# INTERFACING CANBUS TO EPICS AT THE SWISS LIGHT SOURCE

M.Dach†, T.Korhonen, T.Pal, PSI, CH-5232 Villigen, Switzerland

*Abstract*

The SLS accelerator control system is based on EPICS and uses VME crates as control units. In order to monitor the VME crates the CANbus system has been introduced. The challenge was to interface CANbus into EPICS using PC, running Real-Time Linux, and the CAN Open protocol. This paper describes the hardware and software structure of CANbus system in the context of the Real-Time requirements. Several approaches to optimize the deterministic behavior of such a system in LINUX-RT (RTAI - Real Time Application Interface) and EPICS environment are discussed.

## MISSION STATEMENT

The aim of the work behind this paper was to design and implement a Supervisory Control and Diagnostic System (SCDS) to monitor VME crates of the control system of the Swiss Light Source (SLS) at the Paul Scherrer Institute. For the operation of the SLS facility a distributed control system based on EPICS [1] (Experimental Physics and Industrial Control System) infrastructure is used. This system uses VME as operational nodes, and PC/Linux as client platforms. The local Ethernet network links all components of the system. During the commissioning phase of the accelerators, a problem emerged related to VME failures. This refers rather to the software implementation than to the hardware. The SLS control system includes approximately 166 VME crates distributed around the accelerator rings. The *hang up* status of certain operational VME nodes requires immediate operator intervention. In most cases rebooting the faulty node was necessary. A desirable solution to fulfill this requirement would be the installation of SCDS to monitor and control the VME nodes.

## REQUIREMENTS FOR SCDS AT THE SLS

The Supervisory Control and Diagnostic System is a distributed control system and as such should be characterized by following features:

- Reliability – of an item (system) is defined as the probability that it will perform a specified function under specified operational and environmental conditions, at and during the duration of a specified time [2]. The reliability aspect is especially important for SCDS, since it is a supervisory system.
- Determinism – is the ability of the system to react to external events in the defined time horizon. This feature refers to the Real Time systems. Three types of RT systems [3] are considered: Hard RT, Soft RT and Firm RT. The reaction for an event for the Hard RT systems must be done during a specified interval of time, which cannot be exceeded. In our case this type of the system would be the most desirable.

Each VME crate in the SLS is equipped with control-diagnostic interface with 5 digital output signals (mainly the status of VME power supply voltages) and 2 control digital signals (for reboot purpose). The SCDC system should be able to monitor and control above-mentioned signals and in addition to measure the temperature. Hardware wise, it should be independent from the existing VME based control system. From the software point of view the SCDS should be integrated within the EPICS infrastructure to ease future maintenance.

## CANBUS DESCRIPTION

Presently there exist numerous hardware solutions fulfilling the requirements given for SCDS system in SLS. As the simplest and also inexpensive choice, a system based on field-bus was considered. There are, however, many field-bus solutions available. One of the most popular (in Europe) and very reliable is CANbus. The CANbus (Controller Area Network) is a serial communication bus linking intelligent CAN controllers for Real-Time control applications. The CANbus standard (ISO 11898)[4] defines two lowest layers of ISO/OSI reference model: the Physical Layer and the Data Link Layer. The MAC (Medium Access Control) sub-layer (of the Data Link Layer) is the most important from message scheduling and time analysis point of view. CANbus uses deterministic mechanism for bus access called CSMA/BC (Carrier Sense Multiple Access/Bitwise Connection).

The above prevents message collisions and ensures messages arbitration. The arbitration is done with respect to the identifier field of every CAN message. CAN 2.0 A standard uses an 11-bit identifier. From the application point of view CAN does not define the meaning of data carried by CAN messages. There are also some other missing elements like bus management and error handling which is needed to interface CANbus to any application. To compensate these drawbacks, one of the existing approaches i.e. CANopen CiA-301 standard (CAN in Automation) was adopted. CANopen defines an Application Layer with all of the mechanisms and elements required by modern applications. In CANopen protocol the 11 bit long identifier is divided into two parts: one containing 4 bits for function code definition and the second one – 7 bits for CAN node identification. This mechanism clearly characterizes the meaning of data carried by CAN messages. CANopen, having object oriented flavor for data transmission, uses two type of objects: SDO (Service Data Objects) and PDO (Process Data Objects). For the Real Time transmissions the PDOs are recommended since one PDO object is mapped into one CAN data frame (CAN message).

†dach@psi.ch

## TRANSMISSION REQUIREMENTS FOR CANOPEN

For messages transmission CANopen uses SDO and PDO objects and offers the following communication models [4]:

- Model client-server (for SDO objects)
- Model producer-consumer (for PDO objects)
  - o Synchronous transmission
    - Event driven (using SYNC objects) (model "pull")
    - Timer driven (model "push")
  - o Asynchronous transmission
    - Remotely requested (model "pull")
    - Event driven (transmission triggered by the change of certain parameters)(model "push")

CAN, from the access to the bus point of view, can be treated as a centralized system. For such systems, the following message scheduling methods could a priori be considered:

With static priority allocation:

- FIFO – cannot be used for CAN, due to the bit dominance mechanism used for the bus access that clearly appoints the order of messages to be transmitted.
- GRMS (Generalized Rate Monotonic Scheduling). In order to use this method each node of the CANbus has to be configured according to the GRMS paradigm [3], which says that the message priority is inversely proportional to the frequency of message occurrences in the system.

With dynamic priority allocation:

- This type of scheduling is not recommended for CAN since several messages with the same identifier may occur in the system, which leads to the arbitration problem.

With mixed priority allocation:

- It is not recommended. This method requires several bits for user priority identification. In CANopen there are 7 bits remaining for node identification that could be used for user priority allocation. Consequently it leads to a large reduction of nodes in the system. Comparison test showed that the GRMS method gives very similar performance as this one.

The above investigations show that the most suitable method for messages scheduling for CANopen is GRMS. The mathematical representation [3], equation (1), for it is as follows:

$$w(i-1,t) = cf_1\left\lceil \frac{t}{tf_1} \right\rceil + cf_2\left\lceil \frac{t}{tf_2} \right\rceil + \ldots + cf_{i-1}\left\lceil \frac{t}{tf_{i-1}} \right\rceil = \sum_{j=1}^{i-1} cf_j\left\lceil \frac{t}{tf_j} \right\rceil = t$$

$$tz_i = w(i-1,t) + cf_i + bt_i \le df_i \qquad (1)$$

where:

$f_i$ – denotes the message from the series F={ $f_1$, $f_2$,.....,$f_f$}

$tf_i$– period occurrence of periodic messages $f_i$, or the minimal time between non-periodic messages.

$df_i$ – time limitation for the message $f_i$

$cf_i$ – time required for the message $f_i$ transmission.

$bt_i$– blockage time for higher priority message by lower priority message.

$tz_i$ – complete time for the message $f_i$ transmission.

$\lceil x \rceil$- is the minimal integer value greater than or equal to x.

According to equation (1), time t has to be found such that all messages of higher priority than the message fi will be transmitted. This could be done during the iteration process.

Taking into account communication models and message scheduling algorithms the following types of data acquisition servers for CANopen could be considered:

- Polling server: periodically sends requests to every CANopen node and waits for their responses. This is a deterministic way for data acquisition. It, however, requires higher bandwidth of the transmission media.
- Waiting server: waits for non-periodic messages to be transmitted by CANopen nodes upon the change of measured values, like temperature. Data acquisition is much faster compared to the polling server, but a blockage of lower priority messages, by too frequent occurrence of higher priority messages, can happen.
- Mixed server: encompasses advantages of both polling and waiting servers.

Simulations and tests of the three types of servers showed that the polling server is best suited for the SDCS system. It is fully deterministic, which cannot be said for the waiting server. From the complexity point of view, it is simpler in operation compared to the mixed server, and therefore there is lower probability of its malfunctioning. The functional behavior of the polling server for SDCS system is presented in the UML sequence diagram (see Fig. 1).

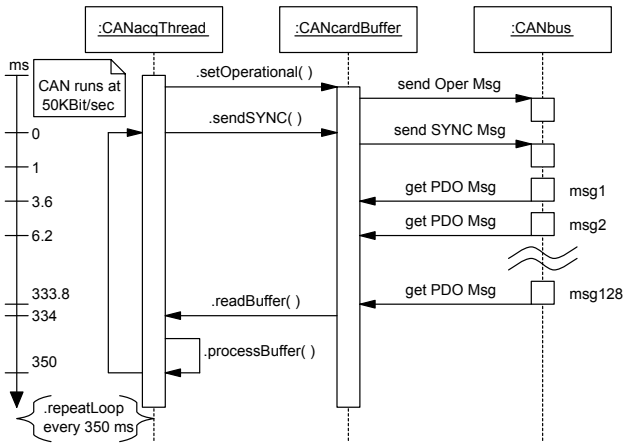


Figure 1: Polling server – sequence diagram

The data acquisition for all CANopen nodes takes one cycle of 350ms. This period of time is fixed and fully deterministic. For data acquisition, an event driven mechanism based on SYNC message transmission is used. CANopen nodes are configured in such a way that all of them respond to it. To calculate the total data acquisition time, equation (1) was used. It could,

however, be simplified because the period of message occurrence for every CANopen node is identical. The total acquisition period is the sum over all nodes multiplied by the time required for every message to be transmitted.

## STRUCTURE OF THE SCDC AT THE SLS

The SCDS is based on CANbus with CANopen protocol. For the integration within the EPICS infrastructure, the CAN-EPICS gateway server is used. It is conceptually divided into two parts: EPICS and CAN. The CAN part for data acquisition, from CANbus, uses the pooling server mechanism. The CAN-EPICS gateway server operates on the PC/LINUX gateway, which links CANbus with the SLS local network. The hardware structure of the SCDS system is shown in figure 2.

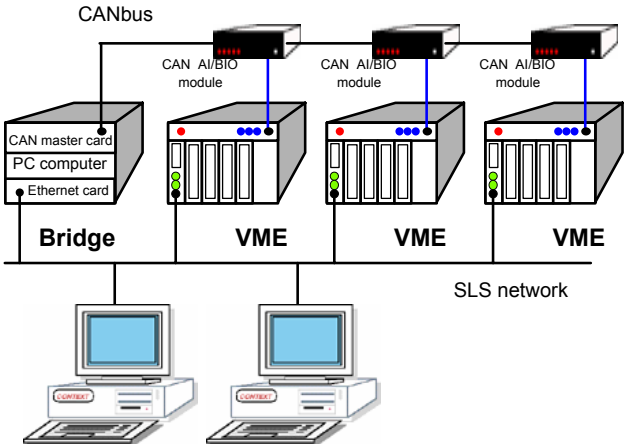


Figure 2: System layout

The software structure of the CAN-EPICS gateway server is shown in figure 3.

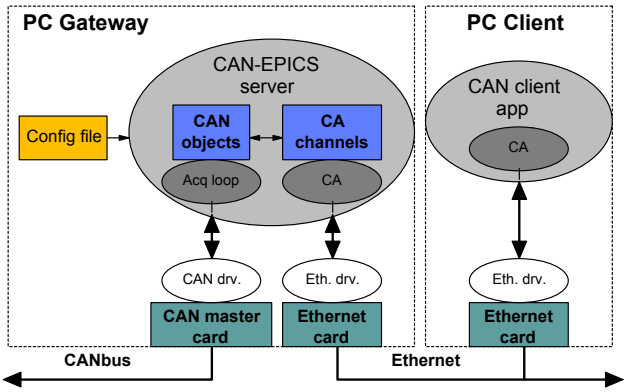


Figure 3: The CAN-EPICS gateway server

The weak point of the SCDS system from the Real Time requirements point of view is the CAN-EPICS gateway server running on Linux. In order to improve the deterministic behavior of the system, the RTAI (Real Time Application Interface) extension for Linux was incorporated. This idea explores the usage of the Linux RTHAL (Real Time Hardware Abstraction Layer)[5]. The RTAI package, by means of this layer, takes over the PC interrupt handling. In order to use the Real time capability of RTAI extension, the CAN-EPICS gateway server was modified, and the CAN acquisition part was implemented as the real-time loadable module into the kernel workspace. The modified structure of the CAN-EPICS gateway server is shown in figure 4.

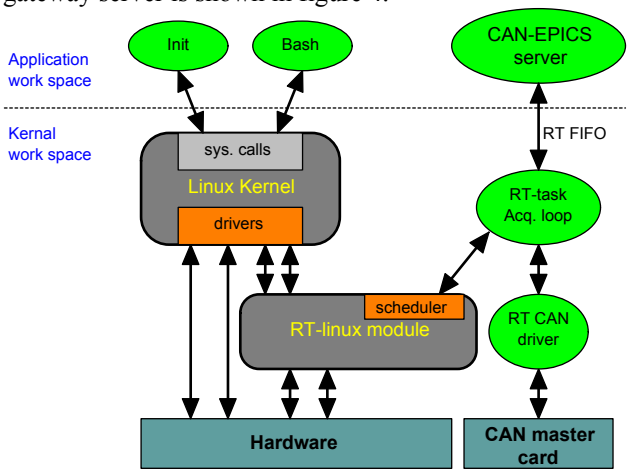


Figure 4: The CAN-EPICS gateway in Linux-RTAI environment

The CAN to EPICS conversion part of the server remains unchanged since it is not critical from the timing viewpoint.

## RESULTS AND OUTLOOK

The SCDS system was implemented and tested. It is working reliably, in a deterministic way. During the phase of server implementation the performance of Linux-RTAI was not known. To test the deterministic behavior of the Linux-RTAI, a PC parallel port was taken. The experiment was done on a PC Pentium II 250 MHz. (Linux kernel 2.4.17 with the RTAI package 24.17.2). The test was to run the RT periodic thread within Linux-RTAI kernel space to set/unset the bit of the parallel port. The waveform observed on the oscilloscope (LeCroy LC 534AM) was symmetrical. Reducing the control RT thread repetition period, it was observed that the shortest one was 20 µsec, when the waveform was still symmetrical. This observation largely exceeds the requirements for the CAN acquisition server. As a complementary feature of the Linux-RTAI test, the EPICS driver to control the Parallel port in Linux-RTAI environment was implemented.

## REFERENCES

[1] EPICS: http://csg.lbl.gov/EPICS/RecomendedDocs.html
[2] P.Kales, “Reliability” ISBN 0-13-485822-0 1998
[3] J.Werewka, T. Szmuc “Analysis and design of real-time computer systems with different distribution grade” ISBN 83-86856-33-5 2001
[4] CAN in Automation http://www.can-cia.de/cg.htm
[5] Real Time Application Interface: http://www.aero.polimi.it/projects/rtai