EXPERIENCE WITH AN OBJECT ORIENTED CONTROL SYSTEM AT DESY

G. Grygiel, O.Hensler, K. Rehlich, DESY

ABSTRACT

For the TESLA Test Facility (TTF) we have developed the object oriented control system DOOCS. The Linac is now in operation for more than one year. The control system is implemented on PC's with the Linux operating system and on SPARC processors with Solaris. During the last years a lot of experience with the design of C++ libraries has been gained as well as implementing device server programs with the help of these libraries. A device is added to the system with the assistance of a template and a script to automatically generate a new server process. Only device specific code has to be programmed for the server. We will also describe the progress we made in creating operator applications using object libraries. The DOOCS Data Display tool, based on these object libraries, allows us to create graphical operator panels without programming.

INTRODUCTION

The TESLA Test Facility (TTF) [1] at DESY was built to demonstrate the feasibility of superconducting cavity technology as a prototype for a 30 km long e+/e- collider. TTF consists of an infrastructure to prepare and test cavities and a 100 meter long linac to test the superconducting acceleration modules. The final linac will be used as a user facility including a Free Electron Laser (FEL).

The Distributed Object Oriented Control System (DOOCS) was designed for the TTF linac. It is completely written in C++ language and follows the object oriented programming paradigm [2] since the design idea from the device level, including the device servers, up to the user display is based on objects.

This object oriented abstraction model helps for clean programming interfaces and in addition it helps in the overall system design including the hardware for a machine. It is a significant step forward in the goal to improve software productivity and quality. A properly designed class library allows the reuse of code for very different applications without losing flexibility and extensibility. Also the maintenance phase of the software cycle gains by applying object oriented technics since the programmer has to deal with a well defined interface only.

ARCHITECTURE

A DOOCS design goal was to implement a device server as an independent program that completely controls a number of devices and provides the device data to the network and receives messages from the clients. The architecture (Fig.1) consists of device servers with hardware connections on the lowest level, middle layer servers to implement gateways, data bases or sequencers, and client application programs at the top level.

DOOCS is a distributed system because the device definition is done on the server side only and is transparent to the

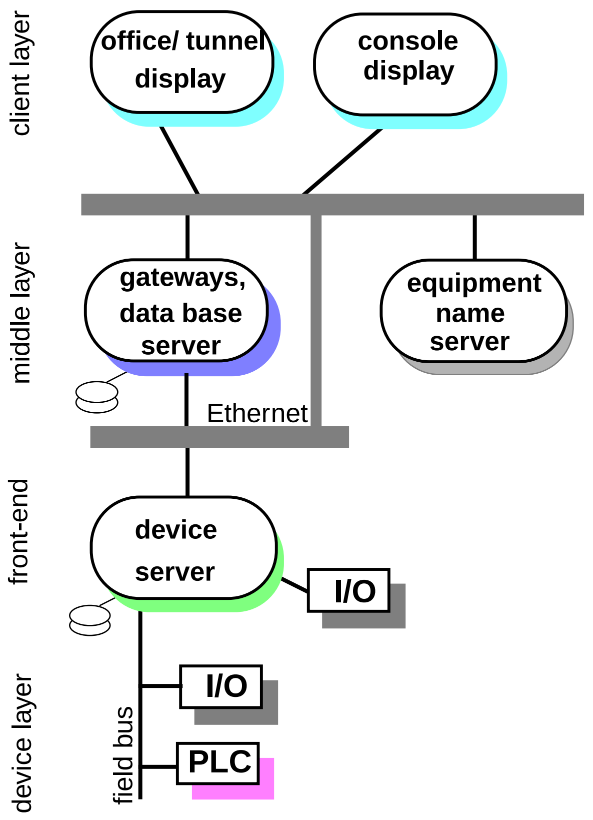


Fig.1: DOOCS Architecture

clients. Whenever a server is started with new device instances or with new properties added, these changes are immediately available on the network; there is no central database to hold these items. Client programs may request an actual on-line list of all devices and properties by means of a query request.

Many different servers can run on the same hardware or these servers can run on different CPU's. Even the same type of a server with different device locations is able to run on many CPU's.

The names (IP-Addresses) of servers are resolved by an Equipment Name Server (ENS). If an application performs the first communication to a device server, a request is send to the ENS to resolve the host name of the server. Further data transfers are directed straight to the device server. For redundancy, the Equipment Name Server may run on multiple computers.

SERVER OBJECTS

A DOOCS server is a UNIX process that consists of objects from a C++ server library. OO programming is described in many articles [3] and is used successfully in a lot of projects. In DOOCS the fundamental object oriented concepts are used in the following way in the device servers:

- **Objects and Classes**: objects are the actual created instances of a class. They combine the attributes and the code of a certain device type and of the individual properties of a device.
- **Encapsulation**: the data of an object is part of the class definition and can be hidden from accesses outside of the class. A class defines the interfaces to its protected data. A device property, that is accessible on the network, implements such a protected data item. It provides an interface to both the external network access and the internal device server routines. Side effects from modifications of the program are greatly reduced by the encapsulation and the program is more robust in case of extensions.
- **Inheritance**: this key concept of object based systems provides a great step in programming efficiency. All standard network data objects of a device server inherit from a basic class that defines the main procedures for the network interface. The actual data object, that represents the device dependent function, inherits from the standard data object and adds a specific part only. Objects with a common base class can be easily combined in a container with different kinds of device properties for instance.
- **Polymorphism**: classes with an identical interface but with different behaviour are used in many places in the server library. In the device property example the interface to read a value from the network is identical for all properties. But, the method could implement a simple read from memory or activate a call on the fieldbus including error handling.

To represent a device in a DOOCS server two main objects where identified and are specified in classes. One class implements a device and the second one defines a data property of a device.

A server process has a container of device classes. Each of these objects represents one instance of a certain device type. E.g. the view screen server contains one instance of the view screen class for each screen in the machine.

Each device server class is composed of a number of data property classes. Data classes represent the attributes of a device that are accessible on the network. All data objects inherit from the D_fct base class. Standard data classes in the library are integers, floats, structures for filter settings, arrays and complex data types like historical trend data or spectra, etc. Actual servers overload from these standard classes and just add the device specific data functions.

A DOOCS device server always contains some standard parts to handle the device errors and on-line status, to read and write the configuration file, to control the server updates and to do some statistics. This standard part is identical in all servers. Since a client application can query all device names and property names during run-time, it was possible to assign device names dynamically. A moving pump station is implemented in this way for instance.

In addition to the device and network data classes the server library collects a lot of helper classes. These include the archiving for historical trends, the device accesses for VME or fieldbusses, the read and write of the configuration file or error logs. As mentioned earlier, it was a design goal to run a device server autonomously. Therefore, it keeps all its data bases locally to be independent of the network.

A large number of different DOOCS servers have been written with the server library. The specific code was always added by overloading some functions of the base classes. Examples for the different server categories are: calibration data base (the data is stored in memory only), normal device server (data is read from and written to the device), dual process (one process for the network communication and the second one for the real-time process are connected by a shared memory), server for ‘intelligent’ devices (PLC or DSP front-end), middle layer server (data is fetched from device servers on the network), gateway servers (controls a device via a different control system protocol), supervisory server (watches other servers, keeps them running and collects system statistics).

AUTOMATED SERVER CREATION

To further speed up the process of server creation, a script was developed to generate a device server from an ASCII template file. In this template file the basic attributes of the instrument to be controled are defined including the number and location names of this equipment in the machine. The script creates from this information a makefile and the complete server code. A UNIX Make on these files then produces the actual server process. When this process is started all properties and all instances of this equipment are available on the network and are immediately accessible with the generic DOOCS tools.

Although a complete definition of all classes and overloaded functions can be defined in the configuration file, the generated code is usually used as a template for a device server only. This rapid prototyping process allows premature system testing since all values of the full programmed device are available in an early design stage.

The main advantage of this automated code generation is of course to release the programmer from the burden of writing again and again the templates for device servers and to help beginners to get a running server process in a short time.

Display Objects

Real devices are reflected in a device server object. A further device instance is added to the system by a simple copy, no matter how complicated the equipment is. The same idea of duplication applies to the device views in the graphical user displays. A view of a device has to be created once and a copy and paste in the graphic editor creates further instances. Different views of a device could be an animated overview, a control panel, an expert window or a window with statistics.

To create such components, windows and panels the DOOCS Data Display (ddd) program [4] was developed. The program is a graphical editor for a synoptic and online data display. Animated device objects are completely created from graphical elements, no line of code has to be written. Equipment addresses and other display attributes are entered by forms only. Simple shapes as lines or text can be drawn with the editor as well as complex device data or graphs. The creation of a graph with historical device data in a window is as simple as drawing a line. In a form the type of the graph, some attributes and the device address are specified. The default scale settings are read directly from the device whenever the window starts up.

All drawn elements may be combined with other components to build complex libraries of reusable parts. These library items are then usable and scalable in further drawings. In this way hierarchical structures of device synoptic displays can be build. All elements of such a structure keep their relative device addresses that are combined with the base address from the top level of the component. Adding a new complex device in a display is done by a copy and paste command followed by a simple specification of the base address of the new device in a form.

Furthermore, control elements to set data in a device, to execute a UNIX command or to start a new window are part of the editor toolset. These commands may appear as a visible button or as an invisible field.

Conclusions

During the last few years we gained a great deal of experience to operate a linac with object oriented technologies. The object oriented method is not just good programming practice but it also leads to cleaner system designs in the whole field of control systems i.e. the device hardware, the device servers, the user interfaces and the program libraries. Object oriented methods save significant development time and more important, create easier to maintain and reliable systems.

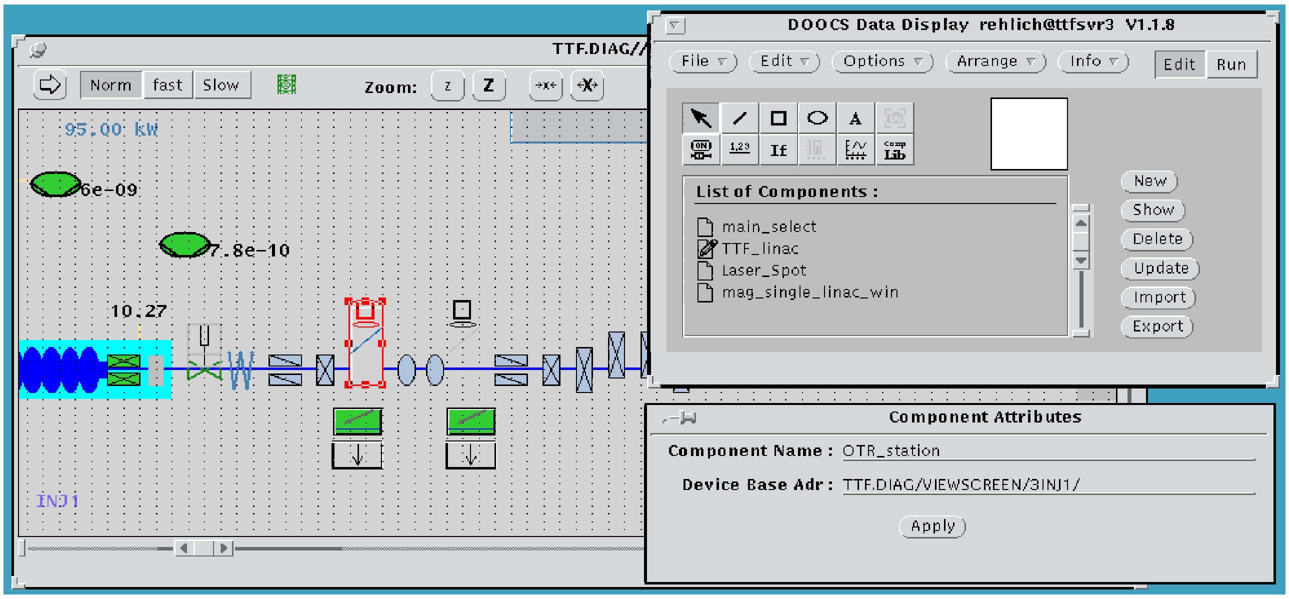
References

[1] D.A.Edwards et al, “TESLA Test Facility Linac - Design Report”, DESY-Print, March 1995

[2] K.Rehlich et al, “DOOCS: an Object Oriented Control System as the integrating part for the TTF Linac”, Proceedings ICALEPCS ’97, Beijing China

[3] D.Quarrie, “Object Oriented Programming and High Energy Physics”, CERN School of Computing, Geneva 1995

[4] K.Rehlich, “An Object-Oriented Data Display for the TESLA Test Facility”, Proceedings ICALEPCS ’97, Beijing China



DOOCS Data Display rehlich@ttfsvr3 V1.1.8

List of Components :
main_select
TTF_linac
Laser_Spot
mag_single_linac_win

Component Attributes
Component Name : OTR_station
Device Base Adr : TTF.DIAG/VIEWSCREEN/3INJ1/

Fig.2: Zoomed view of the TTF linac display in the ddd editor. A selected view screen station is shown with the corresponding panel to specify the device name