Data Acquisition Server for Mini Distributed Control System

Dariusz Rzońca, Andrzej Stec, and Bartosz Trybus

Rzeszow University of Technology, ul. W. Pola 2, 35-959 Rzeszów, Poland
{drzonca,astec,btrybus}@kia.prz.edu.pl
http://www.prz.edu.pl

Abstract. The paper describes PACQ, a distributed control mini system. WWW and data acquisition functionality of PACQ server is presented from the user perspective together with a discussion on security issues. The communication between the server and control modules is modelled with hierarchical timed coloured Petri net. Programmability of the control modules using CPDev environment and their system software is also characterized.

Keywords: CPDev, DCS, HTCPN, Petri nets, data acquisition.

1 Introduction

PACQ is a prototype small distributed control and measurement system (mini-DCS) developed in Rzeszow University of Technology. It consists of multiple control modules and the main data acquisition and WWW server (Fig. 1). The control modules equipped with inputs and outputs are programmed in CPDev environment in ST, FBD or IL languages [1]. Process variables are exposed by the modules to PACQ server via a custom, Modbus-like communication protocol. The collected data is stored in the server database. WWW interface is used to access the data from the Internet.

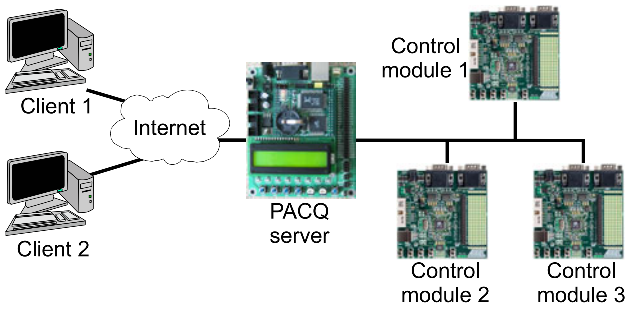


Fig. 1. Structure of PACQ distributed system

2 PACQ Data Acquisition and WWW Server

PACQ server (*microserver*) is used to record measurement data transmitted from the control modules via RS-485 interface. A dedicated master-slave protocol similar to Modbus is used to communicate with the modules, as described in Sect. 4. The collected data can be viewed using a web browser.

PACQ server prototype has been developed on EVBnet02 evaluation board (Fig. 1, center) [2]. The board is equipped with ATmega128 microcontroller, RAM and DataFlash memories, 10 Mbit/s Ethernet controller, UART, RTC, LCD display and 4 buttons. WWW and FTP services run under Nut/OS operating system [3].

Software for PACQ server has been written in C. The application part consists of three major components (Fig. 2):

- serial communication subsystem for acquiring data from control modules,
- archiving service for storing the data in a database,
- web server CGI extensions generating dynamic web pages.

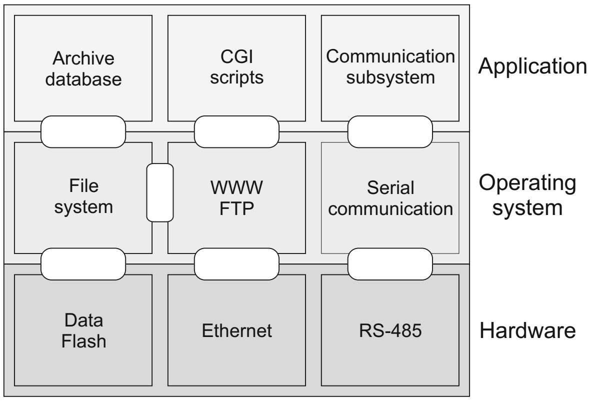


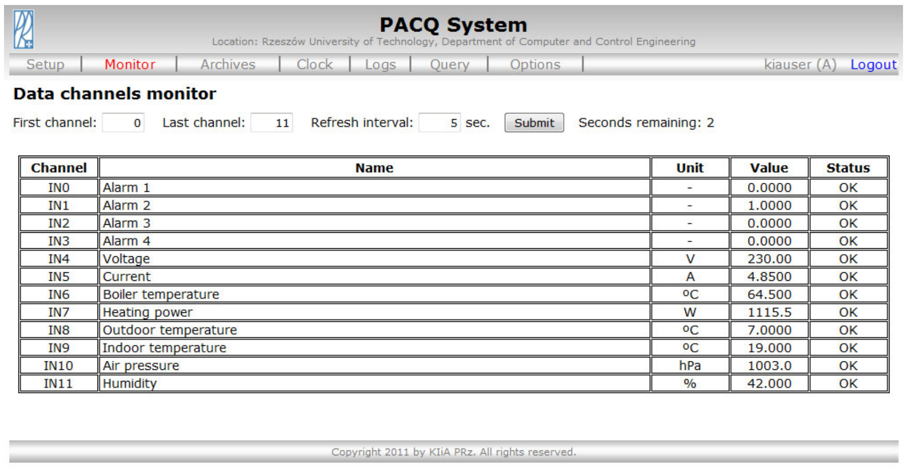
Fig. 2. Software and hardware components of PACQ server

Web pages are written in HTML with addition of JavaScript and CSS. Dynamic web pages are generated by CGI scripts. GET and POST methods are used for processing HTML forms. Because of limited performance of the server as well as to reduce the size of transferred pages, the server sends raw data from the database to the browser. The final processing is done on the client by JavaScript.

PACQ server web page can present both current (on-line) and archived data (Fig. 3). The server periodically acquires data from control modules and stores it in the archive database located in DataFlash memory. The update interval is configured independently for each module.

The most important features available to the web user are:

- monitoring of current data,


PACQ System
Location: Rzeszów University of Technology, Department of Computer and Control Engineering
Setup
Monitor
Archives
Clock
Logs
Query
Options
kiauser (A)
Logout
Submit
Copyright 2011 by KIiA PRz. All rights reserved.

Data channels monitor

First channel: 0 Last channel: 11 Refresh interval: 5 sec. Seconds remaining: 2

Channel	Name	Unit	Value	Status
IN0	Alarm 1	-	0.0000	OK
IN1	Alarm 2	-	1.0000	OK
IN2	Alarm 3	-	0.0000	OK
IN3	Alarm 4	-	0.0000	OK
IN4	Voltage	V	230.00	OK
IN5	Current	A	4.8500	OK
IN6	Boiler temperature	ºC	64.500	OK
IN7	Heating power	W	1115.5	OK
IN8	Outdoor temperature	ºC	7.0000	OK
IN9	Indoor temperature	ºC	19.000	OK
IN10	Air pressure	hPa	1003.0	OK
IN11	Humidity	%	42.000	OK

Fig. 3. Browsing data on the PACQ web page

– browsing archived data,
– customizing the user interface,
– managing of the server configuration,
– setting the real-time clock (RTC),
– viewing server logs.

A user enters login information (username and password) on the startup page. According to the granted privileges, appropriate pages are presented by the browser. A common user is allowed to view the data, while an administrator is also able to modify some parameters of the system. For example, the configuration and RTC management is available only for admins.

Moreover, some settings of the system can only be changed by a special management software. Due to security issues, the software communicates with the server locally (via the serial port). FTP protocol can be also used to modify configuration files. However, this possibility is blocked by default. Unblocking requires pressing a button on PACQ board and lasts only for a short period of time (e.g. 10 minutes) or until the server reset.

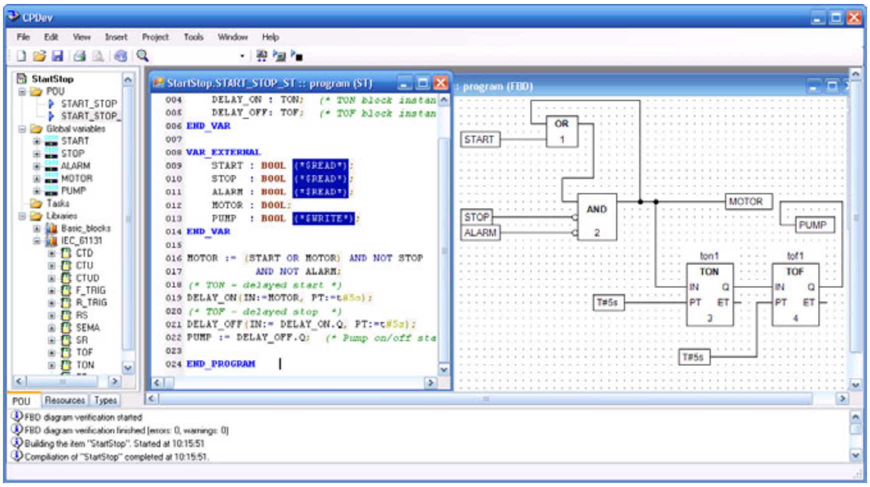
PACQ server has been designed primarily for local networks (LAN). Such networks are the most common in the industry. If remote access is needed to a particular network within a plant, creating a secure VPN tunnel is the preferred solution to get sufficient security.

3 Control Modules

Control modules in PACQ system are small programmable controllers equipped with analog or binary inputs and outputs. The module prototype is based on

ARM7 microcontroller development board AT91SAM7S-EK from Atmel [4] (Fig. 1, right). The board has been equipped with extra I/O hardware and RS-485 bus.

Each module can be programmed to perform process control. Programs are written in CPDev (*Control Program Developer*) engineering environment developed in Rzeszow University of Technology [1]. CPDev integrates tools for programming, simulation, hardware configuration, on-line testing and running control applications. Programs can be written in ST and IL textual languages (*Structured Text, Instruction List*) and in FBD graphical one (*Function Block Diagram*), according to IEC 61131-3 standard [5]. Main window of CPDev environment with programs in ST and FBD is shown in Fig. 4



```
    DELAY_ON : TON;   (* TON block instan
    DELAY_OFF: TOF;   (* TOF block instan
END_VAR

VAR_EXTERNAL
    START : BOOL (*$READ*);
    STOP  : BOOL (*$READ*);
    ALARM : BOOL (*$READ*);
    MOTOR : BOOL;
    PUMP  : BOOL (*$WRITE*);
END_VAR

MOTOR := (START OR MOTOR) AND NOT STOP
            AND NOT ALARM;
(* TON - delayed start *)
DELAY_ON(IN:=MOTOR, PT:=t#5s);
(* TOF - delayed stop  *)
DELAY_OFF(IN:= DELAY_ON.Q, PT:=t#5s);
PUMP := DELAY_OFF.Q;  (* Pump on/off sta

END_PROGRAM
```

```
FBD diagram verification started
FBD diagram verification finished [errors: 0, warnings: 0]
Building the item "StartStop". Started at 10:15:51
Compilation of "StartStop" completed at 10:15:51.
```

Fig. 4. Main window of CPDev programming environment

Programs for PACQ control modules are able to process several elementary data types defined in the IEC standard i.e. BOOL, BYTE, SINT, INT, WORD, DINT, LINT, DWORD, LWORD, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME. The elementary types may be used to define derived types such as arrays and structures.

Data can be processed with functions defined in IEC standard. Examples are:

– numerical functions: ADD, SUB, MUL, DIV, SQRT, ABS, LN,
– Boolean and bit shift functions: AND, OR, NOT, SHL, ROR,
– selection and comparison functions: SEL, MAX, LIMIT, MUX, GE, EQ, LT,
– type conversions: INT_TO_REAL, TIME_TO_DINT, etc.

A portion of code can be used as a *function block*. The CPDev package provides two standard libraries containing IEC-61131 blocks (flip-flops, counters, timers, edge detectors) and basic blocks (pulsers, filters, signal analysers). User-developed functions, function blocks and programs can be stored in custom libraries. *Native blocks* are components of the user program providing hardware dependent functions internally. They are written in C/C++ and linked to CPDev. A library of such blocks has been created for PACQ modules providing read/write of program variables into non-volatile memory and handle I/O.

CPDev programs are compiled into the intermediate, universal code executed by the runtime interpreter, called *virtual machine* [1]. The machine is written in C, so it may run on different hardware platforms, from 8-bit microcontrollers up to 32/64-bit general purpose processors. It has also been adapted to PACQ control modules with their ARM7 microcontroller.

Software for the control modules is run under FreeRTOS real-time operating system [6]. FreeRTOS is an open source, time-sharing multitasking operating system, what means that the CPU time is shared among running processes. It is also preemptive, so if a higher priority process becomes ready to run, the currently running process is suspended.

FreeRTOS in PACQ modules runs several tasks (processes) including:

– CPDev virtual machine executing control programs,
– communication subsystem exchanging data with PACQ server,
– handlers of native blocks.

FreeRTOS tasks exchange data via *queues* and use *mutexes* for mutual exclusion when accessing shared resources such as virtual machine data area [6]. The machine control cycle is kept constant by FreeRTOS task delay facility.

4 HTCPN Model of PACQ Communication

Hierarchical Timed Coloured Petri Net (HTCPN) [7] model of communication between PACQ server and control modules is shown in Fig. 5. Upper left part of the model represents PACQ server communication subsystem, upper right part is related to the transmission link, and bottom part models behaviour of the control modules.

Communication between PACQ server and the control modules relies on a custom master-slave protocol. The server is continuously acquiring data from the slaves. Although common Modbus protocol could have been used, the customized one turned out to be better suited, because the control modules provide not only the actual values of data, but also their timestamps.

In every communication cycle, the master (PACQ server) chooses next slave to read process variables from. This behaviour is modelled by *Counter* place and related inscriptions on the arcs. This place stores the number of the next message to be sent. According to the value of the `l` variable, the appropriate message is chosen from the *Messages*. When the transition *Sending* fires, the selected message is put to the *Outgoing buffer* and the value of the token stored in the *Counter* is appropriately changed. Additionally, a token is put into *Waiting for response* place, with timestamp delayed by `tout` units of time. Thus, *Timeout* transition will not be ready until this time passes. However, the transition *Microserver processing* can be fired earlier (immediately after receiving the

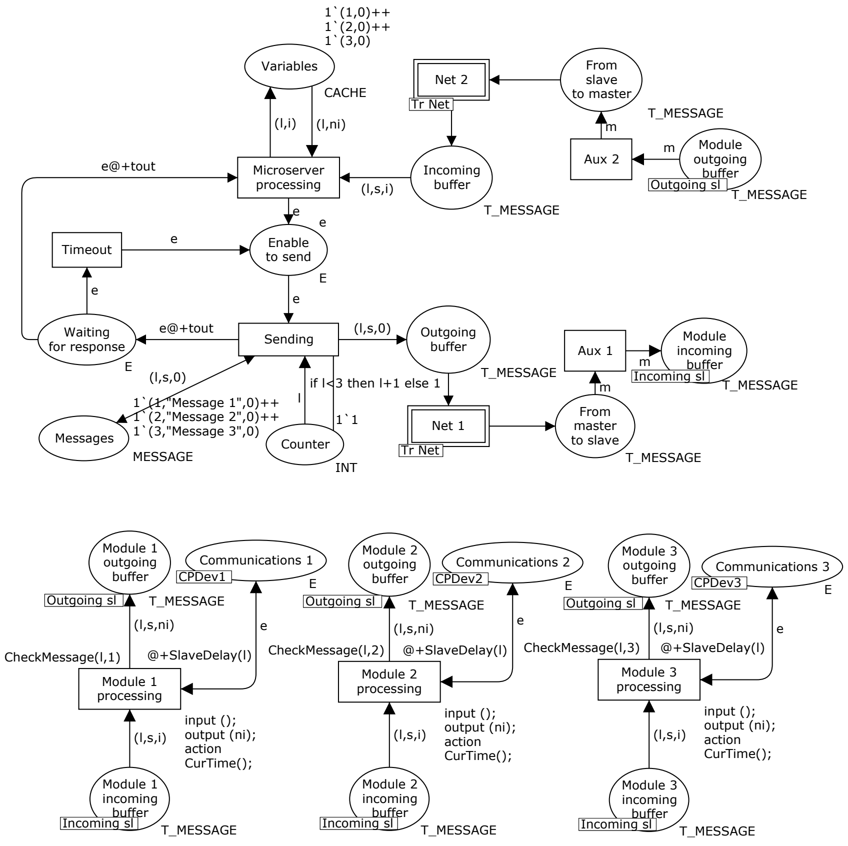


Fig. 5. HTCPN model of PACQ communication

response token in the *Incoming buffer*), because the inscription on the arc from *Waiting for response* to *Microserver processing* allows this token to be consumed without delay.

The message put into the *Outgoing buffer* is transmitted to the slaves. The transition *Net 1* is a substitution transition, introducing to the model hierarchy an instance of *Tr net* subnet (a subpage). This way many different models of the transmission link can be used if necessary, without a change of the main superpage. It has been shown in [8], that various parameters of the transmission link can be easily modelled in such subnet, e.g. additional delay (transmission time) related to the message type, or a packet loss with assumed probability. Similar technique has been applied here.

The message which has been successfully transmitted through the network is put to *From master to slave* place, and finally, by the auxiliary transition *Aux 1* to the *Module incoming buffer*. This place is a *fusion place*, tokens here are shared by all *Module 1-3 incoming buffer* places (lower part of Fig. 5), belonging to the

Incoming sl fusion. Thus many slave modules can be easily modelled, although only three of them are presented here for simplicity.

Function `CheckMessage` in the guards of *Module 1-3 processing* transitions enables only one transition, depending on the target of the message. This function is defined as follows:

```
fun CheckMessage(no:INT, addr:INT) = (no=addr)
```

As a result only the module addressed by the message is able to respond. However, the transitions *Module 1-3 processing* will not fire unless a token in the appropriate *Communications 1-3* place is put. The place (fusion) is connected to CPDev virtual machine model (not shown, see [8]). Each control module involves the machine to execute programs so receiving the message will cause reading process variables. However, this is possible only during time windows i.e. in every cycle after the program execution. Consequently, processing of the message is delayed until such time window is reached. Additional delay related to the processing time is modelled by the `SlaveDelay` function, when the transition *Module 1-3 processing* fires.

Besides modifying the timestamp of the token, the transition records *current model time* for further analysis. The value represents timestamp of the response, i.e. the point in time when the values of process variables have been determined and considered as valid. This timestamp is held in `ni` part of the token put into *Module 1-3 outgoing buffer* place. Its value is calculated by the following function:

```
input ();
output (ni);
action
CurTime();
```

where the `CurTime` is defined as:

```
fun CurTime() = IntInf.toInt(time())
```

The response from the *Module outgoing buffer* through the auxiliary transition *Aux 2* reaches the place *From slave to master* (upper right part of Fig. 5). Subsequently the response is transmitted through the *Net 2* (another instance of the *Tr net* subnet mentioned earlier) to the *Incoming buffer*. When *Microserver processing* transition fires, the timestamp of the received response (`i` part, not the timestamp of the token) is recorded in the *Variables* place. Additional token is put into the place *Enable to send*, activating *Sending* transition. This way another message to the successive module can be handled.

5 Model Analysis and Results

Simulation of HTCPN model can be used for simple performance analysis at the design stage. Therefore potential problems, like bottlenecks can be predicted and avoided. One of the performance parameters, which can be estimated on the basis of the described model, is latency time of on-line variable values.

As mentioned, PACQ server periodically acquires data from the control modules. After every readout, the *variable buffer* in the server is updated to store current values. However, when on-line data view is presented on the server web page, values of the variables from the buffer may be slightly outdated. HTCPN model can be used to estimate maximum and typical delay according to the number of control modules, transmission time, program cycle, update interval, etc. helping to achieve optimal performance. Such analysis has been performed to determine average latency time depending on the number of connected control modules and CPDev program execution time t_{prog} in each module. It has been assumed for simplicity, that CPDev program cycle time t_{cycle} is equal in each module and set to 200 ms. The simulation results of the model are shown in Table 1.

Table 1. Average latency of variable values

	t_{prog} [ms]				
No. of boards	20	60	100	140	180
3	43	52	80	113	160
5	62	84	121	185	261
7	84	112	165	250	362

The results have been compared to theoretical calculation. It has been stated in Sect. 4 that each control module processess the received message only during time window after the program execution. CPDev program cycle time t_{cycle} is set by the user during development of the program. However, actual CPDev program execution time t_{prog} is shorter, and may vary between cycles. The virtual machine has to wait before starting the next program cycle to keep CPDev program cycle constant. The probability that the message will be received during program execution is $\frac{t_{\mathrm{prog}}}{t_{\mathrm{cycle}}}$. Processing of such messages will be delayed by average time of $\frac{t_{\mathrm{prog}}}{2}$. The message will be received during the time window with the probability of $\frac{t_{\mathrm{cycle}}-t_{\mathrm{prog}}}{t_{\mathrm{cycle}}}$. In this case, the machine will be ready for answering and the message will be processed immediately.

Thus average delay time caused by a module not being ready t_{wait} is:

$$t_{\mathrm{wait}} = \frac{t_{\mathrm{prog}}^2}{2t_{\mathrm{cycle}}} \quad . \tag{1}$$

The average time used by the server to handle each module will be denoted $t_{\mathrm{avg\_1}}$. Apart from t_{wait}, it should also incorporate the delay caused by the network (transmission time) t_{net} and the message processing time t_{proc}. The components have been assumed equal for all modules and messages for simplicity.

$$t_{\mathrm{avg\_1}} = t_{\mathrm{wait}} + 2t_{\mathrm{net}} + t_{\mathrm{proc}} \quad . \tag{2}$$

Total communication time (i.e. time of acquiring all variables from n control modules) $t_{\mathrm{avg\_n}}$ will be n times larger. Average latency of variable values in local buffer t_{lat} will be:

$$t_{\mathrm{lat}} = \frac{t_{\mathrm{avg\_n}}}{2} + t_{\mathrm{net}} \ . \tag{3}$$

The latency calculated from the equation above corresponds to the simulation results shown in Table 1. The differences are caused by additional parameters introduced into the HTCPN model but omitted in the above calculations, like transmission time depending on message size, processing time modelled by the `SlaveDelay` function (Fig. 5), etc.

6 Summary

New distributed control mini system PACQ has been described in the paper. Central unit of the system is PACQ server with data acquisition and WWW capability. The control modules can be programmed in IEC-61131 languages in CPDev engineering environment.

Formal HTCPN nets have been used during development to model the communication between the server and control modules. Simulation and performance analysis of the model have been made with CPN Tools [9]. The results have been compared to theoretical calculations.

Future extensions of the PACQ system are developed to provide secure connection between PACQ server and a web browser.

References

1. Rzońca, D., Sadolewski, J., Stec, A., Świder, Z., Trybus, B., Trybus, L.: Mini-DCS System Programming in IEC 61131-3 Structured Text. Journal of Automation, Mobile Robotics & Intelligent Systems 2(3), 48–54 (2008)
2. EVBnet02 User's Manual, `http://www.propox.com/download/docs/EVBnet02_en.pdf`
3. Nut/OS Software Manual, `http://www.ethernut.de/pdf/enswm28e.pdf`
4. AT91SAM7S-EK Evaluation Board User Guide, `http://www.atmel.com/dyn/resources/prod_documents/doc6112.pdf`
5. IEC 61131-3 Standard: Programmable Controllers. Part 3. Programming Languages, IEC (2003)
6. Barry, R.: Using the FreeRTOS Real Time Kernel – A Practical Guide
7. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Springer, Heidelberg (1997)
8. Rzońca, D., Trybus, B.: Hierarchical Petri Net for the CPDev Virtual Machine with Communications. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) CN 2009. CCIS, vol. 39, pp. 264–271. Springer, Heidelberg (2009)
9. CPN Tools: Computer Tool for Coloured Petri Nets, `http://www.daimi.au.dk/cpntools`