

# An Object-Oriented Schema for Electronic CAD Tools

Eric Hughes, Linus Huang and Eric J. Golin  
University of Illinois at Urbana-Champaign  
Department of Computer Science  
1304 W. Springfield Ave.  
Urbana, IL 61801-2987

## Abstract

The use of an object-oriented database to support a visual design environment is presented. ViDEO, the **V**isual **D**esign **E**nvironment with **O**bjects, is based on VHDL, and incorporates both research and commercial VHDL tools. ObjectStore, an object-oriented database management system, serves as the basis for sharing design data between CAD tools. Database support for editing visual designs, compilation of visual designs into textual VHDL models, and simulation of VHDL models is described. An overview of the use of a database for integration of CAD tools into a design environment is presented.

## Section 1. Introduction

The ViDEO CAD environment consists of a visual model editor, a visual model to VHDL compiler, and commercial VHDL simulation tools. ViDEO is based on a visual design paradigm, where the designer interacts with graphical representations of design information. The ViDEO visual model editor allows the designer to describe a system in visual VHDL [1] (vVHDL). In vVHDL, a design is composed of graphical icons (including text) which represent the various VHDL constructs. vVHDL models are compiled into textual VHDL [2] for analysis and simulation by commercially available tools. Models expressed in vVHDL and VHDL are stored in the database, which facilitates the integration of the visual editor and VHDL simulation tools. Results from simulations are also stored in the database. The design tools share data in an integrated manner, without translating data between representations.

The ViDEO environment supports collaborative design by using a database management system (DBMS) to store data passed between tools. The stimulus for this effort is the design of the data acquisition system (DAQ) for the Solenoidal Detector Collaboration (SDC) Superconducting Super Collider [3]. The SDC DAQ will be designed by many groups, using a variety of design tools and strategies. For this reason, our work is not limited to VHDL, but includes Verilog [4] and MODSIM [5]. Database support for these modeling languages will be reported in a future publication.

The ViDEO environment is based on the ObjectStore [6] object-oriented DBMS. The use of a DBMS is preferable to file-based frameworks, which rely heavily on translators, and offer limited concurrency control. The authors chose not to develop a specialized data manager, which would not offer the features of a commercial DBMS. In addition, relational DBMSes are inefficient and cumbersome for CAD applications. Object-oriented DBMSes have an appropriate data model, and several products are currently available.

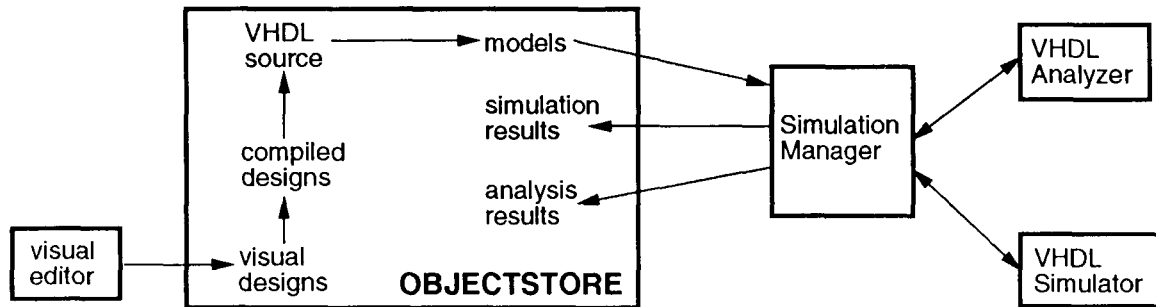


Figure 1: Abstract architecture of the ViDEO environment

The DBMS provides a flexible, documented interface to persistent data, which simplifies the coding of the design tools. This interface also simplifies the addition of new tools to the environment. Integration through the DBMS is compatible with design standards like VHDL. VHDL can be expressed as a set of class definitions, and additional information related to models can be attached in a structured manner. This information is described in the schema and stored in the database, rather than being attached to models in unstructured VHDL attributes.

Figure 1 illustrates the abstract architecture of the ViDEO environment. The visual editor is used to create a visual hardware design. This design is converted into a compiled design, from which textual code is generated. A complete design model is then formed from the code and is fed to a simulation manager, which analyzes and simulates the model. Finally, the results of the analysis and simulation are stored in the database.

Other researchers have investigated the use of data management technology for integrating CAD tools. Chung and Kim have developed SDE [7], a design environment based on an object-oriented representation of VHDL. The SDE data model is intended to represent design constraints in pseudo-VHDL, and has no direct mapping to VHDL constructs. The Berkeley toolset includes Oct [8], which is a data manager tailored to VLSI layout and related design operations. Although Oct is extensible, it requires that entire designs fit in memory, and thus is not as flexible as a DBMS. Cbase [9] is an object-oriented CAD framework which is based on the Vbase DBMS [10]. The Cbase schema supports layout structures like cells and nets, rather than VHDL models.

In this paper, we examine the integration of CAD tools into an environment using an object-oriented DBMS. The tools discussed are based on VHDL. An introduction to object-oriented DBMSes is included in the next section. Section 3 presents an overview of the schema for the ViDEO environment. The following three sections describe database support for visual editing, compilation of visual models into VHDL, and simulation. Conclusions and future work are given in Section 7.

## Section 2. Object-oriented databases

A *database* is a repository for persistent data. A *database management system* (DBMS) provides an interface between the database and the database user and oversees five major tasks: interaction with the system file manager; data integrity enforcement; access security enforcement; backup and recovery; and concurrency control [11].

*Object-oriented* DBMSes (OODBMSes) have been developed in recent years to support a new generation of applications featuring, among other things, complex data structures and large amounts of data. Although there is no single object-oriented paradigm, most commercial OODBMSes share

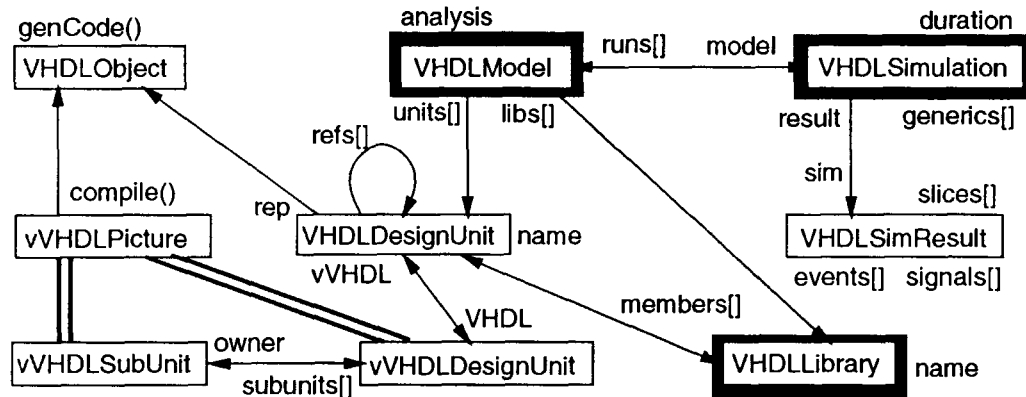


Figure 2: Unified Schema

a number of common characteristics [12]. They are based upon a general-purpose object-oriented programming language, and the data types declared in the application comprise the *schema*, which is used by the OODBMS. They generally support data *encapsulation*, by which objects model not only structure but semantics as well, in the form of *methods*. Most OODBMSes also support type hierarchies and inheritance of attributes and methods.

The ObjectStore OODBMS underlying the ViDEO environment is based upon C++. ObjectStore includes an interactive database browser and schema designer, and a version-control mechanism for support of cooperative work on large-scale designs [6]. The recent 2.0 release includes a schema evolution facility, which keeps data consistent with schema modifications. Although we plan to use schema evolution to maintain design databases, we have not tested this facility.

The schemas illustrated throughout this paper make frequent use of *relationships* and *collections*. ObjectStore relationships allow the user to model inverse attributes and enforce referential integrity, primarily through the elimination of “dangling pointers.” Collections provide users with aggregate data structures of varying behaviors. They are included in the user libraries that come with the ObjectStore software.

### Section 3. A unified schema for VHDL based design

We have designed a unified schema for the database which supports the design tools described in Section 1. The central classes of the unified schema are shown in Figure 2. In this diagram, rectangles represent classes, heavy double lines represent inheritance, arrows represent attributes, and bidirectional arrows represent relationships. Attributes with the suffix “[]” are collections. Outlined classes have an *extent*, which is a maintained list of instances of the class.

The **VHDLLibrary** class (lower right corner of Figure 2) is used to represent libraries of logically related design units. The VHDL library extent is a database root which serves as a starting point for browsing the database. Library names and design unit names are used to identify design units.

The **VHDLDesignUnit** class corresponds to a library\_unit in the VHDL standard [2]. The design unit is the basis of VHDL design work. The **refs** attribute supports analysis of an evolving model. **VHDLDesignUnit** has subclasses entity, architecture, configuration, package and package body.

Instances of **vVHDLPicture** correspond to pieces of a visual model. A visual model is a graphical representation of a VHDL model, which can be compiled into VHDL. **vVHDL** pictures are translated

into `VHDLObjects` by the `compile` method. `VHDLObject` is a superclass of a class for each item in the VHDL grammar. Each `VHDLObject` understands the `genCode` method, which generates textual VIIDL code for the object.

The `vVHDLDesignUnit` class represents `vVHDL` programs which correspond to design units. `vVHDL` units have a graphical representation which is manipulated by the visual editor. A `vVHDL` unit can have `subunits`, which allow the visual design to be decomposed into a hierarchy of subpictures.

The `VHDLModel` class represents *models*, which can be simulated. A model consists of a collection of design units which form a cohesive design. Models, rather than individual units, can be recompiled in the ViIDEO environment. The `libs` attribute lists all the VHDL libraries used by the model.

The `VHDLSimulation` class represents the simulation of a VHDL model to produce a `VHDLSimResult`. A simulation includes settings for the `generics` of the model. A simulation result includes descriptions of the `signals` and `events` monitored, and the resulting waveforms are partitioned into `slices`.

The ViIDEO environment uses the unified schema to describe all data stored in the database. This technique allows the tools to share design data in a natural manner. The following discussions of different aspects of the ViIDEO environment will refer to classes in the unified schema, as well as related class definitions.

## Section 4. Database support for visual VHDL

The visual editor supports the creation and manipulation of programs in the visual VHDL (`vVHDL`) design language. [1] `vVHDL` provides a visual metaphor for the design of electronic systems. The user constructs a design by drawing a collection of pictures in the visual editor. This is similar to the use of a schematic editor, but `vVHDL` is based on the VHDL model of hardware and allows the graphical specification of both structural and behavioral aspects of a design. The constructs of VIIDL, such as architecture bodies, signals, statements and expressions, are mapped into graphical representations in `vVHDL`. Different icons or arrangements are used for each construct, with related constructs (e.g., all concurrent statements) using similar visual symbols.

Hardware designs are often too complex to represent with a single diagram without making the picture very large or the details very small, so `vVHDL` provides a number of *abstraction* constructs. These allow a part of a design to be moved into a separate picture. For example, the design for an architecture body might contain a process statement, with the body of the process given in another picture. This mechanism organizes the pictures into hierarchies. The roots of the hierarchies are the `vVHDLDesignUnit` objects, which correspond to units visible at the library level.

The editor uses the database to store and organize the pictures. The *visual* subschema provides a representation for the hierarchies of pictures. The base class of the visual schema class hierarchy, shown in Figure 3, is the class `vVHDLPicture`. The pictures are divided into two subclasses, `vVHDLDesignUnit` and `vVHDLSubUnit`. The subclasses of `vVHDLDesignUnit` are the pictures that are found at the root of a tree of pictures, and include entities, architectures, packages and configurations. These objects are linked into the unified schema, as shown in Figure 2. The graphical representation of a design element is the `vVHDL` attribute of the corresponding library object.

`vVHDLSubUnit` has subclasses for the pictures which are abstractions of a part of a larger design. Currently, `vVHDL` provides several kinds of abstraction mechanisms. The entire body of a construct, such as a process or procedure, may be abstracted into a separate picture. A special construct called a *link box* provides a link to a general subpicture containing declarations or statements. Pictures are stored as monolithic objects in the database, other than relationships with subpictures. The visual symbols which comprise a picture are not represented by objects, but rather incorporated

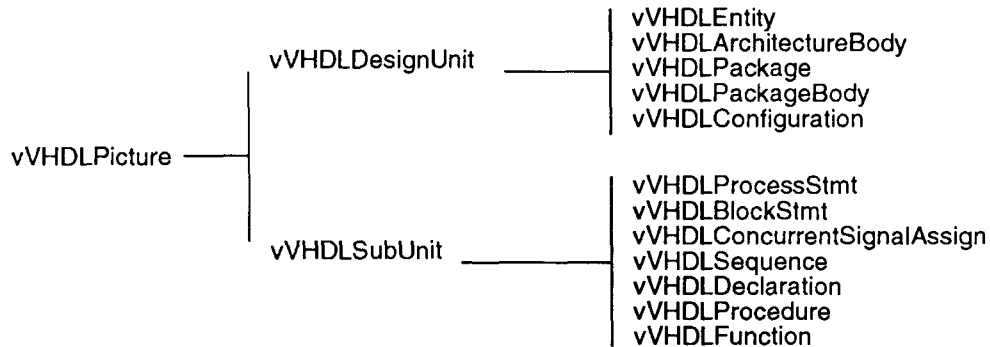


Figure 3: Class Hierarchy for Visual Subschema.

into the picture objects. The `compile` method provides an interface to allow the compiler to access the individual visual symbols.

## Section 5. Database support for parsing vVHDL

Since commercial simulators require textual VHDL as input, the user must direct the vVHDL compiler to convert the visual program into text to simulate a vVHDL hardware design. A key issue is to efficiently go back and forth between the visual and textual representations of the design. For example, the user may wish to track down the (sub)picture which visually represents code that has resulted in a simulator error, so that the user may correct the problem in the visual editor. The compiler therefore produces as output an “intermediate form” from which it is easy to generate VHDL code and to associate the code with the corresponding pictures in the visual program.

A subschema for this intermediate form is suggested by the defining grammar of VHDL [2]. Conceptually, a class in this *compilation* subschema exists for every syntactic construct in the language. In the case of a construct that is an aggregate of sub-constructs, the class contains a data member of appropriate type for each sub-construct.

For example, here are the productions defining the `architecture_body` and `process_statement` constructs of VHDL:

```

architecture_body ::=
    architecture identifier of entity_name is
        architecture_declarative_part
    begin
        architecture_statement_part
    end [ architecture_simple_name ] ;

process_statement ::= [ process_label : ]
    process [ ( sensitivity_list ) ]
        process_declarative_part
    begin
        process_statement_part
    end process [ process_label ] ;
  
```

The corresponding class definitions in the compilation subschema are given below.

```

class VHDLArchitectureBody {
    VHDLString *identifier;
    VHDLString *entity_name;
    VHDLArchDeclPart *arch_decl_part;
    VHDLArchStmtPart *arch_stmt_part;
};

class VHDLProcessStatement {
    VHDLString *label;
    VHDLString *sensitivity_list;
    VHDLProcDeclPart *process_decl_part;
    VHDLProcStmtPart *process_stmt_part;
};
  
```

All classes in the compilation subschema have the prefix “VHDL.” The implemented schema is condensed to filter out detail in the VHDL standard which is not required for translating pictures into textual VHDL. For example, the `VHDLString` class is used in the place of separate classes to represent identifiers, labels, etc.

The root class of the compilation subschema is class `VHDLObject`. A compiled design is incorporated into the ViDEO environment as an attribute of a `VHDLDesignUnit` instance (see Figure 2).

The compilation subschema yields a tree-like structure when the vVHDL compiler is invoked on a visual design. Generating textual VHDL code from this format is a fairly simple task. All classes in the compilation subschema understand the `genCode` method which generates text for its portion of the overall program. Generating code for the entire program thus amounts to an inorder traversal of the tree, invoking the `genCode` method at each “node.” The text, as well, is stored in the database under a very simple subschema with one class, `VHDLSourceFile`. This class has three attributes: a name, a string of textual VHDL, and a backpointer to the `VHDLObject` instance which generated that text.

The association between the visual and compiled representations of a VHDL model is made by establishing bi-directional pointers between corresponding `VHDLObject` and `vVHDLPicture` instances, as illustrated in Figure 4. The compilation subschema, which is much larger than the visual subschema, contains many classes which do not correspond to a subclass of either the `vVHDLDesignUnit` or `vVHDLSubUnit` classes. This means that an instance of one of these classes is visually represented as a component of a picture corresponding to an object higher up in the tree structure of the compiled representation.

## Section 6. Database support for VHDL simulation

The ViDEO environment supports VHDL simulation by the Intermetrics VDE [13] and Vantage Spreadsheet [14] tools. The schema is intended to be extensible to other VHDL simulators, with the addition of classes to represent tool-dependent information. VHDL simulation can be partitioned into the following phases: library creation, analysis, model generation, simulation, and report generation.

The VHDL simulation subschema is given in Figure 5. The symbols used in the figure are described in Section 3. The subschema includes several classes from the schema in Figure 2, including `VHDLDesignUnit` and `VHDLModel`.

In the library creation phase, a repository for design units with a symbolic name is created. The `libs` attribute of a VHDL model is the collection of libraries used by the model. The `libs` attribute

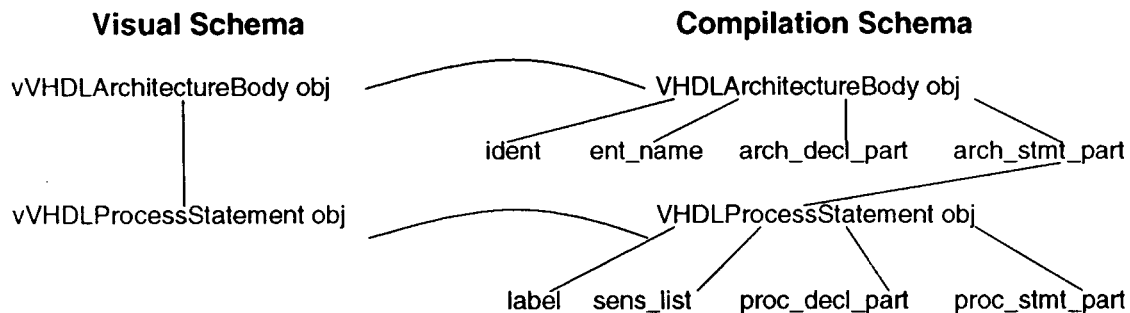


Figure 4: The relationship between the visual and compilation subschemas.

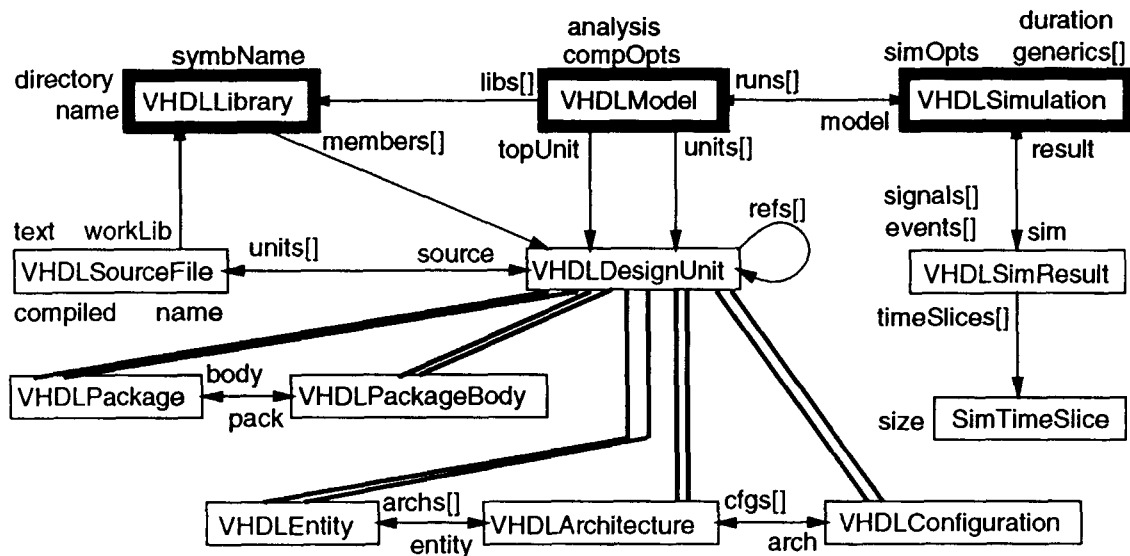


Figure 5: Simulation Subschema

includes libraries provided by the simulation toolset (like the IEEE 1164 library of the Vantage toolset). In addition to a directory name, a VHDL library can have a symbolic name (**symbName**) for reference within a design unit.

The analysis phase consists of parsing (but not linking) VHDL source files. The ViDEO environment supports recompilation of VHDL models, rather than individual design units. The compilation order of design units is dictated by the **refs** attribute and by the *implicit references* of design units. VHDL includes implicit references between entities and architectures, architectures and configurations, and packages and package bodies. The implicit references are represented by the attributes of subclasses of **VHDLDesignUnit** in Figure 5. In the analysis phase, the ViDEO environment must create a sequence of calls to the analysis tool from the set of design units. In addition, both analysis tools allow command line option information, which is stored in the **compOpts** attribute of a VHDL model.

In the simulation phase, a simulation result is generated from a VHDL model, with a given set of generic settings. A simulation is modeled as an instance rather than a method of **VHDLModel** for two reasons. First, generic settings and other information need to be associated with each simulation. Second, it might be useful to combine certain simulation results, which is simpler if each result is created by a particular simulation.

Each simulation can monitor **signals** and **events**, which are associated with the result to facilitate graphical presentation. Monitored signals produce an edge when the value of the signal changes. Events are used to monitor a named condition which is not directly represented by a signal in the model. Event edges can have values attached to them. Events have *tags*, which describe the values attached to corresponding edges. For example, in a model of a queue, **Overflow** might be a named event. The Overflow event might have a tag **WordsLost**, and corresponding edges would have attached values for the number of words lost.

The **VHDLModel** class includes the attribute **topUnit** to support invocation of VHDL simulators. The **topUnit** of a model is either an architecture or configuration which uniquely identifies the top-level design unit of a model. Information for tool-specific simulator options is stored in the **simOpts** attribute of a simulation. The **duration** of a simulation is controlled either by the user or by an

assertion statement with an appropriate severity level.

In the report generation phase, textual result reports are extracted from the binary representations created by the toolsets. The Intermetrics Report Generator uses a report file with report control language (RCL) statements. The report file selects the signals to be included and controls the format of the output. This information is associated with the simulation result. Similarly, the Vantage `comparerdb` tool takes a signal list and format information to produce a textual result. In both toolsets, VHDL models can produce textual output using the TEXTIO package.

The Intermetrics and Vantage textual reports can be parsed to create `VHDLsimResult` objects in the database. In addition, direct textual output from a VHDL model can be parsed to create a persistent simulation result. Finally, multiple results can be combined to produce a single result for each simulation.

In the ViDEO environment, structured simulation results are stored in the database. The result structure facilitates the comparison of results created by different tools. In addition, designers can view results without using the Intermetrics or Vantage toolsets.

## Section 7. Conclusions and future work

The ViDEO environment demonstrates that an object-oriented database is well suited as a basis for integrating CAD tools. ViDEO combines both locally developed research tools such as the `vVHDL` editor and compiler with commercial tools from several vendors, and includes applications developed in C++, C, and LISP. Our approach is to develop interrelated schemas to represent the information associated with the various tools. Key to this effort was the development of a unified schema to provide an overall structure to the database.

The development of the ViDEO environment is under way. Initial prototypes of the individual tools are largely complete, and we are working on the overall integration using the database. The visual editor is in place but is not yet fully integrated with the database. The compilation schema has been designed and implemented but needs fine-tuning. The simulation schema is completed and functioning. The subschemas have not yet been integrated into a unified schema. We anticipate that the overall integration of these components will be completed by summer 1993.

We expect the use of a general purpose object-oriented DBMS to simplify the integration of CAD tools by reducing the effort needed to store and retrieve data. The preliminary implementation of the simulation manager supports this expectation. The authors were able to design a schema which clearly expresses the information required by the integrated tools. It should be noted that the authors are not database researchers, but do have a basic understanding of database issues and how they relate to the needs of CAD tool integration. Using a DBMS also allows the development of tools that exploit complex relationships between the data. We plan to demonstrate this by expanding the ability of the designer to interact with the environment through the visual editor. For example, the DBMS can maintain multiple versions of a `vVHDL` picture, some of which might have existing equivalent textual VHDL design units.

The use of a database also allows the CAD environment to evolve with new technologies. New CAD technologies can be incorporated into an extensible schema, and existing design data can be updated with the help of schema evolution (see Section 2). For example, changes in the VHDL standard can be incorporated into the schema by modifying classes as needed. In addition, the schema evolution facility of the DBMS supports the migration of existing database objects to the fit the new schema.

Several topics remain for further study, including graphical simulation result presentation, synthesis, and database browsing. Control integration (i.e. interprocess communication) of the environment

using message-passing software will be investigated. In addition, support for error handling using visual techniques will be studied. In this work, the efficiency of the final implementation will be an important concern. For example, performance studies can be used to identify the need for indexes and possibilities for clustering related data within database segments. In addition, the performance of different concurrency control mechanisms can be investigated.

## Section 8. Acknowledgements

The authors wish to thank Marianne Winslett and Michael Haney for reviewing this work. The authors also wish to thank Robert Downing, Larry Jones, Dave Knapp, Jon Thaler and the other members of the SSC DAQE Design Environments project for their contributions. This research is supported by the Texas National Research Laboratory Commission, project #047, by National Science Foundation grant CCR-9108931, and by DOE grant DE-FG02-91ER-40677.

## References

- [1] Eric J. Golin and Annette Feng. A visual hardware description language. In *IFIP Conference on Hardware Description Languages*, April 1993. (to appear).
- [2] *VHDL Language Reference Manual (IEEE Standard 1076-1987)*. New York, March 1988.
- [3] *SDC Technical Design Report*, April 1992.
- [4] Gateway Design Automation, Lowell, MA. *Verilog-XL Reference Manual, Release 1.5a*, 1989.
- [5] *MODSIM II Reference Manual, Release 1.7*, January 1992.
- [6] *ObjectStore Reference Manual, Release 2.0*, October 1992.
- [7] Moon Jung Chung and Sangchul Kim. An object-oriented VHDL design environment. In *Proc. 27th DAC*, pages 431–436. ACM/IEEE, 1990.
- [8] Rick Spickelmier, Peter Moore, and A. Richard Newton. *A Programmer's Guide to Oct*. University of California, Berkeley Electronics Research Laboratory, August 1990.
- [9] Rajiv Gupta, Wesley Cheng, Rajesh Gupta, Ido Hardonag, and Melvin Breuer. An object-oriented VLSI CAD framework: A case study in rapid prototyping. *IEEE Computer*, pages 28–36, May 1989.
- [10] T. Andrews and C. Harris. Combining language and database advances in an object-oriented development environment. In *Proc. OOPSLA*, pages 430–440. ACM, 1987.
- [11] Hank Korth and Abraham Silberschatz. *Database System Concepts*. McGraw-Hill, Inc., New York, second edition, 1991.
- [12] R.G.G. Cattell. *Object Data Management: Object-Oriented and Extended Relational Database Systems*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1991.
- [13] *VHDL Design Environment (VDE) User's Manual, Version 3.0*, Intermetrics, Inc. December 1990.
- [14] *Vantage Spreadsheet User's Guide, Version 4.0*, April 1992.