Design of a Tool Interface for Integrated CAD-Environments

Uwe Hunzelmann, Wolfgang Wilkes, Gunter Schlageter

University of Hagen

Abstract

The tool interface which provides mechanisms for tools to implement their data handling facilities on the basis of a common data schema is a key element of a CAD framework. This paper describes a tool interface which has been developed and implemented in the DASSY project. It decouples the CAD tools from the underlying database, and the chosen architecture with an integrated main memory database allows the adaption of the tool interface to different databases.

1. Introduction

Due to the increasing complexity of circuits the design process can only be performed by the use of CAD tools. Usually, the tools are tailored for a specific phase of the design process (e.g. logical design, physical design, or electrical simulation). Thus, for a single design a lot of different tools have to be used, and each tool gets input data from other tools, and its results are again used by several other tools. Thus, it is of major importance to support the interchange of data between different tools.

In most design environments, the data flow between different tools is based on the exchange of files. But the advantages of an integrated database system as the basic means for data handling and data exchange are becoming more and more obvious. Databases support an integrated management of data thus avoiding the consistency problems of storing data redundant data, they provide transaction mechanisms to ensure the synchronization of users and the correctness of data, and they provide logical access operation to the data and enable the application programmer to implement his data handling mechanisms without looking into the physical details of files.

Unfortunately, in many cases it is not possible to directly use the existing commercial (in most cases relational) databases. They do not provide complex object structures as are required for design data which leads to representational (difficult to map design data to a relational schema) and efficiency problems (many expensive join operations). Thus, there are a lot of research efforts in the database area towards more appropriate database mechanisms[5],[6],[7]. At the moment it is unclear which database system and which approach is most appropriate for use in a design environment. Therefore, the DASSY project developed a basic tool interface to provide sufficient structures for the handling of design data. The tool interface is used as a bridge between the tools on one side and (one or more) databases on the other side.

The tool interface is based on a data model called DaDaMo (DASSY data model). DaDaMo is structurally object oriented and contains concepts from the well known semantic and object oriented data models. Furthermore, it provides an explicit relationship concept thus trying to more closely couple the entity relationship and the object oriented approaches. For data definition purposes the language XDDL was defined with a syntax which is oriented towards the standard information modelling language EXPRESS [8]. For data manipulation purposes, appropriate functions are provided.

This paper describes an implementation of the tool interface which is based on a main memory database. Below this rudimentary main memory database, different external databases may be used to store the data persistently. The change of the underlying database can be done transparently for the user since there exists a well defined internal interface between the main memory database and the external database. Currently, the prototype implementation uses ORACLE as its external database.

The paper is organized as follows: Section 2 describes the underlying datamodel of the tool interface, section 3 discusses the general architecture of the system, the main memory data management and the logical concept for accessing objects in the database. Finally, section 4 briefly describes the TCAD/ECAD design environment which is being built on the basis of the tool interface.

2. The data model

The prototype implementation uses the data model DaDaMo which has been defined in the DASSY project.
The basic elements of DaDaMo are objects, attributes and relationships. One of the main objectives of DaDaMo is to allow the explicit definition of typed relationships which can be viewed and traversed from single objects in a hypertext oriented way, i.e. as links to other objects. Furthermore, by predefining specific relationship types, the concept of complex objects can be integrated in a natural way. In the following, the DaDaMo concepts will be briefly introduced as far as they are relevant here. The interested reader is referred to [2].

**Objects and classes**

Objects are the basic entities in DaDaMo. Similar objects are collected in classes. Classes are defined in the schema and describe the properties of their objects. Each object belongs to (at least) one class. The classes form a sub/superclass hierarchy along which properties are inherited.

DaDaMo distinguishes between two kinds of properties: attributes containing the object specific data, and roles representing relationships to other objects.

```
RELTYPE INSTANCE;
  loc:Location;
  trans:Transformation;
  contains:CELL;
  INVERSE is_contained_in:CELL;
END_RELTYPE;

ENTITY CELL;
  cellName:Name;
  INSTANCE contains: CELL;
  INVERSE INSTANCE is_contained_in:CELL;
END_ENTITY;
```

Fig 1: Class CELL and relationship-type INSTANCE in XDDL (EXPRESS like)

**Attributes**

Attributes are typed, and the domain of an attribute is defined by its type. In addition to the usual atomic attributes (integer, float, char, byte, enumeration), DaDaMo allows the definition of complex attributes. Complex attributes are further structured, and each element of a complex attribute is again either of an atomic attribute type or of a complex attribute type. DaDaMo supports the constructors struct, set, list, static array, and dynamic array for complex attributes.

**Relationships**

DaDaMo supports explicitly the concept of relationships. A relationship associates two objects, and the relationship is existentially dependent on both objects. Thus, no relationship may exist without the objects it associates. Each relationship belongs to a relationship-type which is defined in the schema and describes the properties of the relationship. A relationship-type is mainly characterized by the classes whose objects can be related (typically classes which are located very high in the class hierarchy), and by some attributes of the relationships. Those attributes contain data which cannot be associated with one of the related objects but only with the relationship between them.

A relationship can only associate objects which belong to the classes specified in the corresponding relationship type. By means of the roles these relationships are visible from each object, and the object can access the associated object via the role name (in the same way as it would access its attributes). The role names are defined with the relationship type, but they can be overridden for single object classes.

**Example**

(The redundant specification of role names in the object classes makes it easier to see which rolenames are attached to which relationship types.)

The use of a layout cell within another layout cell (instantiation) can be described by a relationship type INSTANCE between objects of the class CELL. The relative position and the geometrical transformation of the used cell cannot be attached to the cell object itself since it may be used several times in various other cells. Instead, this information can only be attached to the relationship of type INSTANCE (see fig. 1 and 2).

**Operations**

DaDaMo provides a set of functions to access objects in the database. Basically, two categories of access operations are provided:

1. Query operation:

```
    Name
    |-----------|
    | Location  |
    |-----------|
    | Transformation |

    CELI
    |-----------|
    | INSTANCE  |
    |-----------|
```

Fig 2: Class CELL and relationship-type INSTANCE in EXPRESS G like Notation (with a special symbol for relationships)
Retrieval of objects:

1. Query: Retrieve a set of objects according to a query specification.
2. Traversal of result sets:
   Traverse the result set of a query operation.
   
   Navigation through the database can be performed by an appropriate combination of these operations: After the initial retrieval of a root object (resp. of a set of similar root objects), objects can be retrieved which are associated by specific types of relationships to this root object. The result set can be traversed by operations of the second kind, and for each object of this set, further objects can be found by following its relationships. Thus, the operations of DaDaMo support basically two forms of queries:
   - the direct access of objects by means of a usual query specification, and the
   - navigational access to objects by means of "relationship queries" which refer to specific objects as starting points and use role names for qualification.

   Especially the navigational access is most important.

```
RELTYPE CELL_2_PORT;
   has_ports:Port;
   INVERSE belongs_to: CELL;
END_RELTYPE;

ENTITY CELL;
   cellName:Name;
   CELL_2_PORT has_ports: Port;
END_ENTITY;

ENTITY PORT;
   portName:Name;
   INVERSE CELL_2_PORT belongs_to:
   CELL;
END_ENTITY;
```

Fig 3:
Classes CELL and PORT together with relationship-type CELL_2_PORT in XDDL for CAD tools as their usual way to operate on their data.

Example:

Navigation from a cell to its ports (see the respective schema definition in figure 3 (XDDL) and figure 4 (graphical notation):

1. Query: Retrieve all objects of class cell with the required property (e.g. name = 'adder')
2. Set traversal: Traverse the result set, and for each element c execute the following

3. Query: Retrieve all objects of class ports which are associated to c via the role name "has_ports".

3. Architecture of the system

The tool interface implements the data model which was described in the previous section and provides the respective operations to the CAD tools. The CAD tools use these operations to define and manipulate their design data. In the end, the persistent storage of the data is performed by means of an external database management system. One of the major objectives of the implementation was to make the tools independent from the underlying database, and even the interface itself was designed to be as independent from the underlying database system as possible. This was achieved by the implementation of a main memory data manager which stores and manages the data the application is just working with. By this approach it will be relatively easy to switch from one database to another one without the need to change the application programs (CAD tools). The necessary changes have to be done at the interface between the internal and the external databases. The result of this design decision was that all partners of the DASSY project could stick to their favorite database system.

This led to the following general architecture of the tool interface implementation: The tool interface contains its own internal data management system which stores the requested objects in main memory. The application which uses the tool interface for the manipulation and retrieval of its CAD data is linked to the tool interface module. Application and tool interface build a process which is
executed on the application machine. The main memory database communicates via an internal interface (invisible for the application) with the external database which is executed as a separate process and could even run on a different machine.

As described in the previous section, the tool interface has to provide two basic access mechanisms: First the query operation which retrieves a specified set of objects and secondly the traversal of such sets. These operations are realized by means of a cursor mechanism. A cursor consists of two components:

- an object pool, and
- a current position

The object pool contains the resulting objects of a query (it can be filled also in different ways, see below). The application sees exactly one object of the object pool, namely the object at the current position of the cursor. This object can be read or manipulated by appropriate operations. Other operations allow to move the current position over the object pool. Thus, the traversal of the result set is implemented as a movement of the current position of the cursor.

A tool may open several cursors. Each cursor is identified by a unique name which allows the call of the respective access-, manipulation-, and cursor-operations. The object pool of a cursor contains only similar objects which are characterized by two common properties:

1. They belong to the same class \( c \), either direct or indirect, i.e. they belong to \( c \) or to a subclass of \( c \).
2. All objects are projected to the same attributes / role names.

After the definition of a cursor, the respective object pool is empty. There are two possibilities to transfer objects into the object pool: A cursor can be filled by a database query or by a specific insert operation for inserting new objects into the object pool. Also the newly inserted objects have to obey the cursor definition, i.e. they have to have exactly the same attributes as defined in the cursor definition.

It may happen that an object is contained simultaneously in more than one object pool. All changes of this object which occur via one of these cursors are immediately visible in the other cursors. Thus, all objects are contained in a virtual object space, and the object cursors can be seen as windows or views to the object space making the current states of the objects visible.

The values of atomic attributes are stored directly within the object. For complex attributes only a reference to an attribute cursor is stored within the object. An attribute cursor contains all elements of one complex attribute of one object. The attribute cursor consists analogously to the object cursor of an attribute pool and a current position. Differently to the object cursors, attribute cursors can be created automatically by the system. This happens when the result of a query contains objects with complex attributes. In this case, for every complex attribute of each object an attribute cursor is created which will be filled with the elements of the respective complex attribute. The references to the attribute cursors are kept in the objects.

Cursor operations

The tool interface provides the following cursor oriented operations:

- create a cursor, definition of the associated object type
- drop a cursor, i.e. the respective view on the object space is deleted (not the objects themselves).
- move the current position of a cursor to the next/previous element or back to the first element
- Fill a cursor with objects by means of a database query (thereby, attribute cursors are implicitly created if necessary)
- insert a new object into the object pool of a cursor
- delete / update the current object of a cursor
- connect an attribute cursor to the current object of a cursor

Furthermore, there exist some boolean functions, e.g. for querying the number of elements within an object pool.

For the user, a cursor operation has the same effect as a database operation and each update is simultaneously visible via all cursors which contain the updated object. But from an implementation point of view, the operations may only be effective in main memory. An update of an object is not necessarily transferred immediately to the external database. This transfer is directed by the tool.
interface automatically, thus decoupling the update operations of the tool and the actual data transfer to the external database. Of course, the tool may cause explicitly the transfer of all objects to the external database by means of a save operation.

Example:

By means of a short example we will show how the tool interface can be used by the CAD tools. The task of the program is as follows: For every cell with name 'adder', add a test port for each halfadder which is an instance of that adder (e.g. to check in a later step the intermediate results of the halfadders). The following program is written partly in pseudo code and refers to the examples of section 2:

```c
/* open the underlying DataBase.*/
openDataBase(username, userid);
/* Create the needed cursors.*/
OC1 = createCursor("CELL");
OC2 = createCursor("CELL");
OP1 = createCursor("PORT");
/* The existing objects of class CELL with a name "Adder" are loaded from the database*/
getDBObjects(OC1, cellName=="Adder");
/* examine all retrieved Objects*/
while(nextCursor(OC1) != NULL)
{
    /* find all cells with name "Halfadder" which are connected via relationship "is_contained_in" with the actual objects */
    getDBObjects(OC2, is_contained_in == OC1
        && name == "HalfAdder");
    /* if cells have been found, then a Port has to be created for them */
    if (cardCursor(OC2) != 0)
        do
        {
            /* create a new Object of class PORT*/
            createObject(OP1, portName = "TestPort");
            /* connect this object to the actual object from Cursor OC1 */
            updateObject(OP1, UPDATE,
                belongs_to = OC1);
        }
    (nextCursor(OC2) != NULL);
    releaseCursor(OC2);
} releaseCursor(OC1);
releaseCursor(OP1);
closeDataBase();
```

4. Adaptation of tools to the tool interface (DASSY design environment)

Many existing tools use the design data in the following way: Once, at the beginning of the program, a file is loaded which contains the required CAD data. The data contained in the file is transferred into tool specific data structures which are accessed during the execution of the program by means of specific access functions. At the end, the data is again transferred from the internal data structures into the file format and written back to the file.

There are two basic concepts to adapt a tool to the tool interface:

1. A complete integration of the tool and the tool interface (white box integration) can only be achieved by changing the internal access functions of the tool which requires some changes of the tool's source code.

2. The second possibility to integrate a tool with the tool interface is the so called 'grey box integration'. This method does not require to change the source of a tool. Instead, the tool integrator has to build a wrapper for the tool. The wrapper allows the tool to work with the tool interface in the same way as with a file. By means of the wrapper the tool can read and write its data from and to the tool interface using the same file operations as before. Thus, the tool initially reads all its data and writes them back at the end, but the wrapper takes care that the data are read by use of the tool interface, providing the tool with data from the CAD database. During its execution, the tool uses still its internal data structures.

But the first step of the tool integration process, prior to all changes of access functions or to the implementation of wrappers, is the definition of a common database schema which describes all data required or delivered by tools. This step should not be underestimated since it provides the basic resource on which all the following implementation steps have to rely on. The schema development should not be done on a case by case basis, but it should be based on existing, possibly standardized information models ([11], [3], [4]).

To validate the concept and implementation of the tool interface, the DASSY project is building a prototypical design environment which integrates several design tools on the basis of the introduced tool interface. Most of the tools have been developed by the DASSY partners themselves. Therefore the objective was directly integrate the functions of the tool interface into the tools (white box integration).

The DASSY design environment contains tools from the VLSI CAD domain and tools from the TCAD domain (Technology CAD). For the CAD domain, the following tools are integrated:
- the graphical editor ICE and the netlist extractor NESSI from the University of Siegen,
- the multilevel editor VEM from the OCT package
- the test pattern generator C-Test and the CMOS fault simulator FehSim from GMD.

By means of these tools, a design can be performed starting from the graphical entry of a circuit on the schematic level down to the fault simulation of the circuit.

For the TCAD domain some tools are being integrated by the Technical University of Hamburg-Harburg. These tools allow to characterize the electrical behaviour of a semiconductor considering its layout and its fabrication and to transfer this information into the VLSI CAD domain.

The work to build this design environment have shown that DaDaMo is an appropriate means to model and to deal with CAD data which are required in VLSI CAD. Furthermore, it was validated that it is possible to fully integrate CAD tools with a schema independent tool interface.

5. Conclusions

This paper has introduced a tool interface which provides a data model and proper operations to allow CAD tools to access their design data in a common way. Thus, the CAD tools get access to a database and can exchange data with other tools via the common tool interface. The tool interface offers generic operations which obtain the respective object types as a parameter. This is opposed to the proposal of the CFI TSC Design Representation for a schema-specific procedural interface: In the CFI DR PI specific operations are defined for specific object types. Our experiences with the integration of tools have shown that the generic approach is much more flexible which pays, in particular, in case of necessary extensions of the schema for the integration of new tools.

Currently, two implementations of the tool interface exist. The prototype which has been introduced here is based on the external database ORACLE. By using SQL-NET it was quite simple to distribute the design environment over different hardware platforms. A further prototype is being implemented at the University of Darmstadt (GRIS) based on POSTGRES. Furthermore, the procedural interface of the CFI TSC DR has been mapped to the DASSY tool interface thus (grey box) integrating all tools into the DASSY design environment which are adapted to that PI.

The tool interface can be used as a part of a framework, but of course, it is not the full framework. It should also not be regarded as a CAD database since many important aspects of a CAD database are missing, as for instance version management or transaction management. But it is planned to use the tool interface as vehicle to adopt and further develop advanced concepts in these areas and make them usable also in different contexts and projects.

Literature