PB16

Graphical Representation of Objects' States for the PF Linac Control System

Mejuev I., Abe I. and Nakahara K.

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

Abstract

This paper describes our approach for creating of a graphical representation of the states of accelerator objects. For every object which needs to display its state dynamics we create a chart element. Any changes in the object state are immediately represented on the chart of this object. We use the objects dependencies facility supplied by Smalltalk to keep track of the objects' states changes. This allows us to remove any graphics related features from the Object Model of the accelerator. Thus, the Object Model can be developed separately and reused in a number of applications.

1. Introduction

Accelerator control requires a convenient graphical representation of the present state of the objects that the accelerator contains. For instance, there could be the possibility either to display accelerator sections, indicating the section state, or, a more detailed representation also including section's objects. For accelerator subsystems, such as the vacuum system and rf system, it would also be convenient to represent the subsystem hierarchy, including the subsystem objects in order to gather information about their states. Usually, in the present control systems graphical routines are mixed with both code and data for the control, this makes it difficult to change the graphical part of the system without reviewing of the control data and code. In this paper we represent an approach that divides a visual representation from model. The Smalltalk language that we used (VisualWorks) provides convenient possibilities for such division. These are object dependencies concepts and application of this concept for the user interface creation: MVC (model view controller) concept [1, 2]. Two separate system part, model and interface, are represented below.

2. Model

The key point concerning the accelerator domain model is ***ControlObject*** class (model object class). This generic system class is used for all system objects representation. The accelerator, subsystem and subsystem objects are all *ControlObject* class instances. The *ControlObject* class defines a number of properties described in [4]. However, for the present consideration only the "state" property is important. This property contains symbolic values representing the current object state. The state examples are: "normal", "alarm" and so on. The object state corresponds to the "state" variable of a *ControlObject* instance. Since this variable is updated every time some object value is changed, it always contains the object state valid at the moment. Further, an instance of *ControlObject* can have a set of dependent objects. The standard dependencies facility supplied by Smalltalk guarantees that if the "changed: #aspect" method is invoked for the object, every dependent receives an "update: #aspect" message. The aspect parameter is used to separate independent object modifications and to suppress

redundant updates [2]. The Dynamic model for the change-update method together with the Object Model for the *ObjectChart* (explained below) and *ControlObject* classes are represented in Fig. 1. We use Rumbaugh method notation [3].

3. Interface

The objects' states are represented in the chart (Fig. 3). A chart element contains fields for the name (top) and state (bottom). The state field dynamically represents the model object state. In the simplest case, it could be only a text string, like "normal", and "running"; however it is possible to map some states to pictures.

4. Update method

For a graphical representation of the accelerator object' states we create the ***ObjectChart*** class. An *ObjectChart* instance corresponds to an accelerator chart element; it is responsible for the dynamic representation of some object's state. The model object reference is contained in the "object" variable of the *ObjectChart*. To assign and get this variable we use methods "object:" and "object" correspondingly. When we set a new object to display by sending the "object:" message, the receiver is registered as a dependent of this object.

If the model object "state" variable is changed through the "state:" message, this object sends the message "changed: #state" to itself. That causes the message "update: #state" be sent to object dependent: *ObjectChart*. Having received an "update: #state" message the *ObjectChart* instance retrieves the state variable value from the model object which is stored in its "object" variable. After that, *ObjectChart* draws on the user terminal picture or text corresponding to the present model object state.

This method allows us to remove any messages responsible for graphics from the domain model. The domain model can be developed independently, and multiple graphical modules representing different aspects of its state can be easily connected to it. It is also possible to reuse graphical interfaces with different models. It is supposed that this approach is capable of reducing the system development and maintenance costs.

5. Conclusions

The method represented above could be used to display a large variety of control system aspects. First, it can represent accelerator subsystems with the "normal" or "alarm" states for every subsystem object. Also, it is possible to collect information about states of the accelerator sections. In this case the state resumes a summary of all the section objects states.

Although in this paper we allow only one object connection to an *ObjectChart* class instance, in the future we will discuss the possibility for a multiple objects connection. With multiple connections the problem of states priority appears. Another way to resolve the multiple objects problem is to create composite object in the domain model representing the objects aggregation.

6. 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.

7. Figures

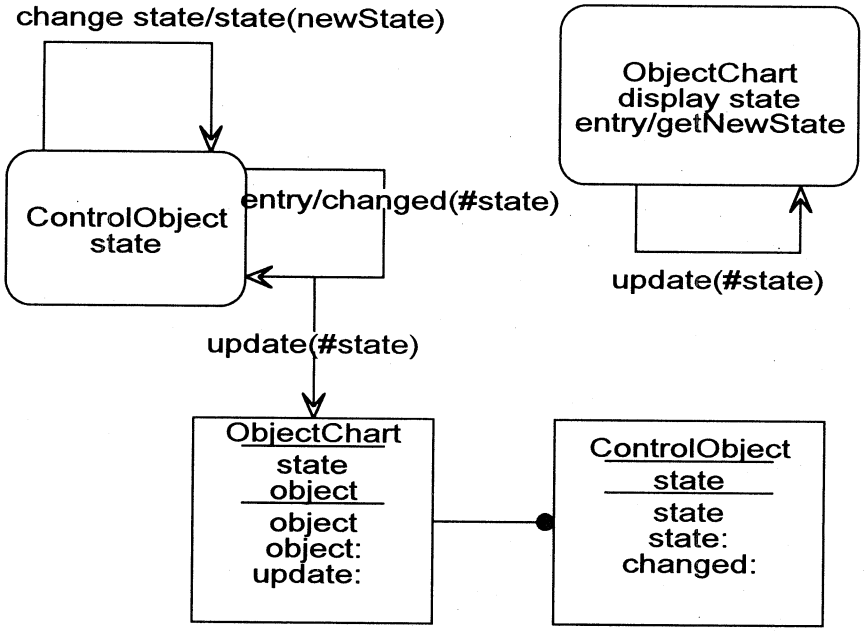

Fig. 1

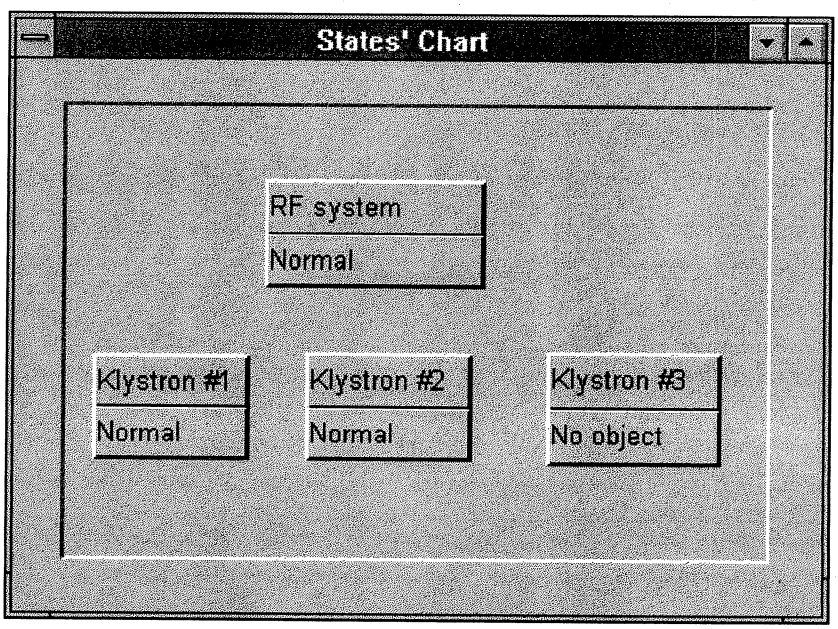

Fig. 2