EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
CERN – ACCELERATORS AND TECHNOLOGY SECTOR

CERN-ATS-2009-104

ON-CHANGE PUBLISHING OF DATABASE RESIDENT CONTROL SYSTEM DATA

K. Kostro, R. Billen, C. Roderick

CERN, Geneva, Switzerland

Abstract

The CERN accelerator control system is largely data driven, based on a distributed Oracle® database architecture. Many application programs depend on the latest values of key pieces of information such as beam mode and accelerator mode. Rather than taking the non-scalable approach of polling the database for the latest values, the CERN control system addresses this requirement by making use of the Oracle Advanced Queuing – an implementation based on JMS (Java Message Service) – to publish data changes throughout the control system via the CERN Controls Middleware (CMW). This paper describes the architecture of the system, the implementation choices and the experience so far.

Presented at the International Conference on Accelerator and Large Experimental Physics Control System (ICALEPCS2009) – October 12-16, 2009, Kobe, Japan

Geneva, Switzerland, November 2009

ON-CHANGE PUBLISHING OF DATABASE RESIDENT CONTROL SYSTEM DATA

K. Kostro, R. Billen, C. Roderick, CERN, Geneva, Switzerland

Abstract

The CERN accelerator control system is largely data driven, based on a distributed Oracle® database architecture. Many application programs depend on the latest values of key pieces of information such as beam mode and accelerator mode. Rather than taking the non-scalable approach of polling the database for the latest values, the CERN control system addresses this requirement by making use of the Oracle Advanced Queuing – an implementation based on JMS (Java Message Service) – to publish data changes throughout the control system via the CERN Controls Middleware (CMW). This paper describes the architecture of the system, the implementation choices and the experience so far.

MOTIVATION

Some essential information on the accelerator is only present in the database, for which especially the value changes are important. Many high-level applications regularly query the database in order to adapt their execution according to this information. Such data polling is not only inefficient but also resource demanding on both sides. The publish/subscribe mechanism, already largely adopted in the implementation of today's applications, solves this issue. Therefore, the possibilities for publishing data from the database have been investigated and exploited.

PUBLISH AND SUBSCRIBE

The publish/subscribe paradigm (pub/sub) is extensively used in modern software to distribute and receive information. Typically, in the accelerator controls environment, front-end computer systems publish data acquisitions of the underlying accelerator components and beam observation equipment. Depending on their interest, software applications can subscribe and get this data. In addition, the subscriber can request to receive filtered information according to his needs, such as less frequent or on-change values.

In the current CERN accelerator control system, the pub/sub paradigm is already extensively used, via the Controls Middleware (CMW) and Java Message Service (JMS) [5]. An extension of these communication techniques towards on-line controls databases would be a natural choice.

DATA IN HIGH-LEVEL APPLICATIONS

The LHC Software Architecture [1] is the current framework and suite of high-level applications that are being developed and used for accelerator operation. These applications are written in Java and are highly data-driven [2]. Data on accelerator *settings* is sent to the hardware and *measurement* data is read back from it, via the controls infrastructure. All of this information is consistently and coherently persistent in the underlying database management systems (DBMS) [3]. On the other hand, some information is exclusively exchanged and used by the software. The following important data falls into this category, with examples for the LHC:

- *Accelerator mode*: operational phase of exploitation for a given accelerator (e.g. "ion physics").
- *Beam mode*: operational phase of a beam in a given accelerator (e.g. "injection probe beam").
- *Bunch configuration*: requested setup of RF-buckets filled with bunches for circulating beam.
- *Experiments handshake*: exchange of status flags with the experiments (e.g. "Atlas veto").

Importance of Changing Data

The actual values of most of these parameters are initialized and modified by control room operators via a programmatically sequenced scenario. For example, during the preparation of the LHC for beam injection, the software updates the corresponding database tables. These on-change modification events are essential for a variety of software applications that in turn need to carry out a number of steps and check different conditions. This is not only limited to LHC operation, but extends to the procedures executed with the participation of the LHC experiments.

Detecting and Capturing Changed Data

Since this requirement existed already for previous CERN accelerator projects, several implementations have been deployed in the past. Applications in the 1980-ies were querying the database tables of interest in a loop construct. This part of the code was the most resource consuming for time critical applications.

In the LEP-era, a more efficient solution was implemented by means of a central process, the *Database Monitor* (DBM). Based on the audit trail views in the database's data dictionary, the DBM detected data modifications of tables for which other applications had expressed their interest [4]. Inter-process communication between the C-programs of that epoch, was based on TCP/IP sockets. By means of a non-blocking method, the DBM clients received a message indicating the data manipulation action that took place.

Driven by today's Java environment, the evolution of technology and the quest for resource efficient software, the possibilities for re-introducing a publishing mechanism, from within the database were investigated.

SELECTION OF THE PUBLISHING MECHANISM

The two middleware communication protocols widely used in the CERN control system – CMW and JMS – would suit the requirements. A direct usage would imply embedding the Java code in the Oracle DBMS. Although this is technically possible, it would be difficult, mainly for reasons of maintenance and stability. Introducing such a dependency on a *third party* software component is generally not envisaged by the database developers.

In order to achieve the optimal integration, the solution was to be found in the Oracle product suite.

Oracle Streams Advanced Queuing

Oracle Streams Advanced Queuing (AQ) is a message oriented middleware, fully integrated in the Oracle database. The message queuing functionality – internally based on JMS – is aimed to automate business process workflows for distributed applications. Therefore, it is well suited for our purposes and in addition, it leverages the functions of the DBMS: Messages are stored persistently, can be queried as any other database table, and be propagated between queues on different computers and databases etc.

Because Oracle Streams AQ is implemented in database tables, all operational benefits of high availability, scalability and reliability are also applicable to queue data. The standard DBMS features such as recovery, restart and security are supported. The usual tools for development like Oracle SQL Developer and management such as Oracle Enterprise Manager can be used to monitor the queues.

Development Integration

An important aspect of the implementation of the database publishing is its integration in our development process. In the case of Oracle Streams AQ this integration is provided via PL/SQL on the database development side and as Java JMS on the Java client side. Consequently, PL/SQL can used to create queues, populate them with data (the updated values in our case). On the client side, a Java-based server retrieves this data (via a JMS *topic* mapped to the queue), and further disseminates the information using the standard control system protocols.

Another important client of this mechanism is our database infrastructure itself. In addition to Java clients, databases can also subscribe directly to other database queues using PL/SQL. We intend to make use of this functionality to allow certain database processes to take into account the distributed data (i.e. mode of the accelerator to condition logging).

TRIGGERING ON DATA CHANGE

At start-up time, the message queues need to be created and populated with initial values for each of the defined subjects. Afterwards, following manipulation of a subject source (database table data), the queue needs to receive the updated subject value. Several options within the Oracle technology stack were considered.

Change Data Capture

The Oracle Change Data Capture (CDC) was introduced in Oracle 9*i*, based on triggers on the source tables. As from Oracle 10*g*, asynchronous CDC is based on a capture process that reads the redo log files of the database. A change record gets pushed into a Streams queue, which can be read by a message consumer. This technique is mainly used for database replication and data movements from source databases to data warehouses.

Data Change Notification

Readily available for Java applications, Oracle Data Change Notification (DCN) methods allows the client to register its interest to the result of certain database queries. By means of a listener method, database change notifications can be received.

Our first investigations using the most recent set of client libraries indicated stability issues, rendering this option as inappropriate for our production environment.

Database Triggers

As database developers, working with user database triggers is straightforward. Simplicity, maintainability with direct and full control on the internals (no need for administrator privileges) are clear advantages. On the other hand, the code does need to be fail-proof and specific triggers need to be created per published data.

The data change triggering is currently implemented with database triggers. Future investigations on the evolution of other techniques are anticipated.

DESIGN AND IMPLEMENTATION

The basic architecture is depicted in figure 1.

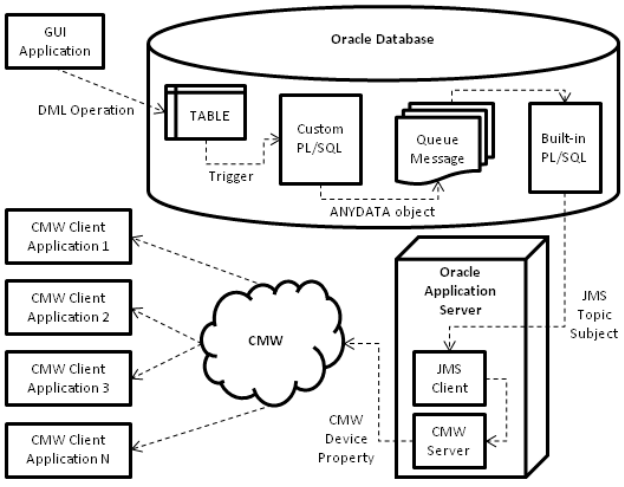

Figure 1 Database Publishing Architecture

As soon as table content changes, the database trigger fires, and the new subject values are pushed into the message queue. Since we are using transactional queues, the new subject values are only published to subscribers

once the transaction responsible for the data manipulation is committed.

The fist deployment only aimed to publish a simple text string (Oracle VARCHAR2 data type). Nevertheless, this allowed familiarisation with the Oracle AQ capabilities and evaluating different aspects (e.g. Queue type, message type, data type mapping, etc.).

Message Data Types

Publication of different data types must be catered for. Oracle Streams AQ has provisions to send Oracle *ANYDATA* objects – a self-describing data type which can contain data of any type – including *user defined* types.

A PL/SQL method must be used to convert native or *user defined* types into an *ANYDATA* object before publishing it into an *ANYDATA* queue.

ANYDATA objects are retrieved from the queue by the Java JMS client as Oracle *AdtMessage* objects.

For the specific implementation at CERN, four *user defined* database types have been defined which represent the data types which are currently required (number, string, number array, and string array) together with a subject name. On the Java client side, custom type descriptors describe each of the *user defined* database types as a Java class which implements the *ORAData* and *ORADataFactory* interfaces.

The published *AdtMessage* objects are parsed using the relevant custom type descriptor, and converted into standard Java native types or collections which are then re-published using CMW.

Re-publishing in the Control System

The queue message content is re-published via the CMW infrastructure, since this makes the information available in a common way to any client in the control system using the usual CMW subscribing methods.

The alternative method of re-publishing via JMS – an obvious choice at first sight – was not preferred for the following reasons: Different JMS implementations are compatible in their API but they are not compatible *on-the-wire,* implying the need for a gateway in any case. Distributing via CMW has the advantage to enlarge the consumer community to C++ clients as well. Finally, the distribution of the *initial value* of a subscription is better handled by CMW.

The mapping between the CMW device/property naming space to JMS topic/subject was done in the simplest way. A single CMW device corresponds to the JMS topic, and the CMW properties are mapped to JMS subjects.

DEPLOYMENT

Application Server

The JMS client reading the Oracle AQ message queue is deployed in a Java container on an Oracle Application Server (OAS). This approach has been taken for all Java applications requiring a tight integration with the Oracle DBMS, deployed in 3-tier architecture.

Administration Tools

The Oracle Enterprise Manager is a powerful tool to monitor and manage the activities on the OAS. Due to the availability of the JMX (Java Management Extensions) interface of the OEM, all information on metrics and performance can be examined easily and dynamically. Since the Java client is also instrumented using Java *managed beans* (MBeans), it is also possible to modify the client configuration on-the-fly using the same OEM JMX interface.

USAGE AND EXPERIENCE

Twenty-six subjects are currently published via the on-change database publishing service to the CERN control system. The early implementation of the service was already used for the first LHC beams in September 2008.

This service was created initially to distribute configuration information to LHC experiments as their control is completely detached from the LHC controls. This is still the main use, but we also expect an increasing utilisation by LHC operational applications.

ISSUES AND FUTURE DEVELOPEMENT

The service is running operationally and behaving as expected. As a general issue, documentation and examples of this Oracle technology in use seem to be scarce and often out-of-date. The only functional issue that needs further investigation concerns the overall latency. The latest observations between data modification and the corresponding reception of the message by the CMW client are around 9 seconds. For the time being, there is no requirement for a faster notification, but there is clearly room for improvement.

REFERENCES

[1] G. Kruk *et al*., "LHC Software Architecture [LSA] – Evolution toward LHC Beam Commissioning", ICALEPCS'07, Knoxville, USA, October 2007, WOPA03, p. 307 (2007)*.

[2] C. Roderick and R. Billen, "The LSA Database to Drive the Accelerator Settings", ICALEPCS'09, Kobe, Japan, October 2009, WEP006.

[3] R. Billen *et al*., "Accelerator Data Foundation: How It All Fits Together", ICALEPCS'09, Kobe, Japan, October 2009, TUB001.

[4] E. Hatziangeli and F. Roeber, "A Description of the SL Measurement and Logging System and the Application Interface Libraries", CERN SL/Note 93-38 (CO), 29 March 1993

[5] K. Kostro *et al*., "The Controls Middleware (CMW) at CERN - Status and Usage", ICALEPCS'03, Gyeongju, Korea, October 2003, WE201, p. 318 (2003)*

* Published at the Joint Accelerator Conferences Website (JACoW) http://www.JACoW.org