

SMI++

Object Oriented Framework for Designing and Implementing Distributed Control Systems

B. Franek, C. Gaspar

Abstract—In the SMI++ framework, the real world is viewed as a collection of objects behaving as finite state machines. These objects can represent real entities, such as hardware devices or software tasks, or they can represent abstract subsystems. A special language (SML) is provided for the object description. The SML description is then interpreted by a Logic Engine (coded in C++) to drive the Control System. This allows rule based automation and error recovery. SMI++ objects can run distributed over a variety of platforms, all communication being handled transparently by an underlying communication system - DIM. This framework has been first used by the DELPHI experiment at CERN for the experiment control. BaBar experiment at SLAC has adopted this framework for the design and implementation of their Run Control system. For this purpose the framework was significantly upgraded. The BaBar Run Control and the underlying SMI++ framework has been in production since the beginning of 1999. SMI++ has recently been adopted at CERN by all LHC experiments for their detector control systems as recommended by the Joint Controls Project. The main features of the framework and in particular of SML language as well as recent and near future upgrades will be discussed. SMI++ has, so far, been used only by large particle physics experiments. It is, however, equally suitable for any other control applications.

I. INTRODUCTION

SMI++ is based on the original State Manager concept [1] which was developed by the DELPHI experiment [2] in collaboration with the DD/OC group of CERN.

Since then, the concept has undergone substantial development through a series of upgrades. These were primarily dictated by the user requirements within the experiments which adopted it as a tool for designing their experiment control. The first significant upgrade (SMI++) was completed in June 1997. This consisted of re-writing its most important tool, State Manager, from ADA to C++. In July 1997 it was extensively tested in DELPHI environment. During that time, the DELPHI experiment control was fully converted from using the 'old' version of SMI to the upgraded version SMI++. At that time it was also adopted by BaBar

experiment [3] at SLAC for the design and implementation of their Run Control. From then on, it has been further upgraded in smaller steps, increasing its flexibility, capabilities and efficiency.

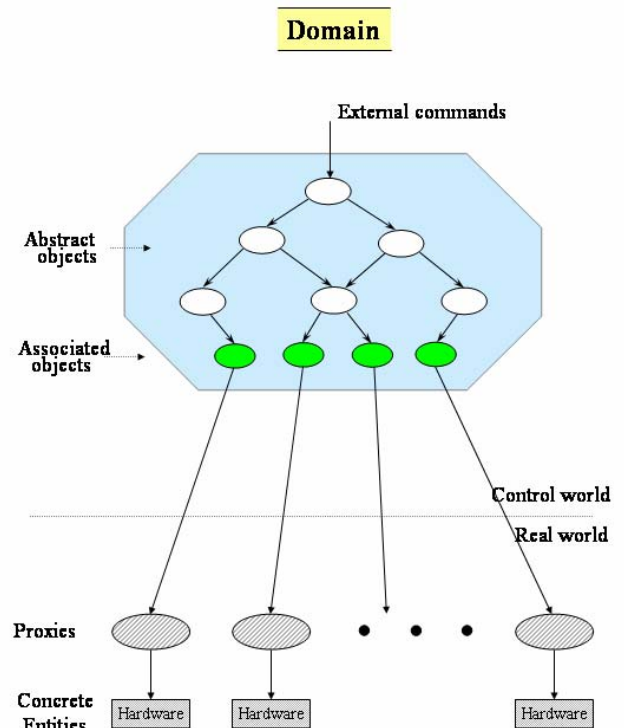


Fig. 1. Basic concepts of SMI++

Recently, all four LHC experiments at CERN [4]-[7] decided to use it either fully or partially for their experiment control. Through this use by major particle experiments and continuous user feedback, SMI++ has become a well proven, robust and time tested tool.

II. BASIC CONCEPTS

The real world to be controlled is typically a set of concrete entities such as hardware devices or software tasks. In SMI++ framework this world is described as a collection of objects behaving as Finite State Machines (FSM). These objects are called **associated** objects because they are associated with an

B. Franek is with PPD, Rutherford Appleton Laboratory, Chilton, Didcot, UK (telephone: +44-1235-445643, e-mail: B.Franek@rl.ac.uk).

C. Gaspar is with PH, CERN, Geneva, Switzerland (telephone: +41-22-7672082, e-mail: Clara.Gaspar@cern.ch).

actual piece of hardware or a real software task. Each of these objects interacts with the concrete entity it represents, through a so called **proxy** process. The proxy process provides a bridge between the real and the SMI++ world while fulfilling two functions. Firstly it follows and simplifies the behavior of the concrete entity and secondly it sends to it commands originating from the associated object.

By analogy, the control system to be designed is conceived as a set of **abstract** (or logical) objects behaving as FSMs. They represent abstract entities and contain the control logic. They can also send commands to other objects (associated or abstract)

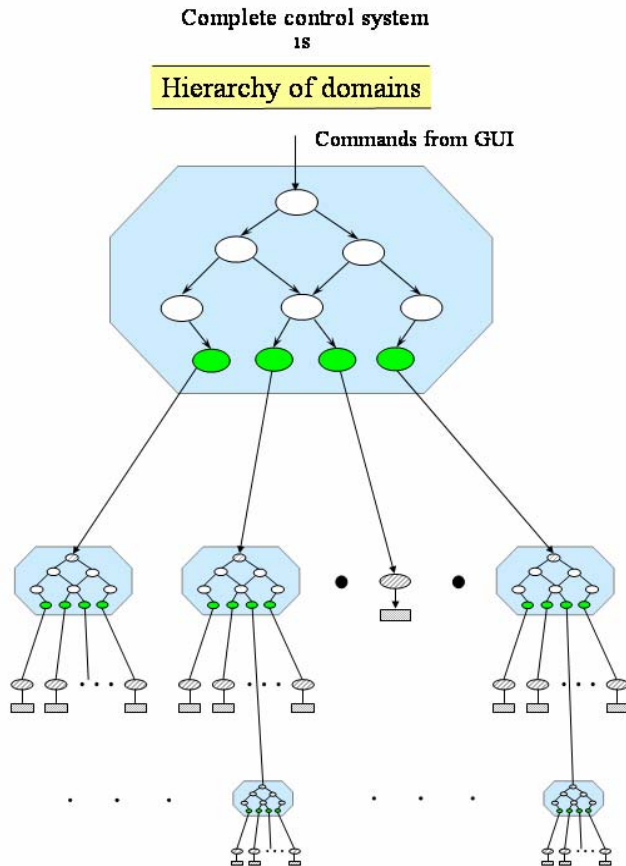


Fig. 2. Basic concepts of SMI++. Hierarchy of domains.

The main attribute of an SMI++ object is its **state**. In each state it can accept commands that trigger **actions**. An abstract object, while executing an action, can send commands to other objects, interrogate the states of other objects and eventually change its own state. It can also spontaneously respond to state changes of other objects. The associated objects only pass on the received commands to the proxy processes.

In order to reduce complexity of large systems, logically related objects are grouped into SMI++ **domains**. In each domain, the objects are organized in a hierarchical structure and form a subsystem control. Usually only one object (top level object) in each domain is accessed by other domains. The final control system is then constructed as a hierarchy of

SMI++ domains. These basic concepts are graphically summarized in Fig. 1. and Fig. 2.

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

III. STATE MANAGER LANGUAGE

The tool used to formally describe the object model of the real world and of the control system is State Manager Language .

```

object : RUN-CONTROL
state : READY
  action : START-RUN
    do INITIALIZE DATA-LOGGER
    if ( DATA-LOGGER not_in_state READY ) then
      move_to ERROR
    endif
    do START CONTROLLER
    if ( CONTROLLER in_state RUNNING ) then
      move_to RUN_IN_PROGRESS
    endif
  ...
state : RUN_IN_PROGRESS
  when ( DATA-LOGGER in_state FILE_FULL )
    do PAUSE
  when ( CONTROLLER in_state ERROR ) do ABORT

  action : ABORT
  ...
  action : PAUSE
    do PAUSE CONTROLLER
    do INITIALIZE DATA-LOGGER
  ...
  move_to PAUSED

!-----
object : CONTROLLER / associated
state : READY
  action : START
  ...
state : RUNNING
  action : PAUSE
  action : ABORT

```

Fig. 3. Example of SML code

This language allows for detailed specification of the objects such as their states, actions and associated conditions. The main characteristics of this language are:

- Finite State Logic

Objects are described as finite state machines. The main attribute of an object is its state. Commands sent to an object trigger object actions that can bring about a change in its state.

- Sequencing

An action performed by abstract object is specified as a sequence of instructions. These consist mainly of commands sent to other objects and of logical tests on states of other objects. Commands sent to objects representing concrete

entities (associated objects) are sent off as messages to the proxy processes.

- Asynchronous behavior

In principle, all actions proceed in parallel. A command sent by object-A to object-B does not suspend the instruction sequence of object-A (i.e. object-A does not wait for completion of the command sent to object-B before it continues with its instruction sequence). Only a test by object-A on the state of object-B suspends the instruction sequence of object-A until object-B reaches a stable state.

- AI-like rules

Each object can specify logical conditions based on states of other objects. These, when satisfied, will trigger an execution of the action specified in the condition. This provides the mechanism for an object to respond to unsolicited state changes of other objects in the system.

Example of SML code is shown in Fig. 3.

IV. STATE MANAGER PROCESS

This is the key tool of the SMI++ framework. At run-time, it organizes and synchronizes activities performed by independent hardware components assigned to the domain and possibly objects in other domains. It does this by using the SML code for the domain. It responds to external events and 'drives' the control system by following the coded control logic and by sending the necessary commands to proxies and objects in other domains. It was designed using an Object Oriented design tool (Rational Rose/C++) [8] and coded in C++. Its main C++ classes are shown in Fig. 4. They are grouped into two class categories:

- SML Classes

These classes represent all the elements defined in the language such as states, actions, instructions etc. They are all contained within the **SMIObj** class (representing SMI++ objects). At the startup of the process, they are instantiated from the SML code.

- Logic Engine Classes

Based on external events, these classes 'drive' the instantiations of the language classes.

CommHandler takes care of all the communication issues. It detects state changes in remote SMI++ objects and 'feeds' the state queue (StateQ). It receives external actions coming from remote objects or from an operator and 'feeds' the relevant queue (ExternalActionQ). It also communicates the state changes in local SMI++ objects to the outside world and sends commands from local SMI++ objects to remote objects.

Scheduler takes the information from the state and action queues and operates on the SMIObj instantiations in such a way that in effect each local object executes its own thread.

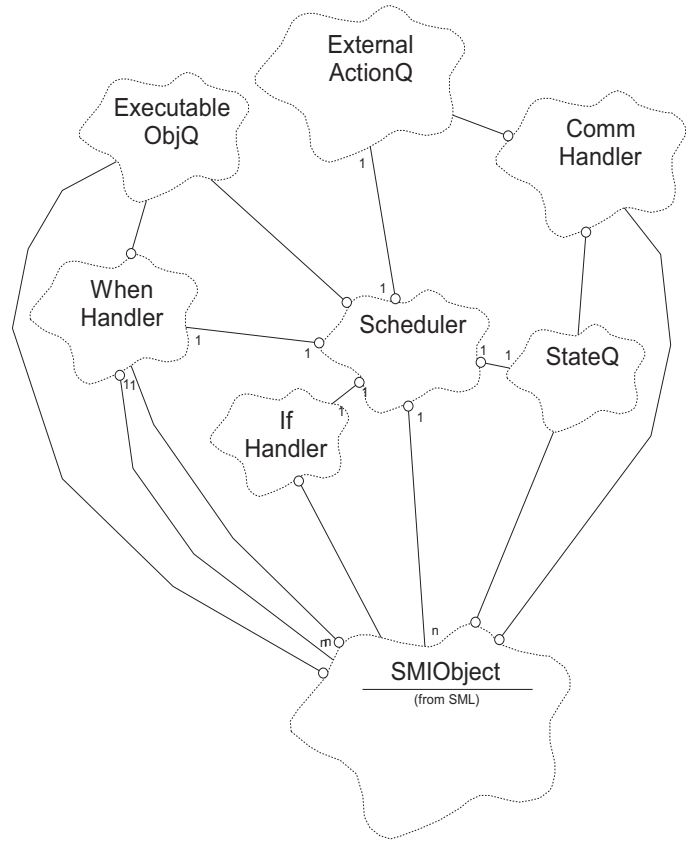


Fig. 4. Main classes of the State Manager.

V. APPLICATION PROGRAM INTERFACE

There are two API libraries available to application designers, in C, C++ and FORTRAN:

1. SMIRTL library

It provides the routines used by proxies to connect and to transmit state changes to their associated objects in the SMI++ world and to receive commands from them.

2. SMIURTL library

It provides the routines used by the processes that require information about the states of objects in the running system. This information is provided in an asynchronous manner – the process is notified about the state change as soon as it happens. The library also provides the routines to send commands to any object in the running system. An example of such a process is a user interface.

There is a generic User interface provided. It is configurable, i.e. the monitored objects can be selectively displayed, moved around the display etc. It is based on Motif. However, we found from experience, that users generally prefer to write their own user interfaces tailored to the specifics of their control systems.

VI. DISTRIBUTED ENVIRONMENTS

State Manager processes representing SMI++ domains can run on a variety of computer platforms. The cooperation

between the domains including all exchanges between objects, are embedded in the SMI++ system. All issues related to distribution and heterogeneity of platforms are transparently handled by the underlying communication system DIM [9] on top of TCP/IP. The asynchronous communication mechanism allows the objects to operate in parallel when required.

At run time, no matter where a SMI++ process (State Manager or proxy) runs, it is able to communicate with any other process in the system independently of where the processes are located. At user level, the name of the object and its domain uniquely determine its location (address). Processes can move freely from one machine to another and all communications are automatically re-established. This feature also allows for machine load balancing.

The communication layer also provides an automatic recovery from crash situations such as restarting a process.

SMI++ is available on any mixed environments comprising: VMS (VAX and ALPHA) and UNIX flavors (OSF, AIX, HPUNIX, SunOS, Solaris), Linux, Windows, OS9, LynxOS and VXWorks

VII. USE OF SMI++ IN DELPHI

In DELPHI the full online system was designed and implemented using this framework. The various areas of DELPHI have been mapped into SMI++ domains: sub-detector domains, DAS domain, SC domain, TRIGGER domain, etc. The full system consisted of about 1000 objects in 50 different domains and distributed over 40 computers.

A high level of automation of the experiment's control system was very important in order to avoid human mistakes and to speed up standard procedures.

Using the SMI++ framework, the creation of a top level domain 'BIG BROTHER' which contained the logic allowing interconnection of the underlying domains (LEP, DAS, SC, etc.) was a relatively easy task.

Under normal running conditions BIG BROTHER piloted the system with minimal operator intervention. During test and setup periods the human operator effectively replaced the top-level object and using the user-interfaces he could send commands to any SMI++ domain.

VIII. USE OF SMI++ IN BABAR

BaBar is a detector that has been designed and built by a large international collaboration of physicists. The collaboration includes over 550 physicists and engineers from the USA, Canada, China, France, Germany, Italy, Norway, Russia, and the United Kingdom. There are currently 72 collaborating institutions. The detector is exploiting the PEP-II facility at SLAC, Stanford, California, USA. Its primary purpose is to study matter-antimatter asymmetry in electron-positron collisions. It does this by collecting and subsequent studying collision events in which pairs of B mesons are produced. Since its startup in 1999 it has so far collected 200

million of such events. The detector consists of many complex sub-systems and weights 1200 ton.

The Run Control was designed, using the SMI++ framework, during 1997-1998 and the first prototype was installed in the second half of 1998, ready for the subsystem groups to test their equipment.

Partial, simplified and SMI++ biased view of the system is shown in Fig. 5. At the top of the hierarchy is SMI++ domain which in Fig. 5. is called 'Master'. It monitors and controls the BaBar detector hardware such as HV power supplies through the sub-detector domains (DCH, DRC,...) It monitors and controls the Data Acquisition system of the BaBar detector through set of proxy processes (see oval shapes in the Fig. 5.). It also communicates with a database from where it retrieves parameters needed for various running conditions. It also monitors the status of the PEP-II accelerator. These tasks are again performed using proxy processes. Under normal running conditions during data-taking, 'Master' monitors, synchronizes and controls its subsystems fully automatically with minimal human intervention. The most important input for this operation is the status of PEP-II accelerator.

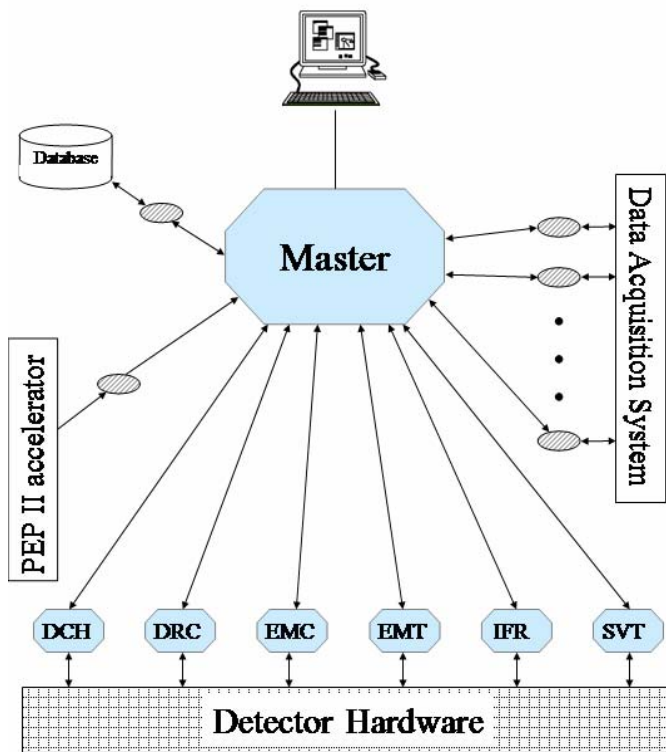


Fig. 5. Schematic diagram of BaBar Run Control

The 'Master' controls yet another part of the BaBar Run Control which handles the calibration of sub-detectors. For the lack of space and in the interest of simplicity, it is not shown in Fig. 5. It consists of 7 domains and dozens of proxy processes.

Since its first prototype, the BaBar Run Control has been developed in response to the experience gained from running the experiment. The inherent flexibility and modularity of the

underlying tool, SMI++, made this development a relatively easy task.

IX. USE OF SMI++ IN LHC EXPERIMENTS

The four LHC experiments at CERN have combined efforts by creating a common control project – the Joint Controls Project – JCOP, in order to develop a control Framework that will be used by different sub-systems to control their equipment.

JCOP has chosen SMI++ as a Finite State Machine toolkit to complement the commercial SCADA (Supervisory Control and Data Acquisition) system that provides the basis for implementing the common control Framework.

The selected SCADA system – PVSS II, provides very useful functionality, like a run-time data base, alarm handling, archiving, a user interface builder, etc but no tools for abstract behavior modeling. SMI++ has been integrated with PVSSII and can thus be used as a component of the Framework.

In this Framework, SMI++ by means of the PVSSII toolkit, has been complemented with a graphic tool to edit and generate the SML code and with a database that allows the archiving of the objects and their states. In order to cope with the common requirements of the four experiments, standard objects are also included by default in all SMI++ domains, providing standard functionality like partitioning, i.e. allowing sub-systems to be excluded or included in the control hierarchy or the enabling/disabling of devices.

All four experiments will use the Finite State Machine component (SMI++) of the framework but to different degrees: ATLAS and CMS will use it for the monitoring and control of their Detector Control Systems (DCS), while LHCb and ALICE will use it for the automation of the complete experiment. The schematic view of the LHCb control hierarchy is shown in Fig. 6. as an example.

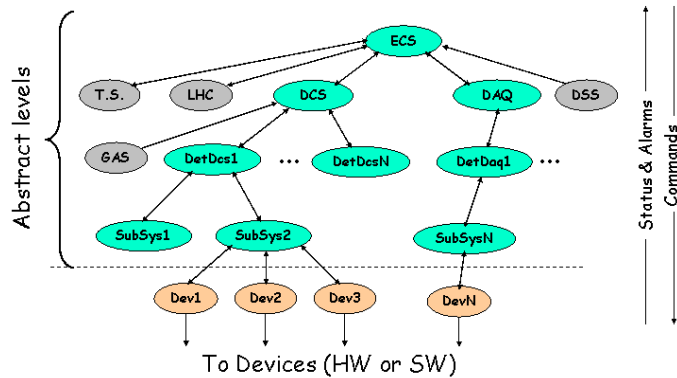


Fig. 6. Schematic diagram of control hierarchy of LHCb experiment

X. CONCLUSION

The SMI++ framework is a powerful tool which, while merging the concepts of object modeling and finite state machines, allows the implementation of a homogeneous, integrated control system by providing a standardized

approach to the control of all types of devices from hardware equipment to software tasks. From a logical point of view, all devices are mapped into, controlled and monitored by, and integrated into higher level control entities. These entities are then responsible for the correlation of events and for the overall coordination, automation and operation of the full system in its different running modes. The system is typically distributed over a set of heterogeneous platforms.

This is achieved by using various SMI++ tools i.e. State Manager Language, State Manager etc.

The SMI++ framework has become a time tested, robust tool through its use by major particle experiments: the DELPHI experiment at CERN in the recent past, the BaBar experiment at SLAC, which is currently using it in production and finally all four LHC experiments at CERN which are now using it for the design of either full or partial experiment control.

XI. ACKNOWLEDGMENT

We would like to thank some of our colleagues at CERN for fruitful discussions. In particular to Ph. Charpentier, M. Jonker, P. Vande Vyvre and A. Vascotto. The ever growing SMI user community also deserves thanks for their valuable feedback.

XII. REFERENCES

- [1] J. Barlow et al., "Run Control in MODEL: The State Manager," *IEEE trans.nucl.sci.*, 36, pp. 1549-1553.
- [2] DELPHI Collaboration, P. Aarnio et al., "The DELPHI Detector at LEP", *Nuclear Instruments and Methods in Physics Research.*, A303, pp. 233-276.
- [3] BABAR Collaboration, B. Aubert et al., "The BABAR detector," in *Nucl. Instr. Meth. A*, 479, 2002, pp. 1-116.
- [4] ALICE Technical Proposal for A Large Ion Collider Experiment at the CERN LHC, *CERN/LHCC/95-71*
- [5] ATLAS Technical Proposal, *CERN/LHCC/94-43*
- [6] CMS Technical Proposal for Compact Muon Solenoid at the CERN LHC, *CERN/LHCC/94-38*
- [7] LHCb – the Large Hadron Collider beauty experiment, Reoptimised Detector Design and Performance, *CERN/LHCC 2003-030*
- [8] Rational Rose/C++, Rational Software Corporation, 2800 San Tomas Expressway, Santa Clara, CA 95051-0951, USA
- [9] C. Gaspar, M. Donszelmann, "DIM – A Distributed Information Management System for the DELPHI experiment at CERN," *Proceedings of the IEEE Eight Conference REAL TIME '93 on Computer Applications in Nuclear, Particle and Plasma Physics.*, Vancouver, Canada