An Experimental Command and Control Information System based on Enterprise Java Bean Technology

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, our ideas, and the experiences we have made in an ongoing project of our institute by developing an architecture for an experimental command and control information system and its implementation. For the implementation new technologies as *Enterprise Java Bean technology* [EJB] and the *Extended Markup Language* [XML] are used, because they are the most promising candidates for implementing a modern extremely flexible and extendable C2 information system.

1. Introduction

In military operations organizational and operational requirements to C2 information systems change more and more frequently. For modern C2 information systems this fact implies that the basic architectures and the applied technologies for implementing such systems are well-suited for an extremely simple and cost-efficient extension and replacing of the systems' functionality. In particular, the technologies have to realize an extension of the functionality online during the operations.

Furthermore, since international forces have to operate in allied operations, the C2 information systems have to be interoperable to other systems of international forces. That is the most important reason, why the application of standards becomes more and more important in the development of C2 information systems.

On the other hand, the scientific and commercial communities involved in system development, recognized that the increased complexity and flexibility of information systems require the definition of standards for information exchange and software development technologies.

The most promising technologies and standards in the context of flexible and extendable C2 information systems are the *Enterprise Java Bean (EJB) technology* [EJB] and the *Extended Markup Language (XML)* [XML]. That is why we decided to develop an experimental C2 information system called *INFIS*, on the basis of EJB and XML technologies. The system INFIS

serves as a testbed for experiments in the context of C2 information systems, in particular in interoperability tests with other nations [ATCCIS] & [MIP].

The paper is organized as follows. In Section 2 we start with some fundamental technical requirements for modern C2 information systems. From those requirements we derive the decision for the applied technologies in Section 3. After hat we design the global architecture of INFIS and explain its different components in Section 4. Its most important component, i.e. the architecture of the application logic is designed in Section 5. Finally, we give some conclusions and ideas for future work in Section 6.

2. Fundamental Technical Requirements for a Modern C2 Information System

Since organizational and operational requirements to C2 information systems change more and more frequently, we develop the system INFIS in such a way that new functionality can simply and cost-efficiently be integrated. In particular, since the organizational and operational requirements are not fixed before the operations start, we are only able to list some technical requirements to C2 information systems that can be derived from the required flexible behavior of the system:

2.1 *Flexibility*

TR1: The system has to be online configurable, because the hardware and the communication mechanisms are not fixed before an operation starts.
TR2: New applications have to be online integrated into the system, because new organizational and operational requirements may occur during the operation and the system has to be able to fulfill these requirements.

2.2 *Interoperability and Multiuser Access*

TR3: The system has to be interoperable to systems of other units of the same country, as well as of other countries.
TR4: The system has to offer multiuser access, i.e. it has to compute a lot of tasks of more than one user in parallel.
TR5: The system has to be online scalable.
TR6: The distribution of the system has to be transparent, i.e. the user does not need to know where a demanded service is computed.

2.3 *Platform Independence and Mobility*

TR7: The most possible platform independence, i.e. the system can be executed on every common platform (hardware & operating system).
TR8: Platform independent access to the system by using web browsers. By this, the mobility of the system is supported and extended.

2.4 *Workflow and Communication*

TR9: The graphical user interface GUI allows the user to perform multiple tasks asynchronously. The GUI must not be blocked by such a task, i.e. the user has to be able to initiate new tasks during the computation of another task, as well as to fill in templates.
TR10: Workflows between different users have to be supported by the system via the GUI.
TR11: The user has to be notified by the system asynchronously (e.g., exceptions or events like changes in the data base or time stamps).

The technical requirements **TR1** - **TR11** seem, in our opinion, to be the most important ones. We mention that some of them are explicitly listed, because they cause problems in the particular context of EJB and XML technologies. The reason why we still use these technologies will be explained in following section.

3. **Why do we use EJB and XML Technologies?**

The motivation for using EJB and XML technologies will be explained with respect to the technical requirements in Section 2. The Java technology offers the most possible platform independence (**TR7**), because the server, as well as the clients are absolutely platform independent. The server will be implemented by EJBs that are processed by an platform independent application server. The GUI will be implemented as a Java Applet. Hence, also the platform independent access by a web browser is realized (**TR8**). This also implies that only a browser and the connection to the Inter/Intranet is required for the client system. Thus, the client systems are online configurable (**TR1**). Furthermore, the EJB technology allows a very simple online integration of new applications (**TR2**).

The interoperability to other systems (**TR3**) is guaranteed by organizing the information with respect to the ATCCIS data model and implementing the ATCCIS replication mechanism [ATCCIS]. On the other hand, standards like EJB and XML support an effective information exchange to other systems. The multiuser access (**TR4**), the online scalability (**TR5**), and the transparent distribution (**TR6**) of the system are automatically managed by the *application servers*. By the definition of the EJB-standard, application servers are the main components of the systems that are implemented by EJB technology. In particular, application servers take off a lot of tasks from the developer for solving problems close to the system level.

These are strong reasons that lead to the decision that we apply the considered technologies in designing the architecture of the system and in its implementation.

But beside the advantages of these technologies, the attentive reader may have recognized that the technical requirements **TR9**, **TR10**, and **TR11** are not discussed in the above motivation. That is absolutely right. We omit these technical requirements, because exactly these requirements can not be fulfilled by pure EJB technology. Nevertheless, in the following sections we will discuss how we integrate other mechanisms from the Java and XML technologies into the system

architecture, in such a way that also these technical requirements are fulfilled by INFIS. For this purpose, we start with the design of INFIS' global architecture in the following section.

4. INFIS' Global Architecture

The global architecture of INFIS is illustrated in Figure 1.

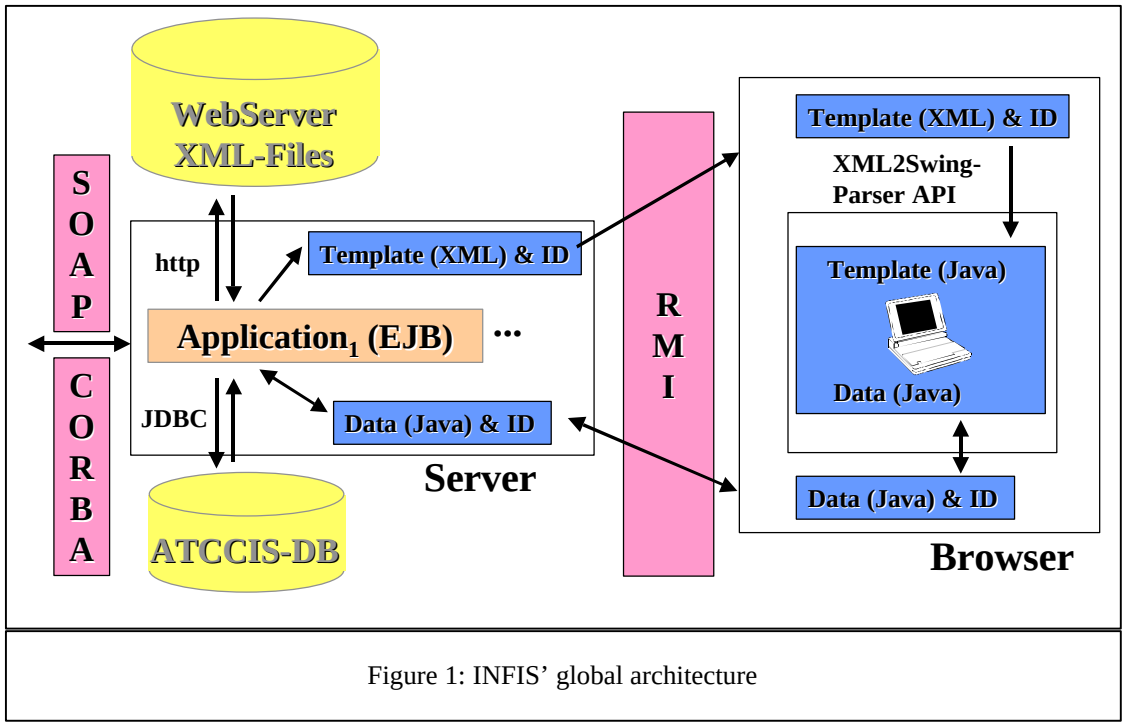

Figure 1: INFIS' global architecture

In the explanation of the design of INFIS' global architecture we point out the number of the technical requirement that leads to the discussed design decision.

INFIS' architecture is designed as a three tier client/server architecture. The *client* that implements the GUI, is realized as a Java applet that can be executed in any browser (**TR7**). The information displayed on the GUI is separated into *templates* and *data*. The correspondence between data and templates (for example, the coordinates of the current position of a tank are components of a template that displays the complete information of the tank) is managed by the server via an ID. Using this ID the client puts data to the right position in the display of the corresponding template. Vice versa, if a user changes data in a template, then the meaning of the new data is identified and it can be written by the server to the right place in the data base.

Templates are described in XML-notation. On the client, these XML-files are online transformed to Swing classes by the *XML2Swing-Parser API*. By this implementation new templates can be online added to the system without changing the client. This behavior is necessary to fulfill requirements **TR1** and **TR2**.

In our first approach we designed the client as a pure EJB-client, but, as mentioned before, pure EJB technology is not able to fulfill the technical requirements **TR9**, **TR10**, and **TR11**. In particular, asynchronous messages and call-backs from the server to the client are not sufficiently supported by pure EJB technology. That is the reason why we decide to connect the client to the server by an *RMI-connection*.

The *server* implements the application logic, the connection to other systems, the access to the data layer, and the connection to the client. It is executed on the *WebLogic 6.1* application server from *BEA* [BEA]. Such an application server manages all tasks of the application logic that are close to the system level. Thus, the developer is only confronted with the implementation of the application logic, rather than with system specific tasks. The architecture of the application logic is explicitly designed in Section 5. At this point we only mention that the application of the EJB technology fulfills the technical requirements **TR2**, **TR4**, **TR5**, and **TR6**. In particular, the server implements the *ATCCIS replication mechanism* [ATCCIS] which fulfills the technical requirement **TR3**. Furthermore, outside (Web) services can be attached via *CORBA* [CORBA] and *SOAP* [SOAP] connections.

The *data tier* is separated into two data bases, where one data base is organized corresponding to the *ATCCIS data model* [ATCCIS]. This is also necessary for fulfilling the technical requirement **TR3**. For realizing a data base independent connection, the data base is connected to the server by *JDBC* [Java]. This supports the simple replacement of the data base system by another one. In our current version an ORACLE 8 data base system is used.

The other data base is a simple WebServer that stores the XML descriptions of templates. As mentioned before the XML descriptions define the structures of the templates which are displayed on the client. The WebServer is, as usual, connected to the server by an http connection. In one further experiment the behavior of an immediate connection between the WebServer and the client will be checked. In this case, the server sends the URL of the XML description to the client instead of the XML description itself. Then the XML description is called immediately from the WebServer.

After this short description of the different components and connections in INFIS' global architecture we illustrate the architecture of INFIS' main component, namely the server that implements the application logic, more detailed in the following section.

5. Architecture of INFIS' Application Logic

The architecture of INFIS' application logic is illustrated in Figure 2.

The discussion of the detailed architecture is motivated by the technical requirements that can not be fulfilled by pure EJB technology. For this purpose, these requirements will be recalled whenever they are needed.

The architecture is structured as a multi layer architecture. The access to the database is implemented by *Entity Beans*. For this purpose, we have studied two different possibilities. On the

one hand, for every table in the data model exactly one bean is defined that represents the table. This leads to a huge amount of construction and destruction operations and a reduction of the performance of the system. On the other hand, we have implemented the entity beans in a hierarchical way. By this hierarchy every outer table of the data model is mapped to exactly one entity bean. Such an entity bean imports Java classes that represent those inner tables that are connected to the outer table corresponding to the entity bean. By this implementation only container for entity beans representing outer classes are processed which leads to an increasing performance of the system.

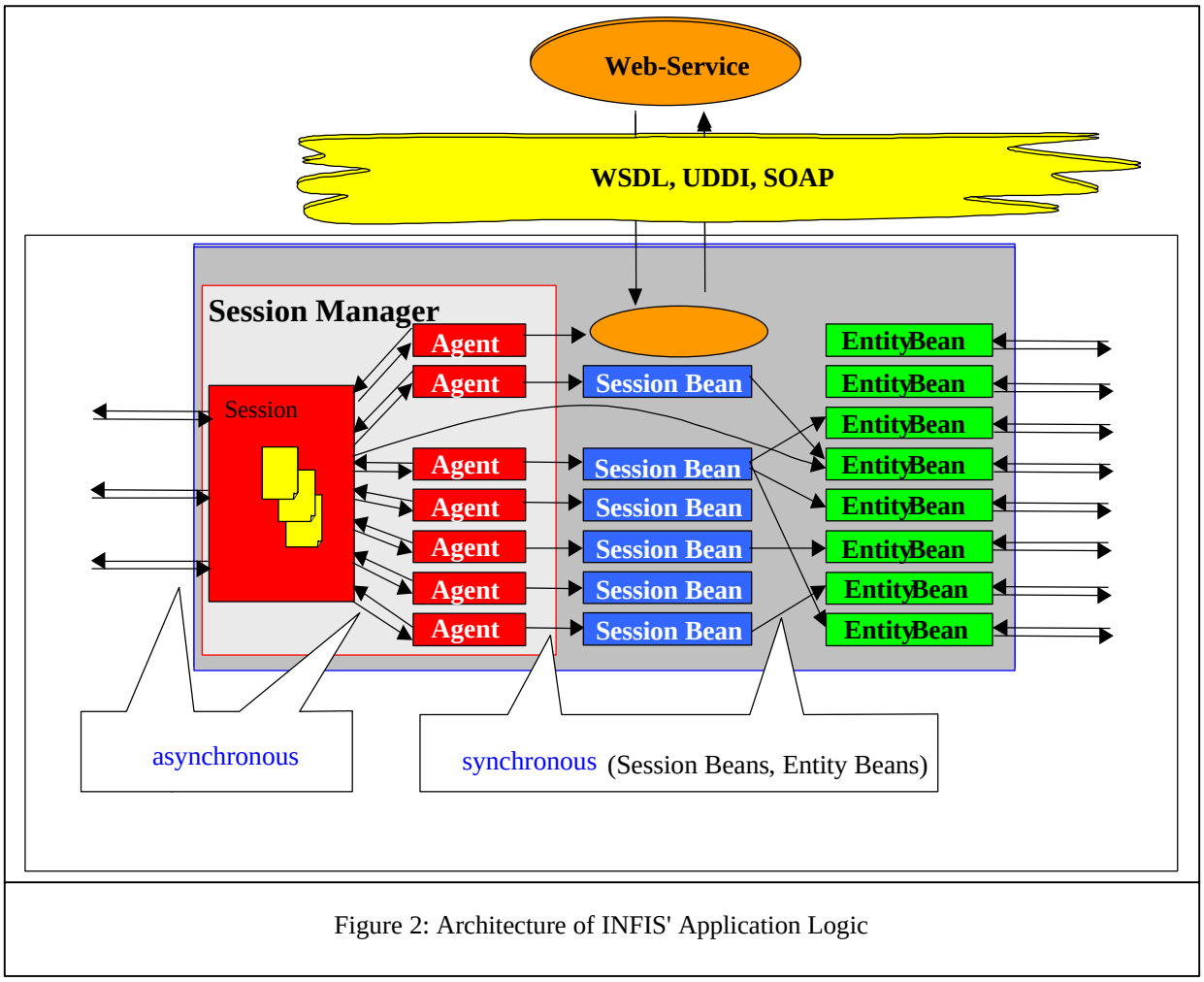

Figure 2: Architecture of INFIS' Application Logic

In the newest version of the EJB-standard, those two implementations are joined by using the new concept of local interfaces. Classes that should be processed in the same container, communicate via local interfaces. Thus, the first solution described above, is combined with the second one by defining an entity bean for every table where only those entity beans that correspond to outer tables, are processed by remote interfaces. Entity beans corresponding to inner tables are processed by local interfaces. That means, only for entity beans corresponding to outer tables containers are processed.

The layer of entity beans is encapsulated by possibly multiple layers of *session beans*. By those session beans the business processes are implemented. The positioning of a session bean into a layer depends on the complexity of the business process that is implemented by the session bean. A session bean that implements a simple process as e.g. *information_about_unit*, may be used by session beans that implement more complex processes as e.g. *display_current_situation*.

Furthermore, the session bean layers may implement the access to *Web-Services* via the standards *WSDL*, *UDDI*, and *SOAP*. The integration of Web-Services is still in an experimental state and shall not be discussed further.

The communications between the layers discussed above, are all synchronous communications, i.e., a client is waiting until the server sends its response. Because of the technical requirement **TR 9** where the user has to be allowed to perform multiple tasks asynchronously and the GUI has not to be blocked by such tasks, an asynchronous behavior of the system has to be simulated. This would not be possible by the former EJB-standards. But, the EJB-standard in version 2.0 introduces the so-called *message-driven beans* that implement exactly asynchronous behavior of beans.

Nevertheless, since also the technical requirements **TR10** (workflow management) and **TR11** (asynchronous message from the server to the client) have to be fulfilled, a separate management of workflows combined with the management of the asynchronous messages is implemented by the additional component called *Session Manager*. The Session Manager is implemented in Java. It manages multiple *Sessions* that communicate with the GUI via an RMI-connection (cf. Section 4).

Because of the technical requirement **TR11** a session has to be able to notify the user via the GUI asynchronously, for example about changes in the data base that result from replications from other data bases. This asynchronous behavior is implemented by using mechanisms as call-backs of the RMI-connection.

Furthermore, the management of workflows **TR10** is implemented by sessions. For this purpose, it must be ensured that the sender of a task does not have to wait until the receiver finishes this task. This is some additional example, where asynchronous behavior has to be implemented. The implementation is realized by components called *Agents* that are positioned between the sessions and the session beans, in the following way: If a session bean is processed by a session that should not be blocked by the computation of the session bean, then an Agent that is nothing else but a thread, is initialized. The Agent, in its turn, communicates with the session bean, but the session is not blocked by the computation of the session bean, because the Agent is implemented as a thread.

As mentioned before, the design of the additional component Session that is not implemented as EJB, but also in Java technology, fulfills the technical requirements **TR9**, **TR10**, and **TR11** that can not be fulfilled by using pure EJB technology. Hence, the described architecture is an example of enriching the pure EJB technology by other components of the Java technology as RMI and

threats for managing asynchronous and workflow behavior that is naturally required by C2 information systems.

6. Conclusions and Future Work

In the paper we have introduced an architecture for a very flexible and extendable C2 information system. Depending on the unstable organizational and operational requirements for military operations the most important technical requirement is the online integration of new applications into a C2 information system. Beside the Microsoft's .NET technology, the modern EJB technology is the most proper technology for fulfilling this requirement. We decide to take EJB technology as basic technology for the system development, because of its platform independent applicability and its strong connection to XML.

But, we find out that *pure EJB technology* is not sufficient to fulfill the requirements of a modern and flexible C2 information system. That is why we enriched the architecture by additional Java concepts as RMI and JDBC.

In our future work we will verify the introduced concepts by an experimental implementation of the architecture. Furthermore, we will study the integration of Web-Services into the system.

7. References

[ATCCIS] *Army Tactical Command and Control Information System* (permanent), SHAPE Policy & Requirements Division, Mons (Belgium)

[BEA] http://www.bea.com

[CORBA] http://www.omg.org/gettingstarted/corbafaq.htm

[EJB] http://java.sun.com/products/ejb/

[Java] http://java.sun.com/

[JDBC] http://java.sun.com/products/jdbc/

[MIP] http://www.dnd.ca/dlcspm/mip/index.html

[SOAP] http://www.w3.org/2000/xp/Group/

[XML] http://www.w3.org/XML/