CIFE CENTER FOR INTEGRATED FACILITY ENGINEERING

Integration of a Three Dimensional CAD Environment into an Interactive Workspace

By

Timo Hartmann, Martin Fischer, Ernst Rank,
Marcus Schreyer, and Frank Neuberg

CIFE Technical Report #146
August 2003

STANFORD UNIVERSITY

If you would like to contact the authors, please write to:

c/o CIFE, Civil and Environmental Engineering Dept.,
Stanford University
Terman Engineering Center
Mail Code: 4020
Stanford, CA 94305-4020

Martin Fischer, Ernst Rank, Marcus Schreyer, Frank Neuberg, Timo Hartmann

Integration of a Three Dimensional CAD Environment into an Interactive Workspace

Authors: Marcus Schreyer, Visiting Scholar, CIFE, Stanford University
Martin Fischer, Associate Professor, Stanford University
Ernst Rank, Professor Technical University Munich
Frank Neuberg, Research Scientist Technical University Munich
Timo Hartmann, Visiting Scholar, CIFE, Stanford University
Publication Date: August 2003
Keywords: Interactive Workspaces, Project Meetings,
Product Model, Virtual Design and Construction
Document Source: http://cife.stanford.edu/online.publications/TR146.pdf

Abstract

In the design and construction process of a building the tasks to accomplish are distributed to many different participants. All these participants use different kinds of software applications which all use a different view of the overall project model.

The project's participants discuss mutual interactions within their models in meetings, in order to find the impacts of their work on the work of others. For example if the architect changes the geometry of parts of the projects building this will have effects on the cost estimation, as the quantities used in the estimation have changed. Due to a lack of adequate interfaces between the different used applications it is common practice within meetings to first explain and describe different circumstances as the one described above using paper based medias [Fischer et al, 2000]. Afterwards all participants of the meeting will update their application models.

Interactive workspaces integrate modern computer technologies in order to enable data exchange and control between applications. This exchange then will hopefully be supportive for describing and explaining different problems occurring within a project meeting by using the software applications with which the participants model their respective views of the building.

The Center for Integrated Facility Engineering (CIFE) is developing such an interactive workspace. A number of different applications used in civil engineering are already integrated. Nevertheless it is not yet possible to work with a CAD

environment commonly used by architects to model the geometrical aspects of the project model. This report describes the integration of such a CAD application within an interactive workspace.

Contents

Chapter 1

Introduction

During the design and construction of a building, the tasks which have to be accomplished are distributed to different participants. Due to mutual interrelations between the different tasks of one project it is often necessary to find solutions for problems beyond the scope of a single party. Thus communication between all involved participants is extraordinarily important.

Research results publicized in [Fischer et al, 2000] show that in current meetings using media which are paper based, 40% of the time is spent on describing project related information, while only 10% is spent on predictive tasks. For the finding of qualitatively good solutions the prediction of effects of different design alternatives on the overall project outcome is extremely important. Thus more time should be spent in project meetings on predictive tasks. This can be achieved by reducing the time spent on describing and explaining circumstances.

One reason why the time spent on describing information is exceptionally high is that all the different parties working together on a project use a different view on the related project data. For example architects use a geometrical model of *structural elements* while project managers use a model describing temporal dependencies between *construction activities*. Thus, currently, a lot of description time is necessary to ensure that all participants have a comprehensive under-

standing of the project.

One approach to solve these problems is to use modern computer based technologies. Relationships between the different models could be established, by using modern database and messaging techniques. The established relationships can then be graphically represented in so called interactive workspaces [Fox et al, 2000]. Interactive workspaces are meeting rooms which integrate a wide variety of modern hardware technology like projectors and touchscreens to display project related information.

1.1 Interactive Workspaces in Civil Engineering

The Center for Integrated Facility Engineering (CIFE) of the Stanford University in California created an interactive workspace called IRoom. The IRoom is intended to create a working environment for group meetings in the construction sector [CIFE - CIW]. It can be used to point out and graphically represent the relationships existing between information items needed for the different tasks which have to be accomplished in the planning process of a building. Thereby the IRoom is intended to support the decision making during the planning process.

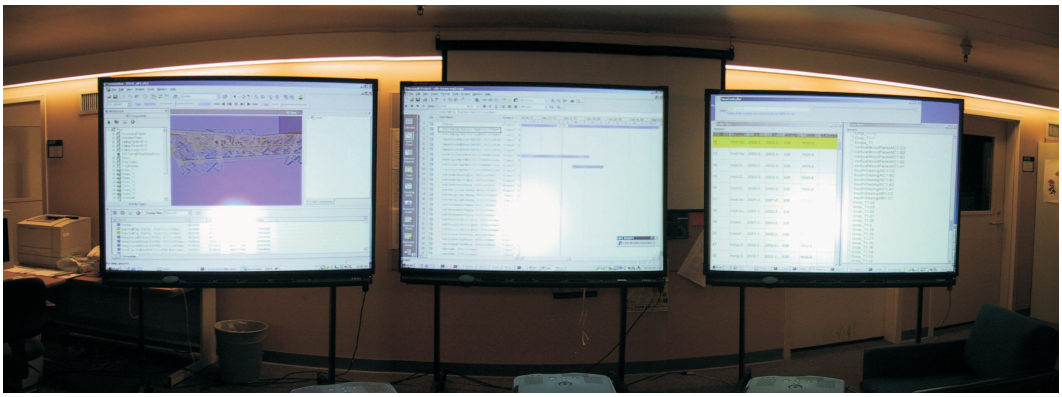


Figure 1.1: Three screen display system of the IRoom

In order to be able to represent the project related data the IRoom provides

three smartboards with projectors (Figure 1.1). By using these smartboards a number of different applications can project their views of the overall model simultaneously. Additionally, defined relationships between the different objects of the applications' data models can be pointed out easily.

Currently, the IRoom already integrates a variety of different software applications. Project managers can use Microsoft Project [Microsoft - Project] to model the different *construction activities* necessary to construct a building. Furthermore a 4D viewer, Commonpoint 4D [CPT Tech. - 4D-Viewer] (CP4D), is integrated, which is able to represent the relations between geometrical properties of *structural elements* and the *construction activities*. For general presentation purposes the Microsoft Office applications Powerpoint, Excel and Word can be used.

1.2 Objectives of the Report

Currently, the IRoom is designed for the use of project management teams. Other persons involved in the planning process who require geometrical project information like architects can not use the IRoom adequately. CP4D can represent geometrical information available in the IRoom environment, but the application does not offer interactive access to geometrical or architectural properties. Thus, today the IRoom can hardly be used to support the detection of the comprehensive effects of a change in the architecture or geometry of the building on other project-related issues.

One of the worlds leading CAD companies, Autodesk, provides Architectural Desktop (ADT), a three dimensional CAD environment [Autodesk - ADT]. ADT is product-model based. This means it is working with objects which represent discrete *structural elements* of the building like walls, slabs or columns (Figure 1.2). These objects can easily be used to create relationships between other non geometrical data models used in the project, like for example the already

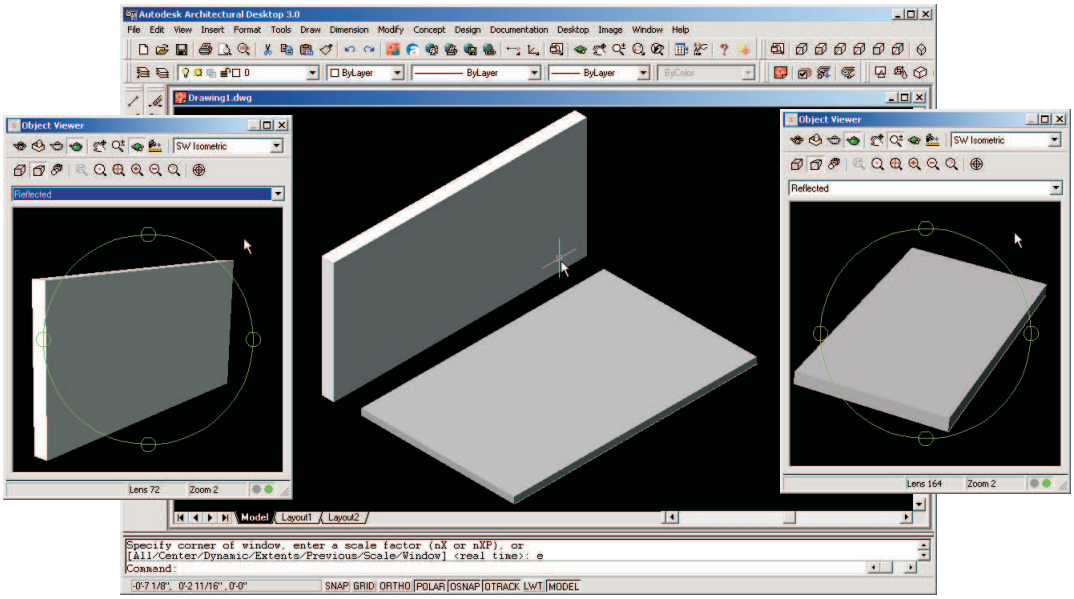


Figure 1.2: Slab and wall objects in the ADT product-model

mentioned project management *construction activities*. Thus a successful introduction of ADT into the IRoom would enable architects to take part in IRoom project meetings. It would be possible to undertake geometrical changes within the interactive workspace. A better understanding of the impacts of geometrical changes would be possible by using the IRoom display techniques and establishing relationships with other already introduced software programs.

The main objective of this report is to develop a method of integrating ADT into the IRoom environment. Besides this major objective the methods used in the integration process should be generalized. In this way the examined issues can then be hopefully used for integrating other software applications as well.

1.3 Motivating Test Case: Bay Street Project

The test case used for the validation and testing of the developed methods is the Bay Street Project. The Bay Street Project is a project of DPR Construction

Inc [DPR - Construction], Redwood City, California. The project is located in Emeryville, California.

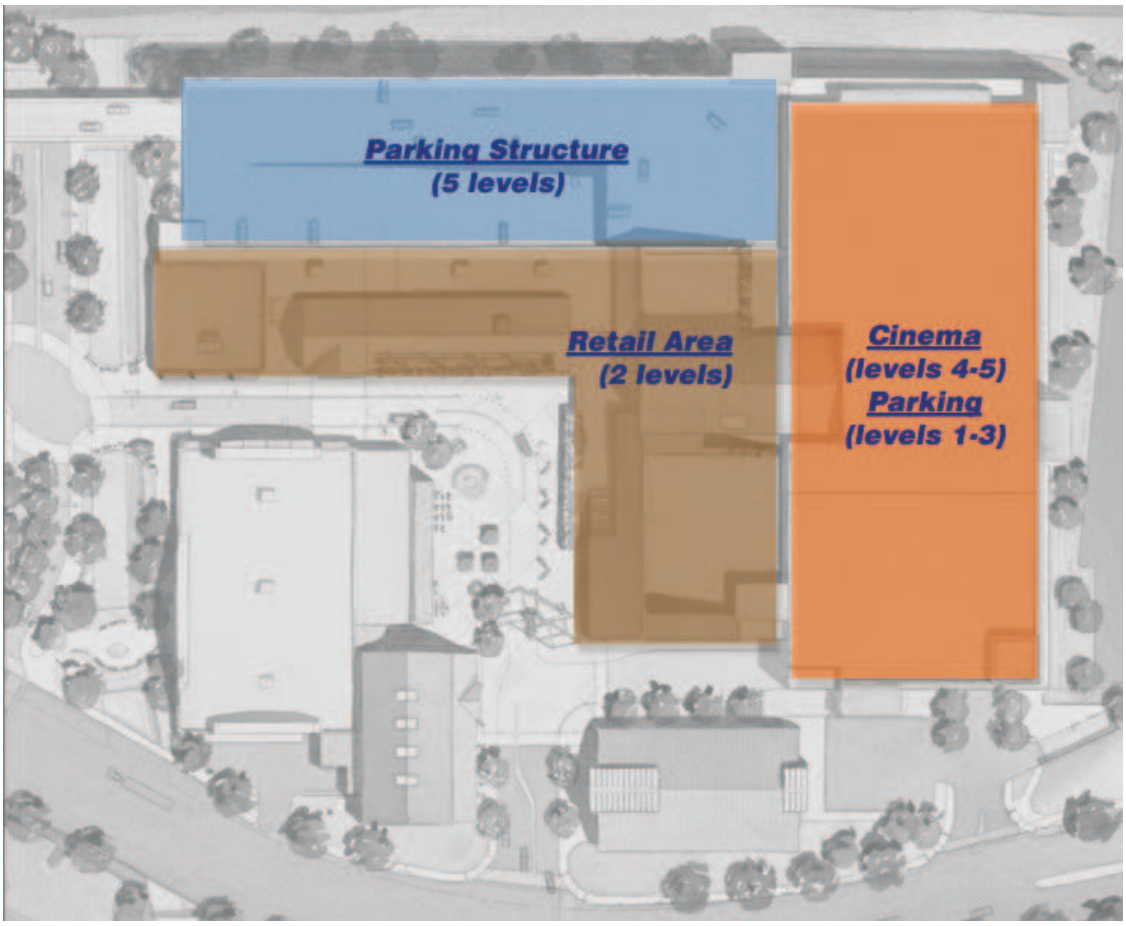


Figure 1.3: Bay Street Project partial site overview

Figure 1.3 shows an overview of the project site. There are three main structures:

- A five level concrete parking structure,
- a two level structural steel retail building, and
- a 15 screen theater, built of structural steel on top of another 3 level concrete parking structure.

The site is very tight due to the neighboring buildings. Thus DPR decided to work with a single crew per activity, reducing the slack available in the overall schedule. Official start of the project was in June 2001. According to the schedule the project had to finish in November 2002, in order to be able to open the shops for the Thanksgiving and Christmas shopping season.

Soon after the official start of the project hazardous materials were found, during the excavation works. These findings and the following examinations led to a two month delay of the critical foundation work activity. All the participating project parties had to meet again for discussions of the overall schedule. In the context of these meetings it was realized how difficult it was to determine the feasibility of a particular schedule adjustment and to find the impacts of changes in the base schedule on the overall project outcome.

In meetings of the above described kind interactive workspaces can be a great contribution. The finding of relations between different sub-domains of the overall project like schedule and cost estimation can be supported. Thus the Bay Street Project is a suitable test case for the development of interactive workspaces.

Chapter 2

The IRoom as an Interactive Workspace for Civil Engineering Project Teams

Since 1998 the Center for Integrated Facility Engineering (CIFE) at the Stanford University in California has been developing an interactive workspace called the IRoom. This workspace integrates modern software and hardware technologies creating an environment which supports the decision making of civil engineering project teams.

The first research results on the impact of these interactive workspaces on the time spent in meetings were publicized in [Fischer et al, 2000]. They show that, using paper based media in meetings, 40% of the time is spent on describing project-related information, while only 10% is spent on predictive tasks. However, these predictive tasks are extremely supportive for generating high quality solutions. [Fischer et al, 2000] suggest that the time spent on describing information could be reduced to 10% through the use of interactive workspaces. In this way, the time spent on predictive tasks could rise up to 50% (Figure 2.1) allowing a qualitatively better decision making process for multi-disciplinary project

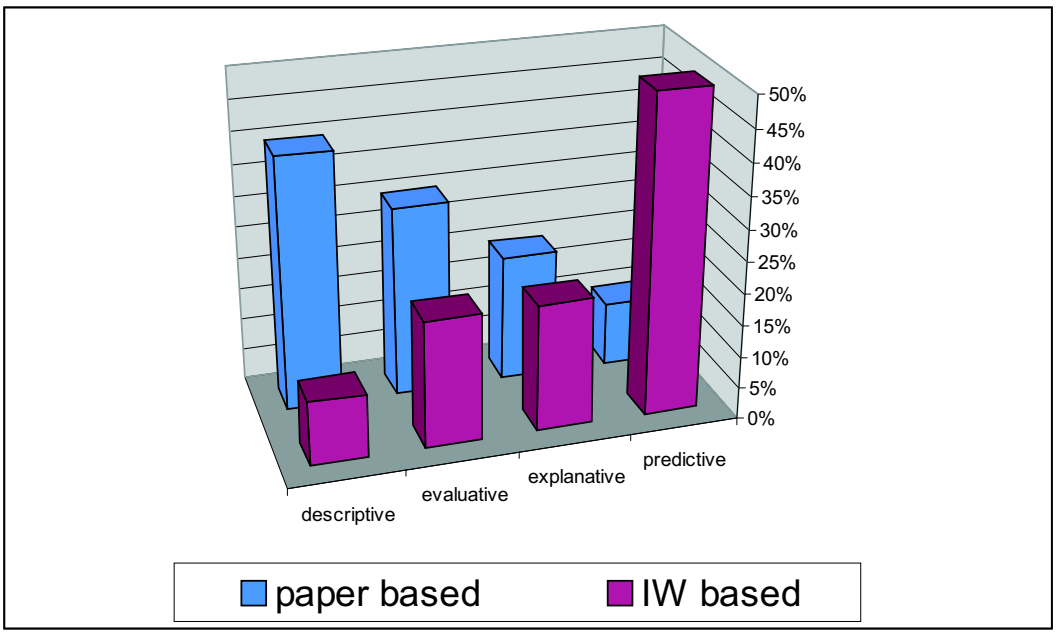


Figure 2.1: Time spent in meetings according to [Fischer et al, 2000]

teams.

To get an understanding of the current possibilities using integrated workspaces, the following chapter describes the established hardware and software components of the IRoom. Further research projects should use and extend these technologies.

2.1 Display and Message Passing System

The IRoom uses three Smartboards with corresponding projectors for display purposes. Smartboards are large screens combined with a touchboard functionality, marketed by Smart Technologies Inc. [Smart Technologies Inc.]. Thus it is possible to use them for the interaction with running software on computers driving the Smartboard projectors. Further Smart Technologies Inc. provides special software which enables to draw on the Smartboards using special electronic pencils. In this way the Smartboards can be used like a normal black

board.

Besides the capability of the IRoom to display information on the three Smartboards, the most important aspect of the IRoom technology is the message passing system [Fox et al, 2000]. It enables all the applications of the IRoom to pass short messages to or receive messages from other applications. These software programs can run either on the same device or on any other device integrated in the IRoom environment.

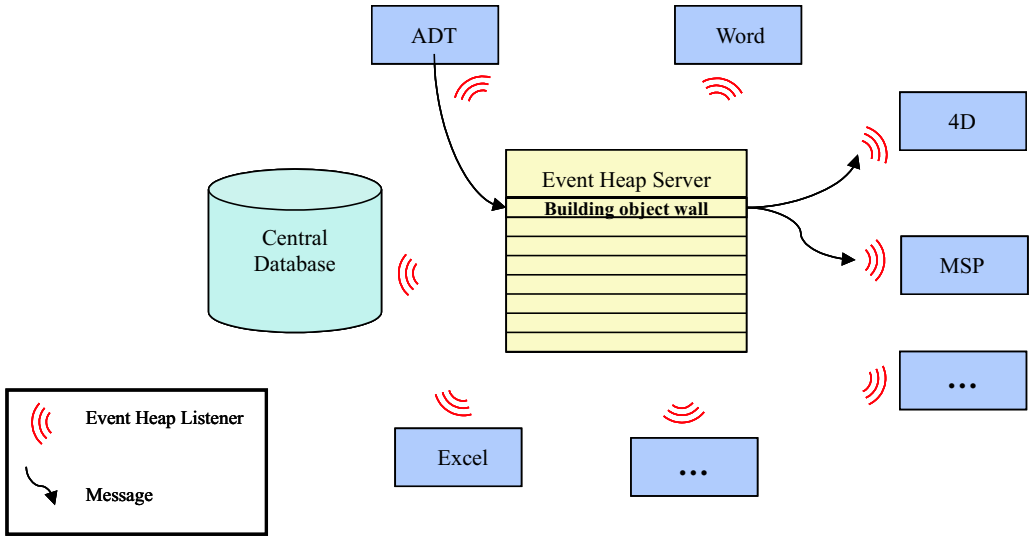


Figure 2.2: Message passing system between the IRoom applications

Figure 2.2 demonstrates the basic system of passing messages between the different IRoom applications. Using a special software called EventHeap messages can be sent to a central event heap located on one of the IRoom computers [Stanford CS - EventHeap]. Here all the messages get collected and stored for a predefined period.

Any software component on one of the IRoom devices can access these messages stored on the event heap. To gain this capability they have to be extended with an event heap "listener". This "listener" observes the event heap constantly enabling any application to pick up messages from the event heap. The type of

messages which should be picked up can be defined. In this way every application has the possibility to just react to specific messages.

For example, in Figure 2.2 ADT sends a message to the event heap containing information about a geometrical *structural element* object. This message is then picked up by two applications, which have a "listener" which was developed to react to the type of message sent by ADT. CP4D picks up the message in order to, for example, highlight the respective building component. Furthermore, Microsoft Project can, for example, establish some relationship between the corresponding *construction activity* and the related geometrical objects.

2.2 IRoom Topology

The IRoom uses three permanent client computers. All three client workstations are Pentium III 866 MHz desktops running under Windows 2000. The clients are used for running common software applications used in civil engineering. These applications are, for example, Microsoft Word and Excel, Microsoft Project and CP4D which enables the geometrical and temporal representation of a project. Furthermore, the three clients drive the three projectors with the corresponding Smartboards which are used for displaying the different models of the used applications.

Other devices like notebooks or palms can also be integrated as clients into the IRoom environment, but it is necessary to install the applications for the basic IRoom technologies. These include the IRoom EventHeap client application, which enables the client to receive and send messages between the applications. If applications on these new devices like Microsoft Project, Microsoft Excel, Microsoft Word and Microsoft Powerpoint have to be integrated within the message passing system, the listeners for all these programs will have to be installed as well.

Applications for general functionality of the IRoom are installed on a server.

The server of the IRoom environment is a Pentium III 866 MHz workstation. It contains the software for the message passing system between all of the clients and their applications. Another useful software application that is used is Pointright [Johanson et al, 2001]. It enables a set of input devices like a mouse and a keyboard to access all other screens of the integrated IRoom clients. This technique is especially useful when working on the three Smartboards, as it is no longer necessary to use the three input devices of the clients running the projectors. For an overview of the complete IRoom topology refer to figure 2.3.

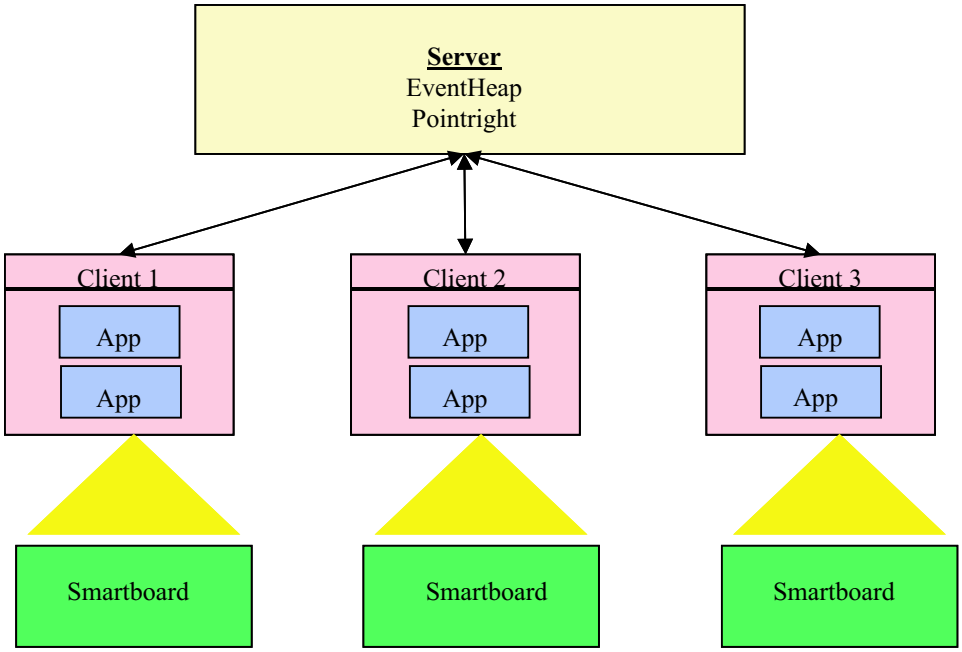


Figure 2.3: Hardware topology of the IRoom

2.3 Using the IRoom in Practice

Using the previously described technologies, the IRoom workspace can provide significant support of the communication process in meetings. The different Smartboards can be used to display different views of the project. The mes-

sage passing mechanism can point out relationships between objects in the different views. Furthermore, the views on the different Smartboards controlled by different computers can be manipulated by a single input device. The two following cases show examples of already realizable scenarios that apply the IRoom technology.

2.3.1 Case 1: Multiple Microsoft Powerpoint Presentations

This rather simple example shows how the IRoom supports multi-screen representations. Using one Microsoft Powerpoint slide show presentation it is often not possible to include all information needed to describe a topic on a single slide. Thus animation techniques or more than one slide are used for the representation of one issue. This often increases the complexity of the presentation and can confuse the audience.

The IRoom coordinates two or even three different slide shows on the Smartboards. All Microsoft Powerpoint presentations can be controlled by a single mouse using Pointright. It is possible for example to use one Smartboard to display the contents of the overall presentation. Another Smartboard might be used for the main slide show, while the third Smartboard can be used to display supporting materials.

2.3.2 Case 2: Interaction between CP4D and Microsoft Project

In large civil engineering projects it is often hard to keep track of *structural elements* which are related to a *construction activity* in the project managers' schedule. Common practice in meetings is to use one drawing of the specific part of the construction site while all participants have a copy of the Gantt chart

containing the *construction activities* at hand. Now, the meeting participants have to manually search for the related construction site position in the drawing. Often this process is very time consuming, due to the complexity of the two-dimensional plans used.

By applying IRoom technologies, it is possible to display the Gantt chart with the *construction activities* on one Smartboard. An application enabled to show geometrical information like CP4D displays a view of the construction site on another Smartboard. Now a previously defined database containing relationships between the *construction activities* and the geometric objects can be used to query the related *structural elements*.

Highlighting of the related objects in the geometric viewer is possible by applying the message passing technique. In this case the Gantt chart viewer program sends a message containing the selected *construction activity* to the event heap. This message can now be picked up by the geometric viewer and can be mapped to the related *structural elements* using the relationship database, and the corresponding *structural elements* can be highlighted. Using this highlighting technique the meeting participants can focus more quickly on the information relevant to a particular issue even though the information is represented in multiple software applications. The software applications on the other hand do not have to be aware of the relationships between related objects like structural elements and construction activities.

Chapter 3

Architectural Desktop as CAD Environment for the AEC Sector

Autodesk is providing Architectural Desktop (ADT) as a CAD solution for the Architectural, Engineering and Construction (AEC) industry. Autodesk's standard CAD-environment AutoCAD is used as the basis of ADT, as all functions of AutoCAD can be used in ADT. Autodesk wants to address two different groups of customers. The first group are the architects, who can use ADT to create or modify three-dimensional models. These models combine different building objects like walls, doors, windows, roofs or slabs into complex buildings as can be seen in Figures 3.1 and 3.2. The second group are software developers who can use two programming libraries, the Object Modelling Framework (OMF) [OMF Developer's Guide, 2002] and ObjectArx [ARX Developer's Guide, 2000, MCAULEY, 2000] to access and extend the provided functionality of ADT by programming new applications.

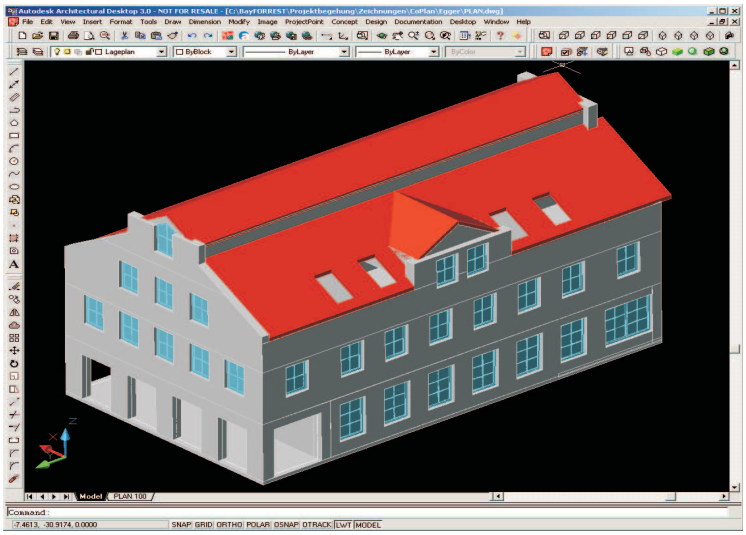


Figure 3.1: Example of a building model in ADT

3.1 Product Model of ADT

Product models are describing buildings with discrete objects like walls, doors, slabs and roofs. A set of these objects is arranged in a hierarchical order. For example, the different zones of a building are combining rooms bounded by walls with windows and doors. The characteristic data, which is needed for the abstract model, are assigned to the individual objects by properties, or in object oriented terms, member data. Product models are highly extensible. It is only necessary that data sets, which are representing "real world" properties of objects, are pre-defined, which means a universal standard has to be introduced by the developers of the respective product model. ADT is defining such a product model, which can be used for the creation and representation of the geometric and architectural properties of a project's building.

As mentioned before it is important to pre-define properties within product models to enable access to these data. Within ADT, important properties which define the objects in the overall project context can be assigned to fixed value types. Users cannot change these underlying types. This means that each object

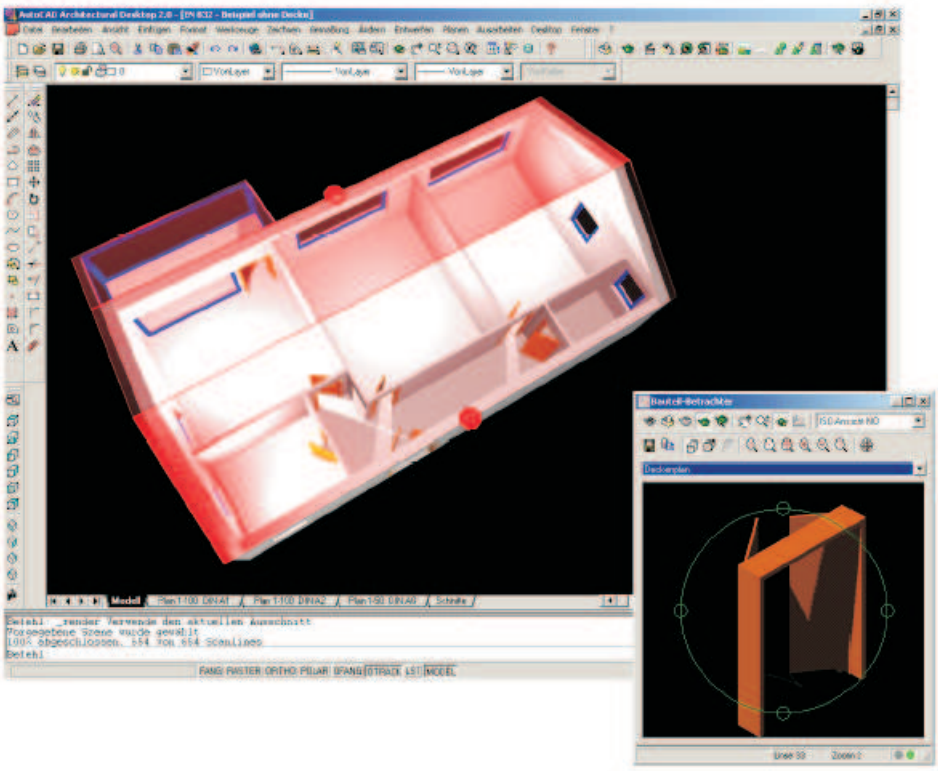


Figure 3.2: Door object in ADT

provides a pre-defined number of properties. Most of them store the geometric information of the objects.

Information which is the same for a variety of objects of one type is assigned to object styles. In the case of walls, for example, the constructional layers are stored in a wall style. Now it is possible to create many instances of different walls, which all have a different geometry. The same layers are attributed to all of the walls which are using the same style. It is no longer necessary to create these layers for every new instantiated wall, and a lot of time is saved during the geometric modeling of the building. In ADT, these styles can be exchanged between different drawings and even between different installations of ADT on different computers.

Arbitrary data needed for the individual design tasks can be attached to ADT entities by *property set* objects. These objects can collect a number of fixed value types. The exact data of these types can then be specified for individual ADT objects to which they are attached. Furthermore, there is the possibility to attach the same kind of *property sets* specifying the same underlying data types to different kinds of ADT entities. For example, information about the provider of pre-fabricated structural elements can be attached to different kinds of entities like walls or slabs.

3.2 Possibilities to Customize ADT

Autocad is known as one of the most programmable, customizable, and extensible design systems on the market. Statistics released by Autodesk claim that almost 70% of Autocad customers are using LISP, Visual Basic, C++ or menu customization to increase the productivity of the basic CAD product [MCAULEY, 2000].

ObjectArx stands for Autocad Runtime eXtension and was introduced with Autocad R13. Since then many new versions of both Autocad and ObjectArx have been released. Currently the latest versions are Autocad 2004 and ObjectArx 2004. ObjectArx are programming libraries for C++ for use with Microsoft's Visual C++ Compiler. Containing some 220 classes and over 3,000 unique member functions it is quite a comprehensive application programming interface (API). Further information on the programming of Autodesk products with ObjectARX can be found in, for example, [ARX Developer's Guide, 2000] or [MCAULEY, 2000].

The Autodesk AEC Object Modeling Framework (OMF) is an extension of ObjectArx. It has been developed to facilitate the programming of Autodesk's Architectural, Engineering and Construction (AEC) applications like Architectural Desktop or Autodesk Building Mechanical and Electrical. OMF supplies classes and functions, which provide the same core technology that is used by

these extension products for Autocad. Objects of ADT like doors, windows or walls can now be shared across diverse Autodesk products or extended while preserving the compatibility with Autodesk's common DWG file format.

OMF is a layer wrapping ObjectArx with an extended functionality. It is still possible to use ObjectArx, but if it is necessary to develop more than trivial applications, the use of the more powerful OMF is advisable. One of the most challenging aspects while programming ADT with OMF is to figure out when to implement the normal ObjectArx functionality and when to use the extended functionality of OMF. The OMF Developer's Guide [OMF Developer's Guide, 2002] gives more information about OMF programming.

With the ObjectArx and OMF library it is now possible to create Autocad extension modules. These modules can be uploaded into ADT during runtime. Thus it is possible to extend the basic ADT functionality easily. Only the desired extension modules, containing the wanted functionality, have to be uploaded.

Using ObjectArx and OMF it is possible to implement the necessary IRoom event heap functionality enabling ADT to pick up messages from the central message heap with the help of a "listener" and sending messages to it. Furthermore developers can use the programming libraries to create and update a central database used within the IRoom environment.

Chapter 4

Communication Systems between Different IRoom Applications

Computer applications used by civil engineers today generally are based on different database structures representing the respective project tasks' views. The data structures of the different databases are independent, but most of them contain information which should be shared in the overall project context. To enable the exchange of the shared data, communication mechanisms between different programs have to be established. This is only possible using mapping mechanisms for objects of the different applications' databases.

Usually all the applications use a variety of common abstract objects. In the design and construction process of a building some of these objects used are, for example:

- Building Components,
- Cost Items,
- Material or Labor Resources, and
- Construction Activities.

The definition of general objects like those mentioned above in a central database is the basis to establish relationships with objects sharing the same information in different applications. In this way it is possible to perform a mapping of objects between different programs. For example, using objects representing *building components* within a central database, the mapping between elements of different CAD applications and geometric viewers can be performed.

Furthermore, the different object types have interdependencies. For example, the object *construction activity* obviously has relationships with *material and labor resources* as well as with *building components* and *cost items*. These interrelations can also be stored in the central database, allowing the mapping between different object types. These relationships in a database schema can be modelled in various ways. Another important issue are the access possibilities to the central database by the different applications.

It is of great importance to thoroughly analyze different possibilities to model the data and different possibilities to access the chosen model before starting to implement software. This chapter describes different alternatives of establishing and communicating these relationships in an interactive workspace environment. At the end, the used database schema and database access mechanism for the integration of ADT into the IRoom are introduced.

4.1 Possible Database Structures for Shared IRoom Application Data

This section describes two possible database schemas for the data mapping between different applications. The first option is to store only the relationships in the database. This approach would lead to the smallest possible amount of information stored within the central database.

On the other hand this approach means that the applications have to store all

data concerning these objects. Thus data exchange between different programs can only be realized by direct communication of the respective applications. Using such a database schema in the IRoom environment would require that a lot of different messages have to be exchanged between the applications using the event heap's message passing functionality. Furthermore, this would require that each software program has to understand a great number of different message types. Software developers would have to spend a lot of time to implement the functionality for all integrated applications. Hence, the integration of new applications into the IRoom would be a lot more expensive if a database is used which just stores the relationships.

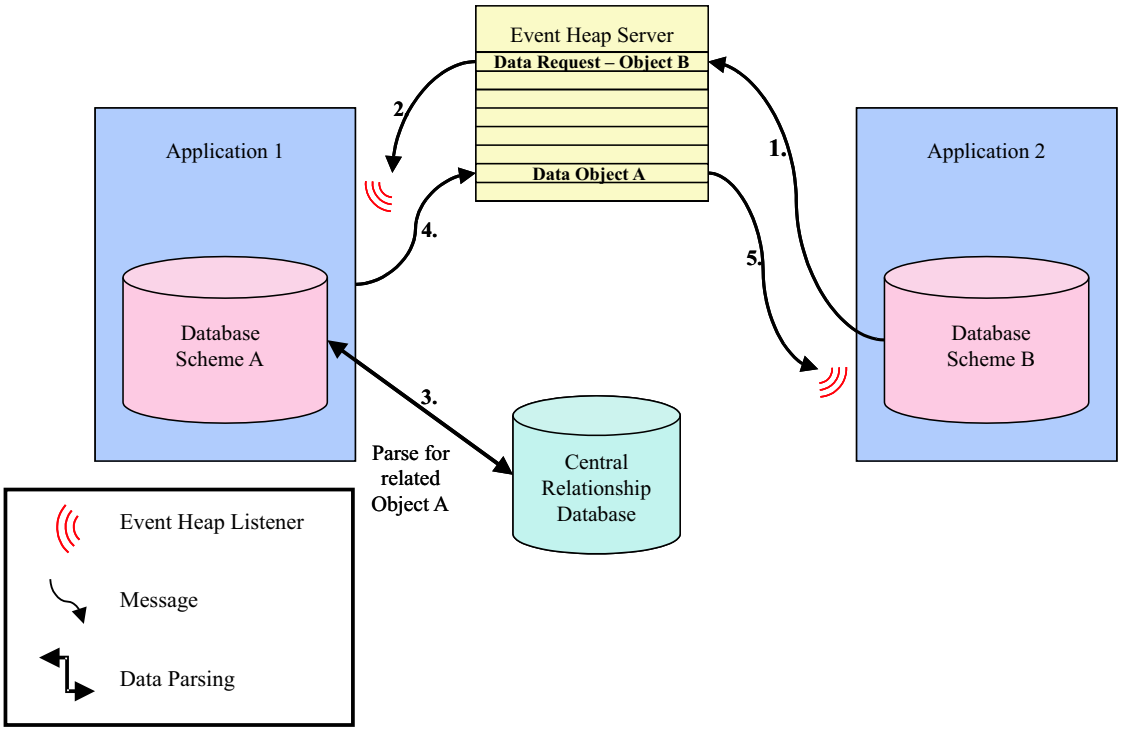


Figure 4.1: Necessary communication using a database storing relationships only

Figure 4.1 shows a typical data transaction architecture between two applications using such a relationship database schema. In this case, *Application 2* wants to receive data from a related object of *Application 1*. As a first step, it

has to send a request message to *Application 1*, containing the *Object (B)* which should be related to the data of some *Object (A)* from *Application 1*. *Application 1* receives the message containing the *Object (B)* and has to parse the central database to map *Object (B)* to the related objects of its own database. After extracting the requested property from the related *Object (A)* the data can be sent again to the event heap, and *Application 2* can pick up the respective message again.

This simple example points out the complexity of the data exchange using a database schema just modelling the relationships between objects. Therefore a significant development effort would have to be invested for each application which is to be integrated into the IRoom. Each application would need to be able to process a number of different message types picked up from the event heap and to be able to send back a number of different message types to the event heap. Another problem is to maintain the data consistency in applications which are using objects of the same type.

The second data storage possibility is to store all the objects that have to be shared by the applications directly into the databases. The relationships between them can be represented as properties of these objects. For example, a *building component* object is established which contains information of its related *construction activities* within the object as relationship properties. References to the applications' internal database entities should be stored as well as object relationship properties. Furthermore, it is possible to store data of the applications' entities which need to be exchanged in the central database. This would reduce the message passing, as the applications can use the database to gain access to object related data, which was formerly only available in the applications. Programs can directly use these data without the need of requesting them from other applications.

Figure 4.2 shows the communication necessary in order to get a reference to the related objects of an other application a central database storing relationship

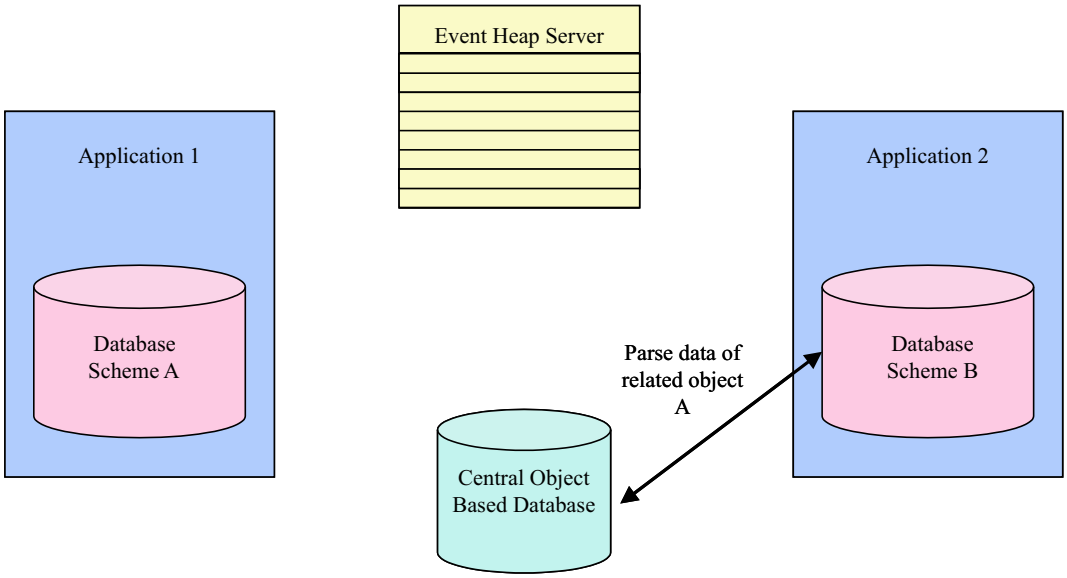


Figure 4.2: Necessary communication using a database storing objects

and data properties with objects. The references to the related objects of *Application A* are stored in the central database. *Application B* just has to query for the objects related to the respective object of *Application A*. The properties of the related objects can then be accessed directly since they are stored as properties of these objects. No direct message passing between both applications is necessary any more.

One disadvantage of this database schema is that relationships have to be stored redundantly with all the objects which are related to each other. Every object needs information of its related objects as relationship properties. If one mutual relationship between two of the objects changes, the relationship properties for both of these objects have to be altered by the database administrator. To simplify the maintenance of the database, these redundancies should be avoided.

This can be realized by using a database structure which stores objects with properties, but defines the relationships between these entities in separate relationship objects. In this way all the data concerning the relations between objects can be normalized. Of course the parsing of databases using this struc-

ture is more complex than that of databases which store the relations as object properties. The reasons for the increased complexity of the database parsing is the increased number of stored objects and the more complex data storage. On the other hand the maintenance of the database gets easier and the schema itself becomes less complex.

4.2 Information Flow within the IRoom Environment

In general, there are two different approaches for database access. In the first approach applications parse the central database directly. An advantage of this approach is that each application accesses the database independently.

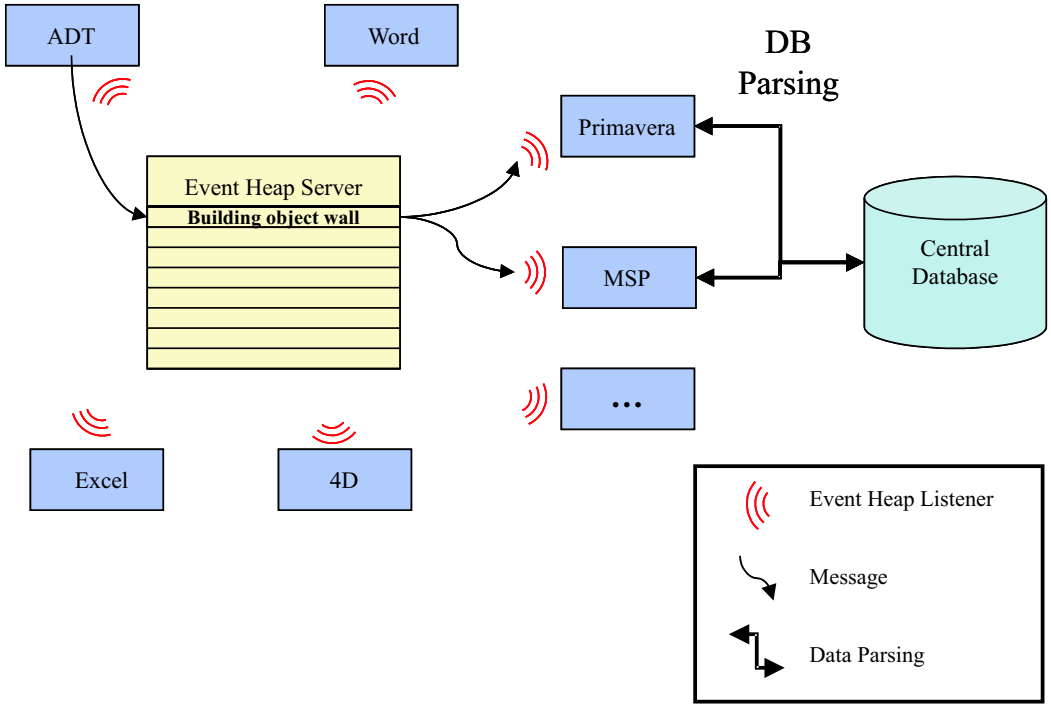


Figure 4.3: Multiple database parsing by direct database access

Of course, this approach leads to a lot more database parsing operations. Ev-

ery application will have to parse the central database separately. Even if there is information which could be shared by different applications, all of them have to perform queries for the shared information independently from each other. Figure 4.3 illustrates this: Primavera and Microsoft Project are applications which represent project management schedule information, like for example *construction activities*. Thus the related objects representing the *construction activities* for a *building component* sent by ADT are the same in the central database. Nevertheless, both applications have to parse the database by themselves looking for the same information. As a result two separated database parsing operations have to be performed.

Another disadvantage is that parsing functionality implemented in one application has to be implemented in all the other applications as well. As most of the programs use different *application programming interfaces* (API) which often support different programming languages, the reuse of already implemented functions is almost impossible. This again results in more programming effort to enable the applications to exchange shared data within the IRoom environment.

One solution for the shortcomings of a direct database access is the use of middle tier software. Such a software application would be responsible for three tasks:

1. Receive data request messages from the applications.
2. Parse the database for the requested information.
3. Send back the required data to the event heap.

Applications can pick up these messages and process the contained information.

A disadvantage of this solution is the dependency of all applications upon this middle tier. All the programs need to pass the messages in the format predetermined by the middle tier application. Furthermore the “listeners” of the

IRoom applications have to be programmed to be able to pick up the messages sent in the specific middle tier format.

Another shortcoming in this respect is that every application has to rely on the middle tier's proper functionality. This is especially a problem with regard to changes within the IRoom environment concerning the central database structure or changes to the integrated applications. As the middle tier application is the central component of the message transaction most of these changes would consequently lead to modifications in the middle tier application as well. Each alteration can cause instabilities within the middle tier due to implementation errors. Thus every code change has to be implemented with great care with respect to the overall stability of the middle tier.

If it is possible to maintain the stability of the middle tier software within the IRoom environment its use offers a wide variety of advantages. As the middle tier software is the central point of the system for the message passing, the message receiving and the database parsing code reuse is possible to a large extent. This means that most of the implemented functionality for integrating one software application in the communication system can be used for the integration of other software programs as well. Furthermore the middle tier concentrates a lot of functionality in one point of the overall system, offering programmers who intend to change or extend the IRoom environment a central spot for their code changes or code extensions.

Another side effect of using a middle tier software is the reduction of database parsing operations and messages sent to the IRoom's event heap. Figure 4.4 illustrates the concise communication within the IRoom environment using a middle tier software. ADT sends a message containing the reference to one of its *building component* objects to the event heap. This message is picked up by the middle tier software. As the middle tier program knows which data are represented by the various applications within the IRoom environment, it can access the database and extract all objects related to the original entity in one

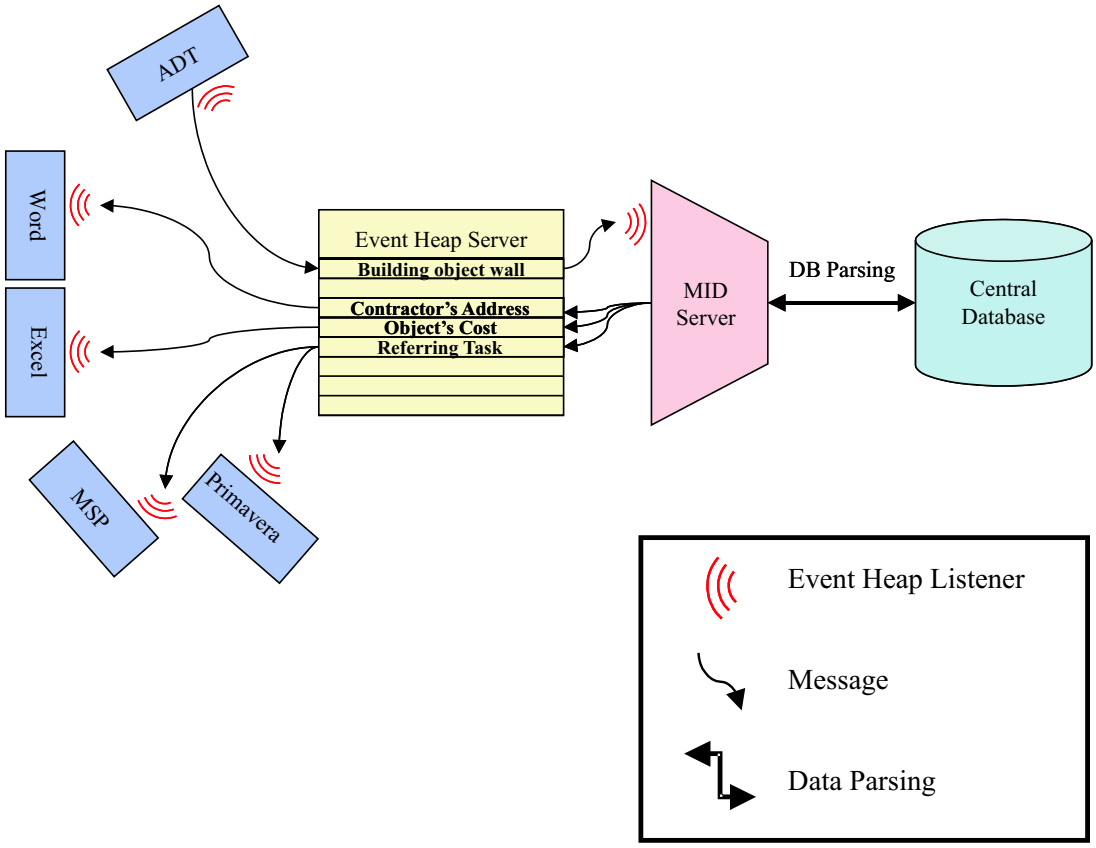


Figure 4.4: Data mapping using a middle tier software

single parsing operation. Now it creates messages containing information needed by the different applications. These messages can then be picked up by the other programs which can process the data included in the messages.

New applications that should be integrated into the IRoom environment can decide whether they use the middle tier software functionality or access the database directly. This offers a great flexibility for the integration of different new programs. Nevertheless, the more applications which do not use the middle tier software are integrated, the more complex the overall message passing and database parsing system gets. Figure 4.5 points out the complexity of message passing for a simple IRoom setup in which three of six integrated applications access the central database directly.

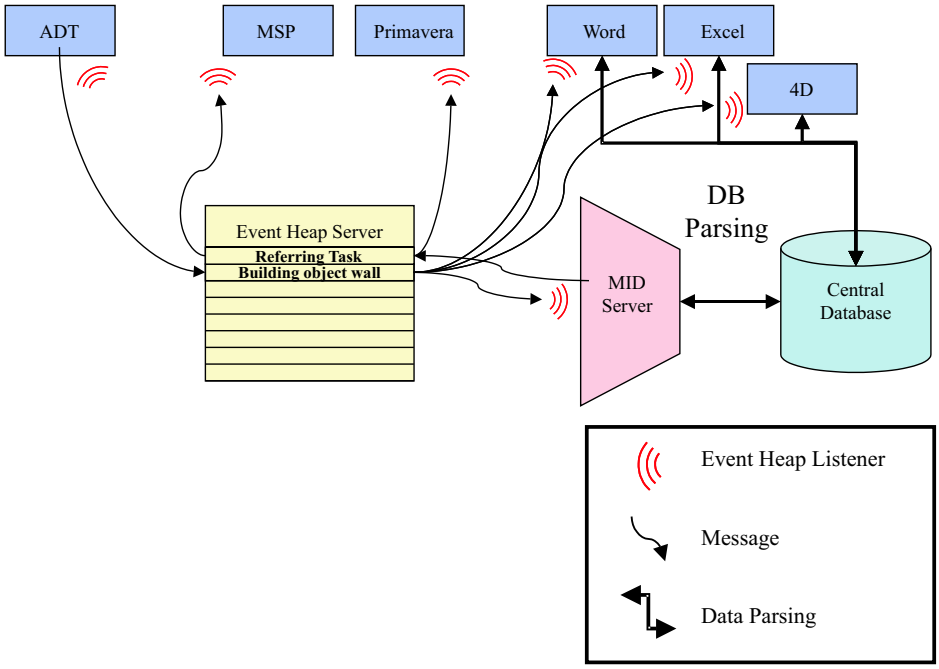


Figure 4.5: Complex communication as result of a combined data base access solution

4.3 Solution for the Integration of the ADT into the IRoom

After carefully taking the advantages and disadvantages of the above described solutions into account an object-based database schema was chosen. Within this schema the existing relationships are modelled by separate objects. These relationship objects represent the links between the objects in the central database as well as the relations to the various representations of these objects in the software applications.

For testing and validation purposes during the implementation the extensible markup language (XML) [W3C - XML] was chosen for representing the database schema and creating simple XML data files. The access to this central XML data files is through the Midserver program which represents a middle tier application.

4.3.1 XML Data Structure

The extensible markup language (XML) is a well established standard for data exchange between applications via the internet. As XML is "a set of rules for designing text formats that let you structure your data" [W3C - XML 10Points] it also can be used to create data files which store information in ASCII format. ASCII files allow an easy exchange of data between different operating systems like Microsoft Windows or Red Hat Linux. Furthermore, ASCII files can be edited by common text editors. This enables the creation and maintenance of XML data files without using a special software application as a variety of different ASCII editors exist for all operating systems.

To create the previously described object-based database for the IRoom, we developed a XML data structure for the integration of ADT representing a variety of different object types, for example, *building components* or *construction activities*. This has been achieved in XML, modeling the different objects as *XML elements*. These elements can be arranged hierarchically. More detailed information can be stored in other hierarchical elements. Thus the information contained in a XML file can be represented at different levels of detail.

Another technique used by XML is the creation of "Document Type Definition" files (DTD). Within these DTDs it is possible to define a general database structure. Using these DTDs it is possible to validate XML data files. A XML data file is valid if its underlying structure matches the structure defined within a DTD. In the IRoom context this technique allows for example to verify whether a specific XML data file can be used for mapping shared data between applications.

Appendix A provides an overview of the general database schema of the central database which is used for the integration of ADT into the IRoom. Furthermore, the different objects and relationship objects are described there. Appendix A also contains the ASCII representation of the DTD file used for this schema. The ASCII code of an XML file describing a part of the Bay Street Project can

be found in Appendix B.

Building Component Object

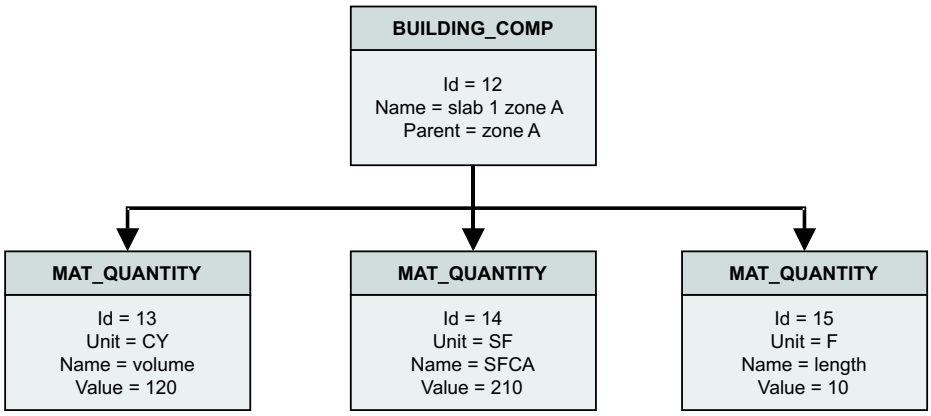


Figure 4.6: Building component object with quantities

The XML data structure uses *building component* objects to represent *structural elements* of ADT within the central XML data files (Figure 4.6). Every structural element of one ADT model can be represented by exactly one *building component* element in a XML database file. At the moment the properties of building components stored in the database are:

- a unique id,
- the name of the component,
- the ids of decomposed components, and
- the id of the parent component.

The unique id has to be explicit for all elements throughout a whole XML data file including elements other than *building components*. This id can be used for directly addressing any element contained in one XML data file. The name of the component can be used for display purposes in different applications. The

ids of the decomposing components and the id of the parent component enable the representation of building component hierarchies within central XML data files.

Every building component can have several material quantity elements in the descending hierarchy of the XML database (Figure 4.6). These material quantity elements are used to store geometric information of the building component objects and can be used, for example, for cost estimation tasks. Examples of such quantities are the volume of the component or the square feet contact area (SFCA) of the surfaces of the component in contact with the formwork. Another example would be the length of pre-manufactured steel columns and beams.

Properties of these quantity elements stored within database files are:

- the unique database id,
- the quantity name (volume, SFCA, length),
- the value of the quantity for the respective building component, and
- the unit of this quantity value.

Other Element Type Objects

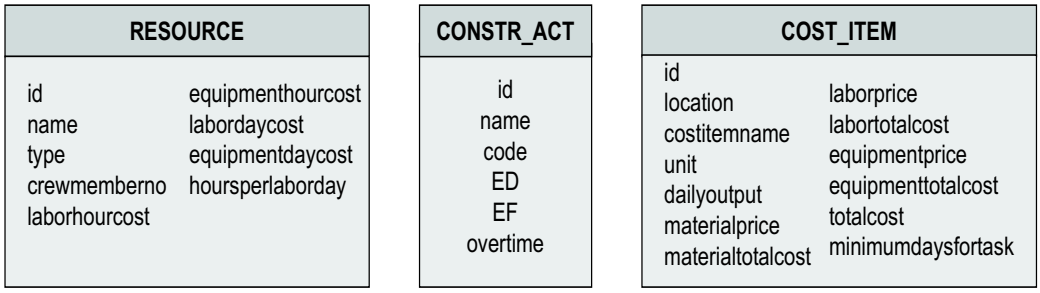


Figure 4.7: Cost Item, Resource and Construction Activity objects

XML data files can define elements of several other types using the described

structure. The XML data file can then use these elements to establish the relationships with ADT building components.

To model the relations with project management applications integrated within the IRoom environment like Primavera or Microsoft Project the XML data structure includes *construction activity* elements.

Furthermore, the data structure uses *cost item* elements and *resource* elements to support the exchange with cost estimation programs like for example Timberline [Timberline] or for the interaction with Microsoft Excel spreadsheets used for the representation of estimation results.

A variety of other elements can also be integrated into the database because the chosen XML data schema provides compatibility between extended database schemas and the previous versions. Thus, by including new elements into the database schema, developers gain the possibility to extend the already implemented IRoom features.

Relationship Objects of the Database

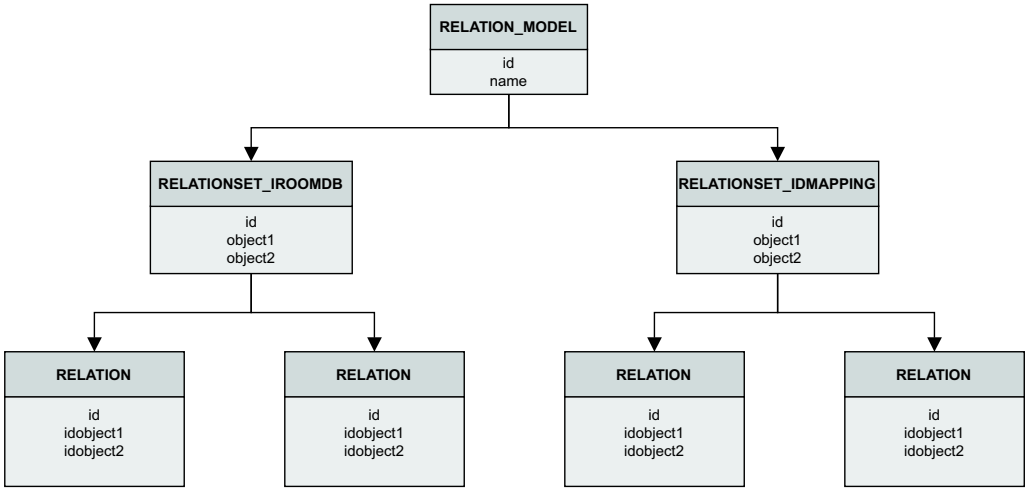


Figure 4.8: Hierarchy of the relationships

As already described, there are two different relationship types to store the

relationship information in central XML data files. The first type describes the relations between objects in the applications' internal databases and the central XML data file. The second describes the relations between two different object types within a central XML data file. A different kind of *relationship set* can be defined for both relation types. These *relationship sets* are a collection of different relation elements representing one kind of relationship. For example, all relationship objects which relate the *building components* within ADT with the elements in the central database are collected within a *RELATIONSHIP_IROOMDB relationship set*. Another example is the *RELATIONSHIP_IDMAPPING relationship set* for the mapping between two objects of the central database like *building components* and *construction activities*.

Information concerning all the relation elements within such a set is defined as properties of *relationship set* elements. *RELATIONSHIP_IROOMDB Relationship sets* with relations concerning the mapping between application and central database objects are storing the following properties:

- the unique database id,
- the application name, and
- the name of the central database element type.

Properties of *RELATIONSHIP_IDMAPPING relationship sets* with relations concerning the mapping between the central XML data file objects are the *unique database id* and both ids of the respective *database element* types.

As the information of the kind of mapping is already stored within the *relationship set* elements one type of relation element can be used to define the specific relationships. These elements have to store information about the unique identifiers of the objects they are relating. In central XML data files this is the *unique database id*. Within every ADT data file the ADT entities can be identified by a unique database id called handle. Thus the string representation of

this handle and the *unique database id* of the referring object within a central XML data file is used within the relation elements of the central database. The following example from a XML data file shows the two different *relationship sets* with corresponding relations:

```
<RELATIONSET_IROOMDB
    id="84"
    object1="BUILDING_COMP"
    object2="CONSTR_ACT">

    <RELATION
        id="86" idobject1="13"
        idobject2="26 29 32"/>

    <RELATION
        id="87" idobject1="16"
        idobject2="22 23 24 25 26 27 28 29 30 31 32"/>

</RELATIONSET_IROOMDB>

<RELATIONSET_IDMAPPING
    id="97"
    object1="IROOM_ID"
    object2="ADT_ID"
    objecttype="BUILDING_COMP">

    <RELATION id="117" object1="13" object2="DD7"/>

    <RELATION id="121" object1="16" object2="DD8"/>

</RELATIONSET_IDMAPPING>
```

As mentioned before, the exact schema of the database DTD and a complete example database XML file can be found in Appendix A.

4.3.2 The Midserver: A Middle Tier Between Applications and Database

We implemented the middle tier program as a standard Microsoft Windows application. This offers the possibility to use predefined graphical user interface techniques. This speeds up the development of user interfaces for the communication with the Midserver application. Using graphical user interaction possibilities it is for example possible to change the underlying XML data file during the runtime of the Midserver program. The extension of the Midserver application is possible if further functionality is necessary in future versions.

The Midserver application is intended to receive data request messages, parsing the database upon these requests and sending back reply messages to the IRoom event heap. Next to the graphical user interface the Midserver needs to support more basic functionalities. First of all the Midserver needs to have an event heap "listener" to be able to pick up the request messages from the event heap. Furthermore, it needs an interface to send messages back to the event heap which contain the reply information which have been obtained from the central database. Both features can be implemented using the event heap software [Stanford CS - EventHeap]. We used Microsoft's MSXML Parser [Microsoft - MSXML] to implement the central database parsing functionality.

Chapter 5

Integrating ADT

In the current state of the IRoom geometrical changes within the context of a project are hardly possible using the IRoom environment. The CP4D application [CPT Tech. - 4D-Viewer] can be used in the IRoom environment for visualisation purposes; a change of the program's underlying data cannot be performed adequately with the existing IRoom technology.

ADT can be used within the IRoom environment to represent, export and change geometrical data. In order to be a contribution to the IRoom, we implemented a software prototypes that extends ADT with the following functionalities:

1. Users have the possibility to create hierarchies of *structural elements*.
2. The prototype enables ADT to automatically export *Building components*, representing ADT *structural components* and the corresponding *RELATIONSET_IROOMDB relation set* to XML data files.
3. Users have the possibility to link the imported *building component* objects with objects of other types within XML data files using *RELATIONSET_IDMAPPING relationship sets*.
4. ADT is able to communicate with the event heap.

This chapter describes these extensions in detail. The following chapters cover the explanation of the implemented prototypes for the ADT extensions and how users can use these prototypes within the ADT environment.

5.1 Hierarchies in ADT Models

Within the planning and design process it is necessary to create a hierarchy of the structural elements of a building. These hierarchies represent the building in different levels of detail. Thus, for example, a more general level of detail can be used to represent opportunities for possible lay down areas on the construction site. Another example would be the use of more detailed views of the model for supporting the installation planning of the building's mechanical systems. These different levels are an important contribution to support the different analysis tasks which have to be accomplished during the overall planning and design process.

Unfortunately ADT's original product model structure does not offer a consistent possibility to create these hierarchies. This is due to the absence of an object which can be used to group *structural elements*. Thus we implemented a *zone* object which can be used to store references to different *structural element* objects of ADT. Using these *zone* objects, it is now easy to create hierarchies. For example for a multiple storey building all the *structural elements* of one level can be collected in a *zone* object representing one level of the respective building. A level based view on the building is now possible by creating such a zone object for each storey of the building.

Using the implemented prototype all *zones* within ADT can have a variety of children elements. These elements can be other *zones* or the common ADT *structural elements*. Parent objects can only be *zones*. Using this technique users can create an arbitrary number of different levels of detail in their ADT model.

5.2 Export of ADT Data to XML Data Files

One of the main problems during the integration of ADT into the IRoom environment is the export of its underlying objects into the XML data files. As described before, the XML data structure has an object type *building component* which is able to represent *structural element* objects within ADT. Further XML *relationship* elements are necessary in order to enable the Midserver to map between these XML data file objects and the respective objects within the ADT application. Hence the implemented prototype should be able to automatically create *relation* and *building component* objects in the used central XML data file for ADT product models that a user wants to integrate into the IRoom environment.

In order to be able to adequately keep the objects within a XML data file and ADT consistent we provided export functions. Users can use these export functionalities to create new elements of the types *relation* and *building component* within IRoom XML data files or update already existing objects automatically.

This automatic export function for structural elements from ADT to the XML data files is performing two tasks. First the function collects all ADT database elements that are representing structural members or zones. Then it extracts all the necessary data from these objects and exports the data into a XML data file.

The prototype stores the respective object ids of children and parents of particular *building components* as XML-attributes of *building component* elements using the attribute names *decomposesInto* and *parent*.

Besides the elements that represent *building components* within central XML data files, the *relationship elements* are created as well. These elements store the unique *ADT database handles* and *the unique ids* of the related *building component* object in the XML data file. Thus using these relationship objects the Midserver can perform the mapping between *structural elements* of ADT and *building components* within the XML data file. The details of the mapping mechanisms in the *relationship elements* of the XML data file have already been

described in detail in chapter 4.

We implemented additional functionalities in order to assist updating of previous established XML data files. Already existing *building component* elements representing ADT *structural elements* in central XML data files can be detected automatically. This is easily achieved by validating whether a *relationship* element is already mapping the unique ADT *database handle* of a *building component* within the respective XML data file. Further the implemented prototype is able to automatically update changed properties of already existing *building component* elements. We realized this by simply comparing the data of the related *building component* element in a XML data file with the respective data of the *structural element* in the ADT database. For example the volume of a slab within ADT is compared with the *value* attribute of the volume *Quantity* element of the respective *building component* within the XML data file.

5.3 Establishing Relationships between ADT Objects and Objects from other Applications

Another functionality with which ADT has been extended is the possibility to establish the relations between the *building component* objects and the other objects of XML data files, such as *cost items*, *labor/material resources* or *construction activities*. These *relationship* elements cannot be created automatically by any of the participating stand alone applications. Users have to specify manually which specific elements have to be linked in the project context. Thus we provided functionalities which enable users to define these relationships between objects of different types. The work to accomplish this new functionality can be described with the following steps:

1. Import all existing XML data file objects of another object type, like *COST_ITEM* or *ACTIVITY* which are to be related with ADT *structural*

element or *zone* objects into ADT.

2. Let the user establish relationships between these objects by providing an adequate graphical user interface within ADT.

3. Export the created relationships back to the respective XML data file.

Especially the second step is a complicated issue. As objects in different branches of the hierarchy tree might have the same names, first all the different objects of two of the above described object types have to be represented hierarchically. Thus users have the possibility to determine for example to which zones different *building component* objects with the same name belong to. A hierarchically representation can be achieved by introducing a tree view which adequately represents the hierarchies in the objects' data structure.

Another even more difficult issue is how the user can relate the objects in the tree to objects in the overall project's context. For example the user needs the exact position of a *structural element* on the site to establish the necessary *relations* with the schedule's *construction activities*. For example it is necessary to know the exact location of a *building component* on the site to establish a relationship with the respective *construction activity* in the schedule. Within the different applications the used objects are represented in a way which allows users easily to gain an understanding of the objects' use in the project's design and planning practice. For example construction activities are represented with their mutual temporal links in schedules and Gantt charts. All the objects in a XML data file, are decoupled of the objects in the applications. In order to be able to link these decoupled objects of different types, the objects' relevance in the project's design and planning practice has to be known by the user. For large projects with a huge number of different objects, users, who want to create object relationships, should be supported in gaining the understanding of the relevance of the different objects. The used application for establishing the relations should provide the

information required. Thus users can adequately create the relationships which are existing in the overall project context between the objects in XML data files.

Here again ADT can be a great contribution to the already existing IRoom environment. As the new implemented relation functionality is working within the ADT environment, it is possible to introduce a highlighting functionality. The to *building component* of a XML data file related *structural element* objects can be highlighted in the ADT representation of the model. Thus the user can easily get a geometrical understanding of one of the *building component* objects selected in the application. This understanding can then support the user in finding the related objects of the other types. For example ADT can display the location of one of the *building component* objects on the site by highlighting the related *structural elements*. This location and context information can be a great help in the selection of the related *construction activities*.

5.4 Connecting Event Heaps

After integrating ADT's *structural elements* and all *relationships* in a XML data file, other applications within the IRoom environment can access this information. In order for ADT to communicate with other applications within the IRoom environment a connection to the IRoom event heap has to be implemented. This event heap connection can be used by other developers intended to program ADT extension functionalities to receive information like specific action commands from other applications.

As described above two different event heap functionalities have to be implemented for every participating application of the IRoom. The first is the "listener" functionality which enables ADT to pick up specific messages from the event heap. The second one is needed by ADT to send messages to the event heap.

Event heap messages which are picked up from the event heap by the "listener"

have the following format:

```
ADT_ID=action:‘...’_app:‘...’_elem:‘...’
```

In this case the first part of the message “ADT_ID” tells the “listener” that this message is intended for ADT. The first tag “action” in the message describes the type of action which ADT should perform upon receiving the message. The second tag “app” holds information about the application which sent the message to the event heap. The third tag “elem” contains the unique ADT database handle of the respective object upon which the specific action has to be performed.

Outgoing messages to the event heap which should be picked up by other applications use the same format. For these outgoing messages the first part of the message has to be the ID of the application for which the message is intended. The “app” tag has to contain the ID of ADT, which is the sending application of the message:

```
‘...’=action:‘...’_app:ADT_ID_elem:‘...’.
```

Chapter 6

Implementation of the ADT Extension Functionalities

We implemented the previously described functions using the object oriented C++ programming language. Due to the modular character of C++ the created functions can be used as basis for other projects, which aim to extend the prototype.

In addition to the already described ObjectARX and OMF C++ libraries we used a variety of different other library extensions during the programming process. Some of them are introduced below.

C++ Standard Library

The C++ standard library is a collection of extension functions to be used with the basic C++ programming language. As the standard library is part of the official C++ standard defined in [C++, 1997], it should be available with each C++ compiler. This ensures that code written in C++ using the standard libraries is platform independent; the written code can be exchanged without alterations between different C++ compilers running on different operating systems like Microsoft Windows or Red Hat Linux.

The parts of the standard library used within this project are the

- container functions,
- string functions, and
- the stream functions.

The *container* functions support the storage of arbitrary data objects. The *containers* differ in how they provide access to the data stored within them. For example provided *containers* are *vectors* programmers can use to access the stored objects by a pre-defined index. Another example are *maps*, which are comparable with *vectors*, but provide the ability to access the stored objects using a user-defined index.

The *string* library offers the possibility to store ASCII text. There are a variety of existing functions which can be used with *string* objects. It is for example possible to search for a character within a *string* object, or extract an arbitrary part of the *string* object into a new *string* object.

The last feature of the standard library which we used for the implementation are the *stream* functions, which, among other things, support the access to files on the harddrive or floppy disk of a computer. More detailed information about the standard library can be found in [Stroustrup, 2000].

Microsoft Foundation Classes (MFC)

The Microsoft Foundation Classes (MFC) are a large collection of C++ functions enabling programmers to access the Microsoft Windows operating system. All the user interface functions provided by ObjectARX and OMF are built on top of MFC. Thus the standard Microsoft Windows behavior for graphical user interfaces is used for extensions of ADT.

In addition to the basic graphical user interface implementation the MFC can be used for a variety of other tasks concerning the Microsoft Windows operating

system. It allows for example the implementation of *multithreading*. *Multithreading* combines different stand-alone processes running at the same time in the same application. Programmers can use these processes to accomplish multiple tasks within one application which have to be performed simultaneously.

Microsoft MSXML Parser

The MSXML Parser version 4 has been used to parse the underlying XML data files describing the shared data of the applications integrated in the IRoom environment. It is possible to read data or write data to ASCII files in XML format using the MSXML parser. The parser's writing functionality ensures automatically that the XML format within the ASCII file is maintained.

Further it is possible to validate the underlying XML data structure with specific DTD files. A short introduction of XML and DTD can be found in chapter 4. More information about the XML parser can be found on the Microsoft website [Microsoft - MSXML].

6.1 Implementation of Zone Objects

We used the programming libraries ObjectArx and OMF to integrate a new *zone* object type within the ADT. Typical built-in ADT entities like doors, windows or walls are characterized by the following features:

- Graphical representations of the entities,
- logical and geometrical relationships to other objects, and
- access to the entities data with the help of graphical user interfaces.

We implemented all these characteristics for the new *zone* entity.

6.1.1 Display Representation of the Zone Objects

In conventional CAD systems, graphical primitives like lines, arcs and circles are used to represent the different objects of a building. Users have to change the used graphical primitives of the objects in order to change its graphical appearance. The user has to change all objects, which should have the same appearance as the changed one. Thus he is loosing a lot of time during the creation and maintenance of the CAD models. Due to its product model the ADT is able to offer a different approach. With ADT it is possible to attach graphical representations to object types. If the appearance of one object type is changed, the appearance of all objects of the same type is adapted automatically by the ADT system. These graphical representations within the ADT are called *display representation* objects of the ADT entities.

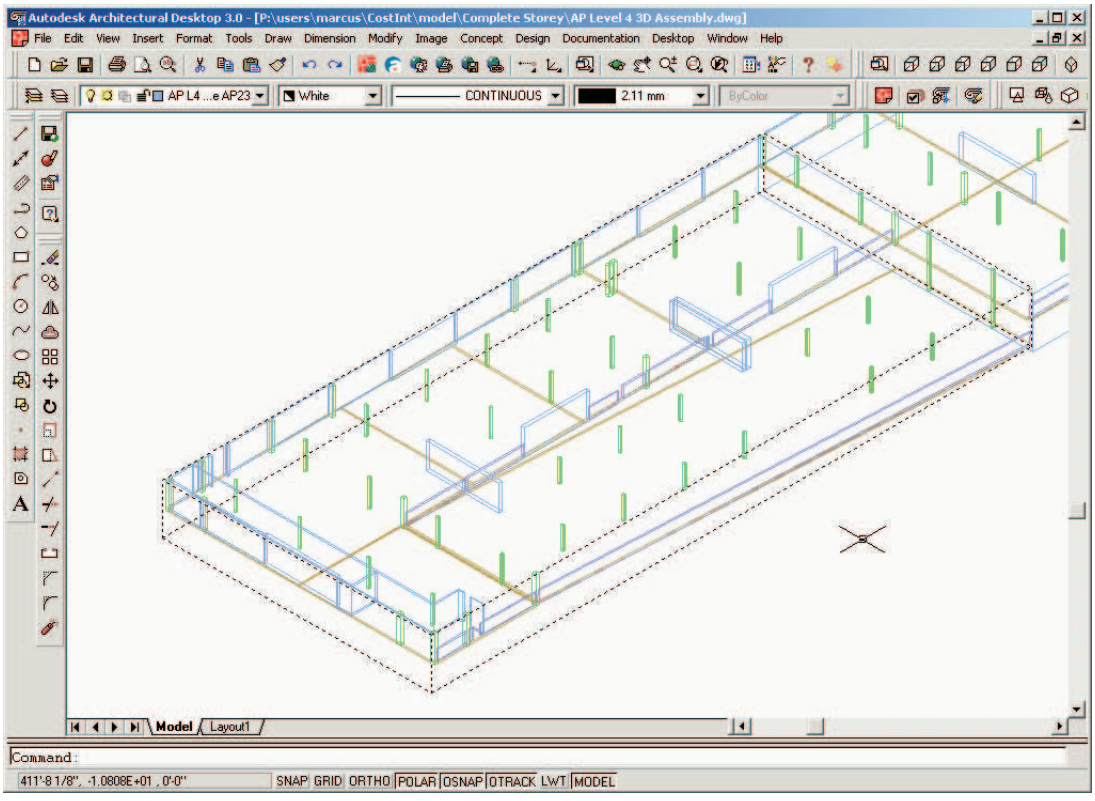


Figure 6.1: *Display Representation* of a zone object

In order to be able to visualize the custom *zone* entities within the ADT we implemented a display representation for these new objects. The *zone* objects are displayed via a bounding box collecting all the zone's structural elements within it (Figure 6.1).

6.1.2 Creating the Relationship between Zone Objects and Structural Elements

The OMF provides a basic support of relationships between different objects. There are two different relations integrated in the ADT, direct links and anchors. Objects, which are related by direct links have a direct knowledge of each other. For instance a wall has a direct reference to its style. Anchored entities do not have direct knowledge of each other. The anchor defines the relationship between both objects; the anchored objects just store a reference to the anchor. For example a wall and a window do not have a direct relation, but an anchor defines how the window is integrated into the wall.

The direct relationships are stored within the ADT in an *object relationship graph*. This graph is automatically updated by the ADT framework every time users edit, erase or add objects. All relationships in the graph are stored bi-directional, that means, that for example a wall is related to a style, and a style is related to all corresponding wall objects. Without the *object relationship graph* ADT would have to perform an extensive data base scan every time it intends to access a related object. ADT stores the following relationships within the *object relationship graph*:

- Object-to-object relationships,
- Class-to-class relationships and
- Objects-to-user interface relationships.

Services provided by OMF using the *object relationship graph* are messaging, delayed notification, object and relationship lookup, and cloning behavior [OMF Developer's Guide, 2002].

A technique to store relationships as properties of other objects is provided by core ObjectArx: the principle of hard and soft pointer ownership. If an entity needs to store a reference, which is pointing to another entity, this reference can either be implemented as a hard or soft pointer. The use of a hard pointer ensures, that the object, which is referenced by the pointer cannot be erased by a user unless the referencing object is erased itself. For example the layer an object belongs to is referenced by hard pointers. Trying to erase the layer is now impossible for users unless all the objects on the layer are purged first. Storing relations with soft pointers does not define any special behavior between the related objects.

The *zone* objects store references to all their *structural member* objects like walls, slabs or roofs. Users have the possibility to purge these elements without erasing the zone first. Hence the *structural members* are stored directly in the zone object with soft pointers (Figure 6.2). We have implemented that all these references are integrated into the relationship graph. Thus we was able to introduce a messaging system which informs the user every time he tries to purge a structural member of one of the defined *zones*.

6.1.3 Access of Zone Object Data by Using User Interfaces (UI)

One of the most important features in modern software architectures are user interfaces. The interaction between the user and the computer must be fast and accurate. ADT already provides a graphical user interface. Any additional elements should therefore be integrated into it by the programmer. In this way users do not have to change any of their normal working behaviors they have

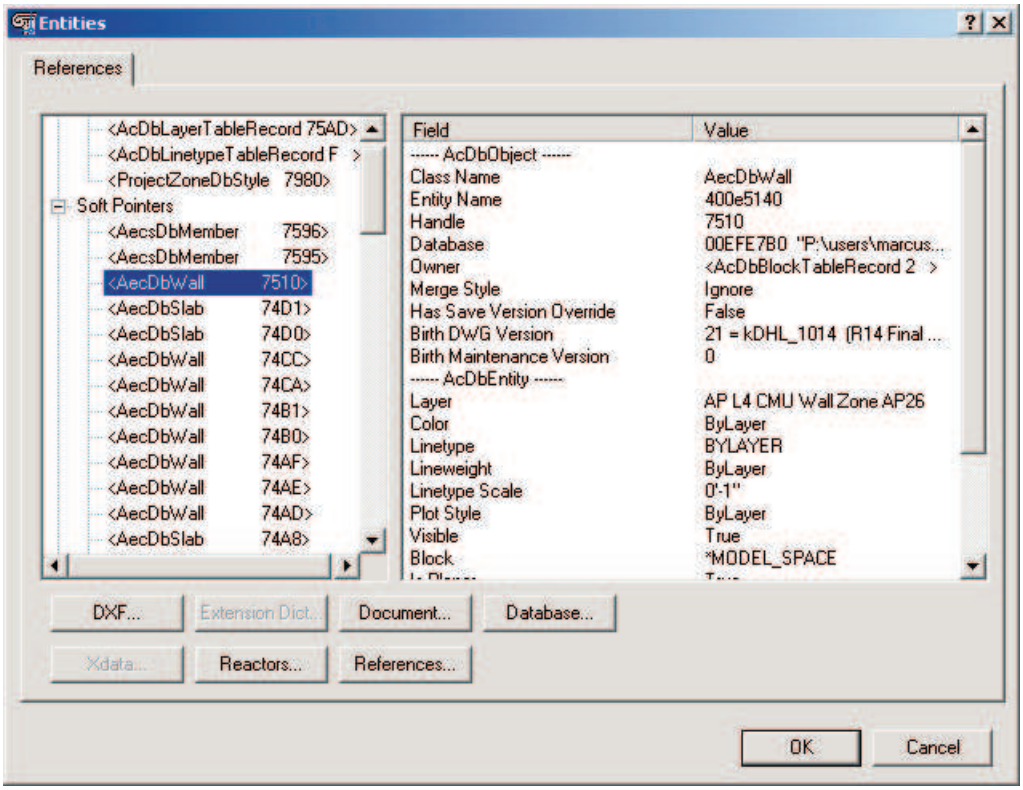


Figure 6.2: References to other objects stored by Soft Pointers in a zone

developed while working with ADT; they can easily access the functionality of the new developed applications.

OMF provides classes and functions, that help developers to modify existing user interfaces or integrate new interfaces within the ADT. Following UI components are supported:

- *Commands*,
- *Prompts*,
- *Dialog Boxes*,
- *Property Pages*,
- *Property Sheets* and

- *Context Menus.*

Prompts are messages on the *command line*, that ask the user for input. OMF provides a variety of different *prompt* types like *prompts* for strings, points, integers or even for entities. Additionally OMF provides functionality to ensure that user input is in the right format or has valid values. For example programmers can implement a *prompt* for ADT entities, which ensures that users can only select entities of a certain kind, all other selected entities are ignored by the ADT prompt. *Prompts* are used by the *zone* objects for the selection of its elements.

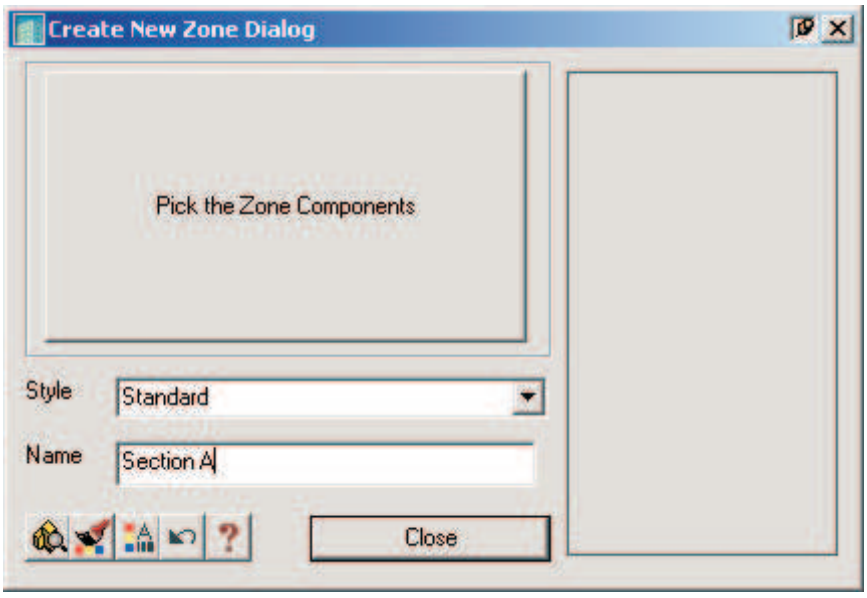


Figure 6.3: ADT standard dialog to add new Zones

Within ADT programmers can use standard Microsoft Windows dialogs in a great variety ranging from general purposes to predefined dialog boxes for special purposes. For example they can use the standard ADT *Create Entity Dialog* for the creation of new entities. For *zone* objects we implemented such a create dialog (Figure 6.3). The user has the possibility to select the elements of the *zone* by pressing the *Pick the Zone Components* button of this dialog. The dialog further enables users to select a style and a name for the respective *zone* object.

The create entity dialog is a good example of the possibilities of the OMF. It enables programmers to extend the ADT without changing the basic functionality provided by Autodesk. Any user who has already gained some experience with the ADT can use the custom *zone* entities accurately, without changing his normal working patterns.

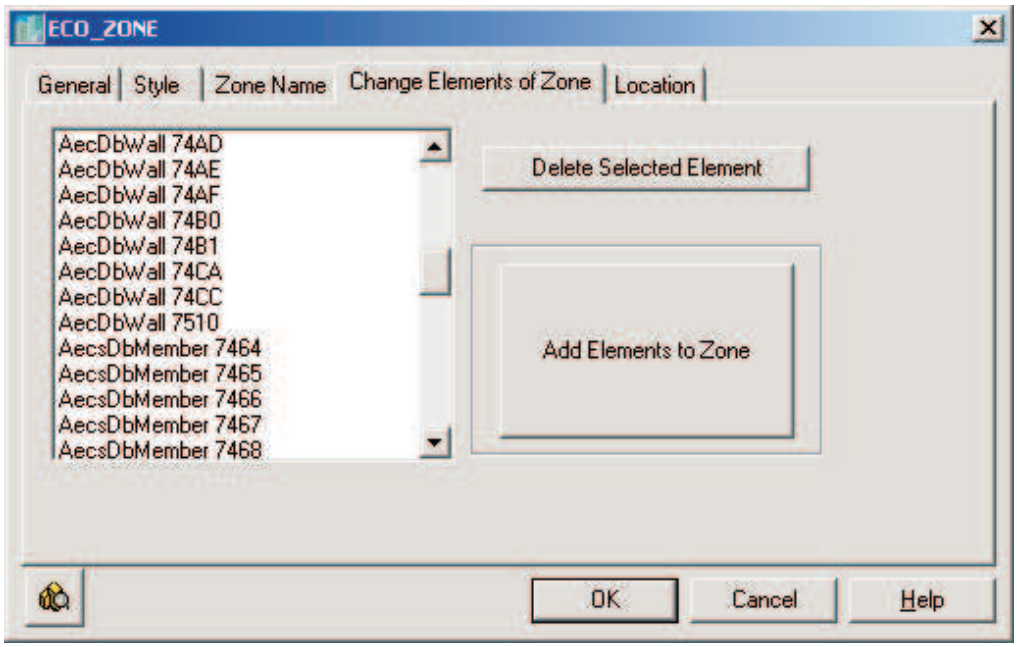


Figure 6.4: ADT standard Edit Property Sheet for zone objects

Table 6.1: Newly added Command Line commands

English Command	German Command	Activates
ZoneStyle	ZonenStil	edit property sheet of zone style
addZone	erstelleZone	create zone dialog
zoneProps	ZonenEigenschaften	edit property sheet of zone entity

In order to enable users to commit changes of the *zone* objects' data after their creation, we implemented a property sheet. *Property sheets* are the tabbed dialog boxes known from Microsoft Windows applications. They can contain

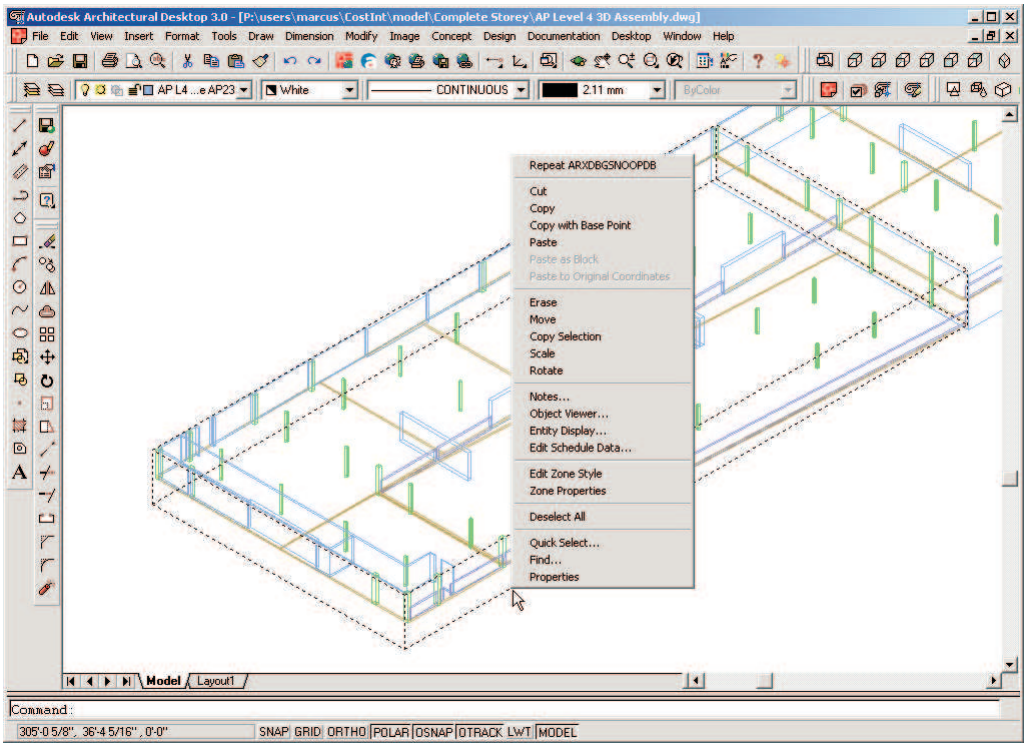


Figure 6.5: Added commands to the context menu for zone interaction

numerous different *property pages*, which allow the user to edit the properties of an object. We implemented two *property pages* for the edit *property sheet* of the *zone* object. Users can change the name of the *zone* with the first page. The second *property page* enables users to delete existing or add new references to *structural elements* to a *zone* object (Figure 6.4).

We further implemented a *right click context menu* (Figure 6.5) in order to enable users to access the new *property sheet*. Further they can access the edit *property sheet* and the *create dialog* of the new *zone* objects via the *command line* commands listed in Table 6.1.

6.2 Implementation of the Data Export to XML Data Files

Using the by OMF and the MSXML libraries provided functionality the implementation of the data export to a central XML data file is straight forward. The overall functionality can be distributed into two tasks:

1. Processing the required information within ADT.
2. Writing the data to a XML data file.

For the processing of the necessary data that has to be exported to a XML data file, we implemented a function that collects all the *structural elements* and *zone* objects within an ADT model. For each of these elements the geometrical data for the *quantity* sub-elements of the respective *building component* objects of the central XML data file are calculated by another function.

The current version of the implemented prototype is able to extract the following quantities:

- volume,
- square foot contact area (SFCA), and
- length.

We implemented the *volume* extraction for all *structural elements* by using ObjectARX. Every *structural element* object is transformed into an ObjectARX body object. These ObjectARX body objects store the volume as a property. Thus we easily could access the volume property and create the corresponding *quantity* object using the obtained value.

The calculation for the value of the *SFCA* is simplified to a great extend. The implemented prototype subtracts the top surface from the overall surrounding

surface of a slab. Thus the function is not working for *structural elements* like slabs on grade which do not need a formwork for the bottom surface. Further the SFCA can only be calculated for *structural elements* with a horizontal or a vertical gravity axis and a regular rectangular shape. A better calculation algorithm should be implemented by programmers for the extraction of the SFCA values in further versions of the prototype.

The prototype is only able to calculate the *length* value for beams and columns. For these elements ADT stores the *length* value as a property and programmers can easily extract it using OMF. For the unit element of the *length quantities* the prototype uses the length unit of ADT and the name of the style. The name of the style of *structural elements* representing beams or columns is commonly used by architects creating ADT models to describe the type of the beams or columns. For example steel elements styles are commonly named by their respective cross-sections, like w16x26, w16x31 or w16x40.

Due to a erroneous function within the ADT version 3.0 the calculation and extraction of the *quantity* objects is only working in ADT version 3.3.

After the prototype has processes the data within the ADT, it creates a new *building component* object for each of the *structural elements*. The prototype adds the respective *quantity* objects as sub elements to each of these *building component* objects:

```
<BUILDING_COMP id="153" name="DD7" parent="230">
  <MAT_QUANTITY id="154" unit="C.Y." name="volume:"
     value="71.9" />
  <MAT_QUANTITY id="155" unit="S.F." name="area:"
     value="625" />
</BUILDING_COMP>
```

In the next step the prototype parses the central XML data file for each of the created *building component* objects. If the object already exists within the XML data file, the prototype updates its properties and quantity sub-elements. If the *building component* object does not exist, the prototype creates a new XML element for the respective object in the XML data file.

Furthermore, for each of the created *building component* elements within the central XML data file the prototype has to create a *relationship* element. These elements use the *unique database ids* of the *building component* in the XML data file and the *unique database handle* of the ADT *structural element* to establish the relationship between the XML data file object and the object in ADT:

```
<RELATIONSET_IDMAPPING id="156" object1="IROOM_ID"
   object2="ADT_ID" objecttype="BUILDING_COMP">

   <RELATION id="157" idobject1="153" idobject2="DD7" />
   <RELATION id="161" idobject1="158" idobject2="DD8" />
   <RELATION id="166" idobject1="162" idobject2="DD9" />
   <RELATION id="171" idobject1="167" idobject2="DDA" />
   <RELATION id="176" idobject1="172" idobject2="DDB" />
   <RELATION id="181" idobject1="177" idobject2="DDC" />

</RELATIONSET_IDMAPPING>
```

6.3 Implementation of the Functionality to Create the Relationship with other Objects

It is not an easy task for users to create the relationships between objects of XML data files with different types. For large civil engineering projects the number of different objects to be linked can easily be more than one thousand. Thus the

implementation of an adequate user interface is extremely important. The user interface has to be able to represent all these objects in a way that the creation of the mutual relationships is easy for the user.

In order to display the objects of two different types, we implemented a dialog with two tree view controls using the MFC library. MFC tree view controls offer the possibility to display hierarchies, in which the XML data file objects are represented. The different elements use the *name* and the *unique database id* of the objects in the underlying XML data file to be displayed in the tree views (Figure 6.6). The use of the *name* is comfortable for the user as it is in most cases descriptive for the task of the object in the overall context. The *unique id* is used because there may be more elements having the same name within a XML data file, especially in the lower parts of the hierarchy.

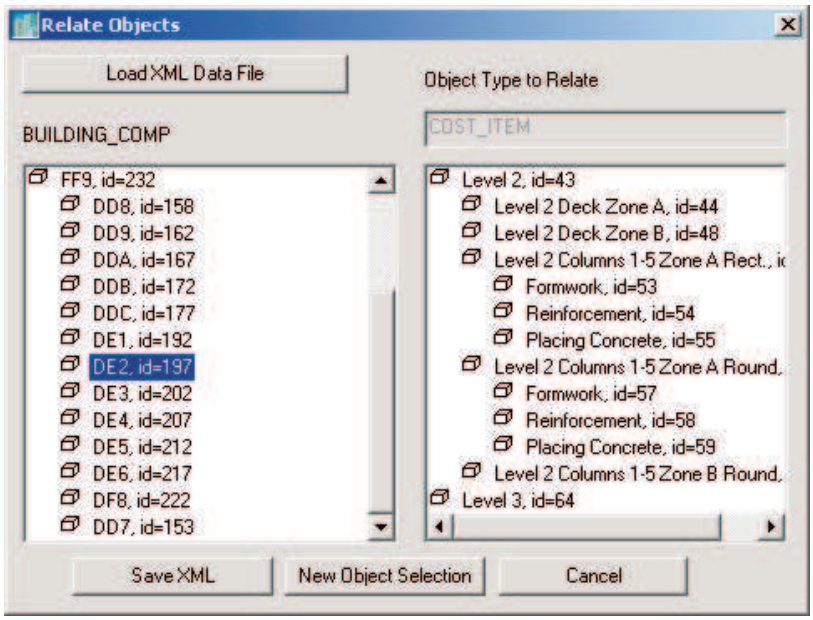


Figure 6.6: Dialog to relate XML Data File objects

We implemented a right click context menu using the MFC functions. It enables the user to get access to further information about the object. Using the *properties* entry within this menu all attributes of the object which are stored in

a XML data file are displayed (Figure **??**).

The implemented relationship application is running within ADT. Thus we was able to implement a highlighting functionality for *structural elements* and *zones* using OMF. The prototype highlights the corresponding *structural elements* in ADT for a selected *building component* in the tree view. Hence the user receives information about the position of the specific *building component* in the geometrical model.

In order to establish the relations, we implemented a MFC drag and drop functionality. Users are able to drag a selected object from one tree to the other. The relationship is created by dropping the selected object over the related object in the neighboring tree.

Selecting the *relations* entry in the right click context menu of an object enables the user to get a list of the already created relationships. Here he has the possibility to erase accidently created relationships with the delete key.

Finally the prototype is able to store all the established relationships within a XML data file. We implemented this functionality using the MSXML parser. As all the relations between objects of the two selected object types are temporally stored within the *relation tool*, it is possible for the prototype to erase all the existing relations of the specific RELATIONSHIP_IDMAPPING element of the XML data file first. Then in a second step the relation tool exports all stored and eventually altered relations to the data file again.

6.4 Connecting ADT to Event Heaps

We implemented a function to access event heaps using the event heap C++ library. In a first step the prototype parses the already mentioned configuration file to specify the name of the event heap and the name of the server the event heap is running on. The configuration file is parsed using the C++ standard library streams. Then the prototype establishes the connection to the event heap using

the event heap C++ library and the parsed information from the configuration file.

We implemented the event heap "listener" in a stand alone process using the MFC multithreading technique. In this way the prototype can observe the event heap constantly without blocking the normal ADT functionality. Every new message on the event heap is picked up by the "listener". The prototype then verifies whether these messages have the right format or not. Then the "listener" sends the messages with the right format to the ADT main process, which is able to perform the necessary actions. At the moment the ADT extension prototype can only process messages for highlighting *structural elements* or *zones* related to *building components*. The messages use the in chapter 5 introduced message format. An example for such a message is:

```
ADT_ID=action:hl_app:MIDSERVER_elem:DD8
```

The last extension functionality of the prototype is the ability to send messages to the event heap from ADT. This feature uses the same connection to the event heap that has been established for the "listener". At the current state there is only one type of message which can be send to the event heap. This message type is containing information about a structural element for which the related objects of other applications should be highlighted:

```
MIDSERVER=action:hl_app:ADT_ID_elem:DD8
```

For creating these messages the user first has to select the respective ADT *structural element* by using the OMF prompt functionality. The prototype extracts the *unique database handle* from the selected element and then it processes the message. Messages of this format can be picked up by the Midserver. Then new messages are created by the Midserver containing the information of related objects in other applications. These applications can now use the Midserver messages to highlight the corresponding objects related to the original ADT *structural element*.

Chapter 7

Use in Practice

Throughout this chapter the use of the implemented ADT extension functionalities will be described. We will explain the steps which have to be accomplished by users for exporting the necessary ADT information to a central XML data file using a test case project in the first part of the chapter. Then we describe the use of the extension functionalities which integrate ADT into the IRoom environment.

We accomplished the implementation of the ADT functionality using ADT extension modules. In order to upload any extension modules ADT is providing the *command line* command "apload". Entering this command the "apload dialog" appears (Figure 7.1). In the upper part of this dialog it is possible to choose a specific extension file. Then this selected file can be uploaded by pressing the dialog's "load" button.

7.1 Used Test Models

We tested the implemented XML data file export functions on ADT models of the Bay Street Project's parking garage consisting of more than 100 different structural elements. Unfortunately it was not possible to carry out any validation

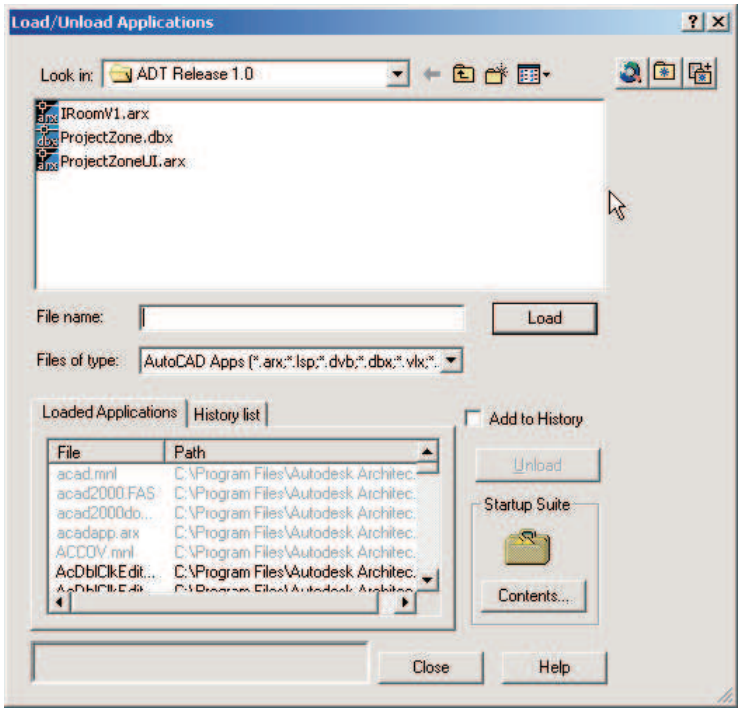


Figure 7.1: ADT dialog for uploading ADT extension modules

for the *relation tool* and the event heap functionality on this large model. This was due to the fact that other applications in the IRoom environment are not yet able to automatically export their objects to the structure of the central XML data files described in chapter 4. The manual generation of such large XML data files that represent a real world civil engineering problem was not possible in the scope of this report.

Thus we chose a more simple scenario. Nevertheless, the test model used throughout this chapter is representing a simplified part of a "real world" civil engineering problem. It consists of one level of the Bay Street Project's parking garage, considering two kinds of *structural elements*, slabs and columns. We purged all other parts in order to keep the size of the model low. Figure 7.2 shows the ADT model of this test case.

We created a schedule for this test case using Microsoft Project. The schedule

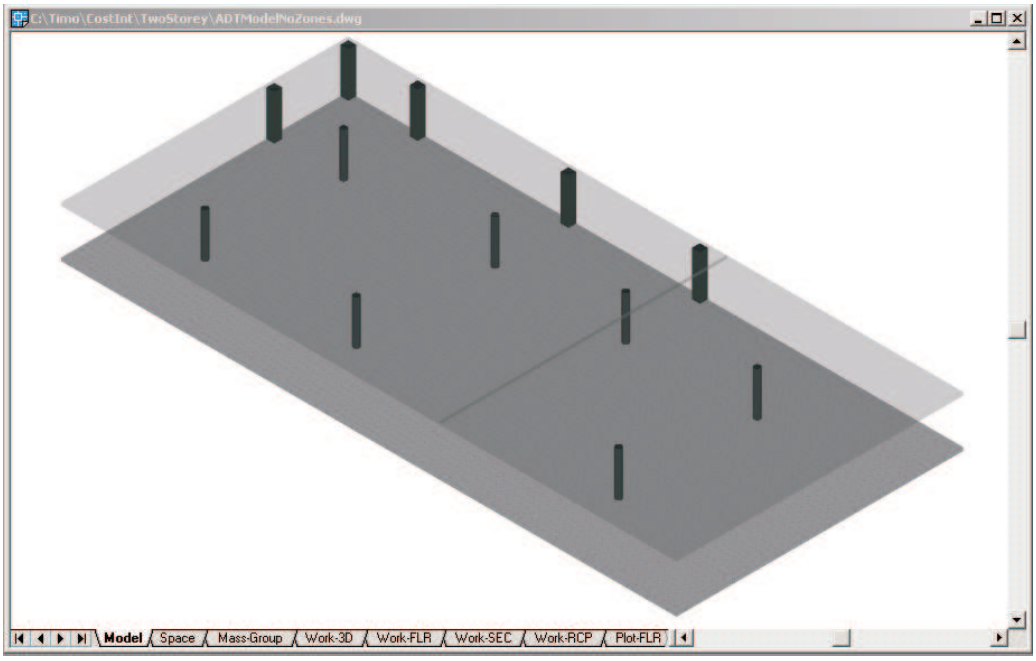


Figure 7.2: Two storey test case model in ADT

defines different *construction activities* and organizes these activities in a hierarchy. The complete test case schedule is illustrated in Figure 7.3.

Furthermore, we created two Microsoft Excel spreadsheets. The first one describes different labor *resources* extracted from the Means collection of Building and Construction Cost Data tables [RSMeans, 2000]. The second one contains the information about a variety of different *cost items*. Tables listing all the labor *resources* and *cost items* used in the test model can be found in Appendix B.

An example XML data file has been created manually. This XML data file contains the above described objects *cost items*, *resources* and *construction activities*. We included no *building component* objects as they can be created automatically with the new ADT export functionality as described later on in this chapter.

Furthermore, we created *relationship sets* with the respective *relation* elements for the object types *construction activity*, *cost item* and *resource*. The

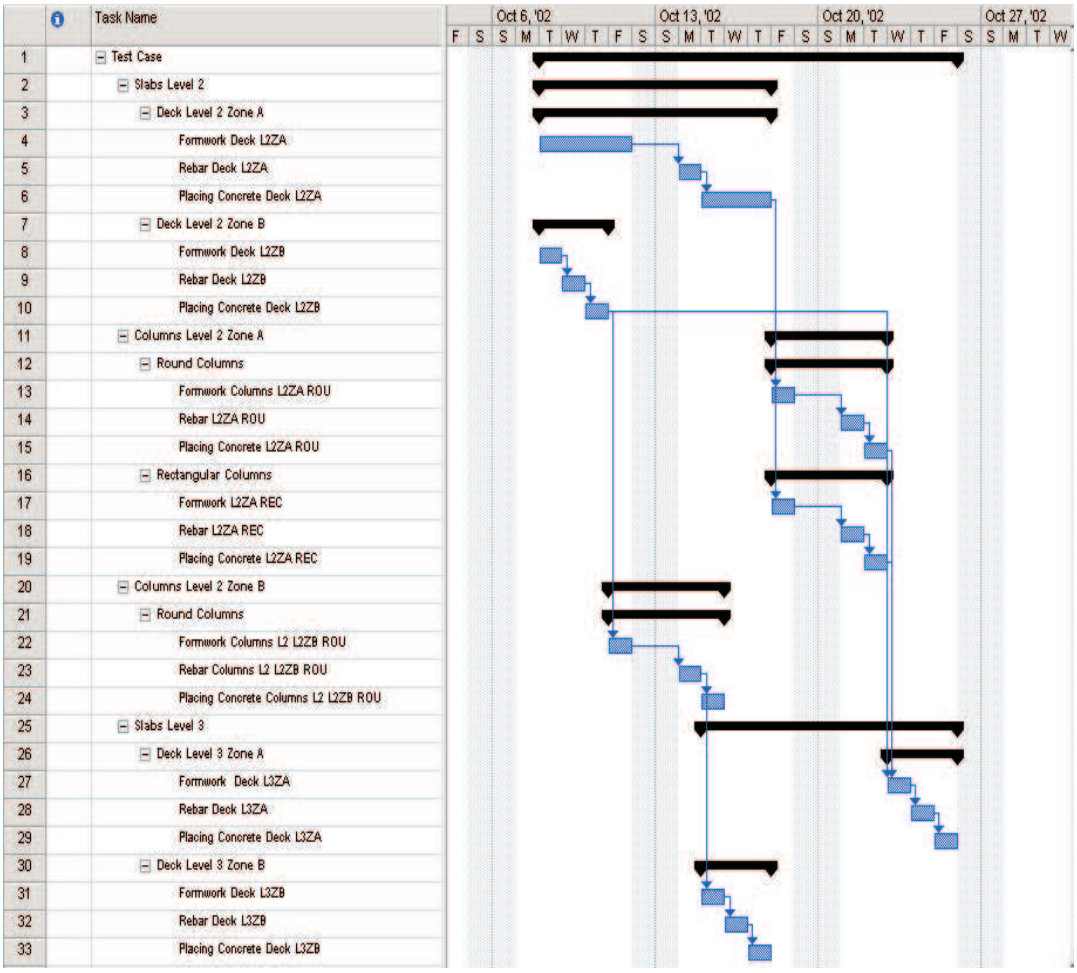


Figure 7.3: Two storey test case model in Microsoft Project

mutual *relation* elements of the different objects with *building component* objects can be created automatically using the *relation tool* as will also be described later on in this chapter.

The created XML data file now can be used as a basis for testing the ADT export extensions. All the necessary steps described in this chapter can be performed using the extended ADT export functionality with this basic file. At the end, a complete XML data file, representing the entire test case should have been created. Interested readers can now use this file to test the extension functionalities for integrating the ADT into the IRoom environment.

7.2 Establishing Hierarchies within ADT

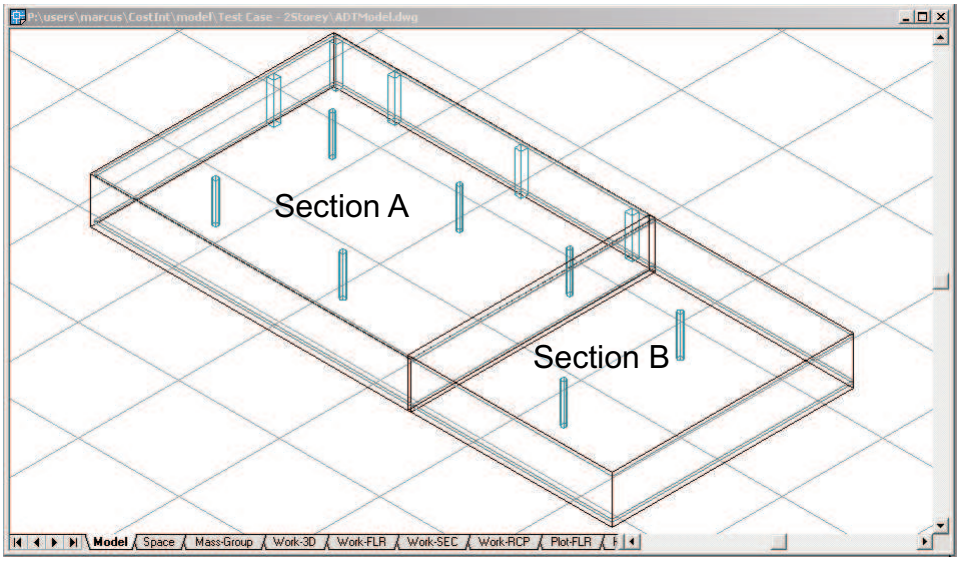


Figure 7.4: ADT test case model with zones

We distributed the implemented functionalities to integrate *zone* objects into ADT into two different ADT extension files. The first, "ProjectZone.dbx", contains all the underlying data functionality of the *zone* objects. The second module, "ProjectZoneUI.arx", provides all the graphical user interface functionality. To use *zones* within ADT users have to upload both extension modules.

Now users can create a new zone within the existing ADT test case model. Using the command "addzone" with the ADT *command line* the "create zone" dialog pops up. The selection of the structural elements can be performed by pressing the "Pick the Zone Components" button. The user is now prompted to select the respective components. The selection can be finished by pressing enter. The "create zone" dialog appears again. In a last step, the zone name and style have to be entered. The created zone is added to the database by pressing the "Cancel" button of the dialog. Users can change properties of already created zones with the "edit zone" property sheet. The change dialog can be called using

the “zoneprops” command or the context menu.

We created two different zone objects for the test case example. The first zone called “Section A” is collecting the two larger slabs with the corresponding columns. The second zone called “Section B” is assembling the two smaller slabs with their corresponding columns. Figure 7.4 shows the newly created zones in the test case ADT model.

7.3 Export from ADT to XML Data Files

The previously described XML data file that contains all necessary data except of the *building components*, can now be used to test and describe the export of the ADT *structural elements* and *zones* and their respective quantities. To be able to have access to the export functionality within ADT the extension module ”IRoomExport.arx” has to be uploaded.

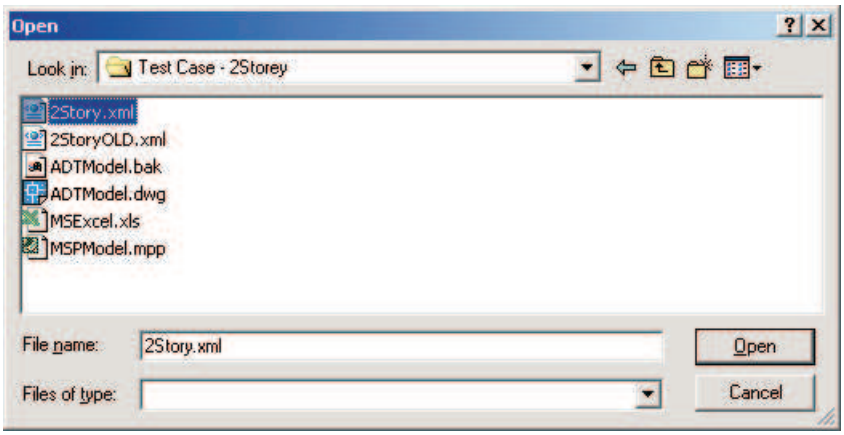


Figure 7.5: Dialog box to select a XML data file for the export

After the uploading of the respective extension module users can start the export function by typing the command “iroomxmlout”. All the user has to do now is to choose the XML data file for the ADT export. This can be easily performed in a dialog for selecting files, known from other Microsoft Windows applications

(Figure 7.5). ADT automatically exports all the *building components* with their respective *quantities* to the selected XML data file. Furthermore the automatic export function creates a *relation set RELATIONSET_IDMAPPING* relating the unique *ADT handles* of the structural elements with the newly created *building components'* ids of the XML data file.

7.4 Creating Mutual Object Relations

After successfully importing the *building component* objects from the ADT into the XML data file, the test case file contains all the necessary objects, needed to describe the project. In a next step the mutual relationships between the objects of type *cost item*, *resource*, *construction activity* and *building component* have to be established by the user. We implemented the *Relation Tool* to support this sophisticated and time consuming task.

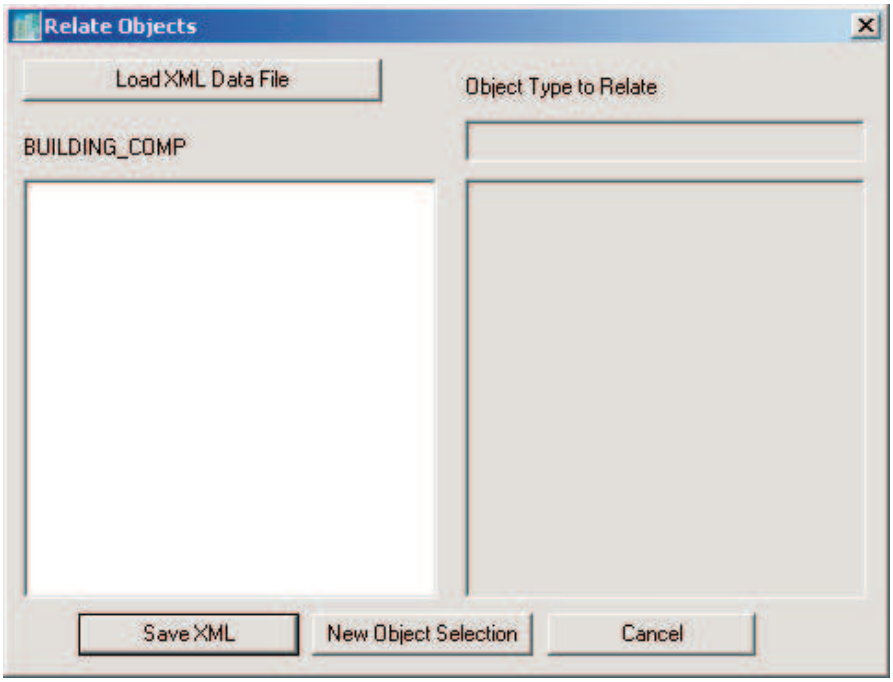


Figure 7.6: The Relation Tool after startup

To be able to start the *Relation Tool*, the Autocad extension "relateInADT.arx" has to be uploaded into ADT first. The *relation tool* itself can be started with the ADT command line command "relateObj". A dialog similar to the one illustrated in Figure 7.6 pops up. The first step in using the *Relation Tool* is to load a XML data file using the "Load XML data file" button, containing the objects which need to be related. Users can again select the respective file using a standard Microsoft Windows file selection dialog box. The left tree control of the *Relation Tool* now already displays the hierarchy of the *building components* in the XML file.

The type name of the objects which are to be related with the *building components* can now be entered in the edit box on the right side of the dialog. For example after entering "COST_ITEM" the right tree control displays the hierarchy of the *cost item* objects of the XML data file (Figure 7.7).

Users can establish the relations between two objects of the trees by simply dragging one object of one tree over another object of the other tree holding the left mouse button down. The relation is created by releasing the left mouse button.

The *Relation Tool* offers several possibilities to support users in establishing the relations. First, the *Relation Tool* highlights all related *structural elements* within the ADT model if the user selects a *building component* within the left tree control (Figure 7.7).

Second, users can gain additional information of a selected object in one of the trees utilizing the implemented right click context menu. There are two different entries within the menu. The complete additional attribute information of an object stored within the XML data file is displayed by selecting the "Properties" entry of the menu. The already related objects of the respective object type in the other tree of the dialog box are displayed by selecting the second menu entry "Related Objects". In both cases a dialog pops up containing the additional information of the object (Figure 7.8).

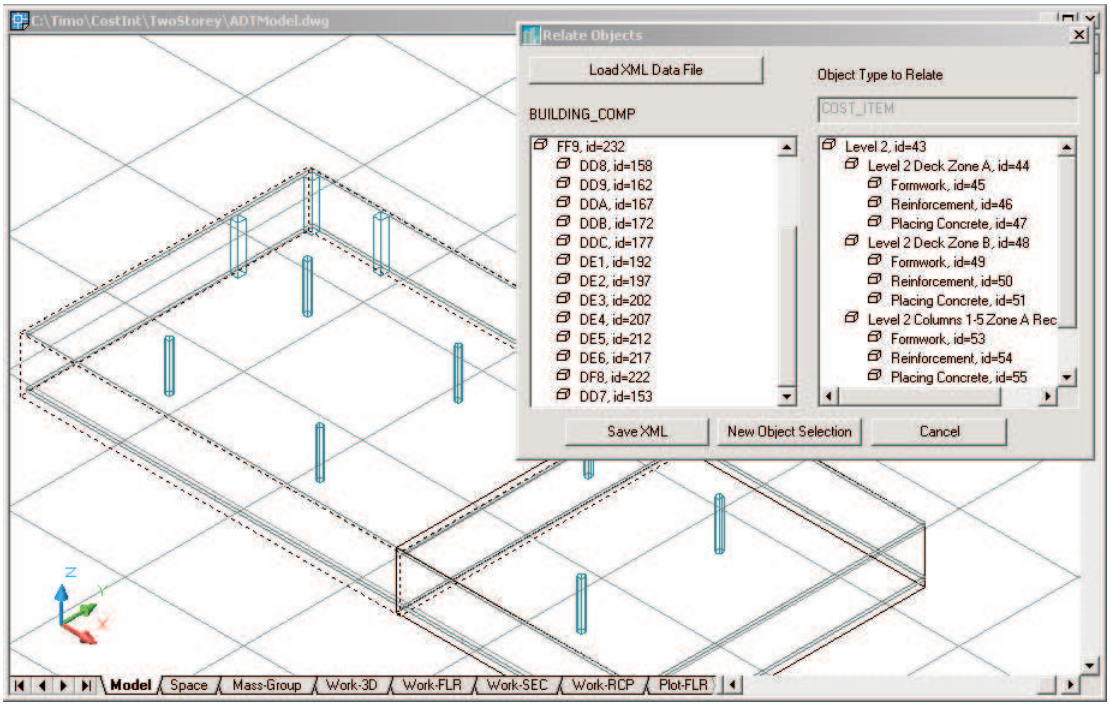


Figure 7.7: Highlighting of structural elements in ADT while working with the Relation Tool

Users can delete wrong relations within the “Related Objects” dialog box by selecting the respective relation and pressing the “delete” key.

In the last step the user can store the created or altered relations within the respective XML file by pressing the “Save XML” button of the *Relation Tool* dialog. A new object type for the relation with the *building component* objects can be selected by pressing the “New Object Selection” button. The “Cancel” button can be used to exit the dialog without saving the created or altered relations to the XML data file.

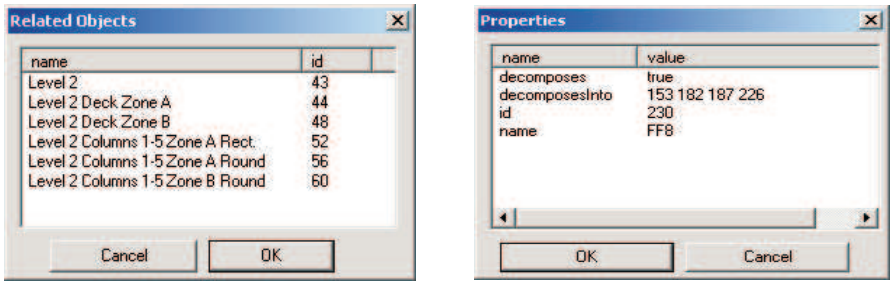


Figure 7.8: Dialogs showing additional object information

7.5 Highlighting Related Objects within the IRoom Environment

After exporting all the necessary information from ADT to a central XML data file, users can now use ADT with the IRoom communication environment. To enable ADT to access the event heap services we implemented another Autodesk extension module, "IRoom.arx". After the user has uploaded the module into ADT the event heap "listener" functionality of ADT is started automatically. ADT can now pick up messages from the event heap, containing information to highlight construction elements or zones within the ADT. An example of the format of these messages can be found below:

```
ADT_ID=action:hl_app:MIDSERVER_elem:DD8
```

This message is then processed by the implemented ADT extension in order to highlight the structural element with the unique database handle "DD8".

Furthermore it is now possible to select a structural element or zone within the ADT for which the related resources, cost items and construction activities should be highlighted within Microsoft Excel or Microsoft Project. An example of the overall message passing between ADT, the Midserver and Microsoft Project is illustrated by Figure 7.10.

In order to be able to send messages which the Midserver is able to transfer into a format understood by Microsoft Project or Excel, users can use the command "highlightSel". After entering the command, the prototype prompts the user to select a set of *structural element* or *zone* objects within the ADT. After finishing the selection with the enter key, the ADT extension functionality creates the necessary messages and sends them to the event heap. There they can be picked up by the Midserver, which creates new messages for the other applications.

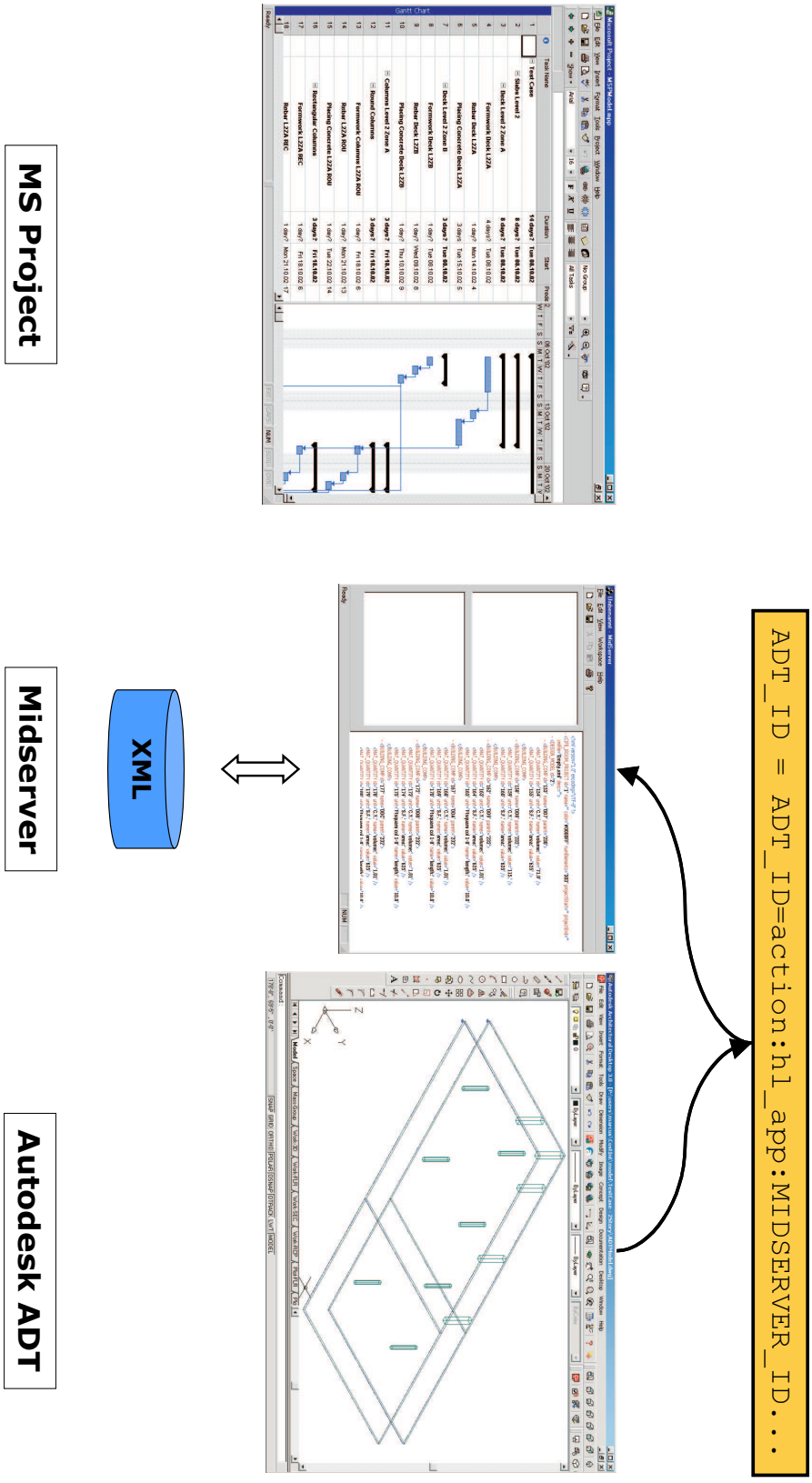


Figure 7.9: Communication between ADT and the Midserver

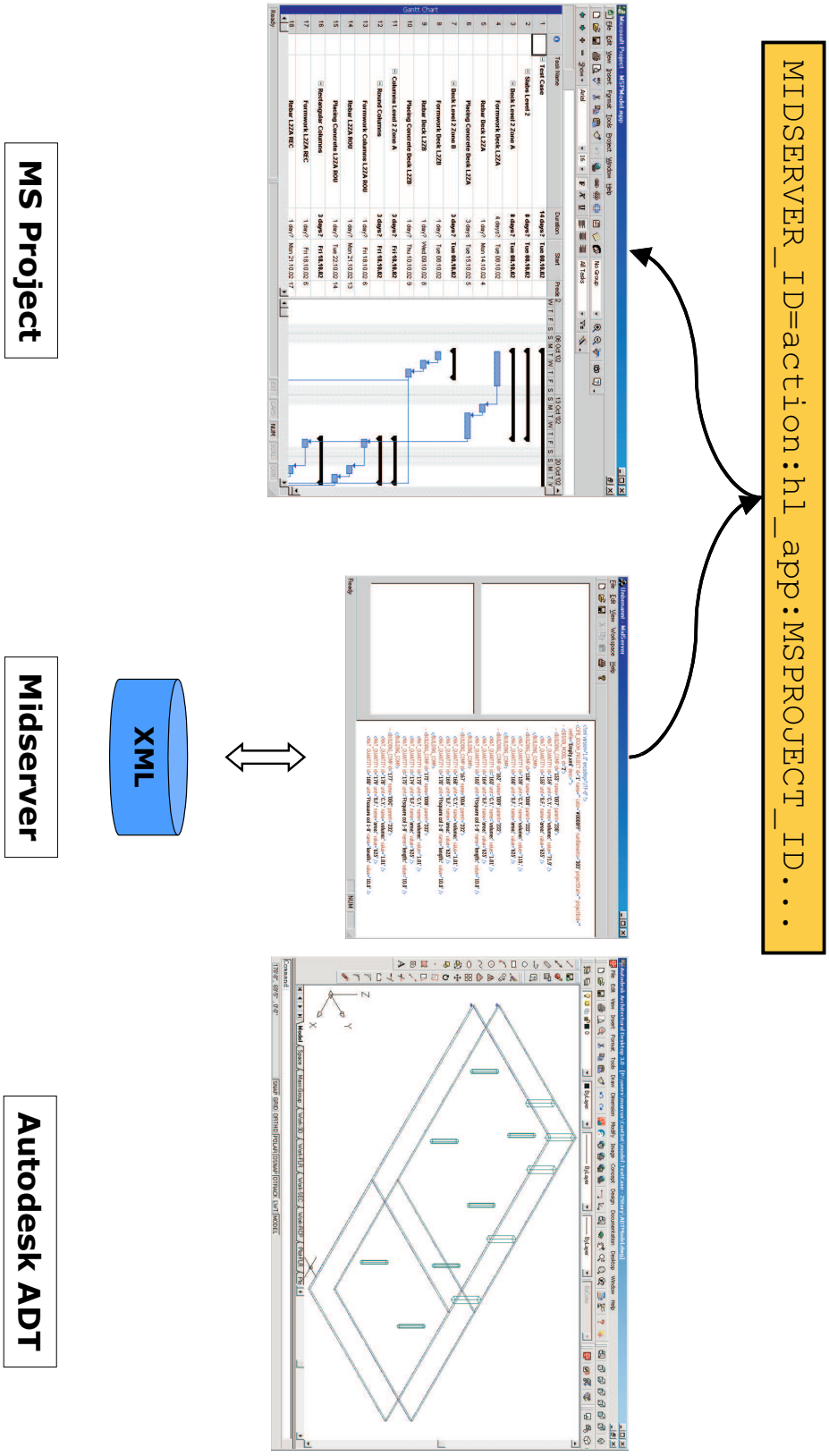


Figure 7.10: Communication between the Midserver and Microsoft Project

Chapter 8

Generalized Methods to Integrate Other Applications into the IRoom

This chapter discusses how to generalize the methods used for the integration of ADT into the IRoom environment. Recapitulating, the ADT integration has been performed in two steps. In the first step, the necessary functions to enable ADT to work with XML data files were developed. In the second step, the necessary functions to allow ADT to communicate with other applications within the IRoom environment were implemented.

8.1 Integration into XML data files

There are two main tasks to be performed for the integration of a new application into the IRoom environment concerning the central XML data file:

- adjusting the XML data file structure and
- implementing an export function for the application.

In the first step the existing XML data structure has to be examined whether it is able to model the new application's objects that need to be shared within the IRoom environment. For the integration of ADT the *building component* object of the data file structure could be used to model the ADT *structural elements*. Nevertheless it is likely that an application represents objects, which cannot be modelled with the current data structure. In this case a new object type has to be introduced. This is possible without affecting the compatibility with former versions of the data file structure.

For example an application can represent the different participants of a project, like owner, architect, contractor and the sub-contractors. The current structure is not able to represent any of these participants with its existing object types *building component*, *cost item*, *construction activity* or *resource*. Hence the underlying data structure has to be extended by a new object type, like for example *participant*.

Every non-relation object within a XML data file needs two attributes, *name* and *id*, to conform with the overall data structure. Hierarchies of the objects should be modelled with the attributes *decomposesInto* and *parent*. There are no other requirements concerning the integration of new object types into the data structure than the two described above. Nevertheless additional object information can be attached as attributes. This additional information can then be used by applications like for example the *relate tool*, to ease the classification of the object in the overall project context. Additional data necessary for the exchange with other applications can be attached to the objects by using sub-elements, like the quantity elements used for *building component* objects.

After the data structure has been adjusted it is important to provide automatic export functions from the new application to XML data files. As seen in the test case of the previous chapter, even a really simplified model is already consisting of a large number of different objects. Thus the manual creation of data files with text editors would be very time consuming and error prone. The

implementation of these export functions is easily possible using XML parsers.

Generally the export has to be performed in two different steps. First, all the XML elements representing the application's objects have to be exported. Only after the completion of this first step the *relation* elements of the *relation set* type RELATIONSET_OBJECTMAPPING can be created. This is due to the fact that the objects' *unique XML data file ids* have to be available in order to create the relations. The export function can create the *relation* elements by using these *unique XML data file ids* of the already exported objects. These *relation* elements are relating the XML data file objects with the respective objects of the applications.

Furthermore, a unique id of the objects within the application should be used with the *relation* elements, like for example the *unique database handle* of ADT. In this way, naming conflicts between different objects within an application can be avoided.

Users can apply the *relate tool* to create the *relation* elements of the *relation set* type RELATIONSET_IROOMMAPPING of the new objects with *building components* within the ADT. If users intend to create relations with other objects they can apply an ADT independent *relate tool* version.

8.2 Implementing the Event Heap Connection

The already mentioned EventHeap technology [Stanford CS - EventHeap] includes an API which can be used to extend the different IRoom applications with the message passing functionality. The implementation of this extension can be carried out using either the C++ or the Java programming language.

There are three different tasks to accomplish for the integration of a new application in the IRoom communication system. First of all the connection to the event heap server has to be established. Using the provided API the implementation of this connection is rather simple.

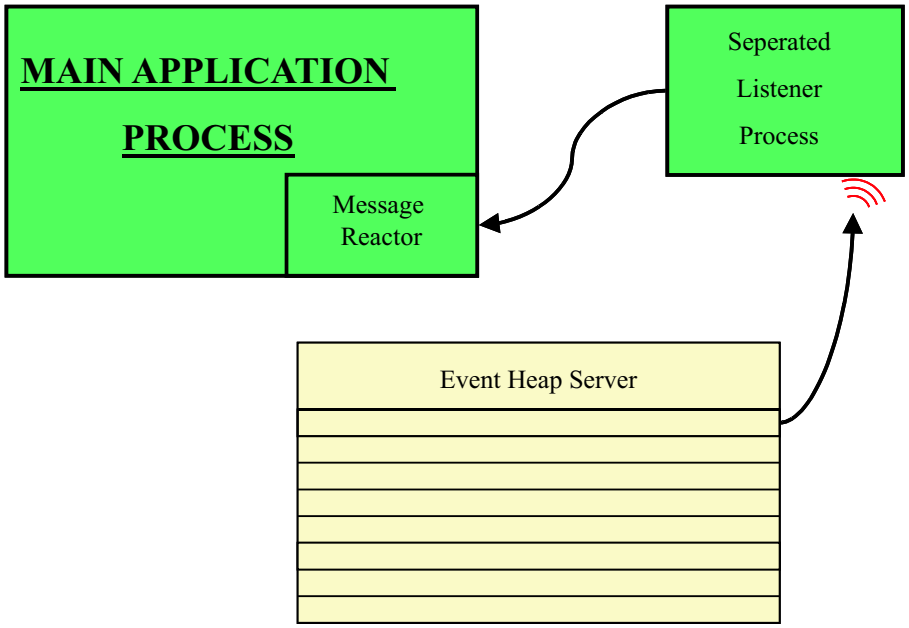


Figure 8.1: Main Application - Listener interaction

The C++ and Java programming languages can be used as well to send messages to the event heap. Furthermore, the sending of messages is possible by using standard URL commands, like they are used with internet browsers:

```
http://cife-32.stanford.edu:8080/iwork/servlet/submitevent?
dest=http://cife-32.stanford.edu:8080/submit.htm&
tuplespace=cife&name0=EventType&type0=int&
name1=TargetID&type1=int&name2=GroupID&type2=int&
value2=1200&name3=NumAccesses&type3=int&
value3=1&name4=TimeToLive&type4=int&value4=-1&
name5=CommandToExecute&type5=string&value1=&
value5=4DViewer\%20&value0=2350
```

Hence it is not even necessary to use any programming language for the sending of messages to the event heap.

The most important task to accomplish is the implementation of the "listener" functionality. The "listener" has to observe the event heap constantly. That means it has to run separately from the main application so it does not block the main program's basic functions. The main program and the "listener" have to run simultaneously in two different processes. If the "listener" now picks up a message it can inform the main program which then has the possibility to react to the message (Figure 8.1).

Unfortunately, most of the Microsoft products like Microsoft Project, Microsoft Excel or Microsoft Word only offer adequate possibility to be programmed by using the Microsoft API Visual Basic (VB) [Microsoft - MSDN]. As the EventHeap API [Stanford CS - EventHeap] does not offer functionality which supports VB another approach was chosen for the integration of the above-mentioned applications. First of all the event heap "listener" was implemented using Java. If the Java "listener" now picks up a message from the event heap, it dynamically starts a VB application. This application in turn remotely drives the Microsoft application to perform the respective reaction. One shortcoming of this solution is that the independent Java "listener" has to be started first in order to enable the Microsoft application to react to messages. Programs offering Java or C++ APIs are able to integrate the "listener" directly into the program by using so called "thread" technologies, which support two different processes running simultaneously in one application. Thus the listener can be started directly by the application, which is of course more preferable. The code details of implementing an integrated C++ "listener" and a combined VB/Java "listener" can be found as well in the appendix.

The sending of messages from applications whose APIs do not support Java or C++ can be performed with the message passing functionality via URLs. Since VB is supporting the sending of URL messages, the passing of event heap messages can be implemented with VB working within the respective application.

Chapter 9

Summary

9.1 Current Possibilities

To be able to use ADT adequately within the IRoom environment four different functionalities have to be provided:

- functionality to create hierarchies of the model,
- functionality to export ADT objects to a central XML data file,
- functionality to relate the exported ADT objects with the other objects available in the respective IRoom setting and
- functionality to use the event heap connection to communicate with other applications within the IRoom environment.

The implemented software prototype extend ADT with these four functionalities. Thus the main objective for this report mentioned in the introduction has been reached. ADT is integrated into the IRoom environment enabling the participation of persons who require geometrical information in IRoom supported meetings.

Therefore it is now possible to use ADT models of the project within IRoom supported project meetings. Using the IRoom highlighting functionality, geometrical information represented by ADT can for example be easily related to schedule information in Microsoft Project or resource information in a Microsoft Excel spreadsheet. Due to these new IRoom functionalities which support descriptive tasks in civil engineering project meetings hopefully more time can be spent on the important predictive tasks in these meetings.

9.2 Necessary Further Developments

One main shortcoming of the current IRoom environment is the missing possibility to automatically export data from other applications like Microsoft Project to the used XML data structure. While ADT export functionality can be used to create the building components of XML data files, the XML elements representing other objects have to be created manually using text editors. This is only applicable for small simplified parts of overall civil engineering projects and is very time consuming and error prone.

Thus as the first step in the further development of the IRoom environment adequate export functionality should be implemented for the already integrated applications. The generalization of the export functions used by ADT can be hopefully a great contribution to this task.

The next problem in the IRoom environment is that more commercial civil engineering applications should be integrated. Especially the Microsoft Excel spreadsheets used for the representation of the *cost item* and *resource* objects should be replaced. Instead applications like SFIRION [SFIRION] or Timberline [Timberline] which are used for estimating civil engineering project costs should be integrated. In further steps other commercial applications can be integrated as well. For example ArchiCAD [Graphisoft] is another three-dimensional CAD environment used in civil engineering which may be enabled to run in the IRoom

environment. Thus architects who are using ArchiCAD for the geometrical representation of project models could participate in IRoom meetings as well. Another example would be the integration of Primavera which is, next to Microsoft Project, another commercial software application representing construction activities in schedules.

Summarizing, it is especially important to have a variety of different integrated applications available in the IRoom environment. This is mainly due to the many different participants working on one civil engineering project, which may all use different kinds of applications. Thus the IRoom can only be supportive in most of the meetings if it is offering a wide range of applicable applications.

Another possibility to extend the IRoom environment's functionality would be the enhancement of the information communicated between the applications. At the current state ADT is only able to receive and send messages in order to highlight its objects or respectively highlight objects in other applications.

In further development steps it would be possible to create messages which could be used to change data in the applications. Thus it would be possible to examine the impacts of changes of the properties of one object on properties of other related objects.

For example in a simple scenario, the thickness of one slab of a building might have to be changed according to a new structural analysis of the building. The change of the thickness automatically would lead to a different volume of the respective slab. The volume again is used by the cost estimation, which again is determining, depending on the labor resource used for the specific estimation item, the duration of the construction activity in the projects schedule.

In the IRoom environment, ADT could send a specific message that a *quantity* of one of its *structural elements* has been changed. This message could now be translated by the Midserver, which creates a new message which tells the Microsoft Excel sheet to update the related *cost item's quantity*. After the calculation within the respective *cost item* has been performed and a change in the

duration of one *construction activity* is occurring, Excel could again send a new message to the Midserver, which is then translated to tell Microsoft Project to change the duration of the related *construction activity*.

9.3 Conclusions

In the current state, the use of ADT in meetings within the IRoom environment is already possible. Due to its geometrical representation possibilities, ADT can be a great contribution within IRoom supported meetings. Especially if problems which need detailed information about geometrical details are discussed, ADT can be very useful. For the finding of solutions which require an overview of the project, the use of the CP4D application is more appropriate.

Nevertheless, the main problem with the current IRoom state is the missing possibility to easily create the XML data files needed for the communication between the different integrated applications. In the current state, an adequate export functionality is only existing for the ADT and the CP4D applications. Unfortunately, the CP4D application is supporting a different data structure which is not compatible with the chosen structure for the ADT export. All other currently integrated applications do not support automatic export functionality. Thus the creation of most parts of a XML data file can only be performed manually using ASCII text editors. This approach is very time consuming and error prone and thus not applicable during the planning and building process of a "real world" project.

This report tried to clarify the steps necessary to completely integrate an application within the IRoom environment. Hopefully the developed techniques can be used for the adequate integration of other applications. These new integrated applications then would enable the IRoom to be supportive in meetings of project teams throughout a whole "real world" civil engineering project.

Appendix A

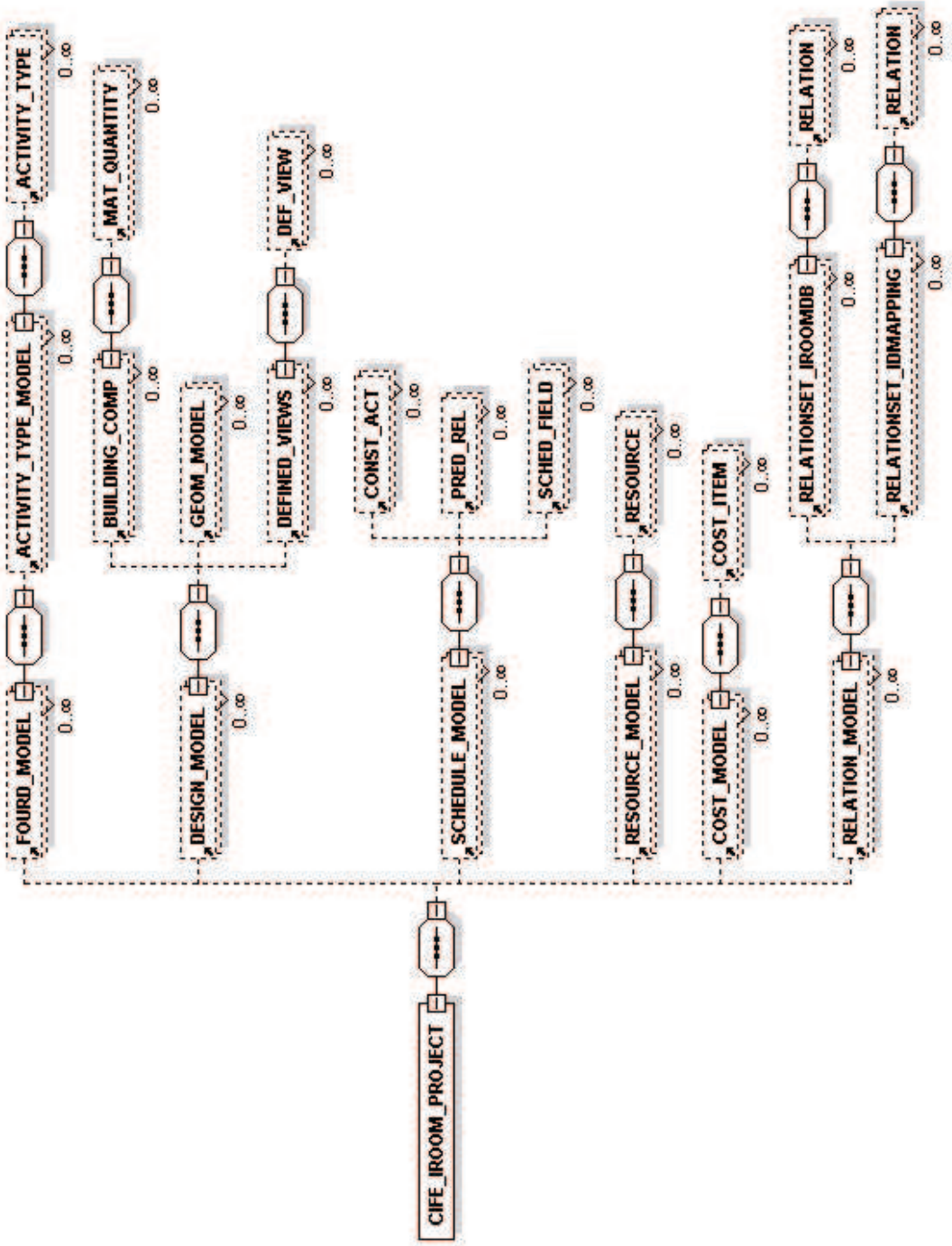
The XML Database DTD

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT CIFE_IROOM_PROJECT
    (FOURD_MODEL*, DESIGN_MODEL*,
    SCHEDULE_MODEL*, RESOURCE_MODEL*,
    COST_MODEL*, RELATION_MODEL*)>

<!ATTLIST CIFE_IROOM_PROJECT
    id CDATA #REQUIRED
    name CDATA #REQUIRED
    color CDATA #IMPLIED
    numElements CDATA #REQUIRED
    projectStart CDATA #IMPLIED
    projectEnd CDATA #IMPLIED
    xmlfile CDATA #REQUIRED
    desc CDATA #IMPLIED
>
<!ELEMENT FOURD_MODEL (ACTIVITY_TYPE_MODEL*)>

<!ATTLIST FOURD_MODEL
    id CDATA #REQUIRED
    name CDATA #REQUIRED
    vfeFile CDATA #IMPLIED
>
<!ELEMENT ACTIVITY_TYPE_MODEL (ACTIVITY_TYPE*)>

<!ATTLIST ACTIVITY_TYPE_MODEL
    id CDATA #REQUIRED
    name CDATA #REQUIRED
>

<!ELEMENT ACTIVITY_TYPE EMPTY>

 <!ATTLIST ACTIVITY_TYPE
    id CDATA #REQUIRED
    name CDATA #REQUIRED
    color CDATA #IMPLIED
    type CDATA #IMPLIED
```

```
>

<!ELEMENT DESIGN_MODEL
(BUILDING_COMP*,GEOM_MODEL*,DEFINED_VIEWS*)>

<!ATTLIST DESIGN_MODEL
    id CDATA #REQUIRED
>

<!ELEMENT BUILDING_COMP (MAT_QUANTITY*)>

<!ATTLIST BUILDING_COMP
    id CDATA #REQUIRED
    name CDATA #REQUIRED
    decomposesInto CDATA #IMPLIED
    parent CDATA #IMPLIED
>

<!ELEMENT MAT_QUANTITY EMPTY>

<!ATTLIST MAT_QUANTITY
    id CDATA #REQUIRED
    unit (Inch | Ft | Y | L.F. | S.F. | SFCA | C.Y. | Ton) #REQUIRED
    name CDATA #REQUIRED
    value CDATA #REQUIRED
>

<!ELEMENT GEOM_MODEL EMPTY>

<!ATTLIST GEOM_MODEL
    id CDATA #REQUIRED
    name CDATA #REQUIRED
    type CDATA #IMPLIED
    filename CDATA #IMPLIED
>

<!ELEMENT DEFINED_VIEWS (DEF_VIEW*)>

<!ATTLIST DEFINED_VIEWS
    id CDATA #REQUIRED
    name CDATA #REQUIRED
>

<!ELEMENT DEF_VIEW EMPTY>

 <!ATTLIST DEF_VIEW
    id CDATA #REQUIRED
    name CDATA #REQUIRED
    camLoc CDATA #IMPLIED
    camDir CDATA #IMPLIED
    upDir CDATA #IMPLIED
>

<!ELEMENT SCHEDULE_MODEL (CONST_ACT*, PRED_REL*, SCHED_FIELD*)>

<!ATTLIST SCHEDULE_MODEL
    id CDATA #REQUIRED
    name CDATA #REQUIRED
>
```

```
<!ELEMENT CONST_ACT EMPTY>

<!ATTLIST CONST_ACT
    id CDATA #REQUIRED
    name CDATA #REQUIRED
    code CDATA #IMPLIED
    ES CDATA #IMPLIED
    EF CDATA #IMPLIED
    overtimework CDATA #IMPLIED
>

<!ELEMENT PRED_REL EMPTY>

<!ATTLIST PRED_REL
    id CDATA #REQUIRED
    name CDATA #REQUIRED
    relConstAct CDATA #IMPLIED
    lag CDATA #IMPLIED
    type (FS | SS) #IMPLIED
>

<!ELEMENT SCHED_FIELD EMPTY>

<!ATTLIST SCHED_FIELD
    id CDATA #REQUIRED
    name    (ACTIVITY | CODE | Duration |
            EF | ES | TF | TYPE)
            #REQUIRED
    isRequired (0 | 1) #IMPLIED
    sortType (0 | 4) #IMPLIED
    colW (100 | 280 | 80 | 90) #IMPLIED
>

<!ELEMENT RESOURCE_MODEL (RESOURCE*)>

<!ATTLIST RESOURCE_MODEL
    id CDATA #REQUIRED
    name CDATA #REQUIRED
>

<!ELEMENT RESOURCE EMPTY>

<!ATTLIST RESOURCE
    id CDATA #REQUIRED
    name CDATA #REQUIRED
    type (crew | equipment) #IMPLIED
    crewmemberno CDATA #IMPLIED
    laborhourcost CDATA #IMPLIED
    equipmenthourcost CDATA #IMPLIED
    labordaycost CDATA #IMPLIED
    equipmentdaycost CDATA #IMPLIED
    hoursperlaborday CDATA #IMPLIED
>

<!ELEMENT COST_MODEL (COST_ITEM*)>

 <!ATTLIST COST_MODEL
    id CDATA #REQUIRED
```

```
    name CDATA #REQUIRED
>

<!ELEMENT COST_ITEM EMPTY>

 <!ATTLIST COST_ITEM
    id CDATA #REQUIRED
    location CDATA #IMPLIED
    name CDATA #REQUIRED
    unit (Inch | Ft | Y | L.F. | S.F. | SFCA | C.Y. | Ton) #IMPLIED
    dailyoutput CDATA #IMPLIED
    materialprice CDATA #IMPLIED
    materialtotalcost CDATA #IMPLIED
    laborprice CDATA #IMPLIED
    labortotalcost CDATA #IMPLIED
    equipmprice CDATA #IMPLIED
    equipmtotcost CDATA #IMPLIED
    totalcost CDATA #IMPLIED
    minimumdaysfortask CDATA #IMPLIED
>

<!ELEMENT RELATION_MODEL
    (RELATIONSET_IROOMDB*,
    RELATIONSET_IDMAPPING*)>

<!ATTLIST RELATION_MODEL
    id CDATA #REQUIRED
    name CDATA #IMPLIED
>

<!ELEMENT RELATIONSET_IROOMDB (RELATION*)>

<!ATTLIST RELATIONSET_IROOMDB
    id CDATA #REQUIRED
    object1 CDATA #REQUIRED
    object2 CDATA #REQUIRED
>

<!ELEMENT RELATIONSET_IDMAPPING (RELATION*)>

<!ATTLIST RELATIONSET_IDMAPPING
    id CDATA #REQUIRED
    object1 CDATA #FIXED "IROOM_ID"
    object2 CDATA #REQUIRED
    objecttype CDATA #REQUIRED
>

<!ELEMENT RELATION EMPTY>

 <!ATTLIST RELATION
    id CDATA #REQUIRED
    idobject1 CDATA #REQUIRED
    idobject2 CDATA #REQUIRED
>
```


CIFE_IROOM_PROJECT
FOUND_MODEL
0..∞
ACTIVITY_TYPE_MODEL
0..∞
ACTIVITY_TYPE
0..∞
DESIGN_MODEL
0..∞
BUILDING_COMP
0..∞
MAT_QUANTITY
0..∞
GEOM_MODEL
0..∞
DEFINED_VIEWS
0..∞
DEF_VIEW
0..∞
SCHEDULE_MODEL
0..∞
CONST_ACT
0..∞
PRED_REL
0..∞
SCHED_FIELD
0..∞
RESOURCE_MODEL
0..∞
RESOURCE
0..∞
COST_MODEL
0..∞
COST_ITEM
0..∞
RELATION_MODEL
0..∞
RELATIONSET_IROOMDB
0..∞
RELATION
0..∞
RELATIONSET_IDMAPPING
0..∞
RELATION
0..∞

Appendix B

Test Model: Simplified Storey of the "Bay Street Project's" Parking Garage

B.1 XML Data File

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE
CIFE_IROOM_PROJECT SYSTEM "iRoom.dtd"> <CIFE_IROOM_PROJECT id="1"
name="" color="#0000FF" numElements="387" projectStart=""
projectEnd="" xmlfile="Empty.xml" desc="">
    <DESIGN_MODEL id="2">
        <BUILDING_COMP id="153" name="DD7" parent="230">
            <MAT_QUANTITY id="154" unit="C.Y." name="volume:" value="71.9"/>
            <MAT_QUANTITY id="155" unit="S.F." name="area:" value="625"/>
        </BUILDING_COMP>
        <BUILDING_COMP id="158" name="DD8" parent="232">
            <MAT_QUANTITY id="159" unit="C.Y." name="volume:" value="115."/>
            <MAT_QUANTITY id="160" unit="S.F." name="area:" value="625"/>
        </BUILDING_COMP>
        <BUILDING_COMP id="162" name="DD9" parent="232">
            <MAT_QUANTITY id="163" unit="C.Y." name="volume:" value="1.01"/>
            <MAT_QUANTITY id="164" unit="S.F." name="area:" value="625"/>
            <MAT_QUANTITY id="165" unit="Ftsquare col 1-8" name="length:" value="10.0"/>
        </BUILDING_COMP>
        <BUILDING_COMP id="167" name="DDA" parent="232">
            <MAT_QUANTITY id="168" unit="C.Y." name="volume:" value="1.01"/>
            <MAT_QUANTITY id="169" unit="S.F." name="area:" value="625"/>
            <MAT_QUANTITY id="170" unit="Ftsquare col 1-8" name="length:" value="10.0"/>
        </BUILDING_COMP>
```

```
<BUILDING_COMP id="172" name="DDB" parent="232">
    <MAT_QUANTITY id="173" unit="C.Y." name="volume:" value="1.01"/>
    <MAT_QUANTITY id="174" unit="S.F." name="area:" value="625"/>
    <MAT_QUANTITY id="175" unit="Ftsquare col 1-8" name="length:" value="10.0"/>
</BUILDING_COMP>
<BUILDING_COMP id="177" name="DDC" parent="232">
    <MAT_QUANTITY id="178" unit="C.Y." name="volume:" value="1.01"/>
    <MAT_QUANTITY id="179" unit="S.F." name="area:" value="625"/>
    <MAT_QUANTITY id="180" unit="Ftsquare col 1-8" name="length:" value="10.0"/>
</BUILDING_COMP>
<BUILDING_COMP id="182" name="DDD" parent="230">
    <MAT_QUANTITY id="183" unit="C.Y." name="volume:" value="456"/>
    <MAT_QUANTITY id="184" unit="S.F." name="area:" value="625"/>
    <MAT_QUANTITY id="185" unit="Ftcirc col 1-4" name="length:" value="10.0"/>
</BUILDING_COMP>
<BUILDING_COMP id="187" name="DDE" parent="230">
    <MAT_QUANTITY id="188" unit="C.Y." name="volume:" value="456"/>
    <MAT_QUANTITY id="189" unit="S.F." name="area:" value="625"/>
    <MAT_QUANTITY id="190" unit="Ftcirc col 1-4" name="length:" value="10.0"/>
</BUILDING_COMP>
<BUILDING_COMP id="192" name="DE1" parent="232">
    <MAT_QUANTITY id="193" unit="C.Y." name="volume:" value="456"/>
    <MAT_QUANTITY id="194" unit="S.F." name="area:" value="625"/>
    <MAT_QUANTITY id="195" unit="Ftcirc col 1-4" name="length:" value="10.0"/>
</BUILDING_COMP>
<BUILDING_COMP id="197" name="DE2" parent="232">
    <MAT_QUANTITY id="198" unit="C.Y." name="volume:" value="456"/>
    <MAT_QUANTITY id="199" unit="S.F." name="area:" value="625"/>
    <MAT_QUANTITY id="200" unit="Ftcirc col 1-4" name="length:" value="10.0"/>
</BUILDING_COMP>
<BUILDING_COMP id="202" name="DE3" parent="232">
    <MAT_QUANTITY id="203" unit="C.Y." name="volume:" value="456"/>
    <MAT_QUANTITY id="204" unit="S.F." name="area:" value="625"/>
    <MAT_QUANTITY id="205" unit="Ftcirc col 1-4" name="length:" value="10.0"/>
</BUILDING_COMP>
<BUILDING_COMP id="207" name="DE4" parent="232">
    <MAT_QUANTITY id="208" unit="C.Y." name="volume:" value="1.01"/>
    <MAT_QUANTITY id="209" unit="S.F." name="area:" value="625"/>
    <MAT_QUANTITY id="210" unit="Ftsquare col 1-8" name="length:" value="10.0"/>
</BUILDING_COMP>
<BUILDING_COMP id="212" name="DE5" parent="232">
    <MAT_QUANTITY id="213" unit="C.Y." name="volume:" value="456"/>
    <MAT_QUANTITY id="214" unit="S.F." name="area:" value="625"/>
    <MAT_QUANTITY id="215" unit="Ftcirc col 1-4" name="length:" value="10.0"/>
</BUILDING_COMP>
<BUILDING_COMP id="217" name="DE6" parent="232">
    <MAT_QUANTITY id="218" unit="C.Y." name="volume:" value="456"/>
    <MAT_QUANTITY id="219" unit="S.F." name="area:" value="625"/>
    <MAT_QUANTITY id="220" unit="Ftcirc col 1-4" name="length:" value="10.0"/>
</BUILDING_COMP>
<BUILDING_COMP id="222" name="DF8" parent="232">
    <MAT_QUANTITY id="223" unit="C.Y." name="volume:" value="115."/>
    <MAT_QUANTITY id="224" unit="S.F." name="area:" value="625"/>
</BUILDING_COMP>
<BUILDING_COMP id="226" name="DF9" parent="230">
    <MAT_QUANTITY id="227" unit="C.Y." name="volume:" value="71.9"/>
    <MAT_QUANTITY id="228" unit="S.F." name="area:" value="625"/>
</BUILDING_COMP>
<BUILDING_COMP id="230" name="FF8" decomposesInto="153 182 187 226 " decomposes="true"/>
```

```
    <BUILDING_COMP id="232" name="FF9" decomposesInto="158 162 167 172 177 192 197 202 207 212 217 222
     " decomposes="true"/>
</DESIGN_MODEL>
<SCHEDULE_MODEL id="3" name="">
    <CONST_ACT id="4" name="Test Case" intid="1" code="1" ES="2002-10-08" EF="2002-10-25" activityType="48"
    decomposesInto="5 14 23 28"/>
    <CONST_ACT id="5" name="Slabs Level 2" intid="2" code="2" ES="2002-10-08" EF="2002-10-17" activityType="48"
    parent="4" decomposesInto="6 10"/>
    <CONST_ACT id="6" name="Deck Level 2 Zone A" intid="3" code="3" ES="2002-10-08" EF="2002-10-17"
    activityType="48" parent="5" decomposesInto="7 8 9"/>
    <CONST_ACT id="7" name="Formwork Deck L2ZA" intid="4" code="4" ES="2002-10-08" EF="2002-10-11"
    activityType="48" parent="6"/>
    <CONST_ACT id="8" name="Rebar Deck L2ZA" intid="5" code="5" ES="2002-10-14" EF="2002-10-14"
    activityType="48" parent="6"/>
    <CONST_ACT id="9" name="Placing Concrete Deck L2ZA" intid="6" code="6" ES="2002-10-15" EF="2002-10-17"
    activityType="48" parent="6"/>
    <CONST_ACT id="10" name="Deck Level 2 Zone B" intid="7" code="7" ES="2002-10-08" EF="2002-10-10"
    activityType="48" parent="5" decomposesInto="11 12 13"/>
    <CONST_ACT id="11" name="Formwork Deck L2ZB" intid="8" code="8" ES="2002-10-08" EF="2002-10-08"
    activityType="48" parent="10"/>
    <CONST_ACT id="12" name="Rebar Deck L2ZB" intid="9" code="9" ES="2002-10-09" EF="2002-10-09"
    activityType="48" parent="10"/>
    <CONST_ACT id="13" name="Placing Concrete Deck L2ZB" intid="10" code="10" ES="2002-10-10"
    EF="2002-10-10" activityType="48" parent="10"/>
    <CONST_ACT id="14" name="Columns Level 2 Zone A" intid="11" code="11" ES="2002-10-18"
    EF="2002-10-22" activityType="48" parent="4" decomposesInto="15 19"/>
    <CONST_ACT id="15" name="Round Columns" intid="12" code="12" ES="2002-10-18" EF="2002-10-22"
    activityType="48" parent="14" decomposesInto="16 17 18"/>
    <CONST_ACT id="16" name="Formwork Columns L2ZA ROU" intid="13" code="13" ES="2002-10-18"
    EF="2002-10-18" activityType="48" parent="15"/>
    <CONST_ACT id="17" name="Rebar L2ZA ROU" intid="14" code="14" ES="2002-10-21" EF="2002-10-21"
    activityType="48" parent="15"/>
    <CONST_ACT id="18" name="Placing Concrete L2ZA ROU" intid="15" code="15" ES="2002-10-22"
    EF="2002-10-22" activityType="48" parent="15"/>
    <CONST_ACT id="19" name="Rectangular Columns" intid="16" code="16" ES="2002-10-18"
     EF="2002-10-22" activityType="48" parent="14" decomposesInto="20 21 22"/>
    <CONST_ACT id="20" name="Formwork L2ZA REC" intid="17" code="17" ES="2002-10-18"
    EF="2002-10-18" activityType="48" parent="19"/>
    <CONST_ACT id="21" name="Rebar L2ZA REC" intid="18" code="18" ES="2002-10-21" EF="2002-10-21"
     activityType="48" parent="19"/>
    <CONST_ACT id="22" name="Placing Concrete L2ZA REC" intid="19" code="19" ES="2002-10-22"
    EF="2002-10-22" activityType="48" parent="19"/>
    <CONST_ACT id="23" name="Columns Level 2 Zone B" intid="20" code="20" ES="2002-10-11"
    EF="2002-10-15" activityType="48" parent="4" decomposesInto="24"/>
    <CONST_ACT id="24" name="Round Columns" intid="21" code="21" ES="2002-10-11" EF="2002-10-15"
     activityType="48" parent="23" decomposesInto="25 26 27"/>
    <CONST_ACT id="25" name="Formwork Columns L2 L2ZB ROU" intid="22" code="22" ES="2002-10-11"
     EF="2002-10-11" activityType="48" parent="24"/>
    <CONST_ACT id="26" name="Rebar Columns L2 L2ZB ROU" intid="23" code="23" ES="2002-10-14"
    EF="2002-10-14" activityType="48" parent="24"/>
    <CONST_ACT id="27" name="Placing Concrete Columns L2 L2ZB ROU" intid="24" code="24"
    ES="2002-10-15" EF="2002-10-15" activityType="48" parent="24"/>
    <CONST_ACT id="28" name="Slabs Level 3" intid="25" code="25" ES="2002-10-15" EF="2002-10-25"
     activityType="48" parent="4" decomposesInto="29 33"/>
    <CONST_ACT id="29" name="Deck Level 3 Zone A" intid="26" code="26" ES="2002-10-23"
    EF="2002-10-25" activityType="48" parent="28" decomposesInto="30 31 32"/>
    <CONST_ACT id="30" name="Formwork  Deck L3ZA" intid="27" code="27" ES="2002-10-23"
    EF="2002-10-23" activityType="48" parent="29"/>
    <CONST_ACT id="31" name="Rebar Deck L3ZA" intid="28" code="28" ES="2002-10-24" EF="2002-10-24"
```

```
    activityType="48" parent="29"/>
    <CONST_ACT id="32" name="Placing Concrete Deck L3ZA" intid="29" code="29" ES="2002-10-25"
    EF="2002-10-25" activityType="48" parent="29"/>
    <CONST_ACT id="33" name="Deck Level 3 Zone B" intid="30" code="30" ES="2002-10-15" EF="2002-10-17"
    activityType="48" parent="28" decomposesInto="34 35 36"/>
    <CONST_ACT id="34" name="Formwork Deck L3ZB" intid="31" code="31" ES="2002-10-15"
    EF="2002-10-15" activityType="48" parent="33"/>
    <CONST_ACT id="35" name="Rebar Deck L3ZB" intid="32" code="32" ES="2002-10-16" EF="2002-10-16"
    activityType="48" parent="33"/>
    <CONST_ACT id="36" name="Placing Concrete Deck L3ZB" intid="33" code="33" ES="2002-10-17"
    EF="2002-10-17" activityType="48" parent="33"/>
</SCHEDULE_MODEL>
<RESOURCE_MODEL id="37">
    <RESOURCE id="38" name="C-2" type="crew" crewmemberno="6" hoursperlaborday="8"
    laborhourcost="27.50" labordaycost="840.64" equipmenthourcost="0" equipmentdaycost="0"/>
    <RESOURCE id="39" name="4 Rodm" type="crew" crewmemberno="4" hoursperlaborday="8" l
    aborhourcost="31.50" labordaycost="1320.00" equipmenthourcost="0" equipmentdaycost="0"/>
    <RESOURCE id="40" name="C-20" type="crew" crewmemberno="6" hoursperlaborday="8" l
    aborhourcost="26.27" labordaycost="2104.96" equipmenthourcost="11.44" equipmentdaycost="91.52"/>
    <RESOURCE id="41" name="C-1" type="crew" crewmemberno="4" hoursperlaborday="8"
    laborhourcost="27.50" labordaycost="2104.96" equipmenthourcost="11.44" equipmentdaycost="91.52"/>
</RESOURCE_MODEL>
<COST_MODEL id="42">
    <COST_ITEM id="43" location="Level 2" name="Level 2" unit="" dailyoutput=""
    materialprice="" laborprice="" equipmprice="" decomposesInto="44 48 52 56 60"/>
    <COST_ITEM id="44" location="Level 2 Deck Zone A" name="Level 2 Deck Zone A"
    unit="" dailyoutput="" materialprice="" laborprice="" equipmprice="" parent="43"
    decomposesInto="45 46 47"/>
    <COST_ITEM id="45" location="Level 2 Deck Zone A" name="Formwork" unit="S.F."
    dailyoutput="520" materialprice="1.44" laborprice="2.54" equipmprice="0" parent="44"/>
    <COST_ITEM id="46" location="Level 2 Deck Zone A" name="Reinforcement" unit="Ton"
    dailyoutput="2.9" materialprice="590" laborprice="350" equipmprice="0" parent="44"/>
    <COST_ITEM id="47" location="Level 2 Deck Zone A" name="Placing Concrete" unit="C.Y."
     dailyoutput="160" materialprice="9.55" laborprice="4.58" equipmprice="14.13" parent="44"/>
    <COST_ITEM id="48" location="Level 2 Deck Zone B" name="Level 2 Deck Zone B" unit=""
    dailyoutput="" materialprice="" laborprice="" equipmprice="" parent="43"
    decomposesInto="49 50 51"/>
    <COST_ITEM id="49" location="Level 2 Deck Zone B" name="Formwork" unit="S.F."
    dailyoutput="520" materialprice="1.44" laborprice="2.54" equipmprice="0" parent="48"/>
    <COST_ITEM id="50" location="Level 2 Deck Zone B" name="Reinforcement" unit="Ton"
    dailyoutput="2.9" materialprice="590" laborprice="350" equipmprice="0" parent="48"/>
    <COST_ITEM id="51" location="Level 2 Deck Zone B" name="Placing Concrete" unit="C.Y."
     dailyoutput="160" materialprice="9.55" laborprice="4.58" equipmprice="14.13" parent="48"/>
    <COST_ITEM id="52" location="Level 2 Deck Zone A" name="Level 2 Columns 1-5 Zone A Rect."
     unit="" dailyoutput="" materialprice="" laborprice="" equipmprice="" parent="43"
     decomposesInto="53 54 55"/>
    <COST_ITEM id="53" location="Level 2 Deck Zone A" name="Formwork" unit="SFCA"
     dailyoutput="187" materialprice="1.77" laborprice="4.55" equipmprice="0" parent="52"/>
    <COST_ITEM id="54" location="Level 2 Deck Zone A" name="Reinforcement" unit="Ton"
    dailyoutput="2.3" materialprice="560" laborprice="375" equipmprice="0" parent="52"/>
    <COST_ITEM id="55" location="Level 2 Deck Zone A" name="Placing Concrete" unit="C.Y."
     dailyoutput="91" materialprice="83" laborprice="16.9" equipmprice="8.05" parent="52"/>
    <COST_ITEM id="56" location="Level 2 Deck Zone A" name="Level 2 Columns 1-5 Zone A Round"
    unit="" dailyoutput="" materialprice="" laborprice="" equipmprice="" parent="43"
    decomposesInto="57 58 59"/>
    <COST_ITEM id="57" location="Level 2 Deck Zone A" name="Formwork" unit="L.F." dailyoutput="150"
     materialprice="2.25" laborprice="5.70" equipmprice="0" parent="56"/>
    <COST_ITEM id="58" location="Level 2 Deck Zone A" name="Reinforcement" unit="Ton"
    dailyoutput="2.3" materialprice="560" laborprice="375" equipmprice="0" parent="56"/>
```

```
    <COST_ITEM id="59" location="Level 2 Deck Zone A" name="Placing Concrete" unit="C.Y."
     dailyoutput="90" materialprice="83" laborprice="17" equipmprice="8.05" parent="56"/>
    <COST_ITEM id="60" location="Level 2 Deck Zone B" name="Level 2 Columns 1-5 Zone B Round"
    unit="" dailyoutput="" materialprice="" laborprice="" equipmprice="" parent="43"
     decomposesInto="61 62 63"/>
    <COST_ITEM id="61" location="Level 2 Deck Zone A" name="Formwork" unit="L.F." dailyoutput="150"
    materialprice="2.25" laborprice="5.70" equipmprice="0" parent="60"/>
    <COST_ITEM id="62" location="Level 2 Deck Zone A" name="Reinforcement" unit="Ton"
    dailyoutput="2.3" materialprice="560" laborprice="375" equipmprice="0" parent="60"/>
    <COST_ITEM id="63" location="Level 2 Deck Zone A" name="Placing Concrete" unit="C.Y."
    dailyoutput="90" materialprice="83" laborprice="17" equipmprice="8.05" parent="60"/>
    <COST_ITEM id="64" location="Level 3" name="Level 3" unit="" dailyoutput="" materialprice=""
     laborprice="" equipmprice="" decomposesInto="65 69"/>
    <COST_ITEM id="65" location="Level 3 Deck Zone A" name="Level 2 Deck Zone A" unit=""
    dailyoutput="" materialprice="" laborprice="" equipmprice="" parent="64" decomposesInto="66 67 68"/>
    <COST_ITEM id="66" location="Level 3 Deck Zone A" name="Formwork" unit="S.F."
    dailyoutput="520" materialprice="1.44" laborprice="2.54" equipmprice="0" parent="65"/>
    <COST_ITEM id="67" location="Level 3 Deck Zone A" name="Reinforcement" unit="Ton"
    dailyoutput="2.9" materialprice="590" laborprice="350" equipmprice="0" parent="65"/>
    <COST_ITEM id="68" location="Level 3 Deck Zone A" name="Placing Concrete" unit="C.Y."
     dailyoutput="160" materialprice="9.55" laborprice="4.58" equipmprice="14.13" parent="65"/>
    <COST_ITEM id="69" location="Level 3 Deck Zone B" name="Level 2 Deck Zone B" unit=""
    dailyoutput="" materialprice="" laborprice="" equipmprice="" parent="64" decomposesInto="70 71 72"/>
    <COST_ITEM id="70" location="Level 3 Deck Zone B" name="Formwork" unit="S.F."
    dailyoutput="520" materialprice="1.44" laborprice="2.54" equipmprice="0" parent="69"/>
    <COST_ITEM id="71" location="Level 3 Deck Zone B" name="Reinforcement" unit="Ton"
    dailyoutput="2.9" materialprice="590" laborprice="350" equipmprice="0" parent="69"/>
    <COST_ITEM id="72" location="Level 3 Deck Zone B" name="Placing Concrete" unit="C.Y."
     dailyoutput="160" materialprice="9.55" laborprice="4.58" equipmprice="14.13" parent="69"/>
</COST_MODEL>
<RELATION_MODEL id="73" name="">
    <RELATIONSET_IROOMDB object1="COST_ITEM" object2="CONST_ACT" id="76">
        <RELATION idobject1="43" idobject2="14" id="77"/>
        <RELATION idobject1="43" idobject2="23" id="78"/>
        <RELATION idobject1="43" idobject2="5" id="79"/>
        <RELATION idobject1="44" idobject2="6" id="80"/>
        <RELATION idobject1="45" idobject2="7" id="81"/>
        <RELATION idobject1="46" idobject2="8" id="82"/>
        <RELATION idobject1="47" idobject2="9" id="83"/>
        <RELATION idobject1="48" idobject2="10" id="84"/>
        <RELATION idobject1="49" idobject2="11" id="85"/>
        <RELATION idobject1="50" idobject2="12" id="86"/>
        <RELATION idobject1="51" idobject2="13" id="87"/>
        <RELATION idobject1="52" idobject2="19" id="88"/>
        <RELATION idobject1="53" idobject2="20" id="89"/>
        <RELATION idobject1="54" idobject2="21" id="90"/>
        <RELATION idobject1="55" idobject2="22" id="91"/>
        <RELATION idobject1="56" idobject2="15" id="92"/>
        <RELATION idobject1="57" idobject2="16" id="93"/>
        <RELATION idobject1="58" idobject2="17" id="94"/>
        <RELATION idobject1="59" idobject2="18" id="95"/>
        <RELATION idobject1="60" idobject2="24" id="96"/>
        <RELATION idobject1="61" idobject2="25" id="97"/>
        <RELATION idobject1="62" idobject2="26" id="98"/>
        <RELATION idobject1="63" idobject2="27" id="99"/>
        <RELATION idobject1="64" idobject2="28" id="100"/>
        <RELATION idobject1="65" idobject2="29" id="101"/>
        <RELATION idobject1="66" idobject2="30" id="102"/>
        <RELATION idobject1="67" idobject2="31" id="103"/>
```

```
    <RELATION idobject1="68" idobject2="32" id="104"/>
    <RELATION idobject1="69" idobject2="33" id="105"/>
    <RELATION idobject1="70" idobject2="34" id="106"/>
    <RELATION idobject1="71" idobject2="35" id="107"/>
    <RELATION idobject1="72" idobject2="36" id="108"/>
</RELATIONSET_IROOMDB>
<RELATIONSET_IROOMDB object1="COST_ITEM" object2="RESOURCE" id="109">
    <RELATION idobject1="45" idobject2="38" id="110"/>
    <RELATION idobject1="46" idobject2="39" id="111"/>
    <RELATION idobject1="47" idobject2="40" id="112"/>
    <RELATION idobject1="49" idobject2="38" id="113"/>
    <RELATION idobject1="50" idobject2="39" id="114"/>
    <RELATION idobject1="51" idobject2="40" id="115"/>
    <RELATION idobject1="53" idobject2="41" id="116"/>
    <RELATION idobject1="54" idobject2="39" id="117"/>
    <RELATION idobject1="55" idobject2="40" id="118"/>
    <RELATION idobject1="57" idobject2="41" id="119"/>
    <RELATION idobject1="58" idobject2="39" id="120"/>
    <RELATION idobject1="59" idobject2="40" id="121"/>
    <RELATION idobject1="61" idobject2="41" id="122"/>
    <RELATION idobject1="62" idobject2="39" id="123"/>
    <RELATION idobject1="63" idobject2="40" id="124"/>
    <RELATION idobject1="66" idobject2="38" id="125"/>
    <RELATION idobject1="67" idobject2="39" id="126"/>
    <RELATION idobject1="68" idobject2="40" id="127"/>
    <RELATION idobject1="70" idobject2="38" id="128"/>
    <RELATION idobject1="71" idobject2="39" id="129"/>
    <RELATION idobject1="72" idobject2="40" id="130"/>
</RELATIONSET_IROOMDB>
<RELATIONSET_IROOMDB object1="RESOURCE" object2="CONST_ACT" id="131">
    <RELATION idobject1="38" idobject2="11" id="132"/>
    <RELATION idobject1="38" idobject2="30" id="133"/>
    <RELATION idobject1="38" idobject2="34" id="134"/>
    <RELATION idobject1="38" idobject2="7" id="135"/>
    <RELATION idobject1="39" idobject2="12" id="136"/>
    <RELATION idobject1="39" idobject2="17" id="137"/>
    <RELATION idobject1="39" idobject2="21" id="138"/>
    <RELATION idobject1="39" idobject2="26" id="139"/>
    <RELATION idobject1="39" idobject2="31" id="140"/>
    <RELATION idobject1="39" idobject2="35" id="141"/>
    <RELATION idobject1="39" idobject2="8" id="142"/>
    <RELATION idobject1="40" idobject2="13" id="143"/>
    <RELATION idobject1="40" idobject2="18" id="144"/>
    <RELATION idobject1="40" idobject2="22" id="145"/>
    <RELATION idobject1="40" idobject2="27" id="146"/>
    <RELATION idobject1="40" idobject2="32" id="147"/>
    <RELATION idobject1="40" idobject2="36" id="148"/>
    <RELATION idobject1="40" idobject2="9" id="149"/>
    <RELATION idobject1="41" idobject2="16" id="150"/>
    <RELATION idobject1="41" idobject2="20" id="151"/>
    <RELATION idobject1="41" idobject2="25" id="152"/>
</RELATIONSET_IROOMDB>
<RELATIONSET_IDMAPPING id="156" object1="IROOM_ID" object2="ADT_ID" objecttype="BUILDING_COMP">
    <RELATION id="157" idobject1="153" idobject2="DD7"/>
    <RELATION id="161" idobject1="158" idobject2="DD8"/>
    <RELATION id="166" idobject1="162" idobject2="DD9"/>
    <RELATION id="171" idobject1="167" idobject2="DDA"/>
    <RELATION id="176" idobject1="172" idobject2="DDB"/>
    <RELATION id="181" idobject1="177" idobject2="DDC"/>
```

```
    <RELATION id="186" idobject1="182" idobject2="DDD"/>
    <RELATION id="191" idobject1="187" idobject2="DDE"/>
    <RELATION id="196" idobject1="192" idobject2="DE1"/>
    <RELATION id="201" idobject1="197" idobject2="DE2"/>
    <RELATION id="206" idobject1="202" idobject2="DE3"/>
    <RELATION id="211" idobject1="207" idobject2="DE4"/>
    <RELATION id="216" idobject1="212" idobject2="DE5"/>
    <RELATION id="221" idobject1="217" idobject2="DE6"/>
    <RELATION id="225" idobject1="222" idobject2="DF8"/>
    <RELATION id="229" idobject1="226" idobject2="DF9"/>
    <RELATION id="231" idobject1="230" idobject2="FF8"/>
    <RELATION id="233" idobject1="232" idobject2="FF9"/>
</RELATIONSET_IDMAPPING>
<RELATIONSET_IDMAPPING id="234" object1="IROOM_ID" object2="MSEXCEL_COSTITEMID" objecttype="COST_ITEM">
    <RELATION id="235" idobject1="43" idobject2="1"/>
    <RELATION id="236" idobject1="44" idobject2="2"/>
    <RELATION id="237" idobject1="45" idobject2="3"/>
    <RELATION id="238" idobject1="46" idobject2="4"/>
    <RELATION id="239" idobject1="47" idobject2="5"/>
    <RELATION id="240" idobject1="48" idobject2="6"/>
    <RELATION id="241" idobject1="49" idobject2="7"/>
    <RELATION id="242" idobject1="50" idobject2="8"/>
    <RELATION id="243" idobject1="51" idobject2="9"/>
    <RELATION id="244" idobject1="52" idobject2="10"/>
    <RELATION id="245" idobject1="53" idobject2="11"/>
    <RELATION id="246" idobject1="54" idobject2="12"/>
    <RELATION id="247" idobject1="55" idobject2="13"/>
    <RELATION id="248" idobject1="56" idobject2="14"/>
    <RELATION id="249" idobject1="57" idobject2="15"/>
    <RELATION id="250" idobject1="58" idobject2="16"/>
    <RELATION id="251" idobject1="59" idobject2="17"/>
    <RELATION id="252" idobject1="60" idobject2="18"/>
    <RELATION id="253" idobject1="61" idobject2="19"/>
    <RELATION id="254" idobject1="62" idobject2="20"/>
    <RELATION id="255" idobject1="63" idobject2="21"/>
    <RELATION id="256" idobject1="64" idobject2="22"/>
    <RELATION id="257" idobject1="65" idobject2="23"/>
    <RELATION id="258" idobject1="66" idobject2="24"/>
    <RELATION id="259" idobject1="67" idobject2="25"/>
    <RELATION id="260" idobject1="68" idobject2="26"/>
    <RELATION id="261" idobject1="69" idobject2="27"/>
    <RELATION id="262" idobject1="70" idobject2="28"/>
    <RELATION id="263" idobject1="71" idobject2="29"/>
    <RELATION id="264" idobject1="72" idobject2="30"/>
</RELATIONSET_IDMAPPING>
<RELATIONSET_IDMAPPING id="265" object1="IROOM_ID" object2="MSPROJECT_ID" objecttype="CONST_ACT">
    <RELATION id="266" idobject1="4" idobject2="1"/>
    <RELATION id="267" idobject1="5" idobject2="2"/>
    <RELATION id="268" idobject1="6" idobject2="3"/>
    <RELATION id="269" idobject1="7" idobject2="4"/>
    <RELATION id="270" idobject1="8" idobject2="5"/>
    <RELATION id="271" idobject1="9" idobject2="6"/>
    <RELATION id="272" idobject1="10" idobject2="7"/>
    <RELATION id="273" idobject1="11" idobject2="8"/>
    <RELATION id="274" idobject1="12" idobject2="9"/>
    <RELATION id="275" idobject1="13" idobject2="10"/>
    <RELATION id="276" idobject1="14" idobject2="11"/>
    <RELATION id="277" idobject1="15" idobject2="12"/>
    <RELATION id="278" idobject1="16" idobject2="13"/>
```

```
    <RELATION id="279" idobject1="17" idobject2="14"/>
    <RELATION id="280" idobject1="18" idobject2="15"/>
    <RELATION id="281" idobject1="19" idobject2="16"/>
    <RELATION id="282" idobject1="20" idobject2="17"/>
    <RELATION id="283" idobject1="21" idobject2="18"/>
    <RELATION id="284" idobject1="22" idobject2="19"/>
    <RELATION id="285" idobject1="23" idobject2="20"/>
    <RELATION id="286" idobject1="24" idobject2="21"/>
    <RELATION id="287" idobject1="25" idobject2="22"/>
    <RELATION id="288" idobject1="26" idobject2="23"/>
    <RELATION id="289" idobject1="27" idobject2="24"/>
    <RELATION id="290" idobject1="28" idobject2="25"/>
    <RELATION id="291" idobject1="29" idobject2="26"/>
    <RELATION id="292" idobject1="30" idobject2="27"/>
    <RELATION id="293" idobject1="31" idobject2="28"/>
    <RELATION id="294" idobject1="32" idobject2="29"/>
    <RELATION id="295" idobject1="33" idobject2="30"/>
    <RELATION id="296" idobject1="34" idobject2="31"/>
    <RELATION id="297" idobject1="35" idobject2="32"/>
    <RELATION id="298" idobject1="36" idobject2="33"/>
</RELATIONSET_IDMAPPING>
<RELATIONSET_IDMAPPING id="299" object1="IROOM_ID" object2="MSEXCEL_RESOURCEID" objecttype="RESOURCE">
    <RELATION id="300" idobject1="38" idobject2="2"/>
    <RELATION id="301" idobject1="39" idobject2="3"/>
    <RELATION id="302" idobject1="40" idobject2="4"/>
    <RELATION id="303" idobject1="41" idobject2="1"/>
</RELATIONSET_IDMAPPING>
<RELATIONSET_IROOMDB object1="BUILDING_COMP" object2="COST_ITEM" id="304">
    <RELATION idobject1="153" idobject2="48" id="305"/>
    <RELATION idobject1="182" idobject2="60" id="306"/>
    <RELATION idobject1="187" idobject2="60" id="307"/>
    <RELATION idobject1="226" idobject2="69" id="308"/>
    <RELATION idobject1="158" idobject2="44" id="309"/>
    <RELATION idobject1="162" idobject2="52" id="310"/>
    <RELATION idobject1="167" idobject2="52" id="311"/>
    <RELATION idobject1="172" idobject2="52" id="312"/>
    <RELATION idobject1="177" idobject2="52" id="313"/>
    <RELATION idobject1="192" idobject2="56" id="314"/>
    <RELATION idobject1="197" idobject2="56" id="315"/>
    <RELATION idobject1="202" idobject2="56" id="316"/>
    <RELATION idobject1="207" idobject2="52" id="317"/>
    <RELATION idobject1="212" idobject2="56" id="318"/>
    <RELATION idobject1="217" idobject2="56" id="319"/>
    <RELATION idobject1="222" idobject2="65" id="320"/>
</RELATIONSET_IROOMDB>
<RELATIONSET_IROOMDB object1="BUILDING_COMP" object2="CONST_ACT" id="321">
    <RELATION idobject1="153" idobject2="10" id="322"/>
    <RELATION idobject1="182" idobject2="23" id="323"/>
    <RELATION idobject1="187" idobject2="23" id="324"/>
    <RELATION idobject1="226" idobject2="33" id="325"/>
    <RELATION idobject1="158" idobject2="6" id="326"/>
    <RELATION idobject1="162" idobject2="19" id="327"/>
    <RELATION idobject1="167" idobject2="19" id="328"/>
    <RELATION idobject1="172" idobject2="19" id="329"/>
    <RELATION idobject1="177" idobject2="19" id="330"/>
    <RELATION idobject1="192" idobject2="15" id="331"/>
    <RELATION idobject1="197" idobject2="15" id="332"/>
    <RELATION idobject1="202" idobject2="15" id="333"/>
    <RELATION idobject1="207" idobject2="19" id="334"/>
```

```
            <RELATION idobject1="212" idobject2="15" id="335"/>
            <RELATION idobject1="217" idobject2="15" id="336"/>
            <RELATION idobject1="222" idobject2="29" id="337"/>
        </RELATIONSET_IROOMDB>
        <RELATIONSET_IROOMDB object1="BUILDING_COMP" object2="RESOURCE" id="338">
            <RELATION idobject1="153" idobject2="38" id="339"/>
            <RELATION idobject1="153" idobject2="39" id="340"/>
            <RELATION idobject1="182" idobject2="39" id="341"/>
            <RELATION idobject1="182" idobject2="40" id="342"/>
            <RELATION idobject1="182" idobject2="41" id="343"/>
            <RELATION idobject1="187" idobject2="39" id="344"/>
            <RELATION idobject1="187" idobject2="40" id="345"/>
            <RELATION idobject1="187" idobject2="41" id="346"/>
            <RELATION idobject1="226" idobject2="38" id="347"/>
            <RELATION idobject1="226" idobject2="39" id="348"/>
            <RELATION idobject1="226" idobject2="40" id="349"/>
            <RELATION idobject1="153" idobject2="40" id="350"/>
            <RELATION idobject1="153" idobject2="41" id="351"/>
            <RELATION idobject1="158" idobject2="38" id="352"/>
            <RELATION idobject1="158" idobject2="39" id="353"/>
            <RELATION idobject1="158" idobject2="40" id="354"/>
            <RELATION idobject1="162" idobject2="39" id="355"/>
            <RELATION idobject1="162" idobject2="40" id="356"/>
            <RELATION idobject1="162" idobject2="41" id="357"/>
            <RELATION idobject1="167" idobject2="39" id="358"/>
            <RELATION idobject1="167" idobject2="40" id="359"/>
            <RELATION idobject1="167" idobject2="41" id="360"/>
            <RELATION idobject1="172" idobject2="39" id="361"/>
            <RELATION idobject1="172" idobject2="40" id="362"/>
            <RELATION idobject1="172" idobject2="41" id="363"/>
            <RELATION idobject1="177" idobject2="39" id="364"/>
            <RELATION idobject1="177" idobject2="40" id="365"/>
            <RELATION idobject1="177" idobject2="41" id="366"/>
            <RELATION idobject1="192" idobject2="39" id="367"/>
            <RELATION idobject1="192" idobject2="40" id="368"/>
            <RELATION idobject1="192" idobject2="41" id="369"/>
            <RELATION idobject1="197" idobject2="39" id="370"/>
            <RELATION idobject1="197" idobject2="40" id="371"/>
            <RELATION idobject1="197" idobject2="41" id="372"/>
            <RELATION idobject1="202" idobject2="39" id="373"/>
            <RELATION idobject1="202" idobject2="40" id="374"/>
            <RELATION idobject1="202" idobject2="41" id="375"/>
            <RELATION idobject1="207" idobject2="39" id="376"/>
            <RELATION idobject1="207" idobject2="40" id="377"/>
            <RELATION idobject1="207" idobject2="41" id="378"/>
            <RELATION idobject1="212" idobject2="39" id="379"/>
            <RELATION idobject1="212" idobject2="40" id="380"/>
            <RELATION idobject1="212" idobject2="41" id="381"/>
            <RELATION idobject1="217" idobject2="39" id="382"/>
            <RELATION idobject1="217" idobject2="40" id="383"/>
            <RELATION idobject1="217" idobject2="41" id="384"/>
            <RELATION idobject1="222" idobject2="38" id="385"/>
            <RELATION idobject1="222" idobject2="39" id="386"/>
            <RELATION idobject1="222" idobject2="40" id="387"/>
        </RELATIONSET_IROOMDB>
    </RELATION_MODEL>
</CIFE_IROOM_PROJECT>
```

B.2 Tables

Test Case: Cost Items, General Overview

ID	Location	Name	Comment	Total Cost
1	**Level 2**			
2	Deck Zone A			
3		Formwork	Means p.93, 1050, 2use	$18,913.00
4		Reinforcement	Means p.106, 0400, 7%	$15,266.00
5		Placing Concrete	Means p.111, 1500, 2use	$3,278.00
6	Deck Zone B			
7		Formwork	Means p.93, 1050, 2use	$11,844.00
8		Reinforcement	Means p.106, 0400, 7%	$9,670.00
9		Placing Concrete	Means p.111, 1500, 2use	$2,063.00
10	Columns Rect. Zone A			
11		Formwork	Means p.93, 6000 & 6500	$2,111.00
12		Reinforcement	Means p.106, 0250, 7%	$683.00
13		Placing Concrete	Means p.111, 0600 & 0800, Material p.517	$561.00
14	Columns Round Zone A			
15		Formwork	Means p.92, 0550	$398.00
16		Reinforcement	Means p.106, 0250, 7%	$327.00
17		Placing Concrete	Means p.111, 0600 & 0800, Material p.517	$274.00
18	Columns Round Zone B			
19		Formwork	Means p.92, 0550	$398.00
20		Reinforcement	Means p.106, 0250, 7%	$131.00
21		Placing Concrete	Means p.111, 0600 & 0800, Material p.517	$109.00
22	**Level 3**			
23	Deck Zone A			
24		Formwork	Means p.93, 1050, 2use	$18,913.00
25		Reinforcement	Means p.106, 0400, 7%	$15,266.00
26		Placing Concrete	Means p.111, 1500, 2use	$3,278.00
27	Deck Zone B			
28		Formwork	Means p.93, 1050, 2use	$11,844.00
29		Reinforcement	Means p.106, 0400, 7%	$9,670.00
30		Placing Concrete	Means p.111, 1500, 2use	$2,063.00

Test Case: Cost Items, Composition

ID	Location	Name	Unit Cost Mat.	Unit Cost Labor	Unit Cost Equipm.	Total Unit Cost	Quantity	Unit	Overall Cost
1	**Level 2**								
2	Deck Zone A								
3		Formwork	$1.44	$2.54	$0.00	$3.98	4752	S.F.	$18,193.00
4		Reinforcement	$590.00	$350.00	$0.00	$940.00	16.24	Ton	$15,266.00
5		Placing Concrete	$67.00	$9.55	$4.58	$71.13	116	C.Y.	$9,411.00
6	Deck Zone B								
7		Formwork	$1.44	$2.54	$0.00	$3.98	2976	S.F.	$11,844.00
8		Reinforcement	$590.00	$350.00	$0.00	$940.00	10.22	Ton	$9,607.00
9		Placing Concrete	$67.00	$9.55	$4.58	$71.13	73	C.Y.	$5,922.00
10	Columns Rect. Zone A								
11		Formwork	$1.77	$4.55	$0.00	$6.32	334	SFCA	$2,111.00
12		Reinforcement	$560.00	$440.00	$0.00	$1000.00	0.73	Ton	$730.00
13		Placing Concrete	$67.00	$16.90	$8.05	$81.95	5.2	C.Y.	$478.00
14	Columns Round Zone A								
15		Formwork	$2.25	$5.70	$0.00	$7.95	50	L.F.	$398.00
16		Reinforcement	$560.00	$440.00	$0.00	$1000.00	0.35	Ton	$350.00
17		Placing Concrete	$67.00	$17.00	$8.15	$82.15	2.53	C.Y.	$233
18	Columns Round Zone B								
19		Formwork	$2.25	$5.70	$0.00	$7.95	20	L.F.	$159.00
20		Reinforcement	$560.00	$440.00	$0.00	$1000.00	0.14	Ton	$140.00
21		Placing Concrete	$67.00	$17.00	$8.15	$82.15	1.01	C.Y.	$93
22	**Level 3**								
23	Deck Zone A								
24		Formwork	$1.44	$2.54	$0.00	$3.98	4752	S.F.	$18,193.00
25		Reinforcement	$590.00	$350.00	$0.00	$940.00	16.24	Ton	$15,266.00
26		Placing Concrete	$67.00	$9.55	$4.58	$71.13	116	C.Y.	$9,411.00
27	Deck Zone B								
28		Formwork	$1.44	$2.54	$0.00	$3.98	2976	S.F.	$11,844.00
29		Reinforcement	$590.00	$350.00	$0.00	$940.00	10.22	Ton	$9,607.00
30		Placing Concrete	$67.00	$9.55	$4.58	$71.13	73	C.Y.	$5,922.00

Test Case: Labor Resources, Used Crews

ID	Name	Comment	Labor Hour Cost	Equipment Hour Cost	Number Crew Members	Hours/Labor Day	Labor Cost/day	Equipment Cost/day
1	C-1	Crew Formwork Columns	$26.27	$0.00	4	8	$840.64	$0.00
2	C-2	Crew Formwork Slabs	$27.50	$0.00	6	8	$1,320.00	$0.00
3	4 Rodm	Crew Rebar	$31.50	$0.00	4	8	$1,008.00	$0.00
4	C-20	Concrete Crew	$23.92	$11.44	11	8	$2,104.96	$91.52

List of Figures

List of Tables

Bibliography

[Fox et al, 2000] Armando Fox, Brad Johanson, Pat Hanrahan, Terry Winograd. Integrating Information Appliances into an Interactive Workspace. IEEE Computer Graphics & Applications, Vol. 20, No. 3, May/June 2000. Avaiable online at http://graphics.stanford.edu/ projects/iwork/papers/ieee-pda00/ ieee-pda00.pdf

[CIFE - CIW] Center for Integrated Facility Engineering. Interactive Information Workspace. Avaiable online at http://www.stanford.edu/ group/4D/workspace/ workspace-main.htm

[Microsoft - Project] Microsoft. Microsoft Office - Microsoft Project Homepage. Avaiable online at http://www.microsoft.com/office/project/default.asp

[DPR - Construction] DPR Construction Company. Webpages. Avaiable online at http://www.dprinc.com/about/index.cfm.

[Autodesk - ADT] Autodesk. Architectural Desktop. Avaiable online at http://usa.autodesk.com/ adsk/section/ 0,,630501-123112,00.html

[Fischer et al, 2000] Fischer, Martin; Liston, Kathleen; Kuntz, John. Requirements and Benefits of Interactive Workspaces in Construction. The 8th International Conference on Computing in Civil and Building Engineering. August, 2000 Stanford University Silicon Valley, CA, USA

[CPT Tech. - 4D-Viewer] Common Point Technologies, Inc. Solutions - 4DViewer. Avaiable online at http://www.commonpointinc.com

[Smart Technologies Inc.] Smart Technologies, Inc. Online at http://www.smarttech.com/products/index.asp

[Stanford CS - EventHeap] Stanford University - Computer Science. Interactive Workspaces - Event Heap Documentation. Avaiable online at http://graphics.stanford.edu/projects/ iwork/software /html-pages/documentation.html

[Johanson et al, 2001] Stanford University.Brad Johanson, Greg Hutchins, Terry Winograd. PointRight: Pointer/Keyboard Redirection for Interactive Workspaces. Avaiable online at http://graphics.stanford.edu/ stone/ steelcase/pointright_ubicomp.PDF

[ARX Developer's Guide, 2000] Autodesk Inc. ObjectArx Developer Guide. 2000. http://usa.autodesk.com

[MCAULEY, 2000] C. McAuley. Programming AutoCAD 2000 Using ObjectArx. Autodesk Press. United States. 2000.

[OMF Developer's Guide, 2002] Object Modeling Framework Developer's Guide. Autodesk. 2002.

[Microsoft - MSDN] Microsoft. Developer Network. Avaiable online at http://msdn.microsoft.com/

[W3C - XML] W3C. Extensible Markup Language. Avaiable online at http://www.w3.org/XML/

[W3C - XML 10Points] W3C. XML in 10 Points. Avaiable online at http://www.w3.org/XML/1999/XML-in-10-points

[Timberline] Timberline Software Cooperation. Avaiable online at http://www.timberline.com

[Microsoft - MSXML] Microsoft. XML Webservices. Avaiable online at http://msdn.microsoft.com/

[C++, 1997] X3 Secretariat. Draft Standard - The C++ Language. X3J16/97-14882. Information Technology Council (NSITC). Washington DC. USA.

[Stroustrup, 2000] Bjarne Stroustrup. The C++ Programming Language - Special Edition. Addison-Wesley. Florham Park, New Jersey. 2000.

[RSMeans, 2000] RSMeans. Building Construction Cost Data. 58th Annual Edition. RSMeans. Kingston, MA. 2000.

[SFIRION] SFIRION Projektleitsysteme. Avaiable online at http://www.sfirion.de/

[Graphisoft] Graphisoft Deutschland GmbH. Avaiable online at http://www.graphisoft.com/