

A VHDL-93 HARDWARE DESCRIPTION BROWSER

Laura C. DeBrock and Krishnaprasad Thirunarayan
Department of Computer Science and Engineering
Wright State University, Dayton, Ohio-45435.
tkprasad@cs.wright.edu

Abstract

This paper describes the design and implementation of the VHDL-93 Hardware Description Browser, which is a tool for the intelligent retrieval of information from VHDL designs. The Browser consists of two UNIX processes: a TCL/TK Graphical User Interface and a Prolog search engine. The GUI elicits queries from the user and submits them to the Prolog search engine via a two-way communication pipe. The search engine satisfies queries by traversing a forest of parse trees corresponding to the associated VHDL designs. The results are then sent to the GUI for posting.

Two public-domain tools were used to implement the Browser: SWI-Prolog, chosen for its pattern-matching capabilities and its use as an executable specification language, and TCL/TK for its high-level windowing operations and text-substitution capabilities to allow meta-programming.

1 INTRODUCTION

VHDL (VHSIC Hardware Description Language) has been developed for the specification, simulation, verification, and documentation of hardware designs. It has greatly facilitated the portability of designs and interoperability of various CAD-tools built by different vendors. Because VHDL descriptions are extremely large and complex, plain text searches are unwieldy and insufficient for determining the structure and organization of the designs. Access to design details in an efficient and convenient form is critical, as a VHDL design can be subjected to many modifications by different programmers throughout its life cycle. Therefore, the VHDL-93 Hardware Description Browser (DeBrock-96), a tool for the intelligent retrieval of information from VHDL designs, was developed.

2 DESIGN

The Browser was designed to satisfy the following requirements.

- 1) It should accommodate the full IEEE 1076-1993 Standard of VHDL.
- 2) It should not require user knowledge of programming or database languages.
- 3) It should efficiently perform a minimal set of frequently-used queries. Furthermore, the query-set should be easily extendible by experienced users.
- 4) It should support a graphical user interface that allows a user to browse VHDL-93 hardware descriptions via a series of mouse clicks, as well as construct queries at a high-level of abstraction.
- 5) It should be implemented with high-level languages to minimize development time and facilitate future enhancements and maintenance.
- 6) It should be implemented using public-domain software.

VHDL-93 is an immense, comprehensive language. So, satisfying the requirement that the Browser accommodate the full VHDL language was no small task (Perry-93). Prolog, in its role as an executable specification language and with its pattern-matching facilities, has significant software engineering strength for our project. *The description in Prolog enables us to closely mirror the syntax specification given in the IEEE 1076-1993 standard of VHDL, and additionally, exploit the formal syntax to control and guide the browsing search process.* In fact, the knowledge encapsulated in the syntax was used to garner a set of search paths holding potential for success, as well as to terminate search paths destined to fail.

The search engine was built on top of a VHDL parser/pretty-printer system (Thirunarayan-95). The idea was to apply the parser to generate a forest of parse trees, one for each design unit. A query would be phrased as a parse tree (equivalently a Prolog term) with nodes containing bound and/or unbound variables. The search process would consist of attempts to unify the query tree with the trees and/or sub-trees of the parsed VHDL hardware descriptions. In order to perform a more complex query, a set of constraints would be applied to the trees matching the query tree. Trees that unified with the parse tree and satisfied the constraint set would be returned as the results of the query.

Another requirement on the Browser was that it be easy to use and that it not presume user knowledge of Prolog. Thus, a graphical user interface was designed, to interact with the Prolog search engine. The GUI directs the querying process by presenting the user with a set of menus. The menus implement the set of frequently-used queries. Queries outside this set can be formed by a user knowledgeable in Prolog and VHDL syntax.

TCL/TK was selected for implementation of the GUI because it is a high-level language available in the public domain with primitives for such objects as windows, buttons, and menus, and handles events such as mouse clicks (Ousterhout-94). Additionally, we were able to harness the text substitution powers of TCL/TK to dynamically generate queries to be sent to the search engine, thus adding a meta-level of programming which greatly reduced development time.

In summary, the Browser consists of a GUI and a search engine configured in a master/slave relationship. The GUI elicits queries from the user and sends them to the search engine via a two-way communication pipe. The search engine traverses parse trees generated by the parser to represent VHDL hardware descriptions. The search engine sends results back to the GUI for posting in windows. A results window is created for each query, enabling the user to concurrently view the results of many queries. (See figure at the end of the paper.)

3 IMPLEMENTATION

The Browser is implemented as two UNIX processes, a TCL/TK GUI (approximately 700 lines of code) and an SWI-Prolog search engine (approximately 500 lines), that are connected via a two-way communication pipe. The following sections discuss the implementation details of the GUI, the search engine, and the communication mechanism.

3.1 TCL/TK - Prolog Communication

The TCL/TK GUI process and the Prolog search engine have a master/slave configuration, with the GUI being the master. The TCL/TK process spawns the Prolog process and establishes a two-way communication pipe. This pipe is used to transmit query requests to the search engine and to receive query results back from the search engine.

When the TCL/TK GUI process spawns the Prolog process and sets up the two-way pipe, the logical values of Prolog's *stdin* and *stdout* are re-mapped to that of the pipe. Thus, *stdin*, which is usually mapped to the keyboard, and *stdout*, which is usually mapped to the monitor, are re-mapped to the pipe. When a Prolog goal is written to the pipe and flushed, the Prolog process automatically evaluates it just as if a user had typed the goal at the keyboard. Prolog writes the results to *stdout*, which re-mapped to the pipe, automatically sends the results to the GUI.

This re-mapping phenomena was of immense value to us in developing the GUI. We determined a list of services that the GUI would like the Prolog search engine to perform and then wrote a set of Prolog predicates corresponding to those services. To request a service or perform a query, the GUI simply writes the appropriate Prolog goal, dynamically created via TCL/TK's text substitution operations, to the pipe for the search engine to satisfy. Any information that Prolog would write to *stdout*, is now automatically written to the pipe. The GUI employs an interrupt handler to alert itself to new data in the pipe which is then read and

processed. Flags are used to distinguish intended versus extraneous communications.

3.2 Prolog Search Engine

The Prolog search engine contains three sets of predicates:

- 1) Predicates that provide the TCL/TK GUI - Prolog search engine communication.
- 2) Predicates that implement the minimal query set.
- 3) Predicates that form the generic query mechanism which enables an advanced user to construct and execute her own queries.

The predicates that provide the communication between TCL/TK GUI-Prolog search engine are essentially responsible for adding a layer of control upon Prolog's normal operations. Rather than solely writing data to *stdout*, these predicates also send messages to the GUI stating the commencement and termination of the transmission of query result information. The GUI uses this information to set flags which are used to judge incoming information as intended query results or erroneous and/or extraneous transmissions.

The "canned" queries, which are VHDL-specific, allow the user to:

- 1) *list all* entities, architectures, packages, configurations, files, user-defined data types, statements, and architectures containing processes;
- 2) *show source code* of a given entity, architecture, package declaration, package body, configuration declaration, user-defined routine, and user-defined data type,
- 3) *locate* files, variables, signals, constants, routine calls, instances of VHDL-93 components, user-defined routine definitions and user-defined data types, for a given data type, id, or read/write mode,
- 4) *show the component hierarchy* of an entity, and
- 5) *determine part-subpart-relationships*.

When developing a prototype set of predicates for the minimal query set, it was noticed that the code for the different

queries was very similar. Each query involved traversal of parse trees to search for matches to a given pattern. In some cases pattern-matching was further constrained by additional goals. A tree, whose root node had the same value as the pattern functor, and whose child nodes unified with the pattern arguments, and satisfied a set of constraints, is considered a match. We concluded that the query process could be abstracted to a predicate, referred to as the generic query mechanism, requiring pattern, constraint set, and results field information. The constraint set is implemented as a list of boolean goals. The results field information indicates which child nodes of the matching trees should be returned as query results.

The generic query mechanism is responsible for the traversal of the forest of parse trees in the design database. First, the generic query mechanism culls from the design database a set of parse trees with potential for unification with the query tree. This set is then searched in a depth-first manner. Heuristics based upon the root value of the query tree and current node of the search path are employed to terminate paths destined for failure. In order to be deemed a match, a parse tree or sub-tree must unify with the query tree, as well as satisfy the associated constraint set. The result field information is then applied to the matching trees and sub-trees, extracting desired nodes to form a results list which is sent to the GUI.

The generic query mechanism is, in fact, independent of VHDL-93 syntax. That is, its algorithm focuses on traversal of the parse trees and is independent of the form or content of the trees. Thus, the generic query mechanism can be used for browsing source code written in any language, provided that the Prolog parser is given.

The predicates that perform the queries of the minimal query set employ predetermined query trees whose node values are elicited from the user by the GUI. Most of the queries of the minimal query set invoke the generic query mechanism.

3.3 The GUI

The GUI can be viewed as providing the user with three categories of services:

- 1) basic administrative services such as help, quit, and opening of the desired VHDL-93 hardware descriptions,
- 2) frequently-executed queries for VHDL-93 hardware descriptions, and
- 3) custom-designed queries via the generic query mechanism.

In order to request an administrative service or invoke one of the frequently-executed queries, the user need only click on an appropriate line of a listbox, a menu-like object in TCL/TK. Any additional required information is elicited from the user via a series of dialog boxes. The generic query mechanism is invoked by clicking on a button which pops up a dialog box to garner the necessary Prolog terms from the user and send the resultant query to the search engine.

The procedures defined in the GUI can be categorized into five groups based on their functionality:

- 1) initialization routines,
- 2) routines for elicitation of queries from the user,
- 3) routines for sending information to the search engine,
- 4) routines for receiving information from the search engine, and
- 5) termination routines.

TCL/TK proved to have several powerful assets, as well as some drawbacks for the development of the GUI. The string processing capability of TCL/TK allowed the dynamic generation of Prolog goals. For instance, when a user clicks on a line in a listbox, a TCL/TK captures the textual content of the line and splices it with literals typed in by the user and text strings specified by the procedure to generate a Prolog goal matching a query predicate in the Prolog search engine. An additional strength of TCL/TK was its high-level windowing primitives that enabled the rapid construction of an aesthetic GUI.

However, TCL/TK had some deficiencies too. The primary ones is the lack of closure for local variables, which led to a reliance on global variables, and the independent execution of procedures which made it

difficult to control the sequence of operations. For example, consider the case of a dialog box instructing the user to press a button. The dialog box should not appear until the button is present. The TCL/TK primitive *tkwait* was used to sequence such events.

4 CONCLUSIONS

Prolog and TCL/TK proved to be excellent tools for implementing the VHDL-93 Hardware Description Browser. The pattern-matching abilities of Prolog made it well-suited for development of a search engine. Its high level of abstraction helped minimize coding time, and its modularity readily lends future extension of browsing capability. TCL/TK provided extremely high-level primitives that allowed rapid prototyping of the GUI. By capitalizing on TCL/TK's string processing capability and the re-mapping of *stdin/stdout* of the Prolog process to that of the two-way communication pipe between them, we created a GUI that dove-tailed with the Prolog search engine. The Browser allows the user to quickly list the constructs and dependencies of a VHDL-93 design.

The Browser can be further developed by extending the query set, by enhancing the generic query mechanism, and by interfacing it to a text editor and a VHDL-93 simulator, to produce a full-scale development environment.

The Browser is available from <ftp://www.cs.wright.edu/pub/vhdl>.

References

(DeBrock-97) L. C. DeBrock, "A VHDL-93 Hardware Description Browser", Master's Thesis, Wright State University, 1997.

(Ousterhout-94) J. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

(Perry-93) D. Perry, *VHDL*, McGraw-Hill: New York, NY., 1993.

(Thirunarayan-94) K. Thirunarayan, R. Ewing, and P. Reintjes, "VHDL93-Parser in

SWI-Prolog: A Basis for A Design Query System", Technical Report, Wright State University, 1994.

Figure: VHDL Browser Architecture

