Making a C2 Information System Platform Independent by Using Internet and Middleware Technologies

Gerhard Bühler & Heinz Faßbender
Research Establishment for Applied Sciences
Research Institute for Communication, Information Processing, and Ergonomics
Neuenahrer Straße 20
D-53343 Wachtberg-Werthhoven
Germany
Telephone numbers: (+49) 228/9435 376 & (+49) 228/9435 640
E-mail addresses: buehler@fgan.de & fassbender@fgan.de

Abstract

We describe the concepts and experiences we have made in an ongoing project by applying the middleware standard CORBA and the internet programming language Java in redesigning the experimental C2 information system EIGER for making the user able to use the EIGER system from any platform which is connected to the world wide web and which consists at least of a world wide web browser which, in its turn, is able to run Java applets.

1. Introduction

By an ongoing project of our institute we want to demonstrate that an experimental C2 information system which is totally platform dependent, because it is implemented on a sun workstation under Solaris, can be redesigned by using

- internet technologies as web browsers and Java applets [Java, 1999] and
- middleware technologies as CORBA [CORBA, 1999]

in such a way that it becomes totally platform independent and hence, can be used on every computer which at least consists of a world wide web browser which is able to run Java applets and which is connected to the internet.

In the following we want to describe the ideas we have produced and the experiences we have made in the described project and furthermore, we develop a general strategy by which an arbitrary platform dependent C2 information system or more generally a legacy system can be redesigned in such a way that it can be used totally platform independently where only a very small amount of system resources is required.

For this purpose, we have structured the paper as follows: We start with an overview over the existing version of the experimental C2 information system. After that, we motivate the application of CORBA and Java, present the concepts which should be applied and finally, we describe our experiences in applying these concepts to the experimental C2 information system.

2. The Experimental C2 Information System EIGER

In our department *Command and Control Information Systems* of our institute an experimental distributed C2 information system called EIGER, has been developed which can support the work of head quarters of the German army. That means that the system has to satisfy the informational and computational requirements of a large organization with an extremely distributed area of responsibilities. In its turn, the organization consists of many units which are distributed over a large geographical area. Each unit has to manage a restricted but naturally non disjunct amount of jobs and data.

As mentioned before, EIGER is not an operational system. It is an experimental system which, at the moment, serves as test bed for functions which are needed as German part in the context of the ATCCIS project (Army Tactical Command and Control Information System) of the NATO [Wagner and Markmann, 1996] where replication in heterogeneous systems is the main research aspect.

Nearly 90% of the system have been coded in Ada 83. The other 10% have been coded in C for realizing the communication of the system's components. For our current work, EIGER has been changed so far that it can be compiled by an Ada 95 [Barnes, 1995] compiler which ensures that more elegant and more time efficient mechanisms can be integrated into the system. We use the Object Ada system from Aonix as development environment. The current version runs on Sun workstations with Solaris. Since the kernel of the system can be compiled and linked by the Object Ada system on Windows NT, we have realized the first step of transporting the complete system to Windows NT.

Unfortunately, as mentioned before, the architecture of the existing version is *totally platform dependent*. This means that EIGER can be only used on Solaris which extremely restricts its applicability, in particular in the German army where the usage of Windows systems is rapidly increased.

This grievance should be abolished by the new version. And in the following sections we will discuss, how we will redesign the current system such that:

- the system becomes totally platform independently and
- we can use the internet for applying the system.

But, before we are able to do this, we have to discuss some basic aspects of the current structure of EIGER.

EIGER consists of a finite amount of subsystems the communication of which is realized by a communication system (cf. Figure 1). A subsystem consists of a finite amount of computers and their operating systems where there are only few requirements with respect to the computers, the operating systems, and their connections. In general, the computers of organization units with common data are connected by a local network. The subsystems of an organization unit form one area.

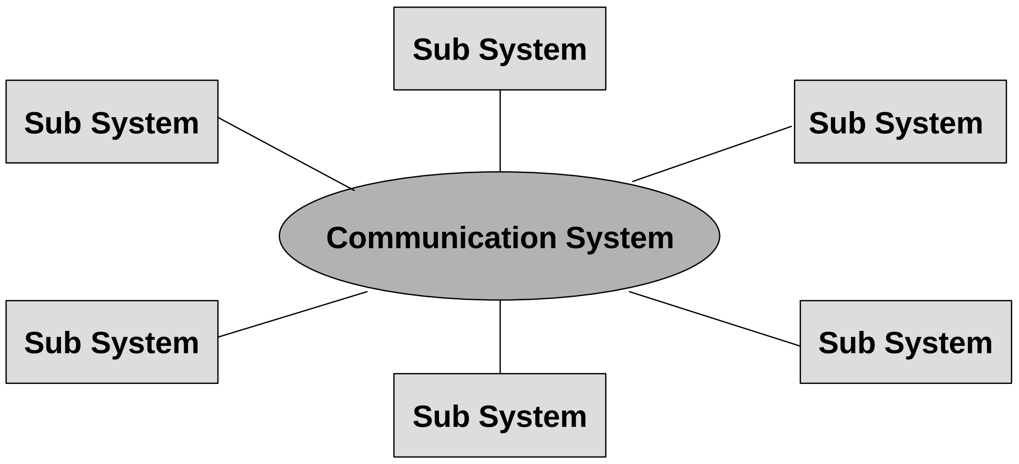

Figure 1. EIGER's topology

The implementation of the communication system is not important for understanding the redesigning mechanisms. For this purpose, we only have to understand, how a subsystem is structured (cf. Figure 2).

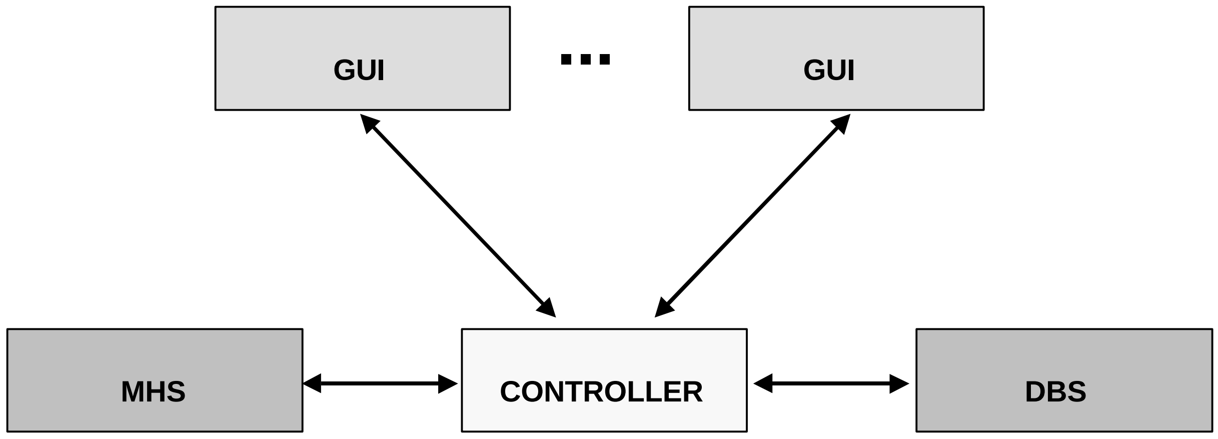

Figure 2. Structure of a subsystem

With great foresight a subsystem has been structured as a multi tier process system. Without the distribution and strong modularization of the subsystem the modification would be not as efficient and simple realizable as it will presented in the following paragraphs.

The central unit of a subsystem is the *controller* which controls the computations and communications of the other components of a subsystem. This component is completely implemented in Ada83.

The controller is connected to a *relational data base system* DBS which stores the data of the organization corresponding to the underlying data model. The connection between the controller and the DBS is realized by a particular process system, because a connection by "embedded

SQL" is invaluable, since this solution would not allow concurrent processes; i.e., the controller would have to wait until the complete result is computed by the DBS. But the controller may need this time for other acitivities. Furthermore, data base transactions may be run in parallel in the controller and this functionality could be not controlled by "embedded SQL".

Furthermore, the controller is connected to an *X.400 message handling system* MHS. This connection is realized by a particular process system, too, where the reasons for the additional process system are identical to those of the process system for connecting the controller to the data base system.

As illustrated in Figure 2, the controller is connected to one or more *graphical user interfaces* GUI which are produced by OSF/Motif and which realize the interface to the users of the system. After a successful login, the user gets some objects on his screen. The central object is the *session* which consists of

- one basket which includes notes,
  By this basket, the user receives system messages.

- at least one basket which includes mails,
  These baskets offer the basic services of a mail system.

- at least one entry basket,
  By these baskets, the user receives some jobs he has to do

- and at least one working basket.
  These baskets include the jobs which are currently worked on by the user.

We stop the introduction of the existing EIGER system at this point, because we hope that the short impression of EIGER's structure is sufficient for understanding the concepts for redesigning subsystems such that EIGER can be used on the internet totally platform independently which will be discussed in the following sections.

A much more detailed introduction to the structure of EIGER is presented in [Bühler, 1998].

3. Concepts for EIGER's Redesign

We integrate the following two concepts into the implementation of a subsystem for getting the desired platform independence :

- A graphical user interface shall be implemented as Java applet. Then it (and thus, the complete EIGER system) can be applied on every computer system which is connected to the internet and which includes at least a web browser which in its turn is able to run Java applets.

- The communication between the controller and the graphical user interfaces shall be realized by a CORBA connection. This also supports the platform independent use of the system, since CORBA, in opposite to DCOM [Sessions, 1998], is a system independent standard which can be implemented on nearly every system for nearly every operating system and nearly every programming language.

Integrating these two concepts leads to the following new structure of a subsystem which is illustrated in Figure 3. Remark that we omit the connections to the data base system and to the message handling system, because they are not changed in this first restructuring step.

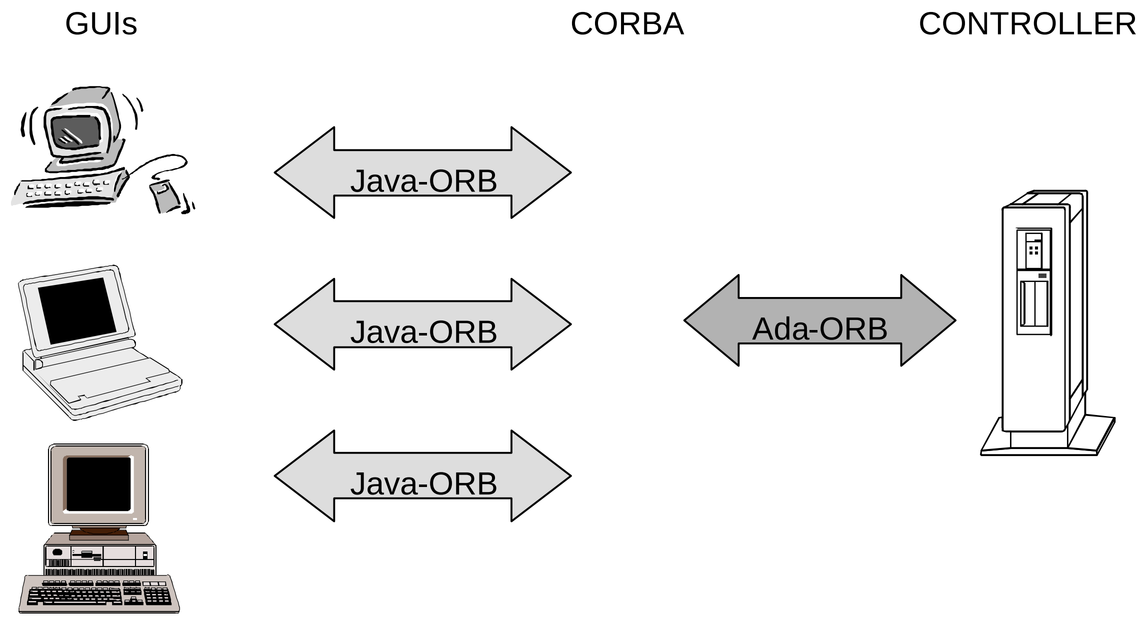

Figure 3. Redesign of a subsystem

4. Problems and Solutions

In this section, we discuss the problems which have to be solved for implementing the structure which is illustrated in Figure 3, and present our solutions.

4.1 Reimplementation of GUIs as Java Applets

In the existing version of EIGER GUIs are implemented with OSF/Motif. Now, they have to be implemented as Java applets.

This problem is completely solved by a new implementation of the GUIs as Java applets under the system JBuilder2 from Inprise (Borland & Visigenic). For example, the first image on the screen, namely the login screen, after starting the connection from the browser to EIGER by loading the html-page is illustrated in Figure 4. Into this image, the user has to enter his user-id (Benutzer-ID), his password (Paßwort), and the kind of the system. After that, the user has to hit the *Ausführen*-Button. Then the first transaction between the GUI and the controller is started. This transaction will serve as an example for explaining the realization of the CORBA connection in the following subsection.

4.2 Realization of the CORBA Connection

In the existing version of EIGER the GUIs and the controller are connected by a TCP/IP connection with 47 defined protocol elements which are sent from the controller to the GUI or vice versa. For example, we illustrate the behavior of the 0ßexisting system during the login phase. If the user hits the *Ausführen*-Button in the login screen of the existing system, i.e. not the screen as shown in Figure 4, but the screen which is produced by OSF/Motif which is defined analogous to the Java applet screen, then the protocol element SS*T_OP_CONN_RQ* (where SST denotes the controller) and three parameters for the user-id, the password, and the kind of system are sent from the GUI to the controller. After that, the GUI is waiting until it will get the protocol element SS*T_OP_CONN_RS* as response. Then the environment of the session, as described in Section 2, is build up on the GUI by a sequence of protocol elements which are sent from the controller to the GUI.

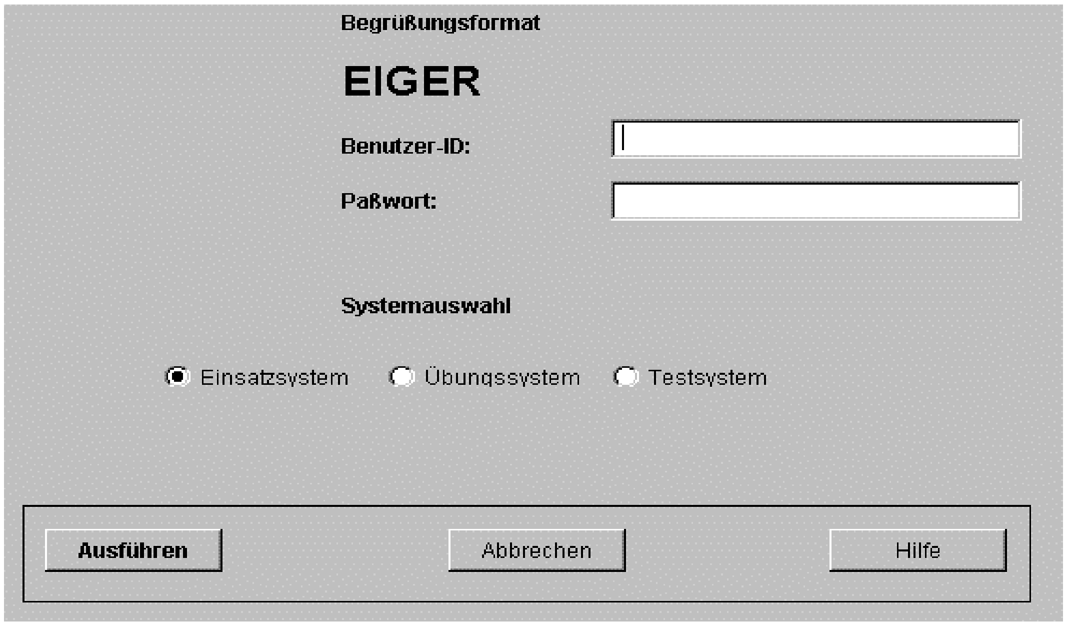

Figure 4. The Login Screen

In the new implementation of EIGER, the previous behavior is realized by a function which is defined in the *Interface Definition Language (IDL)* of the CORBA standard [CORBA, 1999] as follows:

```
void Login_Computation(
 in IDL_Pers_Ids                    Pers_Id,
 in IDL_Names                       Password,
 in IDL_Systemtypes                 Systemtype,
 in GUI_TO_CONTROLLER               Gui_Ref,
 out IDL_Working_Baskets_Lists      Workingbasket_List,
 out IDL_Entry_Baskets_Lists        Entrybasket_List,
 out IDL_Mail_Baskets_Lists         Mailbasket_List,
 out IDL_Note_Baskets               Notebasket);
```

The function Login_Computation has four in-parameters where the first three indicate the user-id, the password, and the kind of system. The fourth in-parameter Gui_Ref indicates the reference to the object which represents the GUI. This reference is needed, if the controller will call services of the GUI, i.e. if the GUI offers server functionality. In this case, the reference serves as indicator of the session.

The four out-parameters of the function Login_Computation indicate the contents of the baskets of the session which are described in Section 2.

The function Login_Computation illustrates the main differences between realizing the connection between the GUI and the controller as TCP/IP connection in the existing version and the CORBA connection in the new version. In the existing version, the connection is realized by sending protocol elements. In the new version, functions with in- and out-parameters are defined in IDL. These functions which are also called services in the following, have to be implemented on server side (in the example, the controller) and can be called from the client side (in the example, the GUI). In particular, if the client and the server are implemented in different languages, the programmer has not to take care of the transformation of data representations between these two languages. This is realized by CORBA. Furthermore, the client gets the pieces of information delivered by the out parameters in one go, whereas it gets it asynchronously by protocol elements in the existing version of EIGER.

In the definition of the IDL we have defined the following two interfaces:

- GUI_TO_CONTROLLER
  This interface specifies 18 services where the controller is the server and the GUI is the client, e.g. it includes the function Login_Computation.

- CONTROLLER_TO_GUI
  This interface specifies 30 services where the GUI is the server and the controller is the client, e.g. it includes a function Delete_Note which deletes a note in the basket of notes of the session.

After giving this general impression of the CORBA connection, we discuss the specific solutions in realizing the CORBA connection in our environment.

4.3 Connection of the GUI by a Java-ORB

As Java-ORB we use the Visibroker from Inprise which is integrated into the JBuilder2 Client and Server Version. We have not had any problem for communications of Java applications. But some problems arose for communications of Java applets. Nevertheless, by using the newest version of Netscape these problems have also been solved.

In the implementation of the GUI an object *GUI-object* of the type CONTROLLER_TO_GUI (cf. previous subsection) is constructed. But, since this object only serves as secondary server for the controller in the specific session, it is not directly connected to the naming service of the ORB. Instead, as mentioned before, the object-id is sent to the controller as fourth parameter of the function Login_Computation and the services of this object can be used by the controller by using this object-id. The behavior that the main server, i.e. the controller can use services from the client, i.e. the GUI is called *Call-Back mechanism* [CORBA, 1999].

Furthermore, in the other direction, the GUI can use services from the controller. Since the user of the GUI shall not wait until the GUI gets the result from the controller before he can call another service, the service calls are realized as threads [Java, 1999] which are concurrently computed.

4.4 Connection of the Controller by an Ada-ORB

We use the ORBADA system from Top Graph'X as Ada-ORB which works well after some discussions with the hotline and some corrections from Top Graph'X in a previous version of ORBADA.

The connection of the Ada-ORB to the controller which is tricky, is realized as follows: Corresponding to the implementation of the GUI, an object *CON-object* of the type GUI_TO_CONTROLLER (cf. Subsection 4.2) is constructed. But, in opposite to the object which is constructed by the GUI and since the controller serves as main server, this object is explicitly bound to the ORB. After that, the CORBA Main_Loop is started which is a procedure that is waiting for demands of services and, after receiving demands, it calls the implementations of the services and returns the results of the services to the demanding object.

This behavior is generally usable and not very tricky. The tricky part of the solution is the implementation of the services of CON-object which will be explained now.

Since EIGER is a distributed system, we have to manage parallel processes efficiently. In particular, we have to ensure that the whole system does not stop because of waiting for the result of a running process, if it could continue its work by running another job. This problem has also arisen in the existing version of EIGER. There it has been solved as follows:

The controller performs an infinite loop for message handling which consists of the following four steps:

- *Receive Message*
  where the messages can be received either from the data base system, the message handling system, or from one of the GUIs.

- *Create Service Control Block*
  A service control block which contains the information of parameters and message code, is constructed for every received message.

- *Delete Message*
  If the complete information which is needed, is stored in the service control block, then the message is deleted.

- *Evaluate Service Control Block*
  The service which is demanded by the message, is evaluated now and its results are sent to the calling entity of the subsystem by help of the corresponding service control block.

This mechanism works well in the existing version of EIGER and furthermore we think that it is also the best solution for the new implementation. Hence, we do not want to change its successful behavior. But, now we have to transport the messages by the CORBA-ORB and therefore, we have to apply a concept which realizes a combination of the described mechanism and the CORBA facilities. This concept works as illustrated in Figure 5.

In particular, every service (which is nothing else then a function) of the CON-object has to be implemented. This is realized as follows: Since the implementation of the service still exists in a form of a process which is evaluated by the process system, we need an adapter for this service which transforms the service from the form which is called by the CORBA Main_Loop into the process which can be evaluated by the process system.

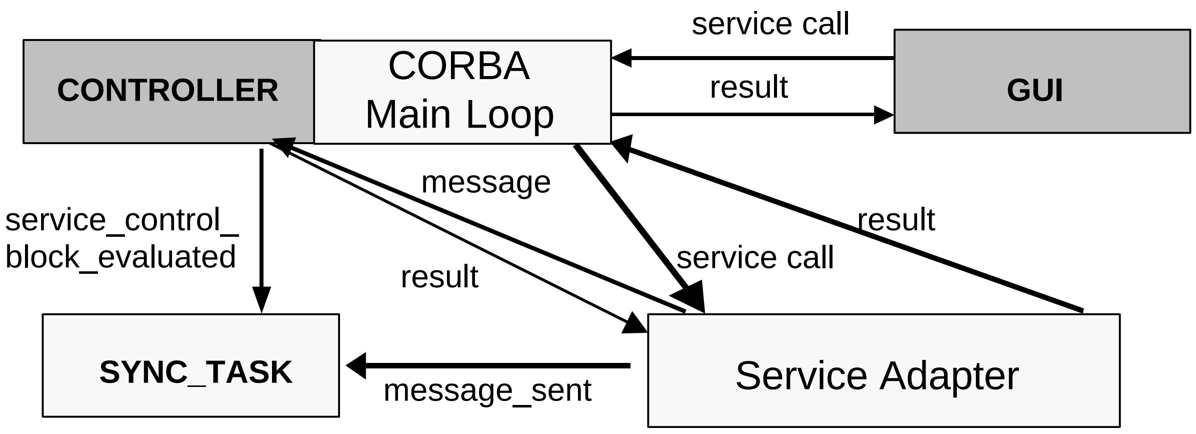

Figure 5. Management of parallel processes.

Each of these adapters which are named by the corresponding service name, has the following form:

```
send message to CONTROLLER

SYNC_TASK.message_sent

send result to CORBA Main Loop
```

where the synchronization task SYNC_TASK which is started for every service call, has the following form:

```
accept service_control_block_evaluated

accept message_sent
```

The synchronization task SYNC_TASK has two entries which realize the rendezvous principle of Ada95, i.e. in this case, the service adapter wants to enter the entry message_sent after it has been started by the CORBA Main_Loop and it has sent a message with the parameters and a pointer to SYNC_TASK to the controller. Since message_sent is the second entry, the service adapter has to wait until the controller enters the first entry service_control_block_evaluated. But this entry is entered not before the controller finished the evaluation of the message and it has sent the result to the service adapter. That means, the service adapter can continue its work by sending the result to the CORBA Main_Loop not before it has got the result from the controller.

We have checked the validity of the presented concepts by implementing the service adapters for some services which are offered by the CON_object.

4.5 Communication between Java-ORB and Ada-ORB

The applied CORBA systems work well together. So, we can state that they are CORBA conform at least in the checked services.

5. Conclusions and Future Work

We have discussed the concepts and experiences we have made in redesigning an experimental distributed C2 information system. By this experiment, we have shown that a large distributed system which is coded in Ada83, can be restructured by applying Ada95, Java, and CORBA in such a way that it becomes totally platform independent.

Nevertheless, a lot of problems which have had to be solved, have arisen, because the market for Ada products is not as well as comparable markets for other programming languages. But, on the other hand, we can summarize that the concepts of Ada95 are well suited for implementing a distributed command and control information system. E.g., the management of parallel processes can be realized extremely well by Ada95.

In particular, we have realized an efficient call back mechanism by our implementation, since the controller as well as the GUIs have server and client functionality, i.e., we have realized a binary communication relation between GUI and controller.

In our future work, for transporting the system to Windows NT, we have to realize a new connection between the controller and the DBS and MHS. The connection to MHS seems to be simple, whereas we want to apply ODBC for realizing the connection between the controller and the DBS. As a further important research topic we have to ensure that the presented concepts are efficient.

6. References:

[Barnes, 1995] John Barnes . *Programming in Ada°95.* Addison-Wesley, 1995.

[Bühler, 1998] Gerhard Bühler. *Einsatz von Ada im Experimentellen Führungsinformations-system EIGER.* in Workshop „Entwicklung von Software-Systemen mit Ada“, Bremen, Germany, Ada Germany, 1998.

[CORBA, 1999] see http://www.omg.org.

[Java, 1999] see http://www.javasoft.com

[Sessions, 1998] Roger Sessions. *COM and DCOM: Microsoft’s Vision of Distributed Objects.* Wiley, 1998.

[Wagner and Markmann, 1996] Karlheinz Wagner & Günther Markmann. *Interoperability Aspects of Command & Control Information Systems with Respect to International Standards and Emphasis on ATCCIS.* Report No. 469 of the Forschungsinstitut für Funk und Mathematik, Wachtberg-Werthhoven, Germany, 1996.