A Procedural Interface To CAD Data Based On EDIF

T. C. O. Young and H. J. Kahn
University of Manchester

Abstract

EDIF is a data interchange format which currently provides a solution to the problem of data transfer between proprietary CAD systems. This paper considers how a procedural interface to a CAD system may be built upon the underlying data model of EDIF.

It outlines the procedural interface, and describes how a novel implementation has allowed access to, and extension of, the database to be particularly flexible, as well as allowing data management functions to be dynamically incorporated into it.

1 Introduction

The current trend in CAD systems is to move away from a set of stand alone tools, each of which has its own private and unique data structures, towards an integrated system in which the function of data management is the responsibility of the system. Thus the data for the whole design becomes available in a single common form to all the tools in the CAD system. The procedural interface described in this paper defines a set of procedures which may be used by any tool within the CAD environment to access the central CAD database or by a new foreign tool being developed to allow that tool to be easily integrated into an existing CAD system. It discusses the data model which is based on EDIF, the procedural specification and the system implementation.

2 CAD Environments and Tool Integration

A CAD system is a collection of programs (software tools) which can be used to assist the process of electronic design. These tools perform such tasks as schematic capture, browsing, design rule verification, simulation, routing or layout - a disparate set of programs to perform a range of very different tasks. The environment is the software system within which the CAD tools reside. The environment has two basic functions: to allow the tools to be run and to allow data to be input and output from them.

At its simplest, the environment may be the operating system of the host computer. The operating system has no knowledge of the functionality of the CAD tools residing within it or of the data structures manipulated by them. The only data structure recognized by the environment is a file. With no support for data management from the environment, each tool must define and maintain its own data structures. As the tasks performed by different tools vary, so typically do the internal data structures used by each tool. This is particularly likely to be the case if the tools were written by different vendors. Hence the data transfer problem, of how the data output by one tool can be used as the input to another, results.

Currently transfer of data between tools tends to be done on an ad hoc basis using sequential files. The format of each file is designed to perform one specific bridging task between two particular tools. Similarly the task of data transfer between systems, if possible at all, is organized on an equally ad hoc basis, again using bridges written to transfer data between two particular systems. EDIF [1] provides a solution to the data transfer problem by defining a neutral data interchange format. If all tools can read and write files of data in EDIF format then data can be transferred between any two tools.

Loosely coupled systems, which may appear to provide a unified environment, have problems concerning design flow management and data consistency. Design flow management means control of the sequence in which the tools are applied. A loosely coupled environment is a collection of many isolated tools and without system support it is difficult for the user to coordinate them effectively. The data consistency problem arises because the data is fragmented across the local databases of many tools. There is, therefore, no point within the system from which consistency can be imposed on all the design data.

The solution is to move to an integrated CAD environment. This can be closely coupled and possibly intelligent [2]. Closely coupled systems are ones in which the intermediate files are hidden, or possibly replaced by a database, and a rigid design methodology may be imposed, if required. An intelligent system, in contrast, is more flexible. It provides a set of CAD tools and a set of meta-tools to assist with design flow.

A CAD Framework is an environment with the capability to support a sophisticated, closely coupled and intelligent CAD environment. It provides a set of resources with which to build the CAD system. A CAD Framework is analogous
to an IPSE (Integrated Project Support Environment) in the domain of software engineering [3] [4]. Of particular interest to this paper is the Framework support for data storage which allows the CAD environment to be integrated around a central database. The Framework provides a set of procedures to configure and access the system database. This procedural interface forms part of a PTI (Public Tool Interface). For maximum effect the CAD Framework should be open. Its PTI should be in the public domain in order that new tools may be written to its specification and easily integrated into the existing environment. This paper describes a procedural interface to a system database which could form part of the PTI of a CAD Framework.

3 The Data Model

3.1 The Structure of an EDIF File

An EDIF file describes a design as a hierarchy. The whole file constitutes a complex design object which is defined in terms of progressively simpler objects until design primitives are reached. The simple hierarchy is complicated by attributes and references. Firstly, nodes may be qualified by attributes and secondly a node may make cross references across the hierarchy. Finally the relationships between nodes have direction. Thus, the result is a directed acyclic graph.

3.1.1 Data Model Objects and EDIF Keywords

EDIF keywords can be categorized into three types: those representing design objects, those which represent attributes and those which form part of a referencing relation. Examples of design objects are library, cell, view and port; attributes are things such as cellType, portDirection or color; the third category are keywords such as cellRef and viewRef. There is no hard and fast rule about what is an object and what is an attribute. In general, an object is anything which has some meaning in isolation.

A decision must be made about the data model granularity. The basis for selection of granularity is, in part, affected by consideration of eventual database implementation. There is, in general, a trade off between the organizational complexity of the database and flexibility of access. The database described in this paper uses a fine granularity with, in general, a one-to-one correspondence between EDIF keywords and database objects.

3.2 Object Identifiers

Each design object in the database is allocated a unique identifier. An object is always referred to by this internal identifier.

In the system described in this paper, an interpretation of identifiers wider than just object identifiers is used. In addition to allocating a unique identifier for each abstract object, a unique identifier is allocated for every basic data type, such as string or integer. The advantage of this is that all manipulation within the system is done using a simple basic type of fixed size, with consequent improvement in efficiency in the implementation, and clarity in the interface procedure headings. A subsidiary database, known as the lexical token converter (LTC) [5], performs the conversion between basic data types and internal identifiers.

3.3 Relationships

Four classes of relationships are supported by the database model. These are definition, reference, sibling and attribute.

Definition and reference relations are both parent relations because, via an instance of one of them, a parent object gives a child object a context. A definition relation occurs between a parent and the original definition of a child whereas a reference occurs between a parent and a previously defined child.

A sibling relation occurs between objects at the same level in the hierarchy which have a common context. These objects form an ordered set. Therefore, successor and predecessor relations exist between them. Sibling relations are sometimes redundant and sometimes vital. For example, the ordering of the component ports of an EDIF portBundle must be stored, otherwise it would not be possible to establish the correct connectivity. Figure 1 illustrates the relationships implicit in the fragment of EDIF shown below:

```
(view v0 (viewType netlist)
  (interface (port PA) (port PB))
  (contents (net N (criticality 4)
    (joined (portref PA) (portref PB)))
  )
)
```

![Figure 1: EDIF Relationships](image)

Note that the is-a relation in this figure means 'is an occurrence of' rather than 'is a sub-class of'.

4 The Procedural Interface

4.1 Operations

The basic requirements of the database are that an object can be found and, once found, some operation can be performed on it. The available operations are shown in Table 1. 

4.2 Specifying An Object

The directed acyclic graph structure contains two concurrent types of structure, a network structure and an associative structure. It should be possible to specify an object within...
Either of these two concurrent structures. Network specification means that an object is specified by its position within the graph, relative to other objects. An example here is the query

'find the interface which is a child of view #1'

Associative specification means that an object is specified by a set of attributes. For example, the query

'find the port with direction INPUT and delay 4 units'

It is therefore possible to specify an object by knowing where it is but not what it is, or what it is but not where it is.

### 4.3 Sets and Wildcards

It is possible to give an incomplete object specification by using wildcards. An object which matches a specification is known as a responder. An incomplete specification may result in a set of responders.

### 4.4 Specifying An Attribute Condition

In general, the interface procedures match objects against position and the two basic attributes of type and name. The match may be further qualified by a set of attributes. These are placed in a conceptual device called the attribute bin. This forms a frame against which a basic responder set must be further matched.

### 4.5 The General Procedural Form

The general form of the procedural interface could either be object oriented or operation oriented. If an object oriented style is used then a procedure or family of procedures is needed to perform each operation on each object. If an operation oriented style is used, then a procedure for each operation is required, with the object class passed as a parameter. The advantage of the operation oriented form is that, although the number of object classes is large and developing, the number of operations is small and static. It therefore provides a small number of procedures which remains static as the data model is extended whereas, with the object oriented approach, the number of procedures increases linearly as the data model is extended. The approach adopted for the procedural interface described here is operation oriented. The associative search procedure is given as an example.

```c
idtype fnobject(type, name, nameset, idset, n, max)

This procedure takes two basic attributes as parameters, the type of the object and the name of the object. Either of these may be wildcards. The search may be qualified by a set of further attributes which would be put into the attribute bin prior to invoking this procedure. The procedure returns two sets, the set of responders (idset) and the set of their names (nameset), as well as the set cardinality (n). Max gives the maximum number of responders to be returned. This may be passed as an unknown, in which case the complete set is returned. Note that storage for the responder sets is allocated by the procedure. This is denoted by the double '*' in the C' language form of the procedure heading. The function value returns the identifier of the first object found. A null parameter may be passed for either the nameset or idset parameters in which case that particular set will not be returned. If both are null then the search will be aborted as soon as one responder has been found.

There are many situations where it is known that at most one responder is required. In these situations the above form is rather long, and so a short form is also provided, as shown below:

```c
idtype fnobject(type, name)
```

A more detailed specification of the procedures and the objects defined for the prototype procedural interface is available in [6].

Some examples of how the procedural interface may be used to implement CAD tools are shown below:

**Example One:** The associative query: 'Find the port(s) with a direction INPUT and a delay of 4 units'.

```c
idtype *sidset, *nameset;
int n;

setattributebin(idofstr("portDirection"),
    idofstr("INPUT"), idofstr("port"));
setattributebin(idofstr("portDelay"),
    idolstr(4), idofstr("port"));

fnobject(idofstr("port"), ?,
    *nameset, *sidset, &n, ?);
```

The first call to `setattribute` loads the `attributebin` with an attribute type of `portDirection`, which has the `string` value `INPUT` and is to be tested against any responders of type `port`. The second procedure call to `setattribute` loads an attribute of type `portDelay` and an integer value of 4. Note that the conversions between values and internal identifiers in this example are done within the procedure headings. The `&n` in the
"C" syntax indicates that nameset, idset and \( n \) are variable parameters.

**Example Two:** Creating a data structure for a cell with

\[
\text{cellName} = \text{"nandcell"}, \quad \text{and} \quad \text{cellType} = \text{ripper}.
\]

\[
\text{idtype cellid};
\]

\[
\text{cellid} = \text{crecadobject(libraryid, idofstr("nandcell"), idofstr("cell"));}
\]

\[
\text{setattributebin(cellid, idofstr("cellType"), idofstr("ripper"));}
\]

The **crecadobject** procedure call creates a new database object of class cell, with the object name nandcell. The **setattributebin** procedure then attaches a cellType attribute with a string value of ripper. Libraryid is the internal identifier of the context object. The new cell created is attached as a child to the defining library.

**Example Three:** A navigation. Consider Figure 1. Given that we have the internal identifier of the net, whose netName is "N"; find a port, whose portName is "PA", in the interface of the context view.

\[
\text{idtype viewid, interfaceid, portid, netid};
\]

\[
\text{viewid} = \text{getpath(netid, idofstr("view"), ?, ?)};
\]

\[
\text{interfaceid} = \text{getchild(viewid, idofstr("interface"), ?)};
\]

\[
\text{portid} = \text{getchild(interfaceid, idofstr("port"), idofstr("PA"))};
\]

The call to **getpath** traces the defining path of the net up the hierarchy until a view object is found. The first call to **getchild** goes down a level in the hierarchy, by searching for a child object of the view, of type interface. The second call to **getchild** leads another level down the hierarchy, by searching for a child object of the interface, with type port and name "PA".

5 The Implementation

The procedural interface must be practical to implement in order for it to be useful. Two possible routes to an implementation are a conventional DBMS and hardcoded data structures.

Conventional DBMS packages are designed for business applications where the data tends to be much simpler in structure and transactions shorter [7] [8] [9]. CAD data is much more complex than the data found in conventional database applications. The data schema could be hardcoded but this approach is inflexible. In a system where the data model is still developed, a data driven approach is more useful.

The SBRM (Semantic Binary Relationship Model) [10] is a good candidate for coping with the complexity of CAD data and has been used to produce an implementation of the procedural interface. The SBRM is based on the established theory of semantic nets [11] [12] [13]. In this model the data is defined in terms of classes of entities and relationships. The model also specifies a set of basic constraints which must be specified for each relation. Each instance of a relationship class forms a triple of the form

\[
< \text{entity relationship entity} >
\]

All data can be decomposed into this binary relational form. Figure 2 demonstrates how the following fragment of EDIF decomposes into binary relational form.

![Figure 2: EDIF Decomposed Into Binary Relations](image)

Note that the **is-a** relation again means 'is an occurrence of'.

```plaintext
(view v0 (viewtype netlist)
    (interface (contents ... )
        (port inputl (direction input))
    )
)
```

This graphical representation of the CAD data is very similar to the graphical representation of EDIF data shown in Figure 1. Thus, an implementation of the SBRM data model would allow a very simple implementation of the CAD data schema.

5.1 Metadata

Metadata is a powerful concept in the SBRM. The constraints governing the structure of the database (the data schema) can be represented by triples in a similar way to the design data. The data schema can be expressed by the same data model as the design data and it can therefore be stored in the same database as the design data. The data representing the data schema is data about data i.e. it is metadata. If the data schema is stored as metadata and enforced by generic procedures which operate on the metadata, then it need not be hardcoded into the procedural interface software.
6 Conclusion

This paper describes an interface which could be used as a common tool interface for the integration of CAD tools into a common environment. It has been shown how a generic approach may be used to produce a simple interface to a complex database structure and how that form allows the data model to be developed whilst allowing the interface definition to remain static. The interface has been used to construct some basic CAD tools, namely a browser, an EDIF reader and a simple HILO [14] netlist extractor. The HILO netlist extractor has shown how data in a quite different format can be extracted from the EDIF data model. These programs have also allowed the ability to view the same structure in both associative and network forms, and in particular the set based approach, to be compared with a system based on one structure or the other. The use of metadata allowed the database model to be developed, once a set of generic procedures was available, without having to alter the procedural interface procedures, or to do any further coding in the database implementation. The SBRM system works by heuristic search and would therefore be expected to be slow. However, unlike a conventional relational database system, the tuples are of very simple known format and the SBRM system kernel can be tuned to be very efficient. Currently the EDIF reader program takes 60 seconds to read a 800 line EDIF file (of complex connectivity data) on an Orion Clipper System. This is certainly comparable with a system built on hard-coded data structures. It is of interest to note that hardware has been constructed to directly support the SBRM model used in this implementation [15]. This has reported a speed up factor of 1000 on a Prolog system, based on similar heuristic search methods. A speed increase of similar magnitude could be expected for the system described in this paper.

7 Acknowledgements

The work described in this paper was supported by the UK Alvey Programme as part of the CAD031 UK EDIF Support Project and by the Science and Engineering Research Council. Previous work done by Philips Components Ltd on a procedural interface to CAD data [16] [17], and by the IFS Group at the University of Essex on knowledge representation formalisms, is gratefully acknowledged.

References