MANAGEMENT OF CONTROL SYSTEM DATA IN AN OBJECT-ORIENTED DATABASE*

M. Bickley, B. Bowling, S. Witherspoon, Continuous Electron Beam Accelerator Facility, Newport News, VA 23606 USA

Abstract

The Continuous Electron Beam Accelerator Facility (CEBAF) will use object-oriented databases (OODBs) to manage accelerator control system data. The managed data will include algorithmic detail of low-level controls, operationally specified control parameters and high-level accelerator modeling data. Tools have been built to support a variety of database users, including developers of both low-level and high-level software, machine operators and accelerator physicists. The planned databases will be both flexible and fast, and should prove to be ideal tools for meeting the diverse needs of their users.

Introduction

CEBAF plans to integrate as much accelerator-related data as possible using a single general-purpose object-oriented database management system (ODBMS). The commercial database system used at CEBAF is ObjectStore, sold by Object Design Incorporated [1]. The database tool itself is closely tied to the language C++. Persistent objects are created using the C++ function *new*, which is overloaded to support the generation of these objects within a database. The first step in the database development process at CEBAF was to identify the kinds of data most suitable for management using the ODBMS. The two types of data chosen were low-level control algorithm data and high-level accelerator modeling data. These data types were selected because of their need for management tools, and their suitability for objectification. More detailed descriptions of the data, and summaries of their integration into OODBs follow.

The first section of this paper is a discussion of the development of a control algorithm database. Background information concerning the data and how it is used as part of the real-time control of the accelerator is presented, as well as how the data are generated by developers of the control algorithms. Following this discussion is a brief introduction to the data objects themselves and a presentation of the data organization within the OODB. The last part of this section discusses how tools constructed with the ODBMS are used by accelerator staff and some performance measurements.

The second section of the paper presents information about the accelerator model database. Firstly we address the origin of the data, and the motivation for using the ODBMS to manage it. This is followed by the data structure of the model objects, the organization of the OODB, and how the model database is used.

I. CONTROL ALGORITHM DATA

Background

The software that controls the CEBAF accelerator is the Experimental Physics and Industrial Control System (EPICS). EPICS is a distributed system which acquires data and executes control algorithms using a real-time operating system running on single-board computers, known as Input-Output Controllers (IOCs). Graphical user interfaces and other client applications communicate with the IOCs using a network protocol [2] based on TCP and UDP.

In EPICS control algorithms are built from atomic elements called records. Each record is made up of a collection of data values, known as fields. Each record has support software which is executed on an IOC when the record is processed. At the time the record is processed, the support software uses the data values associated with that instance of the record to perform the record's control functions.

CEBAF uses 37 IOCs to control over 5000 physical devices. The great majority of these devices are duplicated many times. For example, CEBAF has 42 RF cryomodules, 117 viewers and 550 4-channel BPMs. The algorithm for controlling each of these duplicates is identical; it is only the data associated with each that is different (e.g. hardware addresses or calibration data).

The set of devices controlled by a particular IOC is determined by their proximity to the IOC. This means that the accelerator layout determines which control algorithms and therefore which records execute on a particular IOC. Each different type of device is controlled by an algorithmic unit, or application, which is made up of a number of EPICS records. Because each instance of a device type is controlled identically, there are no algorithmic differences within the executing control software, only data differences. This means the algorithm can be replicated for each instance of each device type.

EPICS provides a powerful environment for developing a modern, flexible control system. It does not, however, provide tools for replicating or managing the more than 100,000 records that make up the CEBAF control system. One of the goals in

* Supported by U.S. DOE Contract DE-AC05-84-ER40150

using an ODBMS is to provide a mechanism for managing those records. A second goal is to pass responsibility for managing operational data to the machine operations group, removing software developers from data management responsibilities in this area.

Database Objects

The control algorithm database contains a variety of different objects. Most of the objects are subclassed from a single object, *schemata*. A *schemata* object has the attributes **name** and **components**. A **name** is a character string used as an identifier for the object and **components** is a collection of pointers to other objects. The pointers can be references to any object, and almost universally refer to objects also subclassed from *schemata*. In this manner, the OODB representation of the entire control algorithm database is a tree of *schemata*-derived objects, Figure 1.

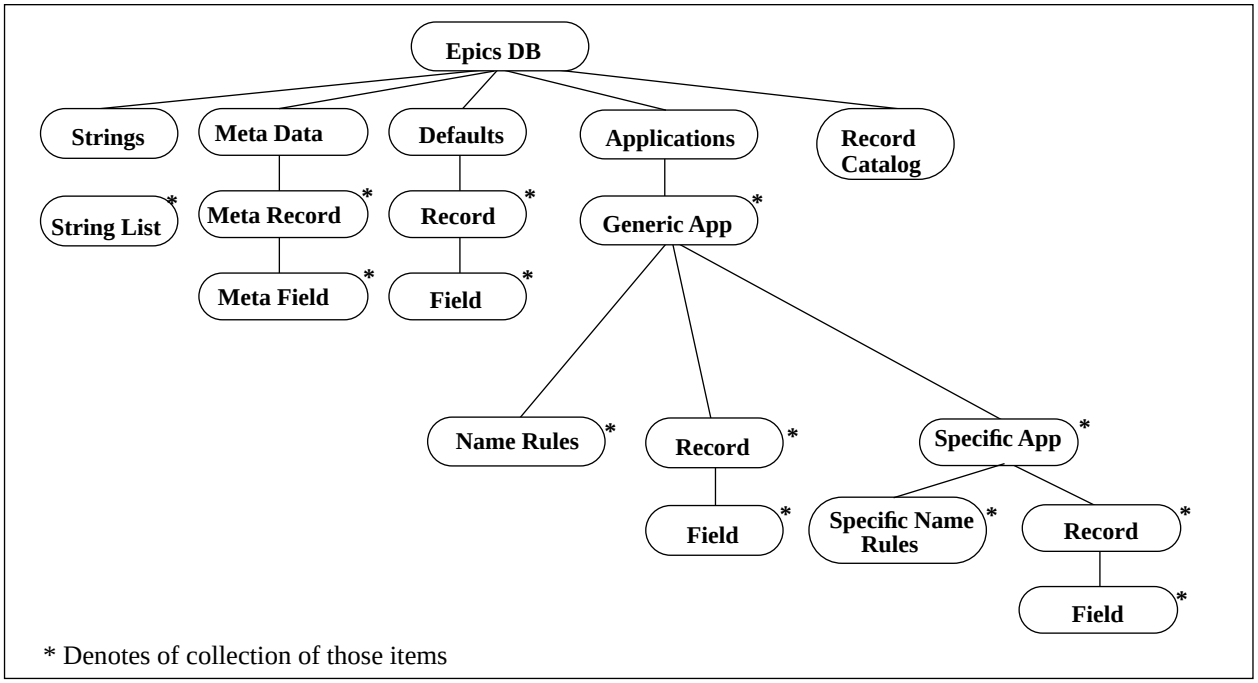

Figure 1: Schematic of the Control Algorithm Database

The control algorithm database is navigated using the name of the object being searched for. Starting from a parent node, the **name** of each of the **components** of the parent *schemata* node are compared against the name being used in the search. A name match indicates that the correct object has been found. When the number of references in a **components** collection is large, making a linear search time-consuming, the ODBMS provides indexing methods that greatly speed up finding a name match. The cost of supporting the indexing is the need for additional storage space in the database.

For example, in order to find a specific application named “RF for NL zone 2”, which is derived from the generic application “RF”, the database is navigated as follows, using Figure 1 as a guide: first the “Applications” object must be found. A linear search of the five “EPICS DB” **components** returns a pointer to the “Applications” object. (The returned pointer must be cast correctly because the search method returns a pointer of type void*.) Using the “Applications” object as the parent now, an indexed search of its child nodes for the **component** “RF” is found. “RF” is one of the collection of generic applications. The generic application “RF” is then used as the parent and a linear search of its child nodes finds the **component** named “RF for NL zone 2”.

Application Developer Support

Part of the purpose of using this OODB is to assist application developers in managing algorithmic control data. One of the ways it can help is in the process of instantiating a control algorithm for a device. As mentioned previously, most of the devices in the accelerator are used repeatedly. For instance, the 42 RF cryomodules require more than 70,000 EPICS records to support data acquisition and control. The algorithm for controlling each of the 42 is identical: all differences are contained within the record field names and in the record names themselves. A developer can build a generic algorithm to control a single cryomodule and this generic can be instantiated, once for each physical cryomodule. The instantiation process must manage name convolution to give each instantiated record a unique name, at the same time setting field values appropriately to keep the algorithm internally consistent.

The application developers have the opportunity ofmanaging their algorithms in one of two ways: they can perform instantiation outside of the database or inside it. External instantiation requires the use of standard UNIX tools like *awk* and *sed* operating on ASCII files that define the control algorithms. The instantiated algorithms can then be installed into the OODB using software developed for this purpose.This method is attractive because the UNIX instantiation tools have already been developed for existing applications. One drawback is that these tools have been developed independently for each application, and there are no standards for the tools. A second and more serious problem with managing control algorithms this way is that data maintenance must be performed when the control algorithms are modified. This is required to insure that operational data is not lost when moving from one version to another.

Alternatively, application developers can perform instantiation within the database. With this technique the algorithm developed to control a device is installed as a generic application. The application developer then specifies the rules that will be used to instantiate that generic. The format of the rules must conform to the standards that are imposed by the instantiation tools. This imposes a burden on the developers to learn the syntax of the instantiation rules, but allows them to depend on OODB tools to do all instantiation. The developers are relieved of having to build instantiation tools of their own, or to adapt some other tools. After the generic algorithm and instantiation rules are specified, the algorithm can be instantiated. This process is automated by tools associated with the control algorithm database. When changes are made to the generic algorithm the new version can be installed, replacing the previous version. The new version can be instantiated immediately, using the already existing rules. Any changes to the database made by operations personnel are automatically migrated to the newly instantiated algorithms. This allow the data to persist across modifications to the generic algorithm and requires no intervention by either the developer or operations staff.

Instantiation inside the database is faster than instantiation externally. For example, generating a new instance of an RF algorithm takes roughly 5 minutes when done outside the database and slightly more than 2 minutes when done internally. A significant fraction of this time difference is due to disk reads and writes, because external instantiation is done with multiple passes through ASCII files, rather than with a single pass.

Operations Support

The second reason for using the OODB is to assist machine operations staff. The distributed nature of EPICS has led CEBAF to use a proliferation of data management tools. Some of the data is managed with the EPICS tool *burt*. Some is managed by software developers, who assure that operational data gets propagated into new versions of control algorithms as they are developed. The remainder of data management is done using a variety of tools, such as scripts which must be executed when IOCs are rebooted, or when a hardware crate is taken out of bypass mode. This variety of methods is burdensome to maintain.

The OODB provides operations staff with a tool that allows them to make persistent changes to field values on the operational machine. That is, a data change can be applied immediately to the running machine, but also to be the control algorithm resident in the OODB. After reboots of the IOC, the new data values will persist. The data values also persist through algorithmic changes, provided that instantiation is done within the OODB. This means the operator-set data values will be automatically migrated to new releases of the algorithm by application developers, assuring continuity of the data.

II. ACCELERATOR MODEL DATA

Background

CEBAF requires a number of high-level applications in order to operate the accelerator effectively. These applications include orbit lock, energy management, autosteering and others. All of the high-level applications share a need to obtain model data that describes the configuration of the machine. The model data is derived from the DIMAD files that were used in the optical design of the CEBAF accelerator and are the authoritative source for its physical description. A DIMAD file describing the machine is a series of descriptions of each of the accelerator elements, including its location, length, element type and data specific to that element type.

The DIMAD input files are awkward to use. They are in ASCII-format and must be parsed in order to be used by software. This makes initialization of high-level applications time consuming, taking approximately 40 CPU seconds on an HP-700 series machine. Further, many applications require data for only a subset of the machine. This means that either every application must parse the complete file, or else many different files are needed, where each file contains a portion of the accelerator data. In addition, users of high-level applications frequently like to examine “what if” scenarios with models of the machine. Using DIMAD files in these situations would require tools to automatically generate such files and a management system to support the tools. Clearly managing the model data with DIMAD files would be difficult, so it was decided to use an OODB for this purpose.

Database Objects

The machine model consist of a collection of beam elements, where each element has as attributes the physical character-

istics of the device associated with that model element. The elements are well-suited to objectification, as each type maps naturally onto a C++ object. Classes for all element types are derived from a single class, *beamElement*. The *beamElement* class has attributes associated with every device in the beamline: optical length, position in the beamline, aperture, misalignment and rotation. Objects which are subclassed have additional attributes. For example, the *BPM* class adds attributes that describe the beam position at the modeled BPM. The *Magnet* class adds attributes related to magnetic field strength and is used to derive all of the magnet classes, such as *Dipole* and *Quad*.

The model OODB supports other objects related to the management of *beamElement*s. One object is the *beamSect*, or beamline section, which is an ordered collection of pointers to *beamElement* objects. An example of a *beamSect* is **lin1**, which has in its collection of *beamElements* pointers to model information for all machine elements in the first pass of the CEBAF north LINAC. Another object type is *beamLine*, which is an ordered collection of pointers to *beamSect* objects. An example of a *beamLine* instance is the entire accelerator from injector to CEBAF experimental hall C. A second *beamLine* instance, for example the entire accelerator from injector to CEBAF experimental hall B, has in its collection pointers to the same set of *beamSect*s as the first, except that the *beamSect* pointer that corresponds to the beam line between the CEBAF extraction region and hall C is replaced with the *beamSect* for the beam line between the extraction region and hall B.

Database Usage

The model database is not used directly by high-level applications. Even with the high performance of the ODBMS, it takes over 20 seconds to access the entire 30 megabyte model database. If each application had to read the entire model database during initialization, program start-up time would be too long. Instead, a model server is used to provide applications with access to the model data. The server keeps an image of the OODB in a block of shared memory, requiring no disk reads to find and send model data to client applications.

The database is built by a modified version of DIMAD, which reads appropriately-formatted input files and generates persistent *beamElement* objects in the database. From there, a separate process reads the OODB and populates the shared memory with the model data. The model server is then started, using the shared memory data as its source, Figure 2.

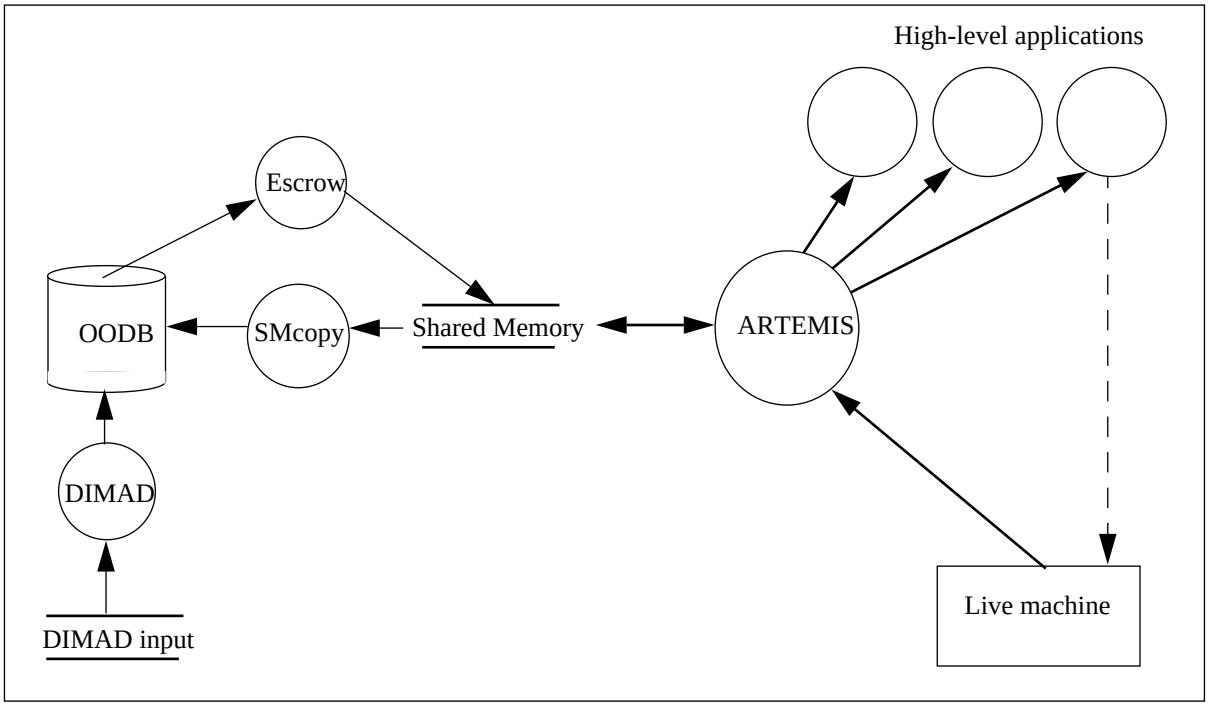

Figure 2: Flow of Model Data to and from the OODB

The OODB can be updated at any time to reflect the current state of the machine as it is modeled by ARTEMIS. This is needed because ARTEMIS can be connected to the running accelerator control system, acquiring real-time data from the machine. The machine state is saved using a tool that reads the shared memory and stores copies of the objects found there to the OODB. The saved machine state can then be used off-line, running high-level applications against that image of the machine.

Models of the machine will come in different flavors: a “design” version which reflects a theoretically perfect accelerator, a “golden” version that models an ideal version of the real machine (with alignment errors, etc.) and a “real-time” version that models the machine as it is currently configured. Each of the versions will be stored in separate database files, making it easier to save images of the machine and use them to perform “what if” kinds of analysis.

III. SUMMARY

The development of the control algorithm and model databases and their associated tools has been completed. Deployment of the software awaits the opportunity to do so, depending on the accelerator operating schedule. The performance of the ODBMS has met our expectations, but its performance can be sensitive to the organization of data within each database. This is because of the way that ObjectStore performs caching of data. A data organization that results in access patterns which frequently span more than a few megabytes will have a significantly worse performance that can be achieved with more localized data accesses.

The tight integration of ObjectStore and C++ makes building and using databases very easy for application developers. Accessing data is as simple as a pointer dereference operation. The database management details are hidden from the developer navigating the database. The flip side of this ease of use for developers is that fact that there is no generic user interface for all data. Instead, the process of navigating the database is dependent on its internal data structure. User interfaces must be custom-built for each database.

We are very happy with the ODBMS tool and are both developing and planning new databases. Some areas of interest at this time are a model status database to track the availability of the various ARTEMIS processes, an event-logging database to help account for beam uptime, and a database to serve as a repository for calibration data for the accelerator RF system.

IV. REFERENCES

[1] Web site at http://www.odi.com
[2] Hill, LANL, “A Server Level API for EPICS”
[3] B. Bowling, et. al. “The Use of ARTEMIS with High-Level Applications.” These proceedings.