A history database for the ESRF accelerator control system.

E. TAUREL
ESRF, BP 220,
F-38043 Grenoble Cedex
FRANCE

November 27, 1995

The ESRF is a 6 Gev synchrotron radiation source which has been in operation now for more than 3 years. This paper describes the recently implemented ESRF machine control system history database. This set of software is based on the ORACLE RDBMS tool (release 7.1). The paper describes how data are stored in the database, the database structure, the filling of the database and the possible storage modes, how the user acts on this filling mechanism and how it is possible to read data for analysis or for long term archiving.

1 Introduction

Even if storing accelerator data is not a mandatory requirement to start commissioning an accelerator complex, requests for such a system came from most of the different teams involved in the accelerator construction and everyday running after a short period of time. Until autumn 1995, at the **European Synchrotron Radiation Facility (ESRF)** there was not a unified way of collecting and storing data for off-line retrieval and analysis. Thus diverse solutions by different people have been adopted. This leads to an uncontrollable and undesirable situation. None (or very little) of the data are guaranteed, different formats which are incompatible are employed, certain parameters are being stored multiple times (e.g. the storage ring current) and the solutions adopted are not always the best. In order to improve this situation, the ESRF controls group decided to provide a single tool : **the history database** which will allow access to all important machine parameters off-line. The data will be guaranteed i.e. stored permanently. Tools will be provided for users to analyse the data to correlate events, look for trends, do preventive maintenance, etc.

2 The ESRF accelerator control system

The ESRF accelerators control system is object oriented and distributed. A object(called a device) can be a piece of hardware, an ensemble of hardware or a combination of these. Device access is based on the client-server model with device servers running on the low level and application software acting as clients [1,2]. A polling mechanism also stores some “read” type command results into a memory buffer (called the data collector [3]) distributed on UNIX machines. Application software get their data either directly from the device server or from the data collector as shown in figure 1.

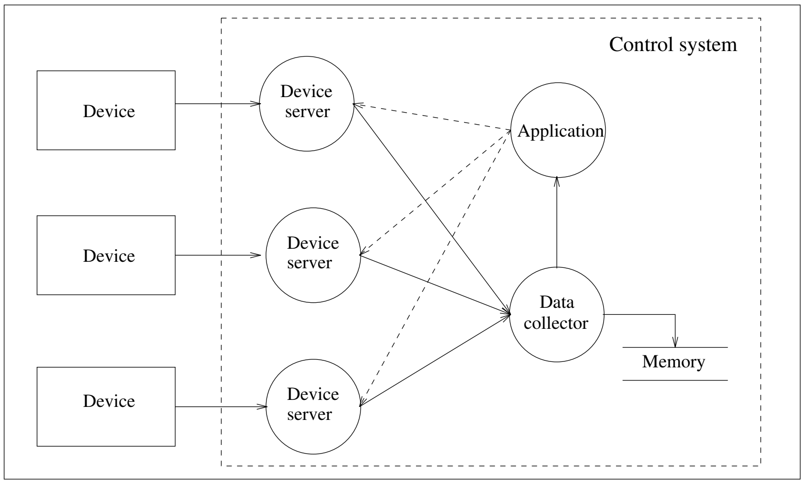

Figure 1: The ESRF control system main blocks

3 The data model

3.1 Signal

Data are stored in the history database as **signal**. Signal is an extension to the device server concept in use at the ESRF. A signal is a data of a simple type extracted from a device command result. Within the ESRF control system, commands executed on a device can return results as several data files stored in a structure. Very often, it is not necessary to store all this structure in a database for off-line analysis but only a part of it. With the help of signal, only the needed part of the structure will be stored. Signal is defined at the class level and so all devices belonging to the same class will have the same signal set. Possible types for signals are boolean, short, long, float, double, string plus char, short, long, float and double arrays. Each signal has a name based on the device name. This name is unique in the control system

3.2 Group

For off-line analysis, it is sometimes very convenient to deal with several signals at the same time. For example, all the signals concerning vacuum in a cell could be joined together in a entity. This entity is called a **group**. A group is a collection of signals and all of them are stored into the history database *at the same time and for the reason defined at the group level*. Within a group, it is still possible to retrieve a simple signal. Within a signal of one of the array type, it is not possible to retrieve part of the signal as an array element.

4 Implementation

4.1 Global design

The history database system is implemented as shown in figure 2. The heart of the system is a Relational Database Management System (RDBMS) from ORACLE Corporation (release 7.1) running on an HP 9000-755 computer. The data source is the data collector. Only data available in the data collector can be stored in this history database. The filling of the database is done by a multi-process sub-system called **filler**. The user interacts with the database

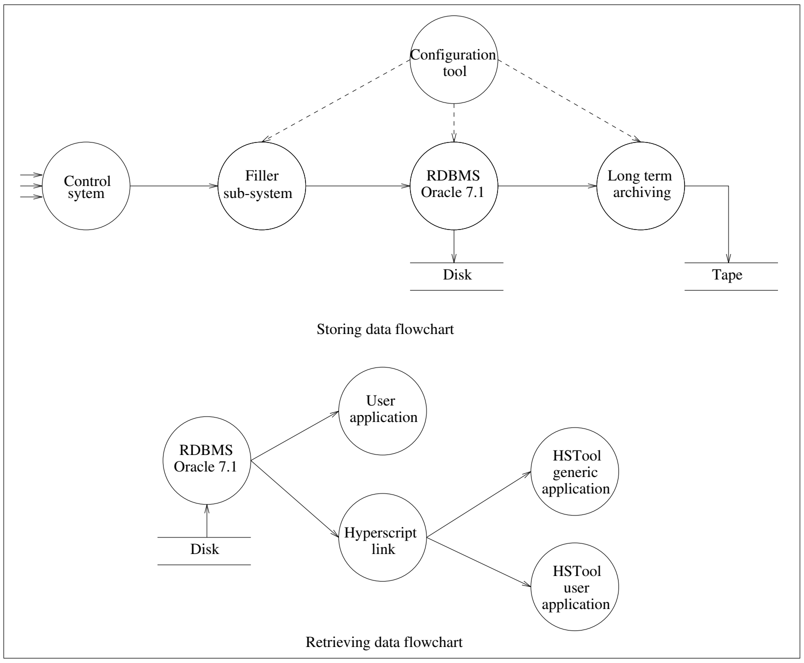

Figure 2: The history database system main blocks

and the filling mechanism via a **configuration tool**. With this configuration tool, users decide to start/stop data recording, define groups, choose storage mode, delete old data etc. Users can get their data out of the database in two different manners which are:

1. In a C program via a C library written for this project.
2. In a commercial spreadsheet (HSTools from IISC)

4.2 Database structure

Within a relational database, data are stored in tables. The most important tables in the ESRF history database are:

1. Signal Definition Table (SDT) where signal definition data are stored such as the device name from which the signal is built, signal alias name, signal identifier...
2. Signal Mode Table (SMT) used to store information on the signal storage mode
3. Signal Value Table where signal data and a timestamp are stored. There is one table for each signal recorded in the database
4. Group Table (GT) used to store general information on the group (group name, group identifier...)
5. Group Definition Table (GDT) needed to keep data concerning the group member list.
6. Group Mode Table (GMT) where information on the group storage mode is kept
7. Group Value Table to store the group member value plus a timestamp. There is one group value table for each group defined in the history database.

Configuration data in the SMT, GDT or GMT tables are also timestamped. It is necessary to keep for six months information on the signal/group storage modes with the the signal/group data. A signal value table (or a group table) is created each time a new signal/group is recorded into the database or deleted when the signal/group is removed from the database. Therefore the database tables number is dynamic.

After a study of the data already registered on files by the different ESRF teams, the disk space required for this history database has been estimated to be **1 giga-byte per month**. With present SCSI disk capacity, this is not a unreasonable amount. The database is running in ARCHIVELOG mode which allows 24 hour a day running with the possibility of doing on-line backup. The archive of Redo-log files generates 200 Mbytes per day, automatically copied to tape.

5 Storing data

5.1 Filler and storage modes

The filling of the database is fully automatic and performed by the filler sub-system. This is a UNIX multi-process ESRF device server. This means that it is able to receive commands from the network and to execute them on top of the regular database filling. It implements 15 commands (start filling for a signal, stop filling for a signal, delete a signal from the list of managed signals etc.). The filler extracts signal data from command results stored in the data collector using a **signal library** [4], analyses the signal value and takes the decision to store data into the database according to the storage mode. Five storage modes are available today. They are :

- Mode 1 : Data are stored at fixed intervals in a day.
- Mode 2 : Data are recorded at regular intervals with a user-defined period
- Mode 3 : The same as mode 2 but some computations are done before storing the data. A short history of command results is stored in the data collector. Four different kinds of computations can be carried out on this history, to find the minimum, maximum, average or RMS.
- Mode 4 : Data are stored when sample n is different from sample n-1. Samples are taken at fixed intervals with a user-defined period.
- Mode 5 : Storage is done when the data is greater, lower, equal or different from a user- defined reference or outside a user-defined window. Samples are also taken at fixed intervals with a user-defined period.

The aim of this history database is to providea **long term history**, it is not a tool for fast recording to study short time events. Therefore, the minimum data acquisition interval is limited to 10 seconds. This value is also set in order to keep the disk space required to a reasonable value. Since this system has only recently been introduced, only the main beam characteristics are stored in the database. Today, around 300 signals (some being of the array type with more than 200 elements) are regularly stored. Most of them are recorded with a update period of one minute. This introduces a load of 9.9 kbytes per minute on the ORACLE server which is not heavily loaded.

5.2 The configuration tool

The user never interacts directly with the database itself or with the filler sub- system. This is done via the configuration tool. This tool allows a user to :

- Display the history database status (number of signals/groups defined, number of signals/groups with active filling, storage modes history for signals/groups etc.)

- Create/Update signal/group. This gives the possibility of defining a new signal or group to be recorded in the database, to define its storage mode and to start/stop recording it.
- Remove a group/signal. This deletes from the database all the data for a signal or a group and also deletes signal/group storage mode data stored in the database configuration tables.
- Delete signal/group data. This feature allows a user to delete signal or group data older than a user defined date.
- Reload old signal/group data. This feature is explained below.

The configuration tool acts directly on database configuration tables and sends commands to the filler sub-system. This software in written in C and the database access is made with ORACLE PRO*C and dynamic SQL.

5.3 Long term archiving

As the disk space is limited, it is not possible to keep data on-line for more than a fixed period of time. Data are kept on-line for **six months**. After this time, they are automatically transferred to tape and the disk space is made available for new data. It is always possible to reload “old data” but only for a limited period of time set to a maximum of **two weeks**. This is done by the user with the help of the configuration tool.

6 Retrieving data

It is possible to get data out of the history database in two different ways, using HSTools or a C program.

6.1 Retrieving data using HSTools

Within HSTools, there is a programming language called Hyperscript. In a Hyperscript program, it is possible to access external functions written in a third generation programming language. All the functions accessible by the Hyperscript language are grouped in a external tool. A external tool written in C above ORACLE dynamic SQL has been written. It incorporates 15 functions. These allow a user who has already developed a Hyperscript program to get the data from the history database instead of from files.

On top of this external tool, a generic application written in hyperscript language is available for any user. The main application window is displayed in figure 3. The user has to select a signal/group name, enter between which dates he want to retrieve data and start the query. The retrieved data will be automatically inserted in a worksheet. Graphical features such as drawing charts are supplied by the spreadsheet software. These features allow users to have a look at their data easily and rapidly.

6.2 Retrieving data in a C program

It is also possible for users who want to develop their own applications to get data out of the history database via a C library without knowing the database structure. This library is written with the help of the ORACLE dynamic SQL. Eleven calls are available. The user must first get the signal/group identifier from its name and is then able to retrieve data from the database with a single call. Today, the power of SQL in the “where” part of a query is not given to the user. To retrieve the storage ring current (stored every minute) evolution during one week (10080 records) needs 10.5 seconds, compared with the 25 seconds needed by the HSTool generic application for the same request.

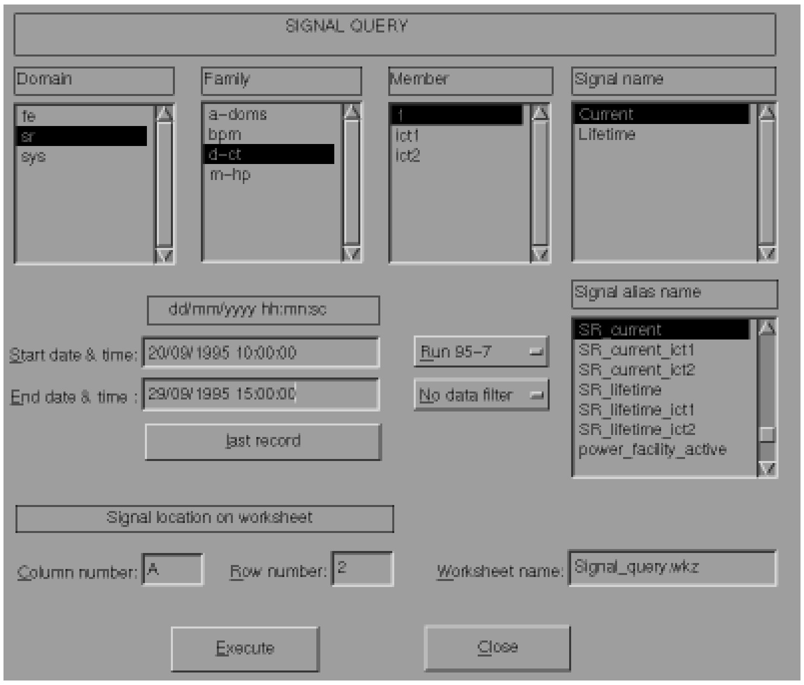

Figure 3: HSTools application main window

7 Conclusion

This history database has been available to the users since beginning of this autumn. Although it has only been in use for a short time, the first reactions from the users remarks are favorable. They appreciate the ease of storing data and the fact that all data are available from the same source. Nevertheless, some of the assumptions we made in the design, such as the response time and storage disk requirements, still need to be confirmed.

Even though a lot of work has already been done to implement this system, some modules are still missing. These include a C library to store data into the database, an X application for the configuration tool and to give the user the full power of SQL to get data out of the database.

8 Acknowledgements

The author would like to express his thanks to F. Moncorgé, B. Scaringella and to S. Boislandon who helped him implement this history database system and to A.Götz and WD Klotz for helpful ideas and support.

References

[1] A.Götz, W.D.Klotz, J.Meyer, “Object Oriented Programming Techniques Applied to Device Access and Control” in *International Conference on Accelerator and large Experimental Physics Control Systems*, Tsukuba (Japan), November 1991.

[2] A.Götz, W.D.Klotz, J.Meyer, E.Taurel, M.Schofield, P.Mäkijärvi, M.Karhu, "A Distributed Control System based on the Client-Server Model" in *Workshop on Control and Data Acquisition*, Calcutta (India), November 1992.

[3] M.Schofield, E.Taurel, "Introduction to data collector/update daemon system". ESRF internal documentation ref: DSN083, February 1993.

[4] E.Taurel "Data collector and signals". ESRF internal documentation ref: DSN105, November 1994.