CIFE CENTER FOR INTEGRATED FACILITY ENGINEERING

CIFE iRoom XT Design & Use

By

Marcus Schreyer, Timo Hartmann, Martin Fischer,
and John Kunz

CIFE Technical Report #144
December 2002

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

The authors would like to acknowledge the contributions
of the following persons to this research:

Peter Allen (DPR Inc.)
Frank Neuberg (TU Munich)
Ernst Rank (TU Munich)
Dean Reed (DPR Inc.)
Vibha Singhal

CIFE iRoom XT Design and Use

Technical Report #144

Title:	CIFE iRoom XT Design and Use
Authors:	Marcus Schreyer, Visiting Scholar, CIFE, Stanford University Timo Hartmann, Visiting Scholar, CIFE, Stanford University Martian Fischer and John Kunz, CIFE
Publication Date:	December 2002
Funding Sources:	German Research Society (Deutsche Forschungsgesellschaft – DFG) Research Scholarship
Keywords:	Interactive Workspaces, Concurrent Interoperability, Project Meetings, PPO Modeling, Virtual Design and Construction
Document Source:	http://cife.stanford.edu/online.publications/TR144.pdf

Abstract:

During the design and planning process of civil engineering projects a variety of different computer applications are used. Each of these programs models a subset of the overall project's context. For the interdisciplinary tasks discussed in project meetings it is important to identify the existing interrelations between these sub-models of the different applications. The hard- and software for interactive workspaces can be used to link the different application models by modeling the mutual relationships between shared data and can therefore support the decision process through cross-application functionalities. In order to achieve this, the workspace has to be able to distribute data between the connected applications.

This report introduces the iRoom XT framework that we developed to be used within the iRoom, an interactive workspace at the Center for Integrated Facility Engineering (CIFE). The introduced framework consists of a general, extensible common data model for the central storage of the distributed project data. The data model provides flexibility towards the different scopes and detail levels which may occur within project meetings. Further the iRoom XT framework consists of a messaging system for distributing project data.

While the first part of the report generally describes the ideas behind the framework, the second part of this document consists of manuals. The first of these manuals describes the modeling of projects using the iRoom XT framework. Two other manuals are intended to be a starting point for programmers who wish to enrich the iRoom environments with further functionality.

Content

ABOUT THIS DOCUMENTATION

The CIFE iRoom XT as documented in this report has been developed during Fall 2002 upon the result of earlier research [1, 3] at CIFE and the CS department.

The report is structured into two parts and focuses mainly on the new respectively renewed components of the CIFE iRoom. In the first part consisting of the chapters “Motivation” and “Overview” the reader should get a general picture of the iRoom that will enable him to understand the methods used to implement new functionalities for interactive workspaces like the iRoom. Therefore it is intended to be a comprehensive conceptual and technical introduction for a user who is new to the CIFE iRoom environment.

In the second part starting with chapter “Manual” the documentation focuses on the description of the usage and the possibilities to extend the current system. Thus it addresses users that want to virtually model projects which can be used in the iRoom environment as well as users who intend to extend the current project data structure respectively the already implemented workspace functionalities.

MOTIVATION OF OUR RESEARCH

The goal of our research was to develop a general, extensible framework for building project information that allows the implementation of interactive functionalities across various, discipline specific AEC applications.

Our approach is based on an already existing iRoom environment that evolved through a series of case studies and research projects at CIFE [3, 5]. These case studies showed possibilities and benefits of interactive workspaces using “real world” applications like for example CPT 4D CAD or MS Project to model the design and construction data of a project. However, the evolution of the underlying iRoom data structures that are the “glue” to integrate the different applications into an interactive planning environment did not happen from a comprehensive perspective. Thus the former iRoom data model rather represented an aggregation of data objects contributed by the various applications connected to the iRoom.

Consequently, this led to redundancies and inconsistencies in respect to the data objects in the data schema. As we tried to integrate 3D product model data out of AutoDesk’s Architectural Desktop (ADT) and Cost and Resource information through Excel worksheets in the iRoom data model, we realized that data, which was structured according to the existing data schema, was difficult to extend.

Another problem has been the lacking of a central and easy to use GUI to start the basic interaction services (eHeap) provided by the iRoom. Especially in situations like during a presentation, we have assumed that such a graphical interface could be very helpful. Further it could be a central point to access and monitor the settings of the iRoom.

We are hoping that the thoroughly remodeled framework of the CIFE iRoom XT will allow other developers an easier and faster implementation of sophisticated extensions (XTensible) since it provides modularized software architecture with reusable class libraries.

OVERVIEW

This chapter is intended for readers who are new to the technology of the CIFE iRoom. It describes the functionalities of the environment with regards to implementation and data modeling issues. If you want to know more about general purposes of the iRoom and its benefits for AEC stakeholders, you should read the papers [3, 5].

This chapter starts with a high level abstraction of the underlying project model that we developed to be used with the iRoom environment. Thus the meaning of the iRoom related terms will be defined here. Later on in this chapter the actual software architecture will be explained in detail, e.g. how the different modules work and how they are structured internally.

iRoom Project Framework

In order to achieve interactivity, two prerequisites have to be fulfilled. First there has to be a common data model for the distributed project data. This model should provide flexibility towards different project scopes that are defined by the data elements contributed by the various applications. Such a general project data model should allow identifying and addressing specific data objects in the iRoom environment. Second a general messaging framework has to be developed on top of the project data model in order to provide cross application functionalities. Ideally this messaging framework should be designed in a way, that it is flexible towards later functional extensions.

In order to develop such a general project data model, we had to overcome the following problems first:

- The project data is distributed in different applications. These applications, used in the miscellaneous disciplines, have separate data models with their individual data structures.
- Data elements can appear in more than one application. Regarding their identification this means that the solution had to be flexible towards redundancy.
- In the data structures of the different applications, these semantically identical data elements are named differently and broken up into different hierarchies or levels of detail.

The basic idea of our solution in order to solve these problems is shown in figure 1. We map the data elements, stored in the data structures of the various applications to a central project data model. Further, we model the relationships necessary to describe this mapping separately. Until now, we have defined two types of relationships: First, the relationships between data objects in the applications and the central iRoom project data model and second the relationships between objects of different types within the project data model itself. It is possible to implement further relationship types in the future, e.g. to represent specific semantic relationships like object hierarchies or priorities to modify values.

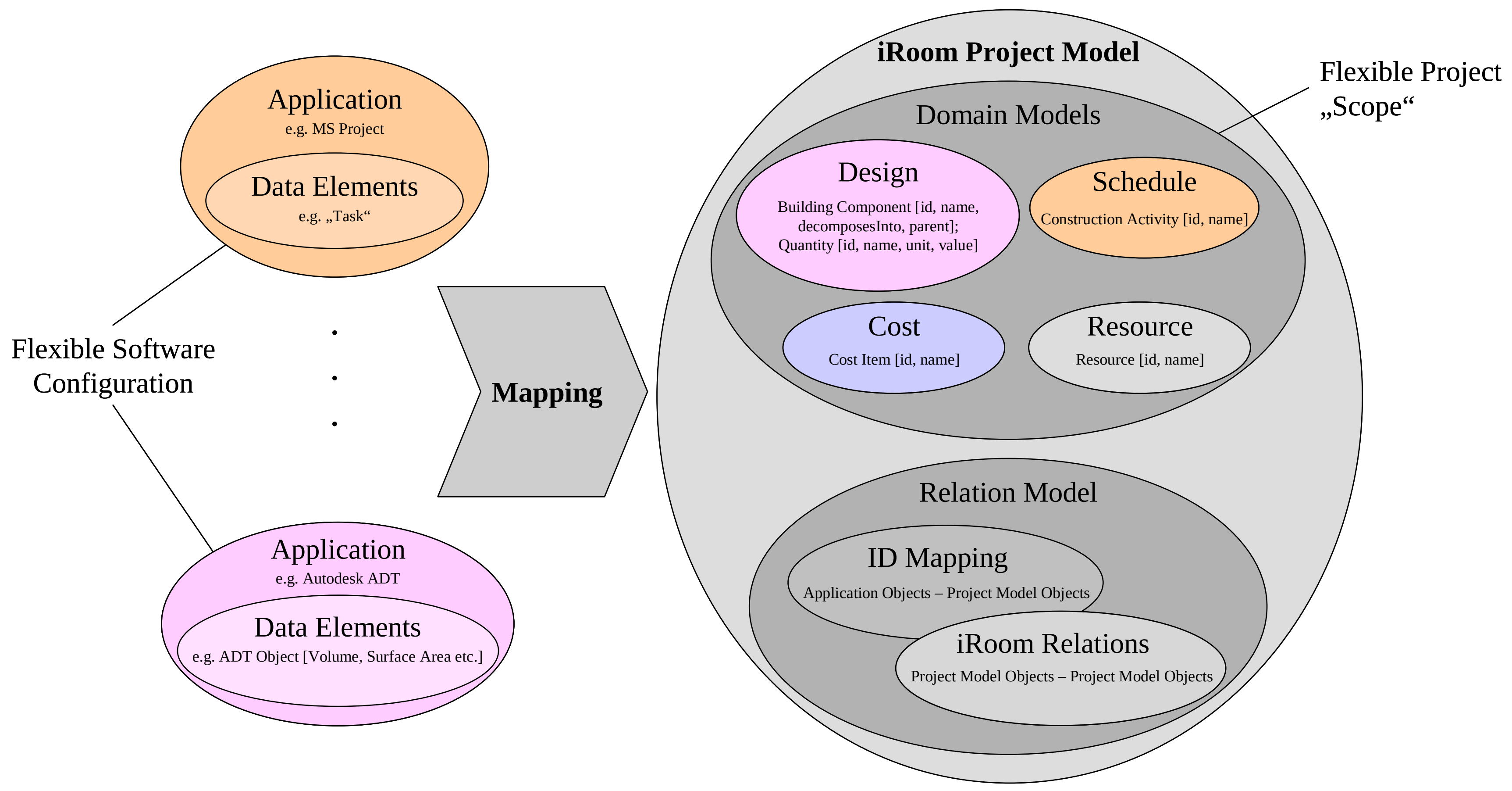

Application
e.g. MS Project
Data Elements
e.g. „Task“
Flexible Software Configuration
Application
e.g. Autodesk ADT
Data Elements
e.g. ADT Object [Volume, Surface Area etc.]
Mapping
iRoom Project Model
Flexible Project „Scope“
Domain Models
Design
Building Component [id, name, decomposesInto, parent]; Quantity [id, name, unit, value]
Schedule
Construction Activity [id, name]
Cost
Cost Item [id, name]
Resource
Resource [id, name]
Relation Model
ID Mapping
Application Objects – Project Model Objects
iRoom Relations
Project Model Objects – Project Model Objects

Figure 1 The basic idea of our approach to develop a data model for the iRoom is to map the data elements, distributed in the various applications, to a central project data model. This provides flexibility towards different project scopes, defined by the applications and the data they provide to the miscellaneous project domains.

The mapping to a central project model taxonomy also provides a way to address the otherwise redundant data elements unambiguously. The programmer of cross application functions can solely focus on the objects of the project model. If the data elements in the applications have been mapped to their correspondent iRoom objects correctly, the modifications to iRoom objects will be reflected on their related data elements as well.

The amount of data that has to be represented in the central project data repository depends on the interactive functionality that should be provided. Until now, we have implemented a generalized highlighting functionality for any arbitrary model scope within the iRoom environment. We have discovered that for the highlighting functionality it is sufficient to store the unique data ID, assigned to the object within the respective applications, and an object name within the iRoom data repository. Our approach therefore led to a rather small project data repository. In chapter 3.4.3 we'll show by example, how this approach to store only necessary information in the central repository, gives the project data model flexibility regarding the extension of the iRoom environment.

Using this general data model we have developed a messaging framework that allowed us to provide interactive functionality across various applications connected to the environment. The following chapters 3.2 and 3.3 explain the software architecture and message format that had to be developed as a result of the framework shown in figure 1.

CIFE iRoom XT Architectural Schema

To design a well structured and therefore easier to extend system upon the domain model described above, the development task has been broken up into three layers separated by functionality:

- Application layer
- Communication layer
- Data storage layer

In this 3-tier architecture, the different AEC applications offer multiple GUIs to access the project data, while a Middleserver application controls the communication functionality that directs the messages between the different applications. Further the data that has to be shared among AEC applications is stored in a XML data file in order to enable the applications to communicate with another. Perhaps the best way to explain the interaction and tasks of the individual system components is by example (figure 2):

For a particular building component selected in ADT an iRoom specific extension of the ADT software called Heap Interface or Heapi generates a text message and pushes this message onto the Event Heap (eHeap).

The EventHeap [Stanford CS - EventHeap] is a central software component located on one of the iRoom computers. Its purpose is to collect and store all messages for a predefined period of time. Any program on one of the IRoom devices can access these messages on the event heap using a so-called event heap "listener" application. This "listener" observes the event heap constantly and builds together with the Event Heap a communication infrastructure for miscellaneous message types. If the listener of an application is restricted to a specific message type, it is possible to address messages for particular applications.

Messages from applications are addressed to the Midserver and contain information to identify the selected building component in the ADT internal data model. Because the Event Heap is a largely passive system component and unable to distribute messages by sending them to the connected applications, the listener has to check the eHeap regularly for new messages addressed to the specific application it extends.

Like the other applications, the Midserver is connected to the iRoom with a listener, too. The Midserver listener checks the eHeap for messages addressed to him and picks up the ADT message of our example to analyze and process its information. In the Midserver, the message is split up in its components. The first part of the message that is examined contains information about the action type (see chapter 3.3) that should be executed. In the case of our example the action type is "highlight (hl)". Having information about the sending application and the ID of the selected element, the Midserver can then begin a query dialog with the iRoom data repository to find out, which data objects in other applications are related to the building component selected in ADT. Microsoft's XML Parser has been integrated to enable the Midserver to parse the XML data file according to the DOM standard [6]. A detailed description of the dialog between the Midserver and the iRoom data model can be found in chapter 3.4.2.

The result of this query dialog is generally a set of application IDs, object types and object IDs that can be reassembled to create another set of messages in the Heapi of the Midserver. When the Midserver Heapi puts these messages on the eHeap, they can again be picked up and interpreted by the listeners of the corresponding AEC applications and produce a reaction according to the algorithms defined by the action type.

Besides the basic Midserver functionality and the messaging infrastructure to establish interactivity between the separate applications, there are further components of the CIFE iRoom XT that can assist the user during the creation of the iRoom project data file. While there is already a Relation Tool that helps the user to create the relationships between iRoom objects in the XML data file (see 3.4.4), the data export into the iRoom isn't yet possible from all of the AEC applications. So far, this functionality is only provided for ADT [2]. If it should be able to use the iRoom for larger projects in the future, such data export components will have to be taken into consideration in order to allow a fast creation of an accordingly large project data file.

Messaging

In order to establish the Midserver as a central controlling instance for the communication between the applications, the message format had to be standardized throughout the iRoom. Hence it is possible to extend the algorithms that analyze the content of the messages, at a central point in the system. Since we assume that the functionality and complexity of the iRoom will evolve over time, the possibility to extend or change the system easily will become more critical. One of the main advantages of the CIFE iRoom XT is the extensibility of its interactive functionalities that has been achieved through the changes to the message format.

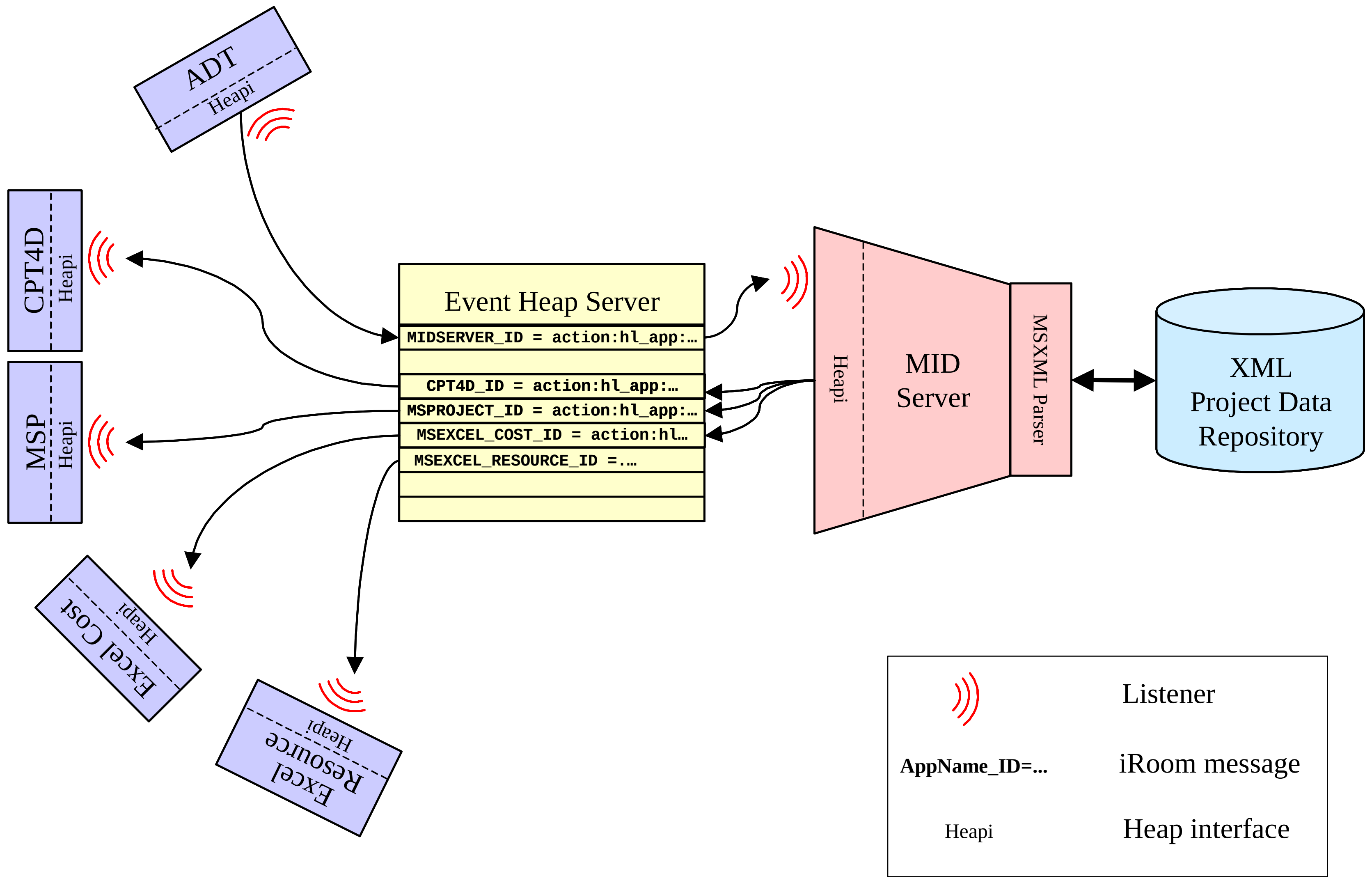


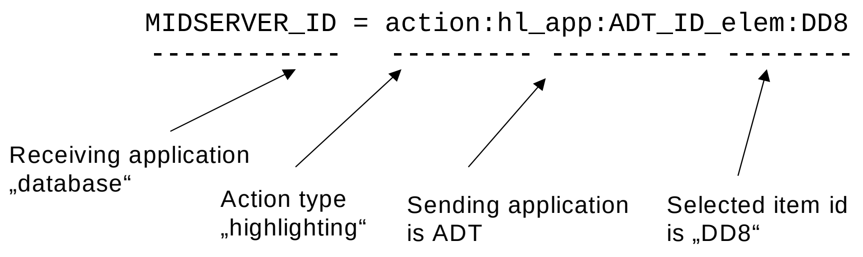
Figure 2 Graphical schema of the CIFE iRoom XT communication and software components

While the original CIFE iRoom used eHeap messages with a syntax like "ca = Pouring concrete Slab A3, Pouring concrete Slab A4" containing information about the object type, in this case *Construction Activity*, and their corresponding names in the applications "Pouring concrete Slab A3, Pouring concrete Slab A4", there was a need for an extensible concept which offers the possibility for a more complex but also more flexible message format.

In order to achieve interactivity across the applications the syntax of the messages to the more complex domain model shown in chapter 2, at least the following information will have to be represented in the messages:

- Sender Application ID
- Receiving Application ID
- Event type
- Object ID

Using these parameters, the following format for messages in the CIFE iRoom XT has been derived:



```
MIDSERVER_ID = action:hl_app:ADT_ID_elem:DD8
```

Receiving application „database"
Action type „highlighting"
Sending application is ADT
Selected item id is „DD8"

The bundling of the communication makes the Midserver to the central receiver and sender of messages. However, in respect to the message format, the Midserver is treated within the communication environment the same way as all the other applications and therefore identified through an application ID. Values for these message parameters, which already have been defined at the time of the publication of this documentation, are shown in Table 1.

Table 1 Defined values for the message components created by the applications' Heap is

Message Component	**Values**	**Comment**
Receiving Application	MIDSERVER_ID, ADT_ID, MSEXCEL_COST_ID, MSEXCEL_RESOURCE_ID, MSPROJECT_ID, CPT_ID	COST and RESOURCE address the two sheets used in MS Excel for the according object types
Action Type	hl	Only the highlight event has been implemented, yet.
Sending Application	MIDSERVER_ID, ADT_ID, MSEXCEL_COST_ID,	See above

	MSEXCEL_RESOURCE_ID, MSPROJECT_ID, CPT_ID	
Selected item	Not predefined	Here the Heapi puts in the element ID of the selected or modified data element

Components

The following chapters describe the functionalities and structure of the different components that had to be built or adapted for the CIFE iRoom XT in detail. Although the descriptions don't contain or explain the complete source code, they should ideally enable a developer to extend the functionality of the iRoom by helping to identify the respective sections in the system. A description of what exactly has to be done to extend the different components of the CIFE iRoom XT will be given later in chapter 4.

Application Interfaces "Listener" and "Heapi"

In order to connect an application with the CIFE iRoom, two functionalities have to be provided for each application: First, the listener has to be connected to the *receiving application* by a software interface that receives the messages and executes the respective commands in the application. Second, another algorithm has to be implemented in order to generate and send messages to the Midserver respectively to the iRoom. This functionality can be implemented either in the same software interface as for the listener connection or in a separate interface.

The answer to the question, which programming tools should be chosen to create these functionalities depends largely on the APIs provided by the particular application. While Microsoft's MS Project and MS Excel offer a popular API to access their internal data structure and their GUI elements via the predefined objects in Visual Basic (VB), the ADT can be accessed using its API for C++. For other applications like e.g., CPT's 4D CAD, Timberline's Precision Estimating or SimVision that unfortunately don't provide a publicly accessible API, it is necessary to contact the developers and get access to the source code in order to find out how to execute the iRoom messages on the application internal data structure respectively in the GUI.

Using the Visual Basic for Application (VBA) API, the software interfaces to connect e.g. MS Excel to the iRoom consist mainly of the following components:

- Java -listener for the iRoom
- Heapi - .EXE file
- VBA macro file

The Java-listener is a component that already existed in the original CIFE iRoom. In order to work with the new message format, the Listener had to be adapted for each specific application to retrieve only messages with the respective new message prefix. As the message queue on the eHeap has to be checked regularly, the Listener has to run in a separate thread than the actual application. In this way the Listener is not blocking the application’s general functionality. In order to access the within the VB Heapi implemented functionality, the Java-listener calls a compiled VB Heapi - .EXE file and hands over the message string as a file attribute.

Within the Heapi, the message string is analyzed for the action type that should be executed. In the case of MS Excel, it is further necessary to analyze to which worksheet the message addresses, since in our test cases Excel was used to manage a worksheet with the cost table and another with a table for the resources. The following step is then to extract the application internal ID(s) and to call the functions like highlighting etc. to be displayed in the application’s GUI.

If the respective application should be enabled to send messages, there has to be another algorithm that generates these CIFE iRoom XT messages. For the two MS Office applications for example a VBA macro is provided that creates a highlighting message (see also chapter 3.3) addressed to the Midserver and containing the ID of an object selected by the user within the respective Office application (see also chapter 4.1).

Another possibility to connect an application to the iRoom is to use a C++ API. In the case of ADT, the functionality to create highlighting messages for the Midserver has been written using C++ dynamic link library files (.dll files) [2]. These .dll files have to be loaded at the start of the respective application in order to enable the desired Heapi functionality.

Midserver

The Midserver has to fulfill three main tasks in the iRoom environment:

1. *Receive incoming messages* from the applications,
2. P*rocess the messages* and identify the relations according to the underlying XML project data file and
3. *Generate the outgoing messages* to the respective applications.

Furthermore, the Midserver GUI offers menu entries to select a specific XML project data file and to choose the eHeap server to which the applications should listen.

In order to understand how the Midserver and therefore the communication in the CIFE iRoom XT works, it is necessary to understand the parsing process that has to be executed every time an incoming message is processed. This parsing mechanism also shows, how flexible the chosen solution is towards different scopes in iRoom sessions, towards different data contents and levels of detail in the XML project data files.

Whenever a message from an application to the Midserver is retrieved by its listener, as it is described in chapter 3.3, the message string has to be analyzed. The first information that the Midserver extracts from the string is the action type it has to execute. This parameter defines the algorithms that will be used for the respective project element. Until now, we have implemented only one action type: the highlighting.

Furthermore, information to identify the selected elements in the application respectively their corresponding objects in the iRoom project data file has to be extracted from the message. The Midserver therefore uses the “sending application” parameter and the “element id” from the message string to parse the XML file for the corresponding iRoom objects. The required relationships between these data elements are modeled as RELATION elements in the RELATIONSET_IDMAPPING XML elements. Taking the message example from 3.3 with ADT as the sending application, the steps to parse the data file are as follows:

First, the Midserver has to identify all RELATIONEST _IDMAPPING elements in the iRoom data file, which map objects of type ADT_ID, to their corresponding objects in the iRoom framework. A code example for such an XML element is given below. In this example ADT objects (ADT_ID) are assigned to iRoom objects (IROOM_ID) of type BUILDING_COMP. While the RELATIONSET element describes the relation type, the specific assignments between actual data elements are defined by RELATION elements. In the data file schema, RELATION elements are child elements of the complex RELATIONSETs.

Then the RELATIONs in the respective RELATIONSET elements have to be parsed for objects with the ADT element name “DD8”. In the code example below, the element name “DD8” can be found in the second RELATION element, where it is assigned to the iRoom object with the IROOM_ID “158”.

```
<RELATIONSET_IDMAPPING id="156" object1="IROOM_ID"
    object2="ADT_ID" objecttype="BUILDING_COMP">
    <RELATION id="157" object1="153" object2="DD7" />
    <RELATION id="161" object1="158" object2="DD8" />
    <RELATION id="166" object1="162" object2="DD9" />
    <RELATION id="171" object1="167" object2="DDA" />
    <RELATION id="176" object1="172" object2="DDB" />
    <RELATION id="181" object1="177" object2="DDC" />

  . . .

</RELATIONSET_IDMAPPING>
```

Summarizing the parsing of the RELATIONSET element above, the Midserver retrieves the following information from the project data file:

- ADT contributes data elements of the type *building components* (BUILDING_COMP) to the project model
- The *application ID* “DD8” corresponds to the XML element in the data file with the *IROOM_ID* “158”.

Since the actual application ID and the iRoom object type are merely predefined values of XML attributes, the iRoom domain framework can easily be extended with other applications or objects without having to change the parsing algorithms.

In the second step of the parsing algorithm all objects of other types within the XML data repository that are related to the selected one have to be found. The relations between objects of the iRoom are described by another XML element type with the name *RELATIONSET_IROOMDB*. These XML elements are further queried for relationsets with objects of the type *BUILDING_COMP*. Since objects of the type *building components* are normally related to a number of different object types, e.g. to *construction activities* and also to *cost items* etc., this query generally returns a couple of relationsets. To keep our example simple, we assume that the parser finds just one relationset with *building components* and *construction activities*. In this case, it retrieves a relation between the *building component* 158 and the *construction activity* 6.

```
<RELATIONSET_IROOMDB id="321" object1="BUILDING_COMP"
    object2="CONST_ACT">
    <RELATION id="325"  idobject1="226" idobject2="33"/>
    <RELATION id="326"  idobject1="158" idobject2="6"/>
    <RELATION id="327" idobject1="162" idobject2="19" />

  . . .

<RELATIONSET_IROOMDB>
```

After the parser has identified the type and ID of all iRoom objects that are related to the data element specified in the message, the Midserver has to map these internal ID's to their correspondent elements in the respective applications. Once more, the RELATIONSET_ IDMAPPING elements have to be parsed to achieve this. In the code example shown below, the iRoom object *construction activity* "6" is assigned to the corresponding application "MSProject_ID" and further to the MS Project internal ID "9".

```
<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" />

  . . .

</RELATIONSET_IDMAPPING>
```

With this information the Midserver can create outgoing messages to the different applications. We decided to bundle elements for the same application within one message. If more than one element in an application is addressed, the parser is programmed to row the IDs of these elements in the message separated by commas in the form of:

```
MS_Project_ID = action:hl_app:MIDSERVER_ID_elem:6,7,8
```

After the Midserver has sent the messages to the eHeap, they can be picked up by the listeners and finally be processed by the respective application's Heapi.

Project Data Repository

The project data repository is a central XML data file in the CIFE iRoom environment that stores the required data in order to allow interactivity between the different applications. It contains information about the shared data objects of the involved applications and the relations between these objects. This information enables the Midserver to provide interactive functionalities.

Compared to the former data structure of the iRoom that was used during the Bay Street case study, the new central data repository provides advantages concerning the extensibility and flexibility of the iRoom environment. The main characteristics of the thoroughly revised database are

- object oriented structure,
- organization in domains,
- consistent data structure,
- compatibility to former iRoom objects.

The repository contains object-oriented structures that represent the corresponding data elements in the applications. On the “_MODEL” level in the data structure (figure 3), the data is organized in different project domains named after the disciplines Design, Schedule etc. Regarding the objects that had to be defined in the project domains, the objects of the former data model used for the Bay Street case study were reused when possible. The former model was further extended by new objects for design, cost and resource data. On the object level, the new data model is now redundancy free. Therefore a specific type of XML element can only occur at a specific position in the structure of the project data file. This allows a much easier maintenance of the project data repository itself and also of the iRoom software components described in this chapter.

In the data model shown in figure 3, objects are defined “optional” by default. This provides flexibility to customize the iRoom data to the current scope. As mentioned above, the object structure can be extended by other objects, e.g., Documents, Specs, Materials or Participants. Of course these new objects will have to be related to respective data elements in the applications using RELATIONSET_IDMAPPING elements in order to enable the Midserver to access them. Further, we have created a DTD file with the name iRoomXT.dtd. This file defines the attributes of the data objects and their occurrence and can be used by the parser to provide an automated validation check of the project data file before it is used with the Midserver. Future extensions of the iRoom data model will have to be reflected in this DTD, too.

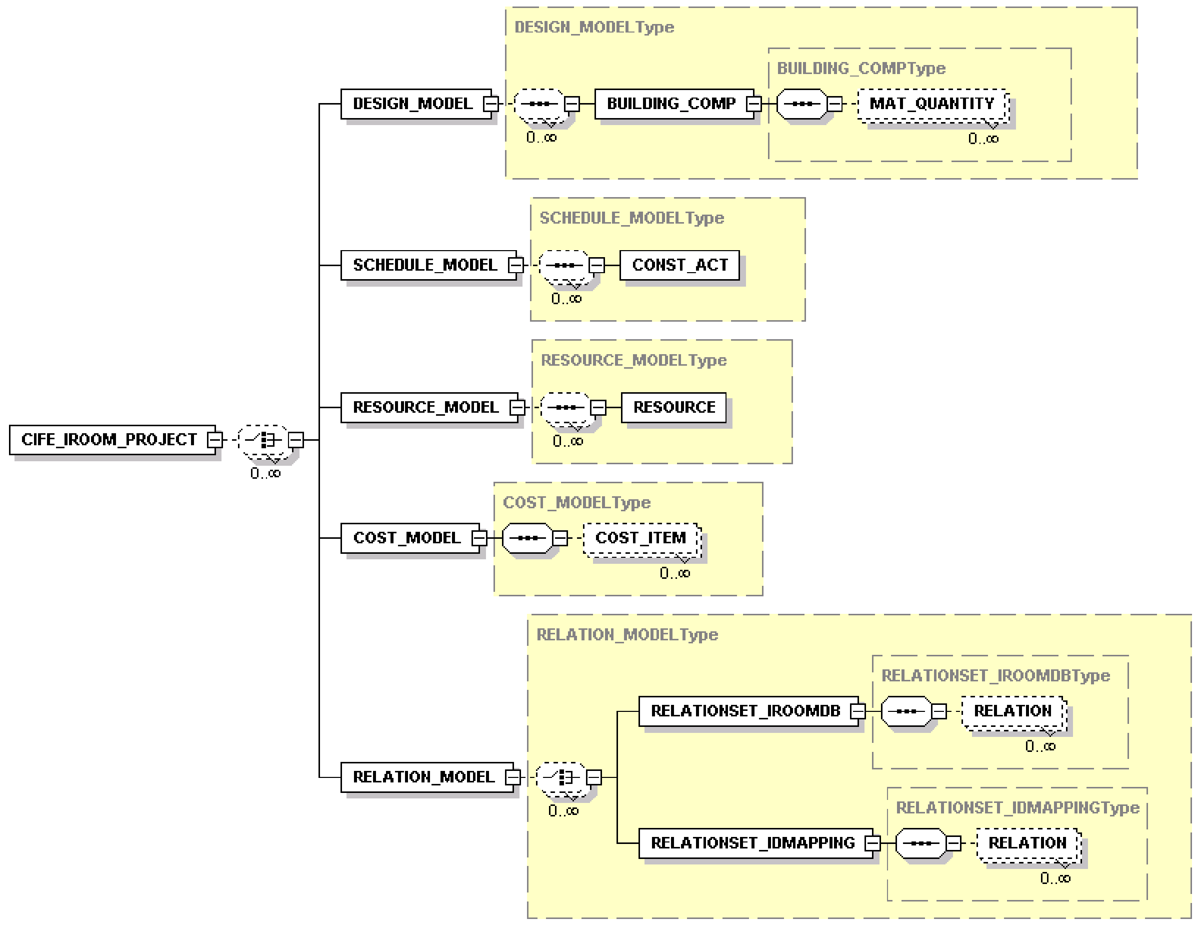


Figure 3 Structure of the XML elements in the CIFE iRoom XT data repository

The biggest difference between the former and the actual data model is the explicit modeling of relations between data objects. This allows not only an easier maintenance of the modeled relations between data elements, but also offers an easy possibility to extend the relation model with other relation types. Currently two *relation-types* are used in the data file: One to model the relationships between data elements within the applications and their corresponding objects of the iRoom framework in the central XML data repository and another one to model the internal relationships between the objects within the data repository of the iRoom.

Relationships between data elements in the applications and their correspondent iRoom repository objects are generally of the type n:1, since generally the shared iRoom representation of the data is not more detailed than its original data model. Therefore, an object in the iRoom data repository represents one or more data elements in an application, dependent on the required level of detail for the XML data used within iRoom meetings. The XML element used to model such relationships is *RELATIONSET_IDMAPPING*. In this element, the value “IROOM_ID” for the attribute *object1* is fixed in order to accelerate the parsing process described in chapter 3.4.2.

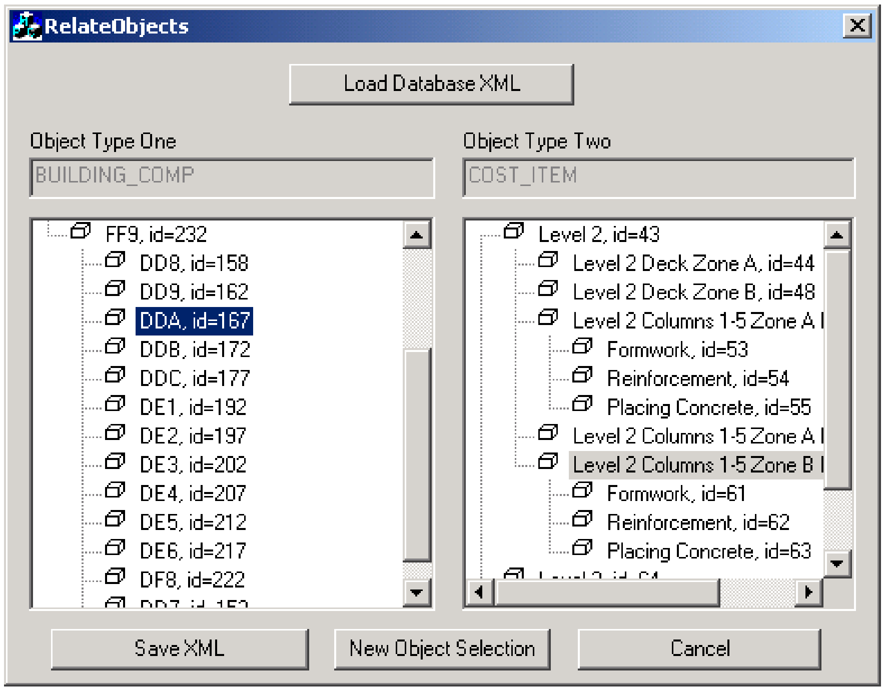
Relationships between the objects in the iRoom data repository are generally of type n:m. E.g. a *building component* can be related to many *construction activities* as well as the other way round. These iRoom internal relationships are modeled in the element *RELATIONSET_IROOMDB*.

Relation Tool

It is not an easy task to generate the relationships between objects in the central iRoom data file as described above. For large projects, the number of different objects that have to be linked can easily reach more than one thousand. Thus the implementation of a software component with an adequate user interface is extremely important in order to be able to use the iRoom efficiently.

Our approach to develop a relation tool with a GUI that allows the relation of objects of two different objects types is shown in Figure 4. The screenshot shows a dialog window with two tree view controls that have been implemented using the MFC library. MFC tree view controls allow the representation of hierarchies between objects, a feature that makes the navigation and especially the relation of object groups much easier than using plain object lists without any hierarchical structures.

The objects are represented in the tree views by their element name and their unique ID. Both attributes are obtained from the XML data elements in the iRoom data file.



Object Type One: BUILDING_COMP

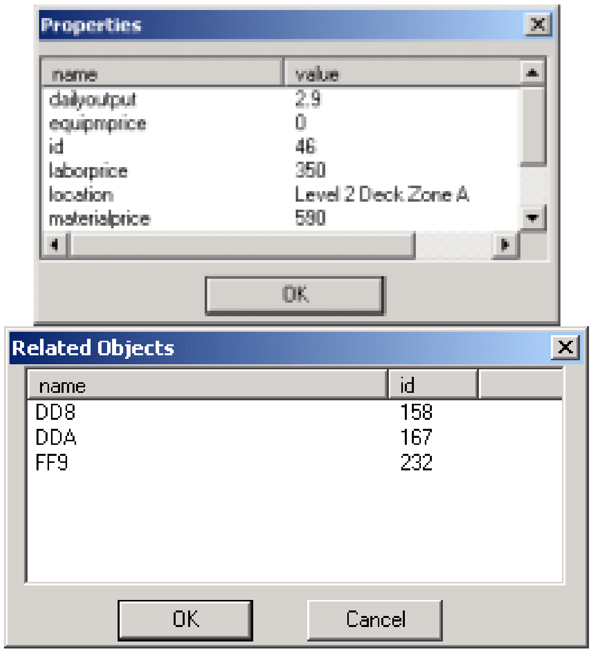
Object Type Two: COST_ITEM

```
FF9, id=232
    DD8, id=158
    DD9, id=162
    DDA, id=167
    DDB, id=172
    DDC, id=177
    DE1, id=192
    DE2, id=197
    DE3, id=202
    DE4, id=207
    DE5, id=212
    DE6, id=217
    DF8, id=222
```

```
Level 2, id=43
    Level 2 Deck Zone A, id=44
    Level 2 Deck Zone B, id=48
    Level 2 Columns 1-5 Zone A
        Formwork, id=53
        Reinforcement, id=54
        Placing Concrete, id=55
    Level 2 Columns 1-5 Zone A
    Level 2 Columns 1-5 Zone B
        Formwork, id=61
        Reinforcement, id=62
        Placing Concrete, id=63
```

Figure 4 Dialog to Relate XML Data File Objects

The MFC further allowed the implementation of right click context menus for the objects in the tree controls. Thus the user can access more detailed information about the objects than just their name. E.g. using the “Properties” entry within the context menu pops up a window showing all attributes of the iRoom object (Figure 5 left). A second menu entry “Related objects” displays the objects that have already been related to the selected one (Figure 5 right).



name	value
dailyoutput	2.9
equipmprice	0
id	46
laborprice	350
location	Level 2 Deck Zone A
materialprice	590

name	id
DD8	158
DDA	167
FF9	232

Figure 5   Dialog windows showing additional object information

Using the Relation Tool, the first step is to load the respective XML data file into the application using the "Load XML data file” button that invokes a standard Microsoft Windows file select dialog. Further, the object types to relate can then be entered in the two input fields above the tree controls. The relations between two objects of the trees can now be established by simply dragging one object of one tree over the correspondent object of the other tree while holding down the left mouse button. The relation will be created automatically by the relation tool by releasing the left mouse button.
After the relations between two object types have been created, they can be saved within the respective iRoom file by clicking the "Save XML” button in the dialog window. New object types can be selected by pressing the "New Object Selection” button without having to start the tool again. The "Cancel” button can be used to exit the dialog without saving the created or altered relations to the data file. Like the Midserver, the Relation Tool uses Microsoft’s XML Parser to access the XML file and to parse the data.

The Relation Tool itself has been programmed as a stand-alone application that can be used to efficiently relate all object types of the iRoom data repository. This version of the Relation Tool can be started by executing RelateObjects.exe. There is also a second version that has been integrated in AutoDesk’s ADT [2]. The Relation Tool version implemented in the ADT is restricted to the *Building Components* for *Object Type One*. It nevertheless offers more features than the version for general relations described above. Since the Relation Tool is running within the ADT, a highlighting functionality for the structural elements and zones that are selected in the tree control could be implemented by using the OMF API of ADT (Figure 6). Hence the user receives information about the position of the specific building component in the geometrical model in a comfortable way.

In order to start the ADT Relation Tool, the .dll-file extension “relateInADT.arx” has to be uploaded into the ADT. The Relation Tool itself can be started with the ADT command line command “relateObj” that invokes a dialog similar to the one illustrated in Figure 4.

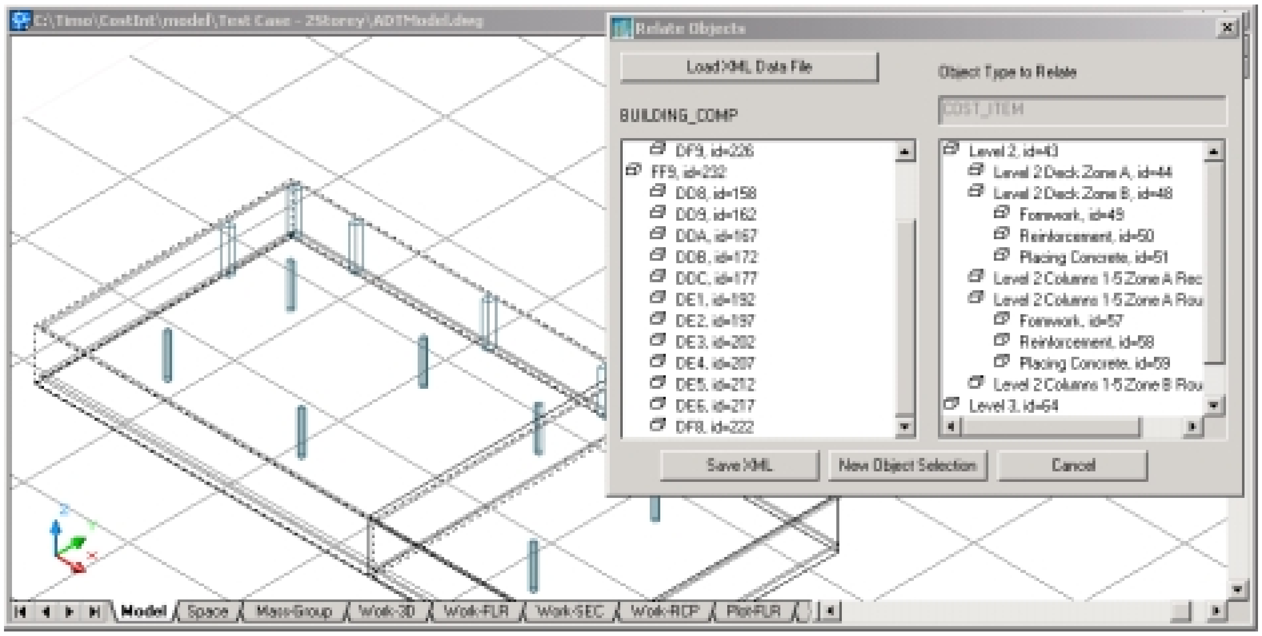


Figure 6   Highlighting of structural elements in ADT while working with the Relation Tool

MANUAL

This manual section mainly is intended to describe the integration of new applications into the already existing iRoomXT framework using examples in “How-To” form. In order to be able to integrate new applications two steps have to be accomplished:

1. Adjust the Central XML Data Schema
2. Connecting the Application to the respective event heap

These two steps will be demonstrated in this chapter using three different example scenarios. All these scenarios are explained at length, documenting all the steps that have to be accomplished in order to get the respective example running.

Before starting with the examples be sure that the Event Heap client software is installed on your machine. If not, install it using the documentation provided on http://www.stanford.edu/group/4D /workspace/documentation/document-frames.htm. Problems mainly occur due to erroneous or not defined environment variables, so ensure that the “classPath” environment variable of your system is set to the directory including the “eheap.jar” java class library. Further ensure that the “Path” variable is pointing to the directory of the file “jvm.lib” of your local Java installation.

The first scenario explains how to extend the existing data schema and create a new XML data file. Further we will set up a simple iRoom scenario in this example, in which two applications communicate using the iRoom Event Heap. In this example we use some of the already implemented tools from the CD accompanying this documentation.

Further two implementation examples are described, which use the EHeap API to access an Event Heap server. The Event Heap server used in the examples is called “cife” and is running on the host machine “cife-32.Stanford.edu” at the Center for Integrated Facility Engineering. If you intend to use another server just replace the machine name and the server name in the following sections. The examples shown can be used as a starting point for the programming of connections to Event Heaps. Nevertheless users who plan to create new applications that work with Event Heaps should also refer to the official documentation of the Computer Science Department at Stanford: http://graphics.stanford.edu/projects/iwork/software/htmlpages/documentation.html.

Adapting the XML Data Schema

In this example we at first extend the existing XML schema with a new object type. In order to be able to relate new objects, in this case data elements obtained from an Excel sheet, a corresponding object type has to be introduced in the XML schema. Further we export some ADT building components from ADT into this file, and use the relation tool to create relations between these building components and the objects of our new data type within the example XML file.

At the moment there are four different object types available within the existing XML data structure which are used to represent the project data: *building component*, *cost item*, *resource* and *construction activity*. If an application should be integrated into the iRoom that is modeling objects that cannot be assigned to the existing object types, the flexible framework of the iRoom XT makes it easy to extend the underlying data schema by new types.

In the following example the data schema will be extended with the new object type *participant*, which models the different parties involved in construction projects like architects and contractors. In the context of our example we want to integrate a hierarchy of participants in a simple iRoom setup:

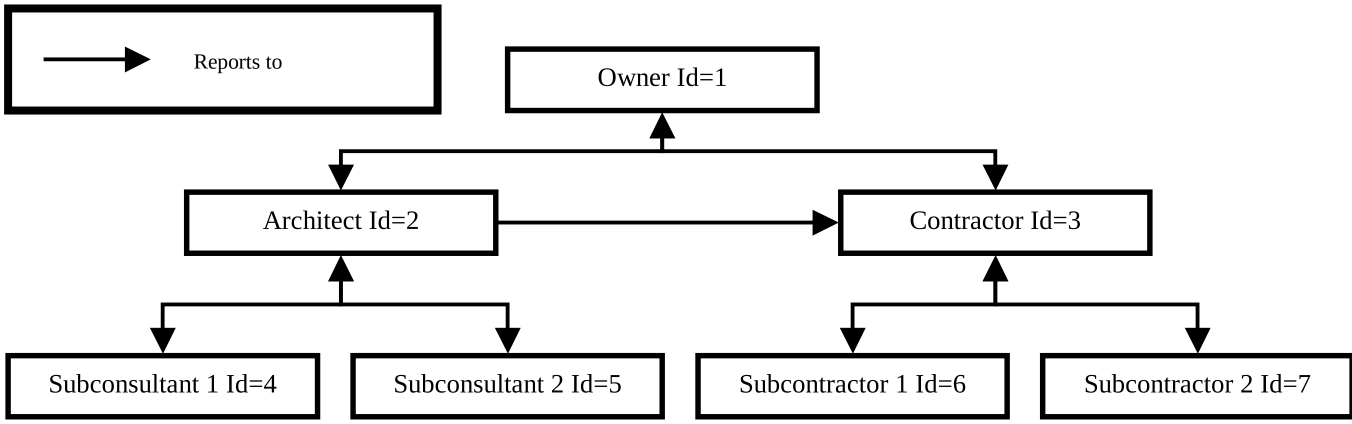
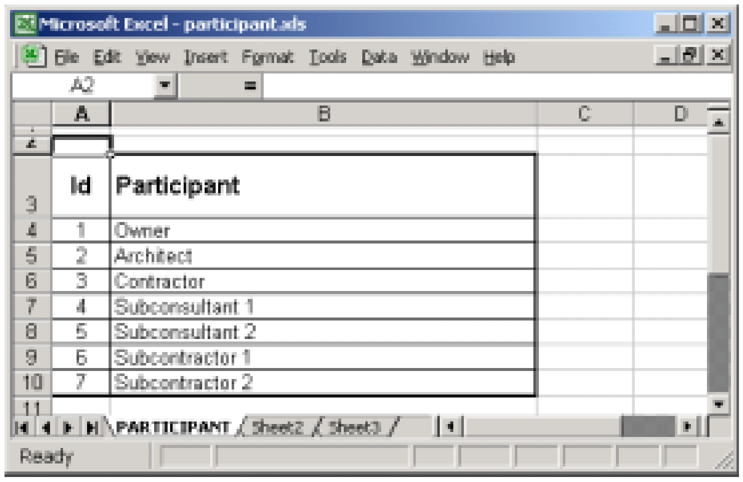


Figure 7 Example for a hierarchical organization of participants in a project

The data about these participants will be modeled in Excel tables like the one shown in figure 8, consisting of information about the name of the participant type and the corresponding ID. In this simplified example, no further attributes of these objects are modeled.


Microsoft Excel - participant.xls
File Edit View Insert Format Tools Data Window Help
A2
PARTICIPANT
Sheet2
Sheet3
Ready

Id	Participant
1	Owner
2	Architect
3	Contractor
4	Subconsultant 1
5	Subconsultant 2
6	Subcontractor 1
7	Subcontractor 2

Figure 8: Participant table for the example.

While creating Excel sheets, which should work with the EHeap Java/VB “listener” for Excel, it is important to define unique ids of objects within the spreadsheet in the A-column, as can be seen in the screenshot above. Further you have to attach a unique name to the sheet, which is later used in the XML data file. Thus we assign the name “PARTICIPANT” to our sheet.
Since the current data schema doesn’t have an explicit *participant* object, we have to introduce this new object type first. Every object within our data schema underlies the following restrictions:

1. Every object needs a name,
2. Every object needs an unique id within the data file and
3. Hierarchies between objects of the same type have to be modeled using the “decomposes Into” and “parent” tags.

In order to introduce our new object we have to edit the underlying DTD-file, which is able to validate XML files containing project data. The DTD-file named Iroom.dtd on the documentation CD has to be copied locally on the hard drive and can then be edited with any text editor, like for example Notepad which comes with every standard Microsoft Windows installation.

First of all we include our new object type *participant* by including the following line into the DTD-file:

```
<!ELEMENT PARTICIPANT EMPTY>
```

Then we have to define the attributes “name”,” id”, “decomposesInto” and “parent”. Further we include another attribute “reportsTo” in order to be able to model the communication between the participants. This can be achieved by including the following code in the DTD:

```
<!ATTLIST PARTICIPANT
       id CDATA #REQUIRED
       name CDATA #REQUIRED
       decomposesInto CDATA #IMPLIED
       parent CDATA #IMPLIED
       reportsTo #IMPLIED
>
```

In this attribute declaration, we set the “id” and “name” attribute to `#REQUIRED`. Thus every created participant XML element has to define a value for these two attributes. The other attributes can be defined arbitrary, thus they are declared as `#IMPLIED`. The last step in the alteration of our underlying DTD file is to subordinate the participant object type to one of the iRoom domains. For our example an appropriate domain is the *RESOURCE_MODEL* domain. In order to integrate the participant type we have to change the following line of code within our DTD:

```
<!ELEMENT RESOURCE_MODEL (RESOURCE*)>
```

to

```
<!ELEMENT RESOURCE_MODEL (RESOURCE | PARTICIPANT)*>
```

After performing the previous steps it is now possible to create a new XML data file modeling the participant hierarchy of our example. Name the previously edited DTD file “iRoomParticipant” and use your text editor to create a new file including the following text:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CIFE_IROOM_PROJECT SYSTEM "iRoomParticipant.dtd">

<CIFE_IROOM_PROJECT id="1" name="" color="#0000FF" numElements="8"
projectStart="" projectEnd="" xmlfile="Participants.xml" desc="">

       <RESOURCE_MODEL id="2" name=”example”>

       <PARTICIPANT id="3" name="Owner" decomposesInto="4 5" reportsTo="4 5"/>
       <PARTICIPANT id="4" name="Architect" parent="3" decomposesInto="6 7"
       reportsTo="3 5 6 7"/>
       <PARTICIPANT id="5" name="Contractor" parent="3" decomposesInto="8 9"
       reportsTo="3 4 8 9"/>
       <PARTICIPANT id="6" name="Subconsultant1" parent="4" reportsTo="4"/>
       <PARTICIPANT id="7" name="Subconsultant2" parent="4" reportsTo="4"/>
       <PARTICIPANT id="8" name="Subcontractor1" parent="5" reportsTo="5"/>
       <PARTICIPANT id="9" name="Subcontractor2" parent="5" reportsTo="5"/>

       </RESOURCE_MODEL>

</CIFE_IROOM_PROJECT>
```

Save the file as “Participant.xml” in the folder containing your created DTD file.

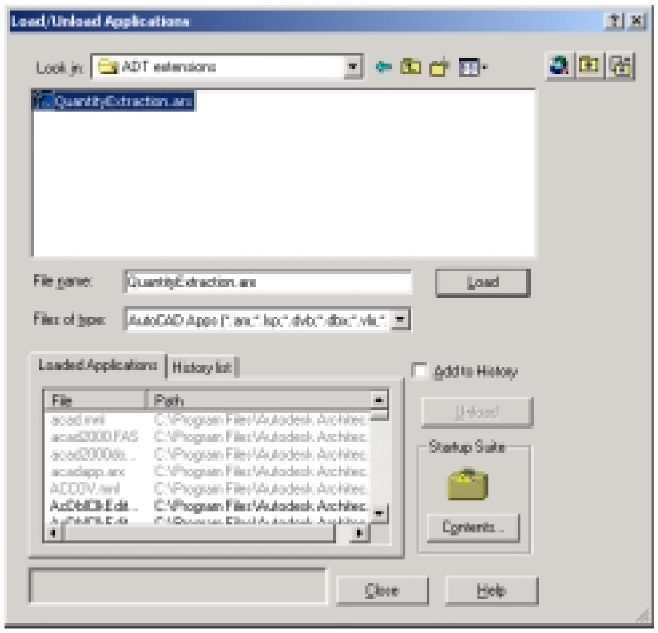
The participants’ unique ids within the data file are not equal to the id used within our Excel example sheet. In order to be able to map these two different ids of objects representing the same participant in the spreadsheet and our database we can now include a “RELATIONSET_ IDMAPPING” relation set in our XML file:

```
       <RELATION_MODEL id="10" name="">

              <RELATIONSET_IDMAPPING id="11" object1="IROOM_ID"
              object2="MSEXCEL_PARTICIPANT" objecttype="PARTICIPANT">

                     <RELATION id="12" idobject1="3" idobject2="1"/>
                     <RELATION id="13" idobject1="4" idobject2="2"/>
                     <RELATION id="14" idobject1="5" idobject2="3"/>
                     <RELATION id="15" idobject1="6" idobject2="4"/>
                     <RELATION id="16" idobject1="7" idobject2="5"/>
                     <RELATION id="17" idobject1="8" idobject2="6"/>
                     <RELATION id="18" idobject1="9" idobject2="7"/>

              </RELATIONSET_IDMAPPING>

       </RELATION_MODEL>
```

Every time you manually edit a XML data file with a text editor, you have to adjust the “numElements” attribute of the XML element “CIFE_IROOM_PROJECT“. Therefore we have to change this attribute to 18, which is the number of our elements now contained within the XML file.

We have created a stand-alone XML file, which can be used to represent the different participants of the Excel example sheet within the iRoom environment. In the next steps we integrate building component elements and relate them with the participant objects creating “RELATIONSET_ IROOMMAPPING” relationship sets with the “Relate Tool”.

First of all we automatically export some building components from ADT to our XML data file using the ADT export implementation. In order to be able to access the export functionality within ADT we have to upload the file “ADTiRoomExport.arx” from the ADT extension folder of the documentation CD. This can be achieved by using the ADT command line command “appload”. The appearing dialog can be used to browse for the “ADTiRoomExport.arx” and upload it by clicking the “Load” button. Now all the entities of a previously loaded ADT model can be exported using the command line command “iRoomExp” and selecting the target XML file.



Look in: ADT extensions

QuantityExtraction.arx

File name: QuantityExtraction.arx

Files of type: AutoCAD Apps (*.arx;*.lsp;*.dvb;*.dbx;*.vlx;

File	Path
acad.mnl	C:\Program Files\Autodesk Architec
acad2000.FAS	C:\Program Files\Autodesk Architec
acad2000do...	C:\Program Files\Autodesk Architec
acadapp.arx	C:\Program Files\Autodesk Architec
ACCOV.mnl	C:\Program Files\Autodesk Architec
AcDblClkEdit...	C:\Program Files\Autodesk Architec

Figure 9	Dialog box for ADT export to the iRoom data repository

Our example XML file should now contain objects of the two different types *BUILDING_ COMP* and *PARTICIPANT*. In order to create the relations between these objects we can use the Relate Tool”. Within ADT we have again first to upload the file “relateInADT.arx” as described above. Then we can start the tool by entering “relateobj” on the ADT command line.

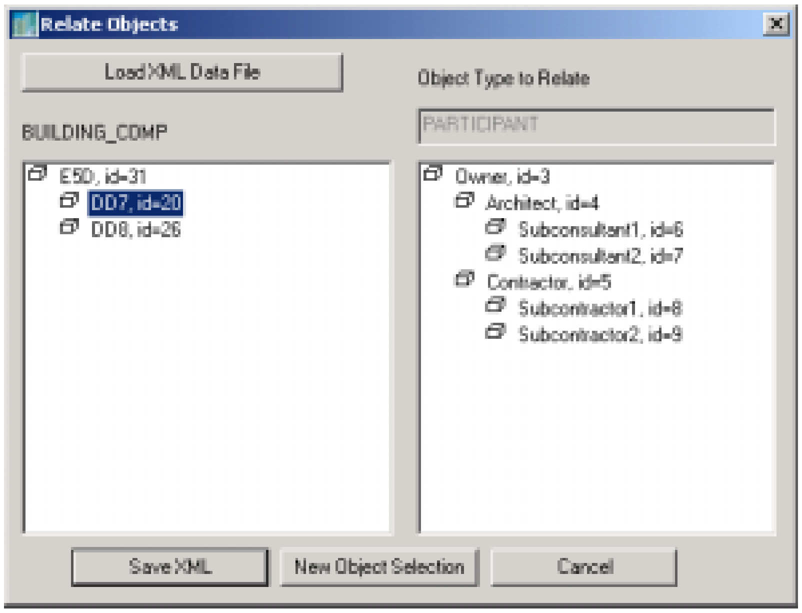
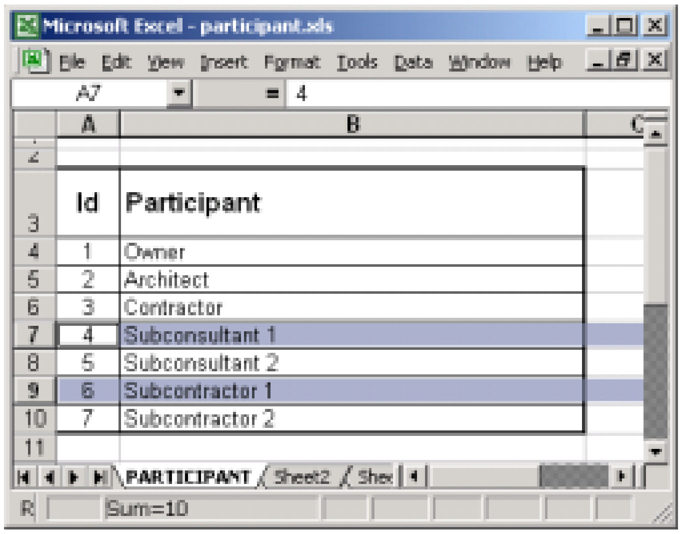


Figure 10 ADT Relation Tool dialog box with PARTICIPANT objects

In the ADT Relation Tool we have to load the example XML file using the "Load XML Data File" button. Then we have to enter "PARTICIPANT" in the edit field. Pressing the "Enter" key on the keyboard, the participants are displayed in the second tree (Figure 10). Next we have to create relations between the two objects by simply dragging one object of one tree on another object of the neighboring tree. For example we can now define a relation between the "Subcontractor1" and "Subconsultant1" to the two slabs of our example project. In order to save these established relations in the XML file we have to click the "Save XML" button. Our XML example file is now ready to be used with the messaging system of the iRoom.

In an iRoom meeting we are now able to use the created XML file, the Excel "PARTICIPANTS" sheet and the ADT sample drawing to highlight related participants within Excel by selecting a slab in ADT. In order to set up our simple scenario we have to first load the example drawing in ADT and upload the ADT extension file "IroomV1.arx" using the "appload" command. Then we have to start the Excel sheet containing our participants. As next step we have to start the MSExcel listener stored on the documentation CD. In the last step in our iRoom example setup we have to start the Midserver.exe file, which is also on the documentation CD. We can start the iRoom services from the Midserver GUI via the "Workspace" menu entry "Start Services". Then we can open our XML file in the Midserver with the "File" menus entry "Open XML database".

Now it should be possible to use the "highlightsel" command in ADT to send a message to the Event Heap. There, the message will be picked up by the Midserver, which generates a message for Excel and sends it back to the event heap. This message is then picked up by the Excel Java "listener" which starts a Visual Basic module that remotely drives Excel to highlight participants related to the selected building component in ADT.


Microsoft Excel - participant.xls
File Edit View Insert Format Tools Data Window Help
A7
= 4
PARTICIPANT
Sheet2
Sum=10

	Id	Participant
4	1	Owner
5	2	Architect
6	3	Contractor
7	4	Subconsultant 1
8	5	Subconsultant 2
9	6	Subcontractor 1
10	7	Subcontractor 2

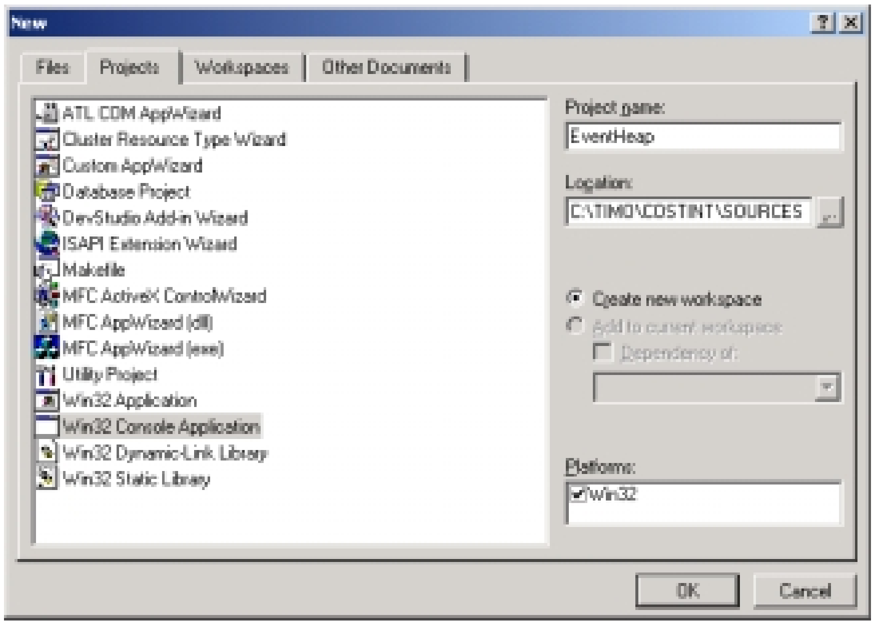
Figure 10 The highlighted PARTICIPANT objects in the Excel sheet

A Sample Windows Console Application Using Visual C++ 6.0

This second example uses the C++ EHeap API to program a simple example connection to the event heap at CIFE. At the end a standalone console application (.exe) should have been implemented which connects to the Event Heap, enables the user to enter arbitrary strings which are send to the heap and picked up by another simultaneously running process. In the example we work with Microsoft's Visual C++ 6.0 programming environment.

Create a new Project in Visual C++ 6.0

1. From the "File" pull down menu of Visual C++ 6.0, select "New."
2. Select the "Projects" tab in the dialog that appears.
3. Select "**Win32 Console Application"** in the list of project types.
4. Enter the desired project name in the Project name edit box.
5. Set the Location to the folder where you want your project to be stored.
6. Select "Empty Project" as the Console application type.
7. Click "Finish" and then "OK".



Project name: EventHeap

Location: C:\TIMO\COSTINT\SOURCES

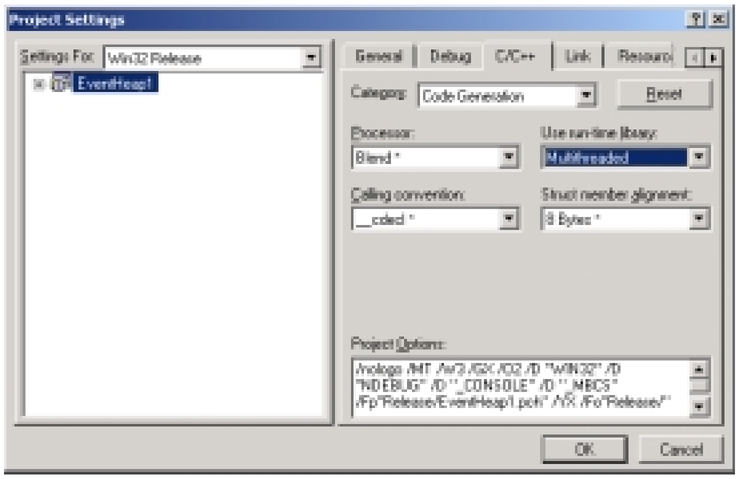
Create new workspace

Platforms: Win32

Figure 11 Settings in the new project menu of the Visual C++ environment

Compiler Settings

8. In the "Project" pull-down menu, select "Settings" to bring up the "Project Settings" dialog.
9. Select the C/C++ tab on the right side of the dialog box to access the compiler settings.
10. In the “Settings For:” list, choose “Win32 Debug”
11. In the “Category:” combo box, select “Code Generation”
12. In the “Use run-time library” edit box, choose “Debug Multithreaded”
13. In the “Settings For:” list, choose “Win32 Release”
14. In the “Use run-time library” edit box, choose “Multithreaded”



Settings For: Win32 Release

Category: Code Generation

Processor: Blend *

Use run-time library: Multithreaded

Calling convention: __cdecl *

Struct member alignment: 8 Bytes *

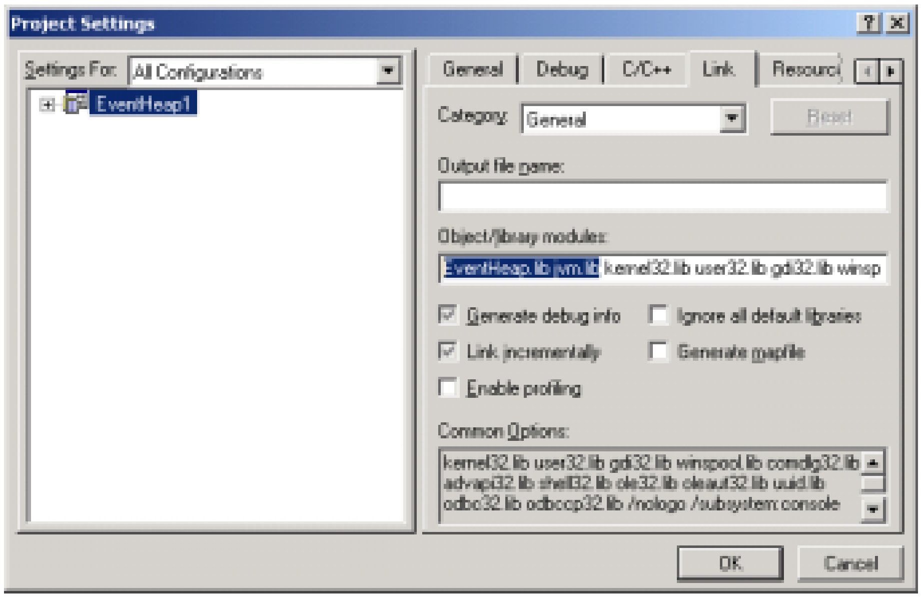
Project Options:

```
/nologo /MT /W3 /GX /O2 /D "WIN32" /D
"NDEBUG" /D "_CONSOLE" /D "_MBCS"
/Fp"Release/EventHeap1.pch" /YX /Fo"Release/"
```

Figure 12 Compiler settings for a Win32 Console Application

LINKER SETTINGS

15. In the "Project settings" dialog box, select the "Link" tab to access the linker settings.
16. Make sure the "Category:" combo box is set to "General."
17. On the left side of the dialog box, select "All Configurations" in the "Settings For:" combo box.
18. In the "Object/library module" edit box, we will need to add two libraries: jvm.lib, which should be found in your Java installation folder, and EventHeap.lib, which can be found on the documentation CD. Copy the EventHeap.lib file in the projects folder chosen in step 6. In order to access the jvm.lib you have to set your environment path to the folder of the jvm.lib file.
19. Close the “Project settings” dialog box by clicking “OK”


Project Settings

Settings For: All Configurations

EventHeap1

General | Debug | C/C++ | Link | Resourc

Category: General

Output file name:

Object/library modules: EventHeap.lib jvm.lib kernel32.lib user32.lib gdi32.lib winsp

[✓] Generate debug info [ ] Ignore all default libraries
[✓] Link incrementally [ ] Generate mapfile
[ ] Enable profiling

Common Options:
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib /nologo /subsystem:console

Figure 13 Linker settings for the example

CONNECTING TO THE EVENT HEAP SERVER

In the "Project" pull-down menu, select "Add to project" "New..." "C++ Source File". Name it "**Main.cpp**" and include the following code:

```
#include "../classes/ceheap.hpp"
int main()
{
        using namespace std;
        // connect to the event heap
        EHEventHeap* theHeap =
                new EHEventHeap("cife", "cife-32.stanford.edu","DOC",-1);
}
```

This already creates an application that builds and establishes a connection to the CIFE Event Heap.

DERIVING AN EVENT CLASS

In the "Project" pull-down menu, select "Add to project" "New..." "C++ Header File". Name it "**MyEvent.h**". In the new file, add the following code to avoid multiple inclusions:

```
#ifndef MYEVENT_H
#define MYEVENT_H


#endif
```

Between the `#define` and `#endif` statement define an integer for the type of the new message and derive a new class from **EHEvent**:

```
#define EH_SAMPLE_EVENT 17

class EHEvent;

class MyEvent : public EHEvent
{


};
```

Within the class body, we need to override the following functions:

```
public:
	MyEvent(EHEventHeap *aHeap);

protected:
	virtual Tuple *getTemplateTuple();
	virtual Tuple *getActualTuple();
```

In the "Project" pull-down menu, select "Add to project" "New..." "C++ Source File". Name it "**MyEvent.cpp**" and add the following includes at the beginning of the file:

```
#include "ceheap.hpp"
#include "MyEvent.h"
```

Further you need to copy the “ceheap.hpp” file from the CD of this documentation to the project folder.

Now define the constructor super-messaging to the parent class. Set the standard field “EventType” to our defined message type integer. Then set the event field “NumAccesses” to 1 and add a new field which can contain a sample message string:

```
MyEvent::MyEvent(EHEventHeap *aHeap) : EHEvent(aHeap)
{

	setFieldValue ("EventType", EH_SAMPLE_EVENT);
	setFieldValue ("NumAccesses", 1);
	addField ("SampleMsg", EHEvent::STRING);


}
```

Define the getTemplateTuple() and getActualTuple() functions, which have to be overridden by every EHEvent derived class. Also define the new created field within the getActualTuple() function using the setNormalFieldValue() function.

```
Tuple *MyEvent::getTemplateTuple()
{

       return EHEvent::getTemplateTuple();

}


Tuple *MyEvent::getActualTuple()
{
       EHEvent::getActualTuple();

       setFormalFieldValue("SampleMsg", "none");

       return ourCurrentTuple;


}
```

Now we have created a new Event class. Objects of this class are defining a new field, “SampleMsg” which may contain an arbitrary string value.

IMPLEMENTING A LISTENER THREAD

In the next step we will implement a new “listener” thread that constantly observes the event heap. This “listener” is able to pick up events of our newly created MyEvent class. Programming threads for windows is a sophisticated field, so be sure to gather more knowledge about thread programming before starting to create your own event heap applications. Information about Multithreading can for example be found in the MSDN documentation shipped with the Visual Studio 6.0 package.

In order to be able to print the string received in the events we use the C++ standard libraries stream function. So add the following include at the beginning of the main.cpp file:

```
#include <iostream>
```

Declare a new global function before the main() function within our main.cpp file:

```
void ListenerFunc(void* source);
```

Implement the Listener function at the end of the main.cpp file:

```
void ListenerFunc(void* source)
{
       // use the namespace of the standard library
       using namespace std;
       cout << endl << "started the thread";

       EHEventHeap* theHeap = (EHEventHeap*) source;
       // create a working copy of the Event Heap
       EHEventHeap tHeap(NULL, NULL, NULL, -1, theHeap);
```

```
      // create a sample Event Type
      MyEvent sampleTemplate(&tHeap);
      MyEvent* newEvent;
      char* msg;

      for(;;)
      {
             // wait for events of the sample event type
             newEvent = (MyEvent*)tHeap.waitForEvent(&sampleTemplate);
             if (newEvent != NULL)
             {
                    cout << endl << "received event ";

                    // get the SampleMsg Field string and stream it to the
                    // console
                    newEvent->getFieldValue("SampleMsg", &msg);
                    std::cout << msg << endl;
             }
      }

      // should not be reached, but you'll never know!
      _endthread();

}
```

In order to be able to receive events we first need to create a working copy (tHeap) of the actual EventHeap, which is passed to this function via the "source" pointer. Then create a new sample event, which can be used within the EHEventHeap::waitForEvent() function to determine the type of message which should be picked up from the heap. This will start an endless loop. In this loop the EHEventHeap::waitForEvent() is used to wait for an event of the respective type. If such an event occurs it will be picked up from the heap and the value of the field "SampleMsg" will be extracted with the getFieldValue() function. Then this value will be printed out to the console. Outside the loop we add the _endthread() function for safety reasons, but it should never be called, as the thread function cannot leave the endless loop.

In a last step we have to start the thread and call our listener function. In order to achieve this add the following includes to the beginning of the main.cpp file:

```
#include <windows.h>
#include <process.h>
```

Then start the listener thread within main(), using the _beginthread function. Be sure to add the function behind the instantiation of your EventHeap object, as the object has to be provided as a parameter for the ListenerFunc() function:

```
// starts the listener thread
_beginthread( &ListenerFunc, 0, (void *) theHeap  );
```

SENDING MESSAGES TO THE EVENT HEAP

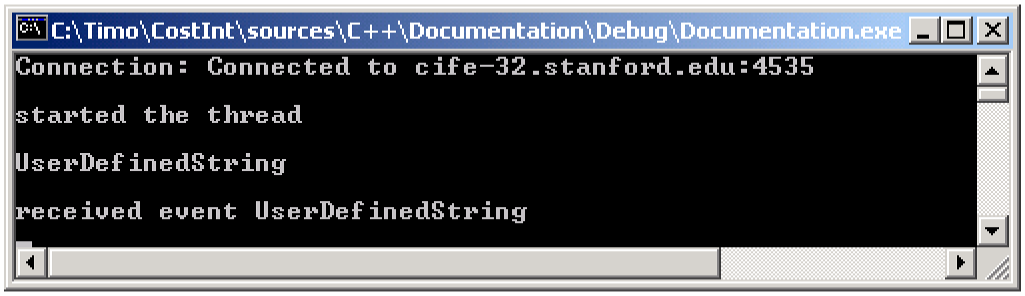
In a last step we'll send messages from our main function to the Event Heap. These messages are using our derived event class. The value of the "SampleMsg" field should be specified by the user via the console. Include the following code before the "`return 1`"-statement at the end of the main() function:

```
for (;;)
{
        // prompt for a string to send
        string in;
        cin >> in;

        if (in == "exit") break;

        // create a char from the std::string object
        char msg[1024];
        lstrcpy(msg, in.c_str());

        // create a new event object, set the “SampleMsg” field an
        // send it to the event heap
        MyEvent* event = new MyEvent(theHeap);
        event->setFieldValue("SampleMsg", msg);
        theHeap->putEvent(event);
}
```

Here we call again an endless loop. Within this loop the user is prompted for a string, via the C++ standard library’s “cin” object. If the string entered is “exit” the program will break out of the loop. If the string contains any other value, a new char “msg” object will be created. Then a new MyEvent object is instantiated and the field “SampleMsg” is set to the value of the “msg” char. Finally the new created event is send to the Event Heap.

RUNNING THE APPLICATION

This application is a simple example to demonstrate the possibilities of events and the event heap. It first starts a “listener” running in a standalone thread, which is waiting for events of a previously defined type. Then messages of that type are sent by the main thread of the application. A “listener”, running in a separate process, then again picks up these messages from the event heap.


C:\Timo\CostInt\sources\C++\Documentation\Debug\Documentation.exe

```
Connection: Connected to cife-32.stanford.edu:4535
started the thread
UserDefinedString
received event UserDefinedString
```

Figure 14 Output of the console application

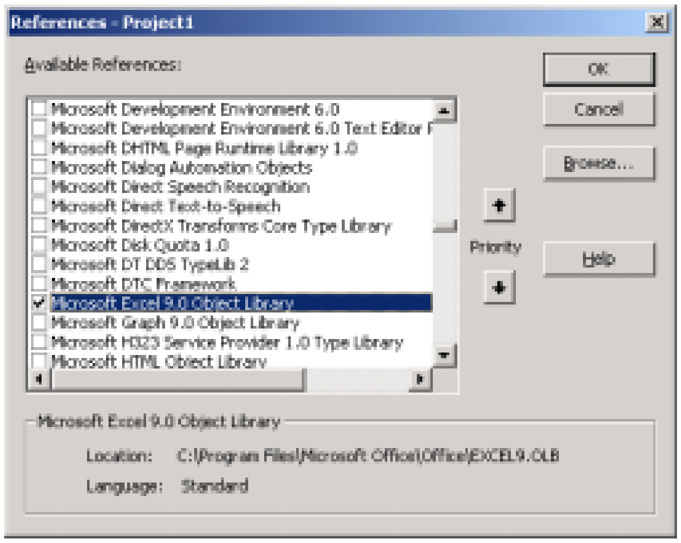
As mentioned before this little example can be used as a starting point in learning to program event heap functionalities using C++. Nevertheless it is strongly recommended to take a closer look at the EHeap API documentation and especially at Multithreading programming using Visual C++ 6.0.

Sample Java/VB Application to Process Message Actions in MS Excel

The third example uses the Java Eheap API to program a simple example connection to the Event Heap at CIFE. At the end a Java “listener” application should have been implemented in Excel which observes the event heap constantly. It is able to pick up messages of a specific format and to start an executable Visual Basic application, which is able to process the messages and remotely drives Excel.

First of all we create a sample VB file, which is able to extend Microsoft Excel with further funtionalities. This VB file simply adds the sent message to the A1 cell of an empty Excel spread sheet. In order to create a VB .exe application we start Microsoft Visual Basic 6.0. In the new Project Dialog we select “Standard EXE” and click the open button. We do not need a form so we remove the standard form with selecting the “Remove Form” command in the right click context menu of the form in the Project Explorer window. If the Project Explorer Window is not active you can activate it using the “View” menu.

In the next step we add a new module, by selecting the “Add Module” entry of the Project menu. We select the Module icon and click the open button. In order to be able to use Visual Basic Excel objects we have to load the reference to the Microsoft Excel Object Library. This is achieved by selecting the “References” entry of the “Project” menu. In the “References” dialog we select the “Microsoft Excel 9.0 Object Library” and click the OK button.



Available References:

- [ ] Microsoft Development Environment 6.0
- [ ] Microsoft Development Environment 6.0 Text Editor F
- [ ] Microsoft DHTML Page Runtime Library 1.0
- [ ] Microsoft Dialog Automation Objects
- [ ] Microsoft Direct Speech Recognition
- [ ] Microsoft Direct Text-to-Speech
- [ ] Microsoft DirectX Transforms Core Type Library
- [ ] Microsoft Disk Quota 1.0
- [ ] Microsoft DT DDS TypeLib 2
- [ ] Microsoft DTC Framework
- [✓] Microsoft Excel 9.0 Object Library
- [ ] Microsoft Graph 9.0 Object Library
- [ ] Microsoft H323 Service Provider 1.0 Type Library
- [ ] Microsoft HTML Object Library

Microsoft Excel 9.0 Object Library

Location: C:\Program Files\Microsoft Office\Office\EXCEL9.OLB

Language: Standard

Figure 15 Referencing the MS Excel Object Library in the Visual Basic programming environment

Now we include the following code in the new created Module:

```
Sub Main()
Dim app As Excel.Application
   Dim sheet As Worksheet
   Set app = GetObject(, "Excel.Application")
    Dim actWorkbook As Object
    Set actWorkbook = app.ActiveWorkbook

    Set sheet = actWorkbook.ActiveSheet

    sheet.Range("A1").Value = Command$
```

```
 End Sub
```

In the code we want the module first to get an object of the running Excel application. If there is no application running the module will exit with an error. Then we get the active sheet in Excel and insert the parameter used to start the VB .exe application in the cell “A1” of the active sheet. In order to create a windows .exe file we can build our VB module with the entry “Build Project1 .exe” of the “File” menu. All we have to do is to enter the name and the path for the application.

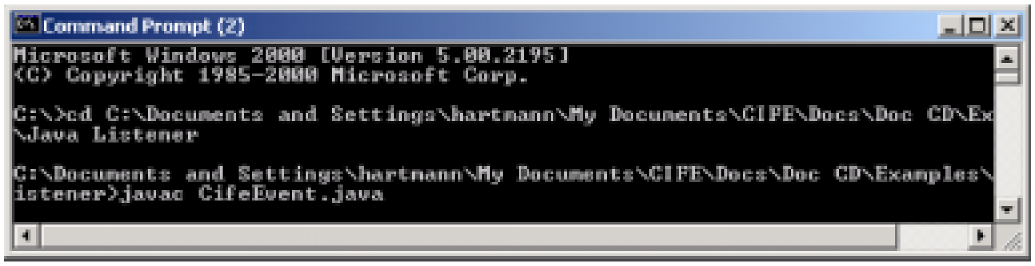
In the next step we will now create the Java listener, which surveys the event heap, picks up any message and starts the built VB .exe application with the a string contained in the event as parameter. In order to be able to pick up messages from the event heap, we first have to create a new Java class, which is derived from the Event class of the EventHeap Java API.

Simply open a new file in a text editor and include the following code:

```
import com.ibm.tspaces.Tuple;
import iwork.eheap.*;



public class CifeEvent extends Event {

    public static String CMDTOEXECUTE = "CommandToExecute";

    public static Integer CIFE_EVENT = new Integer(2350);
       public static Integer GROUPCIFE =new Integer(1200);

    public CifeEvent() throws EventHeapException {
       super();
       setFieldValue( EVENTTYPE, CIFE_EVENT );
      setFieldValue( TIMETOLIVE, 120000 );
       setFieldValue( NUMACCESSES, 1 );
       setFieldValue(GROUPID ,GROUPCIFE);

       addField( CMDTOEXECUTE, String.class );
    }

    public Tuple getActualTuple()
       throws EventHeapException {

       Tuple ret = super.getActualTuple();
       setFormalFieldValue( CMDTOEXECUTE, NULL_STRING );

       return ret;
    }

}
```

In our newly derived class we first define four different fields for our new event. The “EVENTTYPE” field is used by listeners to identify different types of events. The “TIMETOLIVE” field specifies the time the event should be stored on the event heap in milliseconds. The “NUMACCESSES” field specifies the number of possible accesses for the event, before it is erased from the heap. The “GROUPID” field can be used to bundle a number of different event types to an event group. Then we add an additional field “CMDTOEXECUTE” that actually contains the message stringfor Excel.

We have to save our new file in the same directory as the VB .exe application. It is important to assign the name “CifeEvent.java” to the file, in order to be able to compile it with the Java compiler. Now we can compile the file using a Java compiler. We use the standard Sun Java Dos compiler “javac”(Figure 16). However other compilers can be used as well.



```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>cd C:\Documents and Settings\hartmann\My Documents\CIFE\Docs\Doc CD\Ex
\Java Listener

C:\Documents and Settings\hartmann\My Documents\CIFE\Docs\Doc CD\Examples\
istener>javac CifeEvent.java
```

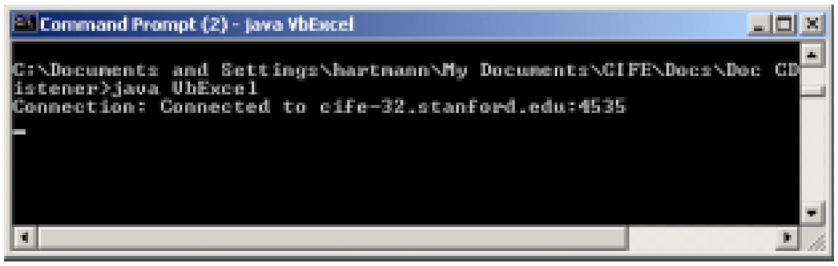
Figure 16 MS-DOS command box view when compiling the java listener with javac

In the next step we have to create another Java class, which starts the listener and waits for events of our new created type. We use again a standard text editor and include the following code:

```
import java.util.*;

import java.io.IOException;
import java.net.UnknownHostException;
import iwork.eheap.*;

public class VbExcel{

  public static void main(java.lang.String[] args) {
       EventHeap myTupleSpace;
       String heap =  "cife";
       String server = "cife-32.stanford.edu";

       if (args.length == 2)
       {
              heap = args[0];
              server = args[1];
       }

        try {
                     myTupleSpace = new EventHeap(heap,server);

                     iwork.eheap.Event newevent=null;
                     if(myTupleSpace!=null)
                     {
                            for(;;)
                            {
                                   try {
                                          CifeEvent template = new CifeEvent();
                                         newevent=
                                         myTupleSpace.waitForEvent(template);
                                          if(newevent!=null)
                                          {
                                          String message=
                                         (String)newevent.getFieldValue
                                         (CifeEvent.CMDTOEXECUTE);


                                                System.out.println(message);
                                                 try{
                                                      Runtime r =
                                                     Runtime.getRuntime();

                                                      Process p ;

                                            p =r.exec
                                                 ("VisualBasic.exe
                                                 "+message);
                                                  p.waitFor();


                                                  }

                                                  catch(Exception e)
                                                 {System.out.println("Error");}


                                            }
                                       }
                                       catch (EventHeapException ehe){}
                                  }
                             }
                        }
        catch(EventHeapException tse) { tse.getDetail(); }
      }
}
```

Similar to the C++ “listener” we wait in the Java “listener” for a new event within an endless loop. If an event of the respective type occurs on the event heap, the Java “listener” starts our created Visual Basic .exe application with the incoming message’s “CMDTOEXECUTE” field’s value. Then the Visual Basic .exe application remotely drives Microsoft Excel to print the message value in the “A1”-cell of the active spreadsheet.

After saving the file as “VbExcel.java” we again have to compile our new class with a Java compiler. Now we can start the new Java class. Using the standard Sun Java [7] installation for Microsoft Windows this can be performed by using the DOS command “java” followed by the class name of the compiled class (Figure 17). A connection to the event heap is established, and the “listener” starts to wait for new events.



```
C:\Documents and Settings\hartmann\My Documents\CIFE\Docs\Doc CD
istener>java VbExcel
Connection: Connected to cife-32.stanford.edu:4535
_
```

Figure 17   Command box view when starting the listener for MS Excel

In order to test our application we can use the Midserver program on the accompanying documentation CD. Within in the Midserver we can create a connection to the Event Heap with the “Workspace” menu entry “Start Services”. Now we can send a test event containing an arbitrary string to the event heap by using the “Workspace” menu entry

"Send Message". The sent string should now appear in the "A1" cell of the active Excel worksheet.

CONCLUSIONS

As our work with the iRoom at CIFE was limited to three months, we concentrated mainly on establishing a clear and stable software architecture, defining a central schema for the data shared by the applications connected to the iRoom. The implemented solutions are intended to be prototypes to validate our assumptions and test the established communication system. Nevertheless they can already be used to set up simple iRoom scenarios and to create parts of a XML data file using the schema described in this document automatically.

Civil engineering projects generally involve many different parties. Unfortunately all these parties use a wide variety of different applications. The integration of more commonly used AEC applications in the iRoom environment would enable practitioners to use their favourite software within iRoom project meetings.

In our opinion, further iRoom developers should therefore favour the task to integrate more applications that are used in modern AEC companies. At the current state, there are just a few applications integrated, like Microsoft Excel, Microsoft Project, Commonpoint 4D and Architectural Desktop. Especially Microsoft Excel, currently used to represent the cost and resource domains, should be replaced with software programs like Timberline or Sfirion. Another example would be the integration of Primavera which would enable project managers to use this software to participate in iRoom meetings without having to create a new project model with MS Project. This will increase the overall acceptance of using interactive workspaces in project meetings.

In a next step the general communication functionality could be extended. Within the here described communication system, we use a character string to distribute information in the iRoom environment. Event heap messages offer the possibility to define multiple fields to store message data. The messages used at the moment consist of only one field, containing the above mentioned data string of the message. Further fields could be defined for these messages to enable the distribution of more data.

Currently only a prototype has been implemented to highlight related objects of different types in different applications. It would be possible to enhance this iRoom functionality by extending the message format with new fields. Thus the propagation of data changes within the message could be established. This would enable participants in iRoom meetings to easily predict the impact a data change of one object has on the data of other objects. For example in a simple scenario the labour resource assigned on a construction activity has to be increased due to problems in the overall schedule. Introducing an accurate communication between schedule and estimation software, the impacts on a related cost position within the cost estimation could be automatically predicted. This communication could be implemented using the messaging techniques of the iRoom environment.

We are strongly convinced that the iRoom can be more than supportive just for descriptive and explanative tasks within meetings. With further development and accurate implementation the iRoom could become an environment that is able to support predictive tasks within multi-disciplinary project meetings using messages to propagate data changes within the integrated applications.

RECOMMENDED READING

[1] iRoom Documentation. http://cife.Stanford.edu/IROOM/

[2] Hartmann, Timo: “Integration of a Three Dimensional CAD Environment into an Interactive Workspace.” Master Thesis. Lehrstuhl für Bauinformatik, TU München / CIFE, Stanford University, December 2002

[3] Liston, K.; Fischer, M. and Winograd, T. (2001). “Focused Sharing of Information for Multidisciplinary Decision Making by Project Teams.” ITCON (Electronic Journal of Information Technology in Construction), Vol. 6, 69-81, http://www.itcon.org/2001/6/paper.pdf

[4] “Interactive Workspaces – Event Heap Documentation.“, Computer Science, Stanford University, http://graphics.stanford.edu/projects/iwork/software/htmlpages/documentation.html

[5] Fischer, M.; Garcia C.; Kam, C.; Kunz, J.; Liston, K.; Schreyer, M.; Singhal, V.: “Virtual Design and Construction Case Study“, CIFE, Stanford University, CIFE Summer Program 2002

[6] W3C: “Document Object Model (DOM).” http://www.w3.org/DOM/

[7] Sun Microsystems, Inc.: “The source for Java technology.” http://java.sun.com/