AN ANALYSIS OF THE CONTROL HIERARCHY MODELLING OF THE CMS DETECTOR CONTROL SYSTEM

Y. L. Hwong, A. Racz, B. Beccati, C. Deldicque, C. Schwick, D. Gigi, E. Cano, E. Meschi, F. Glege, F. Meijers, H. Sakulin, J. A. Coarasa, J. F. Laurens, J. Gutleber, L. Orsini, M. Ciganek, M. Simon, M. Zanetti, R. Gomez-Reino, R. Moser, S. Cittolin, CERN, Geneva, Switzerland
J. F. Groote, T. Willemse, TUE, Eindhoven, Netherlands
A. Meyer, D. Hatton, U. Behrens, DESY, Hamburg, Germany
D. Shpakov, H. Cheung, J. A. Lopez-Perez, K. Biery, R. K. Mommsen, V. O'Dell, Fermilab, Batavia, IL, U.S.A.
A. S. Yoon, C. Loizides, C. Paus, F. Ma, G. Bauer, J. F. Serrano Margaleff, K. Sumorok, MIT, Cambridge, MA, U.S.A.
S. Erhan, UCLA, Los Angeles, CA, U.S.A.
A. Petrucci, J. Branson, M. Pieri, M. Sani, UCSD, San Diego, CA, U.S.A.

Abstract

The high level Detector Control System (DCS) of the CMS experiment is modelled using Finite State Machines (FSM), which cover the control application behaviours of all the sub-detectors and support services. The Joint Controls Project (JCOP) at CERN has chosen the SMI++ framework for this purpose. Based on this framework, the functionality and behaviour of the equipments and subsystems of the experiment is represented as a collection of objects in a hierarchical structure where commands flow down and states flow upwards. The FSM tree of the whole CMS experiment consists of tens of thousands of nodes. Due to the enormous size and complexity of the system, a high level of homogeneity and consistency is desired. The analysis of the current FSM hierarchy of the CMS experiment and the design of a mechanism for the optimization of the FSM logic and structure is presented. The CMS FSM system is discussed in view of most recent research on modelling and analysis of such systems. A methodology for describing and analyzing complex FSM systems is presented.

INTRODUCTION

The CMS experiment uses a general purpose detector to investigate a wide range of particles and phenomena produced in high-energy collisions in the LHC. The Detector Control System (DCS) involves the control, configuration, readout and monitoring of hardware devices as well as monitoring of external systems such as the electrical system, cooling system, etc. The modelling of the control system is implemented as a hierarchy of Finite State Machines (FSM) and is developed using the SMI++ toolkit [1]. The FSM tree of the whole CMS experiment consists of tens of thousands of nodes, which makes the design, implementation and maintenance of a homogenous and consistent system a non-trivial matter. Thus the development of a technique for describing and analyzing systems of such scale becomes an important subject. The micro Common Representation Language 2 (mCRL2) [2] analysis technique is being adopted for this purpose. Using its accompanying toolset, systems can be analyzed and verified automatically.

*This work has been supported in part by a Marie Curie Initial Training Network Fellowship of the European Community's Seventh framework program under contract number (PITN-GA-2008-211801-ACEOLE).
#This project is a collaboration with the Department of Computer Science of the University of Technology Eindhoven.

FINITE STATE MACHINES

The control applications of behaviours of all sub-detectors and support services are modelled as Finite State Machine nodes. In this model there is a state/command interface between a parent and its children. Commands are passed from a parent to its children and the states of the children are propagated to the parent. Two types of objects are defined in this hierarchy:

- Control Units (CUs): They are logical decision units that monitor the states of their children and report an overall state to their parent.
- Device Units (DUs): They interface with the lower level components representing hardware and do not implement logic behaviour.

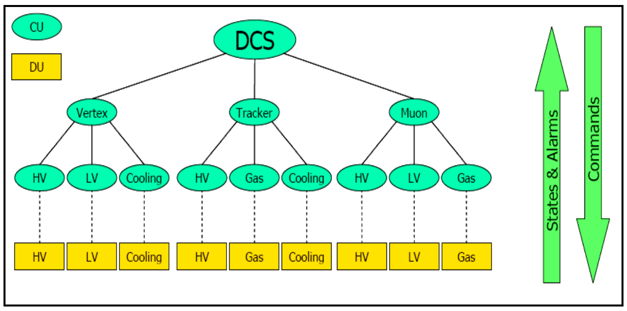

Figure 1: A simple control system modelled using FSMs.

The SMI++ Framework

The implementation of the FSM tree using the SMI++ toolkit is based on the original State Manager concept [3]. To reduce the complexity of large systems, logically related objects are grouped into SMI++ domains; in each domain the objects are organized in a hierarchical structure to form a subsystem control.

The framework consists of a set of tools. The most important ones are the State Manager Language (SML), State Manager Process (SM) and Application Program Interface (API).

The CMS FSM System

Each sub-detector group of the CMS experiment is responsible for the development of its own FSM tree control layer modelling their system, which will then be integrated into a single CMS FSM tree.

Table 1: Parameter Overview of the CMS Sub-detectors

Detector	Monitored Parameters	Channels
Muon Systems	High Voltage	27200
	Low Voltage	10500
	Front End Electronics and other auxiliary services	86500
Calorimeters	High Voltage	3050
	Low Voltage	4800
	Front End Electronics and other auxiliary services	103000
Trackers	High Voltage	4200
	Low Voltage	4200
	Temperature and Detector Control Units	19300

The diversity in the development philosophy of different sub-detector groups and the enormous amount of parameters to be monitored is a fundamental aspect of a large-scale experiment such as the CMS detector. To ensure uniformity and a sound logic implementation throughout a system of such size, the investigation of some desired properties such as deadlock and endless-loop freedom requires an equally, if not more, complex mechanism. The modelling and analysis of the whole CMS FSM system is a challenging task, whereby a simulation and visualization tool is indispensable.

THE mCRL2

The adoption of the mCRL2 toolset, developed at the department of Mathematics and Computer Science of the University of Technology Eindhoven, will be the main focus of this section.

The mCRL2 Philosophy

mCRL2 is a specification language that can be used to specify and analyze the behaviour of large distributed systems and protocols. It is the successor to μCRL[4]. The language is supported by a toolset enabling simulation, visualisation, behavioural reduction and verification of software requirements [5].

The concept of a process is fundamental in mCRL2. Processes can perform actions and can be composed to form new processes using algebraic operators. A system usually consists of several processes in parallel.

A central notion in mCRL2 is the linear process. Complex systems such as the CMS FSM system, consisting of hundreds or even thousands of processes, can be translated to a single linear process. Even for systems with an infinite state space, the linear process (being an abstract representation of that state space) is finite and can often be obtained very easily. Therefore, most tools in the mCRL2 toolset operate on linear processes rather than on state space.

Model checking is provided using *Parameterised Boolean Equation Systems* (PBES) [6]. Given a linear process and a formula that expresses some desired behaviour of the process, a PBES can be generated automatically. The solution to this PBES indicates whether the process satisfies the desired behaviour or not.

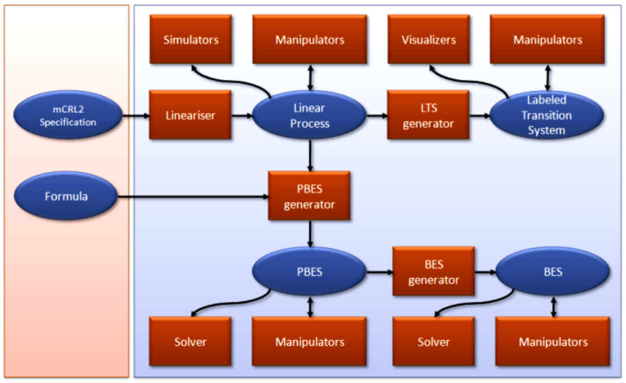

Figure 2: An overview of the mCRL2 toolset.

The mCRL2 Specification

Every analysis starts off by specifying the behaviour of the system being studied. The specification can be seen as a mathematical *model* of the real system: it is a simplified or abstracted version of the reality. Obtaining a specification that is faithful to the real system is far from trivial and to a certain extent requires some human intuition and ingenuity.

A mCRL2 specification is a plain-text file containing a model in the mCRL2 language, with a special syntax defined for this language. A model of the FSM system of the Resistive Plate Chamber (RPC) sub-detector has been defined with the mCRL2 specification and its behaviour has been analyzed.

Linearization

Typically, the specification of a distributed system contains several processes that run in parallel. The first step in the mCRL2 analysis process is to linearize this specification to obtain a *Linear Process Specification* (LPS). This is a mCRL2 specification from which all parallelism has been removed. All that remains is a series of condition – action – effect rules that specify how the system as a whole reacts to a certain stimuli given its current state.

These LPSs are a compact symbolic representation of the state space of the specification. Due to its restricted form, an LPS is especially suited as input for tools; there is no need for such tools to take into account all the different operators of the complete language [5].

Simulation

After having obtained an LPS, a very useful analysis method is by simulating the model. Starting from the initial state, sequences of actions can be performed which can quickly reveal unexpected or erroneous behaviour. It is also a good way of getting acquainted with the modelled behaviour. It is possible to manually trigger state changes, but traces can also be generated automatically, and subsequently inspected.

Visualization

From the LPS, a *Labelled Transition System* (LTS) which is an explicit representation of the state space can be generated. The LTS can be visualized in several ways using interactive GUI tools. The most straightforward way of visualizing an LTS is by showing it as a node-link graph. But the picture produced is often too cluttered for larger LTS. A more sophisticated way of representing a LTS is with the *ltsview* tool which employs a clustering technique to reduce the complexity of the image. It produces a 3D visualization of the LTS and aims to show symmetry in the behaviour of the system. The visualizations also help in scrutinising the model based on unexpected visual anomalies. It is possible to mark transitions and deadlocks for investigation purposes (Figure 3).

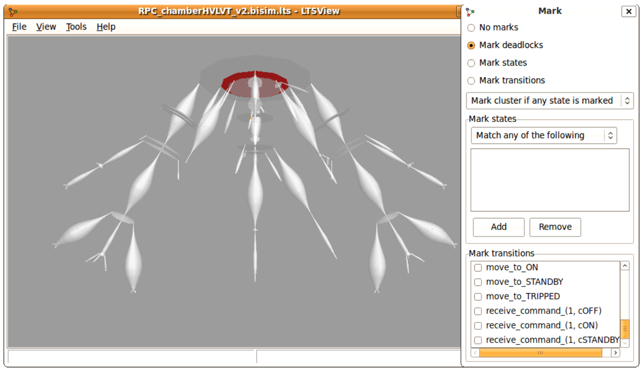

Figure 3: 3D representation of the Labelled Transition System (LTS) for the RPC sub-detector.

Model Checking

A system's analysis often involves showing that the modelled system exhibits certain desired properties (or does not exhibit an undesired one). A powerful verification method which is supported in the mCRL2 toolset is model checking. This is provided using *Parameterized Boolean Equation Systems* (PBESs).

A formula expressing a desired property that the system should not violate (or satisfy) is needed for model checking. Such formulas are expressed in the powerful extensions of the *regular modal μ-calculus* [7] and have its own syntax definition. Given a LPS and a formula, the tool *lps2pbes* produces a PBES in which the model checking question of "does the formula hold for this LPS?" is encoded. By solving the PBES, an answer to this question can be found. The tool for this is *pbes2bool*. It attempts to solve a given PBES and (if successful) returns either *true* or *false*. Examples of the model checking formulas which were applied for the verification of the RPC sub-detector:

- To check for deadlock freedom: *[true*]<true>true*
- To check for the reachability of action move_to_STANDBY:
  mu X. <true>X || <move_to_STANDBY>true
- To check for the inevitability of move_to_STANDBY action:
  mu X. [!move_to_STANDBY]X

CONCLUSIONS

The mCRL2 is a versatile and powerful toolset for the study and modelling of large distributed system, as is seen by its adoption for the analysis and optimization of the CMS RPC FSM system. Further application on other parts of the system is planned and is envisaged to enhance the performance and provide a better insight into the whole CMS FSM system. However, proper training is required for an accurate modelling of the system in such a way that its properties can be efficiently checked using the toolset. As the toolset harbours great potential, an automated tool for the translation from the FSM to the mCRL2 language is foreseen to be a highly rewarding project for the future.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Jan Friso Groote and Tim Willemse from the Design and Analysis of System group at the University of Technology Eindhoven for their generous help and support in the course of this research project.

REFERENCES

[1] B. Frank, C. Gaspar, "SMI++ Object Oriented Framework for Designing and Implementing Distributed Control Systems", SLAC-PUB-12067, http://www.slac.stanford.edu/pubs/slacpubs/12000/slac-pub-12067.html.
[2] http://www.mcrl2.org.
[3] J. Barlow et al, "Run Control in MODEL: The State Manager", IEEE trans.nucl.sci.. 36, pp.1549 – 1553.
[4] J. F. Groote, M. A. Reniers, "Algebraic Process Verification", in: J. A. Bergstra, A. Ponse, S. A. Smolka (Eds), "Handbook of Process Algebra", Elsevier Science Publishers B.V., Amsterdam, 2001, Ch.17, pp. 1151 – 1208.
[5] J. F. Groote, A. H. J. Mathijssen, M. A. Reniers, Y. S. Usenko, M. J. van Weerdenburg, "Analysis of distributed systems with mCRL2", in: M.Alexander, W.Gardnereditors, "Process Algebra for Parallel and Distributed Processing", Chapman Hall, 2009, pp. 99-128.
[6] J. F. Groote, T. Willemse, "Parameterised Boolean Equation Systems", Theor. Comput. Sci. 343, 2005, pp. 332-369.
[7] D. Kozen, "Results on the propositional mu-calculus", Theor. Comput. Sci. 27, 1983, pp. 333 – 354.