EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
CERN – AB DIVISION

CERN-AB-2004-001 (CO)

The CESAR Project
using J2EE for Accelerator Controls

V. Baggiolini, P. Bailly, B. Chauchaix, F. Follin, J. Fullerton,
Ph. Malacarne, L. Mateos-Miret, L. Pereira
CERN, Geneva, Switzerland

Abstract

The CESAR[1] project team used the Java 2 Enterprise Edition (J2EE) to build a controls system for the SPS experimental areas at CERN. This article presents the CESAR architecture and the J2EE platform. It explains the J2EE features relevant for controls and discusses the experience the CESAR project team made with this technology.

International Conference on Accelerator and Large Experimental
Physics Control System
ICALEPCS, Gyeongju, Korea, 13-17 October 2003

THE CESAR PROJECT – USING J2EE FOR ACCELERATOR CONTROLS

V. Baggiolini, P. Bailly, B. Chauchaix, F. Follin, J. Fullerton,
Ph. Malacarne, L. Mateos-Miret, L. Pereira

Abstract

The CESAR* project team used the Java 2 Enterprise Edition (J2EE) to build a controls system for the SPS experimental areas at CERN. This article presents the CESAR architecture and the J2EE platform. It explains the J2EE features relevant for controls and discusses the experience the CESAR project team made with this technology.

INTRODUCTION

The SPS Experimental Areas (SPS-EA) are located in the North and the West of the SPS accelerator ring at CERN. They are composed of ten beamlines with a total length of around 6.3 km, which corresponds roughly to the circumference of the SPS ring. They serve an overall community of around 2000 experimental physicists, both for LHC tests and for SPS experiments. After 25 years of operation, the whole SPS-EA are undergoing renovation in the context of the "Renovation Programme of the SPS Experimental Areas" [1]. The CESAR project (the acronym stands for the "Cern Experimental areas SoftwAre Renovation") is a part of the overall programme, and responsible for replacing the old controls application software with new software, implemented with modern tools and methods.

The users of the SPS-EA belong to five different categories: (1) experimental physicists who use the beamline to test their detectors and to do their fixed target research, (2) beamline liaison physicists employed by CERN who design and set up the beamlines, (3) operators who monitor the whole installation, act as a first line of help and coordinate corrective interventions, and finally (4) hardware and (5) software specialists who are responsible of setting up and maintaining the whole infrastructure.

THE CESAR ARCHITECTURE

The CESAR controls system is structured in three layers, called a "3-tier architecture", as represented in Figure 1. At the bottom of this figure is a beamline with various types of equipment, e.g. magnets, collimators, and detectors. Each of these is controlled by front-end computers (VME technology running real-time software, or PLCs), which are connected to the CERN controls middleware (CMW) [2]. The core of the system, above the middleware, consists of two Java 2 Enterprise Edition (J2EE) [3] application servers running on two UNIX servers. Two servers are needed to guarantee reliability: if one server stops working correctly, the other one takes over. J2EE comes with its own set of middleware communication tools, represented in the figure by the arrow "J2EE Communication". At the top are the consoles (typically desktop PCs) located in the control room and in the various barracks in the SPS-EA where the experimental physicists work. In the vocabulary of 3-tier architectures, the front-ends are called the "Resource Tier", the application servers the "Middle Tier" and the top layer the "Presentation Tier".

The CESAR project is in charge of everything above the controls middleware, i.e., the middle tier and the presentation tier. However, the rest of this paper will concentrate only on the middle tier.

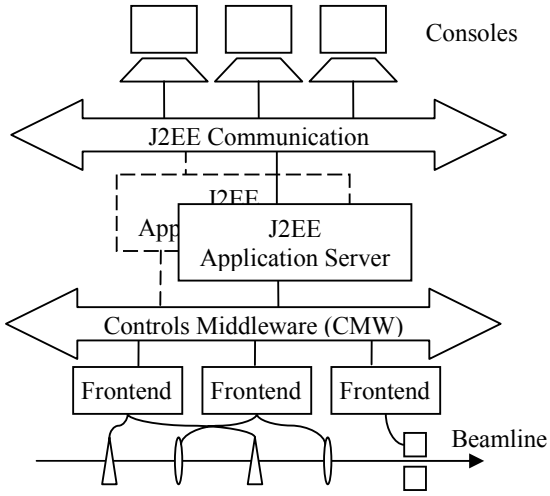

Figure 1: The CESAR 3-tier architecture

Why did CESAR choose a 3-tier architecture, instead of a 2-tier architecture as many other controls systems? The reason is that the middle tier is a robust and well-administrated platform that acts as a mediator between many consoles and many front-ends. In the SPS-EA we have some 35 client consoles and the same number of VME front-end computers. The middle tier carries out common tasks for them, such as data processing and transformations, as well as tuning tasks and algorithms. All database access is done from the middle tier, and transactions and equipment reservations are coordinated here. It is also the only safe place for security-related tasks such as authentication and authorization. Last but not least, redundancy with fail-over from the main server to the back-up server is implemented here.

THE J2EE PLATFORM

The Java 2 Enterprise Edition is a standard framework for building 3-tier applications in Java. It provides guidelines

* CESAR stands for "CERN Experimental areas SoftwAre Renovation"

on how to design and develop 3-tier applications, and a run-time environment called a "J2EE application server" on which to deploy them. A J2EE application server has a so-called "container" inside which software components ("Enterprise JavaBeans" or EJBs) are deployed. The container provides a series of services that implement most of the tasks mentioned above, namely services for remote access, data persistence, security and access control, hot stand-by and fail-over, etc.

To develop EJB components, the developer writes code in Java and adds a so-called "deployment descriptor" in XML. The deployment descriptor specifies how the EJB uses the container services: when the component is deployed, the container configures its services accordingly. In most cases, it also generates some "glue" code between the component and the services. This shall be illustrated with two examples, remote access and persistence. To make an EJB component accessible to the client tier, the developer writes the component and specifies in the deployment descriptor which of its methods shall be accessible remotely. When deploying the EJB, the container generates code necessary for remote access and registers the component with the remote access service. Similarly, to persist information in a database, the developer writes an EJB component and specifies in the deployment descriptor which variables shall be persisted. At deployment time, the container generates the necessary SQL statements to retrieve information from the database and to update it, and configures the persistence and caching service.

USING J2EE FOR CONTROLS

This section illustrates how J2EE can be used in practice for accelerator controls by looking at two further services, transactions (used for settings management) and security.

Settings management is used when the experimental physicists change the kind of beam delivered by the beamline. To obtain a particular type of beam at the end of the beamline (e.g. 120 GeV electrons with a spot size of 2 mm), the physicists use pre-defined beamline settings. Such beamline settings contain a set of control values for all steering equipment in the beamline.

When applying new settings to the beamline, the control system has to keep the equipment on the beamline and the database in a consistent state. For this, it can rely on transactions, which provide "*all-or-nothing*" behavior. The whole process of applying new settings is executed within one transaction. If everything works fine, *all* values should be downloaded to the equipment, and the database updated accordingly. If something goes wrong, e.g. one of the steering equipments fails, the whole transaction should be rolled back to the state before, and *nothing*, neither the database nor the equipment should be updated to the new settings.

J2EE provides a transaction service that, in addition to managing traditional database transactions, can also integrate external resources into a transaction. To participate in a transaction, an equipment has to implement a standard API and handle the "prepare", "commit" and "rollback" commands given by the J2EE transaction manager.

Security and access control is another important feature of J2EE used for the CESAR control system. The different user categories of CESAR have different privileges, which need to be enforced. This requires an authentication and authorization service. Implementing such a service is far from trivial, and managing the usernames and passwords, and the corresponding privileges is a demanding administration task. J2EE is mainly aimed at building E-Commerce application for the Internet. It provides an authentication service that seamlessly plugs into existing password management systems as provided by the UNIX and Windows operating systems. The authorization service implements so-called "Role-based Access Control" (RBAC). RBAC uses the "role" of a user as an intermediate concept to group all users with the same privileges together, instead of keeping track of privileges for each individual user. This drastically reduces the administration overhead. Each SPS-EA user has one or more roles (e.g. operator, software specialist, experimental physicist), and these roles are associated with privileges. For instance, if a person, say "John", works in the role of an operator, he has the privileges of an operator, e.g. amongst others, to load and modify beamline settings.

ASSESSMENT

Based on our experience in the CESAR project, this paragraph presents pros and cons of using J2EE for controls, and possible solutions for the drawbacks. It does not discuss advantages of using a 3-tier architecture in operations, but focuses on J2EE as the most prominent implementation of 3-tier technology in Java.

One major advantage of J2EE lies in the fact that it is an important industry standard with a lot of momentum. All major software and database vendors sell J2EE application servers, and also several open source products are available. Standards prevent vendor lock-in because applications developed on one application server can be easily ported to another one. A lot of development resources are available for the J2EE standard, such as online documentation, books, courses etc. Finally, regarding longevity, it is more likely that products implementing a well-established standard survive over time, rather than a product of an individual company.

The main technical advantage is that J2EE makes the development of 3-tier applications easier and faster, because the developers can rely on existing services instead of having to develop their own. The J2EE application servers implement all system services necessary for building control systems. In general, developing 3-tier applications is not easy, but with J2EE it becomes accessible to the average programmer. Most application servers are delivered with good tools that

greatly facilitate the development and deployment of the EJB components.

J2EE application servers are industry-strength products, well-tested and stable, and backed up by the support of the vendor companies. This is an important issue for mission critical software such as control systems.

As for disadvantages, it is often said that J2EE requires developers to go through a non-negligible learning process. While some learning effort is certainly needed to get started with J2EE, the real issue, in our experience, is not specific to J2EE. To develop server-side applications, (with or without J2EE) knowledge both of Java and of database design is necessary. A person proficient in one of these fields has to invest some time learning the other.

In our view, the only main drawback of J2EE is that – unless special design precautions are taken – a J2EE application only runs in a 3-tier configuration, inside a J2EE application server. This makes testing and debugging more difficult and slower, because the edit-deploy-debug cycle is much heavier: after each source code modification the developer has to re-deploy the application to the application server. In the CESAR project, we managed to alleviate (but not to solve) this issue thanks to a development set-up capable of running the J2EE application server on the development PC itself.

The J2EE developer community has acknowledged this problem, and recent literature describes design patterns and mechanisms to solve it [4, 5]. The idea is to make the business logic (the control system domain software) independent of the J2EE application server, so that it can be executed both in 2-tier and 3-tier set-ups. This combines easy (2-tier) development with robust (3-tier) operational deployment. The business logic can be developed and tested efficiently in a 2-tier set-up, without the overhead of deploying to a J2EE application server. Later, when most development and unit testing has been completed, the business logic is deployed to the operational 3-tier configuration with a J2EE application server. In the accelerator controls group at CERN we are currently exploring these directions, and first experience is very encouraging.

Is there an alternative to J2EE?

So far, we have discussed advantages and disadvantages of J2EE. In our view, the real question is whether there is an alternative to using J2EE. This section investigates possible options and explains why we did not pursue them.

The first idea that comes to a developer's mind is to write parts of J2EE services on his own. This idea should be discarded immediately, because writing such services is a difficult and overwhelming task that requires experts. Accelerator controls experts have neither the mandate nor the expertise to develop such system services.

Another option might be to select a tool or a library for each of the J2EE services. One might take the Java RMI libraries for remote connectivity, an Object/Relational mapping tool for persistence, a library for multi-threading, one for caching, a security framework, and so on. This is better than implementing services from scratch, but it still represents a lot of work. Finding good libraries is difficult, and even more so if they have to be compatible. It then requires a lot of maintenance work to keep libraries compatible over time, when each of them evolves independently. Last but not least, one would still need to implement specialized system services on top of these libraries, e.g. to provide redundancy and fail-over functionality. With a J2EE application server, all this work is done by the vendor.

Another alternative is Microsoft .NET, a 3-tier development platform that is conceptually very similar to J2EE [6]. It could certainly be used for building controls systems, but in our case it was not applicable, because we have a strong background in UNIX and Java, but little experience with Microsoft server-side products.

In summary, for developing 3-tier Java applications we believe that there currently is no good alternative to J2EE.

CONCLUSIONS

This article is based on the experience the CESAR team gained when using J2EE to build a control system.

J2EE makes development of 3-tier applications accessible to the average developer. Once a developer is familiar with the J2EE environment, he can concentrate on the domain-specific code, without having to implement all system service functionality needed for a distributed application. Debugging and testing, however, are more difficult in a 3-tier environment than in a normal 2-tier set-up. We have acknowledged this problem and are pursuing work to overcome it.

Last but not least, a key advantage of J2EE is that it is an industry standard. All major software vendors provide a J2EE application server, and thanks to the standard, one is not locked into one vendor's product but can change. As all successful standards, J2EE is well supported by lots of learning material (books, courses) and other resources (tools, examples, consulting, etc.).

In summary, given the choice of a 3-tier architecture for accelerator controls systems, and Java as an implementation language, in our opinion, J2EE is currently the best solution.

6 REFERENCES

[1] G. Baribaud et al, "The Renovation Programme of the SPS Experimental Areas at CERN", this conference.

[2] K. Kostro et al, "The Controls Middleware (CMW) at CERN - Status and Usage", this conference.

[3] Sun Microsystems, "The Java 2 Platform, Enterprise Edition", http://java.sun.com/j2ee/

[4] http://www.theserverside.com

[5] Rod Johnson, "Expert one-on-one J2EE Design and Development", Wrox Press Ltd, October 2002.

[6] http://www.microsoft.com/net