P-25

Application of an Object-Oriented Analysis for the PF Linac Control System Development

Mejuev I., Abe I. and Nakahara K.

National Laboratory for High Energy Physics (KEK), Oho 1-1, Tsukuba, Ibaraki 305, Japan

Abstract

An Object-Oriented Analysis for developing the PF Linac control system has been carried out. We used the Rumbaugh method for making a logical model of the Linac with its subsystems. A generic system class which provides aggregation relations support, value control and operations is defined. An operation concept which provides standard operations support is introduced. A discussion concerning object-oriented decomposition of complex systems is also presented.

1. Introduction

Software development for a modern accelerator control system requires highly productive software engineering tools which can significantly decrease the development and maintenance costs. In line with our group's intention to use object-oriented technology for all phases of software development, this paper describes a new design concept for creating a logical model of the accelerator in the analysis phase. The Rumbaugh OMT method was introduced using the ObjectMaker tool [1].

The Object Modeling Technique consists of three main models: Object Model, Dynamic Model and Functional Model. The Object Model describes the real world or problem domain in terms of static objects and the relationships between objects. The Dynamic Model describes state transitions and events which cause them. The Functional Model describes methods for the Object Model or events for the Dynamic Model. The Object Model is the most important of the three models. The system should be built around the Object Model because this model closely corresponds to the problem domain. In this paper we emphasize the Object Model development.

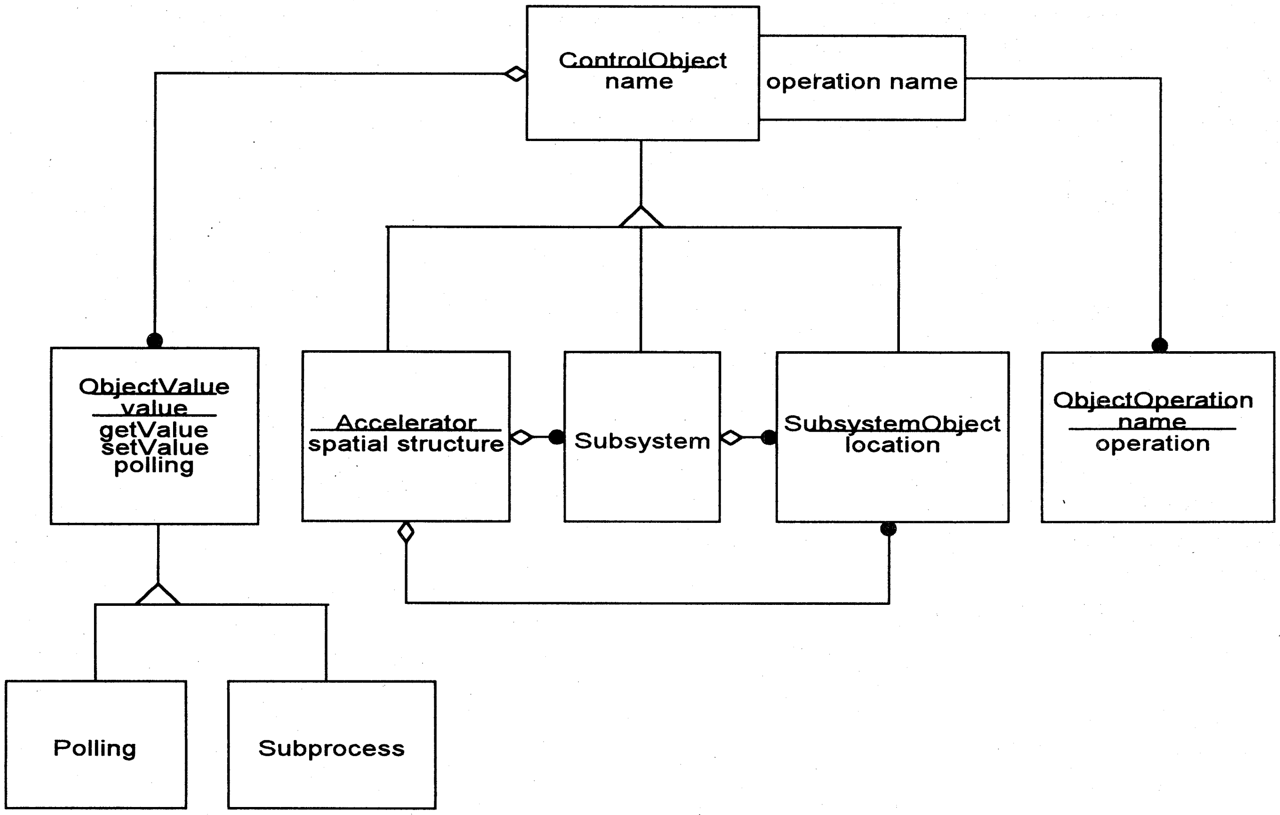
While defining the Object Model of the Linac several ideas must be taken into consideration. The Object Model must be adequate to reflect the complex structure of the accelerator, it must also hide internal details concerning different subsystems when necessary. Also, we must create a standard methodology for operating heterogeneous accelerator subsystems. Thus, while creating the object-oriented model it has been decided to place all common features of an abstract accelerator subsystem object at the top class of the system hierarchy. Those common features are considered to be: aggregation relations, values and operations.

Smalltalk language provides a unique possibility to develop a domain model of the accelerator system separately, without paying any attention to the user interface objects. We can realize this by using its MVC (model-view-controller) concept. In the first stage, we can develop only a system model, and then connect it to the view and controller representing the user interface. The connection is possible owing to the generic Smalltalk dependencies technique. In our case, the model has a set of views as its dependents, so that views will be notified about any model changes [2]. In this paper we consider the first stage: system model development.

2. Design concept

In accordance with our approach, in the initial stage all system classes must be presented in Rumbaugh notation using ObjectMaker tool. In other words, at first we create a domain model of the accelerator.

The main classes of the Object Model are explained below.



A ***ControlObject*** is a generic system object. We derive from this class all important parts of the Object Model. *ControlObject* properties are :

* supported *operations*

* a set of *values*

* *dependent components*

Operation is an instance of the ***ObjectOperation*** class (abstract class) with the possibility to derive for example control operation class, test operation class and so on. Every action in the system is represented as a set of operations. When some operation is applied to the object, we first check whether it is an available operation; only in this case we can proceed. Thus, an operation concept is created to guarantee that only acceptable operations can be applied to a *ControlObject* instance. An *Operation* object contains an operation name as well as a reference to a Smalltalk method which implements the action. In our model we are using qualified association to connect the operation to the object. Qualified association allows objects to refer to operations by name.

This concept also provides the possibility to show to the control system user a list of available operations for selected object. The object of interest could be selected using a graphical representation of an accelerator system.

Value is a generic datum holder, an instance of the ***ObjectValue*** class. Every parameter managed by the system is represented as an *ObjectValue* instance.

Value object features are:

* Standard "setValue:", "getValue" and "polling" messages support for any value type.

* The possibility exists to either control the value by polling or to create a process which can manage that value. In the case of polling there is the possibility to set a polling priority.
* *Condition* and *action* which is invoked if the condition is valid. During the scanning process the condition is used to determine whether it is necessary to do anything or not. The condition contains an expression which includes a value magnitude and is evaluated by the scanning process.

To express hierarchy relations we inherit the *ControlObject* class from the standard *List* class in the Smalltalk environment. However, several operations, such as copying and comparing must be overridden. We also found it convenient to use Smalltalk dependencies (mentioned above) to provide notification of a high-level object about the state of its components.

The ***Accelerator*** is the main root object. The *Accelerator* contains a set of subsystems, and description of the spatial distribution of low-level objects. That includes the data structure representing the accelerating sections, and for each section objects located there. Also, as for any instance of the *ControlObject* class we can define values and operations; in this case those which are important for the whole accelerator are used.

Subsystem: superclass representing accelerator subsystems commonality. It is possible to make an inheritance for creating an rf system, vacuum system and others. A subsystem contains a set of low-level subsystem objects. It can also contain a set of global subsystem values and operations.

The ***SubsystemObject*** class is derived from the *ControlObject* class; also, however, the location of this object in the accelerator structure is added. A *SubsystemObject* represents a very elementary object in the system. In the aggregation hierarchy it is a part of either the *Subsystem* or *Accelerator* class instances.

As mentioned above, system control is achieved by executing actions which are instances of the *ObjectOperation* class. Actions can take control in two ways. The first is when the scanning process calculates the value condition; if it is valid proper action is executed. The second one is external action originating from a control system user who can directly execute any actions for any system object.

3. Conclusion

The Object Model described above is intended to be a domain model for the future control system. The Rumbaugh method allows us to develop a standard approach for representing an accelerator in terms of the Object Model. Further, this model must be supplied with a user interface and a core control program. For this purpose Smalltalk language (Visual Works) is now under consideration. Such features as dynamic type identification, exception handling as well as the pure object-oriented nature of Smalltalk will stipulate this concept realization.

4. References

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

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