# SOFTWARE DEVELOPMENT FOR THE COMPASS EXPERIMENT

**Martin Bodlák[1], Vladimír Jarý[1], Igor Konorov[2], Alexander Mann[2], Josef Nový[1], Stephan Paul[2], Miroslav Virius[1]**
[1]Fakulta jaderná a fyzikálně inženýrská, České vysoké učení technické v Praze
[2]Physik-Department, Technische Universität München
Vladimir.Jary@cern.ch

**ABSTRACT:**
The existing data acquisition system of the COMPASS experiment at CERN is a very complex system that consists of a large number of components. Development of a new data acquisition system has started; the upgraded system is based on the Field–programmable gate arrays which reduces the amount of electronics and therefore increases the reliability of the system. This paper focuses on the software part of the system. The hardware part performs the readout of detectors and also controls the flow of the data. The software is responsible for the control and monitoring of the system. The software is to be deployed on several distributed nodes; the communication between the nodes uses custom protocol based on the DIM library. The set of roles and behavior of the system has been defined; the behavior of the system is implemented in the finite state machines. Minimal version of the system has been implemented and tested. The system is currently being ported to the real hardware. It is scheduled to have the system operational since the data taking in the year 2014.

**KEYWORDS:**
data acquisition, remote control, monitoring, state machines

## INTRODUCTION OF THE COMPASS EXPERIMENT

The COMPASS is a high energy physics fixed target experiment at the Super Proton Synchrotron at CERN built for study of the gluon and quark structure and the spectroscopy of hadrons using high intensity muon and hadron beams, [1]. The scientific program was approved by the CERN scientific council in 1997; it consists of the experiments with muon and hadron beams. After several years of preparations and commissioning, the data taking started in 2002. The experiment has recently entered into its second phase known as the COMPASS-II which studies the generalized parton distributions, Primakoff scattering, and Drell-Yan effect, [2].

At first, the existing data acquisition system of the COMPASS experiment is briefly described and its performance and stability problems are analyzed. Next, the brand new data acquisition system based on a custom hardware is introduced. This paper focuses on the software part of this system that is responsible for the control and for the monitoring. The proposed architecture that is based on the finite state machines and the DIM communication library is presented. The minimal version of the proposal has already been implemented; the first results of performance and stability tests are summarized. Finally, the following development steps are discussed.

## THE DATA ACQUISITION SYSTEM OF THE COMPASS EXPERIMENT

The data acquisition system of the COMPASS experiment takes advantages of the cycle of the SPS accelerator that consists of the acceleration and the extraction period which is also known as a spill. During the extraction period, the beam is delivered on the COMPASS target and the secondary particles are produced. These secondary particles are detected by a system

of detectors that forms the COMPASS spectrometer. A collection of data that describes particle trajectories, particle identification, particle interactions, and particle energies is known as an event.

The data acquisition system consists of several layers, [11]. On the lowest layer, the frontend electronics lie. The purpose of the frontend electronics is to preamplify and digitize analogue data coming from the detector channels. Data from multiple channels are readout and assembled by the concentrator modules that form the following layer of the system. The readout is triggered by the signals from the Trigger Control System that also distributes event identification and timestamp. By adding this metainformation to the raw data from detectors, the subevents are created. The subevents are transferred to the readout buffer servers that make use of the SPS cycle by buffering data in a custom made PCI cards called spillbuffers. This allows reducing the average data rates to one third of the onspill rate. From the readout buffers, the subevents are moved to the event builder servers that reorganize data to form the complete events that are after some delay stored on the tapes in the CERN computing centre.

The DATE software package that has been originally developed for the ALICE experiment performs the data acquisition tasks, [3]. From the functionality point of view, the DATE package performs the data flow control, event building, load balancing, data quality monitoring, run control, and interactive configuration. The package is designed to work in a distributed network environment; each processing node must be compatible with the Intel x86 hardware architecture and must be powered by the GNU/Linux operating system with enabled support for the TCP/IP stack.

During the first year of the data taking (i.e. 2002), 260 TB of data have been recorded, [11]. This number increased to approximately 2 PB in 2010. Additionally, the failure rate of the hardware also rises as the equipment gets older. As the PCI bus is a deprecated technology, the replacing of the hardware would require development and production of the PCI Express version of the spillbuffer cards. Instead, a development of a brand new data acquisition system has started.

## THE SOFTWARE FOR THE NEW DATA ACQUISITION SYSTEM

The new system is based on a custom Field–programmable gate array (FPGA) hardware that performs the readout, the data flow control, and the event building, [10]. Thus, the software is responsible only for the monitoring of the system and for the run control.

FPGA technologies have several advantages over a standard custom made hardware. The high flexibility of the final product is the main advantage. Parameters of the physical experiments are changing over time, thus it is essential to have a possibility of adjusting the data acquisition system accordingly. Low cost in comparison with small series of custom made electronics and fast developing cycle are another advantages of FPGA. On the other hand, FPGAs have disadvantages too - mainly lower radiation hardness and high cost in comparison with large series of the ASIC (application–specific integrated circuit). Lower radiation hardness can be compensated by doubling of the components and implementation of the self repair logic for data, for instance the parity control.

We have evaluated the possibility of using DATE as software for the new hardware architecture, [7]. Unfortunately, the DATE package requires the x86 compatible architecture, additionally, it is too complex software. Thus, we have decided to design and implement brand new software. However, in order to keep the compatibility with tools for physical analysis, the data format defined by the DATE package must remain unchanged. The software will be deployed on several distributed nodes; the communication will be based on the custom protocol and the DIM library which is also used by the DATE package for communication with the Detector Control System. The system should support remote control and also define multiple user roles. On the other hand, the control in the real time is not required. Many

unexpected problems may occur during the life cycle of the software project. Thus a risk analysis and risk management is a very important part of the development, [9]. We have estimated that a change of requirements is a highest risk in our case.

The DIM library provides asynchronous communication in a heterogeneous network environment, [5]. The library is based on the TCP/IP protocols, it extends the client–server paradigm with a concept of the DIM name server DNS. Each DIM service is identified by its name. When a server wishes to publish a new service, it must register its name at the DNS. When a client wishes to subscribe to a service, it must ask the DNS which server publishes the required service. The DNS returns the address of the corresponding server and than the communication continues directly between the client and the server. The exchange of information with the DNS is done transparently by the library functions. The library is written in the C language; however the interfaces to C++, Java, and Python languages also exist. We have compared these interfaces and decided to use the C++ version, [8].

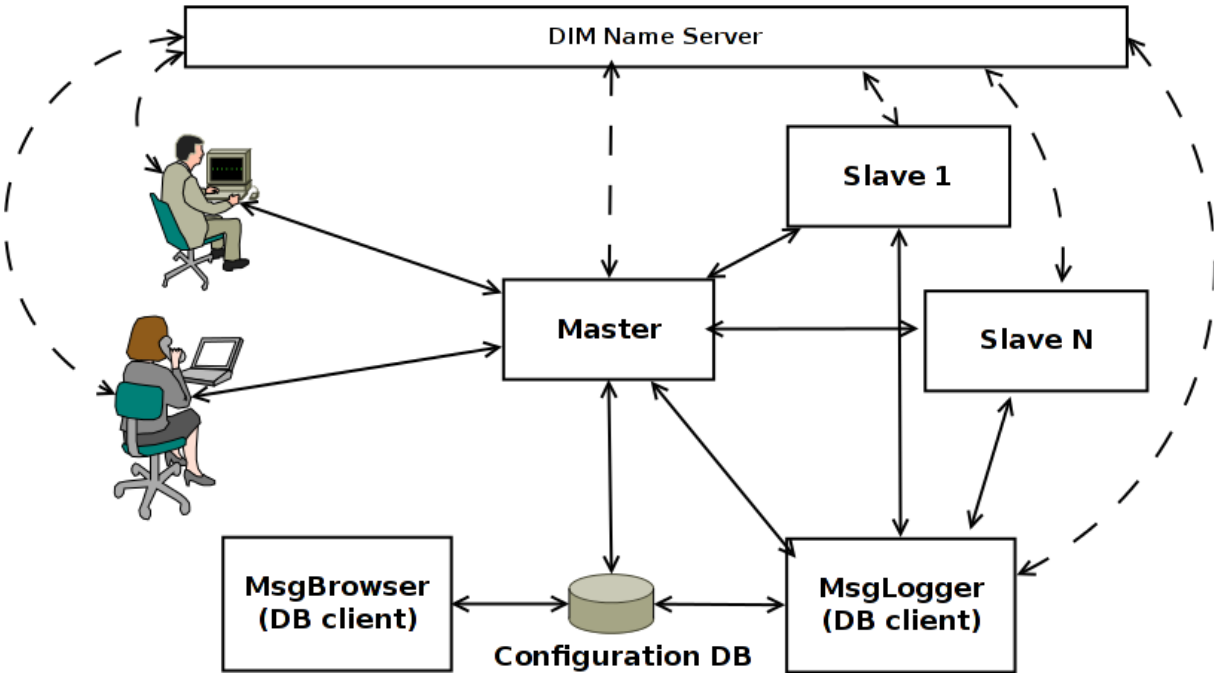


*Figure 1: Roles in the proposed software architecture*

We have defined several roles in the proposal of the system, [6]. The relation between these roles can be seen of Figure 1. The master node is the heart of the system; it acts as a mediator between the user interface applications and the slave nodes that are used to control the custom hardware. The master node receives the commands issued by the user interface applications and forwards them to the slave nodes. The slave nodes receive and execute these commands and send back the information about their state. The master node returns this information back to the user interfaces. Remote control is possible thanks to the use of the DIM library. Additionally, the master node guarantees that only one user can operate the system at the same time. However, multiple user interfaces can receive the monitoring data simultaneously. Each node can send messages to the Message logger node that stores these messages into the online database. These messages can be viewed by the Message browser application. The online database is also used to store the configuration of the system. The configuration is loaded by the master node which distributes it to the other nodes by DIM services. In this way, only the master node requires the database access. The user interfaces, the master node, the Message logger, and the Message browser will be deployed on standard servers. On the other hand, the slave nodes will be installed on the FPGA cards and will be powered by the MICO32 softcore processor and specialized Linux distribution for microcontrollers.In order

to facilitate the porting on the softcore processor, the slaves are implemented in the C++ language. The other nodes are implemented in the Qt framework that extends the object model of the C++ language by the introspection, the guarded pointers, or the signal and slot mechanism. Additionally, the Qt framework contains rich class library that covers the networking, the database access, the multithreaded programming, 2D and 3D graphics, and also the widgets for the graphical user interfaces. Moreover, the applications implemented in the framework are portable between all the major platforms, namely Windows, Linux with X11, and MacOS.

**Message logger**

The message logger is a console application implemented in the Qt framework. Its purpose is to collect data directly from the master and the slave processes (not only redirected messages from the master process). The communication is based on the DIM library (same as the communication within the rest of the system). The Message logger subscribes to several DIM services published by the master and the slave processes immediately after its startup and is able to re-establish the subscription after a restart of a crashed slave process.

The Message logger receives two types of information: messages of the informative character generated for example by an expected change of a state of some process and error messages generated by several types of an unexpected behavior of any part of the whole system. All these messages are transferred according to the custom transport protocol. Every message received by the Message logger is parsed and analyzed. The Message logger then adds some important information (e.g. a severity of the error) and stores the modified messages into the database.

The informative message can result from a start of a slave process (if successful), start of a new run, stop of a current run (if it is expected, i.e. after the maximum spill number is reached), etc. Every message of this type is marked with the "*INFO*" severity. These messages do not require further investigation or user intervention and serve only to store information about important ongoing processes within the system.

The error message can result from an unexpected behavior of any part of the data acquisition system (i.e. the hardware), or even from the control and monitoring software itself. These messages are marked with the "*WARNING*", "*ERROR*", or "*FATAL ERROR*" severity. The "*WARNING*" mark can be used for example when a slave process unexpectedly crashes. As the master process attempts to restart it, it can then generate either an "*INFO*" message in the case it succeeds, or an "*ERROR*" message or a "*FATAL ERROR*" message in the case it fails.

The Message logger stores the following information into the database:

- Date and time of the occurrence: this information is obtained through the original message as the transport protocol contains date and time information in the message header.
- Sender ID: a unique identification number of the process that has generated the message. The process can be then identified by this number in the database.
- Severity - the severity mark contains one of the following values: "*INFO*", "*WARNING*", "*ERROR*", or "*FATAL ERROR*".
- Run number, spill number, and event number: numbers that help to identify the error in a scope of the experiment. These numbers are periodically published by the master process, or are included in the error message.
- Message text: main body of the error message. It helps to identify the error and allows for the further investigation.

**Message browser**

The Message browser is a graphical user interface application developed in the Qt framework. It utilizes the Qt's extensive set of GUI development libraries. It displays messages previously stored into the MySQL database by the Message logger. It allows an interactive configuration of multiple display filters. Applying those filters on the data allows obtaining the required information.

The Message browser runs independently on the whole system, it does not need the DIM library for communication as it only retrieves the data from the database. The application is based on a model–view–controller software architecture with the model being all the messages obtained from the database, the view being a table in the graphical user interface of the application, and the controller being the filtering mechanism.

User-defined filters are set through multiple checkboxes (e.g. for displaying only messages with the "ERROR" severity) and text fields (e.g. for displaying only messages within a given range of run numbers). The Message browser updates the set of messages periodically from the database. The previously set filters also automatically apply to the newly obtained messages.

Because of the independence of the Message browser on the rest of the system, it would be possible to use this application to display and filter data from any database table after performing some minor changes in the source code. It means that it would also be possible to use the Message browser with the current COMPASS data acquisition system based on the DATE package.

**State machine**

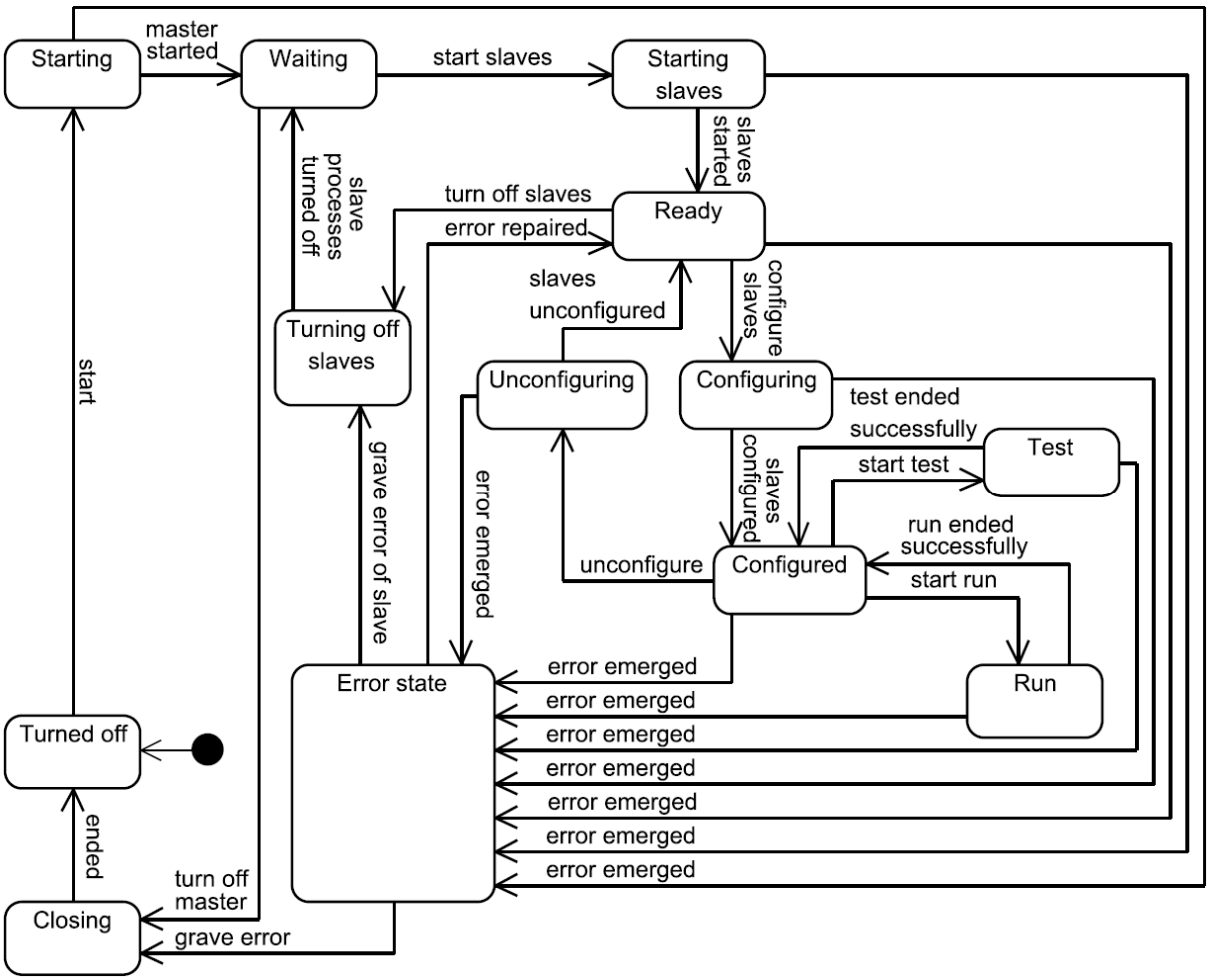


*Figure 2: State machine describing the behavior of the master node*

One of the most important parts of the software development for the physics experiment is the correct and the precise design of the state machines of all processes involved in the data acquisition chain. State machines are used because of the need for precise knowledge of the situation of every part of the system. The Figure 2 shows an example of the state machine diagram that describes the behavior of the master node.

The master node is in the first state *Turned off* when it is not running at all. Directly after the start, the master enters the state *Starting*. At first, it must initialize itself before it can go to following state W*aiting*. In this state, the master node is initialized and waits for the next step. In the fourth state *Starting slaves,* the master controls and monitors starting of the slave processes. Another state *Ready* is defined as state in which all the slaves are on, responding to the master and the master is ready to receive new commands from the user interface. In the state *Configured* every node is prepared for start of the run. *Test* and *Run* are the states in which system takes the data. Settings of the error tolerance and the logging are the only difference between those two states. The *Error* is probably the most complex state; in this state the error recovery logic must be described. Other two states *Turning off slaves* and *Closing* are related to closing and finishing operation.

## PERFORMANCE AND STABILITY TESTS

First tests of new DAQ architecture have been conducted during the winter shutdown of the experiment. During these tests, the slave nodes have been deployed on 8 event builder computers from the present data acquisition system, the master node, the Message Logger, the Message Browser, and the user interface have been running on the computers in the control room, [4]. All nodes have been connected by the Gigabit Ethernet.

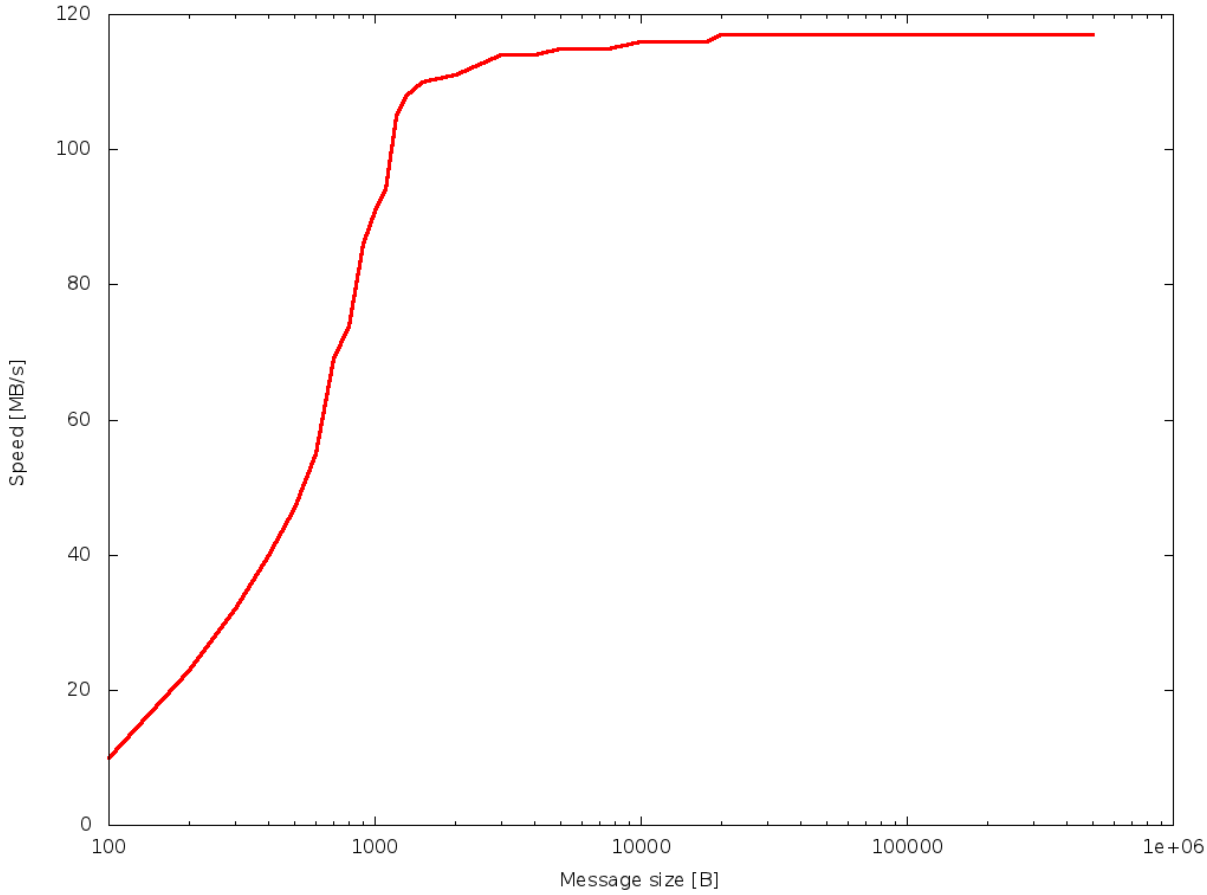


*Figure 3: Results of the performance tests*

The goal was to evaluate the capabilities of the new system and the DIM library. Most importantly, the system is expected to be to exchange at least 200 messages of 1000 B per second. The Figure 3 shows the number of the exchanged messages as a function of the message size. This graph shows that the new system is able to almost fill entire network bandwidth starting from the size of messages of approximately 4000 B. The overhead cost of the network components, mainly switches, is the reason why the system could not fully utilize the network bandwidth. The achieved amount of messages exchanged per second is roughly 90000 at the size of 1000 B, thus both requirements on the performance have been fulfilled.

## RESULTS AND OUTLOOK

We have analyzed the existing data acquisition system of the COMPASS experiment. The cost of scaling of the current data acquisition system is higher than the cost of scaling of the system based on the FPGA. It has been decided to replace the existing system with a brand new hardware and software architecture. The hardware would control the flow of data and event building; the software would provide the run control and monitoring facilities. We have evaluated the requirements on the software and prepared a proposal of the system that fulfills these requirements. Using the finite state machines, we have defined the behavior of the nodes. The system is distributed on several nodes; the communication is based on the DIM

library. According to the proposal, we have implemented the nodes. The system has been tested, the performance exceeds the requirements.

Currently, the communication with the hardware is being tested. These tests include reading and writing data from and to the hardware registers. The microcontroller Linux needs to be deployed on the softcore processor, thus we can start testing the slaves on the real hardware. It is planned to test the fully functional prototype of the system during the technical stop of the CERN accelerators in the year 2013 and put the system in the operation since the year 2014.

**ACKNOWLEDGEMENT**

This work has been supported by the MŠMT grants LA08015 and SGS 11/167.

**LITERATURE**

[1] P. Abbon et al. (the COMPASS collaboration): *The COMPASS experiment at CERN*, In: Nucl. Instrum. Methods Phys. Res., A 577, 3 (2007) pp. 455–518. See also the COMPASS homepage at http://wwwcompass.cern.ch

[2] Ch. Adolph et al. (the COMPASS collaboration): *COMPASS-II proposal*, CERN-SPSC-2010-014; SPSC-P-340 (May 2010)

[3] T. Anticic et al. (ALICE DAQ Project): ALICE DAQ and ECS User's Guide, CERN EDMS 616039, January 2006

[4] M. Bodlák, V. Jarý, T. Liška, F. Marek, J. Nový, M. Plajner: *Remote Control Room for COMPASS Experiment,* In: Konference Tvorba softwaru 2011, Ostrava: VŠB – Technická univerzita Ostrava, 25-27 May 2011, ISBN 978-80-248-2425-3 pp. 1–9.

[5] P. Charpentier, M. Dönszelmann, C. Gaspar: *DIM, a Portable, Light Weight Package for Information Publishing, Data Transfer and Inter-process Communication*, Available at: http://dim.web.cern.ch

[6] V. Jarý: *Towards a New Data Acquisition Software for the COMPASS Experiment*, In: Workshop Doktorandské dny 2011, Prague: Czech Technical University in Prague, Czech Republic, November 2011, ISBN 978-80-01-04907-5, pp. 95–104

[7] V. Jarý: *DATE evaluation*, In: COMPASS DAQ meeting, Geneva, Switzerland, 29 March 2011

[8] V. Jarý, T. Liška, M. Virius: *Developing a New DAQ Software for the COMPASS Experiment*, In: Konference Tvorba softwaru 2011, Ostrava: VŠB – Technická univerzita Ostrava, 25-27 May 2011, ISBN 978-80-248-2425-3 pp. 35–41.

[9] B. Lacko: *The Risk Analysis of Soft Computing Projects*, In: Proceedings International Conference on Soft Computing – ICSC 2004, European Polytechnical Institute Kunovice 2004. pp. 163–169

[10] A. Mann, F. Goslich, I. Konorov, S. Paul: *An AdvancedTCA Based Data Concentrator and Event Building Architecture*, In 17th IEEE-NPSS Real–Time Conference 2010, Lisboa, Portugal, 24–28 May 2010

[11] L. Schmitt et al.: *The DAQ of the COMPASS experiment*, In: 13th IEEE-NPSS Real Time Conference 2003, Montreal, Canada, 18–23 May 2003, pp. 439–444