

# A Dynamic Design Estimation and Exploration Environment

Ole Bentz\*, Jan M. Rabaey, and David Lidsky

Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, 94720

\* Silicon Graphics, Inc., Mountain View, CA, 94043

## Abstract

*The rapid increase in the complexity of systems demands new approaches to design exploration and trade-off analysis. This paper presents an exploration environment that provides a uniform way to perform exploration at the conceptual (pre-specification) stages of design. The environment encapsulates knowledge about how to perform estimations in different application domains, and separates a designer from the mechanics of the estimation process.*

## 1. Introduction

As the complexity of VLSI designs grows it becomes increasingly important to explore design alternatives and compare trade-offs very early in the design process. Decisions made during the conceptual stages of design have the greatest potential impact. Unfortunately, early decisions are often based on estimations that are gathered in ad-hoc ways from disparate sources including data sheets, empirical models, estimators, experience, etc. There are no generalized estimation techniques that apply to all areas of expertise, and even within a given area there are often several techniques that can be used under different circumstances.

From a designer's perspective, performing estimations is simply a means to obtain design related information. However, the heterogeneous nature of estimations, and the time consuming task of finding and applying estimation techniques, prohibits extensive exploration. It is clear that aids must be provided to facilitate design estimation and exploration.

This paper presents a new design environment that provides the necessary features to enable design exploration in a consistent fashion. The environment fills the gap between a designer and a heterogeneous set of estimator tools, models, databases, etc. (figure 1). In this paper we broadly use the word "estimations" to refer to any result obtained from design tools, parameterized models, database look-ups, etc., and the word "estimators" to refer to techniques for obtaining estimations.

The design environment allows the use of intuitive commands, such that a designer doesn't need to know how to manipulate all available estimators. For example, the command for getting an estimate of the power consumption of a chip is "power". There are

many techniques that can be used to obtain estimates of power, so the design environment has to dynamically resolve which technique to use, based on all given parameters and constraints. The more constraints a designer gives, the more accurate or detailed the estimation can be, limited only by the available estimators.

The observation that estimations generally are limited to a specific knowledge domain inspired the organization of estimation techniques, and all related parameters, constraints, etc., into entities called "domains".

**Definition:** A "domain" is an encapsulation of knowledge regarding a specific design-related area of expertise.

The knowledge which is captured in a domain includes parameters, constraints, tool encapsulations, file type definitions, etc. Domains are discussed in detail in section 3.1. A domain is intended to capture one dimension in the design space (e.g. architecture, cell library, technology, etc.), and domains can derive from other domains to capture increasingly confined portions of a particular dimension.

By bringing together orthogonal domains, (i.e. domains that represent different dimensions in the design space) we can create a "context" which contains knowledge in the dimensions represented by the domains. This knowledge is as general or specific as the domains define.

**Definition:** A "context" is a union of domains.

Each design, or portion of a hierarchical design, has an associated context. The context can either be chosen by the designer, or it can be automatically chosen by the design environment, based on what is known about the design (parameters, constraints, properties, etc.). Intuitive commands are interpreted within a context, i.e. it is resolved what the meaning of a command is based on the knowledge contained in the union of domains.

**Example:** To illustrate how intuitive commands are interpreted, consider the nine domains shown in figure 2. The six domains on the bottom are derived from the more general domains on top. The four sample contexts show by example how the command "area" is interpreted based on which domains are in the context.

The presented environment offers ubiquitous access through the world wide web (WWW) to allow designers to use the environment remotely, for example from a portable terminal. The environment has a flexible application programming interface that allows a variety of external user interfaces to be used. In addition, since information resources and design data are becoming increasingly distributed across both the internet and company intranets, future versions of our design environment will be able to communicate with each other to share and reuse information and knowledge.

Previous efforts in this area includes the work presented as "Clio", which offers design assistance within the Odyssey CAD framework [1][2][3]. In Odyssey, knowledge is also organized in domains, but different domain hierarchies are not "orthogonal", i.e. representing different aspects of design. Thus, a context is represented by one domain, instead of a union of domains, and doesn't offer the inherent flexibility of being able to manipulate which domains are included in a context. Another notable effort in the area of design flow management that has influenced our envi-

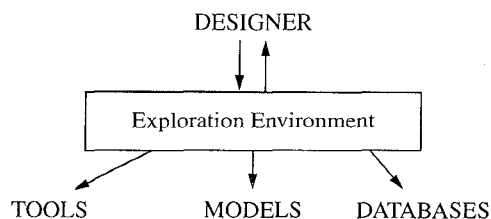


Figure 1. Exploration Environment.

"Permission to make digital/hard copy of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and /or a fee."

DAC 97, Anaheim, California

(c) 1997 ACM 0-89791-920-3/97/06 ..\$3.50

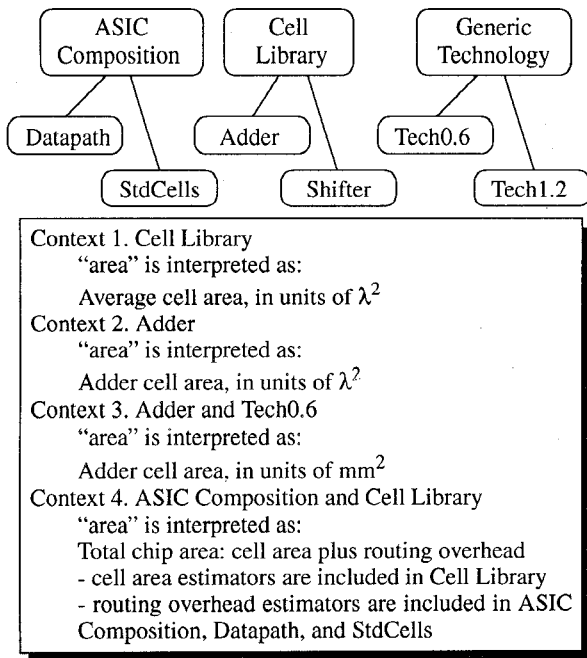


Figure 2. Sample Domains and Contexts.

environment is the NELSIS CAD Framework [4][5]. A broad survey of CAD frameworks can be found in [6].

The rest of this paper is organized as follows. Chapter 2 presents the system architecture, and chapter 3 describes the components of the system in detail. In chapter 4 we discuss user interface issues, and in chapter 5 we show a design example using our framework. Finally, chapter 6 concludes the paper and outlines future directions for this work.

## 2. System Architecture

The architecture of our design environment is shown in figure 3. The environment waits for a user to make a request, which consists of one or more words. The words can be intuitive commands, or names of parameters, constraints, or domains. The words in the request are used by a search engine that traverses all domain knowledge, and finds the domains that best match the words given in the request. A context is automatically assembled from the domains with the most matches, and it is made sure that the chosen domains are compatible. The context is proposed to the designer, who can alter it if desired. If a specific question is given in the request, it is interpreted within the context, i.e. based on the

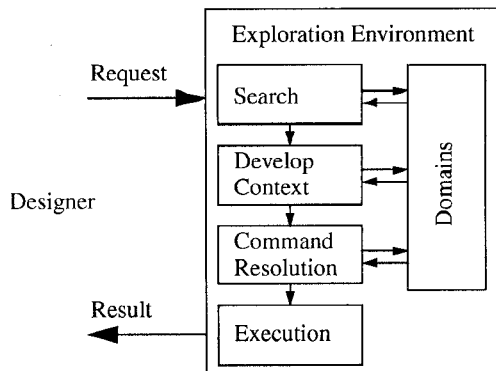


Figure 3. System Architecture.

knowledge contained in the domains in the context. The context limits the possible meaning of the request, so the design environment can more easily resolve which technique, contained in a script, will satisfy the user's request. The chosen script is then executed, and the result is returned to the user.

The importance of the context can not be overstated. A context is the environment in which commands are interpreted, and as such it is the basis of the aid that can be offered to designers. With a large collection of domains it is clearly not practical for a designer to always manually find and select the right domains. On the other hand, while the environment can offer the help of a search facility, a task as important as selecting domains for a context should not be blindly left up to the design environment. The best approach is to allow for a balance of the two approaches, allowing the user to interact with the search results whenever desired.

Within a context there can be many different techniques for satisfying a command. Techniques may offer different levels of accuracy, or they may operate on different types of design data. The design environment attempts to resolve which of the techniques to use based on which types of design files, if any, have been identified. If there are two or more techniques that appear to satisfy the request equally well, the user is asked to resolve the ambiguity.

## 3. System Components

In this chapter, the three main components of the design environment, namely domains, contexts, and the search engine, are described in detail. Our system is written in a version of the Tool Command Language (TCL) [7] that supports object oriented programming (incr-Tcl) [8], TCP/IP communications (Tcl-DP), and interactive program control (Expect) [9].

### 3.1. Domains

To reflect the view that design takes place in a multi-dimensional space, we organize knowledge about orthogonal aspects of design (e.g. architecture, cell library, or fabrication technology) into separate entities called "domains". Domains can be derived from other domains in an object oriented fashion, and each derived domain adds more and more constraints. Figure 2 (above) showed a simplified view of three different families of domains. Figure 4 shows another family or tree of domains that are derived from a general DSP domain.

Domains contain knowledge in these categories:

**identifiers** - a list of descriptive words that characterize the contents of a domain. Used by the search engine to find relevant domains.

**associated domains** - a list of domains that need to be included in any context that the current domain is in. Each listed domain, or a

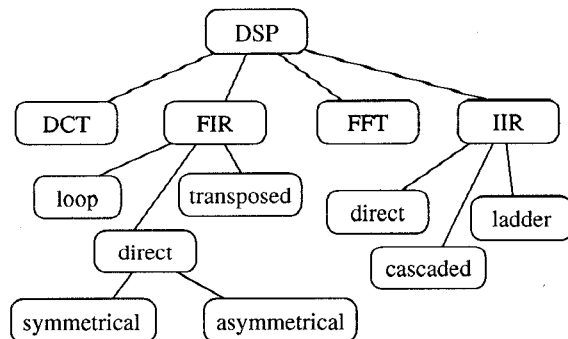


Figure 4. A tree of DSP domains.

domain derived from it, must be included in a context.

**parameters** - a list of common parameters and default values. For example, an FIR filter domain has a parameter "NrOfTaps", and a technology domain has a parameter "lambda" (minimum feature size).

**limits** - a list of typical limits within a domain, either on parameters or on design features. For example, a parameter can be limited to a range of values, or limits can be specified for the area or power of a chip. If a limit is exceeded, for example determined by an estimation, the designer will be notified.

**requests** - a list of requests that can be meaningfully posed within the scope of this domain.

**tools** - encapsulations of design tools, including estimation tools. The encapsulation is similar to the TEF (tool encapsulation format) from CFI [10], but it adds two features: a list of computational resources where a tool can be executed, and an optional script for creating supporting files at run-time, such as simulation scripts/command files.

**file types** - a list of the types of files that are of interest in this domain. Each file type is given a name (e.g. "c-file"), and a way to identify files of that type is given, which can either be a name based rule (e.g., \*.c for C files) or a content based rule (e.g., "look for the string '#!/bin/csh' at offset 0" for C-shell scripts).

**scripts** - a list of scripts that define estimation techniques or "design flows". The scripts are written in the extended TCL language mentioned above. Scripts consist of a descriptive name, the name of the type of file on which the script can operate (if any), and an executable body. Scripts can call tools (using the encapsulations described above), or other scripts. There can be several scripts with the same name, in cases when different techniques can be used to perform a given task.

**translations** - a list of common translations between units. The standard prefixes (e.g., for kilo, milli, etc.) are built in, but other translations have to be given as a procedure (e.g., converting power dissipation to heat).

**stimuli** - a list of stimulus generators. The concept here is that some tools, such as simulators, require more than just design files to run. A stimulus generator is a procedure that is reusable throughout the design flow. It generates a simple list (typically {time,value} pairs) which can be translated to the specific syntax required by the various tools. The translation process is defined on a "per tool" basis in the tool's encapsulation.

Domains can be created by end users but are typically created by expert designers. Domains can be shared between designers, so the novice designer can gain access to the knowledge captured by experts by simply importing their domains. Domains could also be provided by design tool companies along with their regular software distributions, or by cell library companies along with their set of library cells.

### 3.2. Context

A designer has to put a design in context by describing the key characteristics of what is being designed. It can be as broad as "DSP" or as specific as "an ASIC chip performing the function in the file chip.vhd, using synthesis tool X, and cell library Y, implemented in a 0.6 micron technology". By bringing together domains that represent the various aspects of the description, we can create a collection of the knowledge that is relevant to a design.

It is this union of relevant domains that we call a context. A context is the working environment in which commands can be

interpreted and translated to specific scripts. A context is a dynamic entity which evolves throughout the design process. For example, a context that initially contains only a "DSP" domain can be changed to contain the more constrained "FIR" domain while the designer explores the features of an FIR version of a filter. Later, an "IIR" domain may replace the "FIR" domain in the context to enable the designer to compare the features of an IIR implementation.

In some cases it is useful to ensure that certain domains are always included in the same design context. For example, a domain that represents a cell library can have features (e.g., height) that are in units of lambda, but the value of lambda is defined in a domain that represents a specific fabrication technology. Thus, the cell library domain requires a technology domain to provide an interpretation of lambda. Normally, an associated domain should be as general as possible (e.g., a technology domain where lambda is defined as "1.0 lambda") such that, by default, minimal interpretation is made. In other cases it is useful to identify that the estimations from one domain have been made assuming the parameters or constraints that are represented by another domain. In order to make sense, in cases when a context does not include the "assumed" domain, the estimates must be translated (by scaling, inter- or extrapolation, etc.) to account for the actual parameters and constraints that are included in the context. These types of automatic translations are supported by our environment.

The mechanism for creating and changing contexts is simple and allows two different methods to be used. The overriding principle is that a designer should have full control over which domains make up the context, as long as the domains are not incompatible. In cases where a designer knows which domains are relevant, the context can be created or changed by manually adding or deleting domains. However, in many cases it may not be clear which domains are relevant, so it is necessary to have a search facility that can locate relevant domains and propose a context. The search facility will be described in more detail in the next section.

When a domain is added to a context, all the associated domains that are listed in the domain's specification are also added, with the following considerations.

**Rule a.** If the context contains a domain that is derived from the domain to be added, the new domain will not be added. This is done to preserve the most specific domains in a context.

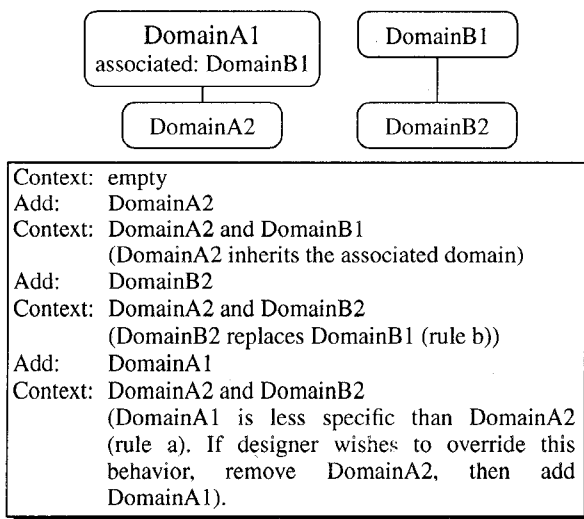
**Rule b.** If the context contains a domain that belongs to the same "family" of domains (i.e. they are derived from the same root domain) then the new domain replaces its family member. This is done to avoid having conflicting domains in the same context, such as a 1.2  $\mu\text{m}$  technology domain and a 0.6  $\mu\text{m}$  technology domain.

In hierarchical designs each design entity is allowed to have its own unique context.

**Example:** Consider the example in figure 5. DomainA1 has an associated domain, DomainB1, and this is inherited by DomainA2. Observe how the context changes when domains are added.

### 3.3. Search Engine

The goal of the search engine in our system is to locate relevant domains and propose a context, or a list of domains, that best matches a user given request. A user request consists of one or more words, and each word is used as a term to search for. Domains are considered relevant if there are items in the domain specification that matches any of the search terms.



**Figure 5.** Context Example.

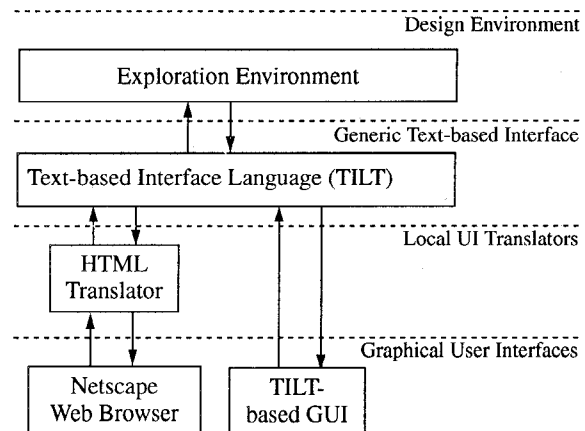
During the search process the engine looks for exact and partial matches for each of the given search terms. When all domains have been searched, four scores are assigned to each domain. The first two are the number of exact and partial matches found in the domain. The last two also represents exact and partial matches, but they are summations of matches over all the domain's associated domains. Based on the 4 scores we choose the domain with the highest score, and create a context with that domain and its associated domains. If that domain (and its associated domains) had matches for all the search terms, the search is over, and the resulting context is proposed. If the context didn't have matches for all search terms, then we choose the remaining domain with the highest score, add it to the context, etc.

In some cases it is useful to propose several contexts after performing a search. For example, a list of contexts would be required to respond to the command "show the area of all adders." Proposing a list of contexts will be a feature of a future version of the search engine.

As a refinement for the search process, we allow two features in the search terms. If a search term contains an equal sign (e.g. word=value) we take that to mean that the word has to match a parameter name and that the parameter's value has to match the given value. If a search term contains a colon and an equal sign (e.g. word:=value) we search for a parameter name that matches the word, and we make a note for later that the parameter's value should be set to the given value, if this domain ends up being used in a context.

#### 4. User Interface Issues

Our design environment uses a client-server model, in that it acts as a server that responds to requests made by a designer through a user interface client. In this model, the user interface is a separate, or disconnected, entity that is not integrated into the environment. Since the user interfaces are not built-in, we developed a generic text-based interface language for the environment (called "TILT") which can easily be translated to the requirements of specific interfaces. Currently we have an HTML translator as shown in figure 6. The benefits of the client-server model is that the user interface can be customized for many different applications (e.g., allowing the environment to be encapsulated by another design tool) and that a designer can access the environment from any-



**Figure 6.** User interfaces are external to the design environment.

where in the network.

Supporting a disconnected user interface poses a number of challenges. First of all, since the design environment can be accessed from anywhere in the World Wide Web, user identification is an important issue. Since the environment has full access to a user's file system (i.e. can read and write files), a lot of damage could be done if an intruder could get access. We have implemented an identification strategy that requires a user to give a name and a password. However, when web browsers are used to access the environment, it would be tedious to have to give a name and password for each interaction. We have augmented the HTML translator to support the "cookies" that are used by popular web browsers, such as Netscape Navigator or Microsoft Explorer. In web terminology, a cookie is a token which the server (in this case the design environment) passes to the web browser when the user has been properly identified, and which the web browser uses as a special password for successive accesses.

Another important challenge arises from the disconnected UI model, since both the user interface and the design environment carry state information. Since the environment has no control over the disconnected UI, it is possible that the UI becomes outdated with respect to the environment. The state information in the environment is basically contained in the contexts (the unions of domains). The state information in the UI is contained in a list of the available commands for the given context. Therefore, the design environment must be able to determine if a command originated from an outdated list, and notify the designer to update the user interface when necessary. This is accomplished by giving each context an identifying number which is annotated on each list of available commands.

The interface also has to provide useful information about what the design environment is doing at run-time. The environment offers two different means for that: execution traces and run-time information. Execution traces are automatically generated whenever the environment executes scripts or tools. These traces can be used as an after-the-fact way of examining what the environment did. Run-time information (e.g., decisions that are made, tools that are invoked, etc.) is a more immediate indicator of what the design environment is doing. However, in the disconnected UI model it is necessary to be able to redirect run-time information if requested by the UI. Normally, run-time info is directed to the terminal where the design environment is started, but upon request it can be piped somewhere else, e.g. to a web browser. If the browser termi-

nates the connection, the run-time info is directed back to the original terminal.

## 5. Example

In this example we show how a designer interacts with the design environment which we call the “Design Server.” The example shows the actions the Design Server performs, and the information the designer receives back. The salient points are that the Design Server supports exploration before the design is completely specified and as the specifications evolve, and that heterogeneous estimation approaches are seamlessly integrated.

Consider the design of a 1024-point FFT algorithm for use in a real-time system. The FFT can be executed on a DSP or general purpose processor, or it can be implemented in special purpose hardware. In either case, it is important that it can be executed under the real-time constraints. The following subsections describe the exploration process using the Design Server.

### 5.1. Start the Design Server.

The designer starts the Design Server and connects via a web browser (figure 7). A menu is returned (figure 8): the items that are underlined represent commands that are always available. The last four items represent categories of requests that are defined within contexts. Since no context has been specified, there are no context specific requests listed.

### 5.2. Algorithmic Explorations

The designer gives the incomplete specification “FFT.” In response, the Design Server searches for domains that contain rel-

evant expertise, and proposes a context for the design. Please see figure 9. Since the specification only identifies an algorithm, the most relevant aid that can be provided is a measure of algorithmic complexity. To find out how this compares to different FFT algorithm alternatives, the designer requests to see the context, which includes a list of available domains (see figure 10). By choosing an alternate FFT domain, namely the Radix-4 domain, a new context is established, and when the designer requests algorithmic complexity again a new result is returned (43,520 operations). Likewise, other FFT styles can be explored.

### 5.3. Exploration of Implementation Platform

Thus far, only algorithmic aspects have been explored. To explore how the choice of implementation platform affects the design, the designer adds the term “DSP56K” to the specification (figure 11). The Server searches and finds a domain that models the given processor, and after adding this domain to the context, the Server is now capable of interpreting the request for execution time. The result is based entirely on models that correlate algorithmic properties with architectural properties. This type of analysis is often too cumbersome for a designer to do, but it is easy with the aid of the Design Server.

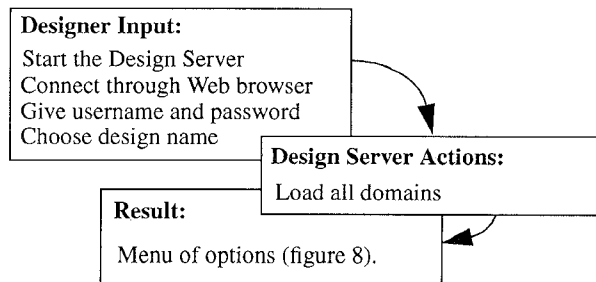


Figure 7. Start the Design Server.

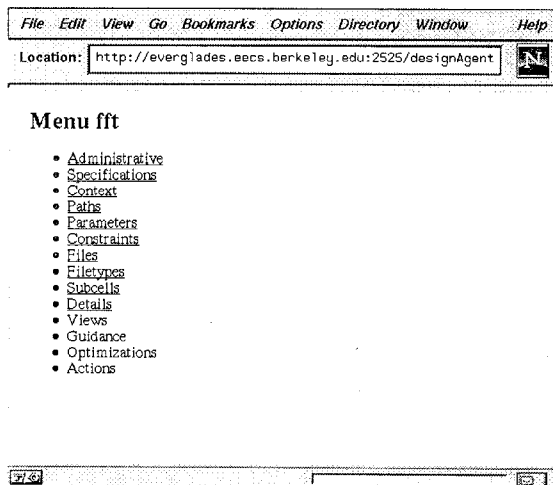


Figure 8. Design Server Main Menu.

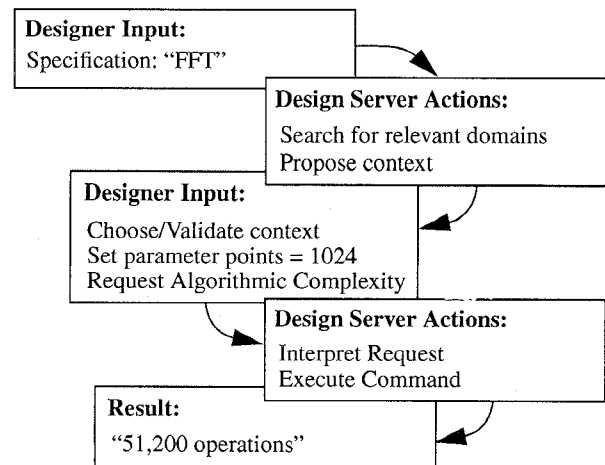


Figure 9. Algorithm Exploration

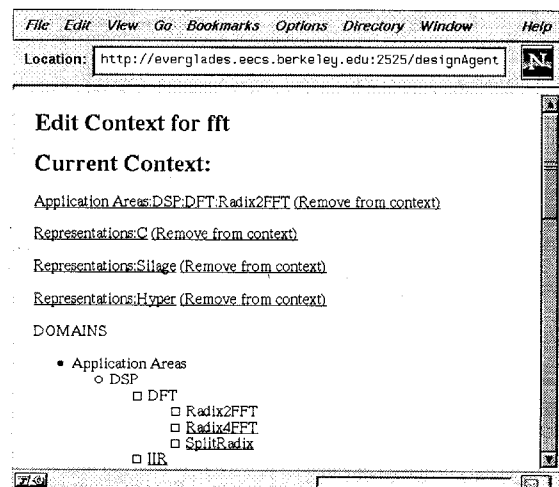


Figure 10. Context menu including list of available domains.

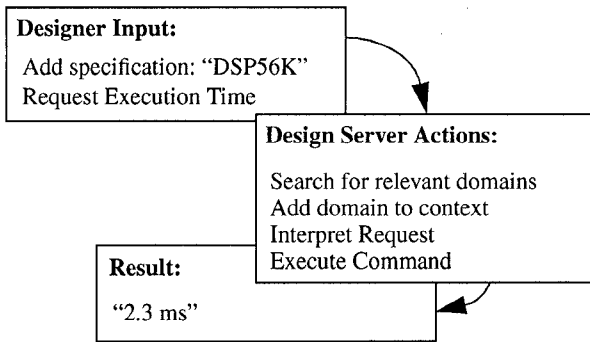


Figure 11. Adding choice of implementation platform.

Before deciding on a particular platform, the designer changes the DSP56K specification to "ASIC," to explore an ASIC implementation of an FFT. The Design Server removes the DSP56K domain from the context, and adds a domain that models features of ASIC designs. When requesting execution time in this context, the result returned is 5  $\mu$ s. Notice how the design environment facilitates exploration by hiding the underlying set of heterogeneous estimation and prediction models.

#### 5.4. Implementation Phase

The Design Server can also aid the designer during the implementation phases of the design process. For example, to get started on writing the FFT algorithm (e.g. in C), the Design Server can point to existing FFT descriptions that are available. In addition, the Server can provide profiling support for code analysis, or can provide more accurate execution time estimates as the code (and thus the design specification) becomes more and more refined.

### 6. Conclusions and Future Work

This paper has presented a new design environment which provides a framework for design exploration. The environment allows integration of heterogeneous estimation approaches, and frees designers from the mechanics of how to gather the quantitative information that is needed to make important design decisions.

The future directions for this work has two main emphases. First, to enable greater sharing and reuse, future versions of Design Servers will be able to communicate with each other and exchange domain knowledge (figure 12). This will allow users to access a much larger body of distributed knowledge, but the search strategy will need to be extended to work efficiently in a distributed environment.

The second thrust is to incorporate support for design guidance. As with estimations and exploration, the design environment

would not give design guidance, since there are emerging tools in that area (e.g., see [11]), but rather provide the underlying infrastructure for handling a heterogeneous set of guidance tools.

### References

- [1] M.F. Jacome and S.W. Director, "Design Process Management for CAD Frameworks," Proc. of the 29th ACM/IEEE Design Automation Conference, pages 500-505, 1992.
- [2] P.R. Sutton, J.B. Brockman, and S.W. Director, "Design Management Using Dynamically Defined Flows," Proc. of the 30th ACM/IEEE-CS Design Automation Conference, pages 648-653, 1993.
- [3] J.C. Lopez, M.F. Jacome, and S.W. Director, "Design Assistance for Cad Frameworks," European Design Automation Conference, pages 494-499, 1992.
- [4] P. Bingley, O. ten Bosch, P. van der Wolf, "Incorporating Design Flow Management in a Framework based CAD System", Digest of Technical Papers, IEEE/ACM International Conference on Computer-Aided Design, pages 538-545, 1992.
- [5] K.O. ten Bosch, P. van der Wolf, and P. Bingley, "A Flow-Based User Interface for Efficient Execution of the Design Cycle," Digest of Technical Papers, IEEE/ACM International Conference on Computer-Aided Design, pages 356-363, 1993.
- [6] S. Kleinfeldt, M. Guiney, J.K. Miller, and M. Barnes, "Design Methodology Management," Proc. of the IEEE, vol. 82, No. 2, pages 231-250, Febr. 1994.
- [7] J.K. Ousterhout, "Tcl and the Tk Toolkit," Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.
- [8] M.J. McLennan, "[incr Tcl]: Object-Oriented Programming in Tcl," Proceedings of the Tcl/Tk Workshop, University of California at Berkeley, June 10-11, 1993.
- [9] D. Libes, "Exploring Expect: a Tcl-based Toolkit for Automating Interactive Programs," O'Reilly & Associates, Sebastopol, CA, 1995.
- [10] CAD Framework Initiative, Inc., "Tool encapsulation specification standard," CFI Doc. DMM-91-G-1, 1991.
- [11] L. Guerra, M. Potkonjak, and J. Rabacay, "System-Level Design Guidance Using Algorithm Properties," IEEE Workshop on VLSI Signal Processing VII, pages 73-82, 1994.

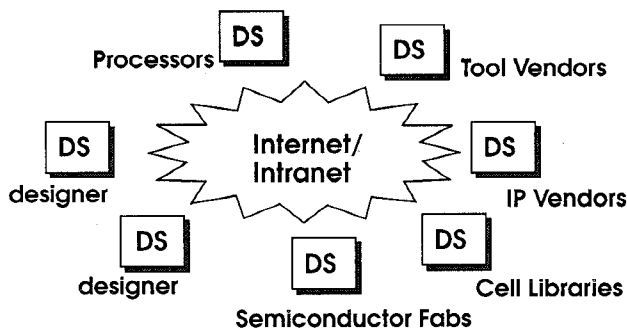


Figure 12. Distributed Design Servers.