ACCELERATOR OPERATION MANAGEMENT USING OBJECTS*

H. Nishimura, C. Timossi, M. Valdez, Lawrence Berkeley Laboratory, Berkeley, CA 94720 USA

Conflicts over control of shared devices or resources in an accelerator control system, and problems that can occur due to applications performing conflicting operations, are usually resolved by accelerator operators. For these conflicts to be detected by the control system, a model of accelerator operation must be available to the system. We present a design for an operation management system addressing the issues of operations management using the language of Object-Oriented Design (OOD). A possible implementation using commercially available software tools is also presented.

I. THE PROBLEM OF OPERATION MANAGEMENT

The Advanced Light Source (ALS) [1] is a facility operated at Lawrence Berkeley Laboratory to produce light for researchers. The facility, composed of an accelerator surrounded by optical beamlines, operates in many states such as start-up and shut-down, injection, and production with light being used by experimenters. Many activities or operations are typically in progress during each of these states; the danger is, that they may conflict. For example, when the storage ring is filling it is not appropriate for the control system to perform closed orbit correction. Conversely, during production, orbit correction should be active but it should have exclusive control of resources on which it depends, such as corrector magnets. Also, depending on the state of the accelerator, certain operations should not be allowed at all. Certainly, calibrating beam position monitors while orbit correction is active, could result in the unwanted loss of beam. Ultimately, these types of conflicts are prevented by the good sense of the operations crew, but it seems reasonable that a system that had knowledge about the operation of the accelerator could prevent these conflicts.

II. THE DESIGN PROCESS

Before starting the design, we first had to decide on the requirements but we were also very interested in methods and tools for supporting object-oriented design. We looked at two methodologies and tools.

A. Requirements

The Operation Management System is configured with a set of accelerator operations, and each operation having a list of necessary resources. The system must not allow conflicting operations to occur. There are essentially two types of conflicts: operational and resource. Resource conflicts occur when two operations attempt to control or lock the same resource (e.g., a bend magnet.) Operational conflicts do not occur on a resource level, but rather when one operation is able to affect another. If an attempt is made to start a conflicting operation, the system will identify the source of the conflict but will not attempt to preempt any of the active operations. When any type of conflict occurs the system will identify the source of the conflict by computer host name, operation name, and, if necessary, resource name. Since control operations are performed across a network of heterogeneous computers, the system must also operate in this environment.

B. Object-Oriented Design

Although there are slight differences in OOD methodologies, our design process was typical. First we identified the objects and their relationships (referred to as the static model). Next we examined typical scenarios for the system to determine the objects' methods (referred to as the dynamic model). Finally, we repeated the preceding steps until a model spanning a useful number of scenarios is complete. In fact, we are still iterating through this process.

C. Software Tools

We started the design wanting to take advantage of one of the new OOD tools which would not only aid in the design phase, but also in the implementation phase with its ability to generate C++ code. Since much of our existing application code is in C++, code generation was certainly useful in a tool. We worked with two methodologies described by: Rumbaugh [2], and Booch [3]. Also, we used two different software tools: OMTool [4] and Rational Rose/C++ [5]. Our choice of the Booch methodology for this design, was driven by our preference for Rational Rose (both the tools and the methodologies are going to be merged). We chose the ObjectStore Object Oriented Database [6] for storing persistent data because of its transparency in C++ programming, its support of data models we wanted to use

* Work supported by the Director, Office of Energy Research, Office of Basic Energy Sciences, Material Sciences Division, U.S. Department of Energy, under Contract No. DE-AC03-76SF0098

(e.g., lists and sets), and its locking model that we hoped to use to support resource locking.

III. THE DESIGN: CONCEPT

During the design phase, we attempt to model the behavior of the operations management system. The modeling process consists of first identifying the classes and then examining operational scenarios to make sure the model is complete.

A. The Object Model

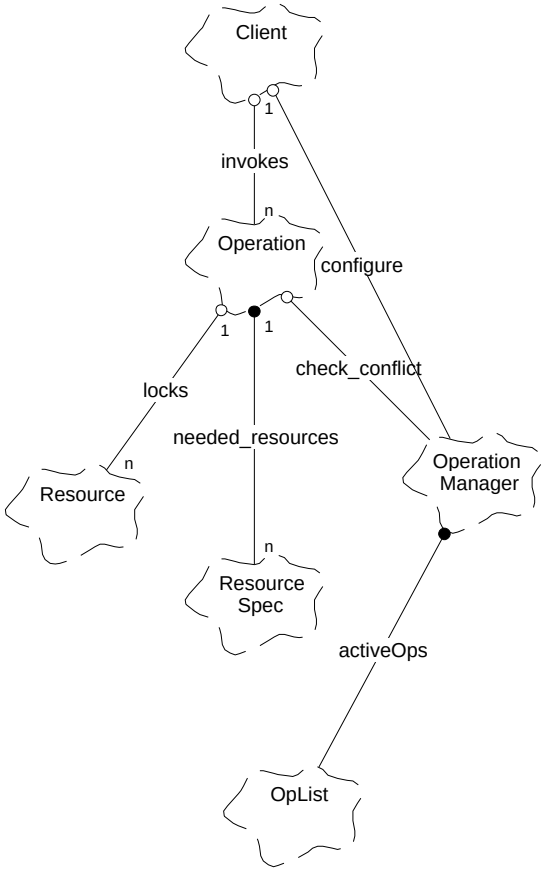
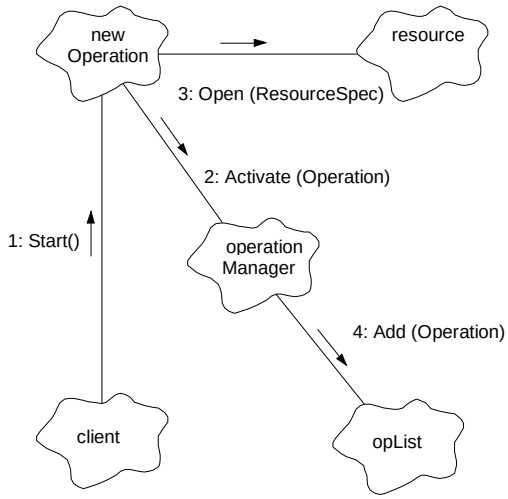


Figure 1: Class Diagram

Figure 1 is the class diagram (*static model*) of the system in Booch notation. Briefly, the broken clouds represent the *classes* (usually the nouns in the statement of the problem). We used the convention of capitalizing only class names and not objects. The lines name the relationships between the classes. These lines are adorned with circles denoting the type of relationship (open represents *uses* and closed represents *has-a*) and numbers representing the cardinality. Figure 1 reads: exactly one **Client** (class) uses n (many) **Operations** along with an **Operation Manager**. One **Operation** uses many **Resources**, contains many **Resource Specs** and uses one **Operation Manager** which contains an **OpList**.

B. Starting an Operation



Starting a new operation:

1: The client starts the operation.
2: If the operation doesn't conflict with another operation
3: and it can open the resources it needs,
4: then operation is added to the list of active operations.

Figure 2: Object Diagram

Figure 2 is the object diagram (*dynamic model*) of the particular scenario in which a new operation is invoked. The (closed) clouds represent specific instances of objects, with the lines representing messages flowing between the objects. The arrow points to the receiver of the message, and is labeled with the method that is invoked in the receiver. Figure 2 reads: A **client** (object) does a *Start()* on a **new Operation**. The **new Operation** asks the **operation Manager** to *Active()* it. The **operation Manager** determines if an operational conflict exists with the currently active operations contained in the **opList** object and the **new Operation**. If none exist then the **new Operation** is added to the **opList** object. If *Activate()* returns successfully, the **new Operation** then attempts to *Open()* its resources. If the *Open()* succeeds, then the **client** can manipulate its resources and continue.

IV. THE DESIGN: DETAIL

A. Operation

An operation object contains a unique identifier and a list of Resource Spec objects. Each of which identifies some resource that is needed for the operation to begin. When an new operation starts, it uses the Operation Manager to check for conflicting operations. It then uses its Resource Spec list to identify the resources it needs and attempts to lock them.

B. Operation Manager

The Operation Manager object contains the information needed to check for operational conflicts. This data is structured as a table of allowed operations stored in an ObjectStore database. This database is pre-configured with the allowed operations for a given state. The operation manager also contains the list of active operations in progress.

C. Resource Spec

The Resource Spec objects contain the resource information needed by the operation. Each Resource Spec object is maintained in a database and includes a resource name and the default settings or parameters for that particular resource. Because the default parameters or settings are included in the Resource Spec objects and not the Resource object itself, an operation can easily have variances of itself.

D. Resource

The Resource objects are used by the operation simply to provide a way in which accelerator resources can be locked. The Resource objects are maintained in a database pre-configured with the available accelerator resources. In some cases, exclusive write access to a resource is required, due either to the nature of the operation or to the nature of the resource. In either case, the resource is marked with the identifier of the operation with exclusive access. Also the host computer of the locking client and client information is kept in this object as well. This is useful for knowing which client on which computer is holding what devices. In general, the identifier of each operation using the resource is kept in the resource object as well as the host computer that the client is located.

E. Starting an Operation

To startup successfully, an operation must be able to mark each resource object in the database to indicate that it is in use by the operation. If one of the resources cannot be marked, because another operation has an exclusive lock, the operation must give up and restore the database to its initial state. Further, race conditions between operations simultaneously marking resources must be prevented. These requirements are implemented using ObjectStore transactions and the locking model; both features are designed to assure the database has a consistent state. The new operation first opens a transaction to the resource database. It can then read to see if any other operation is using the resources it needs. If the resource is free, the operation can put its signature in the object, thus enabling other operations to see which client on what host is using a particular resource. The write to a resource object will either put a write lock on the database, preventing any other operations from writing to it, or will block until the lock can be obtained. Next, if all the resources are available, the transaction is closed, the database is updated, and the write lock is removed. If all the resources are not available, the transaction is abandoned, the write lock is removed, the client is notified of the failure, and no change is made to the database.

V. STATUS AND DISCUSSION

A. Object-Oriented Design

Both the methodologies and the tools are evolving (the Rumbaugh and Booch methods are merging for instance). They are new enough that changes are still coming rapidly, but they are stable enough that some useful principals and tools are present. Certainly, the tools are already worth the investment in time to learn to use them. At worst, they produce quality documentation for a design, at best they generate, and regenerate code and documentation as the design changes. We have also found that the simple principle of separating the design into static and dynamic models provides a useful approach to the design, at least for small systems. Although this system is not yet complete, the process which allows us to expand and eventually complete it remains the same.

B. Future Work

The current model lacks the ability to identify operations that only partly conflict. This means that operations that do not necessarily operationally conflict throughout their whole extent are forced to avoid each other. For example, suppose Operation A and Operation B both conflict only in the beginning of their operations. This model would not allow them to be active together regardless of where they were within their own operational extent.

VI. REFERENCES

[1] "1-2 GeV Synchrotron Radiation Source, Conceptual Design Report," LBL PUB-5172 Rev. LBL,1986. A. Jackson, "Commissioning and Performance of the Advanced Light Source", IEEE PAC93, 93CH3279-7(1993)1432.

[2] James Rumbaugh et al. *Object-Oriented Modeling and Design*. Prentice Hall. 1991.

[3] Grady Booch. *Object-Oriented Analysis and Design*. Benjamin/Cummings. 1994.

[4] OMTool, GE Advanced Concepts Center, King of Prussia, PA.

[5] Rational Rose/C++, Rational Software Corporation, Santa Clara, CA.

[6] ObjectStore, Object Design Inc., Burlington, MA.