DATA EXCHANGE SYSTEM OF COMBINE

S. R. Lockley, M. Sun
Department of Architecture
Newcastle University
NE1 7RU, UK

Abstract

The current Integrated Building Design Systems put emphasis on data sharing and data exchange instead of system intelligence. One important task in the COMBINE project is the development of a Data Exchange System which is able to maintain building data during the design phase and support data exchange between design tools operating on the building. This paper describes the prototyping effort of this data exchange system.

1 INTRODUCTION

The ongoing COMBINE project is funded EC to investigate computer application in the building industry. One of its aims is to provide an operational Integrated Building Design System (IBDS) to architects in design practice. Such an IBDS should be able to demonstrate attractive design support ability and help to improve building design. Building design is an interactive and three dimensional integrated process along design phase, design domain, and performance aspect [1]. Initially, the COMBINE's IBDS is to support a predefined segment of the building design space. Within this segment, the envisage scenario is that a number of architects, engineers collaboratively work on the same design project, they perform different tasks using different tools. The system should enable them to exchange design data in electronic form.

Data integration and process integration are two levels of integration requirements for an integrated building design system. Data integration refers to the sharing of data by all the 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 process integration requires a comprehensive model of design methodology and design process which is unlikely available in the short term. 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 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 store facility and exchange media.

At present, COMBINE is at its second phase, data integration is still considered as the main focus of research. One of the major task is to implement the IDM as a comprehensive Data Exchange System (DES) which is able to provide supports for both on-line and off-line data exchange, as well as version control, concurrent engineering, etc.

This paper focuses on the issues of prototyping the DES and describes the current state of the work and problems encountered during the process. C++ programming language and ObjectStore object oriented database are the main implementation platforms.

2 DES ARCHITECTURE

The requirements of the DES can be summarised as follows:

1. To configure and implement the conceptual Integrated Data Model (IDM) schema.
2. To populate the implemented IDM with data produced by design tools and transmitted to the DES in the form of ISO STEP Part 21 physical files.
3. To produce ISO STEP Part 21 physical files from this populated schema in accordance with the design tools schemas.
4. To develop an on-line interface to the DES for two leading CAD packages, AutoCAD and Microstation and to configure these packages as design tools in the COMBINE context.
5. To support shallow control of the information flow between design tools. That is to control the exchange of data but to have no knowledge about the meaning of the data being exchanged.
6. To perform simple constraint checking at data transaction boundaries.

A DES conceptual system architecture has been defined to support these requirements (figure 1).

In the COMBINE IBDS, there are a number of off-line Design Tools (DTs) and one or two on-line CAD tools. The Data Exchange Kernel (DEK) stores data of a consistent state of a building. The off-line DTs exchange building data with the DEK through Data Interaction Managers Off-Line (DIMOFF) using STEP file as a media and CAD tools communicate directly with the DEK through Data Interaction Managers On-Line (DIMON). The Data Exchange Toolkit is a generic service for accessing and manipulating STEP data, it is used by both the DT interface and DIMOFF. The Schema Handler is responsible for importing IDM schema and DT schemas into the DEK.

The development of the DES is collaborated between Newcastle University in the UK, TNO Building and Construction Research and Delft University of Technology in the Netherlands. This paper concentrates on the implementation of the internal components of the DES, they are DEK, DIMON, DIMOFF and Schema Handler.

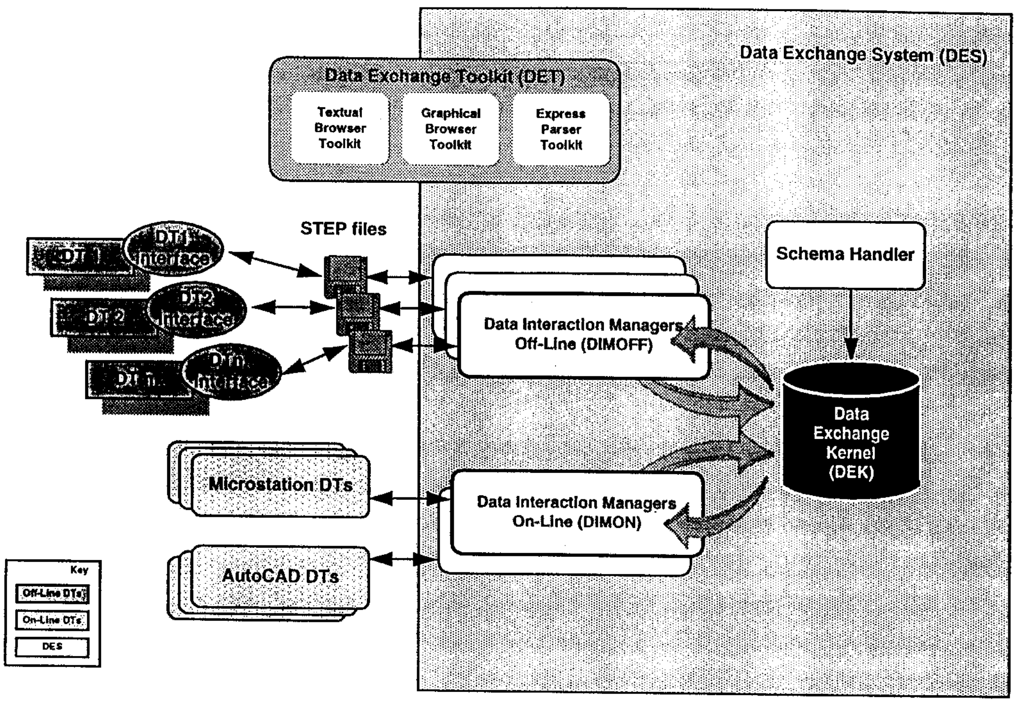
Data Exchange System (DES)
Data Exchange Toolkit (DET)
Textual Browser Toolkit
Graphical Browser Toolkit
Express Parser Toolkit
STEP files
DT 1
DT1 interface
DT 2
DT2 interface
DT n
DTn interface
Schema Handler
Data Interaction Managers Off-Line (DIMOFF)
Data Exchange Kernel (DEK)
Microstation DTs
AutoCAD DTs
Data Interaction Managers On-Line (DIMON)
Key
Off-Line DTs
On-Line DTs
DES

Figure 1 DES architecture and interface with the IBDS

3 DATA EXCHANGE KERNEL (DEK)

The purpose of the DEK is to handle persistent data storage, data management and interaction with on-line and off-line design tools. Two strategies are considered for the implementation of the DEK.

1. An early binding solution which implements the IDM schema as C++ classes.
2. A late binding solution which implements a generic data store and exchange service without depending on any application schema.

The first option is straight forward, in which all the entities of the IDM are one-to-one mapped to C++ classes. The entity attributes are implemented as data members of each class, constraints are also coded into the program. The DES will be a IDM dependent system. In addition to all other limitations, there is one practical reality that prevents the adoption of this approach. At the start of the DES development, the COMBINE IDM was not complete, even today, this schema is still evolving. Continuous changes in the IDM would mean endless re-coding in the DES. Consequently, an alternative option was considered.

The primary function of the DEK 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 DEK 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 DEK 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.

After the decision to adopt a late binding implementation for the DEK, three possible choices were reviewed, (1) STEP Class Library of the National Institute of Standards and Technology [2], (2) ObjectStore database metaobject protocol [3], and (3) ISO Standard Data Access Interface (SDAI)[4]. During this review, SDAI emerged as the most appropriate base for the DEK implementation.

The main 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 DEK, 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 and errors and inaccuracies still exist.

In a populated DEK database, there are two types of data being stored, schema data and model data (figure 2).

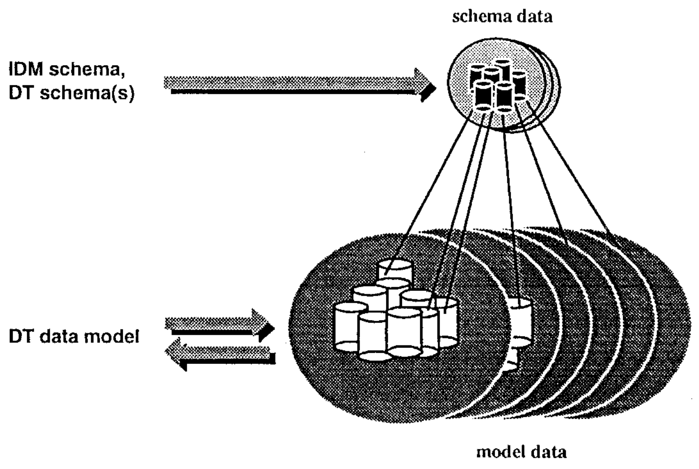

Figure 2 Data stored in a DEK database

The schema data is schema information in an application, e.g., IDM schema, DT schema, etc. Model data is real building data, in the COMBINE context, it includes one or more building models whose schema is defined by the IDM or a DT schema. The data structure is defined by the SDAI specification (figure 3), main features include:

- In a database, there are many models and schema, and each model has a pointer to one and one only schema but each schema can be pointed to by many models.
- A schema consists of a collection of entity definitions, defined types and global rules. Each entity has a name and a set of attributes.
- Every model has a model content which in turn has a data member of all the instances in the model. A model content also has a member of all the instance types in the model's schema as well as a member of all the populated types.
- All instances in a model are application instances, each of which has a pointer to an entity definition in the model's schema. As mentioned, each entity definition defines a set of attributes, each application instance has a corresponding set of values of these attributes.
- The data exchange between DEK and design tools is done on a model base not an instance base.

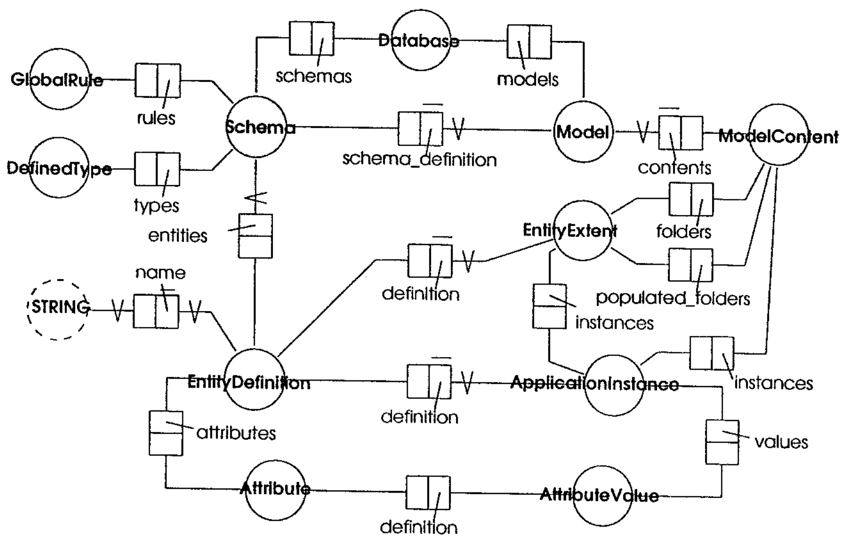

Figure 3 Illustration of the relationship between schema and model data

In such a DEK configuration, the model data import and export are carried out through a series of DIMON and DIMOFFs, and schema data import is done by a Schema Handler.

4 DATA INTERACTION MANAGERS OFF-LINE

Off-line refers to design tools that operate independently from the DES. They could be a separate program on the same computer, or they might reside on different computers, even different sites. There is no direct link between the tools and the DEK, the data exchange between the DEK and design tools is through an intermediary media, the STEP file. When a DT requests data from the DEK, it receives the data in the form of a STEP file, and when the DT submits data into the DEK it returns a STEP file to the DEK. Writing out and reading in data in STEP files is the task of a DIMOFF.

In COMBINE, the IDM schema is an integrated conceptual building data model, a DT schema represents a subset of this model from the point view of the particular DT. Data exchange between the DEK and off-line DTs involves a bi-directional data mapping between a global view of a building and a partial view of it (figure 4).

The process of importing data from the DT STEP file into the DEK 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 the DEK to a DT STEP file is referred to as "stripping", since it maps from a richer view of a building to a more limited view and some entity types and relationships need to be stripped off.

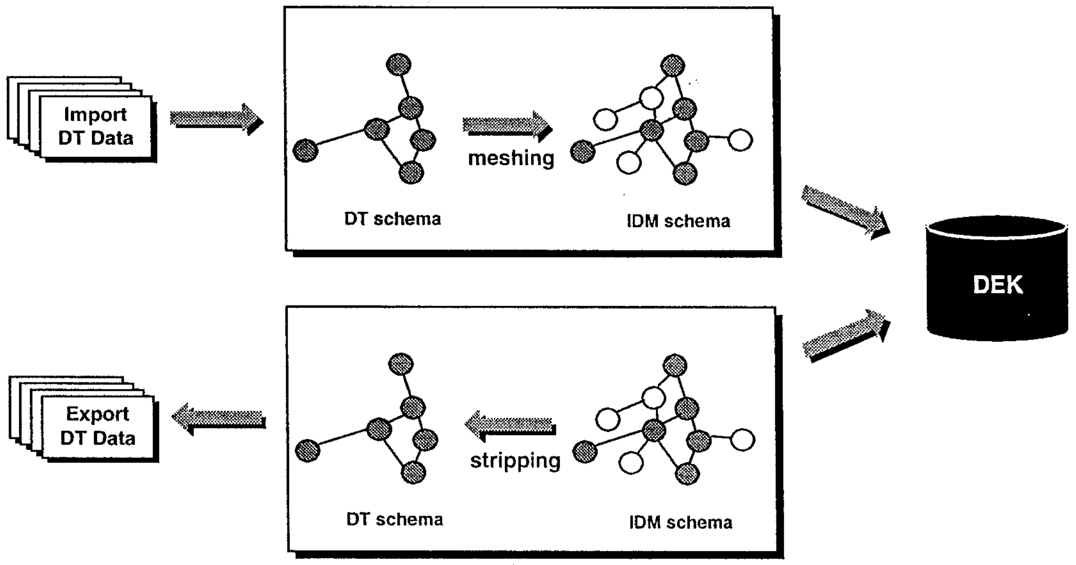

Figure 4 Data exchange between DEK and off-line design tools

The off-line data exchange is done at a schema level not instance level, it is guided by the comparison of a DT schema and the IDM schema. For example, if a "room" entity is a part of a DT schema, it will get all rooms in a building from the DEK instead of a particular room. Instance selection is handled by the DT's own interface kit.

In the existing implementation, "stripping" is implemented as a generic service, but the "meshing" function requires DT specific coding. The only reason for this is that early binding C++ classes are needed for parsing a STEP data file. Therefore, currently one DIMOFF is configured for each off-line design tool. It is envisaged that a general STEP parsing will be developed to eliminate the need for early binding and generic DIMOFF will serve all DTs.

5 DATA INTERACTION MANAGER ON-LINE

The purpose of the Data Exchange Manager On-line (DIMON) is to support on-line data exchange between the DEK and CAD DTs. It consists of the following components (figure 5):

1. Class structure,

2. Geometry and topology related behaviour,
3. Interaction with CAD tools, and
4. Interaction with the DEK.

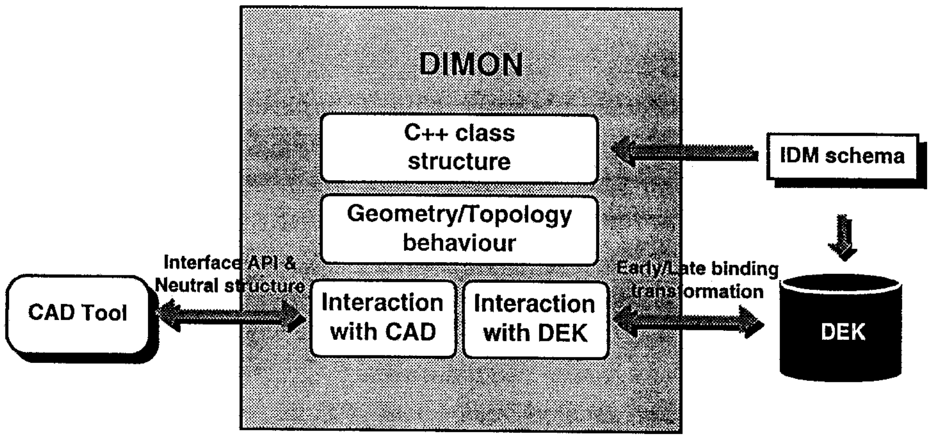
DIMON
C++ class structure
IDM schema
Geometry/Topology behaviour
Interface API & Neutral structure
Early/Late binding transformation
CAD Tool
Interaction with CAD
Interaction with DEK
DEK

Figure 5 DIMON configuration

The class structure is determined by its EXPRESS definition in the IDM. It has to be pointed out that the implementation is not a strict mirror of the conceptual IDM EXPRESS schema. For example, the `CartesianPoint` data member is implemented as `_x`, `_y` and `_z` instead of a list of `_coordinates`. The EXPRESS primitive types are mapped to C++ types, e.g., `INTEGER` to **`int`**, `REAL` to **`double`**, `STRING` to **`char*`**, etc. Aggregate types are implemented as corresponding Objectstore collection classes, `LIST` to `os_list`, `SET` to `os_set`, etc. `INVERSE` attributes are implemented as Objectstore inverse relationships.

Geometry and topology related behaviour is a grey area of the DIMON implementation. It is considered neither sensible nor within the COMBINE resource to develop a full blown geometry/topology application. Instead, third party products will be used to provide support of this aspect. We only seek to implement some essential functions, such as coordinate transformation, moving a surface, extracting a surface from an existing one, adjacent relationship between spaces, etc.

For the CAD interaction, we try to develop a CAD independent API and a neutral data exchange structure which enables data communication with different CAD systems across multiple platforms, such as DOS, Windows and UNIX.

A building model is stored as late binding representation in the DEK, and the same model is represented as early binding one. There is a one to one mapping relationship between early binding C++ model and late SDAI model, early binding C++ object and late binding application instance. The interaction with DEK component of the DIMON is responsible for the transformation between these two representations.

It should be noted that DIMON defines generic services, configuration needs to be made for each specific CAD design tool. In April 1994 at a COMBINE project meeting, a special configured DIMON demonstrated that a complete building can be initialised by a CAD tool as an early binding data model and be transformed into the DEK and late binding representation.

6 SCHEMA HANDLER

The schema handler is responsible for importing, managing, and in the future exporting, EXPRESS schemas. As part of the DEK, the SDAI dictionary data schema is implemented as C++ classes, e.g., SdaiSchema, SdaiEntity, SdaiAttr, SdaiExplicitAttr, etc. IDM schema and DT schemas are stored persistently as instances of these classes. Figure 6 shows Schema Handler architecture and how an EXPRESS schema is imported into the DEK.

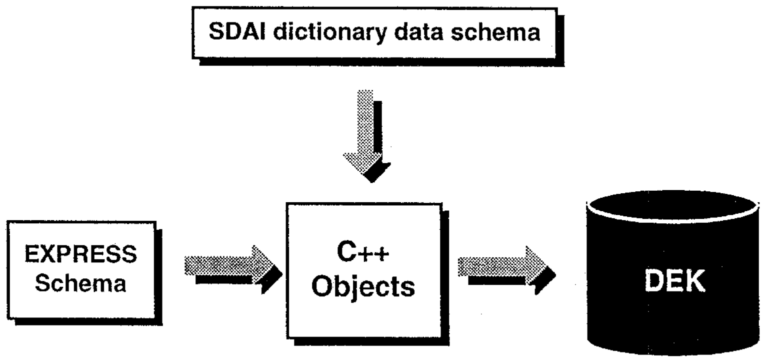

Figure 6 Schema Handler architecture

1. C++ classes are generated based on the SDAI dictionary data schema, these classes, together with an EXPRESS/STEP interface kit developed in COMBINE [5], are used to parse IDM and DT schema files.
2. First step for importing a schema is to translate the EXPRESS file into STEP physical file format using a purpose built tool [6]. This tool also checks the validity of the EXPRESS schema in term of EXPRESS language syntax.
3. The Schema Handler reads in this STEP file to create C++ objects and stores them persistently in the DEK.
4. A DES database only needs to import the IDM schema and a DT's schema once, then it can be accessed in subsequent operations.

So far, the schema importing function of the Schema Handler has been fully implemented. It is envisaged that in a longer term, the Schema Handler will be extended to include functions such as exporting schema as EXPRESS files, NIAM or EXPRESS-G diagrams, allowing manipulation and editing existing schemas by textual or graphic means, producing sub schemas.

7 CONCLUSIONS

The first DES prototype has been produced covering all the essential components described in this paper. Its main functions include importing schema data into the DEK; initialising a building model by an off-line DT STEP file or by an on-line CAD tool; exporting STEP files in accordance with a DT schema and exporting a building model for accessing by an on-line CAD DT. It is expected that a full functional DES will be available to be installed in a design office by the end of 1994.

It is emerged from the initial prototyping that the late binding implementation of the DEK is a right decision. It provides a flexible way of data exchange and an easier route for system extension of the COMBINE IBDS. New design tools can be introduced to the system with minimum programming effort. Furthermore, it allows the DES and the IDM to be developed in parallel. Having said that, it is also clear that ideally the IDM conceptual modelling work should precede the implementation of the DES, since the DIMON requires the IDM schema to be "frozen". Otherwise, changes in the schema would invalidate DIMON application.

The COMBINE Data Exchange System prototype has demonstrated the ability of storing data in a central repository and supporting off-line data exchange using STEP/EXPRESS media and on-line data accessing by CAD tools. Further investigation is needed in areas such as constraint checking, transaction management and system error handling.

REFERENCES

1 Augenbroe G. and Laret L., 1989, Integration of simulation into building design: the need for a joint appoach, Paper for 1990 ASME International Solar Energy Conference, Miami, FL, April 1-4 1990

2 Mclay M. J. and Morris K. C., 1990, The NIST STEP Class Library, NISTIR 441, August 1990

3 Object Design Inc., ObjectStore release 3.0, User Guide, May 1994

4 Standard Data Access Interface, ISO TC184/SC4/WG7 n350, August 31, 1993

5 Soethout L., Plokker W., Vries P. and Rombouts W., 1992, EXPRESS/STEP Interface Kit for COMBINE, May 1992

6 This is a tool developed by TNO-BBI as part of the COMBINE project.