MOVING TOWARDS A COMMON ALARM SERVICE FOR THE LHC ERA

F. Calderini, B. Pawlowski, N. Stapley, M.W. Tyrrell, CERN, Geneva, Switzerland

Abstract

The Large Hadron Collider (LHC) is one of the greatest technological challenges ever faced by accelerator builders. It is due for commissioning in 4 years and will have a lifetime well in excess of 10. The LHC will contain a completely heterogeneous mixture of industrial controls, both hardware and software, as well as dedicated, specialised, 'home' built systems. As part of the control infrastructure of such a complex machine, a number of 'services' will be essential as aids during operation, such as: logging / archiving, post-mortem, sequences, alarm system, etc. This paper describes the approach to be taken in order to define and provide the alarm service necessary for LHC. Details will be given of: the graceful transition from the current LEP alarm system; accommodating the SPS, PS and CERN's technical services; the technologies to be used; the approach of parallel investigations of industrial and 'home' built systems to ensure the best possible solution; and an indication of time scales to provide an operational system.

INTRODUCTION

The alarm service of the new alarm system includes: the naming, collection, management, and distribution of information concerning abnormal situations, ranging from severe alarm states to warning states, hereafter referred to as Fault States (FS). A FS is defined as the triplet: Fault Family (FF), Fault Member (FM), and Fault Code (FC). A FF represents a collection of elements with similar problems, such as power converters. A FM is an instance of a FF, and a set of FC's represents the problems associated with that FF. The structure FF, FM often follows closely that of the class, device, property model [1], because a class represents a set of equipment with similar control characteristics and most probably, problems. Such FS's will be accepted from any part of the accelerator complex and offered in a structured way to any interested party. The core of this system will be the LHC Alarm SERvice (LASER) [2]. LASER will not include the surveillance and detection of FS's. This will be performed by User Surveillance Programs (USP's), and will be the responsibility of the application writers, equipment and operation groups. These programs could run locally in distributed front-end computers, close to the equipment, or centrally in server computers.

INITIAL INVESTIGATIONS

An important consideration for the launching of this project was that several members of the project team were responsible for the continuing operation of an existing alarm system. This introduced constraints, but also ensured that a thorough understanding of existing alarm facilities was known. Constraints covered the following: a graceful replacement of the current LEP alarm facility [3], which includes the SPS and CERN technical and safety facilities; integrating the PS complex; reducing maintenance; and providing new facilities according to the User Requirements [4]. Since the current USP's will not be replaced 'overnight', and to offer a graceful transition from the current to the new system, gateways have been built to ensure both systems receive all FS's whether they are generated by current or new USP's. This will enable validation of the new system against the old.

To find the best possible approach to build LASER, an evaluation was made of the following: SCADA systems, commercial software products, products used in nuclear installations, and new software technology solutions. Issues encountered were: scalability, a nuclear power plant has in the order of 10's of thousands of FS, whereas we are dealing with 100's of thousands; the need to make on-line changes to FS definitions without stopping the system meant that compiled systems were not acceptable; rule based systems did not offer the data management facilities required; and most products did not offer FS reduction during avalanche conditions. As a result of the evaluations, it was decided to build the system using 'state of the art software technology', based on standards.

SYSTEM OVERVIEW

The Architecture

The LASER system is a distributed, layered application. Each layer forms a foundation of services for the layers above and depends on the services provided by the layers below by means of clear interfaces (see Fig. 1). It is deployed over a 3-tier architecture, where:

- The *resource tier* is made of the dispersed set of USP's, detecting and triggering FS changes.
- The *business tier* implements the system business logic and its services. It relies on a knowledge base modelling the domain.
- The *client tier* consists of dedicated consoles and software components consuming the business services.

The Technology

The system relies on the Java 2 Enterprise Edition (J2EE) [5], which defines the standard for developing multi-tier enterprise Java based applications. The LASER system can be considered as a distributed, asynchronous and service based system. All these sensitive aspects are covered by the J2EE specifications: the Java Messaging Service (JMS) [6] for the asynchronous communication and the Enterprise Java Beans (EJB) [7] for a component based modelling of distributed business services. The Oracle9i J2EE Application Server [8] and the SonicMQ [9] JMS messaging system have been integrated to

provide a scalable and reliable runtime environment. In addition, JMS connectivity for non-Java platforms has been developed via a C/C++ interface, based on the HTTP protocol, in order to allow USP's to seamlessly push FS changes via XML based messages from potentially any platform. Finally, dedicated graphical consoles have been built on top of the NetBeans platform [10], a widely adopted infrastructure backplane for complex desktop applications.

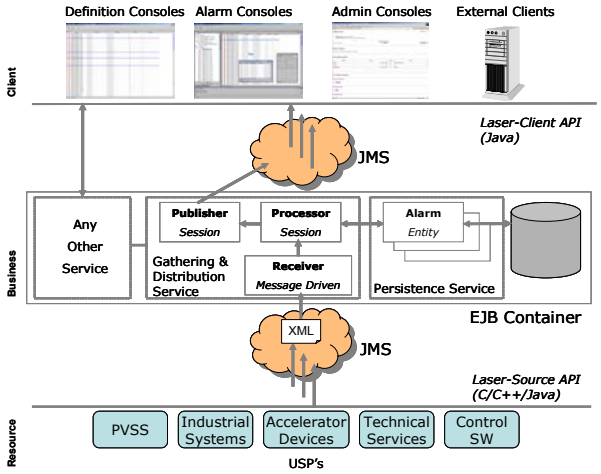

Figure 1: The architecture.

THE RESOURCE TIER

The resource tier is made up of USP's. These are surveillance applications, developed by LASER users that monitor either hardware or software to detect problems. A USP is associated with a set of FS's, which it can switch on, off or change associated properties, depending on the state of the domain being surveyed. The *Laser-source API* has been developed to connect these USP's to the business tier.

LASER has a contrasting and multifarious set of user environments in terms of platforms and languages where USP's run, as well as the equipment to be surveyed. USP's survey the following: accelerator equipment, services such as water and electricity, as well as software and hardware for the control system. Physically, these USP's may be implemented on differing hardware like: PLC's, VME crates with Power PC's, PC's and servers with a variety of OS and software such as LynxOS, Windows, Unix, and SCADA. Each domain can add further limitations such as real-time, embedded software, or no threads. Together, these constraints increase the difficulty of implementing features for the source API. To contend with this, the source API is designed to be small and as simple as possible while being common to all users. It is implemented in Java and C/C++.

The call to the source API is made in the following way. The USP writer builds a message representing a FS by creating a C struct or Java object, and inserting the mandatory data: FS triplet, action such as 'active' or 'terminate', and the time of creation, which could be at the microsecond precision. A number of name / value pairs of data can be added to further describe the FS, for example, demanded and actual magnet current values, a temperature, etc. Communication is done, by pushing one, or a set of FS changes, encapsulated in a message, to a JMS topic, using HTTP for the C/C++ API implementation or JMS for Java. The business tier subscribes to all USP FS topics via a JMS broker. To complement this asynchronous connection, the USP writer is requested to periodically push a 'keep alive' message containing the current active set of FS. The business tier subscribes to these messages and uses this information to check the health of each USP, issuing a FS if all is not well, and verifies its FS's with that of the USP.

THE BUSINESS TIER

The business tier is the core of the system that provides the actual process management, and where the business logic and rules are executed. It implements and offers the following services:

- *FS gathering:* FS changes are asynchronously and sequentially collected from each registered USP. The result is then processed, in parallel, using optimised algorithms and a distributed caching mechanism to absorb FS avalanches.
- *USP status management:* USP's are monitored and their status kept consistent by a periodic synchronisation check.
- *FS grouping and distribution:* FS's are grouped within a hierarchy of domains of interest, referred to as the FS category tree. This tree offers a flexible way for clients to select their FS domains of interest.
- *FS analysis:* FS reduction and masking algorithms are applied to allow clients to filter out redundant FS's when making requests via the category tree.
- *FS persistence:* each FS status is persisted to guarantee consistency at any point in time.
- *FS browsing:* FS definitions, dependencies and status can be browsed
- *FS archiving:* changes to the status of a FS are traced, along with the life cycle of the FS definition itself, allowing historical searches and statistical analysis. Combining this information with that provided by the LHC logging service will allow correlation of data for post-mortem analysis.
- *On-line FS definition management:* FS definitions, their relationships and related data can be updated 'on-line', without system downtime, to maximise availability and maintainability.
- *Alarm console user authentication and configuration:* allows dedicated alarm consoles to authenticate the users and to associate profiles.
- *Scalability and failover:* scalability and failover are guaranteed by clustering techniques.

All the services are implemented via EJB components and offered to both alarm consoles and software clients via specific API's, following session and message façade design patterns.

THE CLIENT TIER

The client tier, which consists of alarm, definition and administration consoles and external software, consumes services of the business tier. These clients, written in Java, use the following Java API's to communicate with the business tier:

- *Laser-client API:* it offers basic functions for accessing FS collected by the business tier. To access 'active' FS, a selection of categories and filters are passed to the business tier via the API as: *property, operator, value*. The current state of that FS selection, as seen by the business tier, is returned. Thereafter, the business tier sends changes asynchronously using JMS. Archive and FS definition browsing are also offered.
- *Laser-console API:* this API is closely related to the alarm console client. It offers login and configuration facilities. Each alarm console user must have an, 'alarm console login'. These users are persisted in the business tier and managed by the LASER administrator.

Currently, the main part of the client tier constitutes the alarm console, which provides a 'window' through which operators and equipment specialists can see the FS situation of the whole or part of the LHC complex. This application is based on the NetBeans Platform, [10], a generic GUI framework for building graphical applications, selected as a standard for the Controls Group. Using this platform enables the developer to concentrate on the application and not worry about the organisation of windows, menus, and toolbars. In order to simplify the development process, some facilities of the GP project, [11], namely the GP Explorer, have been used.

The main purpose of the alarm console is to receive, display and manage FS information in an easy and convenient way. The main modules of the alarm console are configuration, display and management of the different FS lists, namely the: active, memo, inhibit, highlight, etc. Configuration is the personalisation of the alarm console in terms of: FS category selection; filter definitions; FS display fields in terms of their properties; termination behaviour; etc. Each user can define one or more sets of configurations, one of which can be defined as 'default'. This will be used when the user starts a console. If no 'default' is defined, no connection with the business tier will be made and no active FS's will be received. All configuration details are persisted in the business tier. The configuration window displays a list of users and their configurations using the GP explorer. A user can select and browse any configuration, but if another user's configuration is used, a copy is made and attached to the user's profile. Only configurations in a user's profile can be deleted or modified. To enable a 'casual' user to have access to FS information, a 'guest' user has been defined.

The definition console will be reserved for administrators of the different FS domains and will be used to manage FS definitions. A parallel programmable interface, the *Laser-Definition API,* will be made available to allow on-line changes to FS definitions.

TIME SCALES

The mandate for the work was published at the end of 2000. 2001 and part 2002 were spent gathering the user requirements and performing the technological evaluation. Work started in earnest on the vertical, slice, prototype, during 2002. Although a working prototype will be ready by the end of this year, it is clear that feed-back from users, particular in the alarm console area, is essential and practical demonstrations will be given to finalise the design. An operational system should be available by the end of 2004, enabling the current system to be switched off.

CONCLUSION

Much progress has been made, but it is clear that the technologies involved require a steep learning curve. There still remain important areas such as clustering and redundancy, which are new and require further investigations, but we are confident that we will be able to provide the functionality, performance and reliability required for the operational system.

ACKNOWLEDGEMENTS

We would like to thank N. Polivka for her important contributions during the technology survey phase and the modifications to bridge the current system to the new.

REFERENCES

[1] K. Kostro, J. Andersson, S. Jensen, F. Di Maio, N. Trofimov, "The Control Middleware at CERN-Status and Usage," This proceedings.
[2] http://proj-laser.web.cern.ch/proj-laser/
[3] M.W.Tyrrell, "The LEP Alarm System," ICALEPCS 1991
[4] User Requirement Doc., CERN, SL-Note-2002-4CO.
[5] http://java.sun.com/j2ee/
[6] http://java.sun.com/products/jms/
[7] http://java.sun.com/products/ejb
[8] http://www.oracle.com/apperrver/
[9] http://www.sonicsoftware.com
[10] http://www.netbeans.org
[11] V. Baggiolini, L. Mestre, E. Roux, K. Sigerud, "The CERN GUI Platform for GUI in Java," This proceedings.