A STEP Towards a Computer based Integrated Building Design System

Sun M. and Lockley S. R.
University of Newcastle upon Tyne
Department of Architecture
Newcastle upon Tyne
United Kingdom

ABSTRACT

Building design is a multi-sector and multi-task process. In a design project architects, engineers and other specialists need to exchange information in order to produce a coherent design. These design participants often have different views of the design from their own perspectives. The aim of an integrated building design system is to develop a building data model that integrates all views so that building information can be exchanged in electronic form between the designers and also throughout various design stages. This paper introduces an integrated building design system developed as part of the European project, Computer Models for the Building Industry in Europe. It concentrates on the development of the Data Exchange System which is a central data repository implemented using an object oriented database and ISO STEP technology and it is able to support concurrent engineering, versioning, history tracing and other data transaction management.

1. INTRODUCTION

The Computer Models for the Building Industry in Europe (COMBINE) project is funded by the European Commission to investigate computer applications in the building industry (Augenbroe 1995). Its main aim is to explore an Integrated Building Design System (IBDS) that is capable of supporting the multi-tasking and interactive nature of the building design process.

Data integration and process integration are two levels of integration for an integrated building design system. Data integration refers to the sharing of data by all tools in the system; process integration refers to the intelligent invoking of appropriate tools or system functions based on the flow of design. A consensus view of the COMBINE partners is that a full scale process integration requires a comprehensive model of design methodology and design process which is unlikely available in the foreseeable future. Data integration should be the main focus of research at present. Guided by this principle, a prototype integrated system was developed in COMBINE first phase, between 1990 and 1993, which consisted of several design tools and an Integrated Data Model (IDM). It successfully demonstrated the data exchange between tools through the IDM. However, the implementation of the IDM was done in a very much ad hoc fashion, and data exchange control was rather limited. ISO-STEP physical file format was used as both data storage and exchange media.

In the second phase of COMBINE (1993-1995), data transaction control and system management issues were addressed. One of the main tasks was the development of a Data Exchange System (DES) that will maintain a persistent and consistent building model and will support the data exchange between different tools in the COMBINE Integrated Building Design System. The main functional requirements of the DES include persistent data storage, data exchange control, CAD integration, design versioning and history tracing.

453
2. SYSTEM ARCHITECTURE

The COMBINE Integrated Building Design System is one of a series of computer software Design Tools, referred to as DT's, exchanging data in a common format via a central integrated data model (Flynn 1993). Each of these DTs is an existing software package commonly used in the construction industry. The strategy is to add an interface onto a DT so that it can read and write the standard format building data supported by the central integrated data model and map this to and from the internal data representation of the DT. The ability of data exchange in electronic format would result in added value to the existing design tool applications.

In addition, it was recognised by COMBINE that without the support for existing CAD tools the proposed IBDS would be unacceptable to the real design practice environment. It is therefore decided to modify and enhance existing CAD engines so that they can instantiate, manipulate and view geometric representations of the building model in the DES.

To meet the functional requirements a conceptual system architecture was devised for the DES (figure 1). It consists of the following components.

- **Data Exchange Kernel.** It stores data of a consistent state of a building model.
- **Schema Handler.** The Schema Handler is responsible for importing IDM schema and DT schemas into the Data Exchange Kernel.
- **Data Interaction Manager Off-Line (DIMOFF).** The off-line DTs exchange building data with the Kernel through Data Interaction Manager Off-Line using STEP file as a media.
- **Data Interaction Managers On-Line (DIMON).** CAD tools communicate directly with the Kernel through Data Interaction Managers On-Line.

**Figure 1: DES architecture and its interface with the IBDS**
3. DATA EXCHANGE KERNEL

The Data Exchange Kernel is at the core of the COMBINE Data Exchange System, it comprises a set of databases, one for each building model at the current implementation, and a range of database management services required to support the interaction management functionality.

A DES Kernel database is a persistent data repository for the design support system in the COMBINE context. It not only holds the data describing a specific building but also holds data about:

- the COMBINE Integrated Data Model schema
- the design tools that will be applied to the design of the building
- the design functions that will be performed
- the set of input and output schemas for each Design Tool Function
- the content of the data to be exchanged for each design function
- all versions of the design under consideration
- the history of the design evolution
- the coherency of the current data

Since an Object Oriented Database System with C++ language support, Objectstore, has been chosen for the system implementation, the process of instantiating a Kernel database involves the definition of a set of C++ class for each type of object that is to be stored. Three strategies were considered for the implementation of the Kernel.

1. An early binding solution in which the IDM schema will be implemented as C++ classes.

2. A late binding solution which implements a generic data store and exchange service without depending on any application schema.

3. A combination of an early and late binding implementation.

The first option is at first sight, perhaps the most obvious, as all the entities of the IDM have a one-to-one mapping to C++ classes. The resulted structure is easy to read and object behaviours can be implemented for each entity type to manage its interaction with other entities, i.e. a "Wall" entity could have the behaviour to determine the area of "windows" it contains. Unfortunately this would produce a kernel which is not only dependent upon the schema of the IDM but also has embedded knowledge that was specific to a view of the model. The IDM was not fixed at the start of the COMBINE project. It has been evolving and changes have to be introduced into the model from time to time. In a early binding approach, every time the IDM schema changes re-implementation is needed for the Kernel.

The second option is a late binding approach. The primary function of the data exchange kernel is to store data and exchange data with design tools on request. It does not need to know the semantic of the data, for instance, it could treat a building object and a cartesian point object in the same way. Knowledge about the data resides in the design tool side, interpretation of the data can be done at the point when it is exported to design tools. However, the kernel has to store information to enable the interpretation to take place. In order to do this, apart from the value of the data, information about its type needs to be stored. In this approach, because data typing is not coded at compile time, it is called a late binding approach. The advantage is that the Kernel provides a generic data store and
management service. Its implementation and use do not depend on any particular data schema, and it is able to handle any data model as long as its schema conforms to the ISO/STEP EXPRESS standard.

The main advantage of the early binding is that it defines the data semantics explicitly so that on-line data exchange can be carried out directly. Its disadvantage is the need for re-engineering when the data definition or the schema changes. On the other hand, the advantage of late binding is that it is schema independent and schema is stored as data so that it can be manipulated to facilitate data exchange task. Its main constraint is the lack of explicit definition of the data structure in the Kernel. An late binding solution would work well for the off-line data exchange. Because the exchange takes place through a neutral media, data interpretation has to be performed at both sides no matter how the Kernel is implemented. For on-line data exchanges, the late binding approach introduces extra complications. By definition on-line design tools will access the Kernel directly. They need to know not only the value of the data but also the semantics of the data. For this purpose, the data has to be represented explicitly as what it really is not in an abstract form. A late binding Kernel will be not able to provide this support. Therefore, an alternative solution, a combination of early and late binding, was investigated. The aim is to overcome the disadvantages of both approaches while keep their respective advantages. Two key technologies were used to achieve this aim, they are the ISO Standard Data Access Interface, SDAI for short (ISO 1993) and Objectstore database metaclass protocol (Object Design 1994).

The aim of SDAI is to provide an international standard specification for a functional interface to application data whose structure is defined using EXPRESS. To achieve this, SDAI defines three schema, dictionary schema, session schema and abstract data type schema. The dictionary schema defines SDAI data dictionary which is information of other data schema; session schema defines the structure for the management of an SDAI session or an interaction process with a data repository using SDAI protocol; and abstract data type schema is the form in which data is manipulated. In addition, SDAI also specifies functional behaviour of SDAI services as well as programming language bindings. Once the development process started, it was realised that the SDAI scope is well beyond the requirements of the DES Kernel. Strictly following the SDAI specification would not be necessary. Therefore a selective implementation policy was adopted towards the SDAI specification, furthermore, necessary changes were made since the standard itself is still evolving. Figure 2 illustrates a simplified SDAI session and data dictionary schemas implemented inside a Kernel database.

At the root of the database there is a model class and a schema class. The schema class holds the IDM schema initialised by the Schema Handler. The model class holds instances of actual building entities provided through the interaction managers. Typically these will be conveyed to the DES in a STEP physical file. This actual data is held in the model_contents class. Model_contents contains a set of entity extents in its populated_folders. An entity_extent is a link between the EXPRESS definition of the entity and all the instances of that particular entity type in the model. For example there would be an entity_extents containing all instances of the door type, window type etc. Through this link between the EXPRESS definition and the actual instances the Schema Handler and the Interaction Managers can interrogate and manipulate not only the content of the data model in the Kernel but also the EXPRESS structure of the model.
Figure 2: Simplified SDAI implementation in the Kernel

Figure 2 only shows limited selection of SDAI entities, in fact, most of the SDAI data structure and part of its C++ binding functions have been implemented. Full explanation of the SDAI is provided in ISO 10303-Part 22 (ISO 1993).

In Objectstore databases the schema information for the databases and applications is stored in the form of objects that represent C++ types. The metaobject protocol is a library of classes that enable runtime access to schema information and dynamic generation of classes. The DES utilises this feature to create C++ classes for the IDM object at runtime using the IDM schema information stored.

In addition to the SDAI objects and IDM objects, the DES Kernel contains a third type of objects, they are for the purpose of managing the data exchange. These include system_manager, design_tool, design_tool_function and locker. Their relationship with other types of objects is shown in figure 3.

At the top of the structure is the system_manager class, this defines the main Figure 3: Data exchange management schema
behaviour of the interaction managers and provides the link between the IDM schema, model contents and the design tool which make up the COMBINE IBDS. The design tools are created in the DES Kernel during its configuration process by the interaction managers. Each design tool comprises a set of design_tool_functions which represent the task a specific tool will perform. The same software tool can be used for different functions, for example the same energy simulation package could be used to evaluate over-heating risk or insulation requirements. Each design tool function has a data input requirement, defined by its input_schema and a data output, defined by its output_schema.

Finally the locker class is required to efficiently manage the exchange of data between the Kernel and design tools. Its purpose is to store the information about which entity types have been checked out of the database for alteration and to prevent two design tool functions trying to concurrently change the same data.

4. SCHEMA HANDLER

The schema handler is an Objectstore database client which encompasses the functionality needed to work with schemas defined in ISO EXPRESS language. It provides the following services to the interaction managers:

- Transformation of schema between textual form and database instances,
- Initialisation of a DES database structure in accordance with the IDM schema,
- Compilation and validation of the design tool function's sub-schemas.

Figure 4 illustrates the components and system architecture of the COMBINE Schema Handler. Using this architecture database initialisation is performed in three stages.

First step for initialising a Kernel database is to translate the EXPRESS file into STEP physical file format with SDAI data dictionary as the underlying schema (figure 5). It is achieved by using a purpose built tool developed based on the NIST STEP Toolkit, which checks the validity of the EXPRESS schema in term of EXPRESS language syntax.

Figure 4: Schema Handler Architecture
In the second stage the meta representation of the EXPRESS schema is converted into SDAI C++ objects by a STEP parser and stored in an Objectstore database.

At the third stage, the Objectstore Metaobject Protocol is used to create C++ classes for the IDM entities. During this process, entities of the IDM schema are translated into C++ classes and entity attributes are created as class data members. At the end of this stage a DES Kernel is fully initialised.

The mechanisms for processing Design Tool sub-schemas are identical to the first two stages of database initialisation. However, there is no need to generate the C++ code as there is no requirement to have partial views of the model in C++, this is only required in the EXPRESS/STEP representation. Instead the third stage is to validate the subschema against the IDM schema to ensure it is a true subset. Once the sub-schema has been parsed and converted into the SDAI objects representation it is stored as part of the data of its design_tool function object. During the life cycle of a DES Kernel, the schema for a design tool function only needs to be processed once.

5. DATA INTERACTION MANAGER OFF-LINE

Off-line design tools operate independently from the Kernel. They could be separate programs on the same computer, or they might reside on different computers, even at different locations. The data exchange between the Kernel and design tools is through an intermediary media, the STEP file. When a design tool requests data from the Kernel, it receives the data in the form of a STEP file, and when the design tool submits data to the Kernel it does this also through a STEP file. Writing out and reading in STEP data are the tasks of the Data Interaction Manager Off-line (DIMOFF). In the DES, one generic DIMOFF provides the exchange support for all off-line design tools.

Data exchange between the Kernel and off-line DTs involves a bi-directional data mapping between an integrated global view of a building model in the Kernel and a partial view required by the design tool. The process of importing data from the design tool into a Kernel database is referred to as "meshing", because it merges a partial model into a richer
model. On the other hand, the process of exporting data from a Kernel database to a design tool file is referred to as "stripping", since it maps from a richer view of a building to a partial view and some entity types and relationships need to be stripped off (figure 6).

The off-line data exchange is done at a schema level not object level, it is guided by the comparison of a design tool subschema and the IDM schema. For example, if a "room" entity is a part of a design tool function's input schema, it will get all rooms in a building from the Kernel instead of a particular room.

The "stripping" process to extract data for a design tool function is performed in five phases:

1. Retrieving the Design Tool Function schema data. The object representing the targeted design tool function is retrieved from the Kernel database and from this we access the DTF input schema definition. From the Schema we can identify the entity definitions that are required for the data exchange as well as any constraints to apply.

2. Getting the instances. Having identified the name of all the entity definitions to be extracted, the next step is to locate the building model in the Kernel and from this the set of populated folders and all application instances currently in the Kernel. The populated folders are scanned for those entity extents which have a definition corresponding to those identified in step one, the instances associated with each retrieved entity extent are then to be written out to the STEP file.

3. Validation. Prior to processing the instances a validation check can be executed to ascertain that they comply with the specified design tool sub-schema. For example that all non-optional attributes do have a value.

4. Creating a STEP file model. This is a two pass process. The first pass creates for each instance retrieved an associated object which represents a line in a STEP file. This object is part of an abstract syntax tree which later produces the actual file. On the first pass the only part of the associated object that are initialised are the unique STEP file identifier and the object type. On the second pass we process those
Figure 7: Writing an object to a STEP file

```
#125 = ELEMENTARY_SPACE(16,"bedroom 2",.,",",.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.
```

attributes that the design tool sub-schema requires. This is determine from the schema definition stored in the Kernel.

5. Writing out the STEP file. Once the STEP file instance is created in the database, it can then be written out as a STEP physical file to be passed on to the Off-line design tool.

The "meshing" process also consists of five phases:

1. Reading the STEP file. A parser has been written using lex and yacc technologies which takes the STEP file and creates a persistent abstract represent of the file inside the Kernel. During this process the content of the file is syntactically validated.

2. Identifying the schema. As with the stripping process the targeted design tool function object is identified in the Kernel and its schema definition is retrieved. This schema definition is used in the next stage to validate that every instance in the STEP file complies with the design tool function schema.

3. Accessing the instances into the Kernel. The meshing operation is done in two passes. The first pass is to create the collection of association objects. A convention was agreed with all design tool interface authors that they would preserve the object identifier of any object existing object in the Kernel, object identifiers are positive integers. Newly created objects, which therefore did not exist in the Kernel, would be assigned a negative number as their identifier. For each STEP object with a positive identifier, its identifier is used to find the existing instance inside the Kernel. If the identifier is a negative number which implies it is a new object, a new instance of the right type will be created. However, the first pass does not modify the attribute values nor create the attribute values for the new objects.

4. Merging the new instance data into the Kernel. The second pass is to go through the association objects, check whether any attributes need to be modified or created. Only those attributes in the design tool sub-schema are modified.

5. Validation. Finally the resulting model can be validated against the IDM schema constraints.

6. DATA INTERACTION MANAGER ON-LINE

The Data Interaction Manager On-line (DIMON) handles the interaction between the CAD tools and the Kernel databases. In the COMBINE system, two CAD tools are developed:
- Architectural CAD Tool based on the AutoCAD package
- Architectural CAD Tool based on the Microstation software package

DIMONs for these CAD tools fulfil a role that could not be done effectively by off-line neutral file exchange. They support highly interactive and on-line interface with the Kernel databases and are able to associate behaviour with objects inside the Kernel.

Since AutoCAD and Microstation have quite different operating environments and programming language support, it was impractical to have one generic DIMON serving these two CAD tools. To achieve development efficiency, a layering approach was adopted for the implementation of the DIMON. There are four common layers in this architecture, of which three are functional layers and one communication interface layer (figure 8).

The first layer is Objectstore database which provides application schema initialisation, and services for object versioning, locking control, etc. At this level, all objects are treated the same and the semantics of IDM schema is not needed. On top of the database layer is a layer that handles generic object creation, deletion and navigation in the COMBINE context. The implementation of this layer requires the interpretation of the IDM schema as well as object composition knowledge. The third layer is IDM objects geometrical and topological behaviours. Typical examples are the transformation of co-ordinates, calculating surface areas or space volumes. These three layers together define the functions needed by the CAD tools for the IDM objects.

Both AutoCAD and Objectstore support C++ programming environment, the interaction between the AutoCAD tool and the DES Kernel can be achieved by sharing C++ objects in computer memory. On the other hand, the Microstation MDL programming environment is only C compliant, the interaction between Microstation and the Kernel at binary level is not possible. Therefore, a layer of COMBINE Application Program Interface was introduced. This layer defines the standard protocols through which the DIMON functions can be called. With this layer, the two CAD tools are able to use their own way of communicating with the DIMON. The AutoCAD tool adopted C++ static libraries and the Microstation tool used windows socket as a means of inter-process communication.

Figure 8: DIMON system architecture

<table>
<thead>
<tr>
<th>Microstation Engine</th>
<th>Microstation Tool</th>
<th>AutoCAD Tool</th>
<th>AutoCAD Engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket Interface</td>
<td>COMBINE Application Program Interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDM objects behaviour (e.g. space volume)</td>
<td>IDM objects creation, deletion, and navigation</td>
<td>Objectstore database layer</td>
<td></td>
</tr>
</tbody>
</table>
7. CONCLUSION

The Data Exchange System as described in this paper was tested in the context of the COMBINE integrated building design system prototype. It demonstrated that the architectural CAD tools can initialise a product model structured in accordance with the Integrated Data Model, and data representing a range of different views was extracted for design tool functions that need to be performed at the early stages of the design process. The output data produced by these tools in the form of STEP file was imported and meshed into the kernel database. In addition, the DES showed the ability of data transaction management, object versioning and support for concurrent engineering.

A number of conclusions can be drawn based on the COMBINE project experiences, especially the development of the Data Exchange System (Lockley 1995).

1. A data exchange system can be built that is generic enough to handle any conceptual data model, but this by no means denies the crucial importance of a generally accepted data model for a design system.

2. With an Integrated Data Model, it is possible to construct and populate a complex product model capable of meeting the needs of a range of different design tool functions.

3. Existing CAD engines can be configured to operate with the high level of semantic of a product model. However it would be more desirable to replace the simplistic representations in the CAD databases with building specific schemas, such as the IDM. The latest Industry Alliance for Interoperability initiative and the Autodesk’s plan to develop Industry Foundation Classes are set to achieve this goal (Kingham 1996).

4. Object database technology is mature enough to be applied to building design systems.

5. ISO-STEP and EXPRESS are viable tools for data modelling and exchange and they are now appropriate to be accepted as standards.

6. SDAI provides a good conceptual framework for data access mechanisms, but the full standard specification is perhaps too great an overhead for specific implementation.

Data exchange and transaction management is a main task of the COMBINE project. The developed Data Exchange System has met the requirements of this task and is a step towards a full scale integrated building design system.
REFERENCES


