A Proposal for a Central Error server using Web Services

O.Hensler, R.Kammering, K.Rehlich DESY 22607 Hamburg, Germany

Abstract

As the DOOCS control systems, operating the TTF Linac at DESY, is a widely distributed system, running in TTF phase 2 on approximately 50 front-end and middle-layer computers with 5 to 10 server processes on each processor, a system with local log files is not sufficient anymore. In addition a central logging of all error and warnings will be needed.
We will propose a system based on Web Services running JAVA servlets on the central server(s) allowing the operator to get an overview of the machine status, as well as getting information of single devices by browsing through a tree like structure closely connected to the TTF nomenclature. This server stores the information in XML files, allowing the usage of standard XML tools like search functions.
The communication from the front-ends will be implemented reliable, independent and stateless on a common standard to be portable to different architectures.
First implementations will be discussed.

1 MOTIVATION

The first motivation for this project is to improve the system of local log files for every device server and extend it with a concept of a central error and info server. In our terms errors are messages automatically generated by the front-end servers, a class D_error is provided with every location holding information like error number, error message or severity. Whereas info's are messages typed in by an operator or expert to better explain a behaviour of a device, again a property per location is derived from the base class. It is called DEVICE.INFO, is of type D_ustr and containing basically a severity number, a time stamp and a 80 character string. These two types of information should now be combined on a central error and info server. An other idea is to provide the users the choice of combining the logs to their needs, starting from the logs of the complete facility, looking on one device type or analysing just one particular location.
A further motivation is to learn programming in the concepts of Web Services and extend the DOOCS[1] environment towards the Web.

2 REQUIREMENTS

2.1 Front-end server

A lot of the CPU's that will be used in TTF2 are still be old computer with memory between 32 and 96 Mbyte. For these CPU's a programming language that requires much memory is not acceptable. For this reason, the front-end part will be implemented in C++ into our standard DOOCS server library.

2.2 Error and Info Server

The central error an info server will run on a state of the art hardware, so any restrictions about memory or disk space are no concern. For TTF2 the error server needs to keep the actual errors and info strings of up to 150 front-end and middle layer servers with 1 to 200 locations. This leads to approx. 100000 XML nodes in one central DOM tree. This server has to receive calls from DOOCS servers, but other system interfaces may be required at TTF2. In the case of TINE an integration will be simple, because this protocol is already an integral part of the DOOCS server library. For other system a simple socket library could eventually be provided.

2.3 Servlets

There are different servlets needed for transferring the whole DOM tree, sending the actual messages via Java Messaging Service JMS to the active displays and to combine the history XML files for the display. These servlets will work in the Tomcat[4] environment.

2.4 Display

The display should run on a wide range of computers so a Java applet is the obvious choice. The start-up of the display should not take longer then 10 second and actual messages should reach the display in less then a second. Browsing through the devices in a hierarchical tree should be possible, on every node the status of the subtree should be visible.[5] An acknowledgement of an error on the front-end should be possible.
A standard Web browser can also be used to display the error status.

3 TEST OF XML DATABASES

The first idea was to store the XML data into a XML database instead of creating a directory structure in a filesystem and then storing every message into a separate file. The hope was, that this database will support us in combining all the different XML messages and providing a simple search and query solution. For this reason an XML test file with around 160000 XML nodes were created to test the performance of the following databases:

- **XIndice** , an Apache project written in JAVA, but good for small XML data sets (<50KBytes) only. The searching speed is very slow.
- **EXist** , a SourceForge project, is similar to Xindice, base on the same architecture, written in JAVA as well, but has problems in Xpath calls. Because of this no real performance test was possible.
- **Infonyte**, a commercial database for XML documents up to 1TByte, much faster, but still too slow. A search request needs about 7 second to answer. The price is ~5000 Euro and seems acceptable.

- **Tamino,** an other commercial database from Software AG, written in 'C' language. This database seems to be fast enough, but is very complex. The complete XML structure is locked during an update of one node.
  Unfortunately the price is with 40000 US$ / CPU very high.
- **libxml2** is an XML library used for the internal communication in the GNOME project. It is a very fast, comprehensive "C" implementation to manage a DOM tree in memory and to parse XML files. It is an Open Source project and available for different platforms.

4 CONCEPT

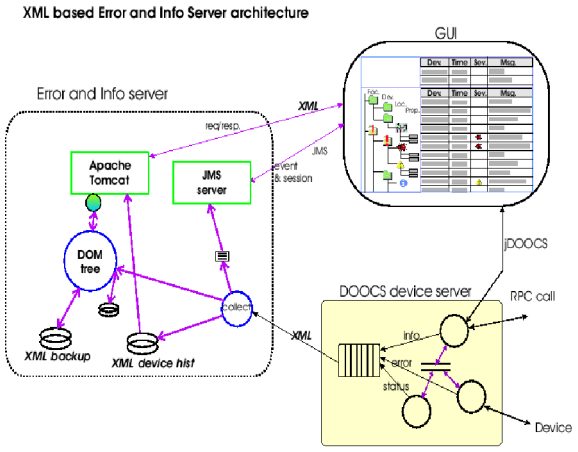

Figure 1

4.1 Front-end server

In order to keep the proved concept of DOOCS, on the front-end server no additional protocols will be added. A new data function class called D_xml is created to hold the XML string. This string can then be send by a usual DOOCS RPC call to any client.
The D_error will be extended with a D_xml type for creating an XML error message. When an error changes, this XML string is send to the error server. To do this call a thread is created in order to avoid interference with the standard server operation. In case of transfer failure, the server needs to store the message to retry later.

4.2 Error and Info server

The error server side is divided into two processes :
A standard DOOCS server will be used to build up the DOM structure of all error and info messages in its memory using the XML library libxml2[2] from the GNOME[3] project. This library is written in C and tests proved it as a very fast implementation. The server receives the error message from the front-ends as an XML string. This string is parsed to find the tags <loc>, <dev>, <loc> and <prop>, which corresponds to the DOOCS nomenclature "FACILITY/DEVICE/LOCATION/PROPERTY". These information are used for some simple tests and then inserted into the main DOM structure. Further they are used to create the directory structure and to store the XML message into a files. The directory structure corresponds directly with the nomenclature of the control system. For every location one directory is created and then all incoming XML files of that particular device will be stored inside. This builds up a kind of logging history for every device.
For every error message one thread will be created to do the processing of the data. This error-server will build up a list of active front-end servers in order to check them from time to time or to start a initial run at start-up time of the error-server. This error server can be configured by a property to accept only logs from a certain facility, e.g. TTF2 or HERA to avoid logs from test systems.

4.3 Java Servlets

A JAVA servlet will be used to transfer the main XML structure to the display applet or to a standard Web browser using the Tomcat[4] Web service engine from the Apache project. The communication with the error server will be done with the Unix InterProcessCommunication IPC. For this purpose the servlet is using the C-interface of Java.

4.4 Display

The display for the operator will be a Java Applet as shown in Figure 2. On the left side, the operator sees the device tree structure that allows to browse through all devices of the Linac. By selecting a node one gets the history entries of some devices or just a single location.
On the top, one has a ticker like window showing the most recent messages from the complete accelerator.

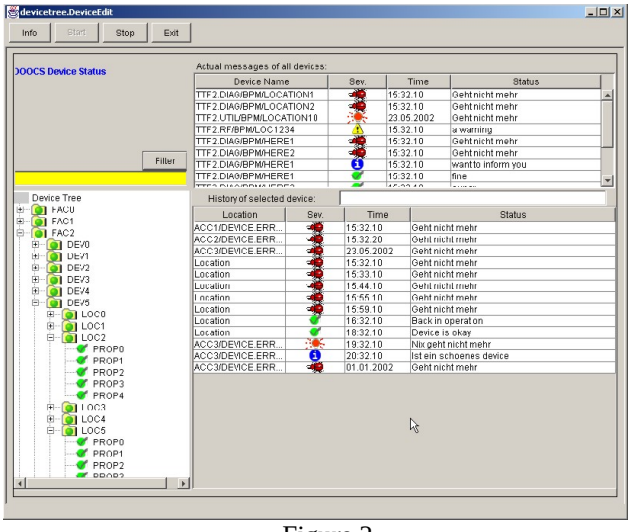

Figure 2

From this applet interaction with the control system like acknowledging of errors will be done by the use of the jDOOCS[6] Java library.

5 WHAT HAPPENS WHEN ?

5.1 The error server starts up

When the error server starts up, it reads the main DOM tree, by parsing the regular stored XML file. A list of active front-end servers is created, then the error server asks the frond-end server via an RPC call to send all information of all locations. This synchronises the main DOM structure of the error server.

5.2 A frond-end server starts up

When a front-end server starts up, it sends all the error and info information from every location to the error server. This is the same data the error server can ask for via the RPC-call.

5.3 Testing for online front-end server

The error server needs to test from time to time, whether a front-end server is still alive or not. It will be done by a similar RPC call like the watchdog. In a case of failure, the complete tree of locations will be marked as error.

5.4 A location is added

A new location of a device can only be added during server start-up. In this stage the front-end server sends all information and the new device is added.

5.5 A location is deleted

If one or more devices are missing after a front-end server start-up, they will be marked as “gone” in the tree. We have to decide whether we need a user interaction to delete the device on the error server or the error server checks for this device on the front-end server and eventually delete it automatically. Eventually this could be done after a certain time period of some days.

5.6 A location is renamed

We have the situation, that some devices have a location name (were they are at the moment) and an alias name (when the device is offline). This will lead to two entries in the directory tree, but this is no problem as long as the front-end server sends a message with the new location name.

6 CONCLUSION

The complete project is still in the development phase, but all relevant parts are tested under real circumstances. So no major obstacles are expected anymore. Unfortunately a more easy implementation with the help of an XML database turned out to be not feasible, cause these databases are too slow. A good tool seems to be the XML library from the GNOME project, since it is very fast and offers all needed functionality. It will be used for the implementation of main parts of the error and info server.

7 REFERENCES

[1] http://doocs.desy.de
[2] http://www.xmlsoft.org
[3] http://www.gnome.org
[4] http://jakarta.apache.org/tomcat
[5] General State and Alarm Monitoring System for ConSys, Torben Worm PCaPAC 2000
[6] jDOOCS – a Java Library for DOOCS
V.Kocharyan YerPhI, Armenia