SMI++
Object Oriented Framework used for Automation and Error Recovery in the LHC Experiments

B. Franek – *Rutherford Appleton Laboratory*, UK   **C. Gaspar** – *CERN*, Geneva, Switzerland

Introduction

This framework provides:
-A method for decomposing a complex system into smaller manageable entities where each entity is modelled as a ***Finite State Machine*** . The control system is constructed as a hierarchical collection of intelligent and autonomous FSMs operating concurrently.

-A ***formal language*** for describing each entity and it's behaviour. This allows the sequencing and synchronization of operations, rule-based automation and error-recovery.

-***Tools*** for the deployment of the complete system across heterogeneous distributed platforms.

Basic Concepts
Objects

The real world to be controlled is a collection of ***concrete entities***

E.g. HV Power supply, gas system, radiation monitor, a software task etc

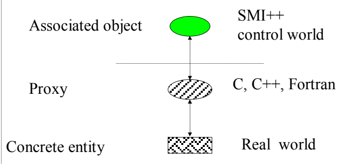
The ***abstraction*** of each concrete entity is ***associated object***

They exist in discrete ***states*** e.g. 'ON'

In each state they accept commands that invoke ***actions*** which can bring about a change of state e.g. 'POWER-OFF'

To implement the ***abstraction*** is the task of ***Proxy*** process

Proxy is a piece of software forming a ***bridge*** between the concrete entity and the associated object in the SMI++ world.

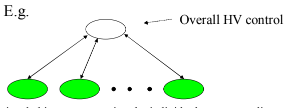


Proxy ***monitors*** the concrete entity and makes it look like a simple object existing in discrete states

It also translates and ***delivers commands*** sent from SMI++ world to the concrete entity.

In analogy, the control system to be designed is conceived as a set of abstract objects.
An ***abstract object*** is an abstraction representing an identifiable logical entity with well defined role in the control system.

E.g.



Associated objects representing the individual power supplies

Abstract objects exist in the SMI++ world in discrete ***states***
in each state they can respond to commands by executing so called ***actions***

An action consists of a sequence of ***instructions.***

During execution of an action:

- object can ***send commands*** to other objects
- it can ***interrogate states*** of other objects
- it can ***respond asynchronously to state changes*** of other objects

Objects can ***execute their actions concurrently*** with other objects

Basic Concepts
Domains

To reduce complexity of large systems, objects are organized in modules called ***Domains***

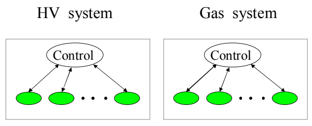
Domain is ***named group*** of logically related objects

Only few objects in each domain are made visible to the rest of the control system
(***modularity*** & ***information hiding***)

Domain is also the ***atomic processing unit*** of control software

Domain
becomes
Process

E.g.



Advantages :

-Encourages
modularity and information hiding

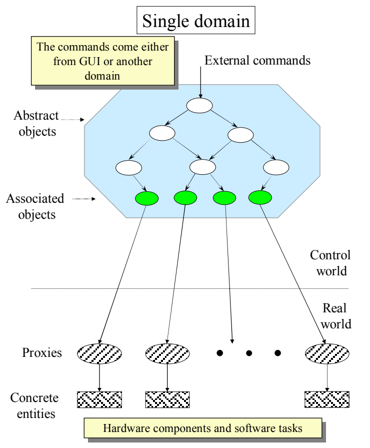
- individual domains can be ***simple*** enough so that they can be easily understood

- Domains can be designed, revised and tested ***independently*** (even by different teams)

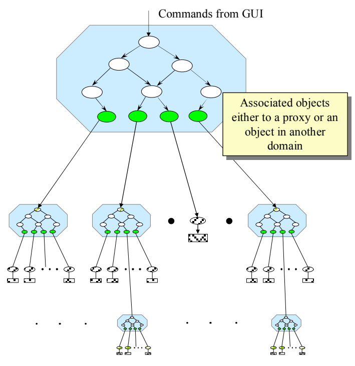
- it is possible to change the design of a domain without affecting the rest of the control system

- encourages ***hierarchical*** structure

- facilitates ***distribution*** of the control software.
Different domains can run on different machines



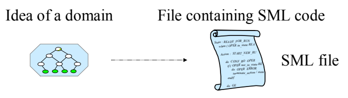
Control system is designed as ***Hierarchy of Domains***


Commands from GUI
Associated objects either to a proxy or an object in another domain

The task of designing the control system is equivalent to that of giving a good description of the real world in terms of domains of objects being controlled and the procedures (embedded in abstract objects) that operate on them

State Manager Language
SML

Formal language for description of objects, their states and their actions (sequence of instructions)



SML file is a full description of the domain

Associated objects. SML code is just a declaration of states and actions

E.g.

```
Object : HV /associated
   State : OFF
     action : SWITCH-ON

   State : ON
     action : SWITCH-OFF

   State : NOT-THERE / dead_state
```

Abstract objects

In addition to the state and action declarations, for every action there is a sequence of ***instructions*** that has to be executed in the course of that action. The language allows object :

- to ***send commands*** to other objects
do instruction

e.g.

```
do  SWITCH-ON  HV-A
do  SWITCH-ON  HV-B
```

will send command SWITCH-ON to object HV-A and HV-B

-to ***interrogate states*** of other objects
if instruction

e.g.

```
if   ( HV-A  in_state  OFF )  then
   do  SWITCH-ON  HV-A
endif
```

- to ***respond asynchronously*** to state changes of other objects
when clause

e.g.

```
Object : ALARM
state : READY
   when (        ( HV-A not_in_state ON)
          or  ( HV-B not_in_state ON )
      )  do RISE-ALARM
```

- etc

Recently, SML was upgraded with advanced features such as Object Sets

- Group of similar objects can be assigned to named Object Set :
e.g..

```
ObjectSet : HV_SUPPLIES {HV-A, …. , HV-Z}
```

- Objects can be dynamically inserted or removed from a Set:
e.g.

```
insert HV-2345 in HV_SUPPLIES
or
remove HV_M from HV_SUPPLIES
```

- it is possible to send commands to all the objects of a given Set with one do instruction
e.g.

```
do SWITCH-ON all_in HV_SUPPLIES
```

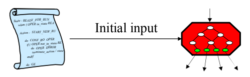
- it is also possible to have constructs such as

```
if  ( all_in HV_SUPPLIES in_state ON )
   move_to READY
or
when ( any_in HV_SUPPLIES not_in_state ON )
   do RISE-ALARM
```

State Manager process
SM

1 / SMI++ domain

The key tool of SMI++ framework.
It is a logic Engine that reads the SML file and 'drives' the model described in the file



SML file   State Manager

- responds to external commands

- responds to asynchronous changes in its environment

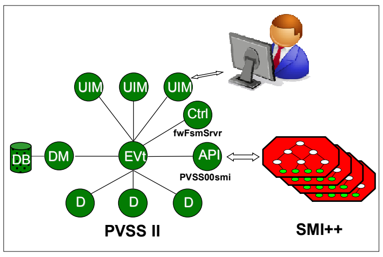
- sends out 'properly' sequenced commands to other domains and proxy processes

Designed using OOD tool ***Rational Rose/C++***

and coded in ***C++***

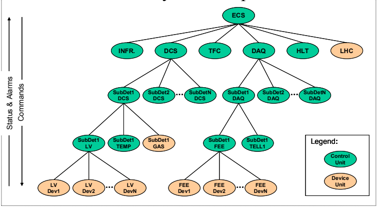
Use of SMI++ in LHC

All four **LHC** experiments at CERN (**ATLAS**, **ALICE**, **CMS** and **LHCb**) have adopted SMI++ framework integrated with SCADA (PVSSII) system

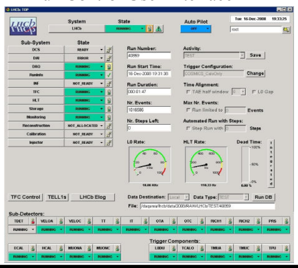


Example - LHCb

Control hierarchy of the experiment



Run Control User Interface



Tue 16-Dec-2008 19:33:25

System	State	Auto Pilot
LHCb	RUNNING	OFF

Sub-System	State
DCS	READY
DAI	ERROR
DAQ	RUNNING
RunInfo	RUNNING
INF	NOT_READY
TFC	RUNNING
HLT	RUNNING
Storage	RUNNING
Monitoring	RUNNING
Reconstruction	NOT_ALLOCATED
Calibration	NOT_READY
Injector	NOT_READY

Run Number: 40859
Run Start Time: 16-Dec-2008 19:31:38
Run Duration: 000:01:47
Nr. Events: 1016586
Nr. Steps Left: 0
Activity: TEST
Trigger Configuration: COSMICS_CaloOnly
Time Alignment: TAE half window 0 L0 Gap
Max Nr. Events: Run limited to 0 Events
Automated Run with Steps: Step Run with 0 Steps
Data Destination: Local Data Type: TEST
File: /daqarea/lhcb/data/2008/RAW/LHCb/TEST/40859

Sub-Detectors:

TDET	VELOA	VELOC	TT	IT	OTA	OTC	RICH1	RICH2	PRS
RUNNING	RUNNING	RUNNING	RUNNING	RUNNING	RUNNING	RUNNING	RUNNING	RUNNING	RUNNING

ECAL	HCAL	MUONA	MUONC
RUNNING	RUNNING	RUNNING	RUNNING

Trigger Components:

L0DU	TCALO	TMUA	TMUC	TPU
RUNNING	RUNNING	RUNNING	RUNNING	RUNNING

Conclusion

SMI++ Framework provides :

Special language (SML) to describe the real world to be controlled and to 'code' the control logic

Set of tools to implement, test and to use the designed control system

It encourages use of OO concepts such as
abstraction, encapsulation, modularity,hierarchy

In the past, it has been successfully used by **DELPHI** experiment at CERN in Geneva and by **BaBar** experiment at SLAC in Menlo Park for the design and implementation of their Run Control.

Currently, **all four LHC** experiments at CERN decided to use it either fully or partially for their experiment control. The complexity of the designed systems runs into thousands of State Managers