Control Software for EUTERPE

P.D.V. van der Stok†, F. van den Berk†, R. Deckers†, Y.van de Vijver†,
J.I.M. Botman‡, J.L. Delhez‡ and C.J. Timmermans‡.
Eindhoven University of Technology
†Department of Mathematics and Computing Science
‡Department of Technical Physics
P.O. Box 513, 5600 MB Eindhoven, Netherlands

Abstract

This paper describes the software design of the EUTERPE synchrotron radiation facility. Applications are developed as a set of separate programs. Services are exported from these programs and can be used by other programs. The programs are built from classes following the object oriented programming paradigm. Objects are created from these classes when the programs are distributed over a set of processors.

The objects of the applications, which represent existing accelerator related objects, also profit from standard facilities provided by the control system software, like: adaptable acquisition and user dependent object views (e.g. B-field for physicist and power-supply for engineer).

This approach makes the application software independent of the underlying control system structure. Applications do not see if the underlying structure is 1-, 2- or 3-layered. Accordingly, the mapping of the application software to the hardware can be postponed until the last moment. Once installed, the control system structure can be adapted to new performance and flexibility requirements without consequences for the application software.

I. Introduction

The Eindhoven University of TEchnology Ring for Protons and Electrons (Euterpe) is currently being designed and constructed at the physics department of the Eindhoven University of Technology (EUT). The accelerator is designed for the production of synchrotron radiation extending from the infrared to the ultraviolet [1]. Apart from radiation production, the ring will also be used to study beam dynamics and to assist in the teaching of accelerator physics. In the same spirit the design for the control system for the ring has been started. The design should not be limited to the most cost effective way to control the accelerator, but it should support accelerator control in general. The object oriented method is investigated for its applicability to accelerator control. Claims about reusability, maintainability and simplicity [2] can be evaluated. In this paper a summary of the main results is presented. A more detailed treatment is found in [3]. The verification of the object orientation claims is only possible in a later stage when the control system is actually used and modified.

II. Requirements

The requirements on accelerator control are motivated by (1) the experimental nature of the installations, the distribution of the accelerator over a large area, (2) the often conflicting wishes of different user groups and (3) synchronous actions of different accelerator components. Less essential are the short time range of many phenomena and the wish to access the equipment from personal computers situated at widely dispersed locations.

The experimental nature of the installations leads to changes in the type and amount of equipment to be controlled. Also changing operational conditions require that the relations between different components are frequently modified. The *flexibility* of the control system that allows additions and removals of pieces of equipment without major control system shutdowns is a major requirement.

During the lifetime of the accelerator, components break down and are replaced by other components. For a high availability of the accelerator, online modifications to the equipment should be possible with a minimum of effort or modifications to the application programs. Consequently, the *reconfiguration* of the equipment and the associated control system components should be possible.

The physical distribution of the accelerator components makes it attractive to group equipment at a number of locations and to control this equipment from computers situated at these same locations. The wish to control all the equipment from one central control room and provide some controlled access from individual workstations necessitates the *distribution* of the control system.

The different groups that build, maintain and operate the accelerator have widely different views on the same pieces of equipment. An engineer is interested in bit patterns, a physicist in current and field values. Consequently, a piece of equipment should offer *multiple views* for an efficient manipulation by a heterogeneous population.

0-7803-1203-1/93$03.00 © 1993 IEEE

Accelerator components are grouped to construct higher level components. Several magnets can be grouped to produce a higher order harmonic magnetic field; a phase and a strength are the only required attributes of such a group. At a lower level, the individual magnets still need to be controlled by individual settings. At a lower level yet the magnets are composed of a power-supply, the magnet status equipment and the timing equipment. *Multilevel* access is a characteristic of accelerator control. Dependent on the situation, groups of equipment or individual pieces of equipment need to be accessed.

Applications can be invoked concurrently on different computers. Many different applications often need values from the same pieces of equipment (e.g. the beam current). The *atomicity* of groups of actions by the control system has the results that: (1) the concurrent access to the same piece of equipment by different applications needs to be organized such that no invalid results are returned to the applications and (2) when a series of actions or acquisitions is done, it is important that the results concern the same time period and that all actions are completely executed or not at all.

III. Distribution

The distribution of the applications over the computers is not usually supported by object oriented languages. Therefore, an extension to C++ has been developed called DEAL [4]. This language allows the system designer to define *programs* that contain all elements which constitute one inseparable piece of coding. A *Process* (instance of a program) can export the procedures of classes and objects which are visible to all code in the process. Other processes can use these exported procedures and objects. Procedures of other processes are invoked with the Remote Procedure Call (RPC) paradigm [5]. These concepts are shown in more detail in Fig. 1. In process (circle) A two objects (rectangles with rounded angles) O1 and O2 are present. The procedures (ellipses) O1.PO and O2.PO are invoked simultaneously from process B, while an unspecified process NN is invoked from O2 in A.

The processes allow the separation of the control system software into independent programs, which at a later stage can be loaded on the target hardware. This strategy makes the software as independent as possible of the underlying hardware-structure. However, at a lower level hidden to the application program writer, the operating system should know the physical locations of the processes.

During the design of the software architecture, knowledge about possible distribution configurations is nevertheless required. When only one program that contains the complete control-system information is developed, the later distribution of the program necessitates that it is split into smaller programs. When the programs are split into functional parts (e.g. one program for all quadrupole power supplies) and a geographical split is required later, the program splitting needs to be redone. It is important to realize as well that every time an object in another process is invoked, time-consuming parameter-copying and process-switching occur. The control-system designer should verify that program splits are both functional (i.e. reflect the structure of the accelerator) and efficient (i.e. no unnecessary overheads are incurred during execution).

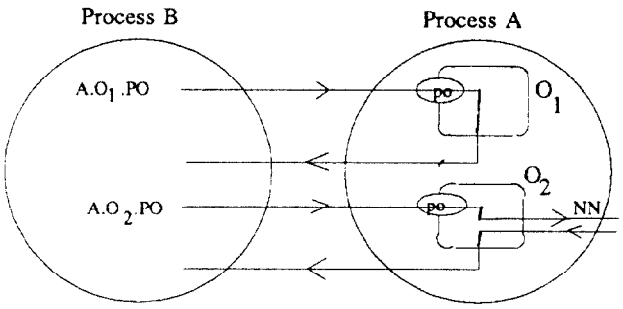

Figure 1: Processes and objects

IV. Multi-level and -view access

The datamodule concept, as successfully defined at CERN [6], is eminently supported by object oriented languages via inheritance. In Fig. 2, a diagrammatic representation of the classes is shown. A rectangle represents a class and the arrows represent inheritance relations. Visible procedures and objects are drawn on the class (rectangle) edge and internal (hidden) ones are drawn inside the class. The *equipment* class (e.g. power-supply) inherits from the different views and the *Device* class. The *Device* class contains one (or more) object(s) of the interface class that represent the interface card(s). General facilities which are always needed act upon the device such as: e.g., enable, disable. Access protection can be defined by adding procedures not shown in Fig. 2 which handle protection and access rights. The code for such an equipment class can then be written by the person responsible for that particular piece of equipment without bothering him with the less interesting other details already defined in the class *Device*. The equipment class with all its views is exported from the enveloping process. In the application process that uses the equipment the object of the equipment class is accessed with the view that suits the application.

The above *multi-view* access is complemented with a *multi-level* access. A family constitutes a piece of equipment to be controlled by an operator. It can be composed of a power-supply, one or more magnets and a timing module. At the family level, modifications to magnets are defined in Gauss or other relevant units. At the lower

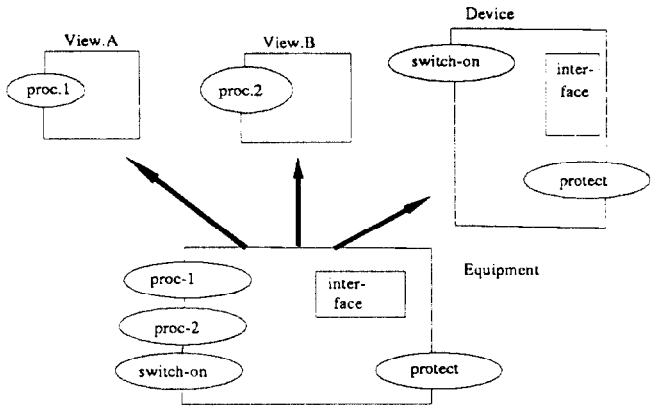

Figure 2: Inheritance from views

equipment level access can be defined in Amps or just bit patterns depending on the view of the user. At the lowest interface-level, the access is purely done in bit patterns. At a higher level, families can be grouped to construct accelerator parameters such as bumps or Q-values.

V. Atomicity

When a periodic access to the same equipment is required by multiple applications, it is more efficient to access the equipment at the highest required rate and to store the values in memory. For example, the reading of Pick-Up electrodes can be done every millisecond locally in one computer. The different applications can then read those values concurrently from memory and select the appropriate ones. When all applications access the equipment directly, the accesses to the equipment have to be strictly serialized to prevent inconsistencies and even equipment damages. These sequential equipment accesses each take longer than the simple memory location accesses. To reduce message overhead, multiple acquisition values can be sent in one message. An advantage is that the data read by different applications are identical and lead to the same result in applications executing concurrently at different locations.

The observer object has been introduced to support this approach. It consists of an action that has to be executed on a piece of equipment and a period that defines the invocation rate. Different rates for different applications can be defined. The observer object orders the rates and returns the corresponding values to the correct applications. Data are stored in shared objects that are read by the application. An action may return one result from a single ADC access or multiple results as provided by a transient recorder. When reading the values, the application needs to specify the first and last value and the total number of values it wants to read.

Two modes of the observer object exist. A single access mode, where only the latest result consisting of one or more values is returned and an averaging mode, where the average of a series of values is returned.

When applications concurrently read the values from observer stations, two problems may occur: (1) while the application reads a set of values of one observer station, the same observer station can be activated one or more times, thus overwriting the shared object and leading to inconsistent results and (2) when the values from related observer stations are read (e.g. a set of pick-up stations), these values are not automatically related in time. Concurrency control algorithms based on time-stamping assure that consistent results are returned to the invoking process [7]. Consequently, applications read data from sets of equipment which are produced over the same time period and multiple data from the same equipment will concern one continuous time interval.

The same concurrency algorithms also assure that modifications to related equipment are executed at roughly the same times and to all equipment involved. When actions on the same equipment are required by two conflicting applications, either both actions are executed sequentially, or one of the actions will fail with an error message that the equipment setting is also modified by another application.

References

[1] J.I.M Botman, Boling Xi, C.J. Timmermans, and H.L. Hagedoorn. The EUTERPE facility. *Review of Scientific Instruments*, 63(1):1569–1570, january 1992.

[2] Bertrand Meyer. *Object-oriented Software Construction*. Prentice/Hall Int., 1988.

[3] F. van den Berk, R. Deckers, and Y. van de Vijver. Object oriented development of the control system for EUTERPE. Technical Report ISBN 90-5282-249-2, Instituut vervolgopleidingen-TUE, 1993.

[4] D.K. Hammer and O.S. van Roosmalen. An Object–Oriented Model for the Construction of Dependable Distributed Systems. In *Proceedings of the International Workshop on Object Orientation and operating Systems*, Paris, september 1992.

[5] A.D. Birrel and B.J. Nelson. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, 2(1):35–59, February 1989.

[6] M.C. Crowley-Milling. The Data Module, the missing link in high level Control Languages. In *Proceedings of the 3rd International conference on Trends in On-line Computer Systems*, Univ. of Sheffield, March 1979.

[7] P.D.V. van der Stok and A.E. Engel. Shared Data Concepts for DEDOS. In *Proceedings of the 10th IFAC workshop on Distributed Computer Control Systems*, Semmering, Austria, September 1991.