ACCELERATOR CONTROL WITH THE LONWORKS FIELDBUS

M. Smolej, B. Jeram, K. Kenda, I. Kriznar, B. Lesjak, M. Perko, U. Platise, M. Plesko;
J. Stefan Institute, Ljubljana, Slovenia, http://kgb.ijs.si

Abstract

The device access layer of the control system of the light source ANKA is almost completely based on LonWorks. We have developed and produced custom I/O boards that use the LonWorks micro-controller (the Neuron). The hardware comprises a high-precision 16-bit DAC/ADC/function generator board, a 40 channel digital I/O+counter board and a serial interface. The device logic has been programmed already at the Neuron level, such that for example power supplies that are controlled either through a DAC/ADC board or through a serial interface look the same on the fieldbus network. The features include state machine, remote command invocation and event driven communication with monitors and alarms. The nodes are automatically configured at start-up time from a PC-resident, version-controlled database for which an ftp-like protocol has been developed. Other tools, which allow for a generic control implementation, are a network node installation and configuration tool, a node inspection and management tool and a template compiler, which allows us to use the same database data on the PC and on the Neuron.

1 INTRODUCTION

This work represents the I/O control sub-unit of the control system for the light source ANKA[1]. The control system is based on PCs, so the appropriate solution for interfacing devices is to use a fieldbus, thereby avoiding the complexity of VME.

We were looking for an integrated solution, which would hide all communication details, so that we could save development efforts. The **LonWorks** technology proved as the most optimum currently available solution for our needs. Its network protocol, which comprises all 7 layers of the ISO/OSI model, reduces network programming to merely formatting the application level packets or even only assigning variables. Each node on the LonWorks network has a special micro-controller – called the **Neuron chip** – with built-in networking, I/O and application functionality. It is programmed with the NeuronC programming language which has, besides normal C syntax, constructs such as declarative syntax for defining I/O objects, network variables and software timers, plus exhaustive run-time library for controlling I/O, network communication, etc. Programming and network communications are event-driven, by simply defining tasks which are to be executed by a built-in task scheduler when the appropriate event occurs.

All these features allowed us to bring much intelligence to the device level instead of just mapping I/O to the PC. We have thus two main processes on different levels: The **device driver** is running on a fieldbus node. It monitors the physical device, generates alarms, sends asynchronous value updates, etc.. The **device server** is running on the PC. It communicates with all device drivers of the same type, manages them, and exports the functionality of their devices as a CORBA server to the rest of the control system.

2 I/O BOARDS

Although many commercial I/O interface boards were already available, we decided to develop our own boards. This decision allowed us to keep the number of different board types as small as possible. A total of three types cover all cases of control system's I/O requirements. All our boards have a MC143150 micro-controller MCU (the Neuron chip) with 24k bytes of SRAM and 32k Bytes of FLASH memory and they are built in small Europa format (160x100 mm). The individual I/O boards are:

Ariadne is a serial interface board, which supports EIA-232, EIA-422 and EIA-485 standards at maximum baud-rate of 115kbps. It has a 16k bytes long buffers on receive and transmit lines and an on-board power supply unit that can source current from 230 V AC line, unregulated DC 7-12 V and regulated 5 V DC.

Hera is a generic digital I/O card with 24 inputs (50 mA), 8 inputs/outputs (50 mA) and 8 solid state relays (1 A). All I/Os are optically isolated. Two operating modes are provided for inputs and input/outputs. There is also a 16-bit frequency counter with range of 0-100 kHz (absolute error 1.53 Hz).

Zeus is a high precision I/O card with a 16-bit ADC and DAC, DAC trigger input, and optically isolated digital channels (8 inputs and 8 outputs). Four analog channels with a nominal sampling frequency of 1 kHz are multiplexed to the ADC, which is oversampled at 4 kHz to ensure 0.3 LSB precision. The DAC operates at a maximum rate of 10 kHz. An additional on-board peripheral micro-controller, specially designed to control booster and storage ring power supplies includes: function generator synchronized with DAC trigger input, 32 kb of memory to buffer DAC function and ADC data, and peripheral self test.

3 SOFTWARE

3.1 Node level

The software controlling the devices in our control system is "device aware" and hides the I/O details from the higher levels. Due to CPU power restrictions of the

Neuron chip, this encapsulation is done only to a reasonable extent, e.g. the conversion to engineering units from hexadecimal values is done on the PC due to poor floating-point performance of the Neuron chip.

Our control system includes over 20 different device types. The requirement for having devices as intelligent as possible increases the amount of device level programming. With all that in mind it makes sense to define a generic API for device driver implementation, which would also include a generic communication interface. With such API we simplify development of new device drivers and also the development of software, which communicates with our devices (in our case device servers running on PC).

The device driver API also allows for implementing multiple devices of the same or different type on the same I/O board, according to our paradigm of "device aware" fieldbus nodes.

A generic device being part of a control system must be able to:

1. receive, process and answer foreign requests
2. propagate monitored parameters and their alarms to the observer
3. device can also have a state machine or additional logic programmed within

With LonWorks, using request/response mechanism for implementing a simple RPC can easily cover the first requirement.

The second is achieved by defining proper network variable types and simply updating the network variables each time the value is read from the device. These network variables are called monitors. An adjustable pair of software timers controls propagation of each monitor variable. "Max timer" defines the heartbeat of the monitor and "min timer" defines minimal time that must pass between two monitor updates. Min timer prevents from flooding the network if the monitored parameter value is changing to fast. Some types of parameters can have a delta threshold defined, which also prevents from too rapid propagation. Each monitor has a complementary network variable called alarm, used to communicate alarms and the values responsible for that alarm.

The monitor and alarm communication is asynchronous since the propagation is triggered by "timer expiration", "change in value" or "alarm condition" events.

The generic device driver library has been written such that any device driver can be compiled into a simulation of the device via a single compiler directive. This is possible due to the small number of different interface boards and exactly defined communications interface. The simulation proved extremely helpful during code debugging and testing.

3.2 PC side communication

Echelon provides LNS (LonWorks Network Services), a platform independent, object-oriented architecture for managing LonWorks networks. Under Windows, there is an ActiveX wrapper of LNS called LCA (LonWorks Component Architecture). Unfortunately, the current version of LCA is too heavily interoperability-oriented. For this reason and to automate and simplify network services, it hides some crucial details from the developer, who is then forced to use the LNS API directly.

At the initial phase of development we used LCA which helped us to deliver the control system prototype in a relatively short period. In second phase we imported the explicit messages functionality and needed the LNS layer for that. Now, we have completely avoided the LCA and thus have gained the full control over the network services and also avoided the overhead of ActiveX.

3.3 Device configuration

In order to eliminate hard-coded constants and thus make the device driver flexible we download configuration parameters to the device. For that purpose we implemented a custom windowed file transfer protocol, similar to one described in LonWorks Engineering Bulletin [2]. Using FTP we are also able to transfer large amounts of data.

Each time the device server is started it checks the node if it has a valid running application. If not, the executable path is found in the database and application downloaded to the node. Node's configuration files are updated if needed.

The configuration data is stored in a database, which is used by both device driver and device server. All the data are stored in engineering units, which must be converted into raw format before being used by Neuron application. To simplify device configuration process and to eliminate occurrence of the same configuration constant in multiple places within the database or in multiple formats, a template compiler has been written. This compiler takes the original header file from the NeuronC application and compiles it into a template, which is then filled with database data and converted into a binary version. This binary version can be downloaded via FTP and copied directly into the same data structure that was used to create the template. So we have a generic way for converting database data into a Neuron application recognizable format.

The FTP feature was implemented by providing an FTP server on the PC who can handle up to 256 multiple transactions, running in concurrent or sequential mode. Integrated notification mechanism allows for handling FTP notification events. Transactions can be started or aborted locally using PC API or remotely by Neuron application.

A good example of using FTP and template compiler for configuration is mapping logical devices to I/O pins on the board. Those maps are part of device's configuration file, which is generated by template compiler and downloaded if needed each time the device server starts.

4 PERFORMANCE

The most demanding case during control of ANKA is when the maximum number of possible nodes (64) on one fieldbus branch send regular updates of 3 parameters at 1 Hz and simultaneously one parameter of one board is set and read-out at 20 Hz. This translates to 212 (3*64+20) incoming unacknowledged packets and 20 request/response packets per second.

Measurements showed that our system is able to handle up to 230 incoming unacknowledged packets per second (which is also the upper limit of “throughput/offered traffic” linearity) and can transmit up to 40 request/response packets per second, i.e. more than we need. Even if the nodes produce too much traffic on the network, the throughput doesn’t fall to zero, but stays at the level of approx. 190 packets/s. The average packet size in the test setup was 12 bytes.

Our test setup included 11 Neuron nodes all running a simulated power supply device. Nodes were connected via 1.25Mbit twisted pair transceivers and Echelon’s PCNSI card was the interface to the PC. The measurements showed that this interface is the bottleneck for data throughput.

5 CONCLUSIONS

The ANKA control systems is now composed of more than 200 Neuron nodes. For efficiency and topological reasons, there are 7 independent branches each connected to a PC. The fieldbus system runs absolutely stable without interference, even when the rest of the controls is being shut down or restarted.

REFERENCES

[1] M. Dach et al, A Control System Based on Web, Java, CORBA and Fieldbus Technologies, PCaPAC99 workshop, Tsukuba, January 1999.

[2] LonWorks Engineering Bulletin, EB176, Echelon Corporation, November 1996