REAL-TIME AND CONTROL SOFTWARE FOR THE NEW ORBIT MEASUREMENT SYSTEM OF THE CERN SPS

J.C. de Vries, S.Baratange, C.Boccard, T.Bogey, D.Coussemaeker, M.Dach, J.J.Gras, H.Hiller, S.Jackson, K.Rybalchenko, CERN, Geneva, Switzerland
J.Brazier, Brazier Systems and Consultants Ltd., Southampton, UK

Abstract

The 240 channel SPS Orbit acquisition system is implemented on PowerPC under the LynxOS operating system, making use of multi threaded real-time capabilities. The acquired data is transferred efficiently by DMA on the PCI bus into the main memory. System configuration aspects were implemented in a Broker architecture, where individual threads communicate with an Oracle database and the acquisition systems. This Broker hides the implementation details of the front-end systems. A versatile configuration client is provided in Java, to provide both local graphical user interfaces and remote WWW access using a dedicated gateway to the SL equipment layer. The timing diagnostics of the acquisition system are provided in a LabView application integrating oscilloscope control and channel multiplex control. This paper describes in detail the technical solutions implemented and reports on the arguments which have led to particular choices.

1 MOTIVATION

With new technology and new expectations from machine physicists the replacement of the old orbit system (COPOS) was inevitable. With the similarity of the trajectory systems (TLPOS) the need for a common configuration strategy (XPOS) was clear. The creation of this environment allowed the transparent migration to the new orbit and trajectory systems (MOPOS).

The replacement orbit system not only has the ability to provide multiple orbits in any point in the cycle but also allows multiple clients to access the system in parallel. In addition, the high performance hardware allows real-time calibration of the trajectories for harmonic analysis tools.

2 FRONT-END ACQUISITION

2.1 General Layout

The real-time software within the MOPOS system is based upon the ROCS core software used successfully in other systems within the SL Division at CERN. This has been more fully described in [1]. The ROCS software is a generalised framework for creating event driven, semi-hard real-time applications. The system allows autonomous real time control of the acquisition hardware, and at the same time allows background data conversation to take place. Additionally, user commands (which enter via network 'Equip' calls) are also executed in parallel.

The software consists of three sections viz. User interface and control modules, timing and sequencing modules and acquisition hardware control modules

These modules consist of a number of processes and threads connected together with Inter Process Communication Links (IPC) and using Shared Memory structures to hold the data. These are all written in C and run on top of the LynxOS operating system. LynxOS is a Unix-like RTOS which runs on the CES RIO2 PowerPC processor.

The User Interface consists of three separate modules, namely

- The 'Equip Server' which receives commands from the rest of the system via an RPC based *Equip* call and passes any data back.
- The command processor *'rocsfe'* packages the command into an Action Structure, tagged with the special event *Now.*
- The Socket Server module 'sockmod' provides an alternative, high speed TCP channel with which to pass large amounts of data back.

The heart of the system is the sequencer and action tasks. This is a multi-threaded program which, on an event, examines a queue of actions and spawns a thread to carry out the action. There are two types of actions: *User* where the action is performed once and then discarded and *Real-time* where the action is left on the queue to be repeated. Examples of real-time actions in MOPOS are those carried out at 'Beam-in' and Beam-out' which start and stop the acquisition logic. The communication between the action code and the sub-systems is done with messages sent to the appropriate modules. Some actions, such as background tasks that filter the raw acquisition data, result in new processes being created.

The timing module drives the sequencer by interpreting the machine timing events (received with the TG8 timing card) and generating a stream of logical event messages.

System tables in the form of the per crate, per channel and per event parameters along with the calibration data are kept in a shared memory area with access routines such that they may be used by any part of the system.

Static versions of these tables are held on a special battery backed, non-volatile memory board.

2.2 DMA

A warning event associated with each elementary cycle invokes a real-time task which configures the driver of the MOPOS acquisition controller interface (MACI). This programs the PCI bus controllers (PLX9080) to transfer the acquisition data from the FIFO memory into system memory. The driver interrupt service routine re-programs the bus controller when data transfer exceeds 8 MB. Important system specifications include:

- Processor PowerPC 604e/400 MHz (CES RIO2 8062)
- PMC slots: 6 (CES PEB 6400)
- Data Rate 8 MB/s
- Acquisition Memory 224 MB
- Beam data capacity 28 seconds.
- Acquisition channels 40 times 2.

Dedicated drivers have been developed for the MACI, the trigger module and the input/output module.

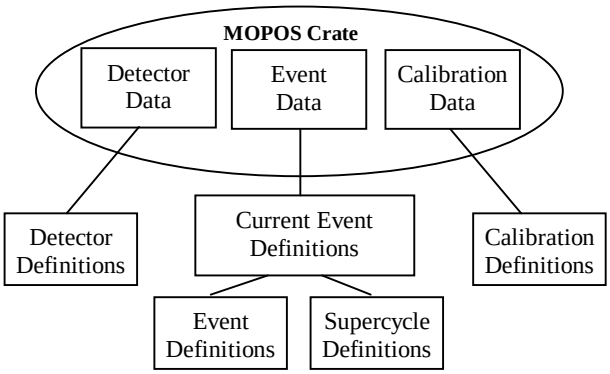
3 CONFIGURING MOPOS

In creating a suitable strategy for MOPOS, many choices had to be made. The chosen strategy should be complex enough to empower users with the facilities they require, but at the same time it should be comprehensible to non-computer experts. In addition, the choice of possible techniques and tools grows each year, hence the choice of implementation strategy also requires careful consideration.

On analysis of the data requirements, it became evident that the data would fall into two categories:

Direct data: This type of data remains constant regardless of the state of the accelerator. An example of direct data is the configuration of detectors.

Indirect data: This data changes depending on the definition of the current running supercycle. An example of such data, is event data which changes according to factors such as timing.



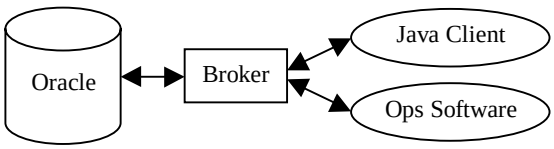
As well as data organization, it was envisaged that the new configuration system should be designed to eventually incorporate the configuration of other beam measurement systems (from the transfer lines). Although the configuration of these other systems differs from MOPOS, many of the same characteristics are shared. Consequently the choice of an object- oriented database initially appeared to be the natural solution. Closer examination of the indirect data however, revealed that a relational database would be more suitable. As a compromise, a relational database (Oracle 7) was chosen as a basis for data storage, and object oriented techniques where used in its design.

Having chosen the basis for storing configuration data, a strategy for database population was required. Proprietary tools supplied by Oracle offered the easiest solution and quickest development time. However these lacked the features offered by the legacy text-based configuration tools used in COPOS - platform independence (Oracle 8 now provides 'web' deployment facilities, but at the time this was not available). It was decided that this platform independence should be kept and as a result, Java was chosen as a development platform for in-house solutions. The choice of Java was further reinforced with its ability to be deployed on the web, and therefore be accessible by the expert from home.

A two-tier development process was therefore pursued, with Oracle Forms as the basis for rapid development for the system developers, and a Java application/applet as the basis for operators and end-users.

4 CONFIGURATION CLIENT / SERVER

With the issue of development environment resolved, the problem of controlled database access was assessed. Tests using the Java DataBase Connectivity (JDBC) package proved this solution would be adequate for communications between the client and database, but an additional factor had to be considered. Although the main access to data would be through the configuration client, other operational software would also require access to the data. If all data access mechanisms were written in Java, this would require existing operational software to be re-written in Java. This was definitely not acceptable and so the concept of the configuration broker was conceived.



The broker provides services to clients for database transactions. In addition, the broker also provides actions for data transfer to and from the hardware. The broker

mechanism not only provides services, but makes the request for these services as transparent as possible. Furthermore, clients may continue execution whilst the broker carries out actions. When both the client and broker are ready the transaction is completed. An example dialogue between client and broker is as follows:

- Client requests service from broker.
- 'class' and location of target system assessed.
- Broker begins execution of 'class' dependant code and returns a request ID (RID) to client.
- Client free whilst request is being handled.
- Client asks broker for results using RID.

The broker is implemented using three threads under C++ in order to achieve good performance and 'class' transparency. Each thread has a designated task - Oracle access, equipment access, and server access. During execution of a service, the broker decides which operations are required, and instructs the relevant thread to perform its task. This allows the server to perform several actions in parallel, thus further enhancing performance.

With the creation of a broker layer, the functionality required from the Java client is reduced to a user interface which allows the user to request services and view the results. Currently, services available from the Java client include configuration of all MOPOS data, diagnostics, news, tuning, calibration and facilities to interrogate and change the current status of hardware.

The deployment of the configuration client as an applet, offers great flexibility in terms of accessibility, but also introduces a new problem of security. To address this, all users must acquire a token (by supplying a user ID and password) from an intermediate gateway (operating from a Linux machine) before communication with the broker may take place. In addition, a record of all client activity is stored in the database.

EXTENDING THE CLIENT

After the success of the Java client, the incorporation of other clients was also considered. An application used to display positional data was one such application. In order to maintain the initial design concepts, an intermediate server, was developed between the equipment and the Java client. This time, the client was a subscription driven acquisition server. The role of the Java client in this case was to provide facilities to plot the acquired data using third party graph classes from the KL group.

5 MULTI-TURN ANALYSIS

Multi turn applications require many seconds of calibrated turn by turn trajectories resulting in megabytes of data to be transferred. To obtain efficient transfer and avoid size limitations in buffer oriented protocols (SL-EQUIP), a TCP/IP socket was used instead. The data is transferred via a set of pipes from the acquisition memory to an external process and then sent to the socket server. Many such external processes can be created to supply user specific data processing.

6 DIAGNOSTICS TOOLS

Acquisition timing diagnostics have been provided by the installation of dedicated low-cost oscilloscopes (Tektronix TDS210) and multiplex modules. The LabView VI for the TDS210 from National Instruments was extended to include the multiplex control and settings handling via XPOS configuration.

7 CONCLUSION

The ROCS system provided the crucial structure to handle beam-synchronous tasks, handle communication channels and provide remote diagnostics.

Using DMA for data transfer into the system memory, has provided an efficient means of acquiring the large amounts of data required. As a result of using DMA the processor power can now be utilized to server user requests and provide calibrated orbits.

The chosen configuration strategy achieves its goal to provide an extendable and manageable means of storing configuration data. It hides the implementation details, and provides clients with services for different ‘classes’ of systems. The decision to use Java to implement clients, has allowed home access for experts through the WWW, and the use of the applications on many platforms. Due to the object-oriented nature of Java, sub-components of the configuration client can be called as stand-alone applications.

Successful experimentation with new plotting packages supplied by the KL group, showed that the use of Java as a serious data analysis platform was feasible.

REFERENCES

[1] A.H.Dinius et all, “Evolution in controls methods for the SPS power Converters, ICALEPCS’95, Chicago, November 1995.