Control and Cybernetics

vol. **39** (2010) No. 4

Response–time analysis of a CAN network used for supervisory control and diagnostic systems*

by

Jan Werewka[1] **and Mirosław Dach**[2]

[1] AGH University of Science and Technology
Departament of Automatics, Computer Science Laboratory
Kraków, Poland
[2] PSI – Paul Scherrer Institut, Villigen, Switzerland
e-mail: werewka@ia.agh.edu.pl miroslaw.dach@psi.ch

Abstract: This paper refers to construction methods of Supervisory Control and Diagnostic Systems (SCDS) based on a CAN bus. Emphasis is put on the construction of such systems. Meeting real time constraints has become one of major aspects of this elaboration.

The study enumerates possibilities of using various communication mechanisms of CAN, applicable for real-time systems. Proposed solutions are especially suitable for general-purpose SCDS. Principles of real-time analysis for CAN bus based systems are discussed as well.

An example of SCDS implementation for a Swiss Light Source (SLS) large scale accelerator project is given. Time analysis is made for the constructed system with real-time conditions met. Experiments performed in the SLS confirm the results of the analysis.

Keywords: real time systems, scheduling, distributed systems, fieldbus, real time analysis, CAN, CANopen.

1. Introduction

Depending on application area, computer control systems have specific reliability requirements. No matter how many sophisticated computer techniques for constructing control systems have been applied, there always exists some risk of a break-down situation. Here, the system reliability is understood as probability that the system will perform some specified functionality in particular operational and environmental conditions over a definite time period. However, the more complex the control system is, the more probable is its malfunction. With respect to modern control systems, this applies both to hardware and software.

*Submitted: April 2008; Accepted: August 2010.

One of the ways of improving control system reliability parameters, in view of idle time minimization, is a Supervisory Control and Diagnostic System (SCDS). The SCDS is a real-time distributed system supervising a subordinate control system. The SCDS belongs to a sub-class of control and diagnostic systems implementing simple control-diagnostic functions.

The concept of SCDS is similar to the subordinate distributed control system (DCS). The SCDS should be highly reliable (more reliable that the subordinate control system) and deterministic.

The SCDS usability is assessed on the basis of reaction-time to certain events. The aim of this paper is to present the principles of time analysis of the proposed SCDS based on CAN bus.

2. The SCDS properties

One of fundamental quality indexes, typical for real-time distributed systems, is the reliability of the system. Two basic solutions considerably improve the reliability requirements (see Fig. 1):

- Redundant control systems
- Supervisory control systems.

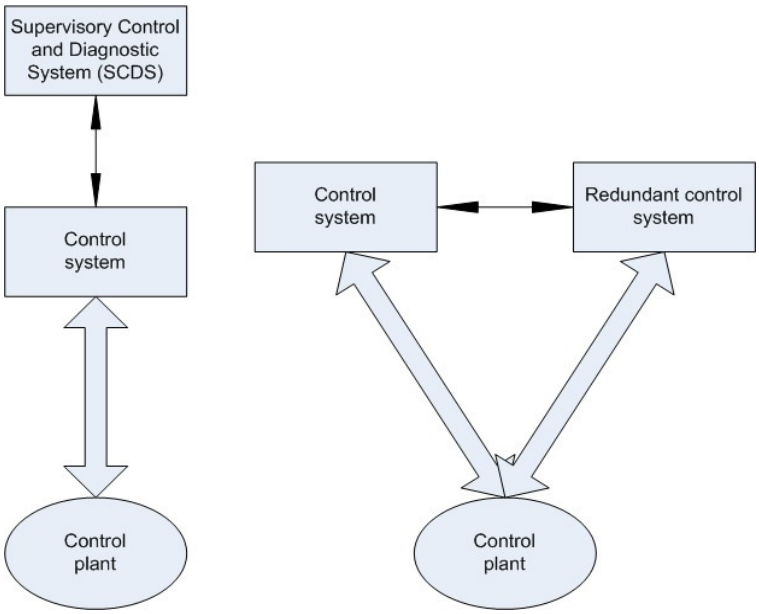

Figure 1. Architecture of supervisory and redundant control systems

Redundant control system duplicates basic controls of the primary system, taking over its functions in case of failure. In the areas where safety constitutes

priority, multilevel redundant control systems are used, e.g. in aviation - in addition to basic control system an auxiliary one is applied, and when it breaks down, an emergency system takes over. The auxiliary control system is usually a functional copy of the primary system, whereas the emergency control system is its simplified version that provides basic required functions only. Although redundant control systems considerably increase reliability of the whole system, they are quite expensive.

An alternative for the redundant control system is a supervisory control system, whose task is to monitor control system operation, and take simple decisions in case of failure. The SCDS surpasses the redundant control systems because of a lower cost.

Unlike redundant control systems, the SCDS does not take over the functions of the proper system in case of failure; its functionality is reduced to monitoring and control the DCS system. The SCDSs are used only in situations which are not catastrophic, e.g. for such distributed automation systems as experimental aerodynamic tunnels, treatment and distribution networks, watering systems for agriculture, etc.

The SCDS is a real-time distributed control system performing simple control-diagnostic functions. Unlike distributed computer control system, its real-time equivalent has the following properties:

- It diagnoses merely the critical physical parameters of the control system to recognize failure states.
- It sends mainly "basic" control signals of, e.g. on/off, initiate type, etc., which considerably influence the DCS operation. These signals are the responses to diagnosed failure states of the DCS.
- It ensures hardware and software independence of the DCS with respect to the control layer. Failure of the DCS does not disturb the SCDS operation. The SCDS and the DCS should not use the same resources, e.g. power, so they do not affect each other.
- It enables integration of the two systems at the visualization layer. So the same user interface can be used for both systems, thus reducing the cost of the whole system.
- Low cost of the system is assumed. Cost criterion is an important factor of SCDS application as compared to the DCS. The SCDS should be considerably cheaper than the DCS.
- It meets predictability requirements (determinism) of the system. Basic function of the SCDS is to monitor DCS operation, therefore the SCDS should react quickly in a predictable time horizon.
- It is highly reliable, at least by one order over the DCS.
- Broadening of the SCDS control-diagnostic properties should not require constructing the system anew. The DCS extension by additional function blocks should easily imply the development of SCDS.

3. SCDS architecture

The SCDS is expected to perform simple control-diagnostic functions only. A generalized three-layer model, assumed for the SCDS, see Meinert (1995) (Fig. 2), consists of visualization layer, data transfer layer and control layer.

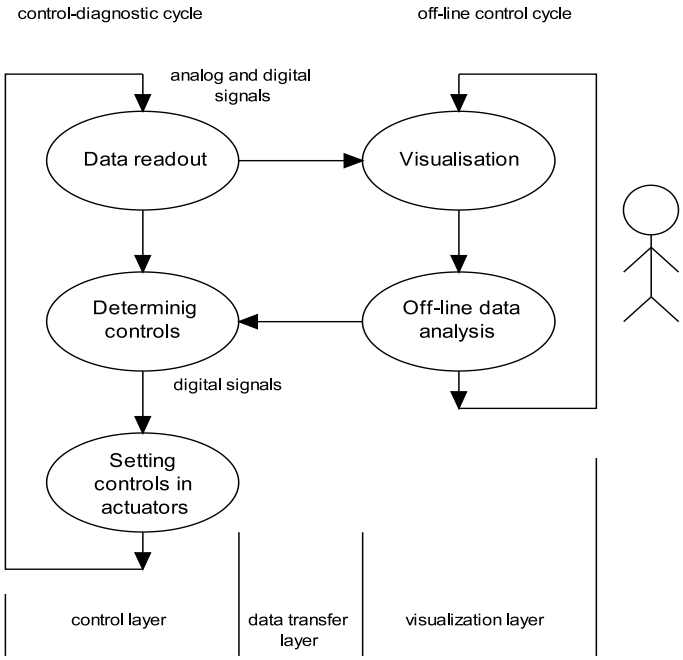

Figure 2. A generalized model of the SCDS operation

The system operation can be presented in the form of two cycles: control-diagnostic cycle (control layer) connected with off-line control cycle (visualization layer). At the initial stage, analog and digital data are read from the sensors providing information on operational state of the DCS. The data are gathered, processed, and transmitted to visualization layer, for the operator (off-line processing).

Control is determined on the basis of data obtained from sensors and operator settings. It has the form of digital signals switching on or off the critical parts of the DCS.

Functional outline of the SCDS architecture can be created on the basis of the three-layer model. The visualization and transport layers assure interaction between the operator and control layer. The SCDS control layer is most important and has the following tasks:

- Controlling and monitoring given physical parameters. This is performed by execution processes.
- Determining controls on the basis of physical parameters and parameters set by the operator. This is performed by the control process.
- Making data available to the operator for monitoring physical parameters.
- Acquiring current control parameters for the control process from the operator.

4. Time analysis of a CAN bus

A distributed control system consists of three layers: visualization, (data transfer) transport and control layer. Control layer of SCDS consists of two sub-layers: control sub-layer and execution sub-layer. The structure of SCDS is presented in Fig. 3.

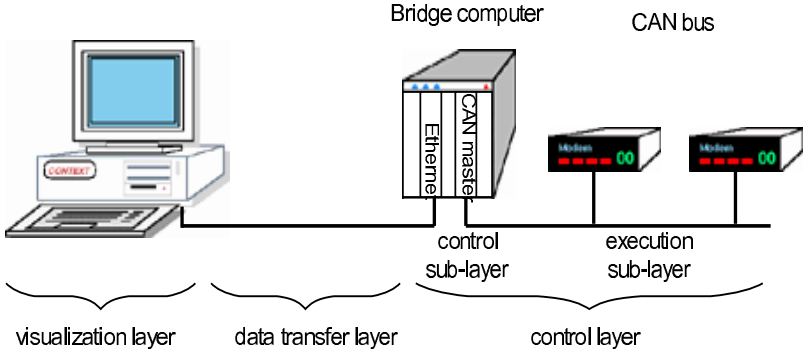

Figure 3. Layered model of distributed control system

This paper presents an SCDS implemented with CAN fieldbus (CAN Specification, 1991; Pfeiffer, 1994; Sinitean, 1996; Zuberi and Shin, 1996; Thomesse, 2005; Cavalieri, 2005).

The CAN standard defines a communication protocol, by means of two lowest sub-layers of the ISO/OSI communication model: data interface sub-layer and physical network sub-layer. These layers are predefined in ISO - DIS 11989 and ISO - DIS 11519 for the CAN. Data connection layer from the ISO/OSI model is additionally divided into Logical Link Control (LLC) sub-layer and Medium Access Control (MAC) sub-layer. The physical medium is accessed in CAN by means of the Carrier Sense Multiple Access/Bitwise Connection (CSMA/BC) method. When messages are sent simultaneously by some nodes, only one of them has the right to transmit the whole message. In order to determine which node accesses the transmission media, the arbitration field of the messages is used. Message identifier located after the Start of Frame (SOF) is the most important component of the arbitration field (Fig. 4). Network nodes which "lost the competition" stop to transmit data and switch into high impedance state.

CAN messages are sent without destination address, therefore each node reads all data transmitted through the network. Messages received by the nodes are recognized on the basis of the ID identifier. The identifier field contains 11 or 29 bits (Fig. 4). Each node has so-called acceptance filter, which compares 8 oldest bits of the identifier with some filter mask. Messages received by the node, when meeting meet the mask condition, are transmitted further to the CPU node where they are processed. Remaining messages are rejected. The ID is also used to determine access priority.

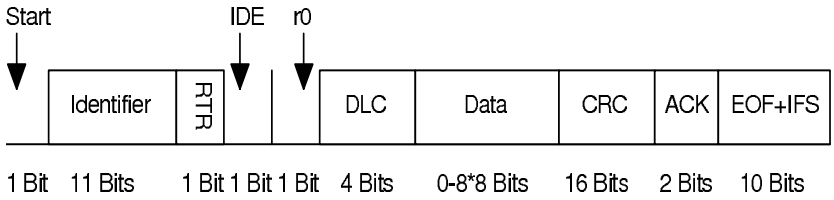

Figure 4. Data frame of CAN 2.0 A

ID with the lowest bit representation has the highest priority. Additionally, four identifier bits at positions 7 - 10 determine message priority within the node, i.e. 0 - 15 (0 - highest priority, 15 - lowest priority). If there are several messages in the buffer ready to be sent, the highest priority message is sent. Three types of frames for sending messages are provided by CAN: data, error and overflow frame.

The data frame (Fig. 4) begins with the Start Of Frame (SOF), followed by 11 bit identifier. Next comes the 7 bit control area (RTR, IDE, r0, DLC). The Remote Transmission Request (RTR) bit determines whether the message is an information frame (RTR = 0), or a request for data from the network (interactive frame). Data field contains up to 8 Bytes. Remaining fields in the frame are generated by the CAN controller: 16 control bits of Cyclical Redundancy Check (CRC) and confirmation bits. The confirmation bits are generated by the reception node to confirm transmission of the correct message (ACK field). Each frame is terminated by 7-bit End Of Frame (EOF). Next frame can be sent after IFS (Inter Frame Space) occurrence.

Time analysis of fieldbuses is performed with respect to message ordering. As far as the ordering is concerned, the Medium Access Control (MAC) layer of the ISO/OSI model is the most important for fieldbuses. This layer determines the conditions under which specific nodes get access to the transmission medium. Due to the type of access to the medium, CAN may be treated as a centralized system of task (message) ordering.

Existing mechanisms of message control (scheduling) for real-time systems can be divided into the following categories (Nolte, Sjodin and Hansson, 2003; Nolte, Nolin and Hansson, 2005; Davis et al., 2007):

- Priority-driven – static, dynamic and mixed-type algorithms.
- Time-driven, e.g. Flexible Time Triggered communication on CAN, see Nolte (2003), Nolte, Sjodin and Hansson (2003), and Pedreiras and Almeida (2000).
- Share-driven – dependent on resource allocation algorithm.

The priority-driven mechanism is the most natural one for CAN (as for centralized systems).

Controlling fieldbus data transmission depends on transmission parameters, i.e. bandwidth and latency. In classical analysis of CAN bus, the worst-case latency of CAN frame transmission is considered (transmission time is the longest), see Nolte (2003), Tindel and Burns (1994), Tindel, Burns and Wellings (1995), Tindel, Hansson and Wellings (1994). This issue is reduced to standard analysis of time fixed-priority scheduling algorithm (Nolte, 2003; Tindel and Burns, 1994; Tindel, Burns and Wellings, 1995; Tindel, Hansson and Wellings, 1994). The response-time is calculated for the worst case, i.e. when all frames are transmitted at the same time. Response-time analysis method are verified and updated in Davis et al. (2007).

While calculating the transmission-time of CAN messages, it is necessary to consider "bit stuffing" mechanism, which is responsible for synchronizing frames at a specific bit level. When the transmitted frame has five bits at the same logical level, a stuff bit is added automatically. Its polarization is opposite to that of its predecessors (Fig. 5).

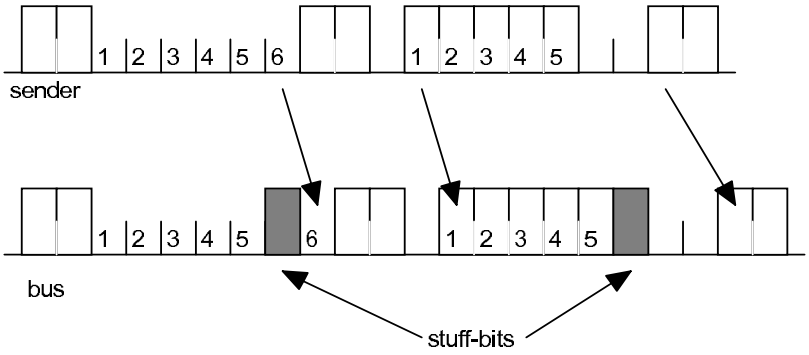

Figure 5. Bit stuffing example

If a frame has 6 or more bits of the same polarization, it is reported as an error. Bit stuffing can make effective number of bits greater than expected from the size of the frame. Higher number of transmitted bits lengthens the transmission. Bit stuffing mechanism is not applied for the last 10 bits of the frame. However, it affects 34 control bits and 0 - 8 data bytes (for CAN 2.0 A standard).

The total number of bits for data frame is calculated according to formula (1), assuming the maximum number of stuff bits:

$$8L_m + g + 10 + \left\lfloor \frac{g + 8L_m - 1}{4} \right\rfloor \tag{1}$$

where:

$L_m \in [0, 8]$ – number of bytes for data frame m.

$g \in \{34, 54\}$ – number of control bits for CAN 2.0 A and CAN 2.0 B standards.

If τ_{bit} is the transmission-time of a single bit, the worst-case latency can be calculated as:

$$C_m = \left(8L_m + g + 10 + \left\lfloor \frac{g + 8L_m - 1}{4} \right\rfloor \right) \tau_{bit} \ . \tag{2}$$

(Maximum number of stuff bits = Floor $\big((34 + 64 - 1)/4\big) = 24$).

The 8 Byte data frame may contain up to 24 stuff bits, which extend transmission time by over 20%. Instead of taking maximum number of stuff bits, as in the worst-case scenario, a probabilistic approach can be employed, where the stuff bits distribution is based on the statistical data. T. Nolte (2003) has shown that response-time in the probabilistic approach is considerably shorter than in the worst-case. Besides, probabilistic results are closer to those obtained from actual measurements. The CAN protocol overhead resulting from the stuff bits is considerable and cannot be neglected.

The CAN standard defines only the way in which data are exchanged in the network. In case of specific application, additional elements should be defined, e.g. significance of data, identifier allocation, response to errors from physical layer, message confirmation, transmission of data larger than 8 Bytes. These elements are specified in application layer. CANopen (2000) protocol is one of the most popular solutions.

CANopen defines two types of data transmission with different characteristics, which meet requirements of automation systems in full application range. They include Service Data Objects, SDO, and Process Data Objects, PDO.

SDO transfers are acyclical and of low priority, used mainly for configuring CANopen nodes. Each SDO contains 16-bit index and 8-bit subindex referring to specific configuration parameter. The client-server communication model is usually applied for SDO. In such a case, controlling information transfer is always triggered by the master node, which acts as a client, whilst the slave node provides information (service), thus acting as a server.

The PDOs are used for transmitting time-critical data. They do not introduce any overhead because one CAN frame corresponds to one PDO. Therefore, data transfer using PDOs is faster and more flexible then using SDOs. The CANopen protocol offers four PDO transmission modes:

- Synchronous mode:
  - o event driven (using the SYNC object) – *client-server model*. The master node (client) sends a SYNC message to the network. In response, the slave nodes (servers) transmit a PDO;
  - o time driven (controlled by internal clock) – *producer-receiver model*. Slave nodes (message producers) send PDOs in constant time intervals. The message is directed to the master and/or slave nodes which expect specific data.
- Asynchronous mode:
  - o remotely requested – *client-server model*. The master node (client) sends a PDO to the network, requesting information from of a specified node. The slave node (server) responds with a PDO involving appropriate data;
  - o Event-driven (e.g. temperature has changed by 0.1^oC) – *producer-receiver model*. Following some event, e.g. change of the temperature by 0.1^oC, the slave node (message producer) sends PDO to the network.

Analysis of CAN bus with CANopen protocol shows that for real-time data transmission it suffices to take PDOs into consideration for the following reasons:

- PDO priority may be changed dynamically.
- Each PDO corresponds to one CAN data frame.

It follows from above that PDOs are best suited for data transmission in CAN by CANopen protocol in RT conditions. They offer wider range of transmission modes than SDOs and flexibility of priority selection.

Another issue related to data transmission is selection of the optimal message scheduling mechanism. Due to the transmission medium access mechanism the priority-driven algorithms are most natural for CAN. In CANopen protocol only a fixed priority-driven algorithm can be used. Dynamic and mixed-type (MTS) priority-driven algorithms are not allowed. In MTS algorithm the identifier bits are manipulated in a way not permitted in CANopen. The 4 identifier initial bits in CANopen define the code of the message-object function in the network.

5. Data transmission models for SCDS

Message scheduling for the proposed system can be implemented by means of the Flexible Time Triggered communication on CAN (FTT-CAN). Bearing in mind assumptions for SCDS, hardware realization of execution sub-layer with CANopen protocol, and also selection of time-driven message scheduling method, the following concept models of data acquisition server have been analyzed for the control sub-layer (Dach, 2004):

- Polling server. In this case the server enquires specific CAN nodes periodically or aperiodically. The server polls the network, and gets responses from the nodes. No answer means error or network failure. The polling method is deterministic and simple, however, it requires a considerable part of the transmission band. The polling server for CANopen protocol can use PDOs in the synchronous transmission mode.
- Waiting server. The server waits for information from the nodes. Occupation of the band strongly depends on configuration of the nodes and can be significantly reduced. Messages may be sent periodically or aperiodically. Only in case of periodic messages the server is able to determine whether all network nodes operate properly. A compromise solution may be such configuration of the nodes that the messages are sent aperiodically, e.g. whenever temperature changes, together with periodic "heart-beat" messages. PDOs are used for this server type.
- Mixed-type server. In this case, polling server and waiting server mechanisms are applied. Depending on the polling server parameters and configuration of the nodes, better results may be obtained than in two other cases.

Applicability of the presented solutions will be discussed in more details. Functional properties and time analysis will be done using UML sequence diagrams.

5.1. Polling server

The idea of polling server is based on elementary cycles periodically initiated by the master node, starting from transmission of SYNC objects (Fig. 6).

Slave nodes are programmed in such a way that they respond to SYNC object by sending PDOs with specific data. PDO priorities are attributed arbitrarily, and do not depend on the address of the source node.

For the worst-case scenario with maximum stuff bits, the system response-time for a single frame may be calculated as folows:

$$R_{sys} = C_{SYNC} + 3\tau_{bit} + \tau_{proc} + \sum_{m=1}^{n} \left(C_m + 3\tau_{bit}\right) \tag{3}$$

where:

R_{sys} – response-time of the system,
C_{SYNC} – transmission-time of a SYNC object; this object does not contain a data field,
$3\tau_{bit}$ – minimum inter-frame space,
τ_{proc} – time needed by the nodes to process SYNC message,
C_m – time of PDO transmission,
n – maximum number of nodes in the system; $n = 127$ for CANopen.

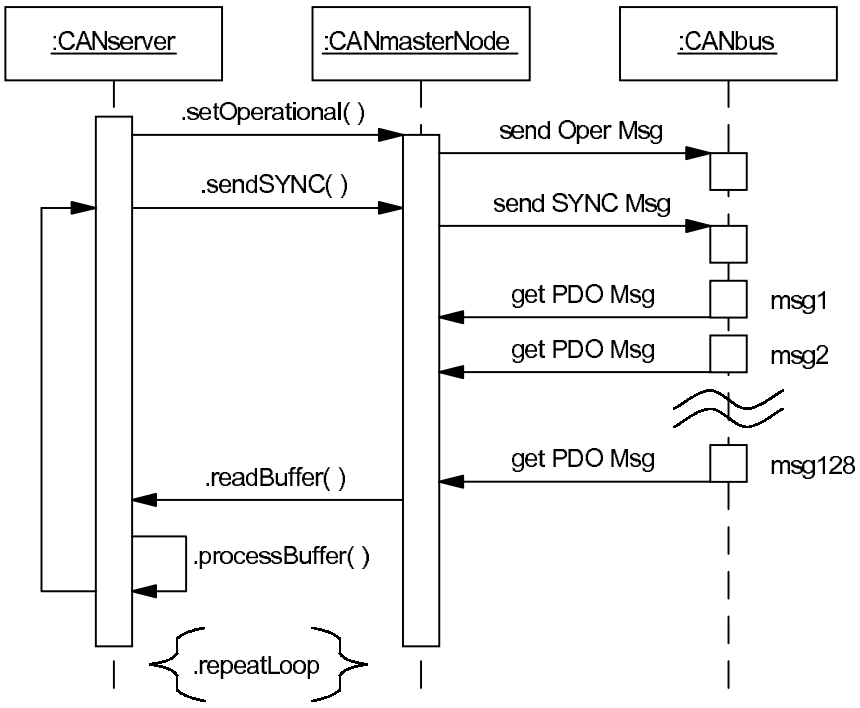

Figure 6. Polling server

Message blocking does not have to be taken into account, since response-time of the whole system is considered (not specific PDOs). Polling frequency should enable all nodes to send PDOs before another cycle begins. Moreover, some time margin should be assumed for sending control or configuration data from the master to slaves.

If some nodes are privileged, then the polling server may send SYNC objects with various ID numbers to specific group of nodes alternately in successive cycles. The synchronous transmission mode with SYNC objects is rarely encountered in other types of fieldbuses. This solution seems to be more efficient than the traditional mechanism of polling individual nodes.

5.2. Waiting server

Waiting server (Fig. 7) does not send SYNC objects. Its operation is based on cyclic reading of the master node buffer. This cycle, however, does not depend on the number of slaves, as it does for the polling server. The slaves are programmed in such a way that the PDOs are transmitted asynchronously after a change in the monitored physical parameters, e.g. temperature. To provide the

highest reliability of the system, slaves employ internal clock, and in fixed time intervals send PDOs informing about their state. This mechanism is sometimes called a "heart-beat", it informs the master whether slaves operate correctly. Any breakdown or physical unplugging of any slave stops heart-beat information in the master, which in turn, is treated as the server error.

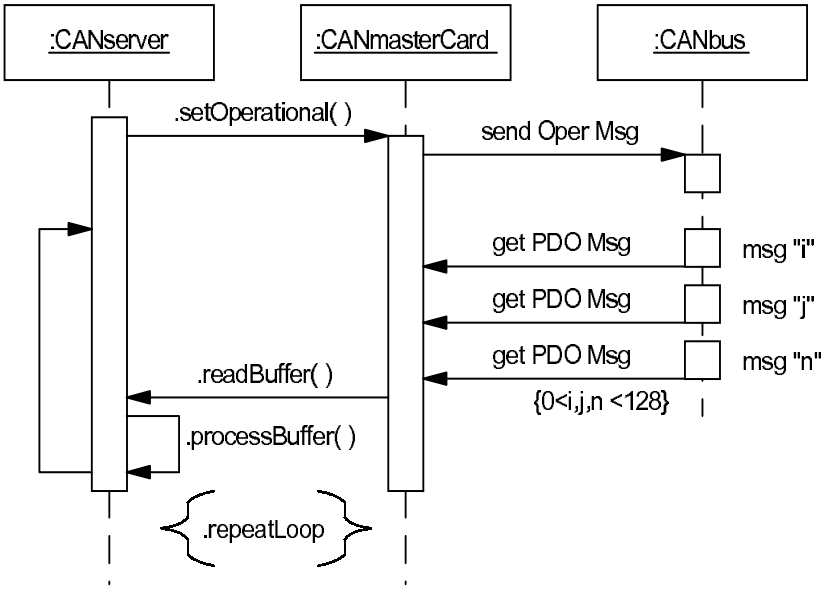

Figure 7. Waiting server

Asynchronously triggered data transmission with PDOs should have higher priority than the heart-beat objects. The period for the heart-beats should be long enough, so as not to overload the network. Waiting server based on such mechanism uses transmission bandwidth more efficiently than the polling one.

Master node passively waits for the asynchronously triggered messages; it does not have any control over the frequency of their occurrence. When physical parameter changes too often in many nodes at the same time, the higher-priority messages may block access to the network. This situation is very disadvantageous, as the master node does not obtain timely information on the state of the entire system. This problem can be solved by assuming appropriate Minimum Interarrival Time (MIT) for aperiodic messages, so that they arrive rarely enough, and do not block lower-priority messages.

Selection of MIT is not trivial. MIT depends on the number of nodes, length of data frames in aperiodic messages, frequency of heart-beat, etc. Another solution to the bus blockade problem may be master node to cut off those slaves, which send the aperiodic messages too frequently. Master can also lower pri-

ority of aperiodic messages sent by those slaves. Cutting off the nodes can be performed by sending specific CAN messages (NMT messages of the highest priority). Network master can control communication state of other nodes (network slaves) of a CANopen network, i.e. it can change the state of all nodes or of an individual node by a single command.

5.3. Mixed-type server

The concepts of polling and waiting servers are combined into a mixed-type server, which handles synchronously and asynchronously triggered messages. Each node is programmed so as to transmit both synchronous and asynchronous PDOs (Fig. 8).

Synchronous messages have higher priority than asynchronous ones. Moreover, slave nodes are divided into groups. In each elementary cycle the master sends synchronization message directed to different group of slaves. The length of each cycle enables each node of a given group to send PDO in synchronous mode, and also enables asynchronous messages from an arbitrary node.

In this approach it is possible to determine maximum impassable time of data acquisition from each node based on synchronous transmission mode. In case of many nodes and moderate network load, the asynchronous messages provide data much faster than synchronous ones.

Mechanism of mixed-type server is presented in figure Fig. 8. The UML diagram represents a system involving 120 slave nodes divided into 12 groups. The mixed-type server collects information from all the nodes by means of synchronous messages, after performing 12 cycles. Notation used in Figs. 7 to 9 differentiates between calling methods (e.g. .sendSYNC) and issuing bus commands (e.g. get PDO Msg).

6. Verification of time constraints

Meeting time constraints for a system based on CAN bus with CANopen protocol, employing the presented server concepts, depends mainly on the rate of data transmission and network load Werewka and Szmuc (2001). Determination of the length of the elementary cycle is the basic problem for polling and mixed-type severs. The elementary cycle should exceed time limit for the entire system; besides, occupation of the transmission bandwidth by synchronous messages should allow for some additional messages, e.g.:

- Asynchronous messages sent by slave nodes (mixed-type server),
- Network management and control messages sent by master to slaves.

Rate Monotonic Analysis (RMA) can be applied for the polling server (Liu and Layland, 1973; Klein, Lehoczky and Rajkumar, 1994; Sha, Rajkumar and Sathaye, 1994). Time constraints can be analytically verified by the equation

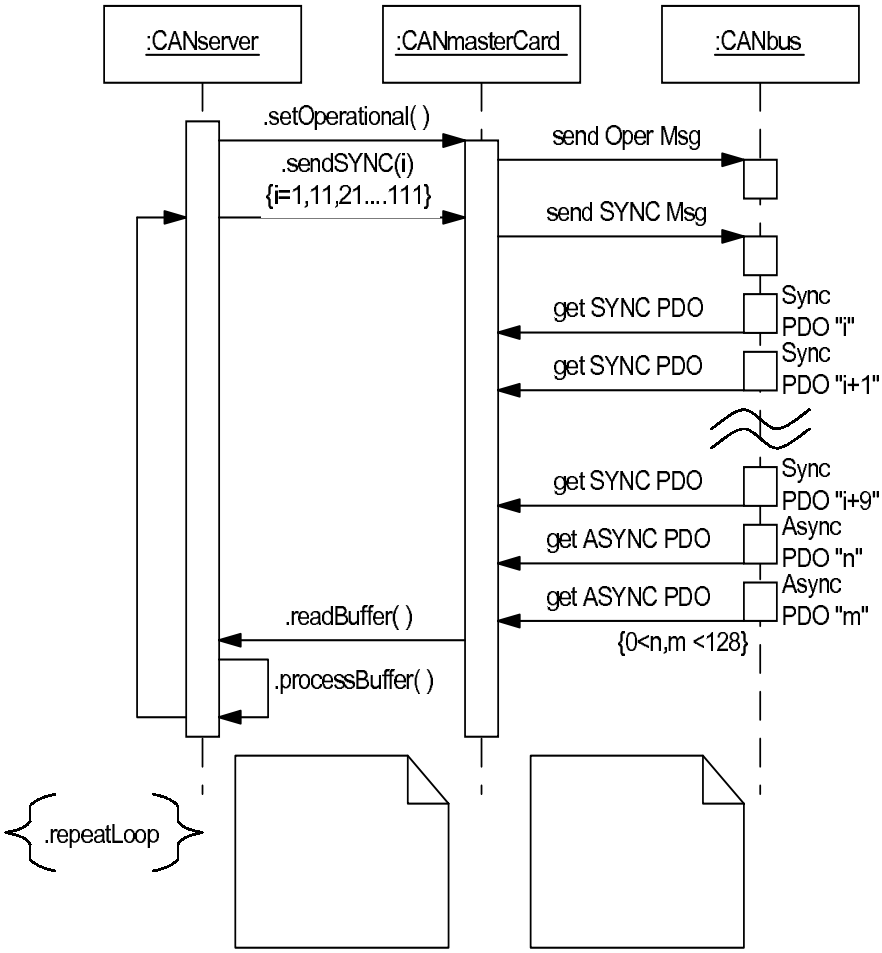

Figure 8. Mixed-type server

and condition (4):

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{\frac{1}{n}} - 1\right) \ . \tag{4}$$

Fulfillment of (4) is sufficient for message scheduling according to the Rate Monotonic (RM) algorithm. The condition (5) can be applied for polling server as follows:

$$\begin{gathered} \frac{C_{SYNC} + 3\tau_{bit} + \tau_{proc}}{T_{SYNC}} + \sum_{i=1}^{n} \left(\frac{C_i + 3\tau_{bit}}{T_i} \right) \leq \frac{R_{sys}}{T_{cycle}} \leq n\left(2^{\frac{1}{n}} - 1\right) \Longrightarrow \\ \Longrightarrow T_{cycle} \geq \frac{R_{sys}}{n\left(2^{\frac{1}{n}} - 1\right)} \end{gathered} \tag{5}$$

where:

- C_i – transmission time of message from node i,
- T_i – arrival-time of message (from node i); for the polling server $T_i = T_{SYNC} = T_{cycle}$,
- T_{SYNC} – period when SYNC messages arrive.
- R_{sys} – time of system response,
- T_{cycle} – duration of elementary cycle,
- n – number of nodes in the system.

The condition (5) is sufficient for determining whether time constraints are met by the polling server.

7. Implementation of control process for CAN

Reliability is key aim of SCDS software, it can be obtained by using object-oriented techniques and as many verified standards and software packages as possible. Application of well adopted standards and known software packages facilitates system servicing by third parties, and increases compatibility of hardware-software. Implementation of the data server and the control process involved Linux with RTAI extension, see Cloutier (2000), RTAI (2006), Dach (2004), Dach, Korhonen and Pal (2003) (Fig. 9).

Data server, denoted in figure Fig. 9 "CAN-EPICS data server", is connected to the control process by RT FIFO queue. The task of the process (denoted "CAN CTRL process") is to control data transmission between CAN and the data server. To ensure deterministic response-time of the whole SCDS, the control process is treated as time-critical. Accordingly, it has been implemented within the kernel workspace of the system. The control process is executed periodically. It operates as the polling server using synchronous method of data acquisition. The mechanism is presented in figure Fig. 10.

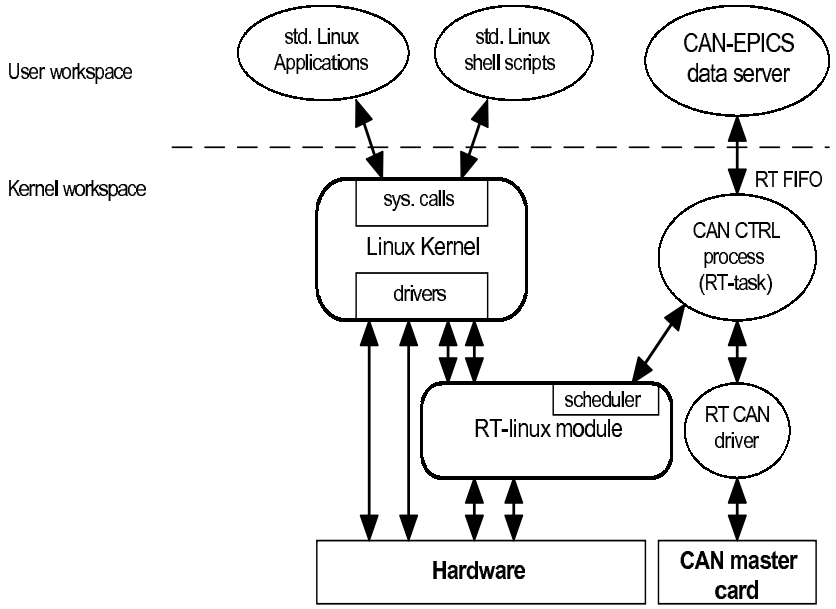

Figure 9. Relations between data server and control process in Linux RTAI environment

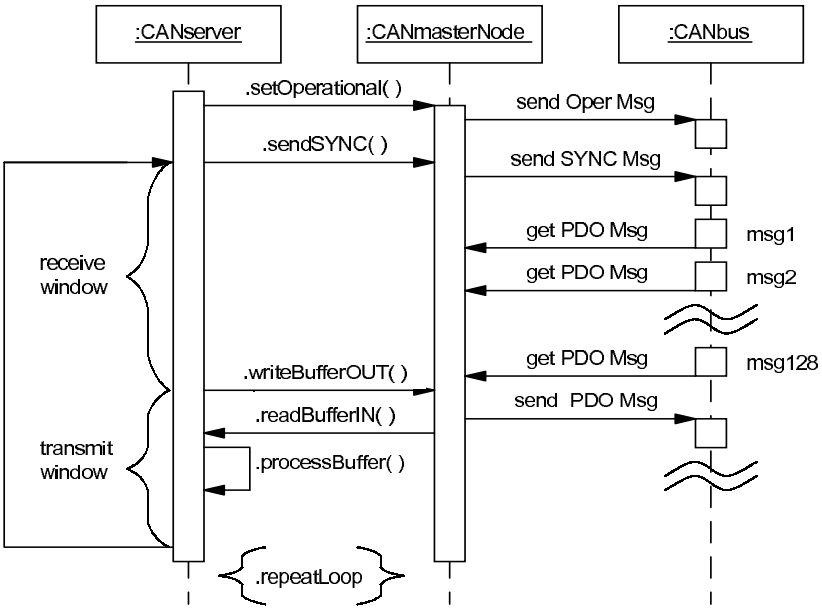

Figure 10. Functionality of the control processes

The control process sends SYNC object to CAN in fixed time intervals, and then switches over to standby-state, waiting for responses from the nodes. Nodes (CAN modules) send PDO message with current state of measured parameters. Each elementary cycle is divided into two parts, called windows: receive window and transmit window. All the messages from the monitored nodes are obtained within the receive window. The transmit window involves control parameters, sent from control process to nodes.

Data exchange between the control process and the data server proceeds through RT FIFO buffer, separately for the incoming and outgoing messages (RT FIFO IN and RT FIFO OUT, respectively). The control process is a real-time task of high-priority. Thus, it is executed before remaining no-RT tasks. Functionally, its design is simple, and does not cause processor overloading.

8. Implementation in SLS

SCDS has been implemented for the Swiss Light Source (SLS) (Synchrotron...., 1999) accelerator control system. From control viewpoint, each accelerator is a complex object of control, consisting of a number of highly distributed subsystems. Any disturbance in operation of VME (VME, 1992) computer requires immediate assistance. The system consists of 166 VME crates distributed along the 288 [m] of accelerator ring. The SCDS consists of two independent segments of CANopen bus. Altogether 222 hardware CANopen modules are applied in SLS (112 for the first segment and 110 for the second). The assumed cable length of 500 [m] for the longer CAN bus segment implied the transmission rate of 125 [Kb/s]. This rate turned out to be too high. Error frames were observed on the nodes located on either end of the bus. After reducing the transmission rate to 50 [Kb/s], no error frames were observed. Other technical details are given in Dach and Werewka (2008).

Main task of the SCDS in SLS is to monitor operation of VME crate power supplies, and send simple control OFF/ON signal to specific VME crate power supply resetting the system bus.

Time constraint of 500 [ms] for SCDS refers to the entire system, not to specific nodes. All monitored data are updated by the master in fixed time intervals corresponding to one cycle (presented in Fig. 10). Analogously, control values are transmitted to distributed nodes in the same cycle as the monitored signals.

Each station of the distributed system involves hardware for controlling and monitoring digital data. One input module for analog parameters, i.e. temperature, can be connected to three stations - VME crates.

The SCDS for SLS has been developed under the following assumptions:

$lw = 166$ – Number of stations to be monitored/controlled. This is the number of (logical) nodes for each segment of the CAN bus, i.e. 84 nodes for the first segment and 82 for the second. Each node uses one digital module and one third of the analog module. So for the 166 nodes the number of modules is 166 + ceiling (166/3) = 222.

$l = 2$ – Number of control parameters per one station.

$k = 6$ – Number of monitored parameters per station (including 5 digital and 1 analog parameter).

$m_{v,p}$ – Signal p being the monitored physical parameter in node v.

$u_{v,q}$ – Control parameter q in node v.

$t_{u_{v,q}} = t_{m_{v,p}}$ – Update time for monitored and controlled parameters is equal to T_{CYCLE}.

T_{RES} is CAN bus response-time, i.e. transmission time of monitored and control parameters:

$$T_{RES} = T_{TRANS} + T_{REC}, \qquad T_{RES} < T_{CYCLE} < D_{SYS} \tag{6}$$

T_{RES} – CAN bus response-time.

T_{REC} – Width of the response window (Fig. 10). Time needed to receive PDOs with monitored data. It includes transmission time of the SYNC object, PDOs with digital and analog data.

T_{TRANS} – Width of the transmit window (Fig. 10). Time needed to send control signals to the nodes. It includes transmission-time of PDOs with digital data.

T_{CYCLE} – Cycle duration.

T_{RES} is determined according to formula (3) for all data transmitted to and from analog and digital modules:

$$T_{REC} = C_{SYNC} + 3\tau_{bit} + \tau_{proc} + \sum_{i=1}^{\eta}\left(C_i^{BIO} + 3\tau_{bit}\right) + \sum_{j=1}^{\lceil \eta/3 \rceil}\left(C_j^{AI} + 3\tau_{bit}\right) \tag{7}$$

$$T_{TRANS} = \sum_{i=1}^{\eta}\left(C_i^{BIO} + 3\tau_{bit}\right) \tag{8}$$

C_{SYNC} – Transmission time of SYNC object to acquire the monitored data.

η – Number of CAN bus logical nodes for individual segment 84 and 82). Number of PDOs with digital data is η, and the number of PDOs with analog data is $\lceil \eta/3 \rceil$.

C^{BIO} – PDO transmission-time for digital measurement/control modules.

C^{AI} – PDO transmission-time for the temperature module (analog data). Single module generates PDO for three stations – VME crates.
$3\tau_{bit}$ – Inter-Frame Space (IFS).
τ_{proc} – Time needed by slave nodes to process SYNC messages.
D_{sys} – Time constraint for the entire system, i.e. 500 [ms].

Response-time T_{RES} is determined independently for each of the two CAN bus segments. The segments are independent, therefore the response-time of the entire system is the longer response-time of the two segments.

9. Experimental results

The SCDS for SLS consists of two independent segments employing the CANopen control modules. The two segments are connected to general-purpose Ethernet through common PC-bridge computer equipped with CAN master card. The computer runs under Linux (RedHat distribution) with RTAI extension.

Transmission parameters have been selected based on the following:

- Analysis of CAN fieldbus functionality,
- Analysis of control process repeatability,
- Measurement of transmission latency between execution and visualization layers.

To analyze the CAN bus, response-time of CAN modules after receiving the SYNC message was measured. Measurements were made for 5, 10, 20, 30, 40, 50, 60, 70 modules of the same type, i.e. digital IN/OUT. Each of the modules was configured to send a PDO with 1 byte after receiving the SYNC. Transmission rate of 50 [Kb/s] was set. To obtain more detailed statistical data, 1000 measurements were made for each bus configuration. Results are listed in Table 1; the time of SYNC transmission is not accounted for. As expected, the response-time of the system increases proportionally to the number of CAN modules.

To compare the measurement and calculation results better, deviation can be calculated from the following equation:

$$\text{Deviation} = \frac{X_{\max} - X_{\min}}{X_{\max}} * 100 \ , \tag{9}$$

where X_{max} and X_{min} are experimental maximum and minimum values, respectively. The deviation (no stuff bits taken into account) does not exceed 4.7%. The pessimism accompanying calculation of response-time with stuff bits as compared to the time obtained directly from measurements stays at constant level, regardless of the number of nodes, and is about 11%.

Determination of response-time for CAN bus is important for selection of (receive-transmit) cycle length and time margin. While determining the response-time for a segment with 108 nodes, 27 modules for analog data and 81 IN/OUT

Table 1. Table of response-times for a given number of nodes

Number of nodes in network	Response-time obtained from the measurements				Response-time determined from eq.(3)	
	Minimum [ms] X_{min}	Maximum [ms] X_{max}	Mean [ms]	Standard deviation from the mean [ms]	No stuff bits [ms]	For worst case [ms]
5	5.752	5.875	5.766	0.034	5.5	6.5
10	11.443	11.561	11.485	0.052	11.0	13.0
20	22.941	23.052	23.003	0.084	22.0	26.0
30	34.448	34.551	34.538	0.111	33.0	39.0
40	46.051	46.153	46.088	0.144	44.0	52.0
50	57.653	57.755	57.683	0.165	55.0	65.0
60	69.050	69.254	69.155	0.206	66.0	78.0
70	80.550	80.652	80.604	0.256	77.0	91.0

digital modules should be taken into account. When calculating the response-time, the following elements should also be considered:

- SYNC message duration,
- 8 bytes of data per PDO for analog data,
- 1 byte of data per PDO for digital data,
- maximum number of stuff bits,
- transmission rate in the network - 50 Kb/s.

The response-time of SCDS was determined on the basis of formulae (6), (7), (8) and (2). The total response-time consists of two elements: response-time for the monitored parameters and response-time for control signals. The first one was determined from the formula (7) what yields

$$T_{REC} = 179.32[ms]\,.$$

Response-time for control signals is determined from (8), so

$$T_{TRANS} = 105.3[ms]\,.$$

Thus, the total response-time is

$$T_{RES} = T_{TRANS} + T_{REC} = 284.62[ms]$$

(for 108 modules). Assuming the target number of 112 modules per each CAN bus segment, the response-time becomes:

$$T_{RES} = 295.12[ms]\,.$$

To meet time requirements for SCDS it suffices to satisfy condition (5). When determining the minimum value of elementary cycle for the polling server on the basis of condition (5), the following is obtained:

$$T_{cycle} \geq \frac{R_{sys}}{n(2^{\frac{1}{n}}-1)} \Longrightarrow T_{cycle} \geq \frac{295.12}{112 \cdot (2^{\frac{1}{112}}-1)} \Longrightarrow T_{cycle} \geq 424.46 \text{ [ms]}.$$

Time constraints for SCDS in SLS require that the CAN bus data be gathered in a period shorter than 500 [ms]. The calculations before and assumptions show that the elementary cycle for the polling server will be equal to 500 [ms] with the margin of over 70 [ms]. The margin will enable some enlargement of bus modules.

The response-time of the system calculated from eqs. (6), (7) and (8) for the formal model of SCDS is considerably pessimistic. This results from the worst-case analysis (maximum number of stuff bits for each PDO). As indicated, the deviation between calculated and measured response-time increases proportionately to the number of CAN modules. Yet, the deviation does not exceed 11%.

10. Conclusions

This presented work refers to issues concerning analysis and design of the Supervisory Control and Diagnostic Systems (SCDS) for large scale distributed control systems. The SCDS systems belong to the sub-class of control systems performing a supervisory function in relation to the underlying control systems. The study led to the elaboration of the methodology to construct a general purpose SCDS systems for large scale distributed control systems. This methodology allowed, in turn, construction of the SCDS system for the SLS (Swiss Light Source) accelerator project.

The paper has presented main principles of SCDS based on CAN. The SCDSs are constructed in order to meet the real-time constraints. This work has introduced the design and principles of SCDS construction to meet these conditions. Time-analysis methods for CAN bus in view of the SCDS have also been described. The CAN field-bus has been chosen for implementation of the SCDS execution sub-layer. The CAN and CANopen standards together with synchronous data transmission mechanism allow for construction of the system whose response-time can be unambiguously and analytically determined.

The investigations, described in this report, are focused on the time response analysis of every part of the SCDS system. This encompasses the time response study of the CAN bus (with CANopen protocol), including the functional analysis of the various data acquisition servers approaches. Also the real time aspect of the Linux operating system with the RTAI (Real Time Application Interface) extension is considered. Implementation of the acquisition server was done by means of the object oriented techniques which ease the server maintenance and future extensions. The server operation under Linux OS with RTAI (Real Time

Extension) has proven its long term deterministic behavior which was tested in SLS environment. The theoretical investigations and experimental analyses have shown high reliability and deterministic functionality of the CAN bus.

References

CANopen Application Layer and Communication Profile (2000) CiA (Can in Automation) Draft Standard 301, www.can-cia.org.

CAN Specification (1991) Version 2.0. Robert Bosch GmbH, Stuttgart 1991, www.semiconductors.bosch.de.

Cavalieri, S. (2005) Meeting Real-Time Constraints in CAN. *IEEE Trans. on Ind. Informatics* **1** (3), 124-135.

Cloutier, P. (2000) *DIAPM-RTAI Position Paper.* Dipartamento di Ingegneria Aerospaziale, Politecnico di Milano.

Dach, M. (2004) *Construction Issues of the Supervisory Control and Diagnostic Real-Time Systems Based on CAN Bus.* PhD thesis. AGH University of Technology, Kraków.

Dach, M., Korhonen, T. and Pal, T. (2003) Interfacing Canbus to EPICS at the Swiss Light Source. *Proceedings of ICALEPPS2003*, Gyeongju, Korea, 506-508.

Dach, M. and Werewka, J. (2008) Accelerator's Supervisory Control System Based on CANbus. *Archives of Control Sciences* **18** (**LIV**) (3), 357-383.

Davis, R.I., Burns, A., Bril, R.J. and Lukkien, J.L. (2007) Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real Time Syst.* **35**, 239-272.

Klein, M.H., Lehoczky, L.P. and Rajkumar, R. (1994) Rate-Monotonic Analysis for Real-Time Industrial Computing. *Computer* **27** (1), 24-33.

Liu, L. and Layland, J. (1973) Scheduling algorithms for multiprograming in a hard real-time environment. *J. Assoc. Comput. Mach.* **20** (1), 46-61.

Meinert, G. 1995 Openness for Automation Networks. *InTech* **42** (10), 29-32.

Nolte, T. (2003) *Reducing Pessimism and Increasing Flexibility in the Controller Area Network.* Malardalen Real-Time Research Centre, Department of Computer Engineering, Malardalen University, Vasteras Sweden http://www.mrtc.mdh.se.

Nolte, T., Nolin, M. and Hansson, H. (2005) Real-Time Server-Based Communication with CAN. *IEEE Trans. on Ind. Informatics* **1** (3), 192-201.

Nolte, T., Sjodin, M. and Hansson, H. (2003) Server-Based Scheduling of the CAN Bus. *Proc. 9^{th} IEEE Conf. Emerging Technologies and Factory Automation* (EFTA'03), Lisbon Portugal, Sep. 169-176.

Pedreiras, P. and Almeida, L. (2000) A Practical Approach to EDF Scheduling on Controller Area Network. *IEEE Workshop*, Porto, Portugal.

Pfeiffer, O. (1994) Feldbusse CAN, InterBus-S und PROFIBUS. *Design & Elektronik* **24**.

RTAI - the Real Time Application Interface for Linux from DIAPM (2006) Dipartamento di Ingegneria Aerospaziale Politecnico di Milano, official website www.rtai.org.

Sha, L., Rajkumar, R. and Sathaye, S. (1994) Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real Time Systems. *Proc. of the IEEE* **82** (1), 68-82.

Sinitean, R.F. (1996) *Controller Area Network*. MET DST 1996.

Synchrotron Lichtquelle Schweiz SLS. (1999) PSI Publishers, Villigen, www.sls.web.psi.ch.

Thomesse, J.-P. (2005) Fieldbus Technology in Industrial Automation. *Proceedings of the IEEE* **93** (6), 1073-1101.

Tindel, K.W. and Burns, A. (1994) *Guaranteed Message Latencies for Distributed Safety-Critical Hard Real-Time Control Networks*. Technical Report YCS 229, University of York, England.

Tindel, K.W., Burns, A. and Wellings, A.J. (1995) Calculating Controller Area Network (CAN) Message Response Times. *Control Engineering Practice* **3** (8), 1163-1169.

Tindel, K.W., Hansson, H. and Wellings, A.J. (1994) Analysing Real-Time Communications: Controller Area Network (CAN). *Proceedings of 15th IEEE Real-Time Systems Symposium*. Puerto Rico 1994, 259-263.

Werewka, J. and Szmuc, T., eds. (2001) *Analysis and Design of Real-Time Computer systems* (in Polish). PTI, Kraków.

VME(1992) *VXI, Futurebus+ compatible products directory.* VITA, VFEA, International Trade Association.

Zuberi, K.M. and Shin, K.G. (1996) Real-Time Decentralized Control with CAN. *IEEE Conference on Emerging Technologies and Factory Automation. EFTA* 96, **1**, 93-99.

Zuberi, K.M. and Shin, K.G. (1995) Non-Preemptive Scheduling of Messages on Controller Area Network for Real-Time Control Applications. *Proc. 1st IEEE Real Time Technology and Application Symposium (RTAS '95)*, Chicago, 240-249 .