ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-WEPGW079

A CHANNEL ACCESS SOFTWARE PLATFORM FOR BEAM DYNAMICS APPLICATIONS IN SCRIPTING LANGUAGES

J. Chrin, M. Aiba, J. Snuverink, Paul Scherrer Institut, 5232 Villigen PSI, Switzerland

Abstract

To facilitate the seamless integration of EPICS (Experimental Physics and Industrial Control System) into high-level applications in particle accelerators, a dedicated modern C++ Channel Access interface library, CAFE, provides a comprehensive and user-friendly interface to the underlying control system. Functionality is provided for synchronous and asynchronous interaction of single and composite groups of channels, coupled with an abstract layer tailored towards beam dynamics applications and complex modelling of virtual accelerators. Equivalent consumable solutions in scripting and domain-specific languages can then be accelerated by providing bindings to the relevant methods of the interface platform. This is exemplified by CAFE's extensive MATLAB® interface, incarnated through a single MATLAB executable (MEX) file, and a high performance Python interface written in the Cython programming language. A number of gratifying particularities specific to these language extension modules are revealed.

MOTIVATION AND CONTEXT

EPICS (Experimental Physics and Industrial Control System) is an established framework that comprises an extensive set of software tools for the development of distributed control systems in the field of particle accelerators and large-scale experiments [1]. Its principal communication protocol, Channel Access (CA), allows for the optimized throughput of data between server and client.[1] A native C-based application programming interface (API) [2] provides remote access to low-level hardware data encapsulated in EPICS Process Variables (PVs) residing in Input/Output Controllers (IOCs) or other devices hosting a CA server. The CA client library has thus provided the means by which extensions or bindings to C/C++-based scripting and domain-specific languages were first created. For each language instance, the resulting CA extension methods are typically intertwined with the specifics of a given domain's C/C++ extension framework. Their context is consequently confined to the system in which they execute and cannot be readily applied to other domains.

Another approach is to enforce a logical boundary between the CA components and that of the domain's C/C++ extension API, by providing a comprehensive C++ software platform, with sufficient breadth and flexibility, to act as host to any number of C/C++-based high-level languages. Significant advantages may be gained in this way:

- The process of creating bindings to scripting and fourth-generation languages is greatly streamlined.
- The inherent simplicity and convenience of maintaining an otherwise complex CA interface code in a single repository.
- A codebase that is well tested through its application from different domains.
- A uniform response to errors and exceptions that facilitate tracebacks.
- In-house CA expertise ensures a rapid response to user requests and problem solving.

It is within this context, coupled with a desire to avoid any deprecated APIs propagating to new facilities, that motivation for the CAFE (A Channel Access interFacE) library has developed [3].

A COMMON INTERFACE APPROACH

CAFE is a modern, multifaceted C++ CA client library that follows recognized practices in CA programming [4], placing careful attention to:

- Management of client-side connections.
- Memory optimization, particularly when connections are restored.
- Separation of data retrieval from its presentation.
- Strategies for converting between requested and native data types.
- Caching of pertinent data related to the PV and its connection state.
- Aggregation of requests for improved performance.
- Adaptive correction procedure, e.g., for network timeouts.
- Capturing and reporting results from method invocations with integrity.

The library provides a concise, diverse, and clean interface with minimal details of the low-level CA implementation disclosed to the user. Functionality for synchronous and asynchronous interaction is provided through a variety of methods. Abstraction layers tailored towards the development of accelerator applications have also been implemented, e.g., related data sets may be addressed as a single logical software entity (optionally configurable through XML). The structure of the codebase is of sufficient breadth to support end solutions in scripting languages by providing ready-made interfaces, with compliant data types, commonly required in control system interactions. (As a point of illustration, a `std::vector` in C++ maps directly to a Python `list`, facilitating the associated adapter layer.) The resulting library is thus equipped to serve as a reliable software platform

[1] Recent EPICS releases have been augmented with an additional network protocol, pvAccess, providing structured data types and improved bandwidth.

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-WEPGW079

that provides ample capabilities for data retrieval and control. The implementation details of the CAFE C++ API are described elsewhere [5].

SCRIPTING LANGUAGE EXTENSIONS

The designation of the CA programming components to a single library results in a reusable codebase that lessens the extent to which adapter objects need be created to expose prescribed control functionality to scripting languages. Furthermore, in implementing the adapters that facilitate the data transfer between the CAFE library and a given domain's workspace, a recognized pattern emerges: after validation of input arguments, the process effectively reduces to mapping C and CAFE structured data types to their most appropriate domain-specific counterpart.

Figure 1 illustrates the software architecture for the integration of CAFE to high-level applications in MATLAB® and Python, as described in the following.

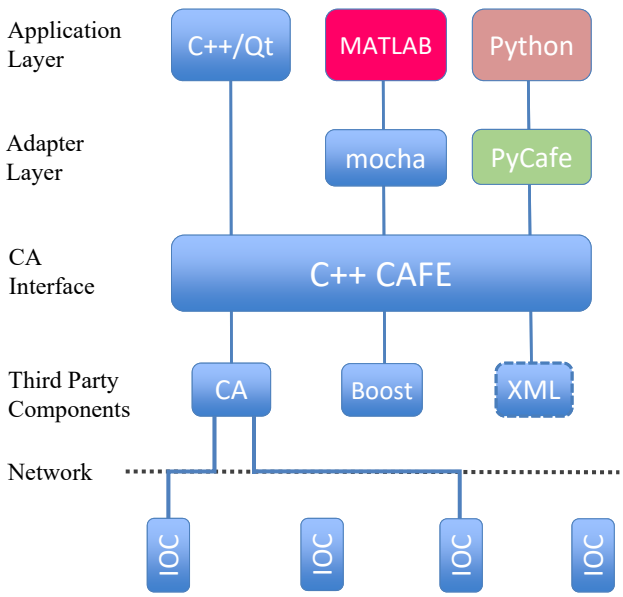

Figure 1: The software architecture for the integration of CAFE to high-level applications. The `mocha` and `PyCafe` adapters have been written in C++ and Cython, respectively.

The `mocha` MATLAB Executable File

MATLAB is an interpretive, fourth-generation programming language and numerical computing environment, providing a graphical user interface framework and support for object-oriented programming. Its use in the accelerator community can be exemplified by the MATLAB Middle Layer (MML) [6] which assembles third-party toolboxes that collectively provide a framework for machine simulation and operation. (The MATLAB Channel Access (MCA) package [7] is one such tool item).

MATLAB was also the preferred language for prototype applications at the SwissFEL Injector Test Facility (SITF) [8]. Here, connectivity to EPICS has been established through a single, dynamically loadable MATLAB executable (MEX) file, `mocha`, which enables CAFE routines to be called from within the MATLAB workspace, in much the same manner as MATLAB built-in functions. The motivation for the `mocha` MEX file originates from a reappraisal of the originally adopted MCA package and requirements dictated by SITF, among which the need for support for all MATLAB data types and compilation on 64-bit Linux architectures (with 64-bit indexing) were paramount. The `mocha` interface is further designed such that the user is able to communicate with the control system with minimal coding. For example, an operation on a PV can be undertaken without the user having to explicitly establish a CA connection *a priori*; this is handled autonomously at the time of initial interaction. An account of the `mocha` syntax and its interface to the underlying CAFE functions is given in Ref. [9].

The `mocha` release is accompanied with the long established MCA scripts wherein the original, but now deprecated, underlying CA calls have been replaced with `mocha` functions. By simply adding the `mocha` MEX file (and associated *mocha-fied* MCA scripts) to the MATLAB path, MCA users can immediately take advantage of CAFE's re-connection management, rigorous error reporting, and improved performance, without having to modify their own code. Applications requiring extended CA functionality may achieve this by calling `mocha` methods directly.

The `PyCafe` Python Module

Python is an interpretative and object-oriented programming language with dynamic typing and bindings, whose syntax is advocated for promoting conciseness and readability. The availability of third-party computational, data visualization, and accelerator design (CPython) modules add to its appeal. The integration of controls system functionality is accomplished by wrapping pre-existing C/C++ code into specialized Python modules. The traditional approach is to use Python's C API, which, although offering optimal performance, appears convoluted in comparison to specialized toolkits, such as SWIG and the Boost.Python C++ library, or the `ctypes` Python package. The procedure presented here differs from other endeavours in that it encompasses the emerging Cython technology [10] to engineer a high-performing EPICS interface through the CAFE library.

Cython is a compiled language that provides a Python-like style of coding while preserving the performance level of C. Cython wrappers have been implemented for a full complement of CAFE methods and made available through the PyCafe extension module. Some particular features include interfaces that cater for Python built-in types that implement the new Python protocol buffer, e.g., `memoryview` and `ndarray`, allowing their data to be shared without copying. The optimized C++ code generated by the `cython` compiler further results in a ~40% performance improvement for a scalar retrieval operation when compared to `ctypes`, which are subject to their Python overhead. The implementation of the Cython interface to CAFE is described in Ref. [11].

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-WEPGW079

SELECTED USE CASES

The CAFE library and its extensions can be readily consumed in various ways and for different purposes, as exemplified by the following use cases.

Virtual Accelerator

A virtual accelerator (VA) [12] integrates an accelerator model with control system channels to simulate the accelerator particle beam trajectory. Its principle motivation is to provide a platform for the early development of beam dynamics applications in advance of the accelerator's completion. The procurement of ready prepared applications hastens the commissioning procedure in readiness for user operation. While virtual accelerators are now integral to new facilities, their implementations differ as advancements in software technologies are made. Here, a virtual accelerator for SwissFEL has been realized entirely in Python. A Python extension module provides access to an in-house, Cython-based, online model that computes the SwissFEL beam-optics parameters (following established accelerator and lattice design principles [13, 14]), while interaction to a CA server hosting the VA PVs is executed through PyCafe.

An important contributing factor in the realization of the virtual accelerator is the relative ease with which a Python callback function may be supplied for any asynchronous CA interaction, whether it be a data retrieval or control operation. The Python callback function is invoked outside the CA context of the underlying C++ callback and, as such, overcomes its inherent non-reentrancy limitation. Interaction to other channels may thus be induced from within the Python callback allowing complex feedback systems and sequences to be invoked. Such a capability is essential in ensuring a realistic response to changes in the VA settings.

The extensive use of PyCafe in the SwissFEL virtual accelerator, deploying several thousand PVs, directed many aspects of the Cython interface, and further verified the underlying C++ CAFE library.

SwissFEL

While MATLAB was the preferred language for SITF, Python is emphasized at SwissFEL. Python applications developed with PyCafe, under the auspices of the virtual accelerator (of which the orbit feedback tool, linac energy manager and feedback, and beam based alignment, are a few), could be readily applied at SwissFEL with little modification. GUI development is realized with PyQt, where the passage of asynchronous data to widgets is triggered from within the supplied callback function using Qt's specialized signals and slots mechanism.

Swiss Light Source (SLS)

A primary motivation of the CAFE C++ library was to target the deprecated CDEV library [15] in use at the SLS. CDEV C++ classes feature in the deployment of a CA server whose persistent objects are accessed by beam dynamics applications through a CORBA middleware layer [16]. The software architecture is to be readdressed in the future in conjunction with the planned SLS-2 upgrade [17]. In the meantime, the most recent of SLS applications have been delivered in C++/Qt atop of purposefully built CAFE interfaces, e.g., for the synchronized readout of multiple waveform records to facilitate linear optics measurements and corrections [18].

Interfaces for specialized tasks are easily provided in CAFE. One such example is a 'set and match' utility that sets a PV to a given value and follows an associated readback PV to verify whether or not it reaches the specified value within a given tolerance and timeout period. The method is particularly suited to motion controllers, e.g., for the operation of insertion devices in accelerator-based light sources. An extended interface further allows multiple PVs to be set with a single interaction and their associated readback PVs to be tracked simultaneously (and likewise verified against their respective set values), as may be required in multidimensional scans.

High Intensity Proton Accelerator (HIPA)

Similarly to the SLS, motivation for CAFE's use at the HIPA cyclotrons facility originated from the requirement to eliminate the dependency on unsupported CA code. The migration to CAFE proved straightforward and its powerful interface resulted in a reduced codebase for several applications. In addition, good practices, such as establishing channel connections in unison at application startup, thereby reducing both network traffic and wait time, were easily implemented.

A project is also underway to reduce the frequency of beam interruptions initiated by the machine protection system (mostly due to beam loss) by recognizing the accelerator beam conditions that lead to their occurrence with the aid of machine learning techniques. Predictive corrections can then be applied in advance to prevent the anticipated interruption. The time evolution of all diagnostic signals, numerous accelerator set points, and additional environment signals, such as temperatures, will serve as input to the decision making process. To facilitate the high throughput of data, CAFE's advanced interface features, i.e., that incorporate the readout of aggregated channels, will be put into good effect.

CONCLUDING REMARKS

The CAFE C++ software platform provides several intuitive and user-friendly interfaces to EPICS that shelter the user from the intricacies of programming with the native C Channel Access library. Its capability to serve as an EPICS integration layer to application developers, in their many programming languages and for their various purposes, has been demonstrated. CAFE has been developed on the Linux platform, but also successfully compiled on macOS and Windows. The open-source software is available from the CAFE website [3] where extensive code examples demonstrate its usage.

ISBN: 978-3-95450-208-0 doi:10.18429/JACoW-IPAC2019-WEPGW079

REFERENCES

[1] EPICS, https://epics-controls.org

[2] J. O. Hill and R. Lange, "EPICS R3.14 Channel Access Reference Manual", http://epics-controls.org/resources-and-support/documents/ca/

[3] CAFE, http://cafe.psi.ch

[4] D. Zimoch, "Channel Access client programming", presented at the EPICS Collaboration Meeting, Jul. 27-29, 2009, NFRI, Daejeon, Korea, https://epics.anl.gov/meetings/2009-07/

[5] J. Chrin, "An update on CAFE, a C++ Channel Access client library and its scripting language extensions", in *Proc. 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15)*, Melbourne, Australia, Oct. 2015, pp. 1013–1016. doi:10.18429/JACoW-ICALEPCS2015-WEPGF132

[6] G. Portmann, J. Corbett, and A. Terebilo, "An accelerator control middle layer using MATLAB", in *Proc. 2005 Particle Accelerator Conf. (PAC'05)"*, Knoxville, TN, USA, May 2005, paper TPAT077, pp. 4009–4011.

[7] T. Terebilo, "Channel Access client toolbox for MATLAB", in *Proc.8th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'01)*, San Jose, CA, USA, Nov. 2001, paper THAP030, pp. 543–544.

[8] T. Schietinger *et al.*, "Commissioning experience and beam physics measurements at the SwissFEL Injector Test Facility", *Phys. Rev. Accel. Beams*, vol. 19, p. 100702, 2016. doi:10.1103/PhysRevAccelBeams.19.100702

[9] J. Chrin, "MATLAB objects for EPICS Channel Access", in *Proc. 14th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13)*, San Francisco, CA, USA, Oct. 2013, paper MOPPC146, pp. 453–456.

[10] Cython C-Extensions for Python, https://cython.org

[11] J. Chrin, "A Cython interface to EPICS Channel Access for high-level Python applications", in *Proc. 11th Int. Workshop on Personal Computers and Particle Accelerator Controls (PCaPAC'16)*, Campinas, Brazil, Oct. 2016, pp. 21–24. doi:10.18429/JACoW-PCAPAC2016-WEUIPLCO04

[12] N. Yamamoto, "Use of a virtual accelerator for a development of an accelerator control system", in *Proc. 1997 Particle Accelerator Conf. (PAC'97)*, Vancouver, BC, Canada, May 1997, pp. 2455–2457.

[13] MAD - Methodical Accelerator Design, http://madx.web.cern.ch/madx/

[14] K. Fuchsberger and Y. Inntjore Levinsen, "PyMad – Integration of MadX in Python", in *Proc. 2nd Int. Particle Accelerator Conf. (IPAC'11)*, San Sebastián, Spain, Sep. 2011, paper WEPC119, pp. 2289–2291.

[15] J. Chen, G. Heyes, W. Akers, D. Wu, and W. A. Watson III, "CDEV: an object-oriented class library for developing device control applications", in *Proc. 1995 Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'95)*, Chicago, IL, USA, Oct./Nov. 1995, paper M4B-a.

[16] M. Böge, J. Chrin, M. Muñoz, and A. Streun, "Development of beam dynamics applications within a CORBA framework at the SLS", in *Proc. 7th European Particle Accelerator Conf. (EPAC'00)*, Vienna, Austria, Jun. 2000, paper TUP7B10, pp. 1354–1356.

[17] A. Streun *et al.*, "SLS-2 - the upgrade of the Swiss Light Source", *J. Synchrotron Radiat.*, vol. 25, pp. 631–641, 2018. doi:10.1107/S1600577518002722

[18] M. Aiba, M. Böge, J. Chrin, N. Milas, T. Schilcher, and A. Streun, "Comparison of linear optics measurement and correction methods at the Swiss Light Source", *Phys. Rev. ST Accel. Beams*, vol. 16, p. 012802, 2013. doi:10.1103/PhysRevSTAB.16.012802