Design of a Remote Data Subscription Mechanism

L.Giulicchi and C.H.Sicard

CERN PS Division, CH-1211 Geneva 23, Switzerland

Abstract

The CERN 26Gev Proton Synchrotron (PS) accelerator complex produces a sequence of different beams in cycles of a duration of 1.2 sec or multiple of it.

A Main Control Room operator, supervising the functioning of (part of) an accelerator for a particular beam, needs to coherently display on his workstation the beam-dependent data, in 'synchronism' with this particular cycle (within a certain time delay).

The classical data collection via synchronous Remote Procedure Calls cannot fully answer to these requirements.

The PS Data Subscription Mechanism addresses the above-mentioned problems:

- it allows a client application in a Workstation to dynamically request accelerator data synchronised to a particular beam;
- the client collects data related to the same beam from different front-ends

The paper exposes the CERN PS specific requirements, and the solution proposed, based on functional data-groups and providing the additional functionality of monitoring process data (i.e., send on change).

I. CONTEXT

The PS control system architecture (Fig.1) connects a network of Unix workstations, via Ethernet and TCP/IP, to front-end processors (FECs) embedded in VME crates.

The PS delivers beam to many different users in a very flexible way [1]. A sequence of 'cycles', each providing beam of given characteristic to a given end-user (denoted below as 'User'), is programmed according to the current

user needs. The cycle sequence (called 'supercycle') can be varied on-line.

On each of these cycles, real-time tasks in a large number (close to 100) of distributed front-end processors capture beam-related data into local object-stores, which all together build a distributed object repository.

Unlike continuous-type processes, due to the pulsed nature of the accelerator cycles, most of these data are of discrete type with respect to time (including a large number of time-scans provided by acquisition systems).

Due to the flexibility of the cycle scheduling and the limited data storage, no cycle-per-cycle logging is kept in front-end memory.

In accordance with regular messages identifying and separating each cycle ('telegrams'), the data storage is triggered by 'machine events', which depend on the process instrument or equipment type, and can be spread along a large part of the accelerator cycle. Some 'software events' can also trigger data storage, e.g. in case of computational objects depending on other data.

The operator at workstation level is usually concerned only with the data produced for one particular cycle type, or even to a specific occurrence of a cycle. Each cycle type may be scheduled at short intervals (1.2 seconds) or less frequently (several tens of seconds), according to the scheduled supercycle.

Each of the several tens of workstations is general-purpose, and can thus be attached to any part of the multi-accelerator PS complex.

A central Oracle Data-base contains the list of all process equipment and their characteristics. An extract of it is distributed among the different file servers to allow fast and

Figure 1: PS Control system general architecture

independent access.

II. REQUIREMENTS

Basic requirements are:

- Each operator station must be provided with a coherent view of the data. This means that the data displayed or computed inside one workstation task must refer to the same cycle occurrence among all the front-end processors.
- The same mechanism should also be usable for processes working on all consecutive cycles (such as displaying particle beam characteristics).
- The delay between data availability in the front-end storage and display on the workstation should be kept to a minimum.
- In case of system overload, the latest available data for a given User has priority on the older ones, which can be discarded (i.e. the system is not designed to guarantee data delivery for all cycles).
- In case of temporary unavailability of some front-end data, the system must continue as much as possible to provide all available data and mark the unavailable one.
- The system must adapt to frequent changes in the subsystem or in the cycle selected by each workstation application.
- Multiple requests for the same data should be treated in an optimised way in terms of front-end processing time and memory usage.
- The network traffic load should be kept low, in comparison with the solution of polling via a remote-procedure call mechanism.

III. DESIGN PRINCIPLES

Due to the large amount of data stored in the front-end processors each cycle, and to the workstation programs most frequent focus on independent accelerator sub-systems, it has been decided to base the design on a subscription mechanism.

Each subscription relates to a given 'user' and to a 'working-set'. A 'working-set' is a well-defined set of objects related to an accelerator sub-system.

Each client program can subscribe to several working-sets, for several users. Whenever a working-set is required by the applications, its structure and all the related information are loaded from the data-base. The working-set structure groups the operational data (properties) into a hierarchy of classes:

working-set: composed by one or more equipment classes.

equipment class: set of equipment having a unique implementation model of the equipment access software in the front-end computers (ex. POWER for magnet power supplies, TRAFO for beam current transformers, etc.).

element: a specific equipment belonging to a class.

property: data related to a specific equipment class element and identified with a property-code (ex. AQN, CTL value, etc.).

The working-set structure and table of attributes are stored in the data-base. Before submitting a request to the receiving service, the application loads the working-set structure (equipment classes and equipment arrays) by means of calls to data-base. At workstation level the working-set contents is split into the working-set contents per each front-end processor. The working-set subscription together with the contents is then transmitted to the front-end processor which completes the working-set data description by loading the list of properties for each equipment class and the related attributes.

This mechanism guarantees that the working-set description, contained in the front-end processor, is equal to the one present in the application.

Figure 2: Architecture of design

IV. ENVIRONMENT

The Data Subscription Mechanism involves both workstation and front-end processor consisting mainly of two parts (Figure 2):

-in the front-end, a **monitor** whose tasks are to acquire data from the equipment object store, to analyse and to transmit them to the receiving service.

-in the workstation, a **receiving service** in charge of transmitting the data request to the monitor and to deliver the data to client application whenever received. The tasks of the receiving service functions may be encapsulated in the client application. The use of either system will be transparent to the monitor which, in both cases, identifies and refers to the subscriber by workstation identifier and the client port number.

The design work reported in this paper focuses mainly on the monitor part.

The Data Subscription Mechanism interacts with the following systems: clients, data-base, telegram, equipment. The commands from the workstations to the front-end processor are transmitted and executed by means of Remote Procedure Calls (RPC). The RPCs implement the following functionalities:

Subscribe: send a request for data transmission;

Unsubscribe: cancel subscription;

Refresh: ask the front-end processor to transmit all the data belonging to a subscription;

Freeze: suspend data transmission;

Unfreeze: reactivate data transmission.

The real-time tasks are responsible for the data acquisition in the front-end. To each real-time task is associated an event-count which is a function of the equipment class and of the front-end processor where it is implemented. The event-count represents the number of events during the cycle which trigger the real-time task's data storage (Figure 3). The real-time task, on such events, sends to the monitor a message containing the following information: User, cycle sequence identifier, real-time task identifier, event identifier.

V. MONITOR

The monitor is the part of the Data Subscription mechanism which is implemented in the front-end processor. Its main activity is: to collect data for each subscribed working-set and user in a synchronised manner, to organise data packages and transmit them to the workstations.

To reduce network traffic, that is due to the amount of data to be transmitted each cycle, an alternative data update mechanism has been implemented. A statistical study has been realised on actual cycle per cycle data change. This study reveals that most of the properties (an average of 90%) which are sent to the workstation during each machine cycle,

are unchanged with respect to the previous machine cycle of same type. The present implementation transmits automatically all requested data whether they are changed or not. Therefore a possible reduction to the network traffic would be a mechanism which, after checking, transmits only the changed data. The new data update mechanism is implemented by the monitor that analyses data before transmission.

Figure 3: Typical data collection timing diagram

The monitor operates through a complex system of objects that can be grouped in three categories:

subscription objects, registering the subscription and the subscriber address (workstation, port id.).

equipment data objects registering the data structure, data values and completion codes;

acquisition element objects: managing the data acquisition synchronisation.

The monitor activity, executing a subscription, can be summarised in the following steps:

- On reception of subscription request from client, register it and get list of real-time tasks and events related to the equipments contained in subscriptions;
- wait for the completion messages from the real-time tasks and register completion for subscriptions which require them;
- when a subscription is complete, collect equipment data, analyse them and send the data to the receiving services of the workstations which subscribed;
- if a problem occurs, such as a real-time task is not completed, after a certain time out, provide the list of equipments which have not been updated.

The unsubscribe command compels the clean-up of all the dynamic attributes of the different objects involved in the subscription execution and frees the dynamically allocated memory spaces.

The freeze, unfreeze and refresh commands, act on some specific flags, associated to each subscription, disabling the comparison mode or in the case of freeze command, interrupting the data process and transmission.

The monitor data analysis algorithm has two operational modes:

"with comparison": before transmitting the data the monitor checks if their values are changed with respect to the previous cycle of the same user.

"without comparison": send all data for any selected cycle. The "non-comparison" mode is required in the following cases: if there is an "AllUsers" subscription, because performing the data check would use a lot of memory and processing resources; if one client application sends a "Refresh" command, which implies transmitting the whole set of data subscribed; if one subscription passes from the Freeze to Unfreeze state for at least one subscribing client.

The monitoring mode is selected each cycle before starting the monitoring process, when the acquisition of data is completed. The mode depends on the comparison flag associated with each subscription. If at least one of the subscriptions for one working-set/user sent by different workstations requires the transmission of the whole set of data, the monitor processes the data without comparison.

VI. RECEIVING SERVICE

For each working-set, a UDP datagram packet made of self-descriptive data is transmitted from the FEC to the destination workstation. When all FECs contributing to this working-set have delivered their message, the client application is awoken and can access the updated data in a fashion similar to the synchronous RPC-based access.

A large part of the workstation applications are generic display and control programs, driven by 'working-set' [2]. In this case, the dataset received matches exactly their needs.

However, some programs work on a variable list of equipments; for these, creating an optional 'dynamic working-set' allows to define uniquely the list of equipments requested.

VII. DISCUSSION

Several other Publish/Subscribe methods exist, such as the commercial package SmartSockets [3] (by Talarian Corp.), where one will find a similar notion of data-groups; however, our unique requirements of synchronised data availability at the client level (due to the 'pulsed' rather than continuous behaviour of the process) could not easily be fulfilled by such packages.

VIII. CONCLUSION

A proper design of such a data-subscription mechanism has been shown to interact with a large part of the control system, from the front-end data storage tasks, via the network layer, up to the client application process data access layer.

Much attention has been given to 'on-time' data delivery, in order to reduce unnecessary delays between operator action and the visible feedback on the workstation screens.

The next step in this work will focus on the transport and reception layer, with the objective to incorporate features currently implemented in two other packages allowing applications to synchronise on machine events and to receive telegram contents through the network.

IX. REFERENCES

- [1] J.C.Bau, G.Daems, J.Lewis, J.Philippe, "Managing the real-time behaviour of a particle beam factory: the C.E.R.N. Proton synchrotron complex and its timing system principles", *submitted to this conference*.
- [2] J.Cuperus, F.di Maio, C.H.Sicard, "The operator interface to the equipment of the CERN PS accelerators", ICALEPCS'93, Berlin, Germany, Oct. 18-23, 1993, *Nucl. Instr. and Meth. in Physics Research A* 352 (1994), pp 346-349.
- [3] on World Wide Web, at URL: <http://www.talarian.com>