AN ARCHITECTURE AND A FRAMEWORK FOR THE DESIGN AND IMPLEMENTATION OF LARGE CONTROL SYSTEMS

C. Gaspar, CERN, Geneva, Switzerland
B. Franek, Rutherford Appleton Laboratory, Chilton, Didcot, Great Britain
Ph. Charpentier, CERN, Geneva, Switzerland

Abstract

The Online Control System of a high energy physics experiment deals with all aspects of the detector's operations at the experimental site. Grouped in domains: data acquisition, detector survey (temperature, pressure, high and low voltages, fluids supply, etc.) and control, infrastructure (cooling, ventilation, electric power), interaction with the outside world (accelerator system, data storage provider, offline analysis, etc.). In many of these respects, sub-detectors have their own independent system which are then integrated in the framework of the experiment, as are also the various domains.

The control system should allow the operation of the full experiment by a limited team of operators in a safe and efficient manner.

In order to increase the operating efficiency and the reliability and maintainability of the system we propose a global approach in the design of the complete experiment control.

1 INTRODUCTION

Our proposal is to use wherever possible a common approach in the design and implementation of the various domains of the Control System.

In order to make this integrated approach possible we will propose:

- A Generic Architecture
  That can handle all aspects of the monitoring and control of a complete Online System
- A Framework
  A collection of tools and mechanisms that allow the implementation of the architecture. This framework should be used by the developers of the individual subsystems in order to build a coherent system.

2 THE ARCHITECTURE

In order to design and implement such a complex system several levels of abstraction are necessary. Therefore, a hierarchical architecture, with as many levels as necessary to represent the full experiment, has been adopted.

This hierarchical structure will be composed of building blocks organized in a tree-like structure. At the leaves of the tree are the blocks which handle real devices and where users can insert their own code, other blocks contain the description, model, of the system and of its dynamic behavior. The behavior description modules should be organized in as many abstraction levels as necessary to describe the full system. Figure 1 shows a possible representation of such an environment.

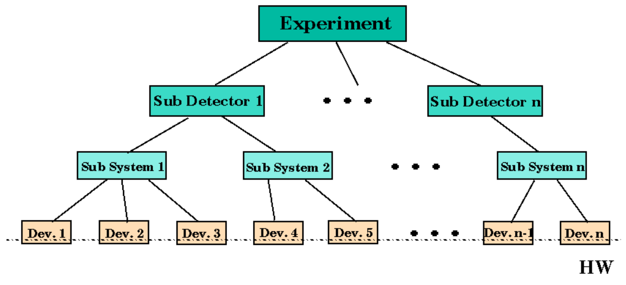

Figure 1: Generic Architecture.

3 THE FRAMEWORK

In order to allow for the implementation of the building blocks composing the proposed architecture a generic framework should be provided to the users. This Framework should provide the following functionality:

- Device handling
  - Static modeling
  - Connections for various HW devices
- Configuration Data Base
- Alarm handling and reporting
- Data archiving and trending
- User interface building tools
- Abstract behavior modeling

All items apart from the last one, which is of vital importance for the implementation of the higher levels of the architecture, are normally made available by commercial SCADA [1] (Supervisory Control And Data Acquisition) systems. Nothing seems to be available commercially for behavior modeling so we will propose SMI++.

4 THE SMI++ TOOLKIT

SMI++ is based on the original State Manager concept which was developed by the DELPHI experiment in collaboration with the DD/OC group of CERN. SMI (State Management Interface), the first implementation of this concept [2], was used in DELPHI since 1990. Due to lack of flexibility in many areas SMI was completely re-designed using Object-Oriented technology and SMI++ implements significant extensions to the SMI concept and a more powerful set of tools.

SMI++ is an Object Oriented Toolkit for Designing and Implementing Distributed Control Systems. The SMI++ methodology combines two concepts: Objects and Finite State Machines (FSM).

Using SMI the experiment can be decomposed and described in terms of objects behaving as finite state machines.

SMI objects can represent concrete entities, for example an hardware device or abstract entities like a logical sub-system. The objects representing concrete entities interact with the hardware they model and control through driver processes or proxies.

The objects are typically organized in hierarchical structures called domains. The interaction between objects can best be seen in the example of Figure 2.

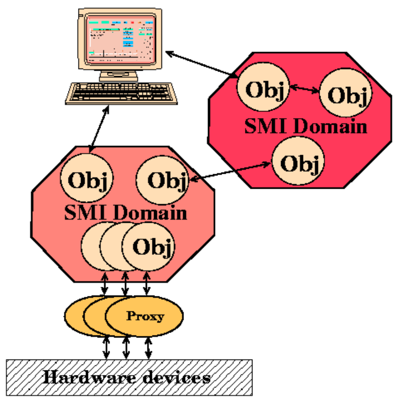

Figure 2: SMI++ Run-Time Environment

The object model of the experiment is described using State Manager Language (SML). This language allows detailed specification of the objects such as their states, actions and associated conditions. The main characteristics of this language are:

- Finite State Logic
  The main attribute of an object is its state. Commands sent to an object trigger actions that can bring about a change in its state.
- Sequencing
  An action on an abstract object is specified by a sequence of instructions, mainly consisting on commands sent to other objects and logical tests on states of other objects. Actions on concrete objects are sent off as messages to the Proxies.
- Parallelism
  Several actions may proceed in parallel. Only a test by an object on the state of another one can suspends the first object if the second one is still in transition.
- Asynchronous
  Objects can specify logical conditions based on states of other objects. These when satisfied will trigger an action on the local object. This provides the mechanism for an object to respond to unsolicited state changes of its environment.

```
!- Example of SML code

object : RUN_CONTROL
  state : READY
    action : START_RUN
      do START READOUT_CONTROLER
      if READOUT_CONTROLER in_state RUNNING
         terminate_action/state=RUN_IN_PROGRESS
      ...
  state : RUN_IN_PROGRESS
    when READOUT_CONTROLER in_state ERROR
      do ABORT_RUN
    action : ABORT_RUN
      ...
```

The SMI mechanism allows an easy reconfiguration of the system: changes in the hardware can be easily integrated by modifying or replacing proxies and logical modifications by changing the SML code.

The decoupling between the actual actions on the hardware (done by the Proxies) and the control logic (residing in the SMI objects) makes the evolution of a system from its first test phase up to final complexity a very smooth process.

4.1 *Distributed Environments*

SMI++ run-time components (SMI++ Domains, Proxies and User Interfaces) are automatically generated from the SML description and they can run distributed over several different machines.

Distribution is embedded in the SMI++ system, all co-operation between the different components of an SMI++ environment is transparently handled by an underlying communication system - the DIM [3] package.

DIM and SMI++ are available in mixed environments comprising the Operating Systems: VMS (VAX and ALPHA), UNIX flavours (HP-UX, IBM-AIX, SUN-OS, SUN-Solaris, DEC-OSF, Linux), Windows NT, OS9, LynxOs and VxWorks.

5 SMI++ IN DELPHI

In DELPHI the full online system is controlled through this mechanism, the various areas of DELPHI have been mapped into SMI++ domains: sub-detector domains, data acquisition (DAS) domain, slow controls (SC) domain, trigger domain, LEP machine domain, etc. The full system comprises about 1000 SMI objects in 50 different domains distributed over 40 machines.

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

Using the SMI mechanism the creation of a top level domain - BIG BROTHER - containing the logic allowing the interconnection of the underlying domains (LEP, DAS, SC, etc.) was a relatively easy task [4].

Under normal running conditions BIG BROTHER pilots the system with minimal operator intervention as shown in Figure 3. In other test and set-up periods the operator becomes the top-level object and using the user-interfaces he can send commands to any SMI domain.

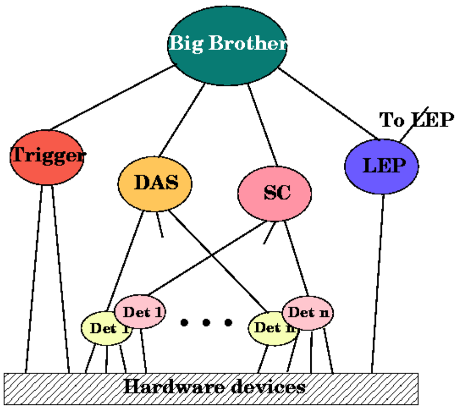

Figure 3: Delphi's Experiment Control

6 SMI++ IN BABAR

In BaBar the HW devices where mapped to software "Components" and controlled using EPICS. SMI++ was then used to model the behavior of the components and to integrate them into "Partitions".

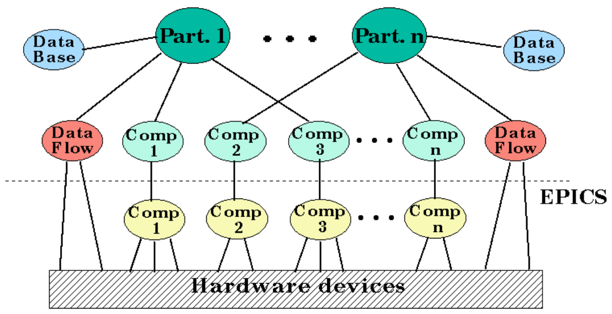

Figure 4: BaBar Control

BaBar's partitioning scheme is totally dynamic. Users can create partitions and allocate the set of components they which to manipulate as shown in Figure 4. During physics data taking one partition takes over and controls all the underlying components. The BaBar Control system is composed of hundreds of SMI++ domains.

7 SMI++ / SCADA INTEGRATION

Since the SMI++ toolkit is a collection of tools developed in C++ it can easily be integrated into a more generic framework based on a SCADA system. This can bring several advantages:

- Provide Behavior Modeling to the Framework
- Use SCADA's database to store the FSM description and configuration, i.e. the same database will contain device description and behavior
- Use SCADA archiving tools to store state changes
- Use SCADA User Interface building capabilities:
    - To build an integrated graphic FSM editor to be used by the developer
    - To build the user interface that will interact with the FSM at run-time.
- Extend SCADA's functionality to control non-supported platforms since SMI++ runs on a larger range of platforms.

8 CONCLUSIONS

In order to build an integrated system capable of controling all aspects of a physics experiment we have envisaged a generic architecture based on building blocks. These blocks are organized in a hierarchical structure with several abstraction levels.

The development of the building blocks should be carried out with the help of a framework. This framework should allow the users to design and implement their sub-system by providing them tools and guidelines. Such a framework could be built around a commercial SCADA system by integrating missing parts one important one being a tool for abstract behavior modeling. SMI++ can easily be integrated and provide the SCADA with the missing functionality.

REFERENCES

[1] A. Daneels, "What is SCADA?", these Proceedings.
[2] J. Barlow et al.,"Run Control in MODEL: The State Manager", in IEEE trans. nucl. sci. 36, 1989, pp.1549-1553.
[3] C. Gaspar and M. Donszelmann, "DIM - A Distributed information management system for the DELPHI experiment at CERN", Proceedings of RT 93, Vancouver, Canada.
[4] B. Franek et al.,"Big Brother - A Fully Automated Control System for the DELPHI Experiment", Proceedings of CHEP 94, San Francisco, USA.