‘Sharable GUI Objects for the Operators’ Console

S.Dasgupta* and Isamu Abe
*Variable Energy Cyclotron Centre, 1/AF Bidhan Nagar, Calcutta 700 064, India
KEK, 1-1 OHO, Tsukuba, Ibaraki 305, Japan

Abstract

The accelerators over the world show a great variety, and the control systems of these are also varied. In the 3-tier control systems, the lowest levels are implemented according to the availability and historical situations regarding the hardware devices. The middle level is also influenced by the types of computers and the lower level devices. The top level (Man-Machine Interface) shows a considerable similarity because of necessities of human interactions. The article discusses an attempt for finding of the most universal objects of such an MMI. This should enable the designers to apply modern tools for object oriented designs of accelerator MMI. The Java language and its tools can be utilised to develope framework of MMI’s which are muti-platform. It will then become suitable for producing re-usable codes for tailoring and fast reproduction at different accelerators.

1 Introduction

Human beings conceive operation parameters and physical principles of accelerators in most similar ways. Therefore it is worthwhile to analyse the MMI objects of dissimilar accelerators for their mutually sharable attributes and behaviours. The Operation Console and its objects pertaining to the linear accelerator PF LINAC of KEK, Japan, and the cyclotron of VECC, Calcutta, are dealt with in this case. This report describes a work to produce specifications for MMI classes and also to specify the public interfaces of the underlying classes and objects to implement a sharable MMI. Descriptions of the decomposition of the concepts are vital for mutually aggreeable object classification.

2 Entity-description

It is necessary to develop concepts about attributes and behaviours of both underlying entities and MMI entities in a cooperative way.

2.1 MMI entities

In absence of high level automation, an accelerator operator relies on a sub-system oriented break-down in his conception of the machine. His interface objects need to implement the functions to control, monitor and log concrete physical parameters implementing the subsystems. Our operation MMI contains subsystems ‘Vacuum’, ‘Injector’, ‘Magnet’, ‘Rf’, ‘Beam Transport’, ‘Target’ and ‘Beam Diagnostics & Beam Handling’. Moreover an operator MMI can also have components to interact at a more macroscopic level, wherever advanced automation encompassing several sub-systems is available.

2.2 Underlying entities

Other kinds of entities which are part of the control system and deal with messages to and from console MMI are

Devices : The hardware and equipment to implement sub-systems
Auxiliary : Other types of entities, working as direct accessories to the Device class
Services : Power supplies, water-flow, air conditioning etc..
Parameters : Physical parameters that require monitoring, control or logging

Our MMI entities must contain the above underlying entities as their ‘known’ attributes.

3 Class formation - MMI classes

Based on studies and operation experiences, entities within individual sub-systems, were listed, as necessary attributes of the proposed corresponding MMI class.

3.1 Attributes- MMI classes

Class MMI_Vacuum
Pumps, FlowSwitches, OilLevel, Gauges, GateValves, Status etc..
Class MMI_Rf
Oscillators, Amplifiers, Co-ax. Switch, Setting, SignalMode, PeakAmplitude, Frequency, MacroFrequency, Harmonics, Phase, SWRatio, PowerSupplies, FlowSwitches etc..
Class Magnet
Coils, PowerSupply, Setting, AverageField, Field-Component, Harmonic_n, PeakAmplitude, FWHM, Rigidity, FlowSwitches etc..
Class MMI_Beam
Particle, Avrg Amplitude, PeakAmplitude, MacroWidth, Power, HarmonicPower, Oscillation Ampltd, Orbit Separation, Size, Profile, Phase, Emittance, WCMonitor, CurrentProbe, ProfileMonitor etc..
Class MMI_Injector
SpeciesMass, SpeciesCharge, BunchState, Electrodes, Pulser, Slit, Puller, PowerSupply, Current, Buncher, Lenses,
Solenoid etc.
Class MMI_BeamTransport

Quadrupole, AnalysingMagnet, EmittanceMonitor, Profile Monitor, WCMonitor, TwissParameter, Emittance, Acceptance, BunchLength, EnergySpread, etc..

Class MMI_Target

Collimator, Sweeper , GateValve, Angle, etc..

4 Class formation - underlying classes

All objects in the control system excluding the MMI objects are defined as objects of same or another Underlying class. One Underlying class is to take care of each 'Underlying entity' described earlier. The 'device' class here is the template for most of the devices and equipments that the MMI communicates to in the first order.

4.1 Attributes- underlying classes

Class Device

Equipments, Instruments, Control Panels, Status, On/Off Switch, Inc./Dec. Control, Setting, Range, FlowSwitches etc..

Class Auxiliary

Motors, Smart controllers, Instruments etc..

Class Services

Status, Magnitude , Mode etc..

Class Parameters

PeakValue, AverageValue, RMSValue, Phase, Frequency, MacroFrequency, HarmonicFrequency, Power, WaveForm, SetValue, HiAlarmValue, LoAlarmValue, Status etc.

5 Formation-root MMI class

Presentation schemes for the data logging, monitoring & control operations and messaging schemes to underlying objects have been designed to be uniform among different MMI objects. So similar working variables can be maintained for all. All these prompt for the construction of a root MMI classs, to serve as the parent for the MMI sub-classes.

5.1 'Group'

For generalisation, data presentations and control interactions are planned to occur groupwise. Operator configured individual 'group's will have a name and contain collections of parameters or control panel frames.

The class tree for the control system with special reference to MMI classes, can be specified as in Fig. 1.

5.2 Attributes-root MMI class

Device m_TheDeviceUnder[ ] : subsystem device objects under the MMI sub-class

Vector m_Params[ ] : interesting Parameters with whom message transfer is needed from the MMI sub-class

Vector m_DevPanels[ ] : interesting Control Panel Frames of the devices in the MMI sub-class

Hashtable m_ParamGroup : table of groupname vs. grouped Parameters / grouped Control Panels in the MMI sub-class

DataBase m_DBUnder : the section of Database containing dynamic parameter values under the MMI sub-class

File m_ArchiveUnder : the archived data about the MMI sub-class

5.3 Methods- root MMI class

Listings of all interactions with underlying classes for each kind of MMI were documented. Out of these, a greater number could be designed as isomorphic across different MMI classes, for both accelerators. These are filtered out as the generic methods of the root 'MMI class'.

A tentative list of such methods for the root MMI class are summarized below.

OpenGroup( ) : Show all params or control panels of the selected group in an appropriate display interface

MakeValueSetting( ) : prepare a list of settings for a named group (of parameters)

SetParam( ) : set a named group of parameters to the specified values

MakeAlarmSetting( ) : prepare list of alarm-settings for a named group (of parameters)

MakeParamGroup( ) : Populate Hashtables with data obtained from opening level dialog boxes

MakeParamsList( ) : Make list of parameters taken from a configuration file

ShowParamList( ) : Show names of all parameters and control panels of this MMI in a selection dialog box

MakeGraphSetting( ) : prepare graph settings for a group of parameters

AckAlarm( ) : acknowledge the alarm and reset

MakeGraph( ) : plot graph for group of parameters

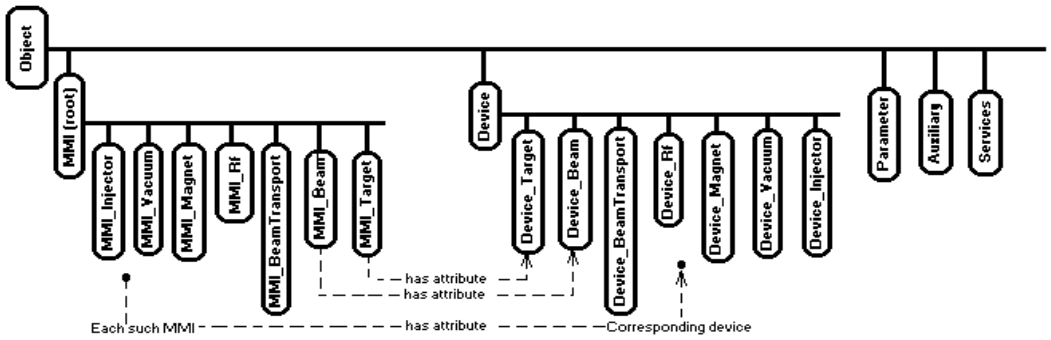

Fig. 1 MMI Class & Relationship

AckAlarm( ) : acknowledge the alarm and reset
MakeGraph( ) : plot graph for group of parameters
SetToArchive ( ) : set parameters in a group to pre-specified values (obtained from archive)

6 Vertical transactions in an example sub-class

A Sub-sytem MMI can be derived as extension of the root MMI class. Each of these, additionally, may have unique attributes and higher level methods. 'Phasing' in the PF LINAC and 'AutoTrim' in the VEC are such examples in respective MMI_RF. An example below sketches the details of such a MMI sub-class.

6.1 Subclass MMI_RF

This sub-class has all attributes and methods of a root MMI, in addition to some unique attributes and higher level methods specific to RF only.

Additional Member Methods

RfPhaseAdjustSystematic( ) : adjust RF phase at one or multiple points, keeping other parameters of RF optimised (automatically)
RefPhaseAdjustSystematic( ) : adjust Ref. phase at one or multiple points keeping other parameters of RF optimised (automatically)
IsolatorOperate( ) : on/off the isolator of the '*IPhiA'*
AttenuatorAdjust( ) : adjust the attenuator of the '*IPhiA'*
Stabilise( ) : enter a correction loop for optimising Dee-voltage, harmonic content etc. for a specified RF setting
FrequencyAdjustSystematic( ) : slide frquency up/down, keeping all other associated parameters of the RF optimised.
AccVoltageAdjustSystematic( ) : slide accelerating voltage up/down, simultaneously keeping other associated parameters of RF optimised
TuneToHarmonic( ) : tune resonator to a specified harmonic frequency

Through its Device_Rf object, or through the control Database object, the MMI_Rf object can exchange messages with the hardware and other entities of the accelerator.

6.2 Sub-class Device_RF

Member Attributes

Auxiliary[] : m_MasterSource, m_PreAmplifier[ ], m_Amplifier[ ], m_Klystron[ ], m_*IPhiA*Controller[ ], m_RefPhaseShifter, m_Trimmer etc.
Services[ ] : m_PowerSupply, m_Coolant-Supply etc.
Parameters [] : all measurable and controllable entities

Member Methods

Operate( ) : to operate on/off control
Regulate( ) : to operate continuous control
Measure( ) : to measure values of parameters

7 Implementation

The codes for the class declarations and definitions for the root MMI class have been done in Java, to benefit from its platform independence and network package. The starting codes for the prototype were developed with Vis. J++ under Vis. Studio 97 on Win95. A set of primitive GUI components have been utilised to prepare simplistic operation screens (fig. 2). With the availibility of more readily usable tools, GUI components specially suitable for an accelerator operation console, will be applied.

Implementation of higher level functions, in accelerator controls, is a continuous process. Because of a clear class-definition, extendibility[1] involves adding up new methods to a derived MMI, as and when developed.

Reference

[1] N.Kanaya, NIM in Phys. Res. A352 (1994) 499

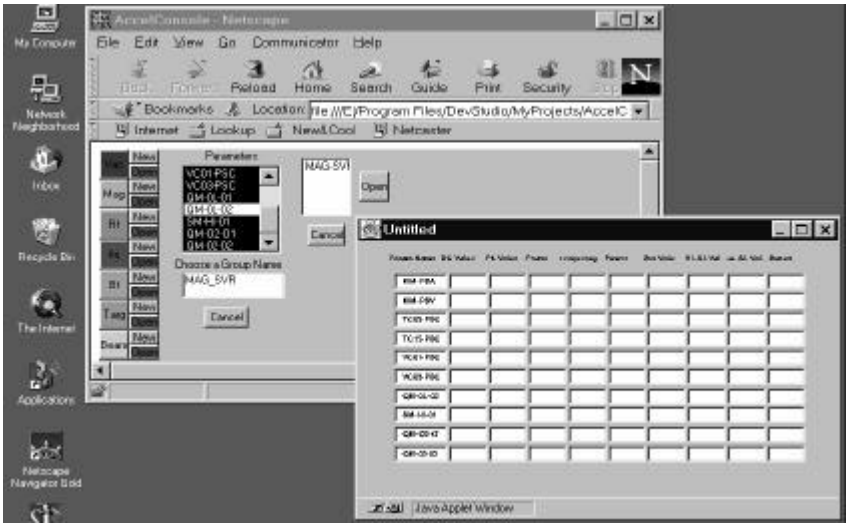

Fig. 2 A Simplistic Test Screen