# A Portable Accelerator Control Toolkit

JLAB-ACC-97-09

William A. Watson III, Thomas Jefferson National Accelerator Facility

*Abstract*

In recent years, the expense of creating good control software has led to a number of collaborative efforts among laboratories to share this cost. The EPICS collaboration is a particularly successful example of this trend. More recently another collaborative effort has addressed the need for sophisticated high level software, including model driven accelerator controls. This work builds upon the CDEV (Common DEVice) software framework, which provides a generic abstraction of a control system, and maps that abstraction onto a number of site-specific control systems including EPICS, the SLAC control system, CERN/PS and others. In principle, it is now possible to create portable accelerator control applications which have no knowledge of the underlying and site-specific control system. Applications based on CDEV now provide a growing suite of tools for accelerator operations, including general purpose displays, an on-line accelerator model, beamline steering, machine status displays incorporating both hardware and model information (such as beam positions overlaid with beta functions) and more. A survey of CDEV compatible portable applications will be presented, as well as plans for future development.

## 1 INTRODUCTION

In almost any book or journal on software development one will find reference to the explosion in the quantity of software development, and the cost and difficulty in developing necessary software in a timely fashion. A typical rule of thumb for accelerators is that the control system costs 10% of the total project, with half of that going to software. In addition, as much as 5% to 10% of operating manpower may go towards ongoing software improvements.

In a decade of declining research budgets, this expense has driven an increasing interest in software sharing within many areas of the research community, including the accelerator controls community.

### *1.1 EPICS*

One example of a successful collaboration to develop accelerator control software is EPICS (Experimental Physics and Industrial Control System). This software, whose history is described in another paper at this conference [1], is now in use at several accelerator sites including the Advanced Photon Source at Argonne, Thomas Jefferson National Accelerator Facility (Jefferson Lab), the B factory upgrades at SLAC and KEK, and several smaller machines.

EPICS provides a framework for developing low level device controls, including hardware interfacing and low level control algorithm development. Many device drivers are shared within the collaboration, as are modular software components for creating control processes. Programs within an EPICS system are tied together by a network infrastructure (the channel access library) based upon reading, writing, and monitoring changes in named variables. A named variable can refer to a hardware I/O point, or a variable within an algorithm

The toolkit contains a number of utility programs which plug into this software bus, including synoptic displays with interactive editors, a save/restore utility, archiving (data logging) and data browsing programs, and an alarm interface. Many commercial and freeware packages have also been interfaced to this bus via a callable library (e.g. a spreadsheet and the tcl/tk toolkit).

Including astronomy sites and large physics detectors, the EPICS collaboration includes over a hundred users and application developers, and represents a notable software sharing effort. For the most part, this sharing is limited to those who use EPICS as the core of their control system.

### *1.2 SOSH*

SOSH (for Software Sharing) is a name given to a series of workshops on the general topic of software sharing for accelerators and large physics detectors. The original meetings were held in conjunction with the International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS).

The current thrust of these workshops is to (1) develop a framework within which shareable applications can be built, (2) develop shareable utility applications (e.g., display or manipulate named control system quantities), and (3) develop accelerator or detector specific control applications. The framework includes a common (abstract) interface to the local control system, with common standards for names of classes of devices and their attributes (or a way of aliasing these to a common set).

At the "Workshop on Software Sharing" following ICALEPCS '93 in Berlin, 19 invited participants agreed in a joint statement that "there is no fundamental reason (from operation and machine points of view) why ... the primary functions in the draft list could not be implemented by common generic (configurable) software and/or using appropriate common software tool kits". [2]

This list of functions included 13 topics related to the application environment including user interface development, on-line help, a sequencer, data logging, archiving, and system configuration. This is the area well covered in a portable way (within EPICS) by the EPICS toolkit.

What is more remarkable is that the participants stated that accelerator applications were equally shareable: magnet cycling (and super cycles), orbit measurement and correction, tune measurement and correction, chromaticity measurement and correction, RF gymnastics, machine

simulations, injection, matching, and extraction. To date, these sorts of applications have seen only limited portability, yet represent an even larger software development effort than all of EPICS at major accelerator labs like CERN, FNAL, and SLAC.

## 2 DISTRIBUTED SOFTWARE TECHNOLOGY

### *2.1 A Software Bus*

If these applications are to be shareable among a large number of laboratories running different (mostly custom) control systems, there needs to be a well defined common interface through which they can connect. This interface is often referred to as a "software bus". Just as hardware modules pass data over a backplane bus, so too software modules (programs) pass information over the software bus.

In May of 1995, CERN hosted a workshop titled "A Softwarebus Common to Accelerators and Large Experimental Physics Control Systems". Twenty-five participants agreed that (1) applications "above the bus" (host side vs. hardware side) held the most promise and benefit for sharing, and (2) CDEV, a C++ framework developed at Jefferson Lab, [3] should be investigated as the framework through which these applications could access the control system. Two additional workshops in the previous 2 years have continued to focus upon CDEV as an enabling technology for portable accelerator control applications.

There are two ways in which a software bus can be defined, and both are used in practice. In one, the *network protocol* is defined, including how resources are located (discovered) on the network, what types of messages between programs are supported, and how these messages are formatted on the network. In the second way, an *application programming interface* (API) is defined, which specifies a set of routines to be called for communicating with other programs. The protocol on the network is not defined, and in fact multiple protocols may be used. The second technique is equivalent to defining a *virtual accelerator*, as presented by Kanaya at ICALEPCS '93. [4]

There are many software buses of each type in existence today, and it is difficult to choose one to be a standard; in fact the choice is somewhat a matter of preference. The choice of CDEV as a potential standard interface was driven by these requirements of the bus:

- ability to connect to legacy control systems
- high performance, with fully asynchronous behavior
- support for a high level view of the control system, dealing with named accelerator devices (magnet, bpm) each with multiple attributes (field, x-position, beam current) instead of a view consisting of hardware addresses or low level control points
- support for object oriented programming, and in particular the C++ language
- support for rich messaging (complex queries with complex replies)

### *2.2 CDEV*

CDEV (Common DEVice) provides an interface (API) to a virtual control system with a simple flavor — the system consists of a set of named devices to which messages may be sent. The client program has no knowledge of the device's software or hardware implementation (location, control system type), and only knows (or discovers at run time) the list of messages to which the device responds.

CDEV is implemented as a C++ framework that provides a standard interface between an application and one or more underlying control packages or systems. It serves as an adaptor, or middleware, between a portable application and a local control system. In addition, it provides a number of features not provided by many control systems.

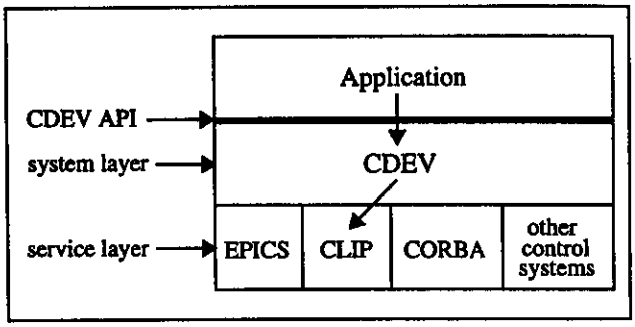


Figure 1: CDEV multi-service architecture.

CDEV does not specify which networking protocol is used between client and server, and can in fact support multiple protocols simultaneously (Figure 1). When used with the EPICS channel access protocol, CDEV can locate hundreds of device/attribute pairs per second, and receive thousands of value changes per second. The overhead of using CDEV instead of the native EPICS API is negligible and well offset by the additional functionality provided, even if portability is not a desired goal.

When used with another custom TCP/IP based protocol (CLIP) developed at Jefferson Lab, CDEV can deal with complex queries to (for example) a model server returning arrays of two-dimensional transfer matrices between specified devices. The client application remains unaware of which protocol is involved.

Additional interfaces to other control systems may be added to CDEV with a modest amount of programming, ranging from a couple of weeks of learning and coding for a simple system, to a few months for a complex system. SLAC has interfaced CDEV to its control system on VAX/VMS computers, and CERN/PS has done likewise for an IBM/AIX system. This is an extremely small amount of effort to support portable software.

### *2.3 Alternatives to CDEV*

One possible alternative to CDEV which was considered (and continues to be evaluated) is CORBA, the Common Object Request Broker Architecture, which defines standards for object-oriented distributed-programming communication mechanisms. [5] Implementations of CORBA

are available from multiple vendors on all major platforms, and the latest version of the standard addresses interoperability among vendors.

While CORBA is well accepted in the marketplace, there are several problems with making it the software bus for control systems, and performance is one. CORBA is about 10 times slower at locating resources on a network than can be achieved with custom protocols — primarily because the location services deal with only a single resource at a time. In contrast, EPICS and CLIP buffer requests for name resolution, achieving much higher throughput for large, complex applications such as interactive displays containing a thousand or more variables.

Another CORBA difficulty is the complexity of the API for dynamic invocation (talking to remote objects whose interface is not compiled into the current program). This dynamic binding capability (discussed further in the next section) is a key feature of many utilities.

Nevertheless, CORBA continues to be of interest. One avenue often discussed is to use CDEV over CORBA, and to use CORBA only to locate servers and to transport messages. In this case one would use a custom (accelerator device) resource locator and CDEV as the API and higher level framework. CDEV could easily support this simultaneously with the existing direct support of other protocols.

## 3 UTILITY APPLICATIONS

### *3.1 Name Based I/O*

There are a large number of useful controls applications which deal only with named values, and are not accelerator specific. These include operator displays (graphical — a meter showing magnet current, or text based — a list of all magnet setpoints and currents), data archiving (current in the magnet for the last year), or save and restore (of the magnet setpoint). These control system values are referenced by a single name (e.g. magnet7-setpoint) or through a pair of names (magnet7, setpoint) corresponding to device name and attribute name.

Because of the proven usefulness of the EPICS utility programs (which are name based), one development activity has been to port those tools to CDEV, allowing them to be used with non-EPICS servers and protocols.

### *3.2 Converting EPICS tools*

Two EPICS applications [6] have already been converted from calling the EPICS channel access library to making CDEV calls.

*stripTool* Strip chart graphical application, with 8 colored pens. Interactively choose variables, including wildcards. Save / restore of display definitions.

*alh* Alarm handler; monitors the alarm (error) status of the referenced values and summarizes the errors in a tree hierarchy. Indicates alarm through color, blink, and beep.

In addition, the following EPICS tool is in the process of conversion; others will be converted as time allows.

*dm* Display manager; one of the two synoptic display programs in EPICS, with the ability to display values as text, color (of a graphic), or through widgets such as meters and push buttons. Menus and push buttons support executing scripts or bringing up additional displays.

### *3.3 New CDEV tools*

Several new utilities have been developed or are currently being developed within the CDEV framework:

*xact* X-windows Automated Correlation Toolkit. Modeled after the SLAC correlation package, this utility can step 1 or 2 variables, and measure hundreds of other variables at each step. As part of each step or measurement, additional actions may be performed including time delay, wait for a value to settle, or invoke a script. Plans include automated min/max optimization of one parameter (done routinely at SLAC with their software).

*zplot* Another tool modeled after a SLAC utility, this program displays attributes of devices (such as *bdl*, the integral field in a magnet) as a function of position along the beam (z) in the machine. While this appears to be specific to accelerators, the attribute representing position could easily be replaced by any other collating parameter.

*xarr* Archive data browser. Originally developed to directly read EPICS archive data files, this program is being converted to a CDEV based client/server architecture. *StripTool* will also be modified to initialize immediately with archived data from the server, and to allow scrolling backwards in time. Additional features in the new archive system are planned. [7]

*cmlog* A distributed error logging system. Includes a logging daemon for each host (Unix and VxWorks), a database server, client logger and browser libraries, a Motif browser, a tcl browser, and (soon) a Java browser. Logging client library supports filtering (suppression of repeating errors). Browser supports interactive suppression of uninteresting errors.

## 4 CDEV COMPONENTS

CDEV is (1) a standard API for communicating with devices, (2) a C++ framework implementing this API, (3) a Java package implementing a (subset of) this API, and (4) a set of applications and libraries useful in building distributed systems. This section will briefly review the highlights of each, emphasizing recent developments.

### *4.1 C++ Library*

The mainstay of CDEV is a C++ class library for developing both applications and adaptors to additional control systems. The library includes:

- *directory services:* look up devices by name or by type, including wildcard matching; discover at run time supported attributes and messages; get type for given device
- *asynchronous messaging*: high throughput, buffered I/O; callback mechanism, time-outs
- string and composite self describing binary data messages, with support for multiple architectures (byte swapping)
- I/O operation *grouping* and *synchronization*
- *collections,* for operations on vectors of devices, with support for passing the device array intact to the underlying control system for higher performance on some systems
- *virtual I/O:* use of multiple control systems from a single calling interface
- support for EPICS, CLIP (plus others at their sites)
- base class for integrating new control systems
- extensive documentation

### *4.2 Java Package*

The Java package is written in 100% Java, allowing applets to be written to run inside of commercial web browsers. [8] It supports the same calls as in C++ for sending messages to devices, with network support for the CLIP protocol also in 100% Java. The package currently does not include support for groups or collections.

In addition to the Java-cdev package, there is also (in beta form) a graphics library for producing animated displays along the lines of those produced by dm (above).

### *4.3 tcl/tk*

It has been the experience at Jefferson Lab and elsewhere that the *tcl* scripting language and its *tk* graphics toolkit provide an extremely productive environment for rapid prototyping of control applications. [9] Tcl has been interfaced to CDEV, allowing scripts to access the entire control system and accelerator model at Jefferson Lab.

### *4.4 Network Components*

The latest extensions to CDEV include a set of network components useful in building up a large distributed system. These components include:

*NameServer* Supports the mapping from a named resource to server address and port. A CDEV device may be implemented as a single resource, or as a set of resources on different servers. Communication with the name server is asynchronous and buffered, locating resources 10 times faster than CORBA.

*Gateway* Allows multiple applications to connect to the control system through a single point, producing only a single connection to any real-time system. Performs protocol conversion from the external protocol (CLIP) to the site-specific protocol. Currently used to connect Java applets to the control system. (See Figure 2.)

*Server Shell* A skeleton server program which can be used to build a new CLIP server by writing a single routine to handle one message. All connection management and message queueing and routing is handled by the shell. Used to implement the NameServer, Gateway, and the model server Artemis (described in the next section).

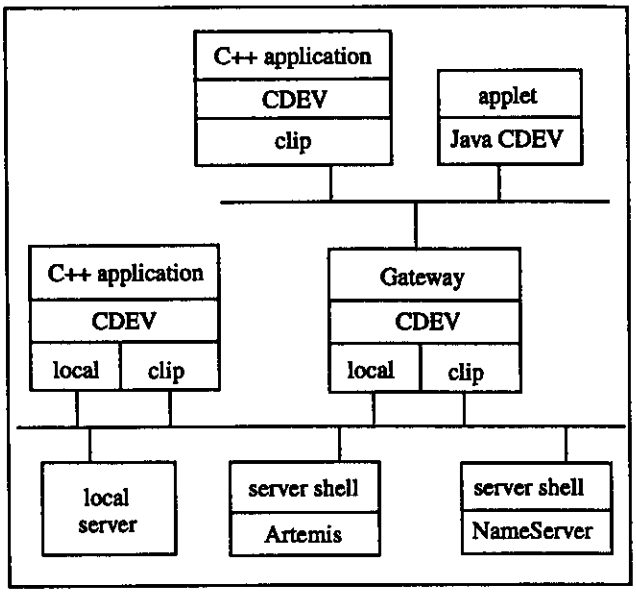


Figure 2: CDEV network components, showing logical network connections for two protocols, with gateway connected applets and applications.

## 5 ACCELERATOR APPLICATIONS

A certain amount of success has been achieved in the past in sharing beam optics modeling codes, such as MAD, DIMAD, PARMELA, and also analysis codes, such as RESOLVE. These applications are off-line applications with no connection to a control system, yet do represent a notable software sharing success.

Sharing of on-line applications is somewhat more difficult, and has met with only limited success. Much of the lack of success can be attributed to the lack of a common interface to the control system.

### *5.1 Standards or Conventions Needed*

The model design codes mentioned above have been successful in moving from site to site because they provide significant capabilities, while enforcing few constraints upon the users. Each program has a simple naming convention for devices, and for classes of devices, and for attributes of devices. For example, DIMAD defines a quadrupole magnet as something of type "quadrupole" having characteristics "L" (length), "K1" (strength, in $1/m^2$), and "aperture" (radius in meters). Instance names are restricted to eight significant characters, and everything is case insensitive.

These are exactly the types of conventions which need to be standardized in order to allow portable on-line applications -- conventions on names of classes of devices, and conventions upon what capabilities (such as read and write attributes) these devices support.

### *5.2 CLASSIC*

Among recent attempts to standardize the definitions of accelerator objects is the CLASSIC project. [10] CLASSIC is an acronym for Class Library for Accelerator System SImulation and Control. Its goal is to provide:

- a C++ class library for accelerator design, simulation and operation
- a mechanism for C++ code sharing and standardization in the accelerator community, and
- a platform to exchange new ideas in code development.

The collaboration includes SLAC, CERN, FNAL, DESY, Jefferson Lab, and the University of Maryland.

CLASSIC includes a standard input file format with mnemonic type codes for all accelerator elements, memory structures to represent these beamline components and composite beam lines, representations of lattice transfer maps, representations of misalignments, interfaces to algorithms, and an interface to the on-line control system (the plan is to use CDEV). This is still a work-in-progress, with the initial software being tested within the framework of a new version of MAD.

### *5.3 Unified Accelerator Libraries*

UAL [11] is another effort to develop an environment for portable accelerator control applications. One major thrust of this effort is to standardize descriptions of accelerator structures. Unlike the CDEV and CLASSIC projects, UAL does not standardize upon C++ as the programming language, but instead uses the scripting language PERL as the glue to bind together a set of programs in potentially multiple languages into a cohesive system.

At this point, the UAL project anticipates using CORBA as the software bus through which applications will gain access to the control system.

### *5.4 CDEV Compliant Accelerator Software*

In addition to the general purpose utility applications listed in the previous section, there are a small number of accelerator optics applications already finished:

*Artemis* Artemis is an accelerator beam optics server for simulation and control. [12] It provides first- and second-order transport matrices, beam envelop propagation, and particle ray tracing. It currently uses DIMAD as a back-end, but is adaptable to other modeling engines. It uses CDEV to obtain current lattice settings and to service clients.

*Atlast* Atlast (AuTomated Lock And Steering Toolkit) is a modular program for beam based energy and orbit corrections. It uses CDEV to monitor beam position monitors, obtain model information, and drive actuators. Multiple algorithms are allowed, with support for SVD and Prosac. [13]

## 6 SUMMARY

Progress has been made in forming a new multi-lab collaborative effort in high level accelerator applications development. The CDEV framework has been used to support a diverse set of on-line tools, including modified EPICS applications, new utilities, and a small number of beam based applications. Portability of applications between EPICS and non-EPICS control systems has been demonstrated.

New developments at Jefferson Lab, SLAC, and other labs will continue to expand the set of CDEV compliant applications, and the work of affiliated groups like the CLASSIC collaboration will further increase the amount of software runnable at sites supporting a CDEV adaptor.

## 7 ACKNOWLEDGMENTS

Work supported by the U.S. Department of Energy, contract DE-AC05-84ER40150.

## 8 REFERENCES

[1] 'Experience with EPICS in a wide variety of applications', M. Kraimer, ANL; M. Clausen, DESY; W. Lupton, KECK; and C. Watson, Jefferson Lab, these proceedings, PAC '97.

[2] 'Panel Session on Software Sharing', 'About the Saturday Workshop', B. Kuiper, ICALEPC 1993 Proceedings, Nucl. Instr. and Meth. in Phys. Res. A 352 (1994) 513–515.

[3] 'An Object-Oriented Class Library for Developing Device Control Applications', J. Chen, W. Akers, G. Heyes, D. Wu, and C. Watson, ICALEPCS 1995 Proceedings. See also http://www.jlab.org/cdev/.

[4] 'Virtual accelerator and fundamental guidelines towards sharable software for accelerator control systems', N. Kanaya, ICALEPCS 1993 Proceedings, 497–500.

[5] Re CORBA, see http://www.omg.org/omg00/wicorba.htm.

[6] Private communication. *StripTool* converted by C. Larrieu, Jefferson Lab; *alh* by Janet Anderson, Argonne National Lab.

[7] 'Design of a new EPICS archive system' presented at the Vancouver spring 97 EPICS collaboration meeting by Matt Bickley.

[8] 'A Java Package for Building Client Applets to Access TJNAF Accelerator Data Across the Internet', C. Quach, master's thesis, Christopher Newport University, 1997.

[9] 'Rapid Application Development Using the Tcl/Tk Language', J. van Zeijts, PAC 1995 Proceedings, Vol. 4 p. 2241.

[10] 'The Classic Project', F. C. Iselin, Computational Accelerator Physics Proceedings, 1996, 325–330.

[11] 'Unified Accelerator Libraries', N. Malitsky, R. Talman, Computational Accelerator Physics Proceedings, 1996, 337–342.

[12] 'The Use of Artemis with High Level Applications', B. Bowling, H. Shoaee, S. Witherspoon, ICALEPCS 1995.

[13] 'Prosac Algorithm', Y. Chao, CAP 96, 319.