SEEDS : Airport Management Database System

Tomáš Hrúz[1,2], Martin Bečka[3], and Antonello Pasquarelli[4]

[1] SolidNet, Ltd., Slovakia, www.sdxnet.com, tomas@sdxnet.com
[2] Department of Adaptive Systems, UTIA, Academy of Sciences of the Czech Republic, Prague ‡
[3] Mathematical Institute, Slovak Academy of Sciences, P.O.Box 56, 840 00 Bratislava, Slovakia, becka@ifi.savba.sk
[4] Alenia Marconi Systems, Via Tiburtina km 12.4, I-00131 Rome, apasquarelli@aleniasystems.finmeccanica.it

Abstract. The article describes an airport database management system, which is a part of large simulation environment developed under the ESPRIT project SEEDS. Airport management database is a distributed computing system written entirely in Java and connected to the rest of the SEEDS system through CORBA interface. The majority of the SEEDS modules is written in C++ and communicates through CORBA. The airport management database consists of an application server, three different clients and a CORBA connection module.

1 Introduction

This article describes an Airport Management Database (AMDB) system which is a part of larger simulation environment SEEDS [1, 6].

SEEDS is a distributed simulation environment composed of powerful workstations connected in local network, suitable to evaluate advanced surface movement guidance and control systems, to validate new international standards and to train operators.

The aim of AMDB software module is to describe various external aspects of the core airport simulation model like meteorological situation and its changes, flight data list (FDL) and its changes, Initial Climbing Procedures (ICP), Instrument Approach Charts (IAC), Standard Instrumental Departure (SID) and Standart Approach Route (STAR) description and visual information. To emphasize the external aspect of AMDB with respect to the core simulator we sometimes use the term external world model instead of AMDB.

The AMDB module is designed to contain a relational database subsystem to handle large data sets and a three level architecture (SQL server [7], application server and clients) to achieve high level of flexibility. Another aspect of AMDB module design is a wide area network operation emphasis, which is leading to the architecture centered around the Java system.

The core of the SEEDS simulator is written in C++ and uses the CORBA standard to communicate between different system modules. The AMDB module

‡ Grant GACR 102/1999/1564

is entirely written in Java. This situation provides an excellent occasion to test the cooperation between Java and CORBA based C++ subsystems.

In particular, the application server (APS) is written in Java to see whether the speed of recent Java virtual machine (JVM) implementations is able to cope with the tasks arising in such complex applications.

As far as SEEDS together with AMDB module contains various hardware and software platforms we have decided to use only open software tools for which we had the source code. This approach has two reasons:

1. In very complex and multi-platform systems we consider the absolute control over the development process provided by source code as inevitable for any high quality software.

2. Open software systems usually contain a less powerful visual development tools and we were interested in knowing whether these high level visual features are really necessary for development of very complex systems.

The article is organized as follows: in Sec. 2 we describe architecture, design specification and basic features of the AMDB system. Then we conclude with some notes and experiences drawn from this work.

2 Airport Management Database System

The information stored in the airport management database can be structured to groups according to the importance and complexity. In the SEEDS airport model architecture [1] the following navigation data have been considered for inclusion to the external world model. (1) Meteo information, (2) FDL and (3) SID, STAR, ICP and IAC charts. These information groups are implemented in the AMDB module through relational database model, APS and clients. The relational database stores the information about the navigation data in the structured way and APS is responsible for all non-relational model aspects. There are three main clients in the system.

1. Meteo client responsible for meteo information rendering and changes.
2. Flight data list client responsible for the FDL processing.
3. Image viewer client responsible for SID, STAR, ICP, IAC charts previewing and processing.

The synchronization, authentication and all other aspects of AMDB allow arbitrary number of clients (limited only by system resources) of each type running at the same time.

2.1 Architecture

Today's computing environment enforces definitively a server/client architecture. The main reasons are flexibility, scalability and reusability. Moreover, the AMDB module architecture is Java (network computing) oriented and has a SQL relational database in the background to meet the design goals. The client

Java classes communicate with the SQL database engine through JDBC layer and driver [8].

However, an architecture conforming to the above mentioned points can be designed in various ways. Let us consider first the architecture in Fig. 1 but without the third quadrant containing AS. With respect to the relation between SQL database and clients this can be called two level architecture because there are two levels: 1.clients and 2.SQL servers. The advantage of such architecture is simplicity, a disadvantage is limited modeling strength and flexibility.

With modeling strength and flexibility we mean the following concept. Any software system can be considered as a model of some process. If we consider a two level architecture as above, we can efficiently construct a model for a process which can be well represented in terms of relational database. However, if the modeled process does not fit well in this class we have to add software modules which represent its non-relational structures. If a two level architecture is used these modules must be split between the SQL server (as embedded procedures) and the clients. Such splitting can be very inefficient from the software design point of view as well as from the final system efficiency point of view. As a solution to the above problem a concept of three level architecture can be used. The three level architecture has: 1. clients, 2. application servers, 3. SQL servers.

Following the ideas above, all non-relational aspects of the model are located in the APS. It is also responsible for managing and forwarding the database queries of clients to the SQL server. In the SEEDS project an absolute necessity of APS comes from the fact that any change of the database state must be transmitted to the rest of the system in form of events, so that all modules can change their behavior according to the new state of the external world. The final three level architecture used for AMDB is shown in Fig. 1 where APS is designed in Java and is called from the clients through Java Remote Method Invocation (RMI). Another advantage of this architecture is that design of such system contains also a task of application protocol development. Moreover, in the case of very heavy server computation related to some non-relational modeling aspects, the server class can call C or C++ libraries through JNI (Java Native Interface).

The airport management database architecture and its relation to the processing of events in SEEDS is illustrated in Fig. 2 on an example of the meteo data processing subsystem. AMDB calls a meteo event server written in C/C++ with a CORBA interface, which is a part of core SEEDS system. The SEEDS modules register to the meteo event server for a particular class of events. The major events with respect to the external world model are changes in the model database state.

For example an operator wants to change the visibility at some airport object. He starts a master mode of client module, which allows him to change the database state. The request is forwarded through RMI to AS, which generates a SQL operation to the SQL server and a notification event to the meteo event server. APS notifies also all registered external world clients about the change so that all other operators see the change on the external world client screen.

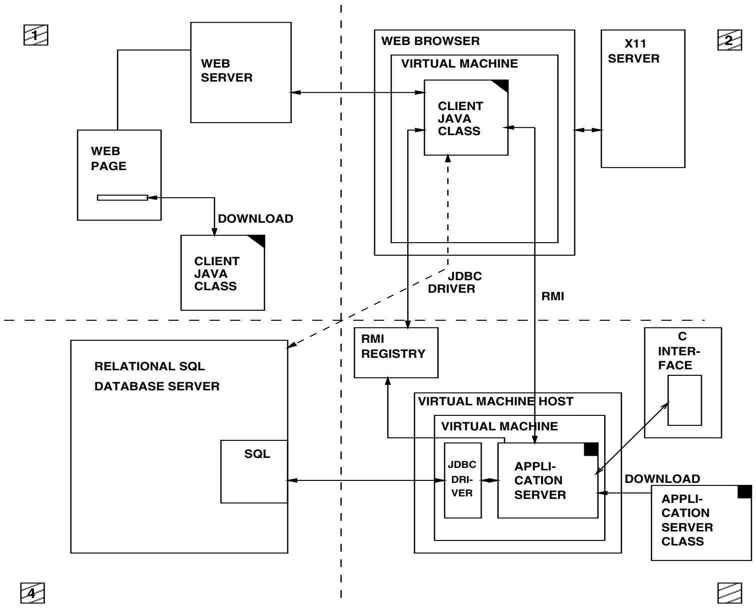

Fig. 1. The Java client/server 3 level architecture for Airport Management Database

Because the event server has been notified all SEEDS modules registered for this event class will obtain a notification about the event. Then they can ask through a CORBA interface of the application server about the new values of AMDB and they can change the behavior according the new data. For example the scenario generation module can change the picture on appropriate operator and pilot screens etc.

To summarize the architecture, the AMDB system consists of the following subsystems: **(a) SQL server.** The SQL server used is PostgreSQL [7], which is running on most UNIX platforms. **(b) SQL relational database.** The AMDB use a database maintained by SQL server. The relational database consists in a set of tables, which provide a relational model of AMDB subsystems like airports, meteo data, navigation procedures etc. **(c) Web server and web pages.** The client applets are stored and transmitted through the Apache web server. **(d) The clients.** The code of clients is stored in signed archives on the web server. Clients are downloaded from the web server to the browsers where they are run on browser virtual machine. **(e) The application server.** APS is running as a Java application under JVM.

In the prototype configuration the application server, the web server and the SQL server are running on Alpha station with Digital 64bit UNIX.

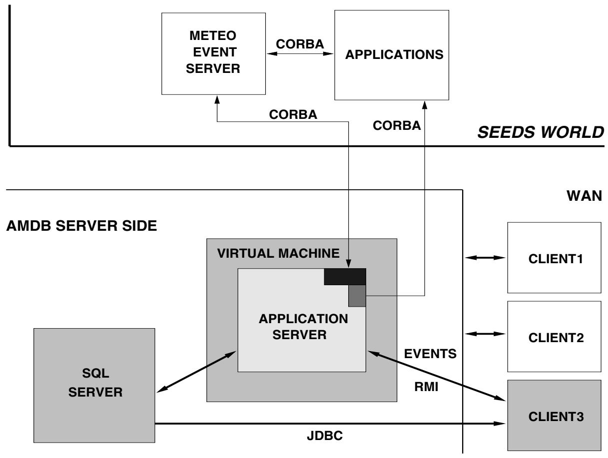

Fig. 2. External World Model Architecture and Communication. The communication pattern between application server, SQL server and clients in the AMDB module is shown as the emphasized triangle structure

2.2 Data transmission rules

The SEEDS system is highly heterogeneous and contains more platforms (e.g. Microsoft NT on Intel, UNIX on Silicon Graphics, UNIX on Alpha, Java etc.) therefore it is necessary to establish communication rules which allow robust and efficient communication between the modules and platforms. For the AMDB module design we have identified the following rules:

(1) All aspects of the system, which can be modeled by relational database are modeled by the database structure and are centered in SQL server.

(2) If a client is interested in data which are stored in the database and do not require non-relational processing they read the data directly from the SQL server as is illustrated in Fig. 2 with an emphasized triangle pattern.

(3) The only agent allowed to write the data to the SQL database is the application server.

(4) All non-relational modeling and processing is centered in application server. In the case of the AMDB module this means for example notification and data transmission between the AMDB module and the rest of SEEDS.

(5) Because of the heterogeneous character of SEEDS it is preferable to convert most of the data which will be transmitted over the network to integer formats. This is an obligatory rule for primary key data. It means that all information entities are coded as integers and only these codes are sent over the network. The clients are responsible for transformation between the integer representation and other forms of representation like strings etc. For this purpose the clients call directly the SQL server.

2.3 Communication with SQL server

Following our open software strategy we have used the PostgreSQL [7] database system. The resulting experience is very positive. The system is robust, stable and efficient. Moreover, it defines a very rich set of database types. On the other hand, the main disadvantage of this system, when compared with commercial systems like Oracle SQL servers is a certain lack of preprocessing, postprocessing and visual tools. However, the minimal price (only the maintenance cost) and the open character of this system have far outweighted its disadvantages especially in a research and development project as is SEEDS.

The clients and APS communicate with the SQL server using JDBC layer defined by Java system [8] which in turn calls JDBC driver which is provided by SQL server manufacturer, in our case by PostgreSQL system.

2.4 Security model

The AMDB module is designed as a network computing structure therefore the security aspects are very important.

The PostgreSQL server is well equipped with security options ranging from a simple password authentication to KERBEROS and data channels encryption. To choose the appropriate option is a matter of configuration depending on the actual AMDB usage.

APS is written in Java as a Java application. One of the specification and design problems here is related to the fact that Java version 1.1, which is still not widely accepted by the browser industry does not contain interface to CORBA. This is provided in a new generation of Java, version 1.2. Therefore, we have adopted the following security design rules illustrated in Fig. 3.

- The application server is written as a Java application.
- The server code is conformant to the Java 1.1 standard.
- The server is compiled and running in the Java 1.2 system. The security restrictions are defined with a special security manager written for the AMDB module.
- The clients are written as Java applets.
- The code of clients is conformant to the Java 1.1 standard.
- The clients are compiled and running in the Java 1.1 system. The security restrictions are handled according to the Java 1.1 system. This means that we generate signed archives stored on a web server.

2.5 Application Server and Clients

APS is responsible for all non-relational modeling aspects. After successful reconstruction of the initial state an object for the SQL communication is constructed, which opens a channel to SQL database server used for data processing. This SQL channel is used to read and write the data from the database during the

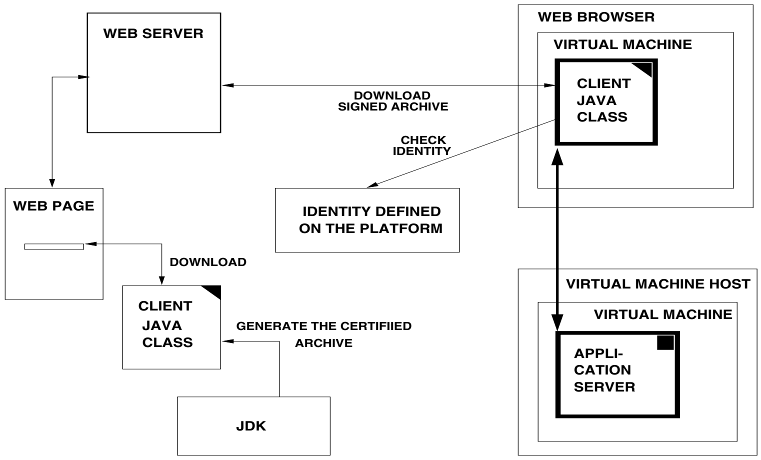

Fig. 3. Illustration of the client security model

whole life period of APS. All other agents are allowed (even sometimes obliged) to read directly from SQL server but not to write.

Clients connect to APS through well defined "channels" which are defined as two sets of remote procedures. One set for each direction. RMI connection of a client is initialized through a well known remote object. The reference to the remote object is obtained through the RMI naming services. Then a private channel is constructed which contains private remote references to the procedures above. APS maintains a list of channels which is periodically checked for dead clients. The channels for dead clients are deleted so that there is no resource deficit during a long server runs. The channel maintenance subsystem use a watch dog mechanism where in regular intervals each client must call the following time synchronization procedure:

Timestamp watchDog(Timestamp clientSimulationTime)

through which the client sends its value of the simulation time and obtains back the server value of the simulation time. If they differ in larger value, the client is obliged to synchronize with APS. At the same time, when APS obtains such call it updates the watch dog structure. This mechanism provides a means how to detect dead channels as well as synchronize simulation time between the clients and APS.

We have implemented three main AMDB clients, which are connected to the database through JDBC. During the client run it is necessary to load information from the database concerning the user identity tests, simulations, airports, aircraft types, airport objects and its sites, predefined meteo values, airport map images, etc. Some of these data are loaded once and cashed in the application, other ones are dynamically loaded when they are needed.

We have introduced a concept of two edit modes of the client, a slave mode and a master mode. In the slave mode a user can only view all information the client offers. In the master mode the user can change the data. The slave mode is default, for switching to the master mode the user's identification and password is required. User is notified about the number of masters working on the same configuration, however users can concurently write to the database.

3 Conclusions

Concerning the software engineering point of view there are three main conclusions (experiences) from the SEEDS project. One concerns CORBA and the other two Java and open software systems.

CORBA together with Internet proved to be extremely efficient in development and integration of large systems which contain different platforms and geographically distant development teams. The introduction of new platform (Java) into the project has stressed the necessity to stick very closely to CORBA standards, otherwise the reusability of the software modules can be decreased.

We used Java extensively in AMDB design to test its reliability and efficiency. Concerning its complexity, Java is surprisingly matured even for industrial applications. We where running the application server on different platforms (Sun, Alpha) with very good results concerning speed and reliability.

The results with open software tools are very positive and the possibility to inspect the source code in some situations has far outweighed the disadvantage of not having a strong visual development tools. This do not indicate that they are not needed in current software development industry but it might indicate that their role is little bit overestimated and that the complexity level on which they are really inevitable lies higher as is usually considered.

In some situations it was necessary to change or correct the source code of systems which we used and this was far more efficient solution as to circumvent the problems with other means.

References

1. Bottalico, S., de Stefani, F., Ludwig, T., Rackl, G. : SEEDS – Simulation Environment for the Evaluation of Distributed Traffic Control Systems. In: Lengauer, Ch., Griebl, M., Gorlatch (eds.): Euro-Par '97, Parallel Processing, LNCS 1300, Springer-Verlag, Berlin Heidelberg New York (1997), 1357–1362
2. Bottalico, S. : SEEDS (Simulation Environment for the Evaluation of Distributed Traffic Control Systems). A Simulation Prototype of A-SMGCS. In: Proceedings of the International Symposium on Advanced Surface Movement Guidance and Control System, Stuttgart, Germany, 21-24 June 1999
3. Hrúz, T., Bečka, M. : Airport Management Database. SEEDS Internal Documentation SEEDS/MISAS-WP4-D4.6A-REV1-SW, 1999
4. Lewis, G., Barber, S., Siegel,E. : Programming with Java IDL. John Wiley & Sons, 1998
5. Flanagan, D. : Java in a Nutshell. O'Reilly, 1997
6. SEEDS official home page, http://www.lti.alenia.it/EP/seeds.html
7. http://www.postgresql.org, 1998–1999
8. http://www.javasoft.com, 1998–1999