Object-Oriented Control System Development Using the Smalltalk Language

I. Mejuev, I. Abe and K. Nakahara

National Laboratory for High Energy Physics, 1-1 OHO, Tsukuba, Ibaraki 305, Japan

Abstract

An approach to and the tools developed for accelerator control system creation are presented. The Smalltalk language environment provides a set of unique possibilities for solving this problem. The most important is the dependencies facility, which allows one to divide the domain model and user interface development. Our method uses the Domain Model Editor tool for developing an object-oriented model of an accelerator. The Chart Editor is used to define graphics for representing the accelerator objects states. The model and chart are stored in an object-oriented database. Using the database file, the Control Panel program scans the model conditions for the execution of operations and displays the state dynamics on a chart. The domain model can be linked to custom-user interface objects in order to implement control at runtime. The user interface objects are created using either the Control View Editor tool or VisualWorks environment. The problem of the control distribution over the computer network is also discussed.

1. Introduction

This paper represents our approach to creating accelerator control systems. Any control system that meets modern requirements should give users the possibilities to easily create and modify their accelerator model using convenient interface tools. For this purpose we have developed the Domain Model Editor, which describes an accelerator using a set of predefined standard classes. Objects created by the user are saved in the object-oriented database for later use. Also, the Domain Model Editor includes the possibility to create new user-defined classes. The domain model created with this tool comprises all of the data necessary for accelerator control.

At runtime it is important to display and update information concerning the current system state. For this purpose graphical representations of the states of the most important objects in the system are necessary. To manage such graphics and map graphical objects to model objects the Chart Editor tool is used.

While the Model Editor and Chart Editor are used in the model-creation phase, the Control Panel is a tool used at runtime. It implements a real control loop to check the conditions for executing system operations. Another part of the system is the Control View Editor, which creates GUI objects to implement custom control tasks, such as the direct execution of operations, the control of values and the providing of detailed information about the system objects.

In our system we widely use the dependencies facility of Smalltalk [1,2], which allows us to divide model development from graphics and user-interface tools development. Usually an interface object is registered as being dependent on the model object, so that it can handle update requests.

2. Domain model

In our system the Domain model of the accelerator is based on a set of fundamental classes. Those classes implement the basic features that the accelerator model contains. By using inheritance a user can choose the exact implementation and system decomposition that best fits his purpose. The main classes allow the user to describe object aggregation, to assign operations acceptable for objects and to define the conditions for operation execution. The condition equations use special kinds of objects: *values* which can be connected to equipment for measurements. The results of object-oriented analysis were first reported in [4].

ControlObject is a root system object; any object in the domain model belongs to this class. *ControlObject* properties are:

name (name used to identify the object, such as “Vacuum system”), and

state (an object state which can be displayed on the states chart [5], such as “Normal” and “Alarm”).

Also *ControlObject* contains associations for supported *operations* and includes *conditions*, *values* and *component objects*.

Operation is an instance of the ***ObjectOperation*** class. A user can define operations for different object types. For example, there can be operations that are applicable for klystrons, vacuum pumps and so on. Any operations should be first defined in an object class. When an object is created, a class constructor associates the object with the correct operations.

The condition (instance of ***Condition*** class) defines the rules for the execution of operations. Every condition is created from some abstract condition specification, like $a<x<b$ with the addition of a parameters list and association to an operation. The condition specification, together with parameters, forms a logical expression. Such an expression is evaluated at runtime while polling and if is it valid, the operation is executed. Another property of the *Condition* class is polling type. It specifies whether it is necessary to create a special polling process for this condition or to use a common one. Condition arguments can be either constants or *values*.

A value is a generic datum holder, an instance of the ObjectValue class. The value magnitude corresponds to the *value* property of this class and value type (string or number) to the *valueType* property. To retrieve and set the value, *setValue* and *getValue* messages are used. These messages redirect the request to low-level software connected to the equipment; for instance, it could be the input-output channel for a VME slot.

Accelerator, ***Subsystem*** and ***SubsystemObject*** classes are inherited from the *ControlObject* class. *Accelerator* is the main root object; it contains a set of subsystems and an object states chart description [5]. The *Subsystem* is an object container for creating an rf system, vacuum system and so on. It may contain a set of global subsystem values. A *SubsystemObject* is used to represent objects which are subsystem components, such as klystrons and pumps.

These classes form a basis for a control system model. If necessary, a user can refine the domain model by using an inheritance. The basic classes for system design are presented in Figure 1 in Rumbaugh notation [3].

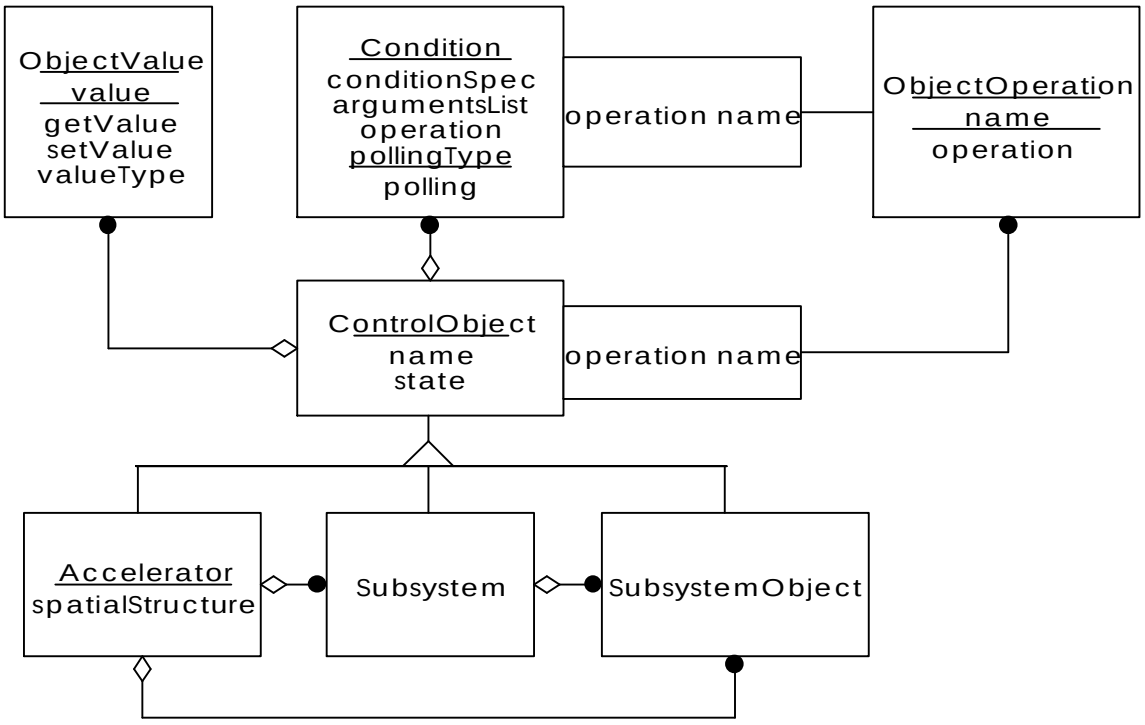

Figure 1: Basic system classes

3. Control loop

The control loop polls all of the model conditions. It is implemented by the Control Panel tool. The Control Panel uses an object database created by the Domain Model Editor. It sends message *polling* to all conditions from top-level *Accelerator* objects to low-level *SubsystemObjects*. The *polling* method checks the logical expressions for operation execution, and executes operations when necessary. Also, the Control Panel draws a states chart and establishes links between the domain model objects and chart elements [5].

The Control Panel creates the main polling process as well as auxiliary processes for conditions with corresponding polling types. Using additional polling processes allows one to increase the polling rate of the most

important system parameters and to improve the system performance. Smalltalk language contains built-in process control possibilities which simplify the development of the Control Panel.

4. Tool relationships

The system core is an accelerator domain model which is stored in an object-oriented database. We use the VisualWorks BOSS file (Binary Objects Streaming Service) for this purpose [2]. However, before storing an object using BOSS it is necessary to remove any dependencies so as to exclude the possibility of an infinite recursion. The BOSS database is used by all system components as a control data repository.

The BOSS file is created by the Domain Model Editor. This program includes possibilities for creating objects as well as classes. It is possible to operate with values, operations and conditions on both the class and object level. An example of the objects' view is shown in the Figure 2 and the classes' view in the Figure 3. Classes play the role of object factories; any features implemented on a class level can be easily replicated. However since by using the Domain Model Editor there is a possibility to customize an object by adding extra values and operations without modifying the class definition, its use should be constrained, since it contradicts the conventional methods of Object-Oriented Design.

A part of the Domain Model Editor is the Chart Editor, which is used to create a states diagram and to connect model objects to chart elements. The Chart Editor just assigns object references to chart element variables. Real dependencies are established at runtime by the Control Panel. The Chart Editor tool is described elsewhere [5].

The Control Panel program is a runtime tool, which implements real control using data created by the Domain Model Editor. The Control Panel starts and stops the conditions scanning processes and also presents a chart with current object states. However, it is a self-contained program which doesn't support the initiation of operations by a control system operator. To build tools for operator control the Control View Editor is used.

The Control View Editor creates data for *control view,* a program used for operator control. Using the Control View Editor, a set of GUI objects can be created, such as: ControlObject value views, ControlObject value controls (sliders, combination boxes), action buttons, action lists and action menus.

The *control view* works together with the Control Panel and shares the same data using a BOSS database file. If the standard set of objects is not sufficient, a control view can be created using the VisualWorks environment. In this case the programmer is responsible for creating an interface between the control view and the domain model.

5. Distributed control

Large control systems can have a large number of values to control; they also can be territorially distributed, so it is necessary to have support for distributing control tasks over the computer network. The main goal is to give computers the possibility of using objects from the models of other computers, requiring value access and remote operation execution. For this purpose we use *remote objects,* which are present in the local computer model, but are really implemented on the remote computer (host computer) domain model. Every remote object includes the name of its host computer. When a remote object value or operation is required, the request is redirected to the host computer by a network dispatcher.

By using the remote objects, condition scanning work can be distributed over a set of computers. Each database representing a part of the system can be placed on a different network segment. For a local database the BOSS file is sufficient. To take advantage of client-server technology, we can also use SQL servers. VisualWorks currently supports Smalltalk objects mapping to Sybase and Oracle databases.

6. Conclusions

The approach introduced in this paper shows that Object-Oriented Design offers a wide range of new possibilities for the development of accelerator control systems. A set of tools has been developed which allow any user to easily customize the system structure. The object-oriented programming style makes it possible to reduce the system maintenance and development costs. The Smalltalk language we used includes process control, a dependencies facility and visual programming tools that aid large-scale project development.

7. References

[1] An Introduction to Object-Oriented Programming and Smalltalk; Lewis J. Pinson, Richard S. Wiener; University of Colorado at Colorado Springs, Addison-Wesley, c 1988.

[2] VisualWorks User's Guide; ParcPlace Systems, c 1994

[3] Object-oriented modeling and design; James Rumbaugh [et al.] Englewood Cliffs, N. J.; Prentice Hall , c 1991.

[4] Application of an Object-Oriented Analysis for the PF Linac Control System Development; Mejuev I., Abe I. and Nakahara K.; Proceedings of the 20th Linear Accelerator Meeting in Japan, Osaka, 1995, p. 212.

[5] Graphical Representation of Objects' State for the PF Linac Control System; Mejuev I., Abe I. and Nakahara K.; Proceedings of the 10th Symposium on Accelerator Science and Technology, October 25-27, 1995, Hitachinaka, Japan, p. 292

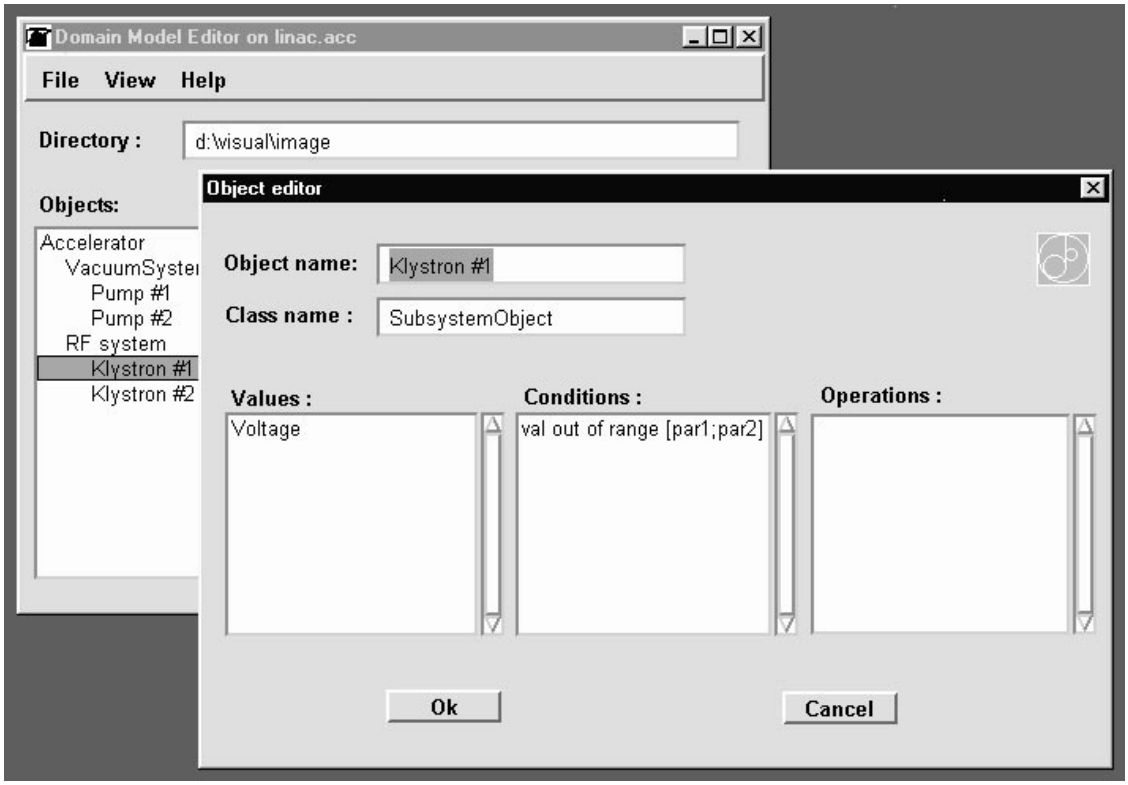

Figure 2: The Control Model Editor (objects view)

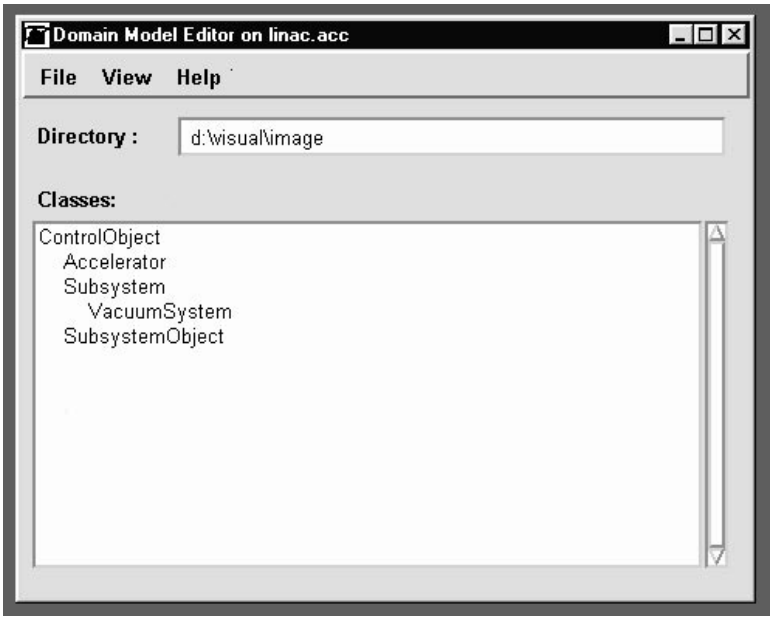

Figure 3: The Control Model Editor (classes view)