CAFE, A MODERN C++ INTERFACE TO THE EPICS CHANNEL ACCESS LIBRARY

J. Chrin, M.C. Sloan, Paul Scherrer Institut, 5232 Villigen PSI, Switzerland

Abstract

CAFE (Channel Access interFacE) is a C++ library that provides a modern, multifaceted interface to the EPICS-based control system. CAFE makes extensive use of templates and containers with multiple STL-compatible access methods to enhance efficiency, flexibility and performance. Stability and robustness are accomplished by ensuring that connectivity to EPICS channels remains in a well defined state in every eventuality, and results of all synchronous and asynchronous operations are captured and reported with integrity. CAFE presents the user with a number of options for writing and retrieving data to and from the control system. In addition to basic read and write operations, a further abstraction layer provides transparency to more intricate functionalities involving logical sets of data; such 'group' objects are easily instantiated through an XML-based configuration mechanism. CAFE's suitability for use in a broad spectrum of applications is demonstrated. These range from high performance Qt GUI (Graphical User Interface) control widgets, to event processing agents that propagate data through the Object Managements Group's Data Distribution Service (OMG-DDS), to script-like frameworks such as MATLAB. The methodology for the modular use of CAFE serves to improve maintainability by enforcing a logical boundary between the channel access components and the programming extensions of the application framework at hand.

CAFE À LA MAISON

The Experimental Physics and Industrial Controls System (EPICS) is an established framework for the development of distributed control systems in the field of particle accelerators and other large-scale experimental endeavours [1]. A dedicated communications protocol, Channel Access (CA), allows for the high throughput of small data packets between the low-level hardware and external programs. Its native client library [2], written in the C programming language, provides remote access to controls data encapsulated in Process Variables (PVs) residing in EPICS Input Output Controllers (IOCs). It is thus the entry point for a number of CA interfaces to various C-based high-level programming languages, including declarative and 4^{th} generation languages such as Python and MATLAB respectively. While each of the resulting CA classes aims to achieve a similar interface and abstract layer, their context is typically confined to the system in which they execute. Another approach would be to enforce a logical boundary between the channel access components and the specifics of a given domain's C/C++ extension framework by providing a single C++ CA library for use across a number of C/C++ based programming languages.

There are several advantages to this library-based approach:

- The inherent simplicity and convenience of maintaining the channel access interface code and the flexibility it offers for library designers.
- New CA functionalities from future EPICS releases [3] need only be integrated in a single repository.
- Bindings to scripting and domain-specific languages are simplified.
- In-house (à la maison) CA expertise ensures a quick response to user needs and problem solving.

CAFE (Channel Access interFacE) is a C++ library that provides a modern, multifaceted interface to EPICS. It is intended to be expressive enough to provide the necessary abstractions in a convenient way and to be flexible enough to act as a host CA interface for other C/C++ based languages.

CAFE provides functionality for synchronous and asynchronous interactions for both individual and groups of channels. Data propagation through channel access is performed exclusively with native data types, although data presented to/by the CAFE interface may be in any meaningful format. This is also true of enumerated types. CAFE makes extensive use of function templates that allow identical code segments to perform a given task for any type of data. These have been used to implement client interfaces, underlying channel access classes, and data type conversions.

Callback functions have been implemented for all operation involving channel access connection handlers, event handlers and access rights handlers. Their invocation triggers their data to be written into a multi-index container provided by the Boost libraries [4]. The container takes ownership of the data object elements, which store all the current details of the associated PV, and provides multiple, distinct interfaces that allow for fast retrieval and modification of the element's data. This is exemplified by an inquiry to the current connection state of a channel which is a factor of $\sim 10^2$ faster compared to the native CA client call. While every gain in efficiency is welcome, it is, however,

the channel access data transfer time (∼ ms) that remains the dominating factor. There is, therefore, some leeway for code optimization within the CAFE Application Programming Interface (API). The most critical issue is to avoid dynamic memory allocation at each channel access transaction as this noticeably impacts performance. The optimal approach is thus for the created PV handle (object reference) to take responsibility for allocating the necessary memory for storing the handle's data set within the element of the container. Memory space is re-allocated whenever the connection event handler is invoked or upon demand.

CAFE aims to hide the low-level technical details of channel access as much as possible. It has thus been preconfigured with a reasonable set of default parameters that allow basic operations to be immediately undertaken without details of CA or other third-party APIs leaking into the application domain. Nevertheless, as users become accustomed to the many CA possibilities, they will wish to optimize connectivity in a manner that best suits the needs of the application, e.g. by influencing certain CA properties such as timeouts, and whether to instigate operations as blocking or non-blocking. Each handle possesses its own set of CA properties that can be easily configured 'on the fly' through designated container access methods. Thus, although the internal mechanics of certain channel access operations may change, the interface gratifyingly remains the same.

CAFE is also equipped with an abstract layer that addresses related data sets as single logical software units. Such data may be retrieved/set with a single method invocation. An XML configuration mechanism is used for the initialization of such CAFE 'group' objects. CAFE thus has its own XML schema for defining collections of related nodes and for constructing groups from collections and individual channels. While such an XML file consisting of collections may be manually instrumented, the practice is to extract nodes of the same type from a master XML accelerator file that defines the topology of the accelerator in Universal Machine Format [5]. A dedicated parser traverses the accelerator XML to identify collection members and to transform their data into the CAFE XML schema. The pre-defined collections are loaded from the generated 'Collections XML file' on initializing CAFE and may be addressed by interfaces through their identifier.

In constructing the channel access interface, effort had been made to follow sound practices presented in [6]. Any remaining errors, however, remain the authors responsibility. The work presented here has been undertaken in a Linux environment and in the context of the Swiss Light Source (SLS) and the 250 MeV SwissFEL Injector Test Facility at the Paul Scherrer Institute (PSI).

SAVEURS DE CAFE

CAFE's suitability for use as a channel access host within different applications is demonstrated with examples ranging from high performance Qt control widgets [7], to Event Processing Agents (EPAs) that propagate data through Object Management Groups's (OMG's) Data Distribution Service (DDS) [8], to script-like frameworks such as MATLAB [9]. Figure 1 illustrates how the CAFE API conceptually extends the different application environments.

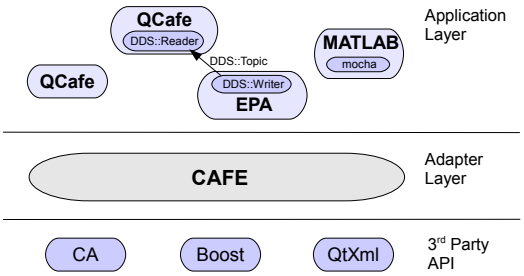

Figure 1: Application objects connecting to CAFE's adapter to channel access.

QCafe

The use of Qt for the development of CAFE-aware GUI (Graphical User Interface) widgets is a natural choice given the commonality in language. Qt offers components for a broad range of customizable widgets that can be readily applied to control system applications, and its graphics canvas and style engine allows modern, sophisticated user interfaces to be created. Qt's low latency coupled with C++'s runtime and memory efficiency further provides for a superior solution for building high performance GUI applications. These aspects could be well exploited for the computation and display of high-volume and high-rate data.

The QCafe package combines the speed, power and flexibility of C++/Qt with CAFE's extensive and robust channel access interface. The QCafe class diagram is shown in Fig. 2. The QCafe widgets are completely decoupled from CAFE as interaction with EPICS PVs is handled solely by QCafe's 'CAFE Interface Class'. EPICS data are then communicated between the interface class and the widgets using Qt's signals and slots mechanism. Some of the control/monitor widgets developed are shown in Fig. 3. The QCafeWheel widget is a customized widget built from other Qt GUI components and exhibits the same behaviour as its counterpart in MEDM (Motif Editor and Display Manager), one of EPICS established GUI builders [10]. Some of the widgets, namely QCafeStripChart, QCafeSlider and QCafeMeter, were developed with Qwt [11], a graphical extension to Qt.

Work is in progress to integrate QCafe widgets into Qt Designer, a tool for designing and building GUIs from Qt components. This will allow EPICS control GUIs to be created in an entirely code-free framework.

EPICS-Qt interfaces have also been developed independently at other facilities [12, 13].

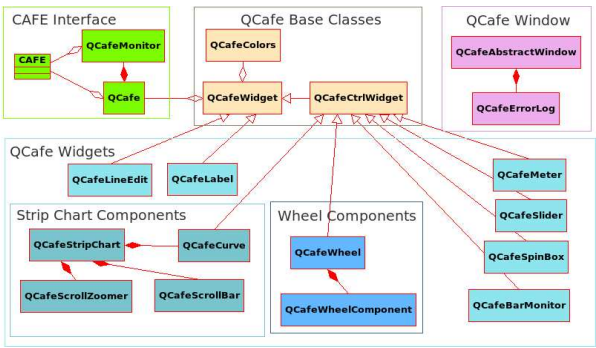

Figure 2: QCafe class diagram.

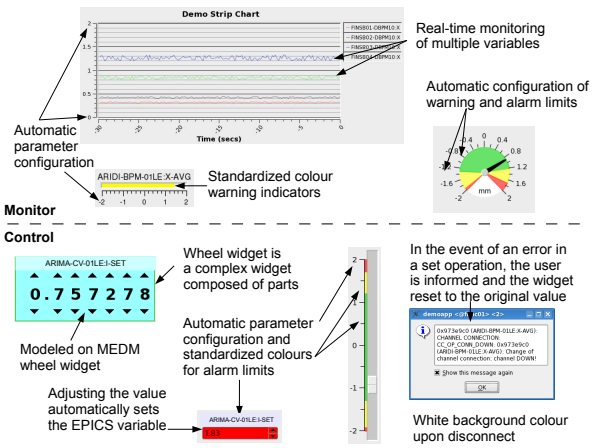

Figure 3: QCafe monitor and control widgets developed with Qt (v. 4.6.3) and Qwt (v. 5.2.1).

CAFE-DDS

High performance messaging middleware is an important component of distributed applications, and one that has been, for many years, an integral part of the high-level applications software infrastructure at the SLS [14, 15]. DDS is a recent OMG standard that supports high performance publish-subscribe messaging with configurable Quality-of-Service (QoS) guarantees. It has been put into good effect at the SwissFEL Injector Test Facility to disseminate summarized results of data acquired from the low-level hardware system. The use of CAFE combined with DDS (OpenSplice Community Edition, v. 5.1 [16]) is exemplified in an EPA that gathers data from the Digital Beam Position Monitors (DBPMs) [17].

The task of the DBPM-EPA is to aggregate, verify and analyze the DBPM data and to publish a summarized data set through a specially designated DDS Topic. The part of CAFE is to provide a simple, intuitive interface to initiate the acquisition of an entire complement of DBPM PVs. This is fulfilled by initiating monitors on the PVs through a single method invocation that includes, as input arguments, a group 'identifier' and a callback function that directs updated values into a designated data container. The DBPM PVs that comprise the group are defined in an XML configuration file by reference to a pre-loaded collection identifier (as described earlier) and the specified device attributes (Listing 1).

On parsing the DBPM-EPA XML configuration file, collections are expanded into their constituent elements and combined with the stated attributes to instantiate a CAFE 'group' object that holds the sequence of required DBPM PVs. The CAFE 'group' object created from Listing 1 is referenced in CAFE methods by its identifier, 'gDBPM'. Static DBPM data (names, positions, etc.) associated with each of the 'gDBPM' group members are extracted from the master SwissFEL XML file, and published to a separate DDS Topic for propagation to GUI clients.

The EPA is thus entirely configurable through XML, and the DDS Topics, which define and identify the data being propagated, provide sufficient information to allow for the dynamic configuration of a GUI client. Changes to the topology of the accelerator may thus be propagated to applications by simple modification to a configuration file, without need to recompile, as demonstrated in [5].

Listing 1: XML Configuration for DBPM-EPA at SwissFEL Injector Test Facility

```
<group id='gDBPM'>
   <collection> <id> cDBPM </id>
        <attribute> X </attribute>
        <attribute> Y </attribute>
        <attribute> Q </attribute>
        <attribute> POS–VALID </attribute>
        <attribute> Q–VALID </attribute>
   </collection>
</group>
```

CAFE Mocha

The use of MATLAB in the accelerator community has surged in recent years and is the principal choice of physics application developers at the SwissFEL Injector Test Facility [18]. (The Qt application presented in the previous subsection is one of a few exceptions.) Its increased usage, however, warrants a more extensive interface to the EPICS control system to the presently used mca (MATLAB Channel Access) package [19, 20].

CAFE mocha (MATLAB Objects for CHannel Access) is a MATLAB Executable file (MEX-file) that interfaces CAFE to MATLAB allowing CAFE routines to be called from within MATLAB in the same manner as MATLAB built-in functions. In many respects, the groundwork for MATLAB channel access connectivity had already been laid by mca. CAFE mocha, however, unlike mca, implicitly shelters underlying CA routines from the MEX API. Since the former are designated to the CAFE library, the resulting MEX-file is greatly simplified. Indeed, establishing communication between CAFE methods and the MATLAB workspace, where variables are passed into a function through input and output arguments, is largely re-

duced to mapping MATLAB data types to their equivalent CAFE/C++ data types.

The mocha interface improves on mca by providing support for all MATLAB data types, a richer set of access methods and a further physics oriented abstraction layer. Its use at the SwissFEL Injector Test Facility has been spurred on by the fact that certain tangible benefits became immediately available. Access to MATLAB primitive data types significantly improved performance in the analysis of camera data used in beam profile measurements. The advent of MATLAB's object-oriented programming capabilities has also allowed us to produce mocha-aware classes for specific operations, such as the acquisition of machine snapshots and orbit data (though some computations have been moved to within the MEX-file to enhance performance). Typically these objects, on initialization, extract static data, such as node positions, from CAFE's interface to an XML-based accelerator database (i.e. the master XML accelerator file). Object methods are also invoked to retrieve controls data with one synchronous group call, or to start monitors which cache updated values and, following the convention in mca, may optionally execute a MATLAB script to update a widget value. Widget updates may alternatively be handled through MATLAB's Event and Listener model, which allows a change in an object property to be monitored and a corresponding action to be triggered.

CAFE mocha's compilation on a 64-bit machine has also enabled a first comparison of the injector data with a computationally intensive three-dimensional space-charge simulation model.

A number of mocha mca-like MATLAB scripts [20], which is the usual way in which mca users access the mca MEX-file, have been provided to facilitate transition from mca to mocha for application developers.

HARD ROCK CAFE

Acceptance tests from several external APIs have been performed and give us confidence that CAFE is in a credible, 'Hard as Rock', state and can now be deployed. A short-term objective is to begin the task of replacing CA interfaces that are frozen to deprecated CA functions (as in [21]) with the CAFE API. Developments with QCafe continue with the aim to provide superior, high performance GUI applications, while CAFE-DDS offers a state-of-the-art publish-subscribe mechanism for applications geared towards a reactive form of programming. CAFE's mocha API has demonstrated immediate tangible benefits and work towards its consolidation draws near. The approach of having an extensive and flexible, C++ channel access library to serve as a host API for C/C++ based language extensions will lead to a thoroughly tested code base for other declarative and domain-specific environments, ultimately simplifying maintenance of code. If proved effective, then "Hard Rock Cafe's" mission, "to love all and serve all", is one that may be equally adopted by our very own brand of "Hard Rock CAFE" [22]!

REFERENCES

[1] EPICS, http://www.aps.anl.gov/epics/.

[2] J.O. Hill, R. Lange, "EPICS R3.14 Channel Access Reference Manual", http://www.aps.anl.gov/epics/docs/ca.php.

[3] A. Johnson, R. Lange, "Evolutionary Plans for EPICS Version 3", ICALEPCS 2009, Kobe, Japan, pp. 364–366.

[4] Boost Multi-index Containers Library, http://www.boost.org/libs/multi_index.

[5] J. Chrin et al., "XML Constructs for Developing Dynamic Applications or Towards a Universal Representation of Particle Accelerators in XML", IPAC 2011, San Sebastián, Spain, pp. 2295–2297.

[6] D. Zimoch, "Channel Access Client Programming", EPICS Collaboration Meeting, 27-29 July 2009, NFRI, Daejeon, Korea, http://www.aps.anl.gov/epics/meetings/2009-07/.

[7] Qt, http://qt.nokia.com.

[8] OMG-DDS, http://portals.omg.org/dds/.

[9] MATLAB®, http://www.mathworks.com.

[10] MEDM, http://www.aps.anl.gov/epics/extensions/medm/.

[11] Qwt, http://qwt.sourceforge.net.

[12] S. Baek et al., "KSTAR Widget Toolkit using Qt Library for the EPICS Based Control System", ICALEPCS 2009, Kobe, Japan, pp. 146–148.

[13] A. Rhyder, A. Owen, G. Jackson, "Qt EPICS Development Framework", PCaPAC 2010, Saskatoon, Saskatchewan, Canada, pp. 30–32.

[14] M. Böge, J. Chrin, "Developments to the SLS CORBA Framework for High Level Software Application", ICALEPCS 2005, Geneva, Switzerland, paper ID: WE4A.1-5O.

[15] M. Böge, J. Chrin, "An Event Service for the Propagation of Data", SLS Note: SLS-TME-TA-2004-0255, Dec. 2004, http://ados.web.psi.ch/slsnotes/tmeta040255.pdf.

[16] OpenSplice, http://www.opensplice.com.

[17] J. Chrin, G. Prekas, "A Taste of CAFE", ICALEPCS 2009, Kobe, Japan, pp. 821–823.

[18] M. Dach et al., "Control System in SwissFEL Test Injector Facility", ICALEPCS 2011, Grenoble, France, *These Proceedings*.

[19] T. Terebilo, "Channel Access Client Toolbox for MATLAB", ICALEPCS 2001, San Jose, California, USA, pp. 543–544.

[20] MATLAB Channel Access (mca), http://sourceforge.net/apps/trac/epics/wiki/MatlabChannelAccess.

[21] J. Chen et al., "CDEV: An Object-Oriented Class Library for Developing Device Control Applications", ICALEPCS 1995, Chicago, Illinois, USA, Paper ID: M4B-a.

[22] J. Chrin, "Hard Rock CAFE", EPICS Collaboration Meeting, 3-7 October 2011, PSI, Villigen, Switzerland, http://indico.psi.ch/event/EPICS.