CERN SPS/89-42 (ACC)

CERN LIBRARIES, GENEVA

CM-P00061476

The CERN/LEP-SPS Librarian System

C. Corniaux, G. Morpurgo

Abstract

The complexity of a software working environment such as the control system for the SPS and LEP accelerators requires a considerable management effort. The software needed to run the machines is continuously evolving, and new releases of application and system programs are introduced rather frequently. The management effort must ideally coordinate all the different software developers and their users (the machine operating team). A software tool, the Librarian, has been written to help them in performing this task. Its ultimate goal is to enable the accelerator operators to retain control of all the software running in the control system. All the source files needed to produce a piece of running software (a "product") are saved together, and can be retrieved either individually or as a whole. Management of the different versions of a product is also implemented, as well as a scheme of protection and access rights depending on the type of user (Librarian Manager, Application Manager, Software Developer). The Data Base ORACLE has been used to maintain logical links between files saved under the Librarian.

Presented at the International Conference on Accelerator
and Large Experimental Physics Control Systems,
Vancouver, British Columbia, Canada,
October 30 - November 3, 1989.

Geneva, October 1989

The CERN/LEP-SPS Librarian System

C. Corniaux, G. Morpurgo
CERN/SPS, 1211 GENEVE 23, Switzerland

Abstract

The complexity of a software working environment such as the control system for the SPS and LEP accelerators requires a considerable management effort. The software needed to run the machines is continuously evolving, as new releases of application and system programs are introduced rather frequently. The management effort must ideally coordinate all the different software developers and their users (the machine operating team). A software tool, the Librarian, has been written to help them in performing this task. Its ultimate goal is to enable the accelerator operators to retain control on all the software running in the control system. All the source files needed to produce a piece of running software (a "product") are saved together, and can be retrieved either individually or as a whole. Management of the different versions of a product is also implemented, as well as a scheme of protections and access rights depending on the type of user (Librarian Manager, Application Manager, Software Developer). The Data Base ORACLE has been used to maintain logical links between files saved under the Librarian.

Introduction

The LEP accelerator control system constitutes a very challenging environment for any software development and maintenance strategy. Several different groups of people have been writing software programs to deal with the wide variety of problems to be solved. Quite a large amount of these programs have been written by temporary people (fellows, cooperants, technical students, etc.) .

The infrastructure over which this software is developed and run consists of a computer network. While some files are visible from every node in the network, others are local to each node, and an additional effort is required to keep consistency between the different nodes.

The software users, which in our case are the accelerator operating team, have little control over the software they run. They are not responsible for the installation of new versions, and they are not always in the condition of knowing where the problems are and who do they have to call, if the software did not work as expected.

This situation is potentially dangerous, and it could lead to huge wasting of time, and dispersion of efforts. It is in this context that the idea of a software tool to keep control over the operational software had first born [1]. Telling it short, we need a system enabling the SPS and LEP operators to know which source code is behind any piece of software running in the control system. We want to be able to answer to such a question like : "this program was running yesterday and now it crashes . What did they change ?".

There are several levels at which a software management strategy can be actuated. The best compromise between rigidity and effectiveness depends very much on the working environment to which the strategy has to be applied, and on the particular

results to be achieved. In such an open and heterogeneous environment like ours, we decided to focalize our attention on the management of the operational stage of the software life, rather than on the development stage.

The tool we have designed and implemented, The Librarian System, is a software product management system. It helps software developers to maintain a record of the different versions of a software product, together with a documentation of the differences between these versions. It also helps software users to run the desired version of a software product. The rest of this article is devoted to describe the principles on which the Librarian is based, and some of the implementation solutions we adopted.

The Librarian System

Let us resume the major requirements of our system.

The Main Ideas

- The Librarian should constitute a centralized place where all the operational software is preserved.
- The Librarian must enable the operator to establish a relation between the software program he is running, and the origin of this program (source code, author, etc.).
- The Librarian should ideally exercise a continuous control on the software running in the control system.
- The Librarian must be able to maintain, in a safe way, different versions of the same software program. It should be straightforward to execute or retrieve any particular version.
- The Librarian must be easy to use for both the software developers and the software users.

We now turn out to describe the way in which the system matches its requirements. For this, we introduce some terminology, to better characterize the **entities** on which the Librarian works, and the typical **Librarian users**

Entities The Librarian works on *objects*, *versions*, *products* and *applications*.

- **Object** : This is the smallest entity in the Librarian System. An object is a file (source file, documentation file, executable file, object file, etc.).
We can also distinguish between *atomic* objects and *derived* ones; an atomic object is something produced directly by the Software Developer, like a source code file, while a derived one comes as the result of some computer operation (for instance, an executable program, "derived" from some source file as the result of a compilation).
- **Version** : A version is a set of objects representing the state of a product (see below) at a given time.

- **Product** : A product corresponds to a working piece of software. Under the Librarian System, all the files necessary to create this piece of software are either kept together, or they are referenced as Librarian objects if they were already saved under the Librarian control. This set of files represents a version of the product. The set of the versions of the product constitute the product.
- **Application** : Set of products. For example, people in the same team can work on a unique application, but each person could develop a different product. The grouping of products into applications is useful from the logistic point of view.

All the information about entities known to the Librarian and their relationship are kept in a database.

The Librarian Users and their roles Two main classes of users can be identified: those who write the software (software developers), and those who run it (software users). Software developers must be registered users of the Librarian System, whereas software users have free access to a subset of it. Among the developers, we assign special functions to the **Application Managers** and to the **Librarian Manager**.

- **Software Developer**: He is one of the programmers who have developed a *product*. Typical operations for a software developer are : installing his product, extracting it, changing the current default version, changing the access rights to his product.
- **Application Manager**: He is responsible for an Application. He has to register all the software developers working on that application. He must inform the users when a new version of a product used in his application is installed, and he can limit the access to the software already installed.
- **Librarian Manager**: He has overall responsibilities (and power) for the Librarian System. He creates new applications and designates the Application Managers. He also has to check and maintain the database integrity.
- **Software User**: The User is only allowed to run a software product installed under the Librarian, and to get information on it.

Software lifecycle

Librarian operations Apart from all the minor details, the main operations which can be performed with the Librarian System are [2] :

- 1. Save a set of files in the Librarian System. Generally, this set of files makes up the latest developed version of a software product.
- 2. Retrieve a version of a software product.
- 3. Use a specified version of a product installed under the Librarian System.

These operations mirror the typical software lifecycle in our working environment. First, a product is designed and implemented by software developers in their own working space . There is **no control** by the Librarian at this level ; in this sense

we say that the Librarian is not a source code management system. The developer is completely free of working in the way he prefers; nevertheless, if he follows some simple rules and prescriptions, he will be sure that the insertion of his product under the Librarian will only require a minimal effort.

A second step occurs when the first version of the product becomes operational, and it has to be made available to the accelerator operators. At this moment the product is *installed* under the control of the Librarian. From now on, the product will be used by the Users, new versions will be added as enhancements or bug corrections, and the old versions are still stored in the system, ready to be used or examined.

Librarian installation as software validation An important point to underline is that the action of installing a product under the Librarian is something more than just copying a set of files to a safe place. The Librarian wants to be really sure that all the necessary files are provided, and that also the procedure to build the derived objects is properly working. In order to do this the Librarian **actually executes** the procedure in its own space, and checks that everything goes on correctly. This could be of extreme importance in case of problems discovered during the software utilization, because it is the key which guarantees that it will be possible to correct the software without worrying about files missing or deleted. One just has to *retrieve* that trouble-giving version of that product into his own working space, and he will be sure of having all the necessary files belonging to the same version.

Note that this "validation" does not have anything to do with the run time behaviour of the software product; it just wants to ensure the consistency of the files saved together.

Librarian functionality

Beside the main and more obvious Librarian operations (creating, updating, retrieving and deleting products; managing users and applications; getting information on what has already been installed; using installed software) some specific points are worth mentioning.

Sharable objects Among the objects belonging to a product, some are only needed inside the product itself (typically source code, makefiles, etc.) , while others can be used by software users (ex. executable programs) or by other software developers (ex. libraries of functions). These latter objects are called "sharable". When inserting a product in the Librarian, the developer has to specify which of the objects are sharable.

Retrieving products It is either possible, when retrieving a product from the Librarian, to restore every file in the exact position from which it came, or to copy them to a given directory. In the first case, if the product was originally contained in a directory tree, this will be fully restored.

Links between objects An important part of our system deals with the possibility that, in order to build a product (A), some objects belonging to another already installed product (B) are used. Later on we will show how developers can use librarian

objects. The point we want to underline here is that we have implemented a mechanism which makes it impossible to delete a product, or a version of it, as long as there exist other products which were built by using it. From the Librarian database it is possible to retrieve all such relationships between Librarian objects. A Librarian feature based on this is the possibility of automatically informing software developers when a new release of a product they are using (for instance a library of functions) is introduced.

Permission strategy We have established a scheme of "access rights" to the Librarian objects. The software developer and his application manager can decide if a product can be extracted, updated or deleted.

Implementation details

The Librarian System is made of two main parts. The first part, used by the Software Developers, consists of a program which enable them to insert, retrieve, update and delete products, to manage applications, and to get information on the software installed in the System. The program also allows the registration of new users, as well as the cancellation of old ones. This program accepts commands via a user interface described in the appendix.

The Librarian System saves on disk all the files provided by the software developers into its own directory structure. Each file is assigned an unique Librarian name (see below). Users will refer to files installed in the Librarian by their Librarian Name.

Two data bases are accessed by the Librarian System. The first one is implemented on top of the ORACLE database system, and collects all the information relative to the software development part of the system (files belonging to a product, which objects are used by other objects , etc.). Fig 1. shows the structure of the tables in the database.

For efficiency reasons, all the information needed to use the software installed in the Librarian (namely, the relationship between Librarian names and UNIX path names) are also kept in a second database, implemented as a normal UNIX file.

The second part of the system is made of a set of utilities enabling the use of the installed software. These utilities are described in the next section.

How to use software installed in the Librarian System

The software programs installed in the Librarian are to be used in the daily accelerator operation. Each piece of operational software can be unambiguously identified by specifying its Librarian name. This name is composed by the names of the application, the product, the object and, optionally, the version. For example, a Librarian name could be

`orbit:first_turn:correct:3.5` ,

where *orbit* is the application, *first_turn* is the product, *correct* the object and *3.5* the version. The actual UNIX name of the file containing the program is transparent to the User of the program. The online database contains the cross reference between the Librarian names and the real file names.

We had to deal here with an important problem : during the development stage, the programmer has to compile and test his program, which is not yet installed in the Librarian system. He has therefore to refer to Unix filenames contained in its own working directory. On the other hand, during the operation, software will be referred via its librarian name. We did not want to oblige the software programmer to make any change in his code (makefiles or programs).

The callable interface To solve this problem, some functions have been written. These functions will either refer to Librarian objects (identified by their Librarian names) or to Unix files (identified by their pathnames) depending on the value of the environment variable **LIBRARIAN**.

A function callable from "C" programs,

```
lib_get(result_pathname, librarian_name, unix_pathname)
```

will place in *result_pathname* either the file name corresponding to the librarian name, or the specified UNIX pathname. *result_pathname* might be subsequently used in a *exec* system call.

The same command *lib_get* can be directly called from UNIX, and can be used in the "makefiles". Similarly, *lib_get_dir*, returns the path to a file rather than the complete file name, and can be used in makefiles to define *include* directories to be searched. Another command, *lib_exec* is available under UNIX, and enables the user to run either the specified Librarian object, or the specified UNIX file, again depending on the value of the environment variable.

Version handling As we saw above, it is possible to run any particular version of a Librarian product. It is also possible to omit the reference to a specific version; in this case the Librarian will use the *current default version*. By default, every time a new version of a product is inserted in the Librarian, it becomes the current default version. It is anyway possible to change the default version of a product via a Librarian command.

Future developments

Our current system is implemented on the SPS network of Apollo workstations. This network **has a** distributed file system, where all the user files are visible from any node, **regardless** from where they effectively are. For simplicity reasons, the first version of the Librarian relies on this file system, and does not deal with files which are local to each node. This could be the subject of some future development, and it is also more important considering that a part of the overall network is made of IBM PCs compatible computers, where all the files are local and no distributed file system is available yet.

Another field for development could be to implement an automatic control and logging facility on the software running in the control system. Such a facility could be built with no relevant difficulty, considering the way in which Librarian controlled software is launched.

Conclusions

The design and implementation of the Librarian has required more or less one man year. Probably the same amount of time will be necessary to refine it, and to carry on the new development discussed. We think of it as time well spent, because our system is specifically tailored to the problems of our working environment. It constitutes another step toward the implementation of a complete infrastructure for software development and operation, missing until now, but needed to support with more efficiency these activities, which represent a relevant part of the manpower resources needed to build and maintain the SPS/LEP control system.

References

- [1] L. R. Evans et al. *Software Managers for Multicycling*. CERN/SPS/AOP/86-10
- [2] C. Corniaux *The Librarian System Users' Guide*. CERN Lep Controls Note 103
- [3] C. Corniaux *How to prepare software products to be installed in the Librarian System*. CERN SPS/ACC/89-7
- [4] G. Morpurgo *ACP : Another Command Processor*. CERN Lep Controls Note 105

Writing software for the Librarian

In order to minimize the effort to install a product in the Librarian, a few precautions are to be followed [3]. Here we list some of them.

- The software developer shall better keep all the files needed to construct a product either in a single directory, or in a directory tree hierarchy made of parallel directories below a common root. In this latter case, it will be possible to insert in the Librarian System the whole directory tree.
- He shall avoid using absolute pathnames (for example for *include files* or in the *makefiles*)
- Inside a product a file name must be unique, even in two different directories.

By following these simple rules, and by using the *lib_get()* function described above, no change will be required when installing the product in the Librarian.

The Librarian user interface

In the current implementation, the software developers communicates with the Librarian program by giving *commands*. The input of these commands and of the related parameters is completely handled by a special library [4]. Through it, commands can be given to the Librarian either from the keyboard or from a file, and a menu input mode is automatically supported. Parameters are prompted for, and all the user input activity is recorded in a *log* file, which can be later on used for documentation, or edited if corrections are required, and submitted again to the system.

APPLICATIONS

N_APPLI	number	APPLICATION NUMBER
APPLICATION	char	APPLICATION NAME
PASSWORD	char	APPLICATION PASSWORD
FIRST_AM	char	FIRST APPLICATION MANAGER

PRODUCTS

N_APPLI	number	APPLICATION NUMBER
N_PRODUCT	number	PRODUCT NUMBER
PRODUCT	char	PRODUCT NAME
PERMISSIONS	char	PRODUCT PERMISSIONS

VERSIONS

N_APPLI	number	APPLICATION NUMBER
N_PRODUCT	number	PRODUCT NUMBER
N_VERSION	number	VERSION NUMBER
VERSION	char	PRODUCT VERSION
COMMAND	char	COMMAND TO BUILD THE PRODUCT
CURRENCY	char	CURRENT VERSION
STATUS	char	COMMAND EXECUTION RESULT

OBJECTS

N_APPLI	number	APPLICATION NUMBER
N_PRODUCT	number	PRODUCT NUMBER
N_VERSION	number	VERSION NUMBER
N_OBJECT	number	OBJECT NUMBER
OBJECT	char	LIBRARIAN OBJECT FILE NAME
SOURCE	char	SOURCE FILE PATHNAME
TARGET	char	LIBRARIAN SYSTEM PATHNAME
SHARABLE	char	SHARABLE FILE
TYPE	char	TYPE OF SHARABLE FILE

Fig. 1. Some of the tables in the Librarian ORACLE database. They represent the entities known to the Librarian and their relationships. For example, all the *objects* with the same values for N_APPLI, N_PRODUCT and N_VERSION belong to the same *version* of a given *product* contained in a given *application*.