DOE/ER/40150--321 CONF-950512--372

MANAGING CONTROL ALGORITHMS WITH AN OBJECT-ORIENTED DATABASE*

CEBAF-PR--95-011

M. Bickley, W. Watson, Continuous Electron Beam Accelerator Facility, Newport News, VA 23606 USA

The Continuous Electron Beam Accelerator Facility (CEBAF) uses the Experimental Physics and Industrial Control System (EPICS) for accelerator control [1]. In EPICS, the atomic element of a control algorithm is a record. Records are grouped together to form generic applications, for example to control a single magnet. The generic applications are then instantiated for each specific item of machine hardware. Instantiated applications are executed on one of the 30 data acquisition and control computers that are used in the control system. There are roughly 125,000 unique, instantiated records at CEBAF, each associated with a specific piece of hardware[2]. Management of these records in a database simplifies the task of application developers by allowing them to concentrate on algorithmic development instead of instantiation details. In addition, it decouples algorithmic development from the specification of operational parameters, allowing responsibility for those parameters to pass to machine operations staff. CEBAF needed an environment to provide support for development of EPICS database management tools. An object-oriented database (OODB) was chosen for two reasons: higher performance and the ability to smoothly manage objects of different types.

I. INTRODUCTION

A requirements analysis of the control system was performed, to determine CEBAF's data management needs. The analysis illuminated some limitations in the existing system and pointed the way to a solution. This section briefly introduces EPICS, and points out some of the problems that were presented to CEBAF.

EPICS provides a solid footing on which to base an accelerator control system. In EPICS, single-board computers using the real-time operating system VxWorks execute control algorithms coded as EPICS "databases". Each database consists of a number of records, which are executed according to rules of association specified prior to downloading of the database to its execution engine. The kinds of rules that are specified include execution order, prioritization of execution of the records and data communication between records.

Despite the solid execution environment that EPICS provides, its application development tools give little assistance in managing large projects like CEBAF. The developmental tools do not provide for data management outside of the application development framework, or for replication of EPICS databases. It is incumbent on each site that uses EPICS to provide its own mechanism for reproducing control algorithms. The typical solution at most EPICS sites is the use of UNIX-based text processing tools to do replication, and incorporation of instance-specific record definition data either during replication or afterwards. These solutions are not typically well integrated into the operational control system, and make it difficult to manage large numbers of records.

When CEBAF first started using EPICS, application developers also chose to use text-processing tools. Developers used a schematic editor tool to create generic control algorithm templates for a particular piece of hardware, such as an RF module. Specific instances, corresponding to pieces of real hardware, were generated from the generic algorithms by making four passes with different tools through successive ASCII files. Complete processing of some of the larger generic files, associated with CEBAF's RF system, took more than five minutes on an unloaded HP-700 series machine. This processing time adds significantly to the burden of developers during the debugging and testing phases of application development.

Another drawback to this style of producing EPICS databases is the lack of management tools. For example, there is no mechanism to prevent different records from having the same name. Such name conflicts are typically found at record execution time (if at all). Other limitations include the inability to perform wild-card queries on the names of records, to identify from a record name the front-end computer on which it resides, or to quickly query the attributes of a particular record.

II. OBJECT ORIENTED DATABASES

CEBAF had a clear need for a data management package to organize the 125,000 operational EPICS records. A requirements analysis of the management problem illustrated the need for support in two areas, machine operations and application development. The requirements analysis supported the choice of an object-oriented database (OODB) system for its speed and flexibility.

Operational Support

The requirements analysis indicated that the data management package had to provide tools for operations staff to manage accelerator operational data. In the past, developers have been forced to maintain operational data in their algorithms. This has principally been because of a lack of tools for non-developers to manage the data. Using a commercial database as a repository for EPICS record information enables the development of those data management tools. This in turn allows developers to release control of operational parameters to the CEBAF operations staff. Once that is done, developers can concern themselves principally with algorithmic function-

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

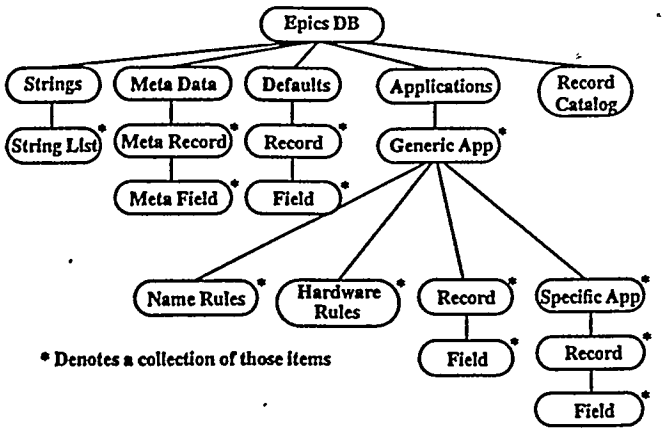

Fig 1. Schematic representation of the database structure

benefits in other ways besides the space savings. Replication of the generic applications is faster, since fewer fields must be created. Further, because each instantiated application is smaller, it can be stored more easily in the memory of workstations accessing the database. This greatly speeds up interaction with the database when queries are being generated.

Figure 1 shows a graphical representation of the organization of the EPICS database within the OODB. In the description below, italicized words refer to objects in the figure. The OODB representation, which includes all record information needed to manage all the EPICS records at CEBAF, is stored in an *Epics DB*, which is a collection of five different objects: *strings*, *meta data*, *defaults*, *applications* and a *record catalog*.

Strings: Some EPICS fields can only take on values which match one in a series of strings. The *strings* object, a collection of *string lists*, serves to support those fields. Each *string list* is in turn a collection of strings, where each string in the list is one of the possible values for a specific field. For example, every EPICS record has a field named "SCAN", which indicates the period with which the record should be executed. Some possible SCAN field values are "1 second", ".1 second" and "10 second". When a non-default value for a SCAN field is specified, the value stored with the field is not the string itself, but a pointer to the appropriate element of the SCAN string list. This format saves space, since only four bytes of storage are required for the pointer, rather than the 12 bytes that would be required for each use of the SCAN field. The *string lists* are also a convenient mechanism to verify the value of a field. If a user tries to specify a value for a field which uses a *string list*, and the value is not on the appropriate list, the user can be warned.

Meta data: The *meta data* object is a collection of *meta records*, with each *meta record* a collection of *meta fields*. Descriptive information about every field of every EPICS record is stored within the *meta data* object. This information includes, for example, the data type of the field value and a prompt string when querying a user for a value for this field. Gathering all of the *meta data* here makes it possible to keep other database objects as small as possible. Every *field* in the *Epics DB* includes a pointer to its associated *meta field*. If meta information for a *field* is needed by a database tool, a pointer dereference provides access to it.

Defaults: The *defaults* object is a collection of *records*, with each *record* a collection of *fields*. Each *field* holds a pointer to its meta data description, and a field value. This value is the default value for the field. When a specific instance of a *field* does not specify a field value, this is the value that is used by EPICS.

Applications: The *applications* object is a collection of *generic application* objects. Each *generic application* is made up a a collection of *naming rules*, a collection of *hardware rules*, a collection of generic *records* and a collection of *specific applications*, which are instantiations of the *generic application*.

The *naming rules* are a series of strings that specify substitutions to be performed on each generic *record* and generic *field* value as they are instantiated into specific *records* and *fields*. The *hardware rules* are a collection of record name/hardware address pairs which are used to associate each instantiated *record* with a particular crate, slot and channel. *Hardware rules* are only needed for those records which perform hardware input or output. The *record* object associated with a *generic application* is a collection of the generic EPICS records that make up the control algorithm for that piece of hardware. Each *record* is a collection of those *fields* which have taken on a non-defaulted value.

Finally, the *specific applications* are each made up of a collection of *records*, one *record* for every *record* in the parent *generic application*. The names of each *record* in the *specific application* have been converted according to the *name rules* associated with the *generic application*. Each *record* is a (usually small) collection of *fields*. The *fields* in each *record* are those which have a value different than the default value, and also different than the value in the generic *record* from which it was instantiated. Some *field* values refer to other *records*, so those values are processed according to the *name rules* for the *generic application*.

Record catalog: The *record catalog* object is a collection of names of all *records* in the database, and pointers to the *records*. This provides a means for performing wild-card searches on all *records* in the *Epics DB* without having to navigate through each *generic application* and *specific application*.

IV. REFERENCES

[1] W. McDowell et. al. "Status and Design of the Advanced Photon Source Control System", *Proceedings of the 1993 Particle Accelerator Conference*

[2] S. Schaffner et. al. "Device Control at CEBAF", these proceedings

[3] W. Watson et. al, "The CEBAF Accelerator Control System: Migrating from a TACL to an EPICS Based System", these proceedings

DISCLAIMER