DOOCS: an Object Oriented Control System as the Integrating Part for the TTF Linac

S. Goloborodko*, G. Grygiel, O. Hensler, V. Kocharyan**, K. Rehlich, P. Shevtsov
DESY, Hamburg; *IHEP, Protvino; **YerPhI, Armenia

Abstract

DOOCS is a distributed control system that was developed for the HERA and TESLA Test Facility TTF applications. It is an object oriented system design from the device server level up to the operator console. Class libraries were developed as building blocks for device server, communication objects and display components. The whole system is written in the programming language C++. DOOCS has been designed as a stand-alone control system and was extended to allow uniform access to all TTF control systems. The architecture is based on an object oriented application programming interface (API) on the client side that provides multiple protocols. A so called Equipment Name Server (ENS) consists of a central data base with all names and protocols in the system and is consulted by the client programs before the first data transfer to the device. This paper describes the object-oriented architecture of the system and the integration of the subsystems.

1 Introduction

The TESLA Test Facility (TTF) [1] at DESY, Hamburg consists of a linac and associated infrastructure to study acceleration structures based on superconducting cavities. Currently the machine operates with the first prototype acceleration section exceeding the design energy of 125 MeV. The final linac will produce more than 1 GeV and will include a Free Electron Laser (FEL) facility. An international collaboration designs and builds this machine. The components for the accelerator and the control system are supplied by different institutes and therefore demanding the integration of the several different software protocols.

The integration technics used to access the data of the subsystems are a multi-protocol application program interface and different types of gateway servers. Both methods which create a homogeneous system are implemented in the Distributed Object Oriented Control System (DOOCS) [2]. This system is a new development that was started with the design of TTF. The multi-protocol client interface is described in chapter 3, the different gateway methods are described in chapter 6.

2 DOOCS Architecture

The design of DOOCS is based on device objects that reflect the real world hardware devices. This leads to a concept of device servers which handle all properties of a particular device and a single software unit controls a complete device instance. Further devices are then created with the powerful methods of object oriented languages. The same object oriented design concepts are also used in the network communication and in client programs. The whole system is programmed in C++ and makes use of the reusable objects provided by the language.

An important design goal was the implementation of 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 consists of device servers with hardware connections on the lowest level, middle layer servers to implement gateways, special calculations 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 clients. Whenever a server is started, new device instances are created or new properties added, these changes are immediately available on the network; there is no central server 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 CPUs. Even the same type of a device server with different locations is able to run on many CPUs. The system is really distributed without a single point of failure.

The DOOCS client Application Programming Interface (API) is able to handle different network protocols in addition to the DOOCS-RPC protocol transparent to the user. So far the EPICS calls are also supported from within the API. Other systems are connected through gateway servers.

The names (IP-Address) of servers are resolved by an Equipment Name Server (ENS) which again can run on multiple computers. A name query service, which is missing in EPICS, is also provided by the ENS. The communication separates the client programs completely from the device servers. All device specific information is

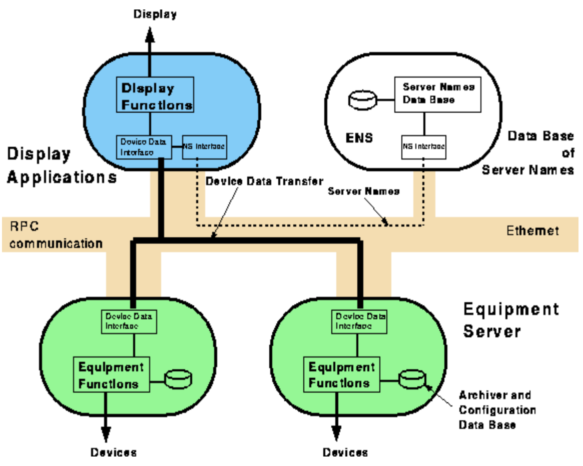

Fig. 1 DOOCS Architecture

hold in the servers. A modification in the layout of a server program has no impact on the client program because of the symbolic access of data.

3 Client Interface (API)

The client part consists of a communication class, an address class and a data class as the client interface plus some internal classes. The data class library has a lot of methods to access, change and convert data and other control parameters. The transferred data also contains type, error and length information. A data item can be a simple integer, float, string or a complex structure of archived spectra or histories or error messages or a list of available services.

With the address class the client program specifies a facility, a location, a device and a property of a device. The 4 parts are ASCII names with up to 16 characters each, the property part can be up to 32 characters long. The addressing of the servers and the handling of the server links is done in the library. A server could run on any computer in the internet. Killing and restarting services are handled in the communication libraries by the automatic creation or recreation of links to these services. The implementation of our communication protocol also allows for reading the parameters of all objects of a certain device server class with just one call. This feature reduces the network traffic significantly and gives the program a chance to read all error flags of one server, for instance, without knowing the actual names or number of objects. There is also provision in the protocol for generalized programs to get the names and properties of all objects in the net. The client communication interface requires only a few calls. These are: two different calls to read data, one to send data and another one to query the actual list of devices and properties. One type of read call implements a blocking, synchronous read with optional data send to the server. The other read call implements a non-blocking monitor call.

The API has access to an Equipment Name Server (ENS), which resolves the necessary information of the servers. These data are stored in a table of the client, therefore the first call to a device server needs a name resolve request to the name server.

Because of the object oriented structure of DOOCS, the general interface is implemented in C++. To incorporate C and LabVIEW applications a C interface was designed on top of the C++ interface. This C interface is used in a set of Virtual Instruments needed for LabVIEW to access all data. A Fortran interface and a library for MATLAB/SIMULINK is also provided.

4 Client Applications

On the client application side DOOCS provides a set of generic and specialized programs and incorporates commercial products. All of these programs use the same API calls to access device data.

rpc_test is a generic program to display and modify all available device data. It reads the actual list of device servers from a name server and the available properties of a certain server from the process itself. Because the program gets all information from the network, all parameters in the system can be read and written without any change in the rpc_test program.

plot_test is a further generic program to display from all devices all data which is plotable. Since it reads the names from the network, as rpc_test does, it allows to show all available data types with historical data or snapshots from scopes for instance. This data may come from actual readings or from an archive.

LabVIEW from National Instruments is used for applications which need to display and control device data. It is useful for measurements including data processing and all kinds of sequencing programs. A Virtual Instrument (VI) library to access device data was created[4].

MATLAB/SIMULINK is a commercial tool which is used to integrate complex RF models with real data from the linac and to control and optimize feedback loops on-line for instance.

DOOCS Data Display program (ddd) [3] which is a graphical editor to create and run control panels. ddd allows to create component libraries in a hierarchical way. The synoptic displays are animated by the status of the devices, subwindows with detailed information or plots are activated by mouse clicks.

save&restore is a tool to read, modify and save groups of device parameters on one page. The tool is configured from files and stores device data in files.

xerror is a tool to display all errors/alarms of device servers including the logfiles of the devices. It continuously scans a list of device servers and displays any error. xerror is configured from a file.

doocsget and doocsput are command-line interfaces to read or write device data and may be used in shell scripts.

knobbox is a hardware device as a user interface to steer any parameter in the system e.g. magnets or RF phase/amplitude by four rotating knobs with a display to show the parameter name and actual setting.

5 Device Server

Device server processes are build from a modular library of C++ classes. An actual server consists of several entries of a device type at different locations in the system. It may contain different device types also. Every instance of a device defines a set of properties. These named properties are the access points on the communication network and are implemented as data objects.

The server library defines a basic class for device servers. This basic class provides the common communication objects like the name, the error status and on/off-line information. The device server inherits this information and adds device specific code and data objects to the standard objects. Data objects are also a part of the library and are defined for status words and bits, correction polynomials, errors, float values, filters, archiver and spectra readings etc. By declaring a data object in a device server it is automatically inserted into the list of named device properties and becomes accessible on the network.

In order to be independent from the Ethernet, a harddisk is attached to every server station. This allows to boot and

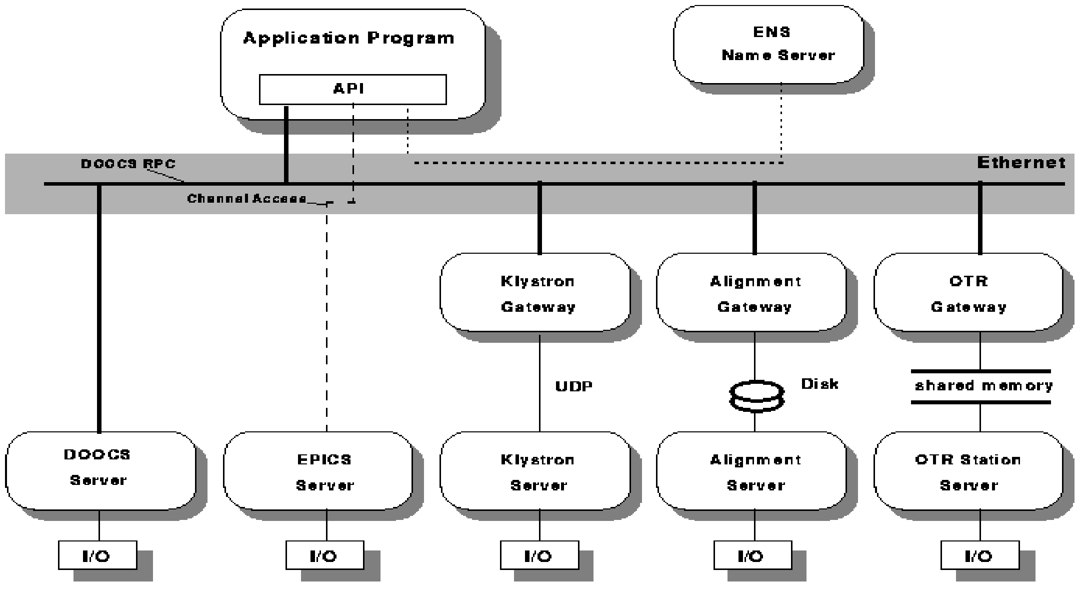

Fig. 2 Integration of the Subsystems.

run individual servers without the network. Because of the local harddisk, archiving history data is done in the server as well. This reduces the network load and allows to analyze problems that happened during network break-down without loosing data. Every device server keeps and permanently updates a local database of its actual configuration and state in order to restore the previous system state after a power failure. Since this data base is maintained locally a server does not depend on the network.

The device servers and middle layer servers in the system are located on SPARCstations, VME-SPARC processors running Solaris 2 and PC's with the LINUX operating system. Most server processes are running on embedded SPARC CPUs in VME crates and talk to various VME - cards via memory mapping or UNIX device drivers. As fieldbuses we support the DESY standard SEDAC, Profibus and CAN.

6 TTF Subsystems Integration

Due to the fact that TTF is build by a multinational collaboration different stand-alone controls were delivered from the other institutes. These subsystems are integrated in different ways. Some examples of integration are presented here:

- The injector is controlled by EPICS [5], which is integrated through our multiprotocol client API. In addition a middle layer DOOCS server is used to archive injector channels.
- OTR screens are supplied with Macintosh computers running LabVIEW. The communication is done through a shared memory in VME with two MACs and one SUN accessing the same data. Commands are transferred via mailboxes.
- The klystrons run a PSOS system and the so called CLASSIC protocol. A gateway server is translating this service into the DOOCS environment. This server also provides archiving.
- The wire alignment system consists of OS-9 front-ends and a Linux data server. A SUN group server is NFS mounting this disk and provides the communication and archiving for this system.

7 Conclusion

Although the DOOCS system was newly developed with very limited manpower, the TTF linac could be commissioned with a full set of software tools in very short time. It took three days to deliver the beam from the injector up to the end of the linac with full design energy.

The TTF control system was developed by five different institutes and five different communication protocols are used. All these subsystems demonstrated successful operation. By means of DOOCS all devices in the subsystems can be accessed from a single program in a transparent way. An operator or a programmer does not have to care about the different protocols and finds the controls to be an integrated system.

REFERENCES

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

[2] G. Grygiel, et al. "DOOCS: A Distributed Object-Oriented Control System on PC's and Workstations", PCaPAC conference, 1996

[3] K.Rehlich, "An Object-Oriented Data Display for the TESLA Test Facility", ICALEPCS 97, Beijing

[4] S. Goloborodko, et al., "Integration of LabVIEW into TTF Control System", Proc. of the XV Workshop on charged particle accelerators, Protvino 1996

[5] F. Gougnaud, et al., The Tesla Test Facility Injector Controls, ICALEPCS 95, Chicago