Sharing Electronic Design Data Via Semantic Spaces*

Karen C. Davis†
Database Systems Laboratory
University of Cincinnati
Cincinnati, OH 45221-0030
karen.davis@uc.edu

Satish Venkatesan
Intel Corporation
RN4-38, 2200 Mission Clg. Blvd.
Santa Clara, CA 95052
satish@scdt.intel.com

Lois M.L. Delcambre
Computer Science and Eng.
Oregon Graduate Institute
Portland, OR 97291-1000
lmd@cse.ogi.edu

Abstract

Electronic Design Automation (EDA) tools, such as layout generators and simulators, have generally focused on algorithms and techniques for hardware design. Data management aspects have not been emphasized, but the volume of data, heterogeneity of data formats, and the evolution/proliferation of tools have made data modeling and data interchange increasingly important research issues. The data sharing problem stems from the fact that related EDA tools are often used in various sequences to manipulate and annotate a single design. In a typical design environment, tools use file-based data storage, with limited data modeling capabilities, and primitive or non-existent query facilities. In order to support current tools, we wish to preserve the semantics of existing hardware description languages in rigorous data models; we propose to capture each existing language in a semantic space *model. We view data interchange as a query against one semantic space that produces objects, i.e., query answers, in another semantic space. We define an integrating meta-model, the* meta-space, *and also define general query operators for transforming objects between semantic spaces. These query operators define both the intension and extension of a query result; the transformed data is described in the type system of the meta-space, thus providing explicit semantics for the shared data. Our modeling approach supports advanced and evolving applications, such as hardware/software codesign, through the ability to retrieve data resident in individual semantic spaces, as well as to share data in semantic spaces from different EDA sources.*

1 Introduction

An Electronic Design Automation (EDA) environment can be characterized as a collection of tools that produce and consume data representing hardware designs. For example, a behavioral specification of a design can be used as input to a simulator that in turn produces data about how the design performs under certain conditions. Simulation data can then be back-annotated into the design specification as further detail about the design. Backannotation is typically accomplished by an EDA tool that augments a hardware design with additional information. The tool's input usually consists of several files with different formats. The main drawbacks of this approach are that

- the semantics of data integration (e.g., annotation of a design with timing data) is embedded in the tool, and
- each language/format is file-based and thus its associated tools lack query processing capabilities.

A goal of EDA frameworks is to allow tool interoperability [1]. At the core of such a system is a repository (or repositories) of design information. Designs are commonly specified in a hardware description language such as VHDL [2]. VHDL has widespread use[1] in primarily file-based environments. In addition, other related languages such as SDF [6] for timing data are also used in data files. Even though various object-oriented database systems have been proposed for EDA [10, 3, 5, 20], in practice the vast majority of design data resides in files, not databases. Existing CAD database research does not consider hardware description languages, and does not focus on specifying integration semantics between related EDA modeling languages/formats. Researchers have begun to examine the ways in which a design specification language influences data management [19]. In this paper, we explore one of the options, modeling data specified using standard EDA hardware description languages in a database system. We focus on sharing data objects between different EDA languages/formats using our database model. We represent objects whose source can be various hardware description languages in our database model. The main contribution of this research is the transformation of objects represented in one language to another through the use of query operators with formally-defined semantics. This means that a designer can easily work with a wide variety of

*Research partially supported by NSF Grant IRI-9210200.

†Author for correspondence. Phone: 513-556-2214. Fax: 513-556-7326.

[1]The VHDL International organization has over 3000 technical members and over 20 corporate members; industry analysts estimate that VHDL holds 53.8% of the market share for hardware design languages, and is expected to have a compound annual growth rate of 28% (http://www.e2w3.com/vi). It is estimated that 30% of US Electrical Engineering departments teach VHDL in their undergraduate curriculum; an extensive education program for VHDL is planned under the ARPA RASSP Education and Facilitation Efforts program (http://rassp.scra.org).

tools without worrying about data transformation and be sure that the semantics of the objects are correctly preserved.

In our work, each hardware description language employed in an EDA environment corresponds to a *semantic space*. Our approach to modeling EDA data captures and preserves different (but related) semantic spaces and allows sharing of objects between spaces in a formally defined way. Users of a particular language/format can then retrieve objects from a semantic space using a query language; tools leveraged to use the database can issue queries to find objects of interest rather than having to include file parsing/searching algorithms in addition to their EDA algorithms.

Our model and query language have been realized in an implementation of a database system called *Odyssey* [15]. File-based tools can also use *Odyssey* since a parser and composer are included in the system architecture to input and output formatted files, respectively. *Odyssey* has been integrated with several state-of-the-art EDA tools for hardware/software codesign and cosynthesis [11, 15, 4, 17]. *Odyssey* processes queries issued by a tool that partitions system specifications into hardware and software specifications, and another that uses case-based reasoning for component reuse.

In this paper, we define a meta-space for representing the semantics of inter-related modeling languages. The meta-space allows for user-defined types and instantiation of the types (collections) with respect to the semantics of individual semantic spaces. The dashed lines indicate that objects can be migrated between spaces via queries or other transformations. Our query language is defined over the meta-space; it allows retrieval of objects defined within a semantic space and from different spaces.

Section 2 describes an EDA application where data sharing semantics are currently embedded in tools. Tools are responsible for parsing VHDL and SDF files and combining the data into annotated VHDL files. As an alternative to current practices, we migrate data between semantic spaces, VHDL and SDF, using formal, tool-independent transformations. The semantic spaces and the meta-space are introduced in Section 3. Our query algebra and examples are described in Section 4. Contributions of this paper and directions for future work are discussed in Section 5.

2 Data Sharing Example: VHDL and SDF

A typical Application Specific Integrated Circuit (ASIC) design process [18] is illustrated in Figure 1. Tools are shown in ovals and the input/output is labeled on the directed arcs. The initial input is a register transfer level (RTL) VHDL design, and the eventual output is a VHDL circuit model backannotated with SDF delay information. The process occurs via transformations carried out by the tools. A synthesis tool transforms the RTL design into a VHDL circuit model file, a netlist file, and a layout description. These three files serve as input to an ASIC delay calculator tool. The calculator tool produces a Standard Delay Format (SDF) file. The VHDL circuit model and SDF files are input to a simulation tool that simulates the activity of the design on input data and produces simulation results. The simulation tool has several functions, including (1) parsing SDF and VHDL, (2) elaborating VHDL (e.g., expanding parameterized components into instances of those components), and (3) backannotating the VHDL design with delay information. Note that the tool must perform data integration activities in order to fulfill its primary function of simulation. Any tool that reads and combines data from multiple distinctly formatted sources has embedded data integration semantics.

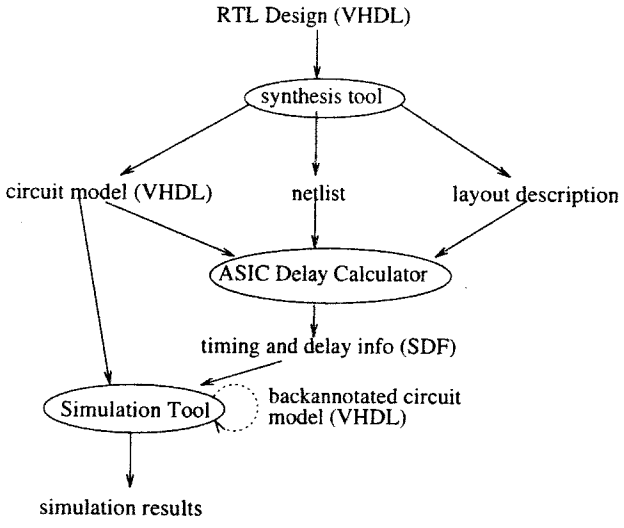

Figure 1: Typical ASIC Synthesis Design Process

We propose to move some of the data integration semantics into the data modeling realm, thus relieving tools of reinventing this functionality in application code. We accomplish this by formally defining the semantic space for each relevant EDA format/specification language, and defining transformations between spaces. The query language described in Section 4 defines general transformations. Customized transformations can also be defined over the model. Using the model discussed in the next section, we would integrate VHDL and SDF via a query, resulting in an object in the VHDL$^+$ semantic space. In related research, another transformation produces VHDL with embedded annotations from VHDL$^+$ [13].

Figure 2 depicts integrating instances in the VHDL and SDF semantic spaces via a query. The VHDL semantic space depicts an instance (*Full Adder*) of a VHDL Architecture; the full adder contains two instances (*ha1*, and *ha2*) of half adder components. The SDF semantic space depicts instances containing delay data (computed by a delay calculator tool) for the full adder object in the VHDL space. It contains delay information for design regions (component instances in the VHDL space). These instances from the VHDL and

SDF spaces are integrated via a query to generate an *Annotated Architecture* type and its instances in the VHDL$^+$ space. While the picture depicts data integration between instances, the query language facilitates integration at the collection level. Intensional and extensional semantics of query operators are described in Section 4, along with the query that produces the *Annotated Architecture* type.

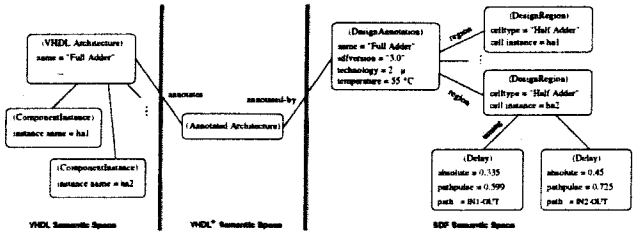

Figure 2: Combining Collection Instances from Different Semantic Spaces

3 Model

In this section, an overview of the meaning and organization of semantic spaces is given. Intuitively, a semantic space consists of a set of types and direct subtype relationships. OMT [7] notation is used informally here to describe the types and relationships between types in a semantic space. Types are shown with rectangular boxes; an instance of a collection is shown with a rounded-edge rectangular box. Collections are either implicit (all instances of a type) or are explicitly formed via a query expression. Types may have multiple extents since a subset of a collection may have the same type as the collection.

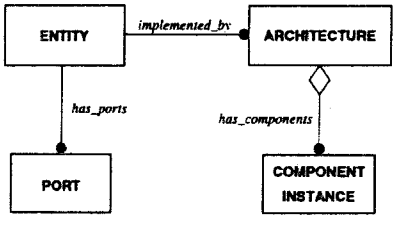

Figure 3: Excerpt of Types in the VHDL Semantic Space

In Figure 3, an excerpt from our VHDL semantic space is given. The full model (over 70 entities) is described elsewhere [14]. For purposes of the example here, consider the primary types of VHDL to be *Entity*, *Architecture*, *ComponentInstance*, and *Port*. Three relationships are shown, (i) an *Entity* is *implemented_by* one or more instances of an *Architecture*, (ii) an *Entity* is related to an its input/output ports by the one-to-many relationship *has_port*, and (iii) an *Architecture* may consist of zero or more connected components. These types describe VHDL design units that can be instantiated to form valid designs. The excerpt identifies a semantic space that expresses meaningful VHDL objects and their relationships. Instances of this model constitute input and/or output for VHDL-based tools.

The model satisifies our requirement of maintaining existing EDA functionality because it captures the core abstractions of design interface (*Entity* and *Port*) and hierarchical design implementation (*Architecture* and *ComponentInstance*). VHDL can be input and output from the database, and tools that use it need not be aware that their data is coming from or going into a database. Two questions arise:

- how can advanced tools that need query capabilities utilize the database, and
- how can non-VHDL EDA data be incorporated into the database?

Note that meeting these requirements should not interfere with the use of VHDL data by existing applications.

The challenge here is to capture all of the related models (e.g., VHDL and other similar languages) in one meta-space so that semantically meaningful queries over different semantic spaces can be expressed. Our approach is to define the meta-space as well as the individual semantic spaces in our schema and use a query language that defines the intension and extension of queries over collections. In this way, data sharing can be accomplished between different formats. Each query answer will include a type from a semantic space as the intension and a set of instances as the extension. Transformations that add attributes to VHDL designs create objects in the VHDL$^+$semantic space; this data can be transformed to annotated VHDL designs, thus making new data usable by VHDL-based tools [13]. We also support transformations that hide VHDL attributes in order to support use of the data by tools that are not VHDL-specific. Designs that are missing intrinsic VHDL attributes belong to the semantic space VHDL$^-$. Currently, the semantics of similar transformations are either embedded in tools or are non-existent.

The meta-space for organizing and relating semantic spaces is given in Figure 4. The purpose of the meta-space is to describe features of the entities in the semantic spaces and provide a uniform basis for query language semantics. All semantic spaces have an intensional and extensional component. The primary entities are *Intension* and *Extension*. *Intension* models the type hierarchies of semantic spaces; *Extension* models the collections of objects that are resident in the database. An *Extension* instance may be described by a *query expression* and may be a superset/subset of other collections. *Intension* is further divided into *SDF Space* and *VHDL Space*, the two hardware description formats defined for our prototype. The actual intension for *SDF Space* appears in Figure 5 and is discussed below. All of the types in SDF are instances of *SDF Space*.

A very small but representative portion of the *VHDL Space* [16, 14] is shown in Figure 3. Characteristics of the intensional component of *VHDL Space* are further elaborated in Figure 4. The types in the intension for *VHDL Space* can be further described as *Annotatable* and *Non-Annotatable*. Intuitively, annotations are external characteristics of a design that result from analyzing its structural or behavioral specifications, usually via a tool. In general, non-primitive types are annotatable. All of the entities shown in Figure 3 are instances of *Annotatable*. *Non-Annotatable* types, such as *VHDL Built-in Type* and *VHDL User-defined Type*, can be used to annotate annotatable types.

Intensional types that model EDA domain-specific constructs are instances of *VHDL Domain*, and types derived from these types are *VHDL Derived*, either in $VHDL^+$ or $VHDL^-$. If intrinsic VHDL properties are removed from a *VHDL Domain* object via a query operator, it becomes a $VHDL^-$ object; adding properties makes it a $VHDL^+$ object. The entities and relationships depicted in Figure 4 are formally defined as semantic domains in our denotational definition.

Additional semantic spaces can be defined for other data models or formats. For example, a Standard Delay Format (SDF) [6] model appears in Figure 5. The primary types supported by SDF are *DesignAnnotation*, *DesignRegion*, and *TimingSpecification*. A *DesignAnnotation* denotes the design being annotated; it consists of annotations for *DesignRegion* where each region may be a component of the design. The annotation data depicted here are timing specifications; a specification may take various forms such as delay, timing checks, and timing environment information. An SDF design annotation abstraction corresponds to a VHDL entity and architecture; a design region to a component instance in VHDL.

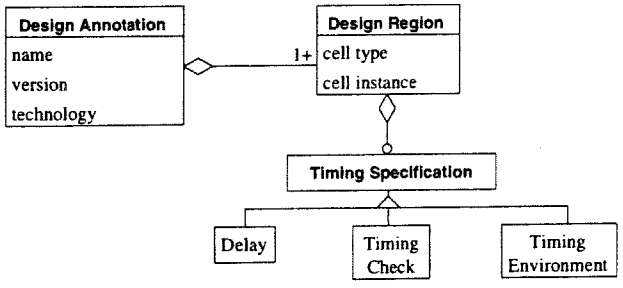

Figure 5: SDF Semantic Space

A VHDL object annotated with SDF data can be output as a VHDL object (with SDF data as embedded annotations), and is thus usable by VHDL-based tools.

4 Query Algebra

This section describes our query algebra and illustrates it with some example queries. The query operators are used to share data across semantic spaces, in addition to allowing queries to be expressed over individual semantic spaces. The query operators facilitate retrieval of designs based on search criteria, where this capability was not previously possible in a file-based environment. In addition, the ability to share data between formats is now accomplished by formally defining the semantics of data integration, where this capability previously existed only in application software.

4.1 Operators

The input and output of queries are collections; a collection has an intension (type) and an extension (set of members). All operators create new collections. The **project**, **extend**, **union**, **intersect**, and **restrict** operators create a new intension if necessary. The definition of the query language infers a query result's intension from the intension of the input and the query operator.

The **select** operator is a set-manipulating operator that returns a subset of the members from a source collection. The operator has the syntax <*output_collection*> := **select**(<*predicate*>) <*input_collection*> where <*predicate*> is a function on the source collection and its associated intension. The semantics of select are to collect all the input members satisfying the predicate into the output. The output collection has the same intension, i.e., the same type, as the input collection.

The **project** operator derives an intension from an existing intension; the properties to be preserved are specified as an argument. The new intension is a supertype of the input intension. and may belong in a different semantic space depending upon its list of properties. For example, if the input intension is in the VHDL semantic space and some of its builtin properties are missing in the output, the resulting intension will be in the $VHDL^-$ space. The set of members in the output collection remains the same.

The **extend** operator defines a new intension by adding a new property to an existing intension. The annotation argument to extend may take one of two forms. In one form, the property being added takes values from existing data. The annotation is specified as a query expression; values are derived from existing data. The *Annotated Architecture* view illustrated in Figure 2 is defined using this form; this **extend** operator is similar to Scholl's extend [8]. In the second form, the annotation definition consists of only a property name and type; values are provided later. This is a schema evolution operation with both the new and old intensions and extensions being maintained. The old intension continues to support existing applications while the new intension is available for further processing.

The **restrict** operator limits the domain of a property to a subtype of its existing domain.

The **union** operator returns a set of objects consisting of members in either or both of the input collections. The resulting intension is the lowest common supertype.

Members of the output collection of the **intersect** operator consist of objects that are in both the input collections. The intension of the result is the greatest

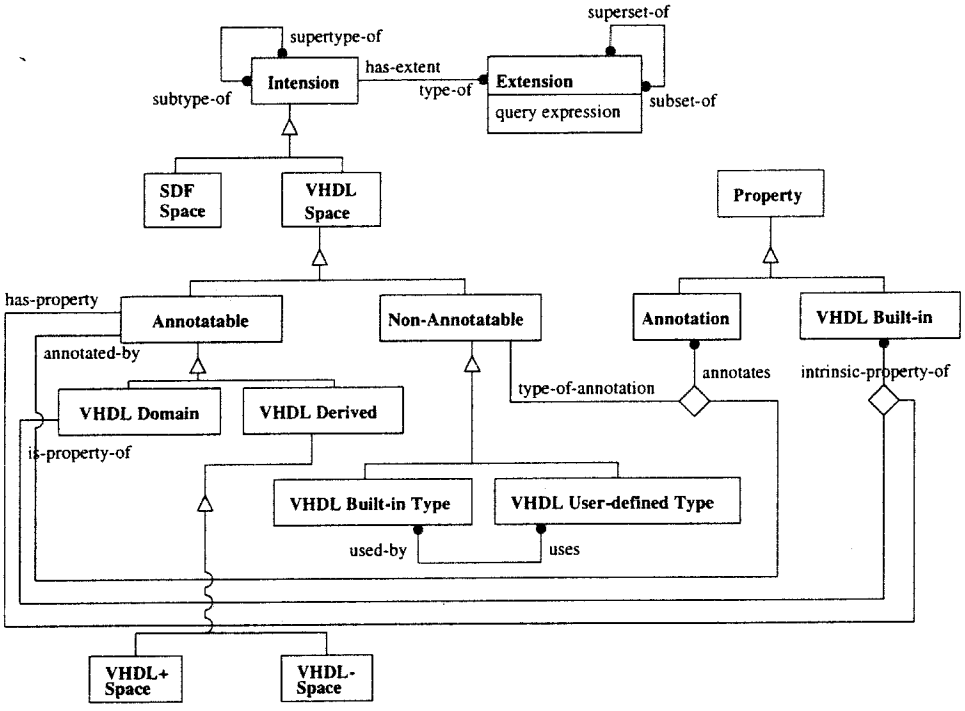

Figure 4: Meta-space for SDF and VHDL

common subtype of the intensions of the input collections.

The **rjoin** operator performs a reference join between the input collection and the members of the codomain of one of its properties.

The **flatten** operator operates on sets of sets and returns a set of objects. It is similar to the flatten operator in Shaw and Zdonik's algebra [9].

Example 1: Figure 2 depicts integrating instances in the VHDL and SDF semantic spaces via a query. The *VHDL Architecture* and *DesignAnnotation* instances are members of collections *ArchCollection* and *TimingDataCollection*, respectively. The following query creates an *AnnotatedArchCollection* consisting of *Annotated Architecture* instances:
AnnotatedArchCollection = **restrict** (timing = select (self.name == timing.name) TimingDataCollection) ArchCollection

The query (i) extends objects in *ArchCollection* with a *timing* attribute, and (ii) specifies attribute values via a join (based on *name*) between *ArchCollection* and *TimingDataCollection*. *AnnotatedArchCollection* may now be queried to retrieve VHDL architectures based on timing data. Other EDA tools may perform different analyses over the same VHDL architectures and accordingly define other tool-specific views. Both intensional and extensional results of queries are automatically inferred.

Example 2: Figure 6 illustrates example queries issued by synthesis and simulation processes. The goal of the synthesis process is to generate a structural description from a high level behavioral specification; both the structural and behavioral descriptions are in VHDL. The synthesized design must satisfy constraints on, among other things, area, power, and timing. The goal of the simulation process is to verify functional correctness of designs as well as to obtain accurate estimates on timing. Under the methodology used for this example, SDF is used to model timing data and constraints on area and power are specified using simple attributes.

Database activities of the simulation process are to (i) retrieve and classify VHDL designs as simulatable, and (ii) annotate them with analysis results (SDF space); these are depicted in queries ②, ④, and ⑥. Database activities of the synthesis process are to (i) identify designs suitable for synthesis, (ii) annotate them with constraints on *area* and *power*, and (iii) retrieve designs based on constraints. The first two activities are depicted in queries ①, ③, and ⑤.

In this example, both simulation and synthesis processes share VHDL data but analyze and annotate them separately. Annotations from simulation and synthesis are then integrated via queries to obtain query answers in Ⓒ and Ⓓ. The intension for collection *SimSynArchitectures* (Ⓓ), *SIM-SYN-ARCH*, is in the VHDL$^+$ semantic space; *SIM-SYN-ARCH* is derived from *ARCHITECTURE* (VHDL space) by enhancing it with *timing* (SDF space), *area*, and *power* properties. Projecting *power* and *area* properties from *SIM-SYN-*

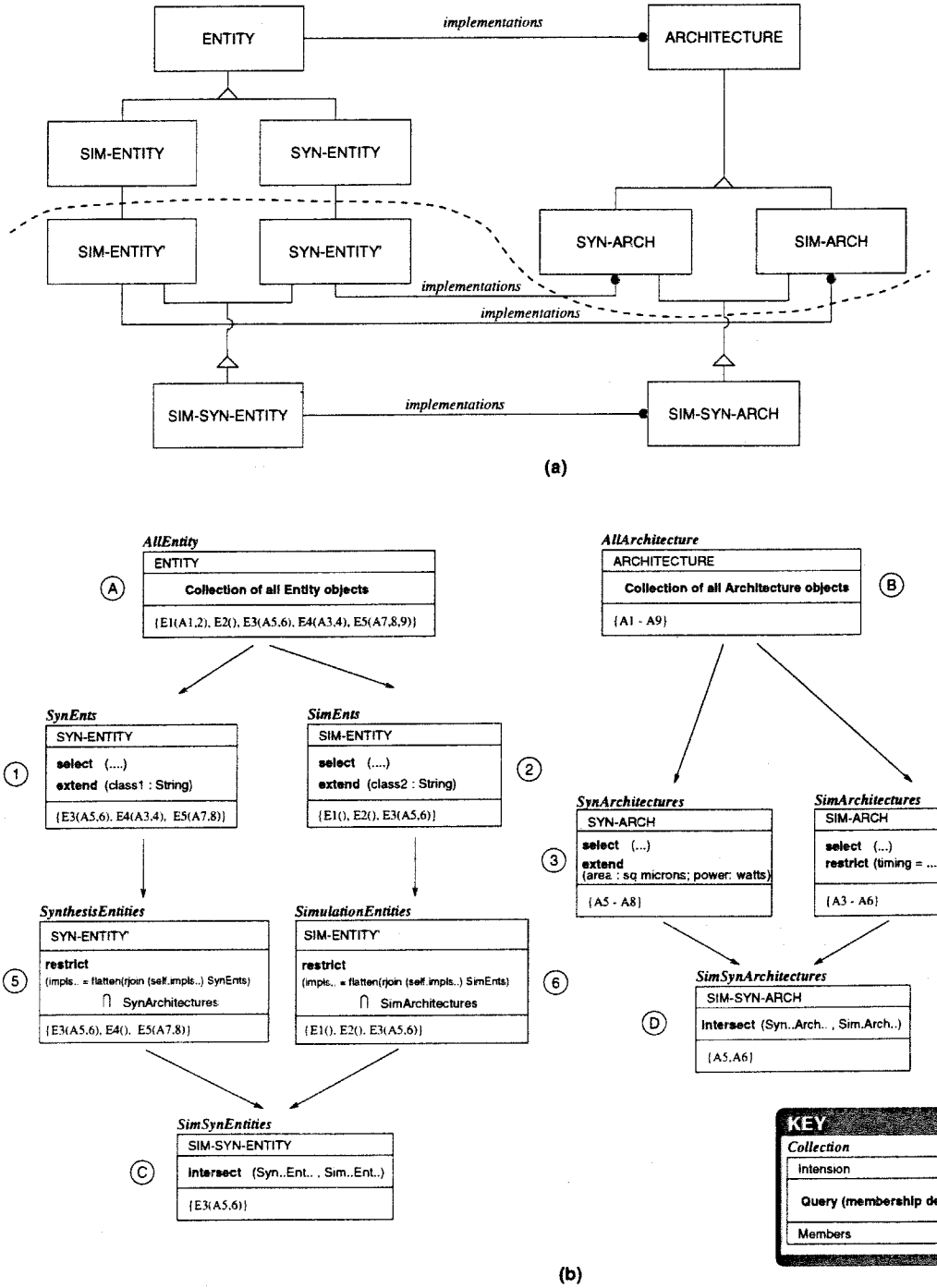

Figure 6: Example: Extend, Restrict, and Intersect

ARCH would generate a result in the VHDL$^-$ space.

Tools can now query collections © and Ⓓ to retrieve designs satisfying timing as well as area and power requirements; this can now be accomplished without parsing multiple files in VHDL and SDF formats. Matching designs are input to subsequent processes such as floorplanning and layout generation.

Multiple inheritance conflicts arising from redefinition of properties (such as by **restrict**) are automatically resolved, an algorithm and proof of correctness are available [12].

5 Conclusions

In this paper, we examine an application area, electronic design automation (EDA), and discover opportunities for introducing database technology into the EDA software environment. In particular, we address the lack of query processing capabilities, and the proliferation of data formats. We propose a meta-model that relates models of EDA data formats (called semantic spaces here). To allow queries over semantic spaces, we define query operations as general transformations over semantic spaces. Contributions of the research have two aspects: one is the fundamental research in data modeling and query language semantics, and the other is the practical impact that the research has for EDA environments.

Contributions related to modeling are summarized below.

- We propose a semantic space model for formally capturing and preserving the semantics of related EDA file-based modeling languages/formats.
- The meta-space defines a clear separation of types (intension) and collections (extension), and it supports multiple extents per type.
- The semantic space model provides the basis for formally defining transformations for migrating objects from one semantic space to another. In other words, importing and exporting objects defined using related models is automatically accomplished via the semantics of the operators used to define object collections.

The impact of the research on EDA tools and environments is described below.

- Query capabilities are added to an environment previously lacking them.
- Existing tools are transparently accomodated.
- Evolving applications can be supported by queries or customized transformations; new semantic spaces can be defined and incorporated as well.
- Data integration semantics are formally defined in a tool-independent manner.

Considerations for future work are outlined below.

- A future extension would be to model the constraints of each semantic space rather than just structure and relationships.
- Query optimization, both over the meta-space and for individual semantic spaces, could be explored.
- We have considered two disjoint semantic spaces in this paper, but we have not addressed the overlap between semantic spaces (such as VHDL and CFI Design Representation) here. We address semantic mapping in an earlier work [16], but not in terms of semantic spaces.
- Currently, semantic spaces are defined by domain experts. Users and tools can instantiate existing spaces, and generate new intension/extension via query operators or other transformations. Automatic or partially automated generation of semantic spaces is a topic for future investigation.

References

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

[2] *IEEE Standard VHDL Language Reference Manual.* New York, NY, 1993.

[3] W. Kim, J. Banerjee, Hong-Tai Chou, and J. F. Garza. Object-oriented Database Support for CAD. *Computer Aided Design*, 22(8):469–479, October 1990.

[4] R. Miller, H. Carter, K. Davis, and S. Venkatesan. Hardware/Software CoSynthesis: Multiple Constraint Satisfaction and Flexible Component Retrieval. In *International Conference on Engineering Complex Computer Systems*, Montreal, Canada, October 1996.

[5] Tapas K. Nayak, Arun K. Majumdar, Anupam Basu, and Santonu Sarkar. VLODS: A VLSI Object Oriented Database System. *Information Systems*, 16(1):73–96, 1991.

[6] Open Verilog International, Los Gatos, CA 95032. *Standard Delay Format Specification, Version 3.0*, May 1995.

[7] James Rumbaugh et al. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, NJ 07632, 1991.

[8] Marc H. Scholl, Christian Laasch, and Markus Tresch. Updatable Views in Object-Oriented Databases. In *Proceedings of the 2nd International Conference on Deductive and Object-Oriented Databases*, pages 189–207, Munich, Germany, Dec 1991.

[9] G.M. Shaw and S.B. Zdonik. A Query Algebra for Object-Oriented Databases. In *Proceedings of the 6th International Conference on Data Engineering*, pages 154–162, Los Angeles, CA, Feb 1990. IEEE Computer Society.

[10] E. Siepmann and G. Zimmermann. Object-Oriented Datamodel for the VLSI Design System PLAYOUT. In *Proc. of the 26th ACM/IEEE Design Automation Conference*, pages 814–817, Las Vegas, NV, 1989.

[11] Ranga Vemuri, editor. *Proceedings of the COMET Project Review Meeting*, Cincinnati, OH 45221-0030, Sep 1995. University of Cincinnati.

[12] Satish Venkatesan. *Database Modeling for Electronic Design Automation Environments.* PhD thesis, Electrical and Computer Engineering, University of Cincinnati, Cincinnati, OH 45221-0030, January 1996.

[13] Satish Venkatesan and Karen C. Davis. Attribute Management in a DBMS for VHDL-based CAD Environments. Technical report, University of Cincinnati, 1995.

[14] Satish Venkatesan and Karen C. Davis. Design of a DBMS for VHDL-based CAD Environments. In *CHDL'95, IFIP International Conference on Computer Hardware Description Languages*, pages 539–544, August 1995.

[15] Satish Venkatesan and Karen C. Davis. Odyssey: An Electronic Design Automation Database. In *2nd International Conference on Applications of Databases*, pages 147–157, Santa Clara, CA, December 1995.

[16] Satish Venkatesan and Karen C. Davis. A Metamodel and Semantic Mapping Methodology for Hardware Design Data Management. *Journal of Integrated Computer-Aided Engineering, special issue on Integrated Product and Process Data Management*, 3(1), 1996.

[17] Satish Venkatesan and Karen C. Davis. Flexible Component Retrieval. *Current Issues in Electronic Modeling, Special Issue on Hardware/Software Co-Design and Co-Verification*, 8, December 1996.

[18] VITAL. *VITAL: VHDL Initiative Towards ASIC Libraries - Model Development Specification*, v2.2b, March 1994.

[19] Flávio R. Wagner. Design Management Requirements for Hardware Description Languages. In *EURO VHDL 95*, 1995.

[20] Flávio R. Wagner et al. Design Version Management in the STAR Framework. In M. Newman and T. Rhyne, editors, *3rd IFIP Workshop on Electronic Design Automation Frameworks.* Elsevier Science Publishers B.V. (North-Holland), 1992.