

Applying Ada, Java, and CORBA for Making a Command and Control Information System Platform Independent

Gerhard Bühler

Research Establishment for Applied Sciences
Research Institute for Communication, Information
Processing, and Ergonomics
Neuenahrer Straße 20
D-53343 Wachtberg-Werthhoven
49-228-9435-376

buehler@fgan.de

Heinz Faßbender

Research Establishment for Applied Sciences
Research Institute for Communication, Information
Processing, and Ergonomics
Neuenahrer Straße 20
D-53343 Wachtberg-Werthhoven
49-228-9435-640

fassbender@fgan.de

1. ABSTRACT

In this paper, we describe our experiences in applying CORBA and Java while redesigning an experimental command and control information system that is programmed in Ada 83. Our goal was to allow the user to use the system from any platform with minimal requirements to system resources.

1.1 Keywords

Distributed system, CORBA, Java, use of several languages, legacy system

2. INTRODUCTION

Using our institute's ongoing project as an example, we want to demonstrate how an experimental command and control (C2) information system that is totally platform dependent can be made platform independent. The original software was platform dependent because it was implemented on a Sun workstation under Solaris. It was redesigned by using

- internet technologies such as web browsers and Java applets [4], and
- middleware technologies as CORBA [3]

in such a way that it became totally platform independent. Hence, it can be used on every computer with a World Wide Web browser that can run Java applets and that is

connected to the Internet. In the following, we describe our ideas concerning and experiences with the project. Furthermore, we developed a general strategy with which to create an arbitrary platform independent C2 information system. More generally, a legacy system can be redesigned in such a way that it can be completely platform independent, and require only a very small amount of system resources.

3. THE EXPERIMENTAL DISTRIBUTED C2 INFORMATION SYSTEM EIGER

In our institute, an experimental distributed C2 information system called EIGER has been developed that can support the work of headquarters of the German Army. Nearly 90% of the system has been coded in Ada 83. The other 10% has been coded in C for communication between the system's components. For our current work, EIGER has been changed so far that it can be compiled by an Ada 95 [1] compiler that ensures that more elegant and more time efficient mechanisms; e.g., object oriented features, can be integrated into the system. We use the Object Ada system from Aonix as the development environment. The current version runs on Sun workstations with Solaris. Since the kernel of the system can also be compiled and linked by the Object Ada system on Windows NT, we have realized the first step of porting 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, given that usage of Windows systems is rapidly spreading.

That is the main reason why the current system has been redesigned such that:

- the system can be used totally independent of platform, and
- we can use the Internet for using the system such that only a minimum of system resources is required.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. SIGAda'99 10/99 Redondo Beach, CA, USA © 1999 ACM 1-58113-127-5/99/0010...\$5.00

Before we are able to describe the redesigning mechanisms, we have to discuss some basic aspects of the current structure of EIGER. EIGER consists of a finite amount of sub systems the communication of which is realized by a communication system (cf. Figure 1).

concurrent processes; i.e., the controller would have to wait until the complete result is computed by the DBS. Furthermore, the controller is connected to an *X.400 message handling system* MHS (in the lower tier). This connection is realized by a particular process system, too,

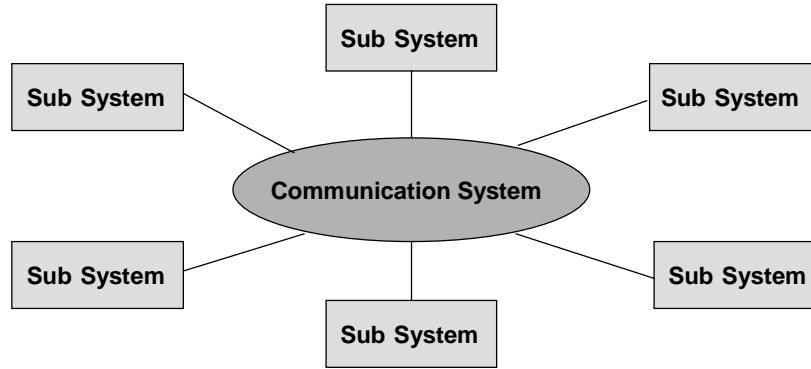


Figure 1. EIGER's topology

The implementation of the communication system is not important for understanding the mechanics of the redesign. For this purpose, we only have to understand the architecture of a sub system (cf. Figure 2).

where the reasons for the additional process system are identical to the one above. The controller is also connected to at least one *graphical user interface* GUI (in the upper tier) that is produced by OSF/Motif. After a successful

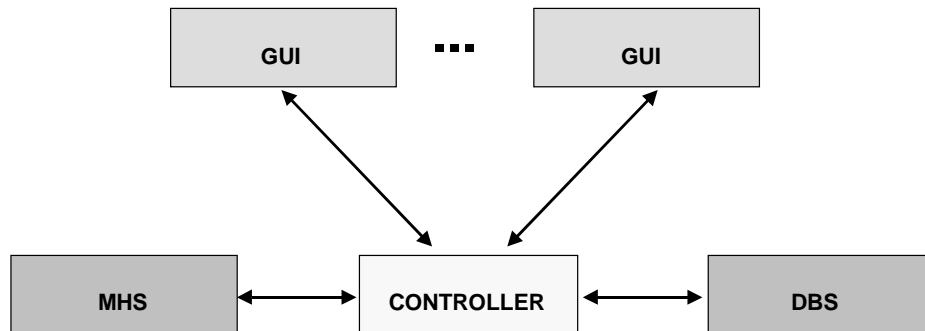


Figure 2. Structure of a sub system

A sub system has been designed as a three-tier process system that proved to be a starting point well suited for the redesigning process, because the number of program changes is minimized by the distribution and strong modularization of the code. Furthermore, the structure of the code could be kept. The central unit; i.e., the middle tier of a sub system, is the *controller* that controls the computations and communications of the other components of a sub system and its communications to other sub systems. This component is completely implemented in Ada 83. The controller is connected to a *relational data base system* DBS (in the lower tier where we use an ORACLE7 data base) that stores the data of the organization corresponding to the underlying data model. The connection between the controller and the DBS is realized by a process system, because a connection by embedded SQL is invaluable, since this solution would not allow

login, a session is started on the user's screen, where a session consists of:

- one basket that includes *notes* of system messages.
- at least one basket that includes *mails*, i.e. these baskets offer the basic services of a mail system.
- at least one *entry* basket, by which the user receives some jobs he has to do.
- and at least one *working* basket that includes the jobs that are currently worked on.

We hope that the short impression of EIGER's structure is sufficient for understanding the concepts for redesigning sub systems that will be discussed in the following sections. A much more detailed introduction to the structure of EIGER is presented in [2].

4. CONCEPTS FOR EIGER'S REDESIGN

We integrate the following two concepts into the implementation of a sub system 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 in every web browser that is able to run Java applets.
- The communication between the controller and the GUIs shall be realized by a CORBA connection. This also supports the platform independent use, because CORBA, in contrast to DCOM [5], is a system-independent standard that 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 sub system that is illustrated in Figure 3. Note that we omit the connections to DBS and MHS, because they have not changed.

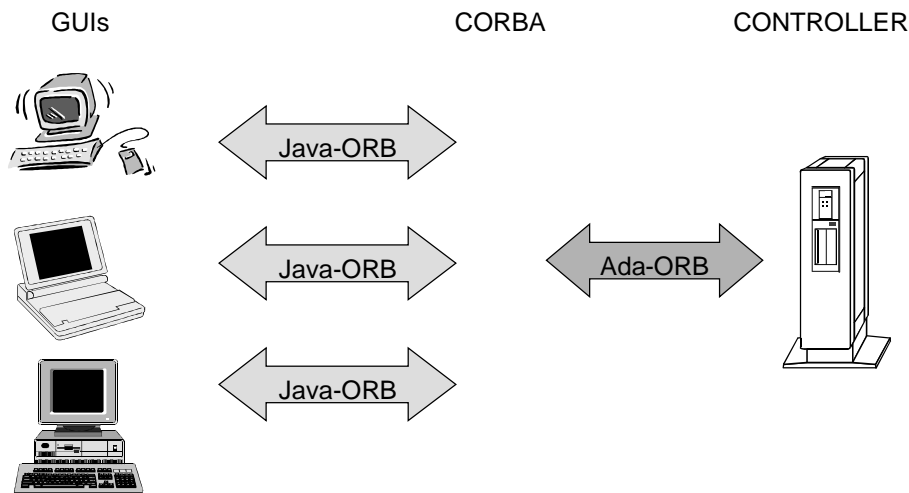


Figure 3. Redesign of a sub system

5. PROBLEMS AND SOLUTIONS

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

5.1 Reimplementation of GUIs as Java Applets

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

This problem is completely solved by a new implementation of GUIs as Java applets under the environment JBuilder2 from Inprise. Figure 4 illustrates the first image on the screen, namely the login screen, after

starting the connection from the browser to the controller by loading the HTML-page. The user has to enter his user-id, his password, and the system kind. After that, the user has to hit the Apply-Button. This starts the first transaction between the GUI and the controller. In the following subsection, this transaction will serve as an example for explaining the realization of the CORBA connection.

5.2 Realization of the CORBA Connection

In the existing version of EIGER, GUIs and the controller are connected by a TCP/IP connection with 47 defined protocol elements that are sent from the controller to the GUI or vice versa. For example, we illustrate the behavior of the existing system during the login phase. If the user hits the Apply-Button in the login screen of the existing system then the protocol element *CONTROLLER_OP_CONN_RQ* and three parameters for the user-id, the password, and the system kind are sent from GUI to the controller. We are referring not the screen as shown in Figure 4, but the screen that is produced by OSF/Motif and that is defined as being analogous to the Java applet screen. After the user hits the

Apply-Button, the GUI waits until it receives the protocol element *CONTROLLER_OP_CONN_RS* as response. Then a session, as described in Section 3, is built up on the GUI by a sequence of protocol elements that are sent from the controller to the GUI. In the new implementation of EIGER, the previous behavior is realized by a function that is defined in the *Interface Definition Language* (IDL) of the CORBA standard [3] 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 Wb_List,
out IDL_Entry_Baskets_Lists Eb_List,
out IDL_Mail_Baskets_Lists Mb_List,
out IDL_Note_Baskets Notebasket);

```

- **GUI_TO_CONTROLLER**
This interface specifies 18 services where the controller is the server (implementation in Ada) and the GUI is the client (call in Java); e.g., it includes the function `Login_Computation`.

Figure 4. The login screen

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 that 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 an indicator of the session. The four out-parameters of the function `Login_Computation` indicate the contents of the baskets of the session that are described in Section 3. 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 the 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 does not have 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 receives asynchronously by protocol elements in the existing version of EIGER. More generally, in the definition of the IDL, we have defined the following two interfaces:

- **CONTROLLER_TO_GUI**
This interface specifies 30 services where the GUI is the server (implementation in Java) and the controller is the client (call in Ada); e.g., it includes a function `Delete_Note` that 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.

5.3 Connection of the GUI by a Java-ORB

As Java-ORB, we use the Visibroker from Inprise that is integrated into the JBuilder2 Client and Server version. After solving some problems, Java applets are connected well.

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 applied 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 [3]. The calls of services from the controller by a GUI are realized as threads [4] that are concurrently computed because of reasons of efficiency.

5.4 Connection of the Controller by an Ada-ORB

We use the ORBADA system from Top GraphX as Ada-ORB that works well after some discussions with the hotline and some corrections from Top GraphX.

The connection of the Ada-ORB to the controller is tricky, and is realized as follows: Corresponding to the implementation of the GUI, an object CON-object of the type GUI_TO_CONTROLLER (cf. Subsection 5.2) is constructed. But, in opposite to GUI-object and since the controller serves as main server, this object is explicitly bound to the naming service of the ORB. After that, the CORBA Main_Loop is started. It is a procedure that is waiting for demands to services and, after receiving demands, calls the implementations of the services and

Delete Message

If the complete information needed, is stored in the service control block, then the message is deleted.

Evaluate Service Control Block

The service that the message demands is evaluated and its results are sent to the calling entity of the sub system by help of the corresponding service control block.

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

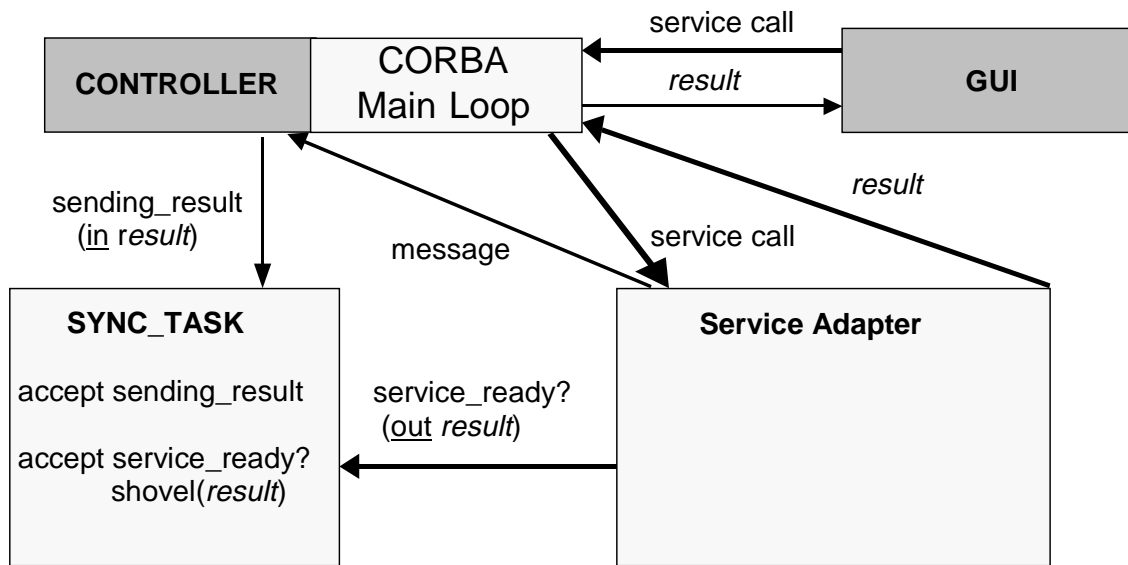


Figure 5. Management of parallel processes

returns the results of the services to the demanding object. Since EIGER is a distributed system, the services of the CON-object have to be managed as parallel processes. 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 that consists of the following steps:

Receive Message

where the messages can be received either from DBS, MHS, or from one of the GUIs.

Create Service Control Block

A service control block that contains the information of parameters and message code, is constructed for every received message.

In particular, every service of 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 that is evaluated by the process system, for every service, we need an adapter that transforms the service call from the form that is called by the CORBA Main_Loop into the process that can be evaluated by the process system.

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

```

send message to CONTROLLER

SYNC_TASK.service_ready?(out result)

send result to CORBA Main Loop
  
```

For every call of a service adapter, a synchronization task SYNC_TASK is started that has two entries that realize the rendezvous principle. After the service adapter has been started by the CORBA Main_Loop, it sends a message with the parameters and a pointer to SYNC_TASK to the controller. It then enters the entry service_ready?, and thus offers containers for the result. Since service_ready? is the second entry, the service adapter has to wait until the controller enters the first entry sending_result. The controller only does this after it has finished the evaluation of the message and sent the result to SYNC_TASK. Then, the result is shoveled into the containers and thereby transported to the service adapter that, in turn, sends the result to the CORBA Main_Loop.

5.5 Communication between Java-ORB and Ada_ORB

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

6. 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 that is coded in Ada 83 can be restructured by applying Ada 95, Java, and CORBA in such a way that it becomes totally platform independent. However, on the one hand, a lot of problems have arisen because the market for Ada products is not as robust as the markets for other comparable programming languages. On the other hand, we can summarize that the concepts of Ada 95 are well suited for implementing a distributed C2 information system; e.g., the management of parallel processes can be realized extremely well by Ada 95.

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, we will realize the communication system between different subsystems by CORBA, too. Furthermore, in order to port the system to Windows NT, we have to realize a new connection between the controller and the DBS and MHS. The connection to MHS should also be realized by CORBA, whereas we want to apply ODBC for realizing the connection between the controller and the DBS. As a further important research topic, we will redesign the controller by using new features of Ada 95 (in particular, object oriented features) for increasing the system's performance.

7. ACKNOWLEDGMENTS

We thank Ann S. Brandon for editing a previous version of our paper.

8. REFERENCES

- [1] Barnes, J. Programming in Ada 95. Addison-Wesley, 1995.
- [2] Bühler, G. Einsatz von Ada im Experimentellen Führungsinformations-system EIGER. in Workshop „Entwicklung von Software-Systemen mit Ada“, Bremen, Germany, Ada Germany, 1998.
- [3] <http://www.omg.org>.
- [4] <http://www.javasoft.com>.
- [5] Sessions, R. COM and DCOM: Microsoft's Vision of Distributed Objects. Wiley, 1998.