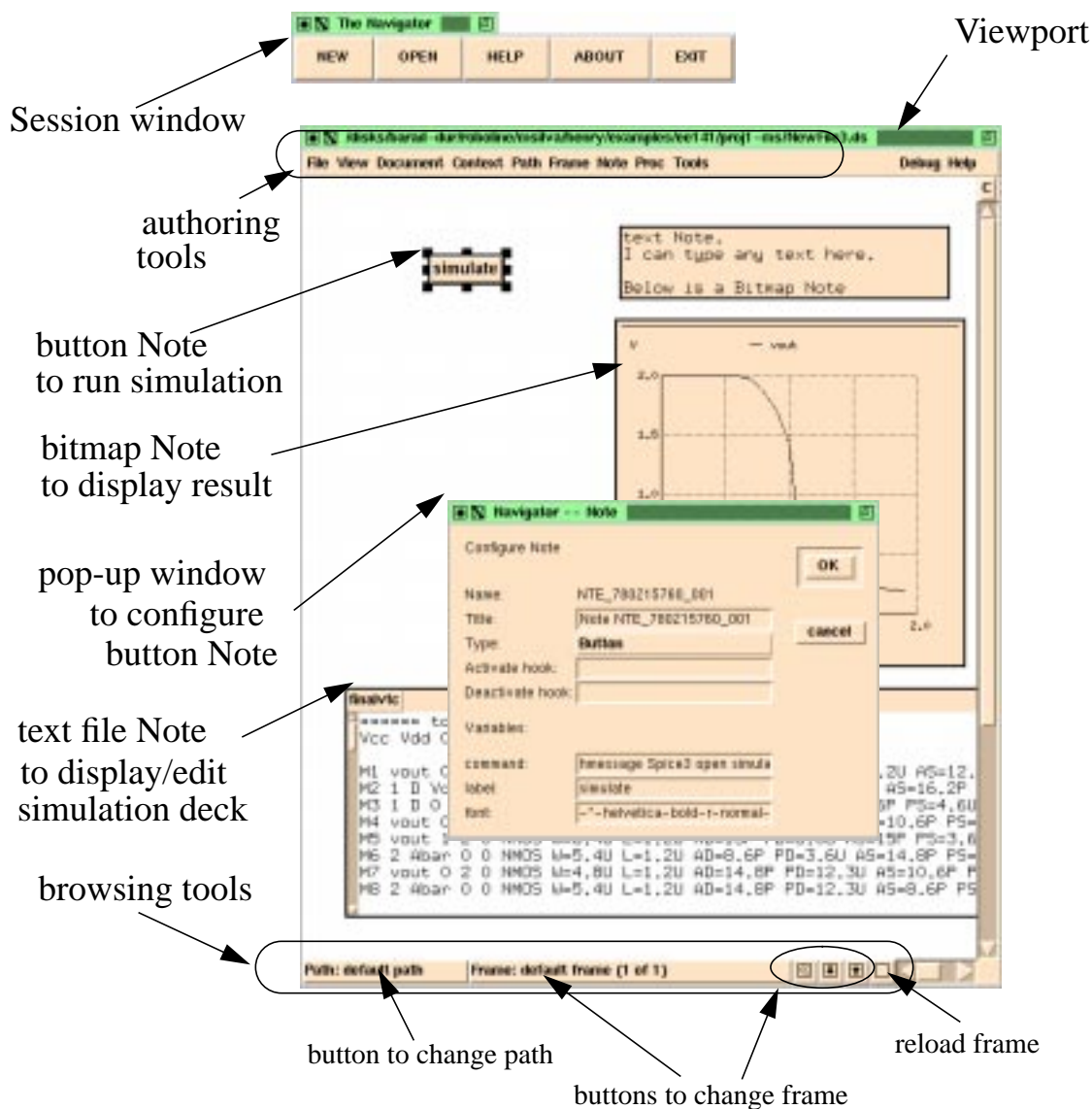


Figure 6.6 The Elements of the User Interface of the Navigator

The user interface of the Navigator was inspired by FrameMaker's user interface. However, we have incorporated extensions to facilitate simultaneous authoring and browsing of active documents. In the figure, we label some of the elements of the Navigator's user interface. The layout chosen for the various controls clearly separates the browsing functions (at the bottom of viewport windows) from authoring functions (at the top). However, both are easily available. For instance, a simple mouse-click on a *note* containing an active object (such as a button) has the effect of activating the object as expected by a document reader. However, a mouse button press in simultaneous with the control key on the same object immediately pops-up a window with a dialog presenting the options for modifying the object.



for instance to update the contents of the information displayed or to send a synchronization command to another tool or database. The buttons we added enable user re-activation of active objects, such as frames, which do not have a button-like behavior<sup>1</sup>.

- *Multiple viewports for editing a document.* In the Navigator, a *viewport* is the main window that displays a frame in a document. By having multiple viewports, we can change the contents of two frames concurrently. This is useful, for example, when simultaneously exploring two possible design alternatives in order to include the result of both analysis in the same document. On the other hand, FrameMaker supports two or more open windows into the same document, but only one can be used to modify the contents of the document.

#### 6.4.1 The Navigator Versus Other Documentation Tools

We summarize the differences between the Navigator and two other documentation processing tools that have been integrated in the Henry System, NCSA Mosaic and FrameMaker, in Table 6.3.

The Navigator, being a prototype tool, retains several limitations that preclude its use in an *industrial-strength* VLSI project. There is no support for printing DocScript documents, association of hypertext commands to words in text notes, display of color images, or running of embedded applications. These limitations result from the lack of certain features in the current version of the Tk toolkit (3.6). These shortfalls are expected to be remedied in the next release of the toolkit.

The Navigator's main limitation, its poor text processing capabilities, would require a complete redesign of the tool, but one that would add little value to the project. This involves extending DocScript to support structured documents and the capability to import and export documents in the SGML format. However, many document processing

---

1. This is similar to the "Reload" button available in Mosaic to enable users to refresh the contents of an URL.

	<b>FrameMaker</b>	<b>Mosaic</b>	<b>Navigator</b>
Text Processing	excellent	poor	poor
Remote Control	hard to setup	limited	very good
Access to External Data	poor	excellent	poor
Customizability	good	poor	very good
Simultaneous Authoring + Browsing	poor UI	none	good

Table 6.3 Feature Comparison Between FrameMaker, Mosaic and the Navigator

The table compares key features in the three main documentation tools used in the Henry System. Framemaker is very good text processing. The Navigator is controlled from remote tools very easily and is very easily customizable (this is inherent from being built using the Tcl language and the Tk toolkit). Mosaic is excellent for pulling-in documentation from external resources. The table shows that the tools, in their current stage, are mostly complementary. The integration of these three tools into the Henry system, makes it superior in terms of the overall capability for handling design documentation.

systems already exist, with their vendors committed to support SGML in the near future. We believe that adapting one of these documentation systems to operate as Henry's notebook would be a more feasible option than redesigning the Navigator.

## 6.5 Summary and Conclusions

An electronic notebook is a key component of an integrated design and documentation environment. Its main feature is the capability to make designers perceive the design as organized in an information web. An electronic notebook provides uniform access methods to design tools, component libraries and design process management services.

SGML, despite its limitations for describing active documents having sophisticated embedded dialogs, is quickly becoming the de facto standard for on-line documentation representation. Given the importance of making documents easily portable, one should

consider sacrificing the flexibility of using a user interface language such as Tcl/Tk as the basis for representation of active documents in favor of SGML.

We have no knowledge of a single documentation tool combining all the needed features for browsing a design information web. The Henry system provides integration at the inter-tool communication level between three documentation tools that complement each other: the Navigator, Mosaic and FrameMaker. This combination provides most of the necessary features. However, to properly support documentation in a wide-area multi-organizational design environment, these should be better integrated. In addition, Mosaic and the Navigator have documentation processing limitations that makes difficult their use in an industrial-strength VLSI design.

None of the documentation tools we integrated supports concurrent editing of a document. The lack of groupware tools integrated with Henry, strongly limits the adoption by large teams of a methodology based in simultaneous design and documentation. However, we believe that it would not be difficult to integrate these tools with the Navigator. The study of integrated design and documentation aided by groupware tools is future research.

“Plan to throw one away; you will, anyhow.”

— Frederick P. Brooks, Jr.

## **Chapter 7**

# **The Henry System Implementation**

In this chapter, we present the implementation strategy and some of the details of the tools and support infrastructure that compose the Henry prototype. We also describe scenarios that we used to test and evaluate the operation of Henry.

### **7.1 Introduction**

Our implementation goal was to quickly prototype an environment to help us develop the architecture and tools described in the two previous chapters. We tried to reuse existing software as much as possible. In many situations, the results are far from optimal. Many implementation details must be understood to install and use the system that would be hidden in a more robust release. On the other hand, this approach had the advantage of providing us with an early test vehicle to evaluate the feasibility of many of our ideas with reduced implementation effort.

The existing prototype of the Henry System consists of a set of design and documentation tools, that communicate with a common message HUB. This is used for passing messages

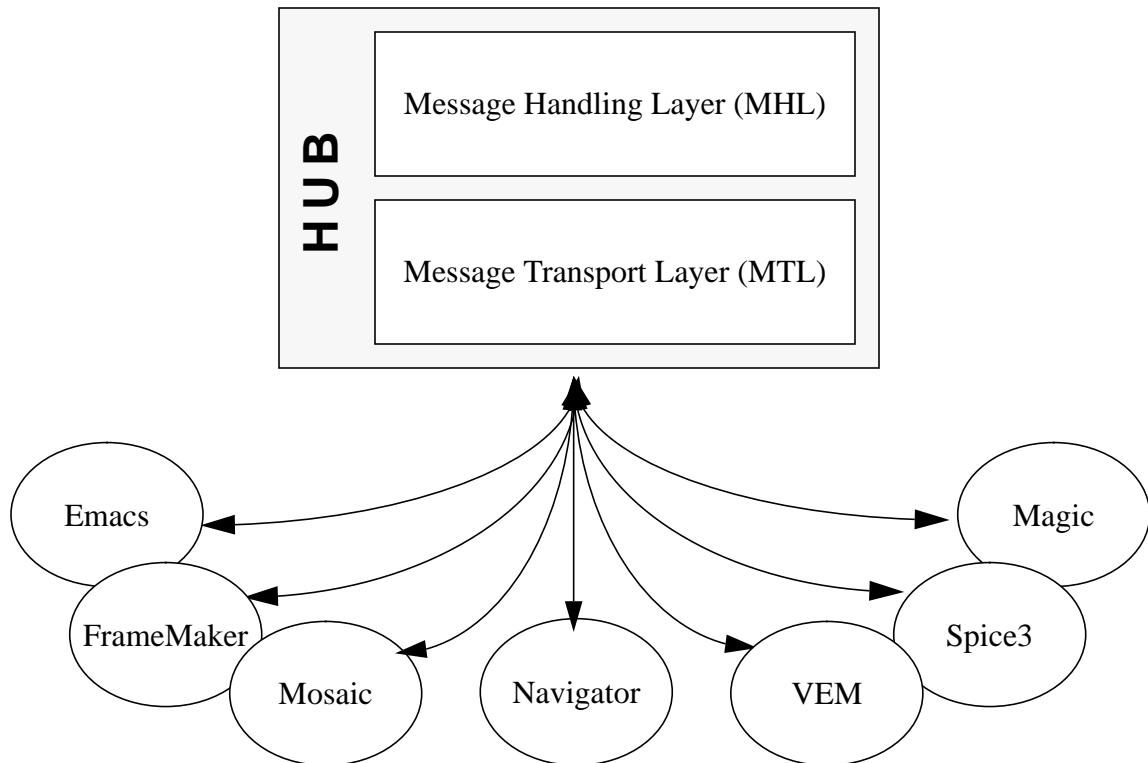


Figure 7.1 The Components of the Henry Prototype

The Henry prototype is composed of the message HUB and a set of design and documentation tools. The tools have been adapted to communicate with the HUB. The HUB supports all the interfaces used to communicate with the tools in its Message Transport Layer. The actual dispatching of messages between the tools is performed at the HUB's Message Handling Layer.

between the tools. Some of the tools had to be modified to communicate with the HUBs. These incorporate a collection of inter-tool communication interfaces we designed for sending and receiving commands (see Figure 7.1).

This chapter is organized as follows. We start by presenting our implementation of two operating system services required by the HUBs, but which are not available in the UNIX systems used to develop Henry. In Section 7.3 we describe the implementation of the interfaces between the HUBs and the tools. In Section 7.4, we discuss the implementation of the Navigator, Henry's electronic design notebook. This is followed by the presentation of the implementation of the interface to the World Wide Web in Section 7.5. Section 7.6 presents design scenarios involving access to resources available on the Internet using

Henry. In Section 7.7, we review the evolution of the Henry System since its inception. Finally, in Section 7.8, we summarize the chapter and present our ideas for implementation of a next version.

## 7.2 System Services Implemented by the HUBs

For the implementation of the HUBs we had to develop two system services, not offered by the current versions of the operating systems available in most UNIX workstations. These are the *session manager* and the *services registry*.

The session manager is invoked to assert that a tool has been started. First, it determines if the user is already running the tool in one of the machines of the network. If not, it starts it on the host selected by the user, using the UNIX *rsh* command, and waits for it to respond. If the tool is accepting input from remote clients, then the session manager simply returns.

The services registry of Henry is intended to keep track of the host and port locations of the applications a user has running on the network. In our environment, design tools dynamically assign an unused port for receiving commands from clients. Once they have the port assigned, they inform the registry of the host and port where they can accept commands. This registry is used by the Henry session manager for rendez-vous with the tools.

We describe the implementation of these two services in more detail in the remaining of this section.

### 7.2.1 Session Management

The session manager is invoked when the HUB receives a message addressed to a tool. To deliver the message, the session manager attempts to establish a connection with the Internet port in the network assigned to the tool in the services registry. However, as the tool's execution may have been stopped or it may not have been started, the connection may not be possible. At this point, the session manager restarts the tool and waits until the new listening socket is registered. The tool then becomes ready to receive the message.

In our implementation, the session manager is not an independent module, but a set of procedures. For each tool there is a module that groups the procedures that handle communications with the tool. In each module, there is a *start* procedure, which is invoked at the beginning of all other procedures used for communication in the modules. The collection of these start procedures constitutes our implementation of the Henry session manager.

The procedures that form the session manager are built upon a collection of procedures for starting interactive applications anywhere on the network. These procedures in turn invoke the commands of *expect*, an extension to Tcl for controlling interactive sub-processes [Libe95].

### 7.2.2 Services Registry

The session manager needs to locate the listening address of running programs. In the Internet, well-known sockets are only defined for a small number of general services. Some applications register in pre-defined TCP/IP ports, but this is bad programming practice, as it may create conflicts between programs using the same pre-defined port. Most applications listen for requests in a dynamically assigned port. After obtaining such a port, they make the port known to the network using one of many available naming mechanisms. Typical examples include a file containing the port number in a common file system or a pre-established property in the X Window System server [Sche92, oS90] used by the application. To integrate a new tool in the Henry System, there are two approaches. In the first, we modify the tool to use Henry's own registry directly. In the second, we find what registry the tool uses, start it and wait for it to register. Then we transfer its listening port location information to the Henry registry.

We chose not to implement the Henry registry as another separate process or as part of the HUB. Instead, we decided to use the property tables available in X servers to maintain a database of the ports and hosts where tools are accepting connections. This works well in our environment, where all interactive tools need a X server to run. It also makes the system more flexible, as design and documentation tools can be started independently of the

HUB. In addition, it also makes the system more robust. If the HUB crashes at some point during a design session, a new HUB process can be restarted without affecting the operation of the tools. This is possible because the HUB uses exclusively stateless protocols and can find the listening ports of the running tools by once again consulting the registry database in the X server.

The HUBs also register with the associated user's X server, as would any other tool. This enables tools to find the port where the HUB listens when they have to transmit a message as a response to a user command. However, some applications and external HUBs without access to a user's display may also need to find its listening port. For instance, consider the situation when active messages have been sent to the design environment of another user. These are sent to his electronic mail address or to an URL that his associated with his name. In both cases, messages have to be delivered to the recipient's HUB from a program that operates as a gateway and has no access to the X registry. For this reason, we also make HUBs advertise their host and listening port in a pre-defined location in the file system.

To make tool integration easier, we created *servrg*, a library that provides functions to help applications register the address of their listening sockets in the Henry common registry. It also provides functions for clients, such as the HUB session manager, to find the host and port where they are listening to their requests. The library offers three language interfaces, for C/C++ programs, and Tcl and UNIX shell scripts. This gives to system integrators several alternatives for making tools use Henry communication services.

### **7.3 Interfaces with Design Tools and Frameworks**

The Henry System already contains a diverse collection of commonly used design and documentation tools that we have integrated. These include:

- FrameMaker, a documentation processing system with hypertext support, developed by Frame Technology, Inc.



- Magic, a VLSI layout editor [Oust85]. Magic can also operate as a front-end to IRSIM, a logic simulator.
- SPICE3, a circuit simulator which is linked to *nutmeg*, a front-end for waveform displaying.
- GNU Emacs, an extensible text editor based on the LISP language. Hypertext extensions have been developed for Emacs. GNU Emacs runs also as a front-end to a very sophisticated software development environment.
- VEM, the front-end to the Octtools VLSI Design Framework [Harr86].
- The tools developed at the NCSA to interface with the World Wide Web, namely Mosaic [Andr93] and the *httpd* server.
- The new tools we wrote to support integrated design and documentation, such as the Navigator, presented in the previous chapter.

This list gives a good snapshot of the different types of interactions performed by current system designers. It includes tools used for information retrieval, software development, integrated circuit layout and simulation, and documentation. We believe that other tools addressing design aspects not covered by those we integrated, such as logic synthesis and printed circuit board design, use fundamentally the same types of interactions and could be integrated in similar ways.

All the tools integrated in the Henry System have bidirectional communication capabilities with the message HUB. The HUB is written as a shell that accepts commands in the Tcl language. It runs a script distributed across several files in Tcl that perform the dispatching of messages coming in from its communication interfaces. The HUB also accepts interactive commands but this capability was added for enabling easy interactive debugging. When it starts, the HUB places itself running in the background, with its window iconified.

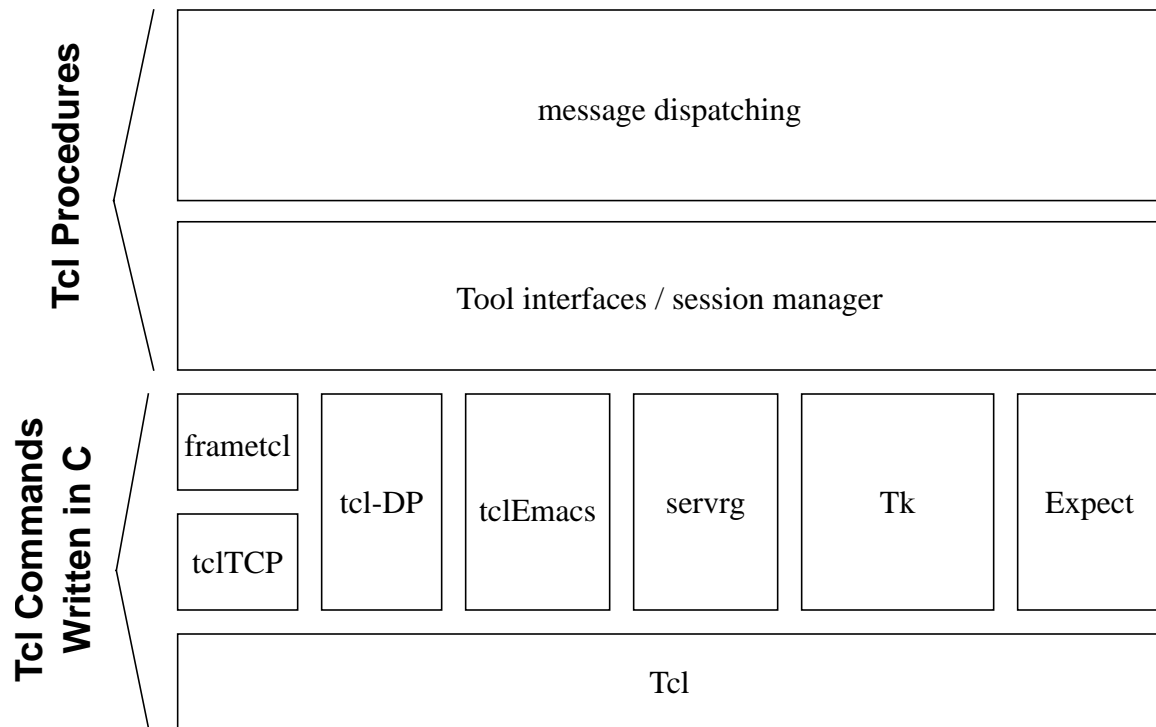


Figure 7.2 Extensions Loaded by the HUB Tcl Interpreter

The HUB is a C program running a Tcl interpreter and several extensions. Some were developed by us, while the others were developed externally and put in the public domain. The extensions add new commands to the basic Tcl language appropriate to communicate with the tools used to produce active documents. The tool interfaces and the session management services are implemented as Tcl procedures that call the commands added by the extensions. On top of the tool interfaces, we have the message dispatching routines, also completely written in Tcl.

The organization of the HUB's software is shown on Figure 7.2. We incorporated a collection of extensions into the basic Tcl shell. These are written as C libraries that add new Tcl commands to the language. The extensions offer primitives for communicating with the tools in the form of additional Tcl commands. We organized these extensions as generic packages for inter-tool communication between any generic Tcl program and the tool to which the package interfaces. These include:

- Expect, for spawning tools by the session management services, as described in the previous section.

- Tcl-DP, a package to support distributed processing, developed at Berkeley [Smit93]. We use Tcl/Dp's TCP sockets manipulation primitives to communicate with Magic and Spice3.
- TclFrame and TclTCP, developed at GE Labs. TclTCP replicates the functionality available in Tcl/Dp for handling TCP/IP sockets. TclFrame uses TclTCP to interface with FrameMaker using its public Sun RPC-based interface [Fra]. We have ported this software to interface with the latest release of the FrameMaker system.
- tclEmacs. We have written this package to interface with the GNU Emacs editor. The package modifies *gnuserv*, an extension to Emacs for accepting commands from remote clients. These may invoke a set of shell commands that create the connections with the server running on the Emacs editor. We modified *gnuserv* to register the emacs server using our server registry conventions and modified the shell commands to be run as Tcl procedures implemented in the C language.

In the remainder of this section, we describe in more detail the implementation of the interfaces to the design tools integrated with the Henry System. Each of these interfaces is supported by a library of Tcl procedures. Although these are currently used only by the HUB, they are ready to be incorporated as an extension into any Tcl-based program.

### 7.3.1 Interface to Spice3

We modified the code of *nutmeg*, the simulation front-end distributed with Spice3, to integrate this tool with the Henry System. The front-end is controlled via commands in an interpretive language specific of this application. We changed it to listen to an unassigned Internet port and pass the commands received on that port to the local interpreter. Upon initialization, the modified *nutmeg* registers the host name and port number in the X server to which it is connected. For registration, it follows the conventions defined for the Henry System and implemented by the *servrg* library, presented in Section 7.2.2 on page 127.

It is also possible to send commands from nutmeg's main window to any Henry application. This enables a user to ask for the presentation of a layout or the description of a design being simulated. We wrote a macro in nutmeg's command language that executes in turn a tcl-DP script as a child process. The macro uses nutmeg's shell command, that provides a user interface to the UNIX *system()* call.

### 7.3.2 Interface to Magic

We also had to make changes to the source code of Magic to integrate it with the Henry system. For receiving commands, we modified the Magic *helper*. This is a subprocess started from Magic to receive events from the attached X display. The helper collects key inputs, writes them into a UNIX pipe to the parent and signals it when a command line is ready for processing. The changes to Magic's helper were similar to those we made to the Spice3 front-end. We modified it to listen in a new Internet port, register it according to the Henry's server registry conventions, and multiplex the commands received with those typed-in at the graphic user interface.

To send commands from Magic to other tools, we added a command to the parent process, similar to the shell command available in nutmeg. We also added a new command for retrieving the contents of a label and pass it for evaluation by the shell. This makes it possible to execute annotations to the design. Annotations could then contain for instance commands to display related documentation. Magic supports the addition of new operating modes for the mouse, called *tools*. With a new *hypertext* tool, it is possible to use a layout as a map for quickly guiding a designer to the documentation of specific detail of the layout.

### 7.3.3 Interface to VEM

VEM is the front-end to the Octtools environment. To enable communication between Henry and the Octtools, we created a new program that runs as an independent process and operates as a gateway between VEM and the HUB. This process is started from VEM

as one of its RPC<sup>1</sup> applications and is also linked with the Tcl/Tk libraries. This makes it possible to pass commands in both directions.

When a designer starts VEM from the Henry environment, it is called with the options to automatically start the gateway. This registers as the VEM editor in Henry's registry. It is then possible to invoke VEM commands through this gateway.

Communication from VEM to the HUB is also possible. When the gateway starts, it also calls VEM to register a new command, called *Exec-Tcl-Command*. When this command is typed-in at VEM's window associated with the gateway, its arguments are passed back for execution on the gateway. This interprets the arguments as a those of a *send HUB* Tcl command, which is evaluated in the local Tcl interpreter. As a result, the arguments of the Exec-Tcl command in VEM end-up being evaluated in the HUB's Tcl interpreter.

This integration approach has the main advantage of not requiring any changes to VEM's source code. However, it also has limitations. It forces the creation of an additional process and routing of messages through it. This would be acceptable if we could have a single gateway running all the time in the background. However, because VEM forces commands to be typed from windows and only allows one-to-one mappings between its RPC applications and its windows, designers are forced in practice to start and stop the gateway almost continuously. This happens whenever they have to run another RPC application on the *facet* connected to the gateway or need to send a command from a remote tool to a *facet* displayed in a different window.

## 7.4 Navigator Implementation

The Navigator is implemented as a C++ program with a Tcl/Tk based user interface. The fundamental and more complex data structures in the tool are described in C++. These include the data structures that describe open documents and the viewports created for

---

1. RPC — Remote Procedure Call.

interacting with them. There is a set of Tcl commands to call the C++ methods that manipulate the data structures. The user interfaces and the linking between the visual controls and the Tcl commands to manipulate the data structures are written in Tcl/Tk.

It is very simple to communicate between the Navigator and its associated HUB. We could have used Tk's send command, which is used to invoke a Tcl procedure running in a remote interpreter attached to the same X display. The procedure invocations and the results are transported on the X protocol through the common display server. However, we have chosen to use Tcl-DP, an extension to Tcl that offers the same capability over a TCP/IP connection. We did not decide to incorporate this extension for remote command invocation because of its faster response time. Latencies in both cases are in general smaller than a few seconds, hence acceptable by the users. However, sometimes networks have much larger response times. In that case, the behavior of Tk's send command is uncontrollable, generating many unwanted time-out exceptions. Tcl-DP provides a much better way to handle communications under such situations.

The Navigator uses DocScript, Henry's native language-based environment for manipulating the data structures representing active documents<sup>1</sup>. DocScript is implemented as a set of C++ classes, one for each basic documentation concept supported by the language. Each class constructor generates a new Tcl command which is then invoked to configure and retrieve the attributes of the class and call its methods. The DocScript library contains about 13,000 lines of code.

Custom user interface dialogs can be introduced in any document by defining new DocScript *note* types as a set of Tcl procedures. There is also a set of simple pre-defined note types that can be instantiated in any document. Currently, these include a button, a bitmap viewer, a text entry, labels and a text file editor. Adding a new note type is relatively simple for those who now how to program in Tcl. As an example, the Navigator's file contain-

---

1. . We described DocScript in Section 6.3 on page 111

ing the Tcl procedures that describe a button note, contain approximately 50 lines of code. This process could also be automated, by adding a user interface builder to the system. This is an interactive tool for creating the user interface of a dialog from a palette of available widgets. XF is an example Tcl/Tk user interface generator that could be adapted for this purpose [Delm93].

The advantage of having active documents described in a programming language is not limited to the gained flexibility. This feature had a dramatic impact on the speed of development and debugging of the Navigator. We could write the initial active documents with a text editor. Later on, the same tool could be used to detect and fix errors in active document's descriptions.

In addition to the document data structures, the C++ part of the Navigator was also used to describe the data structures for the run-time environment. These describe the sessions open with each document and the viewport windows used to manipulate their contents. In the Navigator, a session is a class containing pointers to the DocScript data structures that describe the documents being edited and the objects with visual representations used to manipulate them. The C++ part of the Navigator contains approximately 7,000 lines of code. The Tcl/Tk part, containing the user interface has about 6,600 lines of code.

The run-time performance of the Navigator is very acceptable, considering it was designed as a throw-away prototype without any concern for performance. DocScript documents have on average 5,000 kilobyte per frame and can be parsed on a Sun SparcStation2 workstation in less then one second per *frame*. The rendering time depends considerably on the complexity of the *frames*, which depends on the number and type of *notes* contained. Typical times are in the range of 2-5 seconds/*frame*.

## 7.5 Interface with the WWW

As described in Chapter 5, the Henry System system is designed to exchange active messages. These are represented using an extension to MIME. In the current implementation,

HUB processes do not support MIME directly. The Message Handling Layer is simply a procedure that dispatches messages using the *hmessage* interface with the Message Transport Layer.

As a result, in the current prototype, messages sent between the tools that run locally in a user's environment are not converted into MIME. Exchange of MIME formatted messages only takes place in communications with other users and external servers. Decoding of MIME messages is made by a separate program, based on the publicly available MIME support software. This translates the *multipart/enabled-mail* MIME format used for active messages into sequences of *hmessage* commands to the HUB and a set of data files, whose names are passed as arguments to the *hmessage* commands.

Figure 7.3 shows the software modules used to process messages when communication with external HUBs and Internet-based services is involved. There are two interfaces for exchanging information with tools running in external user environments. One uses SMTP, the Internet protocol for electronic mail [Post82]. The other uses HTTP, the client-server communications oriented protocol of the World Wide Web (described in Section 3.4 on page 50). The rest of this section presents these two interfaces in more detail.

### 7.5.1 Active Messages Transported by Electronic Mail

To send an active message by electronic mail from a tool, a user of the system clicks on a button or selects highlighted text in one of the tools. This has the effect of sending an *hmessage* to the HUB. As the HUB runs a Tcl interpreter, it is straightforward to send a file to another user. The following *hmessage* accomplishes it:

```
hmessage HUB [exec mail user@host < file]
```

In Henry, we also have *mailmessage*, a script that we wrote to generate active messages from *hmessages* and send them by electronic mail. *Mailmessage* generates a *multipart/*



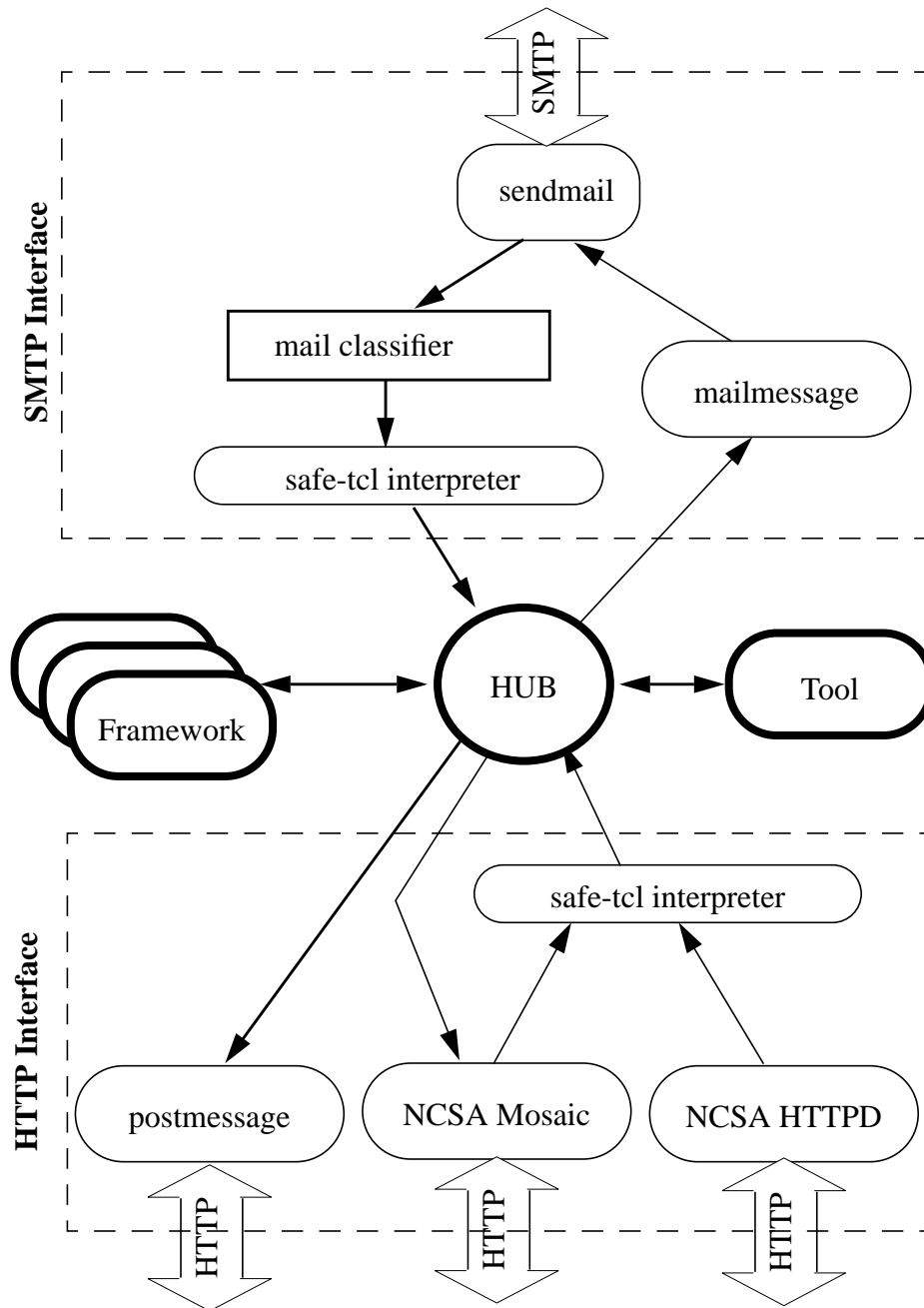


Figure 7.3 Implementation of Henry's Interface to the WWW.

The current version of the HUB message handling system does not manipulate MIME formatted messages directly. It supports only the *hmessage* format, developed for sending commands to design and documentation tools. However, the Henry System can process active messages. It uses external software modules to translate active messages into files and sequences of *hmessage* commands that are then sent to the HUB. Postmessage and mailmessage are two scripts that can translate an *hmessage* command into an active message containing the command and the files it references.

*enabled-mail* MIME message<sup>1</sup> with the commands and files indicated as arguments, and pipes it to *sendmail*, the UNIX program to send mail over the Internet.

To deliver active messages received by electronic mail in the Henry environment, a user needs to configure his mail agent program to automatically activate *multipart/enabled-mail* type messages. We achieve this via a mail classifying program<sup>2</sup>. The user's environment is modified to pipe automatically all messages to the classifier upon delivery (in the *~/.forward* sendmail configuration file). The classifier is configured to store non-active messages in the user's mailbox and folders. Active messages, however, are piped directly into *swish*, the Safe-Tcl interpreter of the Enabled-Mail software distribution. Henry's active messages contain directives to load our extensions. These consist of Tcl procedures to convert active messages into messages in the *hmessage* format and dispatch them to the HUB of the receiver. HUBs are assumed to be running while the associated users are in session. If a HUB to which a message has to be relayed is not running at the time of delivery, the mail classifier simply places the message into a special folder. The message can then be read and possibly re-activated at a later time.

### 7.5.2 Active Messages Transported by the WWW Protocol

The sequence of operations for sending an active message using the WWW interface is similar to the one used to send it via electronic mail. The only difference is that we use a different *postmessage* script. While *mailmessage* takes a user's electronic mail address as argument, *postmessage* takes an URL. This runs on a Tcl interpreter with the *expect* extensions. *Postmessage* spawns a sub-process running the *telnet* program which in turn connects to the WWW server of the URL and sends the generated active message using the HTTP *POST* command.

---

1. *multipart/enabled-mail* is the MIME extension we adopted for encoding active messages, as described in Section 3.3 on page 46.

2. There are several programs available for this purpose. An example is *slocal*, which is part of the MH mail handling system [Rose85]

Users may receive active messages from the HTTP interface in three ways:

- As a reply to posting in a URL using the *postmessage* script. HTTP servers in general return a HTML document with information about the result of the execution of the commands they receive.
- As a reply to retrieving the contents of a URL when using Mosaic to browse the WWW. The *hmessage*

*hmessage Mosaic open URL*

starts Mosaic and sends it a command to GET the URL indicated.

- Through the HTTP server running in their environment. In this case, we adapted the same Enabled-Mail support software used to dispatch an active message received by electronic mail to a HUBs to interface with NCSA's *httpd* server. We use its Common Gateway Interface (CGI) [McCo94] to activate *swish* and pass it the contents of the received active message.

## 7.6 Exploring Internet-based Design Scenarios

We now describe how electronic commerce services could be accessed and provided using our software. What follows is not a report of our experience using Henry in actual designs, but the description of sequences of design operations that can actually be performed with the existing prototype. The design of the Henry architecture proceeded through the development of these mock-up scenarios. We used them to validate the architecture and inter-operation between the various components of the environment.

For the first scenario, we consider a SPICE simulation service that is accessed over the Internet. As a second scenario, we consider a designer selecting and ordering an off-the-shelf chip, its documentation, models and application notes from a catalog on the WWW.

It is not simple to create usable electronic commerce services for designers. A number of challenges must be addressed, including:

- Authentication of clients and security of communications. For these, we can use cryptography techniques. Certification authorities [Chok94], digital signatures [oS91b] and Privacy Enhanced Mail [Linn93, Kent92] are developed technologies that could be used for this purpose [Brow94]. Certification authorities are network services that can prove the authenticity of a user or an electronic document produced by the user. These are the electronic equivalent of today's notaries. Digital signatures are special checksums that are appended to electronic documents to certify that their contents have not been tampered and could not have been produced by anyone else but their authors. Privacy Enhanced Mail is an Internet standard describing algorithms and conventions for encrypting electronic mail messages.
- Billing. For this we could use existing software for automatic placement of orders and payment, based on EDI, the Electronic Data Interchange standard [oS91a].
- Intellectual property protection. This is of major importance both to service providers and users. They need to have guarantees that their simulation models will not be given to someone else. We believe that a combination of legal and technical mechanisms could be created to provide the necessary protection, such as automatically exchanged and electronically authenticated non-disclosure agreements or binary representations of simulation models.

Our goal in developing scenarios of how designers could use of electronic commerce focuses on finding the appropriate flow of information and sequencing of tool invocations required to implement the services. In the development of the scenarios presented here, we assume that the above problems could be addressed by re-using existing software. In the rest of this section, when we mention billing, authentication or encryption, we refer to the point in the flow simulated by the scenario where these operations would be performed. In the mock-ups we built using the tools integrated in Henry, these are not actually executed.

### 7.6.1 The Remote Circuit Simulation Service

A remote simulation service could be very useful to small design organizations that cannot afford to purchase the expensive hardware accelerators and software for prototyping complex systems. In our mock-up, we used *Spice3* as the simulator and *nutmeg*, its front-end for viewing simulation waveforms. Other simulation tools could easily be included too.

Network-based simulation services have been implemented before. These were accessed either made available by electronic mail, a simple client-server protocol or the creation of an account at the service provider. However, these services' availability and usage accountability were limited. We describe below a protocol for making this service billable and available to anyone on the Internet. We also assume that the service is used for long-running expensive simulations. As a result, its users need to have feedback about the simulation status after submission of the job and the capability to abort it on demand.

In our scenario, the remote circuit simulation service is organized as follows (see also Chapter 7.4). There is a WWW home page that advertises the service. From there, it is possible to retrieve the terms and conditions for its use. When a designer decides to use the service, he or she can fill-in the contractual forms interactively. Once completed and authenticated, the designer receives a document with a digitally signed contract. The document includes a URL that can be used in the future to request simulations. Simulation requests received at this URL are authenticated and billed to the client when completed.

To send a simulation request to the service, the designer runs either the *postmessage* script to send the simulation deck to the assigned URL in the service's HTTP server or mails the deck using the *mailmessage* script. In both cases, the request is encrypted with the simulation service's public key and authenticated with the designer's digital signature. In response, the designer receives another document with the acceptance of the request.

For designers without the Henry software, a variant of the *postmessage* script is available from the simulation server as an active message. When activated, the message prompts the

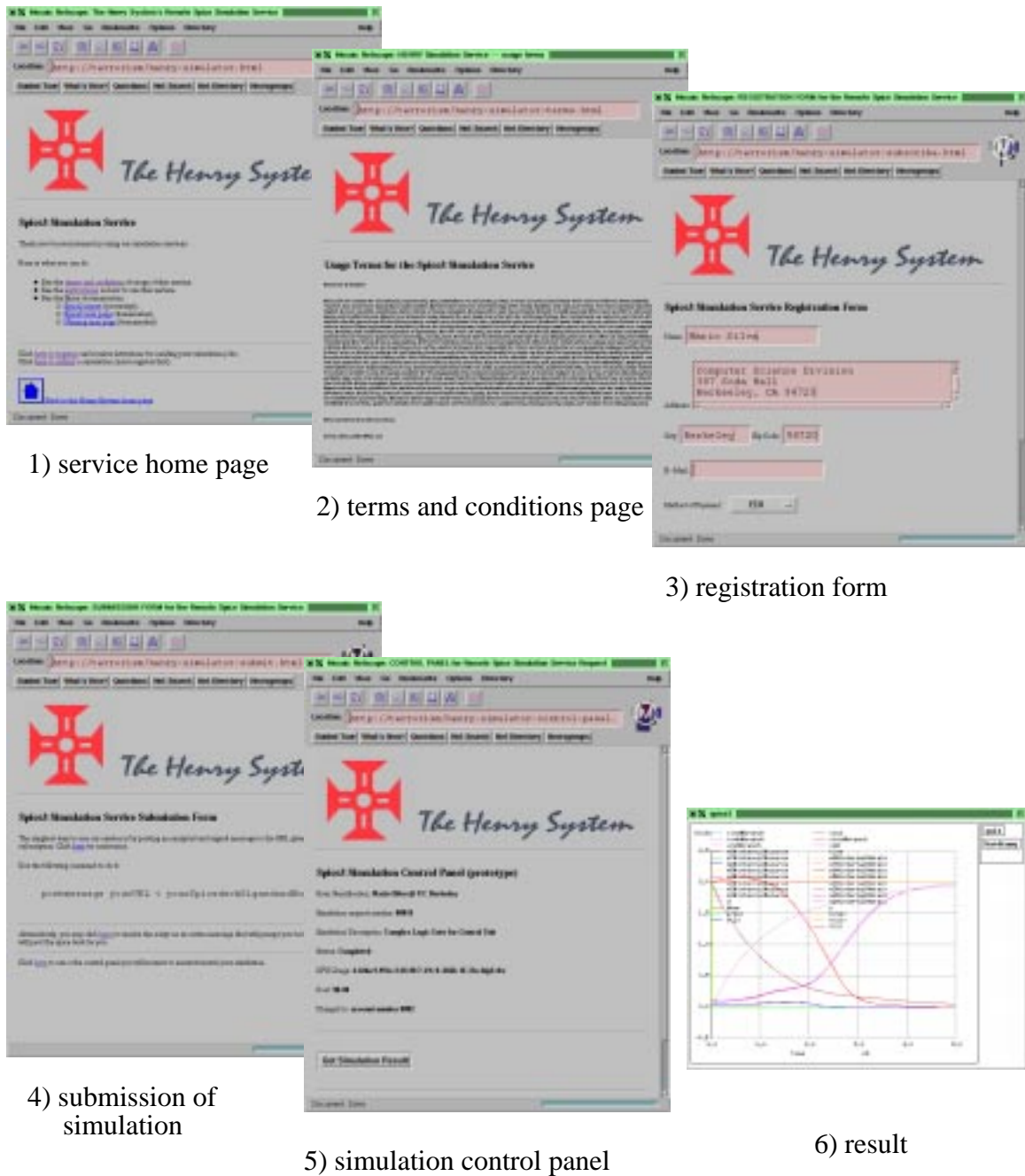


Figure 7.4 Flow Diagram of the Simulation Server

The figure shows the sequence of steps performed by the first time user of the prototype simulation service. There is an initial registration process (steps 1-3). Once registered, designers use a simple script to submit a simulation deck (step 4). They can then connect to the service to monitor and control the execution of the simulation and retrieve the result (steps 5 and 6).

designer for the location of the simulation deck and automatically submits it, encrypted and signed.

The service acceptance document contains an anchor to a new URL that can be used by the designer to observe the simulation status from Mosaic. *Postmessage* waits for this receipt, saves it into the designer's archives and then signals Mosaic to open it through its Remote Control interface.

The simulation service runs the Henry System tools. A simulation request is received by a script that is invoked by *httpd*. Once the simulation deck is decrypted and authenticated, an asynchronous request is sent to the HUB to start the simulation. This request is actually a script with *hmessages* to Spice3 to load the deck, run the simulation, plot the results and finally send an *hmessage* back to the HUB to execute a program to notify the client about the end of the simulation.

In the submission form, and also while checking the status of a simulation, clients can set options about how the simulation result is to be delivered. This is indicated by specifying a URL of the receiver. The URL may refer to an electronic mail address or an HTTP server where the simulation result may be posted. Users may also opt to receive only the indication of the URL in the service provider that may be used to retrieve the simulation result once it is finished.

When the simulation is complete, the form with the simulation status also includes a button which, when activated, retrieves the simulation result. In our prototype environment, results are formatted as MIME messages containing Spice3 *raw* files<sup>1</sup>. Mosaic is configured to automatically invoke *nutmeg* to display the contents of MIME messages of this data type.

---

1. Spice3 raw files contain the waveforms produced in a simulation in binary form.

The implementation effort to prototype this scenario using the software of the Henry System was rather small, around two weeks. The main limitation was the lack of a tool to compose and send a simulation request. The current version of Mosaic has no support for posting large files to HTTP servers. The form-based user interface of Mosaic when user input request is required is also somewhat limitative. In our view, this is another argument for organizing design systems as an ensemble of tools capable of accessing the WWW instead of having one single tool that centralizes all the data presentation and communications with Internet services.

### 7.6.2 The On-line Component Selection and Ordering System

Our goal for this scenario was to develop a mechanism for selling complex VLSI components on the Internet. Information would be presented in a similar way to that used by the MSU Microsystems Prototyping Lab library project<sup>1</sup>. However, we made different assumptions about how this information would be available. Access to part of the information would be restricted and given for a fee. A business transaction would be performed automatically using electronic commerce. In addition, instead of providing bitmaps of the layouts in GIF format and delay information as tables, we wanted to be able to send the layouts in a CAD interchange format and simulation models along with propagation times tables. In addition, we wanted to have the files automatically installed in the clients databases via active messages.

To order a component, a designer first consults a manufacturer's database with their specifications and application notes illustrating their use. Once connected to the database, he or she receives a document with a catalog of the available information. From the catalog, he or she can retrieve a *preview*, containing publicly available information about the component, such as its basic characteristics, cost and usage terms. Next, if the designer decides to

---

1. The Mississippi State University Microsystems Prototyping Lab Standard Cells Library is located at  
URL <http://www.erc.msstate.edu/mpl/libraries/stdcells/>




order it, he fills-in an electronic form containing the company's identification, type of framework where the component models and schematics will be installed, interchange formats accepted, address and payment method. In return, the designer receives a HTML document. This contains the transaction receipt and information on how to retrieve the information.

Clients can retrieve the information in several forms. The simplest way is by activating the hyperlinks to the URLs in the library server pointing directly to the simulation models and schematic symbols for the purchased component. With minimal extensions to the MIME configuration files on both sides, we can have the appropriate tools invoked to display design files directly from Mosaic. However, as in this method we retrieve the files one at a time, activation of links between the files that constitute the component's information package is not possible. This is because the links use relative addressing to refer to other files.

Full browsing capability only becomes possible when clients have the Henry tools installed. These may download all the component's information in a single active message. This contains the complete set of files for that component plus a script to install them in a directory structure reflecting that of the server. Links between the files in the package can then be directly activated.

In our implementation of this scenario, the component library runs the NCSA *httpd* WWW server. Clients access it using Mosaic and retrieve the design information as active messages. Figure 7.5 shows the windows seen by the user when retrieving the component information and Figure 7.6 shows a diagram with the information flow. The component catalog and order forms are written in HTML. The library is simulated with directories containing different implementations of various class projects in CMOS technology. The designs and associated documents were produced by the students of a VLSI design course who used an initial prototype of the Henry System. Each directory contains files in various formats, including FrameMaker and Navigator's documents, Spice3 simulation decks and



The screenshot displays the installation of the Henry system. It starts with a terminal window showing the command `ls -l /usr/bin/henry` and its output, which lists the file `henry` with permissions `-rwxr-xr-x` and size `10736`. Below this, another terminal window shows the command `cat /usr/bin/henry` and its output, which is the content of the `henry` script. The script is a Perl script that sets up the Henry system, including creating a directory `/usr/bin/henry` and installing the `henry` script into it.

```

ls -l /usr/bin/henry
-rwxr-xr-x 1 root root 10736 Jan 10 10:10 /usr/bin/henry

cat /usr/bin/henry
#!/usr/bin/perl

# Henry system installation script
# This script will install the Henry system on your machine.
# It will create a directory /usr/bin/henry and install the
# henry script into it.

# Create the directory
mkdir -p /usr/bin/henry

# Install the script
cp henry /usr/bin/henry/

# Set permissions
chmod 755 /usr/bin/henry/henry

```

The screenshot displays a MATLAB/Simulink workspace with several key components:

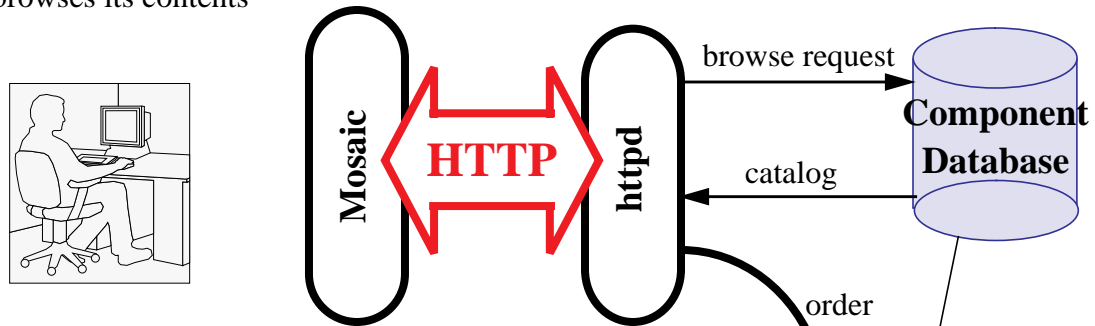
- Top Panel:** A MATLAB command window showing the execution of the `nn` function. The output indicates the network is a feedforward neural network with 10 hidden units and 1 output unit, trained using the Levenberg-Marquardt algorithm. The training process is summarized in the following table:

Table 1 (Training Results)						
Step	Mean	C	D	std	std	std
0-1	0	0	1-0	1.23e-01		
0-2	0	0	0-1		2.77e-01	
0-3	0	0				
0-4	0	0				
0-5	0	0				
0-6	0	0				
0-7	0	0				
0-8	0	0				

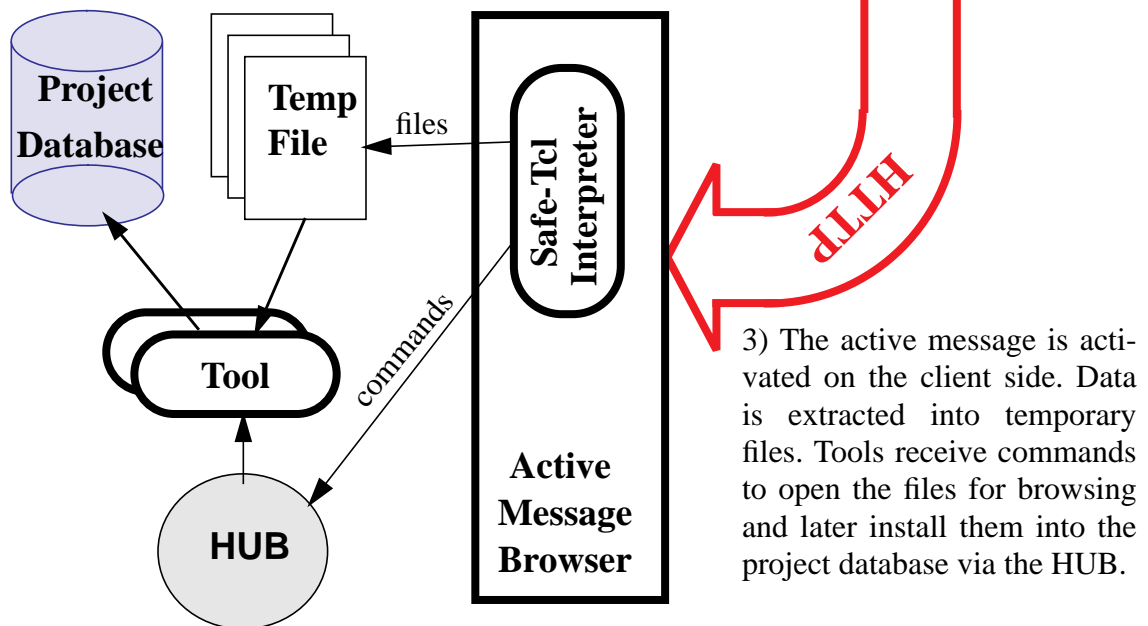
- Left Panel:** A plot titled "nn" showing the training error (mean) versus iteration. The error starts at approximately 1.2 and decreases rapidly, stabilizing around 0.2 after 10 iterations.
- Right Panel:** A plot titled "nn" showing the training error (std) versus iteration. The error starts at approximately 1.2 and decreases rapidly, stabilizing around 0.2 after 10 iterations.
- Bottom Panel:** A plot titled "nn" showing the training error (std) versus iteration. The error starts at approximately 1.2 and decreases rapidly, stabilizing around 0.2 after 10 iterations.
- Workspace:** A list of variables in the workspace, including `nn`, `nnnet`, `nnnet1`, `nnnet2`, `nnnet3`, `nnnet4`, `nnnet5`, `nnnet6`, `nnnet7`, `nnnet8`, `nnnet9`, `nnnet10`, `nnnet11`, `nnnet12`, `nnnet13`, `nnnet14`, `nnnet15`, `nnnet16`, `nnnet17`, `nnnet18`, `nnnet19`, `nnnet20`, `nnnet21`, `nnnet22`, `nnnet23`, `nnnet24`, `nnnet25`, `nnnet26`, `nnnet27`, `nnnet28`, `nnnet29`, `nnnet30`, `nnnet31`, `nnnet32`, `nnnet33`, `nnnet34`, `nnnet35`, `nnnet36`, `nnnet37`, `nnnet38`, `nnnet39`, `nnnet40`, `nnnet41`, `nnnet42`, `nnnet43`, `nnnet44`, `nnnet45`, `nnnet46`, `nnnet47`, `nnnet48`, `nnnet49`, `nnnet50`, `nnnet51`, `nnnet52`, `nnnet53`, `nnnet54`, `nnnet55`, `nnnet56`, `nnnet57`, `nnnet58`, `nnnet59`, `nnnet60`, `nnnet61`, `nnnet62`, `nnnet63`, `nnnet64`, `nnnet65`, `nnnet66`, `nnnet67`, `nnnet68`, `nnnet69`, `nnnet70`, `nnnet71`, `nnnet72`, `nnnet73`, `nnnet74`, `nnnet75`, `nnnet76`, `nnnet77`, `nnnet78`, `nnnet79`, `nnnet80`, `nnnet81`, `nnnet82`, `nnnet83`, `nnnet84`, `nnnet85`, `nnnet86`, `nnnet87`, `nnnet88`, `nnnet89`, `nnnet90`, `nnnet91`, `nnnet92`, `nnnet93`, `nnnet94`, `nnnet95`, `nnnet96`, `nnnet97`, `nnnet98`, `nnnet99`, `nnnet100`, `nnnet101`, `nnnet102`, `nnnet103`, `nnnet104`, `nnnet105`, `nnnet106`, `nnnet107`, `nnnet108`, `nnnet109`, `nnnet110`, `nnnet111`, `nnnet112`, `nnnet113`, `nnnet114`, `nnnet115`, `nnnet116`, `nnnet117`, `nnnet118`, `nnnet119`, `nnnet120`, `nnnet121`, `nnnet122`, `nnnet123`, `nnnet124`, `nnnet125`, `nnnet126`, `nnnet127`, `nnnet128`, `nnnet129`, `nnnet130`, `nnnet131`, `nnnet132`, `nnnet133`, `nnnet134`, `nnnet135`, `nnnet136`, `nnnet137`, `nnnet138`, `nnnet139`, `nnnet140`, `nnnet141`, `nnnet142`, `nnnet143`, `nnnet144`, `nnnet145`, `nnnet146`, `nnnet147`, `nnnet148`, `nnnet149`, `nnnet150`, `nnnet151`, `nnnet152`, `nnnet153`, `nnnet154`, `nnnet155`, `nnnet156`, `nnnet157`, `nnnet158`, `nnnet159`, `nnnet160`, `nnnet161`, `nnnet162`, `nnnet163`, `nnnet164`, `nnnet165`, `nnnet166`, `nnnet167`, `nnnet168`, `nnnet169`, `nnnet170`, `nnnet171`, `nnnet172`, `nnnet173`, `nnnet174`, `nnnet175`, `nnnet176`, `nnnet177`, `nnnet178`, `nnnet179`, `nnnet180`, `nnnet181`, `nnnet182`, `nnnet183`, `nnnet184`, `nnnet185`, `nnnet186`, `nnnet187`, `nnnet188`, `nnnet189`, `nnnet190`, `nnnet191`, `nnnet192`, `nnnet193`, `nnnet194`, `nnnet195`, `nnnet196`, `nnnet197`, `nnnet198`, `nnnet199`, `nnnet200`, `nnnet201`, `nnnet202`, `nnnet203`, `nnnet204`, `nnnet205`, `nnnet206`, `nnnet207`, `nnnet208`, `nnnet209`, `nnnet210`, `nnnet211`, `nnnet212`, `nnnet213`, `nnnet214`, `nnnet215`, `nnnet216`, `nnnet217`, `nnnet218`, `nnnet219`, `nnnet220`, `nnnet221`, `nnnet222`, `nnnet223`, `nnnet224`, `nnnet225`, `nnnet226`, `nnnet227`, `nnnet228`, `nnnet229`, `nnnet230`, `nnnet231`, `nnnet232`, `nnnet233`, `nnnet234`, `nnnet235`, `nnnet236`, `nnnet237`, `nnnet238`, `nnnet239`, `nnnet240`, `nnnet241`, `nnnet242`, `nnnet243`, `nnnet244`, `nnnet245`, `nnnet246`, `nnnet247`, `nnnet248`, `nnnet249`, `nnnet250`, `nnnet251`, `nnnet252`, `nnnet253`, `nnnet254`, `nnnet255`, `nnnet256`, `nnnet257`, `nnnet258`, `nnnet259`, `nnnet260`, `nnnet261`, `nnnet262`, `nnnet263`, `nnnet264`, `nnnet265`, `nnnet266`, `nnnet267`, `nnnet268`, `nnnet269`, `nnnet270`, `nnnet271`, `nnnet272`, `nnnet273`, `nnnet274`, `nnnet275`, `nnnet276`, `nnnet277`, `nnnet278`, `nnnet279`, `nnnet280`, `nnnet281`, `nnnet282`, `nnnet283`, `nnnet284`, `nnnet285`, `nnnet286`, `nnnet287`, `nnnet288`, `nnnet289`, `nnnet290`, `nnnet291`, `nnnet292`, `nnnet293`, `nnnet294`, `nnnet295`, `nnnet296`, `nnnet297`, `nnnet298`, `nnnet299`, `nnnet300`, `nnnet301`, `nnnet302`, `nnnet303`, `nnnet304`, `nnnet305`, `nnnet`

The figure shows the sequence of windows presented to the designer when ordering from the Henry Catalog.

1) The designer connects to the component library and browses its contents



2) When an order is placed, an active message is generated with the requested information



3) The active message is activated on the client side. Data is extracted into temporary files. Tools receive commands to open the files for browsing and later install them into the project database via the HUB.

Figure 7.6 Information Flow in Transactions with the Component Library

The figure shows the flow of information between the Henry design environment and an Electronic Component library, from selection and ordering to installation into the local project database.

Magic layouts. As a result, each project's information package is an active document, with files of various types containing hyperlinks between them. The active message with the information for a component is formatted as a MIME *multipart/enabled-mail* message containing 1) a MIME composite message, whose elements are the individual design data and documentation files to be installed and 2) a Safe-Tcl script.

The advantages of speed and reduced work to retrieve this information are obvious. Once the standard protocols and appropriate tools are in place, we can replace paperwork and many tool commands with a few button-clicks and the filling of an electronic form.

One way to protect information in a component library, while giving designers the possibility to use it, is by supplying clients the component's interface specifications and documentation along with an authorization to use a remote simulation service, such as the one described in Section 7.6.1. Only the simulation service has access to the detailed model and this can be set up to restrict its use to simulations submitted from authorized clients. This way, designers can see how their systems would work with a component without actually having access to its model.

## **7.7 How the Henry System Evolved**

We have been studying the integration of design and documentation via active documents in VLSI environments for the past four years. Over this time, the architecture of the Henry environment underwent major reformulations, as our perspective of the approaches to be taken changed. The major goals however remained the same. We wanted to provide a path for evolving from separate design and documentation environments to a new environment integrating both domains, while re-using as much as possible the existing software.

Initially, we were interested in using an existing hypermedia system, combined with existing CAD tools. The hypermedia system would be used as front-end to access the design data via the tools. We considered several options, including:

- Gain Momentum, a commercial hypermedia system

- HIP, a research hypermedia system built on top of the Picasso application framework [BSB90].
- FrameMaker, a text processing system with hypertext facilities.

We found severe problems with using any of these tools for our purpose. Gain and Picasso demanded about 64 megabytes of memory to run, and would take several minutes to start. In a 16M machine, the start-up time was around 15 minutes and the application would swap continuously. Two years later, we had to conduct our experiments using even less powerful machines while running the design tools concurrently<sup>1</sup>. For this reason alone, these systems would have to be excluded. In addition, we also found other major problems related with the authoring capabilities. Both Gain Momentum and Picasso were essentially closed systems with respect to inter-tool communication. Apart from a facility to invoke shell commands, no other interface was offered.

Gain Momentum seems to have been designed for creating graphically sophisticated presentations that could later be browsed repetitively by many users. On the other hand, we were interested primarily in tools for environments where the authors and readers are the members of the design team and authoring does not require knowledge of a new complex user interface.

FrameMaker is considerably less demanding in terms of resources. It was still a resource demanding application for our intended experimental environment, but we still ended up integrating it as the choice tool to display formatted documentation. FrameMaker has a very sophisticated API that enables automatic insertion of insets from other running tools, emulation of all keyboard interactions and simple hypertext support. However, its extensibility for our purposes is very limited. There is no extension language, only a keyboard macro facility which does not allow invocation of hypertext commands. With Version 4, Framemaker now enables the extension of the user interface with the addition of new

---

1. The usability experiments of the Henry System will be described in detail in the next chapter.

menus. However, this facility is still just a minimal supplement for our needs and did not exist at the time. In addition, Framemaker needs to operate in a separate major mode to make hypertext commands are available, which disables all other commands. This makes simultaneous browsing/authoring of FrameMaker documents very hard.

After evaluating and dismissing all the options based on an existing hypermedia front-end, we decided to write our own. The Navigator was conceived as an electronic design notebook, architecturally similar to the VNS [Gorr91]. The Navigator had a much less sophisticated user interface and documentation processing facilities. This made it a smaller application, suited to run in the low-end workstations where our initial experimentation with the use of the system in actual design took place. We designed the Navigator to make the creation of links to external design tools much simpler than in any other system. It supported multiple communications protocols and was easily extended, as it was programmed in Tcl.

However, our experience with the system made us consider a major shift in the overall architecture of the environment to make the system more flexible. There were too many questions for which the solution based on a common front-end provided no good answers. What would happen when the information in the notebooks had to migrate to other documentation tools to produce the final user's documentation? Why was it necessary to have the Navigator running when activating a "link" between two external tools, such as Spice and FrameMaker? At this point, we began to understand that it was necessary to separate the communications subsystem of the Navigator from the user interface for manipulating documents. We also realized the true importance of designing the system as a symmetric tool ensemble, where every tool could be run both as a client and as a server of every other tool.

The communications subsystem of the Navigator became what is the HUB in the current version of the Henry System. The HUB runs as a separate process. We began to look to the "commands" sent from the Navigator to design tools as "messages."

This architecture also looks identical to *Tooltalk* [SunS92]. The Navigator and Tooltalk adopted the same basic principles for intertool communication, such as stateless communications. At the time, we considering replacing of the HUB by a commercial Tooltalk server. However, Tooltalk only became available recently and there are practically no design tools supporting it even today.

After further experience, we began to realize that a pre-defined set of messages to interface with tools would not provide the necessary support for some of the interactions we needed to perform from active documents. A fairly large sequence of messages is required to supply the inputs for a simulation, activating a simulation and request the return of the results to a simulator. Some interactions, require the orchestration of a sequence of commands that have to be sent concurrently to distinct tools.

That lead us to consider active messages, containing programs for evaluation instead of messages with pre-defined formats. Tcl seemed the natural choice, given our experience and the possibility of reusing most of the existing code. In addition, as design teams begin to span multiple organizations and electronic commerce services become available, there is the need to send design data in the messages. Careful attention to the security aspects of the protocol is necessary. At that point, we decided to adopt the Internet standard message exchange protocols and MIME message formats for intertool communication. Then, we learned about Enabled-Mail using MIME and Tcl. We reused its conceptual model for active message delivery at the architecture level and most of the Enabled-Mail prototype implementation in the HUB.

## **7.8 Summary and Conclusions**

The Henry System is now on its second major version and its architecture and operational model have reached stability. We believe that new design flows in future industrial projects integrating design and documentation tools will reuse many of the techniques first demonstrated by the Henry prototype.

In a future version of the system, rather than refining the implementation of its components, we would try to make use of more robust software, now available in most workstation operating systems, such as session management services.

While we were developing Henry, Tooltalk was introduced, becoming part of COSE, and is now endorsed by CFI. Although there are not many design tools currently supporting Tooltalk, we expect its popularity in the CAD world to increase significantly in the next years. The next version of the HUB would certainly re-use Tooltalk's message services, using specialized communications for those tools not supporting Tooltalk's message standards.

Henry's Message Handling Layer services still need further development. They need to support a links database and additional services based on it, such as a new process management system. This implies that we need to support more complex data structures. This will necessarily imply a re-implementation in C++. Based on our experience, a Tcl-only implementation such as the one we have now would not be manageable.

Instead of using public domain design tools, the new version of Henry should have interfaces to the best commercial tools available. This is now becoming possible, as more CAD vendors provide their tools as part of CAD frameworks that provide programming interfaces for controlling the tools.

Finally, the Navigator should be replaced by a more user-friendly document editor with support for inter-tool communication and more sophisticated text processing capabilities. Among many other additional features, we believe that it should also read and write SGML documents and be well integrated with the World Wide Web. Documents should be opened as easily when given their location in the form of a URL as they are now when we give their location in the file system.



“There is nothing more difficult to carry out, nor more doubtful of success, nor more dangerous to handle, than to initiate a new order of things. For the reformer has enemies in all those who profit from the old order and only lukewarm defenders in those who would profit by the new order...”

— Machiavelli

## Chapter 8

# Active Documentation Experiment

This chapter describes our experiences with making available an early prototype of the Henry system for use by the students of a VLSI design course. There were two main reasons for undertaking this experiment. First, we wanted to verify that Henry accomplished its main goal, the production of active documents combining design and documentation tools. We believed that these could be obtained without adding a significant amount of overhead in documentation or data management work to the design team. Secondly, to understand the strengths and weaknesses of the documentation model, we needed users. The design methodology adopted for the Henry was based on *user centered design*, that is, obtaining early user feedback and iterative development [Norm86]. The study involved five groups of two students during one semester. In this chapter, we discuss the goals set for the experiment, how it was run and what we learned from it. The experiment enabled us to detect many design flaws of the initial version of Henry. The comments received were encouraging about the possibilities opened with the new documentation tools used during the design process, while pointing at the same time to the limitations of the prototype.

## 8.1 Introduction

We followed usability engineering techniques [Niel92] for the design and implementation of the Henry System. These include prototyping, early use, testing and iterative development. We discussed the implementation of the Henry prototype in the last chapter. Here, we describe its use, testing and the modifications to the system derived from our observations.

The version of the system under test described here is not the same as the one described in the previous chapters, but an initial prototype. This underwent substantial architectural changes after we incorporated the changes by the experiment described in this chapter (see Section 7.7 on page 148 for a description of the organization of the system at the time the experiment took place).

The initial users were a group of students of *EE141 — Digital IC design*, a course offered in the Fall Semester of 1993. There were 80 students enrolled, mostly undergraduates in the senior year. About 10% of the attendants were graduate students majoring in IC design. The course covered the design of digital logic gates and their interconnection [Raba95]. Students attended eight laboratory sessions and completed two design projects. Four lab sessions, called the *software labs*, involved the use of design tools; the other four, the *hardware labs*, required the use of instrumentation to measure electrical characteristics of digital ICs and circuits assembled with discrete components.

Of these students, we selected ten from a group of volunteers. They used the Henry System throughout the semester, for completing the design and documentation of the software labs and design projects. Students worked in teams of two. We had to limit the participation because the prototype makes extensive use of FrameMaker, for which we only had five licenses. In addition, the experiment had to take place during the regularly scheduled laboratory hours, and we wanted to provide the most focused assistance. As a result, the number of subjects, while not large enough for producing statistically significant conclu-

sions, turned out to be ideal for the first real attempt to use the system to produce simple designs.

The remainder of this chapter is organized as follows:

<b>Section</b>	<b>Presents</b>	<b>Page</b>
8.2	The goals we set for the period of test.	155
8.3	Adaptations to the system to operate in the educational environment it was set to run.	156
8.4	The documents produced during the experiment.	158
8.5	The collected data.	164
8.6	Hypotheses and validation criteria for a second, industrial strength experiment.	172
8.7	Summary of the main results and conclusions.	174

## **8.2 Setting the Goals**

With this experiment we wanted to observe the result of exposing the students to new methodologies and design aids for designing and documenting their class projects. In the process we would also remove the obvious bugs and detect any design flaws at the architectural level. The knowledge gained would then be applied to the design of subsequent versions of the prototype.

The experiment took place under special constraints. Participation in the project was voluntary. We felt it wouldn't be ethical to force the students to use a system at this early stage of its development. For this reason, we made very clear to the subjects of this experiment that:

- They only had to participate if they wanted to.
- They could leave the experiment at any time.
- Their participation in the experiment had no influence on their grades.

- Reports produced with the Henry System would be graded based exclusively on the quality of the designs; the media used to write them would have no influence.
- Occasionally they would receive small questionnaires, but they were not obliged to respond to them.
- Their interactions with the tools would be automatically recorded, but the information would be kept confidential and would not be used for grading purposes.

An experiment run under this setting could not be used to produce scientific evidence demonstrating the advantages of Henry's approach. The number of participants, complexity of designs and duration of the experiment were far too small to produce results that could be extrapolated for an industrial VLSI design project.

In this experiment, we wanted to observe users and see if they were spontaneously combining design and documentation tasks. We also wanted to check if the design methodology supported by Henry leads to better quality designs and documents and if it is more time consuming.

The main goal for this period of initial usage was to obtain the best possible indication of the usability of the Henry System. We designed a few questionnaires to be filled voluntarily. We gave these to our subjects just after the projects made with Henry were completed. In addition, we tried through direct informal contacts to perceive the reactions of our users.

### **8.3 Adapting the Prototype to the Users Environment**

We started the development of the Henry System well before we were given permission to undertake the experiment with the students of this VLSI design course. The system we had before the experiment was substantially different than the one the students used. In this section, we describe the major extensions and modifications made to adapt Henry to its experimental users. In the preparation stage we were driven by Paul Heckel's rules for

designing a user interface [Heck91]. These emphasize knowledge of the needs of the audience and design optimization to satisfy those needs.

The tools used in the course had to be integrated. Henry was initially designed to operate with the Octtools framework [Harr86]. In this course, the students would be primarily using the Magic layout editor and the Spice simulator. This forced us to write an interface to send and receive commands from these tools.

Henry had to be modified to consume as little system resources as possible. A methodology employing simultaneous design and documentation is more demanding in terms of computational resources than the traditional design followed by documentation methodology. This is because the tools that support these two activities must run concurrently. Further, the workstations available to the students were somewhat underpowered: DEC 2100 workstations with 8MB of memory each<sup>1</sup>. The waiting time to start one of the tools in this environment was about 2 minutes!

To work around these limitations, we extended the Henry session manager to enable tools to be launched from any machine of the network. We also programmed it to start by default all instances of the each tool on a dedicated machine. This way, applications could share memory and start quicker. The Henry session manager also was modified to pop-up a substantial number of informational messages. These entertained users with data about the progress of their tool invocations. Progress messages had the effect of reducing the frustration associated with using such a complex system under such limited resources. Once the tools were started however, command activation would take place in a few seconds. With these modifications, we were hoping to have made the environment minimally usable for the experiment.

We converted the manuals and tutorials of the tools the students would be using during the course into active documents. This was achieved by translating the *troff* based documenta-

---

1. Later in the experiment, we had access to DEC 3100 machines with 24 MB of memory.

tion into FrameMaker hypertext documents. In these documents, we added messages with commands to the tools being taught. Students could see the commands being sent to the tool being introduced as they read about its operation. The on-line tutorials were used to show prospective users the advantages of an integrated design and documentation system, and what could be done with the Henry prototype. Then, as they decided to enroll in the experiment, on-line tutorials were used to introduce students to the operation of active documents while learning the tools required for the course.

We wanted the system to look as simple and intuitive as possible to the users. For this reason, we removed all the features and commands that would not be used in the experiment. For instance, as the students would not be interacting with a design database, but would simply keep their data in files, we removed all the facilities to manage design contexts. Individual design configurations would be managed by simply putting the files related to a design configuration in a directory. Interaction between design documents and the various configurations was going to be achieved by programming the active documents to invoke the Unix *cd*<sup>1</sup> command automatically every time a new page describing a different configuration was about to be rendered.

## 8.4 The Documents Produced

Table 8.1 summarizes the documents produced during the experiment. There were two main types of documents that we wanted the students to produce: software lab reports and project reports. The contents of these two types of documents and how they were produced are described in the remaining of this section.

One group decided to use Henry to write some of their hardware lab reports. However, as the students had no access to workstations in their hardware labs, these documents were elaborated *after* the measurements and observations were captured. These documents are less interesting to our study because they do not contain interactive design tools' activa-

---

1. *cd* — change directory.

<b>Week Due</b>	<b>Title</b>	<b>Description</b>	<b>Num of Documents</b>
3	swlab 2	Magic Layout & DRC	5
4	swlab 3	Magic Circuit Extraction & Spice Simulation	5
6	hwlab 1	CMOS Inverters (extra)	1
9	proj 1	Design Logic Gate	1
10	swlab4	Detect error in layout of ALU using IRSIM simulation	3
11	hwlab 3	BJT Switching times (extra)	1
15	proj 2	Transmit signal through a long metal line and output pad	1

Table 8.1 Summary of the Documents Produced During the Experiment

Students used the Henry System throughout the course. The table indicates, for each document type, the week in the course when it was due and how many groups of the initial five returned active documents

tions. They simply present the captured data in tables, bitmaps and text annotations. Hence, they are not active documents in the sense we understand them.

### 8.4.1 Software Lab Reports

For the software laboratories, we created templates that allowed the students to follow the sequences of operations they had to perform and introduce their observations directly into the active documents.

The first software lab session was dedicated to “hands-on” familiarization with the tools. The students that volunteered to participate in the experiment had access to the on-line active tutorials and manuals. For instance, they could read the Magic tutorials on one FrameMaker window and then type in the suggested commands on an adjacent Magic window. Or, they could simply click in a hypertext button in the Navigator or FrameMaker and see the result of the execution of that command on the Magic window (see

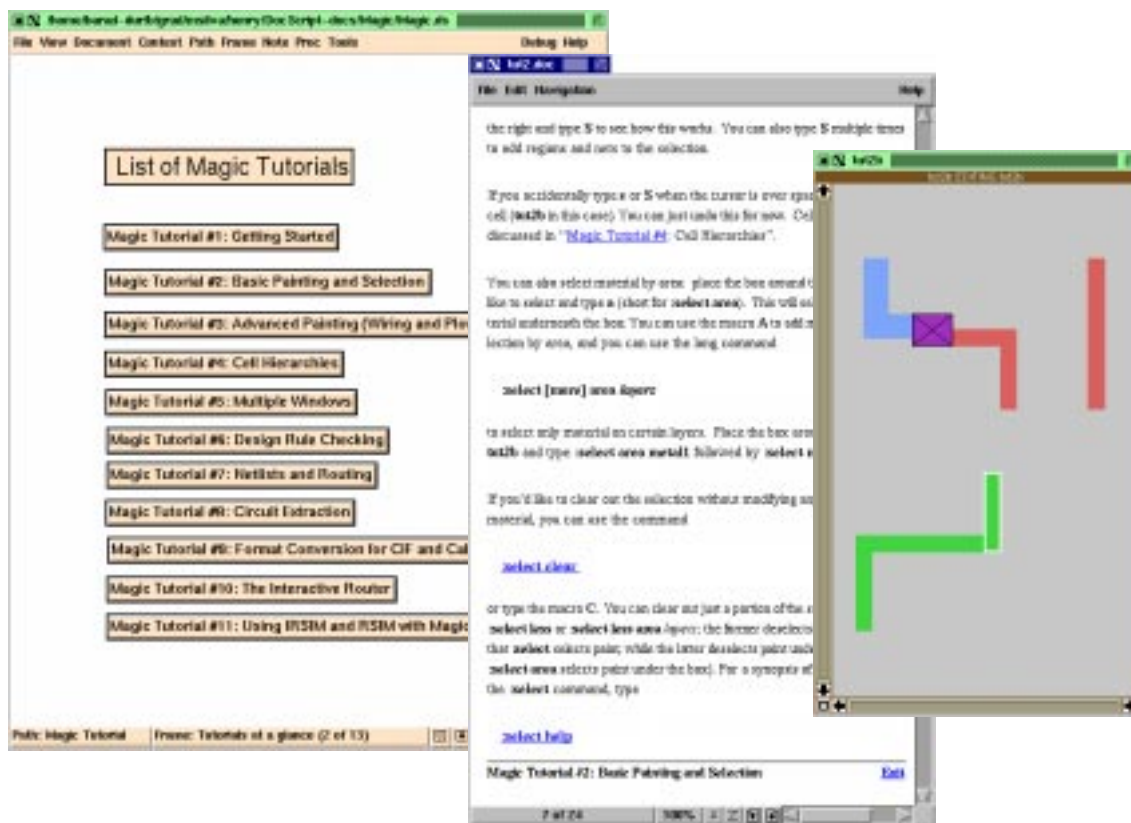


Figure 8.1 The Henry Help System as an Active Document

The tutorials and *man pages* for the tools used were converted into FrameMaker documents and then extended with hyperlinks to the design tools. We chose small page layouts and large fonts, so they could be easily readable from the screens while not taking too much real-estate from the displays. This way, users could “play” with the tools while reading their documentation. At the same time, active documents could send the commands being explained to the tools for illustration of the concepts being presented. We created indexes of the FrameMaker documentation in the Navigator, so that users could be exposed to it since the beginning. The figure shows (1) the Navigator’s index to the Magic tutorials, (2) a Magic tutorial page in FrameMaker, and (3) the layout described in the FrameMaker page shown by Magic.

Figure 8.1). The students who did not participate in the experiment had to sit with the binders with all the documentation open on their knees, while they typed-in the commands they were going to learn in that tutorial.

On the second lab session, the students had to design their first layouts (2 inverters and one NAND gate). Through the on-line tutorials, they learned how to produce designs of



minimum area respecting the spacing constraints imposed by the manufacturing process. We provided a Navigator template. The template had one frame for each gate that had to be designed with the buttons already configured to point to the files they would have to create and the tutorial pages they had to read.

The third lab session consisted on the extraction of the netlist representations of the gates designed in the previous session, followed by observation of their electrical characteristics with SPICE3. This lab session was run the same way as the previous. This time however, the Navigator templates were enriched with more buttons to access the text files containing the simulation decks, run the tool that extracts a SPICE netlist representation from a Magic layout, and invoke the simulator. Figure 8.2 shows the windows of an active document produced by a group of students as a report for this lab session.

In the fourth lab session, students started from the layout of an eight-bit ripple-carry adder which contained some errors. They had to perform logic simulations and identify the signals that did not have the expected behavior. Based on that information, they had then to identify the wrong connections in the layout, fix them and verify that the modifications produced the intended result. Figure 8.3 shows a Navigator frame and the windows started from it for working on the identification of design errors. For this lab, we decided to provide a lab template distributed between two documentation tools. Hyperlinks were mostly invoked from the Navigator, while data was to be entered using FrameMaker. The decision to take this approach was based on the user's feedback. Each tool was to be used for the purpose it was best suited. The Navigator acted as a control panel for invoking the tools and synchronizing the contents of the various windows, while FrameMaker was used to display the tutorials and type-in the answers to the lab questions. This strategy was very well accepted by the users. Two of the groups that had given up using the system, decided to give it a second try and returned their reports as an active document. We also observed that the students at this time were finally mastering the use of the system as it was intended: two of the reports were returned at the end of the three-hour lab session, instead of one week later as they would normally have to. This was only possible because they

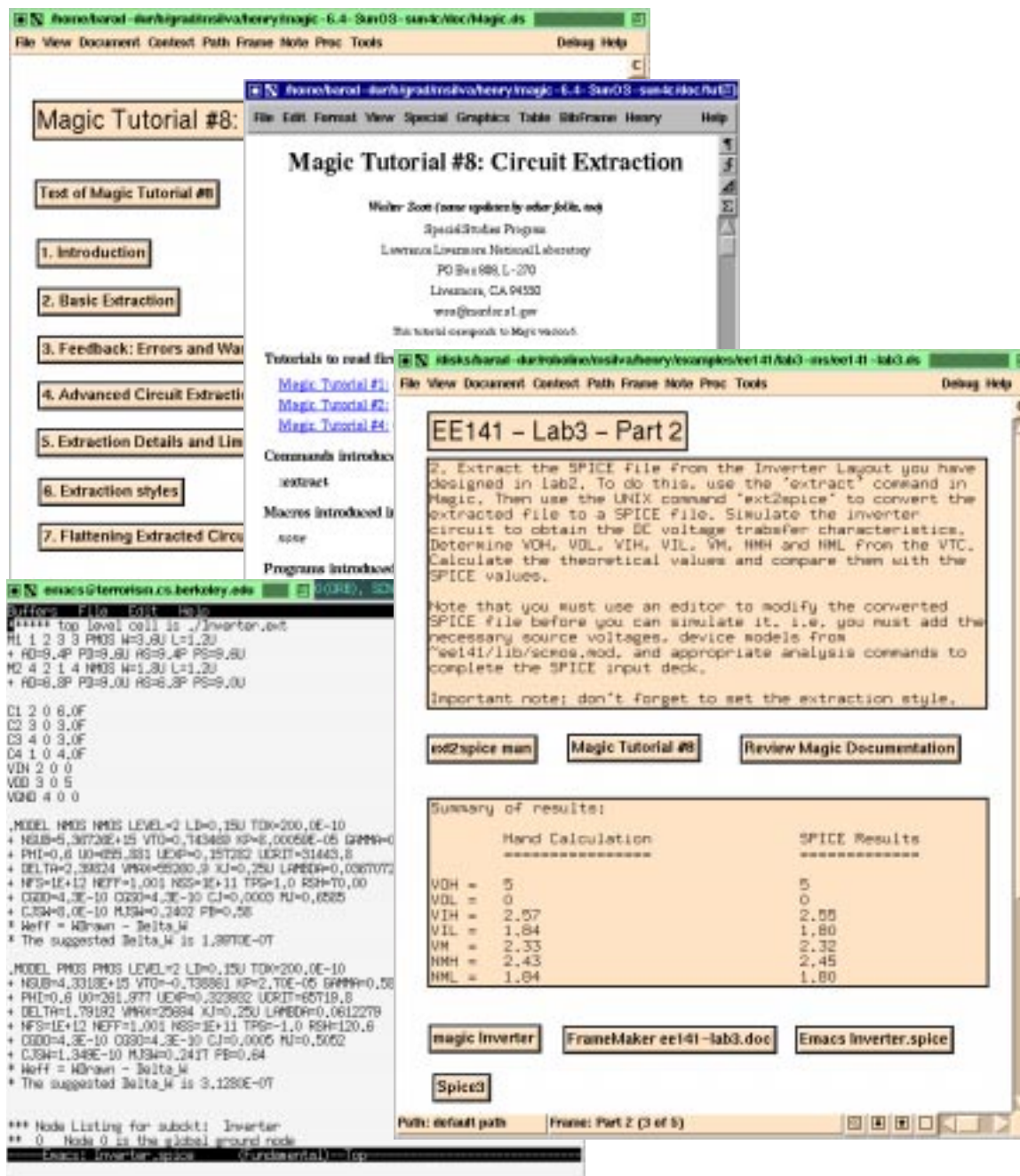


Figure 8.2 An Active Document Containing the Report of Software Lab 3.

The figure shows a frame in the Navigator for performing and documenting the design tasks in part 2 of software lab 3. The original template included buttons to access design files and the pages in the help system relevant to the task to be performed. Students have enriched the frame by filling in the text notes and adding new buttons to edit the files they produced.

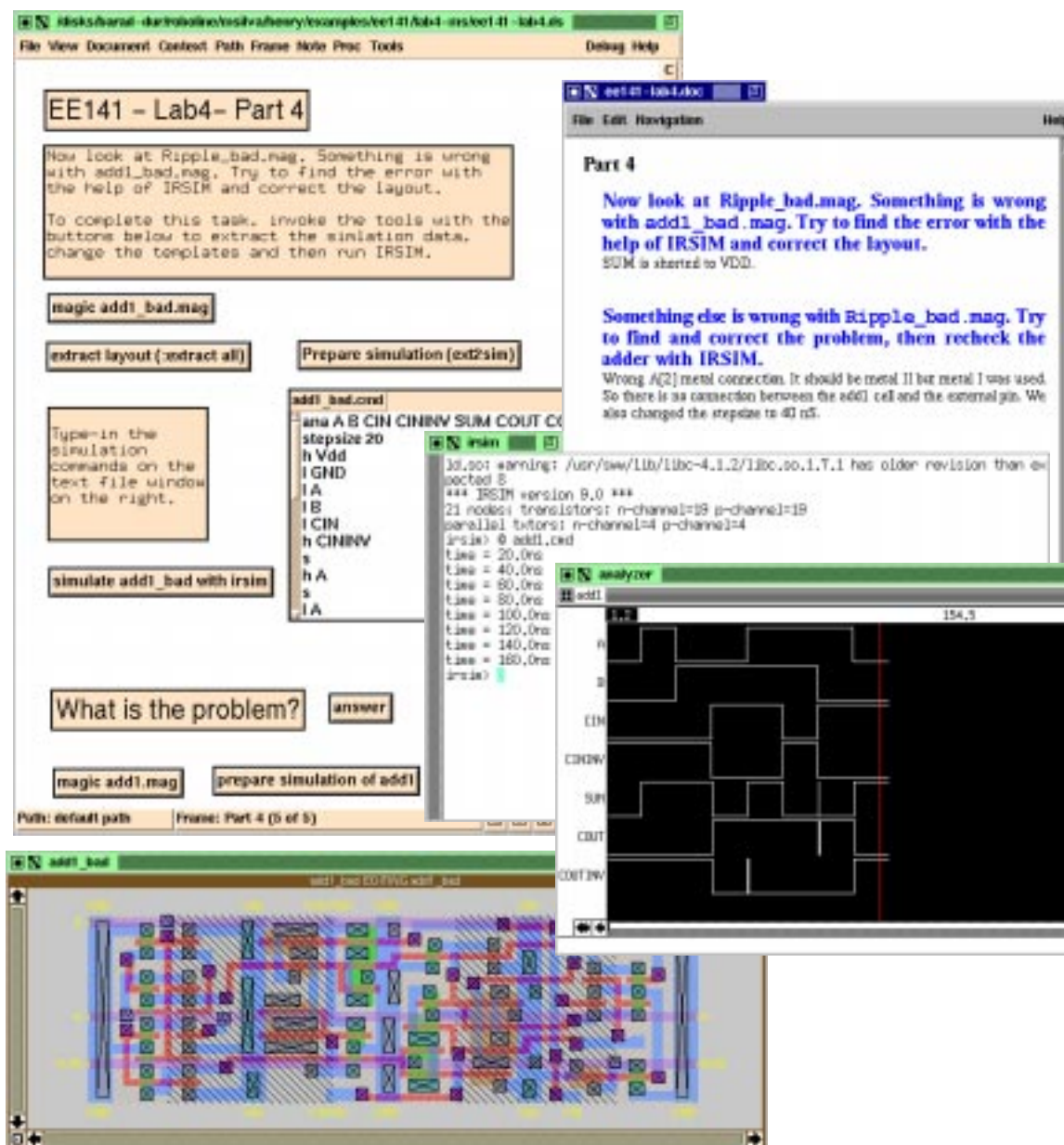


Figure 8.3 An Active Document Describing the Detection of a Layout Error

The figure shows a report of the fourth software lab session. This involved layout extraction, detection of errors with a logic simulator and their correction using the layout editor. In this session, we used the Navigator (window on top, left) mostly as a control-panel where the students could place or modify the behavior of the buttons to activate the tools. The answers were typed-in on FrameMaker (top right). In the center, we can see the windows for entering simulation commands and visualizing waveforms (from the IRSIM simulator). The Magic window at the bottom shows the layout.

were focusing on completing the design and documentation parts of the problem at the same time.

### 8.4.2 Projects

The first project consisted in the design of a CMOS gate whose logic function of given. The best design would minimize the product  $A \times t_p^2$ , where  $A$  is the area of the gate and  $t_p^2$  the square of the gate propagation time. All logic implementation styles presented in the lectures were to be tried. Students had to look to all approaches, select the two most promising, create and simulate an initial version of the two, and finally optimize the one that produced the best result. We provided them with only a very crude template. This time the objective of the project was to let students find the appropriate design flow. Figure 8.4 shows the windows of an active document with a design report for this project.

The second project consisted in the design of a circuit to send a signal across a chip on polysilicon, and then off-chip onto a load capacitance. The design had to meet an average propagation delay requirement while minimizing a combination of power and area. The signal propagated across two sections. In the first, the students had to design buffers to operate as repeaters at equally spaced points in the transmission line. The second section involved the design of a multi-stage output buffer to drive the off-chip load. The approach to be followed consisted in finding the optimum design point iteratively, through variations in several design parameters, such as the division of delays between the two sections the circuit. Figure 8.5 shows a representative frame of an active document for this design.

## 8.5 Observation Data

We collected feedback on the usage of our system in three forms:

1. Traces of tools' usage.
2. Questionnaires.
3. Direct observation and informal contacts with the subjects.

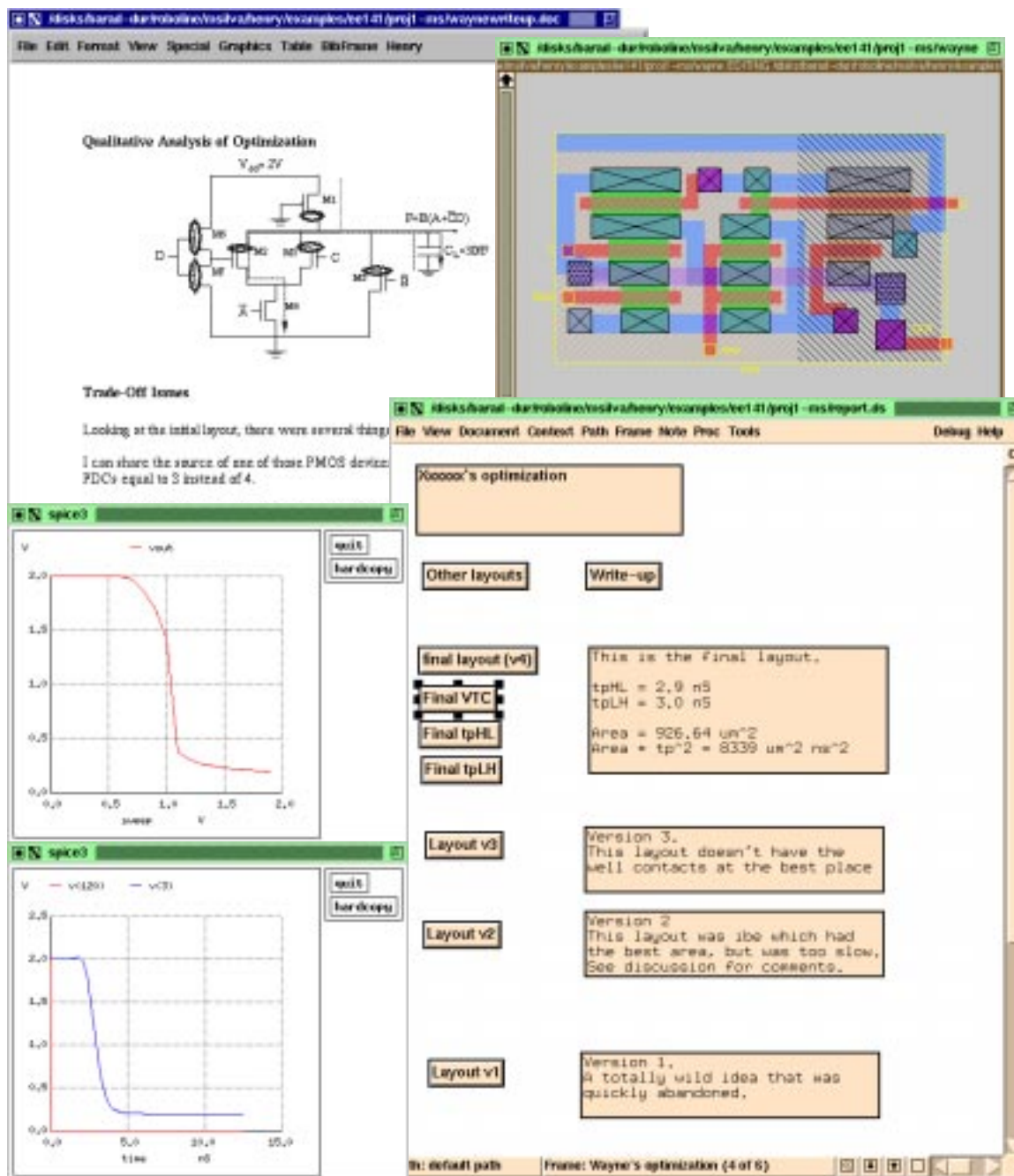


Figure 8.4 An Active Document Illustrating the Design of a Logic Gate

The Navigator's frame summarizes the project results. There are buttons to pop-up the windows of (1) FrameMaker, to show the schematic and discussion of the optimization process, (2) Magic, to display the final layout, and (3) Spice, to simulate the VTC and propagation times. The Navigator's window also contains the summary of the design process. It shows the basic characteristics of previous versions, why they were abandoned, and the buttons to display their layouts on demand.



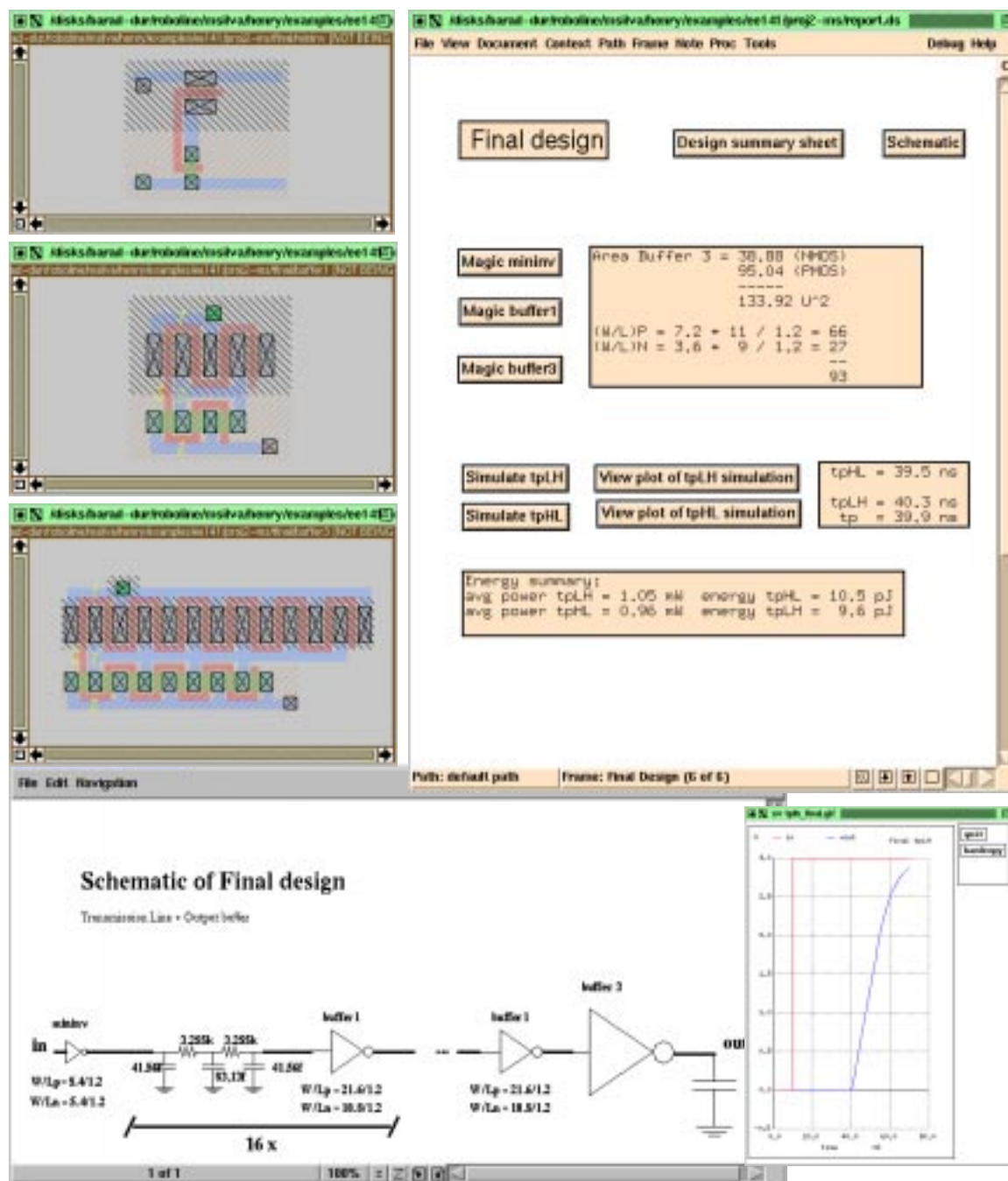


Figure 8.5 An Active Document for the Design of a Long Interconnect

The Navigator Frame summarizes the project data. It contains buttons to display a design “summary sheet,” the schematic of the final design and the layouts of the various buffer stages. From the Navigator’s window, the reader can choose to re-simulate the final propagation time (an operation that takes several minutes) to quickly display an image with the simulation result.

We discuss each of these types of information in the remaining of this section.

### **8.5.1 Automatic Usage Statistics Collection**

We decided to instrument the tools to collect usage data. This had the appealing advantage of enabling us to trace the users interactions with a document in a non-intrusive way. From the traces it would be possible to observe how the system was being used. For instance, we would be able to see if the users were designing and documenting simultaneously, or were still designing before documenting.

For the usage data collection, we set up a server process running on a pre-defined host and listening on a predefined port. This server collected messages sent automatically by each tool of the Henry System to signal various types of events we wanted to monitor. As an example, we built wrappers for the design tools used in the experiment to send a message when a tool was started or stopped. This would allow us to compare the number of times the tools were started from the navigator or independently and the time lengths tools were active.

Unfortunately, as we later found, the data collected for this purpose were not meaningful, given the conditions under which the design projects were completed. The ratio between tools being started from active documents to tools started independently we monitored was 0.11. However, we observed that the heaviest users of the Henry System did not use the instrumented design tools for their projects most of the time. They opted for using the original versions, available in the much more powerful workstations of the various research groups they were working for at Berkeley. They used non-monitored tools to do most of the design and documentation work. The project's data and documentation was moved to the instructional network only near the end of the project, to create and test the links between the various pieces of data.

The Navigator was instrumented to send a number of messages when an usage event occurred, such as flipping a frame or a new document was opened. Table 8.2 lists the event

<b>Event</b>	<b>Description</b>	<b>Number of Events Collected</b>
OPENVPT	opened new viewport on a document	662
CLOSEVPT	closed viewport on a document	448
SAVEDOC	saved document into file	423
CLOSEDOC	closed document	427
FLPFRAME	jumped to a new frame	2355
CGHPATH	selected a new path	16
TOOLSTART	sent “start” command to a tool	82
TOOLEXIT	sent “exit” command to a tool.	0
TOOLOPEN	sent “open” command to a tool.	584
TOOLCLOSE	sent close command to a tool	0
TOOLCMD	user sent any other command to a tool	263

Table 8.2 Navigator Usage Statistics

The events related to user’s interaction with the Navigator that were logged into the statistics server. Each message logged contained not only the event type but also identification information of the user, document edited and information about the number of objects of each type in the document’s data structures. That allowed us to trace the number of times documents were consulted and how they grew over time.

types that could be generated from the Navigator, their meaning, and the number of occurrences logged during the period of the experiment. Within these messages there was information about the number of objects of various types within the active document, such as the number of frames and notes. This allowed us to observe how the documents grew as the projects evolved.

### 8.5.2 Questionnaires

We planned to offer three questionnaires, at different times in the experiment:



1. After the tutorials and the first two lab reports were due (week 8);
2. Immediately after the first project was returned (week 10);
3. Immediately after the second project was complete (week 15).

We strived to ask questions that could be answered easily by the participants and give us an idea of how the system was being used. We also encouraged the students that were not using the active documentation system to answer the questionnaires.

The first questionnaire was given relatively early and was designed to collect the general reactions to the use of the system. The other two were much more objective: we requested more quantitative answers. We wanted to compare the relative times spent doing design and documentation and the grades obtained by the participants and see if the students using the Henry System were getting comparatively better grades and/or spending less time.

The answers received were small in number, but high in relative terms. Three out of the 5 groups involved in the usage of the system replied to all questionnaires. The first questionnaire was answered by all groups. Unfortunately, given the voluntary involvement of the students and the absence of a motivation to participate by those not using the Henry tools, we did not receive answers from the students not using the system.

Answers to the first questionnaire were sent back not immediately after it was handed out, but several weeks later. Most were received 5 weeks before the end of the semester, when the second round of mid-terms was finished. Table 8.3 summarizes the results.

The answers to first questionnaire give the best insight on the students perception of the Henry System. After the tenth week of the semester, competition for the best grades became all consuming. For many students, the final grade in this course decides their admission to the best graduate schools. It was very difficult to get any of the student's time for collaborating in the active documentation experiment once that point in the semester was reached.

Question/Possible Answers		Replies
1.	How have you used the Navigator so far (check all that apply)?	5
1.1	Never	0
1.2	To access the tutorials	3
1.3	Return my reports	4
1.4	Start design tools.	2
2.	For how many hours?	5
2.1	Less than 1	0
2.2	Between 1 and 5	2
2.3	Between 5 and 10	1
2.4	More than 10.	2
3.	3. What is your impression so far? (Comment)	3
4.	Are you spending more or less time writing your reports than you would if you were writing a paper-based report?	5
4.1	More time	3
4.2	Same time	0
4.3	Less Time	2
4.4	Comments	5
5.	Which documents are easier to understand? Active documents or the paper documentation?	5
5.1	Active documents	4
5.2	Paper documentation	1
5.3	Comments	3
6.	What do you think of the general idea of combining the design tools with the documentation tools the way the Navigator does?	5
6.1	Good idea	5
6.2	No big deal	0
6.3	Bad idea	0
6.4	Comments	5

Table 8.3 The First Questionnaire

The Table summarizes the 9 questions and answers of the first questionnaire. The questionnaire was handed-out just before they started the first design project. At this point, students already had considerable experience with the design and documentation tools of the Henry System and the templates for filling-in the lab reports.

Question/Possible Answers		Replies
7.	Resources. Do you think the speed, available memory, and screen size available are appropriate for running the Navigator?	5
7.1	Not enough resources	5
7.2	Resources OK	0
7.3	More than enough	0
7.4	Comments	1
8.	What is the main limitation of the current prototype?	4
8.1	The way tools inter-operate	3
8.2	The user interface	0
8.3	Other: describe	1
8.4	Comments	2
9.	Have you ever thought of anything that could improve the usability of the system? (Comment)	3

Table 8.3 The First Questionnaire

The Table summarizes the 9 questions and answers of the first questionnaire. The questionnaire was handed-out just before they started the first design project. At this point, students already had considerable experience with the design and documentation tools of the Henry System and the templates for filling-in the lab reports.

The few answers to the two questionnaires about the projects indicate that students saw no advantage to writing on-line documentation. Contrary to the template-based lab reports, project reports were much more efficient to write by hand with the Henry tools. Except for one group that was determined to use the system until the end, they all decided to abandon the use of the tools for documenting their projects. The reason was that introducing schematics and typing math formulas was too time consuming and was not worth the effort: project reports would be thrown away once graded by the instructor.

### 8.5.3 Informal Contacts

Informal contacts did not give us quantitative data, but they were the source of the most important feedback during the experiment. We encouraged “bug reports” and suggestions. We received many, both directly and by electronic mail. In the first weeks of the experi-

ment, there were many obvious design flaws and programming errors that had not been exercised. This form of contact provided a very fast feedback path that enabled us to produce quick fixes. Without this fast feedback, the experiment would not have survived. The students' interest fades rapidly if the prototype under test does not meet minimum usability requirements and the main complaints and suggestions for improvement are not addressed.

The most interesting suggestions for improvement were given orally. Students tend to lose interest in using the system if they do not see a continuous genuine interest in their participation. They also need to perceive that someone is listening to their input. In addition to being with the students during the regular sessions, we also decided to make frequent unsolicited visits to the instructional software labs during the off hours, while the students were working on the class projects. This way, we could be in more direct contact while they were using the system and also provide better assistance.

## **8.6 Hypotheses for an Industrial Strength Experiment**

A complete human factors study was out of the scope of this experiment. This would require a level of involvement and commitment from the students to the study we could not expect. For example, in an evaluation scenario we developed, we considered dividing the participants into two sub-groups to prove the effectiveness of Henry (the first hypothesis). One would use the Henry System and the simultaneous design and documentation approach. The other would use the traditional tools and methodology. Both would be under observation. Protocol analysis of the design process would be used to determine the findings from the experiment [CWE91].

However, the students' single motivation to participate was the excitement of having an opportunity to use a new tool and being the first to attempt a new circuit design methodology. In this setting, it would be impossible to convince students in the group not using the Henry System to freely accept to participate in the experiment. It would also be impossible to prove the third hypothesis (better reusability) under these conditions. To accomplish

this, we would need the users to work on a follow on project that used their previous project. However, it would be unrealistic to assume that we could find and convince anybody of the scientific interest of re-using or re-designing an already graded class project.

The Henry System is now at a stage where it is ready to go for a new iteration of the usability test cycle involving a test group of designers in an industrial environment. In our view, Henry only needs to integrate a set of commercial VLSI CAD tools before being submitted to a test by such group. In the new test, we would like to prove the following hypotheses:

*Hypothesis 1:* The Henry System effectively supports design methodologies involving simultaneous design and documentation.

*Hypothesis 2:* Documents produced with Henry do not demand significative additional effort.

*Hypothesis 3:* Designs and their documentation when produced with Henry are more reusable.

The following criteria could be established for testing the relative validity of each hypothesis:

1. For the first hypothesis, we would again trace tool invocations and the size of design data and documentation files produced by the students using the Henry System. The traces would indicate whether design and documentation data had been produced concurrently or in sequence.
2. To measure the second hypothesis, we would count how many students were still using the system by the end of the experiment. If only a relatively small fraction of the users decided to abandon the experiment, we could assume they were finding it useful.

3. To assert the validity of the third hypothesis, we would give a questionnaire to future users of the data produced by the test group. The questionnaire would ask for qualitative comparisons between the design information produced before and after the introduction of the Henry System.

## 8.7 Summary and Conclusions

During this initial period, Henry was used to help the design and documentation process in multiple situations that are common in VLSI design. Henry was substantially tested in design projects involving interaction between data entry tools, translators and simulators. Projects included designs that followed a pre-established flow, based on document templates, and designs for which there was no pre-established rationale and designers had to create their own methodologies as part of the design process.

In general, the comments received were encouraging about the possibilities offered by the new documentation tools used, while pointing out their limitations.

The main benefit we obtain from this experiment was the learning experience. By exposing the prototype early, we were able to eliminate many design flaws of the initial prototype. This enabled us to demonstrate the integrated design and documentation architecture presented in this dissertation.

The current version of the Henry prototype contains many of the modules of the initial system tested by the students. The fundamental difference is that they are organized in a different way. Most architectural changes resulted from our observations during the experiment. These included:

- Handling of inter-tool communications and link dispatching functions in a separate process, independent of all the tools (the HUB). This makes it possible for a designer to activate links between tools, such as FrameMaker and Magic, without running the Navigator.

- The abandonment of the Navigator as a documentation tool in favor of FrameMaker. The advantages of a better interface for simultaneous design and documentation do not compensate for the data entry limitations.
- Introduction of support for handling messages and commands sent asynchronously. This results in a more user-friendly way of processing commands that take very long to execute, such as starting a new design tool. When a tool needs to be started, a window that indicates the progress of the operation is displayed, giving the user the chance to perform other interactions concurrently.

The experiment could not prove the advantage of using an integrated design and documentation approach in VLSI design in general. However, the experiment indicates that a methodology based on active documents can be effective when pre-defined design flows are used. In this case, templates can be made available for the design data and documentation, considerably accelerating the production and improving quality of the design information.

The experiment also shows that the use of active documents requires designers to spend a considerable amount of time learning the mechanisms of link creation and activation between the tools. However, this limitation is likely the result of having to use an initial prototype. A more robust implementation of the Henry System would integrate a user interface supporting compound active documents such as those available on current personal computers and the design tools we have in Unix workstations. In such an environment, the operations for creation and operation of links between portions of active documents would be much more intuitive.

“An artist never finishes his work; he merely abandons it.”

— Paul Valéry

“We are inventing new forms of doing business.”

— J. Marty Tenenbaum

## Chapter 9

# Conclusion and Directions for Future Work

We conclude this dissertation with a summary of the contributions, recommendations for a future experiment and directions for further research.

### 9.1 Research Contributions

The main proposal of this dissertation is the integration of design and documentation in a VLSI design system using hypermedia technologies. We introduced and demonstrated the use of active documents in VLSI design. These are multimedia presentations that incorporate invocations to the tools to display and modify the design data.

The combination of design and documentation systems offers a new way of creating integrated environments for designers in general, and introduces a new paradigm for VLSI design. Our experience developing and using the Henry System indicates that:

- For design methodologies that follow a pre-established flow, it is possible to produce better documentation more quickly. This is possible through the reuse of existing documentation as a template for new designs' documentation.



- It is possible to develop effective design methodologies that enable creating design and documentation in a single thread without intruding in the design process. Traditionally, VLSI design and its documentation are done separately. In our prototype system we have shown design documents being dynamically created as the designer enters commands to activate the tools.

Another contribution is the use of active documentation as a new paradigm for creating a common interface to heterogeneous tools and data used in system design environments:

- We added a new integration layer that hides heterogeneity by enabling designers to control the flow of information between independent systems through a document manipulation paradigm.
- We have shown how information-based services available through electronic commerce could be integrated into the design environment using this paradigm. Active documents become a vehicle for transporting design data and operations between frameworks in independent organizations, enabling the creation of virtual enterprises for development of electronic systems.

We developed a realistic model for creating a system supporting this integration, based on a new infrastructure that attempts to re-use existing framework services and design and documentation tools with minimal modification:

- From the data point of view, it introduces a new layer within the design database. This new layer contains descriptions of how the design data is organized and presented. It is a data structure with references to the design data, configuration and history data. It also includes mechanisms to tool invocations to present the data. The new presentation layer is organized as a set of active documents. Designers manipulate them just like the documents produced by documentation processing systems.
- Our model supports integration of heterogeneous tools in a unique way: we require tools to support a mechanism for remote command invocation, but do not require tools to comply to a specific protocol. Having a remote command invocation protocol makes

it possible to control the consistency between related information displayed by different tools. By not requiring tools to adopt an established protocol, we can integrate applications whose behavior we cannot modify. To achieve integration, we proposed the concept of an extensible communications server that can accommodate multiple tool-specific communication protocols.

With respect to design and documentation tools, we developed the notion of a document building toolkit. This designates a set of design specific aids that complement those offered by a generic hypermedia authoring toolkit. These generate or maintain updated parts of a project's documentation from the design data and metadata. For instance, one tool could generate a project's report with a structure reflecting the design hierarchy. Another tool could be used to retrieve the design history and append it to a section of an active document for the project.

We also proposed an innovative interface to an electronic design notebook. This eliminates the separation between authoring and reading modes that are prevalent in other electronic notebooks and other more general hypermedia authoring tools. We found that this separation is the main obstacle to simultaneous authoring and browsing, one of the designers's most basic requirements.

The conceptual model for representing active documents in the notebook extends a basic hypertext model with procedural attachments to document's objects. These are automatically activated when the documents' objects are rendered or modified. These procedures are used to maintain contexts between design and documentation data. Our experience shows that this model provides a simple yet powerful mechanism for maintaining consistency between the various pieces of information being manipulated.

## **9.2 Notes and Recommendations for a Future Experiment**

A research project is never completed, and ours is no exception. Our usability experiment was limited in the number of users, size and duration of the projects. It did not produce

definitive proof of the advantage of using an integrated design and documentation methodology. That could be achieved only with a much larger and longer study involving a more advanced prototype.

Obtaining strong evidence on the advantages of our approach would require conducting an experiment without the limitations we faced:

1. Lack of a real design environment. A large VLSI project is developed by a much larger design team and involves communication via documents with many people, such as the manufacturing team and system integrators.
2. Contrary to real VLSI design projects, the documentation for the course projects did not have to be maintained for a long period. They are never looked at again!
3. The scale of the projects undertaken (a few gates only) was smaller by orders of magnitude than a typical VLSI project.
4. The number of designers involved was too small to produce statistically significant data.

These constraints make a class setting inappropriate for this kind of study. Nevertheless, a research university environment like ours still provides unmatched conditions for early testing of a prototype system introducing a new paradigm. Two final recommendations, for those who wish to conduct similar experiments in the future:

1. Voluntary participants do not work for free. They only collaborate in the gathering of observation data when they perceive a personal benefit in their participation. It is hard to obtain comparative data from design projects not using the same approach. It is better to assume from the beginning that no comparative data will be available. Given that they all have different schedules, it is virtually impossible to find extra hours to meet with all the students participating in the experiment. Do not plan activities in the final third of the semester. They will be too busy as the final deadlines approach and the experiment inevitably gets the lowest priority.

2. Make sure that the goals set for the class projects by the instructor are compatible with the goals of the experiment. For instance, in our experiment, the goal set by the instructor for the first project was to reach the best possible point in the design space. Design documentation was of little importance. As a result, there was no motivation for high quality documentation, and students didn't document as they would in an industrial design. For the second project, the quality of the project documentation was given much more importance in the grading criteria. This is illustrated by the project cover sheets shown on Figure 9.1.

### **9.3 New Directions**

Many of the architectural concepts for design environments advanced in this dissertation have not yet been implemented. Others were implemented in the existing prototype system, but never extensively tested. We summarize some of them here, as the basis for directions for future research.

#### **9.3.1 Message Oriented Design Management Systems**

The Henry System represents only an initial contribution to the grand vision for a new generation of design environments (outlined in Section 4.3 on page 63). More work is necessary to demonstrate convincingly the advantages of using a complete, integrated, information centric design environment, where the tools in multiple heterogeneous frameworks can be accessed from a single documentation-based user interface.

One major missing piece is a design flow management system controlled by the users through active documents. Such a system could also produce information about the status of the design in the form of active documents.

Existing frameworks' process management systems are designed to control the execution of sequences of batch tools that read and write into files. With the introduction of new operating system environments, these will evolve into systems that interact with OS ses-

**EECS 141: Design Project #1 - Phase II**  
Due 3 PM Friday, October 29

(Partner 1)  
(Partner 2)

Parameter	Team Data	Partner 1	Partner 2
Design Style	_____	_____	_____
Number of Transistors	_____	_____	_____
Number of Stages	_____	_____	_____
Worst Case Path for $t_p$	_____	_____	_____
$t_{HL}$ Worst Case	_____	_____	_____
$t_{LH}$ Worst Case	_____	_____	_____
Area	_____	_____	_____
Area $\times t_p^2$	_____	_____	_____

**Grading Scheme**

Part	Possible	Partner 1	Partner 2
Initial Implementation	25	_____	_____
Verilog Logic	_____	_____	_____
Verilog SPOKE Delay	_____	_____	_____
Power Evaluation	_____	_____	_____
Design Tuning	20	_____	_____
Creative Ideas	_____	_____	_____
Systematic Trade-off Studies	_____	_____	_____
Performance Metrics	_____	_____	_____
<b>TOTAL</b>	<b>55</b>	_____	_____

**EECS 141: Design Project #2**  
Due 3 PM Monday, November 22

(Partner 1)  
(Partner 2)

Parameter	Value	Units
Number of Stages	15	_____
Number Identical Stages	16	_____
$[W/L]_{M1} + [W/L]_{M2}$ Largest	90	_____
$t_{HL}$	39.5	ns
$t_{LH}$	40.2	ns
$t_{HL}$	39.5	ns
$E_{LH}$	9.4	pJ
$E_{HL}$	10.5	pJ
$E_{AVE}$	10.85	pJ
$E_{AVE} \times [W/L]_{M1} + [W/L]_{M2}$ Largest	324.45	pJ

**Grading Scheme**

Part	Possible	Score
Organization	25	_____
Conciseness	25	_____
Completeness	25	_____
Performance Metrics	25	_____
<b>TOTAL</b>	<b>100</b>	_____

Figure 9.1 Project Summary Sheets

These sheets had to be filled-out by the students. Instructors used them to grade the projects and they gave the students an idea of the relative importance of the various parts of the design. The first project was designed to be “self-graded.” The product

$A \times t_p^2$  obtained by each student was by far the most important parcel of the final grade.

On the second project, the instructors have put more much more emphasis on the clarity, conciseness and presentation of the design.

sion managers that invoke design tools, and monitor and activate their executions through sequences of messages.

In an environment with multiple frameworks, active documents should be able to send commands to the individual design management systems to work as a common front-ends. Electronic systems designers should not need to deal with the details of interfacing with

diverse design management systems to create effective active documents describing the entire design process. These documents should interface with a new kind of information manager, controlling the flow of design data between the various frameworks used in the design environment. The information manager would not control the design data and process flow within each framework directly. It only orchestrates the transfer of design data and control flows between them.

### 9.3.2 Design Space Exploration Tools

Historically, the introduction of new data types into VLSI design environments lead to the development of “spot” tools that later become essential to designers. We believe that the same will happen with documentation (see Table 9.1).

<b>new computer-based representations...</b>	<b>... spawn new tools</b>
mask layout	layout DRCs, compactors, editors...
netlist	electrical and logic simulators, logic synthesis, ERCs, editors,...
behavior	behavioral simulators, behavioral synthesis, hardware/software co-design
history	trace management, design estimators
documentation	document consistency checkers, documentation generators

Table 9.1 New Design Data Representation Formats Spawn New Design Tools

Historically, the introduction of new computer-based representation formats into design environments was followed by the development of new tools that manipulate, verify and synthesize the design from the new descriptions. By integrating documentation into the design environment, we are also creating the opportunity to develop a new generation of tools that will become essential to designers.

The classes of tools for integrated design and documentation we enumerated in Chapter 6 are just a sample of the new possibilities. These possibilities include:

- Automatically generating human-readable documents.
- Searching documentation for design-related information and making architectural design decisions based on the information found.
- Check consistency between data and documentation.

### 9.3.3 Use of New Devices to Manipulate the Design Information

The electronic notebook we developed for Henry runs on UNIX workstations. Thus, it cannot replace paper-based engineering notebooks designers use today. Its limitations are derived fundamentally from not being portable and having a poorer user interface for adding information (see Table 9.2). However, the electronic design notebook does have some

<b>Electronic Notebook</b>		<b>Paper Notebook</b>	
-	not portable	+	extremely portable
-	difficult to use	+	very easy to use
+	massive amounts of data can be annotated	-	small amounts of manually entered data
+	sharable, reusable	-	unsharable, not reusable
+	multi-user, distributed	-	single-user

Table 9.2 Electronic Notebooks versus Paper-based Notebooks

The paper notebooks's strongest limitations lie precisely where electronic notebooks are at advantage and vice-versa. However, electronic notebook's current limitations are likely to be overcome in the next 5-10 years.

important advantages. It is not limited in the same ways as its paper-based ancestor. An electronic book can support multiple users, be distributed across a network, and provide

multimedia and hypertext linking. It can be used by groups of designers to share, exchange, and understand information about designs.

Advances in low-power wireless portable terminals will in the long run make these devices attractive to designers. One of these prototypes is being developed by the Infopad project at Berkeley [Shen92]. This project is also researching the use of this terminal to provide an electronic design notebook, based on the Henry Architecture. This capability will be used as the proof-of-concept demonstrator for the entire project.

Adoption of electronic design notebooks in industrial environments will also depend on legal issues. Currently, engineering notebooks are notarized, to prove first-to-invent claims in intellectual property litigation. An electronic mechanism equivalent to this notarization must also be developed before the use of electronic notebooks is widely embraced by VLSI designers.

*Liveboards*, electronic devices that replace *chalkboards*, are examples of other devices that are candidates for integration into VLSI design environments. In our view, we will evolve from the networks of workstations we use today into *ubiquitous computing* environments, where designers could choose from many devices to interact with the design environment [Weis91].

### **9.3.4 Use of Video in VLSI Design**

When we started this research, our initial motivation was to create structured multimedia documentation for VLSI designs, including video presentations. Project meetings would be videotaped. Previous research on the use of video in related design domains indicates that indexing and creation of mechanisms for quickly locating and displaying the relevant video materials are essential, or they are never reused. We would like to experiment with the idea of having the video on-line and indexed. Indexes would be used as anchors for links to and from the design tools. For instance, the layout of a cache module of a CPU could have a sequence of links to the location in the video material of presentations or dis-



cussions about the design. This would constitute an almost effortless process of capturing decisions. However, the software necessary for making this integration possible is currently lacking. To study the usability of video in VLSI design, we would need at least:

- Software for indexing the video material. We would like to be able to use voice recognition software to detect project-related words in the video material and use them as link anchors.
- Software for archiving, transmitting, and presenting the video information.
- An integrated set of design and documentation tools that supported the notion of activating hypermedia links.

These capabilities are not yet in a state that would enable the construction of a prototype for studying the applicability of video to VLSI design. Facing this situation, we focused on the development of a software infrastructure that would enable the use of design tools as part of an hypermedia system. That resulted in the work presented in this dissertation.

However, a study of the use of video in design environments could now be initiated. We now have access to software for voice recognition, video capture and multicast over the Internet, and multimedia video-on-demand servers. These could be used to build databases for the project-related video information that would be accessed from active documents.

## **9.4 Final Words**

When this project started, the idea of creating active documents was unknown to VLSI designers. The introduction of Mosaic in 1993 and the rapid growth of the Internet helped to make the concept of a wide-area hypermedia network one that is now commonly accepted. Active documents provide the interface paradigm that makes it possible for remote programs to produce new active documents as the result. In the past year alone, we grew from a few active documents, in a few experimental systems in research projects around the world, into millions of active documents used daily by millions of users. Our

research in developing a design system centered on active documents is of interest to system integrators in many other domains.

The explosive growth of the World Wide Web and new services supported in electronic commerce creates synergies for introducing other kinds of new services for designers as well as CAD systems that provide seamless interfaces to access them. Services will cover the entire system development process, from marketing to distribution, information gathering to design, and from simulation and prototyping to manufacture. Design will become the process of creating a web of links between these services.

Creating such infrastructure will be a large effort in terms of definition of communication protocols between information systems from independent engineering and business domains. However, we believe this would be a profitable investment for our society. The new infrastructure will enable collaboration between those who have large and highly automated design and manufacture systems and those with creative ideas but limited resources. Some day, anyone with a brilliant idea will have the possibility to marshal the resources necessary for producing a new artifact using the information web. The most sophisticated design tools, computational prototyping facilities, manufacturing technologies and the help of the best consultants will be available at virtually no initial cost. Physical proximity to critical resources or financial power will not be the main cause for differentiation between societies. For system integration industries, competitive advantages will be fundamentally built on the capabilities for defining new solutions and quickly orchestrating the flow of information between the resources required to produce them. This is the vision that inspired the development of the Henry System. To us, this research represents the start of a long endeavor to make this vision a reality. We hope to have inspired our readers to join us in the effort.

# References

- [Andr89] J. André, R. Furuta, and V. Quint, editors. *Structured Documents*, volume 2 of *The Cambridge Series on Electronic Publishing*. Cambridge University Press, 1989.
- [Andr93] Marc Andreessen. NCSA Mosaic Technical Summary. Technical report, National Center for Supercomputing Applications, 605 E. Springfield, Champaign, IL 61820, May 93.
- [App93] Apple Computer. *OpenDoc Technical Summary*, October 1993. Version 1.0.
- [ATL94] ATLIS Consulting Group, Inc., 6011 Executive Boulevard, Rockville, MD 20852, USA. *Pinnacles Component Information Standard*, March 1994.
- [Bahr92] Homa Bahrami. The Emerging Flexible Organization: Perspectives from Silicon Valley. *California Management Review*, 34(4):33–52, 1992. reprinted in *IEEE Engineering Management Review*, Vol 21:4, Winter 1993, pp 94–103.

- [Barn92] Timothy J. Barnes. *Electronic CAD Frameworks*. Kluwer Academic Publishers, 1992.
- [Birr84] A. D. Birrel and B. J. Nelson. Implementing Remote Procedure Calls. *ACM Transactions in Computer Systems*, 2(1):39–59, February 1984.
- [BL93a] Tim Berners-Lee. Uniform Resource Locators (URL) A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network. Technical report, CERN, October 1993. Internet Draft.
- [BL93b] Tim Berners-Lee and Daniel Connolly. Hypertext Markup Language – A Representation of Textual Information and Metainformation for Retrieval and Interchange. Technical report, CERN, 1993. Internet Draft, available from <http://info.cern.ch/hypertext/WWW/MarkUp/HTML.html>.
- [BL94a] Tim Berners-Lee. Hypertext Transfer Protocol (HTTP). Technical report, CERN, 1994. Internet Draft.
- [BL94b] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The World Wide Web. *Communications of the ACM*, 37(8):76–82, August 1994.
- [Bore92] Nathaniel S. Borenstein. Computational Mail as Network Infrastructure for Computer-Supported Cooperative Work. In *CSCW'92 Proceedings*, pages 67–73, November 1992.
- [Bore93] N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. Technical report, Bellcore, Innosoft, September 1993. Internet RFC 1521.
- [Bore94] Nathaniel Borenstein. Email With a Mind of its Own: The Safe-Tcl Language for Enabled Mail. Submitted to Proceedings of ULPAA'94, 1994.

- [Brad91] Stephen R. Bradley and Alice M. Agogino. Design Capture and Information Management for Concurrent Design. *International Journal of Systems Automation: Research and Applications (SARA)*, 1:117–141, 1991.
- [Bray87] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS: Multiple-Level Logic Optimization System. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, pages 1062–1081, November 1987.
- [Brow94] Patrick W. Brown. Digital Signatures: Are They Legal for Electronic Commerce? *IEEE Communications*, 32(9):76–80, September 1994.
- [BSB90] L. A. Rowe B. S. Becker. HIP: a Hypermedia Extension of the Picasso Application Framework. Technical Report UCB/ERL M90/21, University of California, Berkeley, December 1990.
- [Bush89] M. Bushnell and S. W. Director. Automated Design Tool Execution in the Ulysses Design Environment. *IEEE Transactions on CAD*, 8(3):279–287, 1989.
- [CAD93] CAD Framework Initiative, Inc., 4030 W. Braker Lane, Suite 550, Austin TX 78759. *FrameWork Architecture Reference*, version 1.2 edition, February 1993.
- [CAD94] CAD Framework Initiative, Inc., 4030 W. Braker Lane, Suite 550, Austin TX 78759. *CFI Architecture Revision*, version 0.06 edition, March 1994.
- [Carr90] John M. Carrol. *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. Technical Communication (M.I.T. Press). MIT Press, Cambridge, MA, 1990.
- [Carr91] John M. Carrol and Thomas P. Moran. Introduction to This Special Issue on Design Rationale. *Human-Computer Interaction*, 6:197–200, 1991.

- [Cart92] Donald E. Carter and Barbara Stilwell Baker. *CE Concurrent Engineering: The Product Development Environment for the 1990s*. Addison-Wesley Pub. Co., 1992.
- [Caso91] Andrea Casotto. *Automatic Design Management Using Traces*. PhD thesis, University of California, Berkeley, March 1991. Technical report UCB/ERL M91/22.
- [Chiu92] Tzicker Chiueh. *Papyrus: A History-Based VLSI Design Process Management System*. PhD thesis, University of California, Berkeley, November 1992. Technical report UCB/CSD-93-724.
- [Chok94] Santoshi Chokani. Toward a National Public Key Infrastructure. *IEEE Communications*, 32(9), September 1994.
- [Come91] Douglas E. Comer. *Internetworking with TCP/IP – Principles, Protocols and Architecture*, volume I. Prentice Hall, second edition, 1991.
- [Come93] Douglas E. Comer and David L. Stevens. *Internetworking with TCP/IP – Client-Server Programming and Applications*, volume III. Prentice Hall, 1993.
- [Conc91] E. Jeffrey Concklin. A process-oriented approach to design rationale. *Human-Computer Interaction*, 6:357–391, 1991.
- [Conk87] J. Conklin. Hypertext: An Introduction and Survey. *Computer Magazine*, 20(9):17–41, sep 1987.
- [Conk88] J. Conklin and M. L. Begemann. gIBIS: a Hypertext Tool for Argumentation. *ACM Transactions on Office Information Systems*, 6(4):303–331, October 1988.

- [Corb91] John R. Corbin. *The Art of Distributed Applications : Programming Techniques for Remote Procedure Calls*. Springer-Verlag, 1991.
- [Croc82] David H. Crocker. Standard for ARPA Internet Text Messages. Technical report, University of Delaware, August 1982. RFC 822.
- [CWE91] Jr. Charles W. Ennis and Steven W. Gyeszly. Protocol Analysis of the Engineering Systems Design Process. *Research in Engineering Design*, 1991(3):15–22, 1991.
- [Dani89] J. Daniell and S. Director. An Object-oriented Approach to Distributed CAD Tool Control. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 197–202, 1989.
- [Davi92] William H. Davidow and Michael S Malone. *The Virtual Corporation: Structuring and Revitalizing the Corporation for the 21st Century*. Harper Collins Publishers, New York, 1992.
- [Davi92] Hugh Davis, Wend Hall, Ian Heath, and Gary Hill. Towards an Integrated Information Environment With Open Hypermedia Systems. In *Proceedings of the Fourth ACM Conference on Hypertext*, pages 181–190, Milan, Italy, November 92.
- [Deli86] N. Delisle and M. Schwartz. Neptune: a Hypertext System for CAD Applications. In *Proceedings of ACM SIGMOD’86*, pages 132–142, Washington, DC, May 1986.
- [Delm93] Sven Delmas. XF – Design and Implementation of a Programming Environment for Interactive Construction of Graphical User Interfaces. Technical report, Technische Universit at Berlin, 1993.
- [Etzi94] Oren Etzioni and Daniel Weld. A Softbot-Based Interface to the Internet. *Communications of the ACM*, 37(7):72–76, jul 1994.

- [Fisc91] Gerhard Fischer, Andreas C. Lemke, Raymond McCall, and Anders I. Morch. Making Argumentation Serve Design. *Human-Computer Interaction*, pages 393–419, 1991.
- [Foun90] Andrew M. Fountain, Wendy Hall, Ian Heath, and Hugh C. Davis. MICROCOSM: An Open Model for Hypermedia With Dynamic Linking. In *Hypertext: Concepts Systems and Applications / Proceedings of the First European Conference on Hypertext*, pages 298–311. Cambridge University Press, 1990.
- [Fra] Frame Technology Corporation, 1010 Rincon Circle, San Jose, CA 95131. *Integrating Applications with FrameMaker*.
- [fS86] International Organization for Standardization. *Information Processing, Text and Office Systems, Standard Generalized Markup Language (SGML)*. International standard 8879. International Organization for Standardization, Geneva, Switzerland, 1st edition, 1986.
- [Gajs83] D. D. Gajski and R. H. Kuhn. New VLSI Tools. *IEEE Computer Magazine*, 16(12):11–14, December 1983.
- [Garc92] Ana Cristina Bicharra Garcia. *Active Design Documents: A New Approach for Supporting Documentation in Preliminary Routine Design*. PhD thesis, Stanford University, 1992.
- [Glic93] Jay Glicksman and Vinay Kumar. A SHARed Collaborative Environment for Mechanical Engineers. In *Proceedings of Groupware'93*, 1993.
- [Gold90] Charles F. Goldfarb. *The SGML handbook*. Oxford University Press, 1990.
- [Gold92] Yaron Goldberg, Marilyn Safran, and Ehud Shapiro. Active Mail – A Framework for Implementing Groupware. In *CSCW'92 Proceedings*, pages 75–83, November 1992.



- [Gorr91] G. A. Gorry, K. B. Long, A. M. Burger, C. P. Jung, and B. D. Meyer. The Virtual Notebook System: An Architecture for Collaborative Work. *Journal of Organizational Computing*, 1(13):233–250, 1991.
- [Gray91] T. W. Gray and J. Glynn. *Exploring Mathematics with Mathematica*. Addison-Wesley Pub. Co., 1991.
- [Grou88] Network Working Group. RPC: Remote Procedure Call Protocol Specification Version 2. Technical report, Sun Microsystems, Inc., June 1988. RFC 1057.
- [Grud91] Jonathan Grudin. CSCW Introduction. *Communications of the ACM*, 34(12):30–34, December 1991.
- [Grud94] Jonathan Grudin. Computer Supported Cooperative Work: History and Focus. *IEEE Computer*, 27(5):19–26, May 1994.
- [Haan92] Bernard J. Haan, Paul Kahn, Victor A. Riley, James H. Coombs, and Norman K. Meyrowitz. IRIS Hypermedia Services. *Communications of the ACM*, 35(1):36–51, January 1992.
- [Hala88] F. G. Halasz. Reflections on Notecards: Seven Issues for the Next Generation of Hypertext Systems. *Communications of the ACM*, 31(7):836–852, July 1988.
- [Hans90] Wilfred J. Hansen. Enhancing documents with embedded programs: How Ness extends insets in the Andrew Toolkit. In *Proceedings of 1990 International Conference on Computer Languages*, pages 23–32, Los Alamitos, CA, March 1990. IEEE Computer Society Press.
- [Harr86] D. S. Harrison, P. Moore, R. L. Spickelmier, and A. R. Newton. Data Management and Graphics Editing in the Berkeley Design Environment. In

*1986 IEEE International Conference on Computer-Aided Design*, pages 24–27, November 1986.

- [Harr90] D. Harrison, R. Newton, R. L. Spickelmeier, and T. Barnes. Electronic CAD Frameworks. *Proceedings of the IEEE*, 78(2):393–417, February 1990.
- [Heck91] Paul Heckel. *The Elements of Friendly Software Design*. Sybex, Alameda, CA, 1991.
- [IEE87] IEEE. *VHDL Language Reference Manual*, 1987. IEEE Standard 1076-1987.
- [Jaco92] Margarida F. Jacome and Stephen W. Director. Design Process Management for CAD Frameworks. In *Proceedings of the 29th ACM/IEEE Design Automation Conference*, pages 500–504, june 1992.
- [Kell86] Mahlon Kelly and Nicholas Spies. *FORTH, a Text and Reference*. Prentice-Hall software series. Prentice-Hall, Englewood Cliffs, NJ, 1986.
- [Kent92] S. Kent. Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management. Technical report, BBN, February 1992. RFC 1422.
- [Klei94] Sally Kleinfeldt, Michaela Guiney, Julia K. Miller, and Marina Barnes. Design Methodology Management. *Proceedings of the IEEE*, 82(2):231–250, February 1994.
- [Knut84] Donald E. Knuth. Literate Programming. *The Computer Journal*, 27(2):97–11, May 1984.
- [Lee87] William Lee. "?": A Context Sensitive Help System based on Hypertext. In *24th ACM/IEEE Design Automation Conference*, pages 429–435, 1987.

- [Libe95] Don Libes. *Exploring Expect: A Tcl-based Toolkit for Automating Interactive Programs*. O'Reilly & Associates, 1995.
- [Linn93] J. Linn. Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. Technical report, IAB IRTF PSRG, IETF PEM WG, February 1993. RFC 1421.
- [Maes94] Pattie Maes. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):31–40, 146, July 1994.
- [Malc91] Kathryn C. Malcolm, Steven E. Poltrock, and Douglas Schuler. Industrial Strength Hypermedia: Requirements for a Large Engineering Enterprise. In *Hypertext'91*, pages 13–24, San Antonio, Texas, December 1991.
- [McCa90] Raymond J. McCall, Patrick R. Bennet, Peter S. D'Orozio, Jonathan L. Ostwald, Frank M. Shipman, and Nathan F. Wallace. PHIDIAS: Integrating CAD Graphics into Dynamic Hypertext. In *Hypertext: Concepts Systems and Applications / Proceedings of the First European Conference on Hypertext*, pages 153–165, INRIA, France, 1990. Cambridge University Press.
- [McCo94] Rob McCool. The Common Gateway Interface Specification. Technical report, National Center for Supercomputer Applications, 1994. URL <http://hoohoo.ncsa.uiuc.edu/cgi/interface.html>.
- [Meyr86] Norman Meyrowitz. Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework. In *OOPSLA'86 Proceedings*, pages 186–200, September 1986.
- [Mic93] Microsoft Corporation. *OLE 2.0 Design Specification*, April 1993.
- [Mins86] Marvin Minsky. *The Society of Mind*. Simon and Schuster, 1986.

- [Nati90] National Institute of Standards and Technology and European Computer Manufactures Association. *Reference Model for FrameWorks of Software Engineering Environments*. NIST special publication. U. S. Dept. of Commerce, National Institute of Standards and Technology, Gaithersburg, MD, 2nd. edition, 1990.
- [Nech93] Anna-Lena Neches. FAST - A Research Project in Electronic Commerce. *Electronic Markets*, October 1993. URL <http://info.broker.isi.edu/9/fast/articles/em-oct93.ps>.
- [Newt81] A. R. Newton, D. O. Pederson, A. L. Sangiovanni-Vincentelli, and C. H. Séquin. Design Aids for VLSI: the Berkeley Perspective. *IEEE Transactions on Circuits and Systems*, 28(7):666–680, July 1981.
- [Niel90] Jakob Nielsen. *Hypertext and Hypermedia*. Academic Press, Boston, 1990.
- [Niel92] Jakob Nielsen. The Usability Engineering Life Cicle. *IEEE Computer*, 25(3):12–22, March 1992.
- [Norm86] Donald A. Norman. Cognitive engineering. In Donald A. Norman and Stephen W. Draper, editors, *User Centered System Design: New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates, 1986.
- [oS90] National Institute of Standards and Technology. *X Window System : Version 11, Release 3*. Federal Information Processing Standards Publication. National Institute of Standards and Technology, 1990.
- [oS91a] National Institute of Standards and Technology (NIST). *Electronic Data Interchange (EDI)*, volume 161 of *Federal Information Processing Standards Publication*. National Institute of Standards and Technology, 1991.

- [oS91b] National Institute of Standards and Technology (NIST). *Proposed Digital Signature Standard (DSS)*. Federal Information Processing Standards Publication. National Institute of Standards and Technology, sep 1991.
- [Oust85] J. K. Ousterhout, G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor. The Magic VLSI Layout System. *IEEE Design & Test of Computers*, pages 19–30, 1985.
- [Oust94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.
- [Pear89] Amy Pearl. Sun’s Link Service: A Protocol for Open Linking. In *Hypertext’89 Proceedings*, pages 137–146, Pittsburgh, Pennsylvania, November 1989.
- [Post82] Jonathan B. Postel. Simple Mail Transfer Protocol. Technical report, Information Sciences Institutien, August 1982. RFC 821.
- [Putt90] J. J. Puttress and N. M. Guimaraes. The Toolkit Approach to Hypermedia. In *Hypertext: Concepts Systems and Applications / Proceedings of the First European Conference on Hypertext*, pages 24–37, INRIA, France, 1990. Cambridge University Press.
- [Raba95] Jan M. Rabaey. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, 1995. to appear.
- [Rett91] Marc Rettig. Nobody Reads Documentation. *Communications of the ACM*, 34(7):19–24, July 1991.
- [Riec94] Doug Riecken. Intelligent agents. *Communications of the ACM*, 37(7):19–21, 1994.

- [Ritt70] W. Rittel and W. Kunz. Issues as Elements of Information Systems. Working Paper 131, Center for Planning and Development Research, University of California, Berkeley, 1970.
- [Rose85] M. T. Rose and J. L. Romine. *The Rand MH Message Handling System: User's Manual*. Department of Information and Computer Science, University of California, Irvine, January 1985.
- [Rusu94] Stefan Rusu. Building a Multi-vendor Integrated Environment for Custom VLSI Design. In *Proceedings of the 1994 Electronic CAD Interoperability and Integration Conference, EII'94*, pages 7–11, Oakland, CA, May 1994. CAD Framework Initiative, 4030 W. Braker Lane, Suite 550, Austin, TX 78759.
- [Scal94] Todd J. Scallan. Concurrent Design through CFI-based Interoperability. In *Proceedings of the 1994 Electronic CAD Interoperability and Integration Conference, EII'94*, pages 43–52, Oakland, CA, May 1994. CAD Framework Initiative, 4030 W. Braker Lane, Suite 550, Austin, TX 78759.
- [Sche92] Robert W. Scheifler and James Gettys. *X Window System: The Complete Reference to Xlib, X Protocol, ICCCM, XLFD*. Digital Press X and Motif Series. Digital Press, Bedford, MA, 3rd. edition, 1992.
- [Schn82] Ben Schneiderman. The Future of Interactive Systems and the Emergence of Direct Manipulation. *Behavior and Information Technology*, 1:237–256, 1982.
- [Sé83] Carlo H. Séquin. Managing VLSI Complexity: an Outlook. *Proceedings of the IEEE*, 71(1), January 1983.
- [Shen92] S. Sheng, A. Chandrakasan, and R. Brodersen. A Portable Multimedia Terminal. *IEEE Communications Magazine*, December 1992.

- [Sher90] M. Sherman, W. J. Hansen, M.I. McInerny, and T. Neuendorffer. Building Hypertext on a Multimedia Toolkit: An Overview of Andrew Toolkit Hypermedia Facilities. In *Hypertext: Concepts Systems and Applications / Proceedings of the First European Conference on Hypertext*, pages 13–24, INRIA, France, 1990. Cambridge University Press.
- [Smit93] Brian Smith and Larry Rowe. Tcl Distributed Programming. In *Tcl/Tk Workshop*, pages 50–51, Berkeley, CA, June 1993.
- [SunS92] SunSoft. The ToolTalk Service. Technical report, SunSoft, Inc., SunSoft, Inc, 2550 Garcia Avenue, Mountain View, CA 94043, October 1992.
- [Taft90] Ed Taft and Jeff Walden. *PostScript Language Reference Manual / Adobe Systems Incorporated*. Addison-Wesley, Reading, MA, 2nd edition, 1990.
- [Terr90] Douglas B. Terry and Donald G. Baker. Active Tioga Documents: An exploration of Two Paradigms. *Electronic Publishing – Origination, Dissemination and Design*, 7(2):105–122, May 1990.
- [Toye93] George Toye, Mark R. Cutkosky, Larry J. Leifer, J. Marty Tenenbaum, and Jay Glicksman. SHARE: A Methodology and Environment for Collaborative Product Development. In *Second Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1993.
- [Uej90] Wayne H. Uejio. An Electronic Design Notebook for the DARPA Initiative in Concurrent Engineering. In *Proceedings of the CERC on Concurrent Engineering*, 1990.
- [Weis91] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, September 1991.
- [Wolf91] Stephen Wolfram. *Mathematica: a System for Doing Mathematics by Computer*. Addison Wesley, 2nd edition, 1991.

- [Yake90] K.C. Burgess Yakemovic and E. Jeffrey Conklin. Report on a Development Project Use of an Issue-Based Information System. In *CWCW'90 Proceedings*, pages 105–118, October 1990.
- [Yank88] Nicole Yankelovich, Bernard J. Haan, Norman K. Meyrowitz, and Steven M. Drucker. The Concept and the Construction of a Seamless Information Environment. *IEEE Computer*, 21:81–96, January 1988.