The COSY Control System
A Computer Distributed Multiprocessor System for Accelerator Control

N. Bongers, U. Hacker, K. Henn, A. Richert, M. Simon,
K. Sobotta, M. Stephan, T. Vashegyi, A. Weinert
Forschungszentrum Jülich GmbH, Institut für Kernphysik, Postfach 1913, D-5170 Jülich, Germany

Abstract

The COSY control system architecture is organized strongly hierarchically with distributed intelligence and extensive use of standards. At the top level of computer control hardware work stations give the operator graphical access to the process. For these tasks Hewlett Packard HP 9000 Series 700 computers with HP-UX and X-Windows/Motif are in use. Also used as workcells this RISC computers give computing power for model calculations and long term databases. This computers are interconnected using Ethernet and TCP/IP to the next layer of hardware: the workcells (Fig. 1).

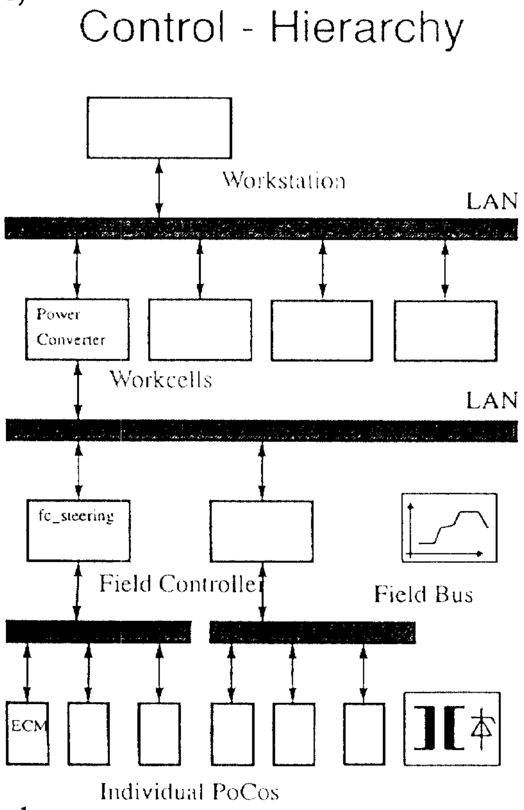


Figure 1

In each subsection a specific Ethernet line connects all field controllers to the corresponding workcell. Disk less VME systems contain the datacom CPU and additional CPUs and I/0 cards. All CPUs are running the realtime operating system RT/OS. For slow I/0 a level of G64 systems is introduced below the VME level and accessed by the PDV-Bus. This gives higher modularity and flexibility in interfacing to geographically distributed devices.

The model of COSY control

The basic idea of the COSY control system is to interpret the whole system as an unique operating system. Every level of the control system is matched to a corresponding level in the operating system. The functional level is assigned to the hardware level. The whole software design is made with this constraint in mind.

The device connectivity

To describe the technology of dataflow the seven layer ISO/OSI model is transformed for device connection. This model fits in a design for all the protocol handling software and the interaction between different places in the control system. This model helps to find the position in the implementation process where different layers interoperate and so different modules are build and connected. This model helps also to identify the positions where protocols will be used and which are the contents of the protocols. This is true for protocol features like address resolution, data validating, handshaking and so on. A second major point is the chance to find with this understanding of protocol states the places in the implementation process where automatic protocol creation tools can be used.

Strategic Design

Keeping all the fundamental ideas in mind we tried to find a very easy analysis of the dataflow between accelerator equipment and the man-machine interface. This analysis covers things like the process picture and a communication tool called CCTP. With these tools the implementation platform for a future extension is given. This extension gives the possibility to connect a knowledge based system. This system needs to handle a formalism which allows functions to analyse individually. The model of data abstraction and data- and function-encapsulation found in the object oriented languages can help to come from the analysis of the system to a design. This design covers all the needs for a later adaption of an expert system (Fig. 2). The models of objects on every layer of the control system is the base for the whole control system. Every object finds a representation of its state and its data in the process picture. The different layers of the control system use all

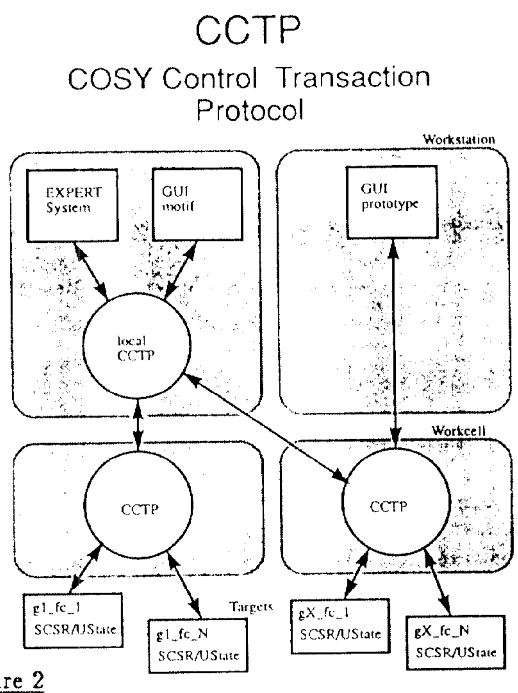


Figure 2

the same data and states but with different models behind it. In the graphical user interface a model simulates the device action and drives a view of this model as graphical representation of the object [1].

Topology of realtime applications

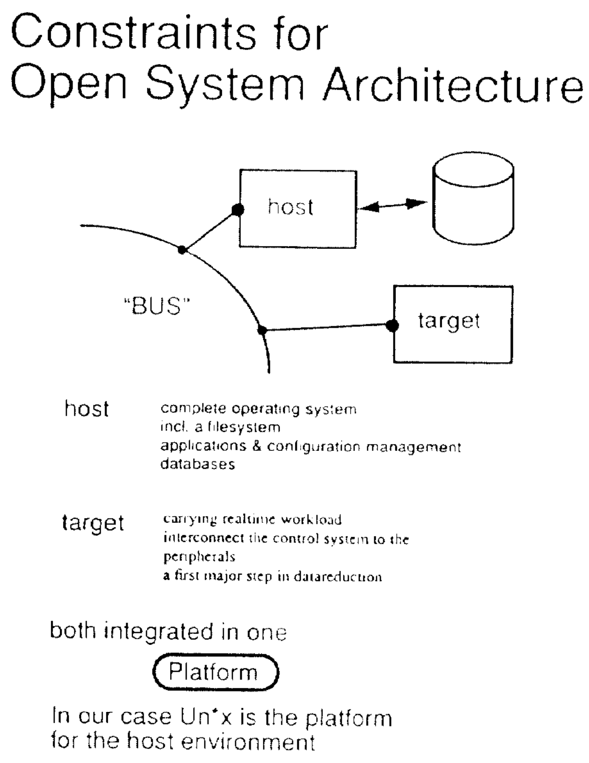
In realtime applications there are different possibilities to design the system. The workload can be carried by one huge minicomputer. Another solution is to split the workload symmetrically. This means every processor in the system does the same job. When the system becomes more complex, the dataprocessing can be achieved in a hierarchical manner. The process I/0 will be done with dedicated realtime systems and the transaction processing will be done with universal computers. This is valid for the operating system, where the process I/0 will be done with dedicated realtime systems and the transaction processing can be achieved by an universal operating system.

Constraints on the operating system software

In our days system software products should be compatible to the different accepted standards. This is true for the operating system, the network components and the graphical user interface. The software development system is based on UN*X, graphics will be done with X windows and networking is still waiting for OSI-protocols using the TCP/IP protocol stack. Realtime system software has to take care of these needs.

The dedicated realtime kernel approach

For complex systems with many separated processors interacting in resolving one automation task, there is a need for a separation of the transaction processing and the realtime part of the overall system. This gives a number of advantages. The computers with a complete operating system including discs are called hosts (Fig. 3). The pure realtime systems are called


Constraints for Open System Architecture

host complete operating system
incl. a filesystem
applications & configuration management
databases

target carrying realtime workload
interconnect the control system to the
peripherals
a first major step in datareduction

both integrated in one
Platform

In our case Un*x is the platform
for the host environment

Figure 3

targets. The advantage of this approach is the easy way to bring targets in operation. These considerations allow small embedded control as well as large VME multiprocessor frames to be handled with one realtime kernel. The COSY control system follows this approach (Fig. 4).

UN*X as host system operating system

As a general operating system UN*X becomes an industrial standard. On an UN*X system all the transaction based tools likes databases are available. For software development all tools for the life cycle are found under UN*X. For an integrated target software development system the programmer will do all tasks under the same system appearence. The revision and backup handling for both sides are done on the host side. No pure realtime load makes the program design and the system tuning tricky. In production phase the overall configuration and application management resides on the hosts (Fig. 5). The target does pure realtime data aquisition and data processing.

The target development system

Development of target software needs a couple of tools. One of the tools is a compiler, which allows to make code for the target system. This can be a native compiler with the feature to make absolute binaries or a cross compiler for the target hardware platform. More

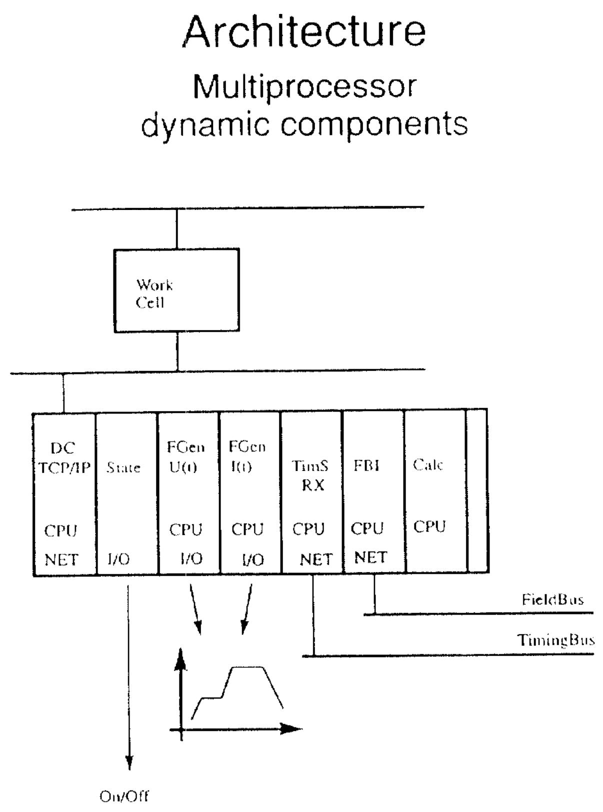


Figure 4

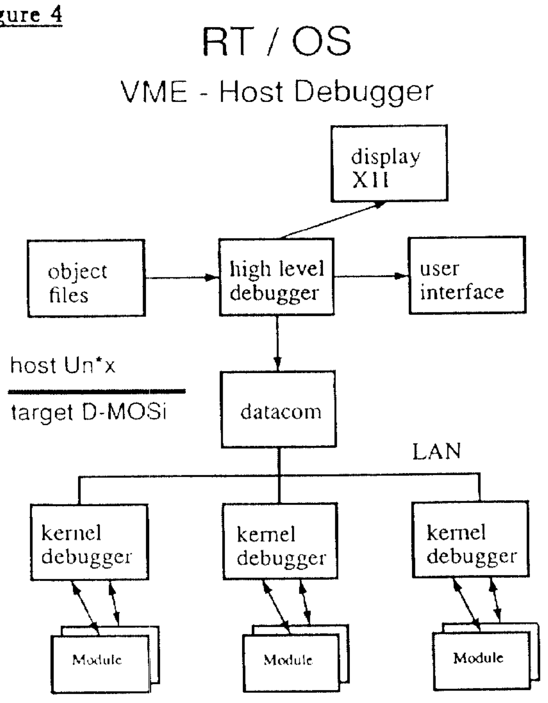


Figure 5

complex is the question of debugging. A remote debugger consists of 2 parts of software residing as well on the host as on the target. Target and host should be connected by different medias like ethernet with TCP/IP. Following this scheme it gives an amplification in handling the target software development process. Only one type of programming tool is needed.

RT/OS the realtime environment

The realtime kernel is developed at Jülich. The interface definition derives from IEEE 855 micro computer system interface [2]. There are a couple of extensions to fulfill realtime need in a multiprocessor system. For many functions there is a BSD 4.3 conformance library, which allows normal programming similar to the development under HP-UX. The networking is achieved with TCP/IP. For process scheduling, interprocess communication and for both the host and the target side symmetrical tools are available. The realtime operating system RT/OS includes also functions to access the configuration database on the base of the major number of a device.

D-MOSI a modular realtime kernel

The design for the realtime kernel is full modular. A system call interface handles the communication and synchronisation aspects between the modules.

The integration of a new piece of hardware into a new device driver and the binding into the kernel is a job for only a few man days. The primary modules are the memory management, the data transfer module, handling the devive driver threads, and the process management. Depending on the application a process communication modul, a network module and a processor to processor communication module can be included.

Conclusions

To build large complex realtime applications including all aspects of dataprocessing, a separation of development and transaction processing on one hand and realtime process I/O on the other hand is needed. Choose a platform like HP-UX, build a software development system on it and this way bring all people to work. The platform has to have an open system architecture including tools like X-windows and networking via TCP/IP. This helps to get rid of the system problems of the beginning 90s [3].

REFERENCES

[1] R. Maier and U. Pfister for the COSY-Team, The COSY-Jülich Project March 1992 Status, this conference

[2] IEEE std 855, IEEE Trial-Use Standard Specification for Microprocessor Operating Systems Interfaces, 1985

[3] U. Hacker, COSY Control System, ICALEPCS'91, Tsukuba, Japan, 1991