CXv4, A MODULAR CONTROL SYSTEM

Dmitry Bolkhovityanov, Pavel Cheblakov, Fedor Emanov,
Budker Institute of Nuclear Physics, SB RAS, Novosibirsk, Russia

Abstract

CX control system is used at VEPP-5 and several other BINP facilities. CX version 4 is designed to provide more flexibility and enable interoperability with other control systems. In addition to device drivers, most of its components are implemented in a modular fashion, including data access at both client and server sides. The server itself is a library. This approach allows clients to access several different control systems simultaneously and natively (without any gateways). CXv4 servers are able to provide data access to clients from diverse CS architectures/protocols, subject to appropriate network module being loaded. The server library, coupled with "null link" client-server access module, allows to create standalone monolythic programs for specific small applications (such as test benches and device test screens/utilities) using the same ready code from large-scale control system but without its complexity.

CXv4 design principles and solutions are discussed and first deployment results are presented.

CX

CX was designed at BINP in late 1990s as a general control system framework which runs on Linux and several flavors of *NIX. Initially made for CAMAC, CX now supports a wide range of hardware attached via different fieldbuses, including PCI/CompactPCI, VME, CANbus, RS232/485 and Ethernet.

CX is used at VEPP-5 Injection Complex [1] and at several other BINP facilities and small-scale experiments [2–4].

VEPP-5 will supply electrons and positrons to two BINP machines — VEPP-4 [5] and VEPP-2000 [6] in the near future. VEPP-4 uses a mix of an inhouse-designed software (with roots dating back to 1970s) [7] and EPICS; VEPP-2000 employs a custom software named VCAS [8]. For VEPP-5 to carry out its mission, its control system must be able to communicate with partners' control systems.

THE PROBLEM OF INTEROPERATION

CX, like most control system frameworks used in high energy physics experiments, is distributed and is based on a 3-layer model (Fig. 1a). A typical framework is closely tied to some network protocol and is often designed jointly with a dedicated protocol. I.e., the client library implements the client side of a protocol, and server implements its side (Fig. 1b); CX belongs to this class.

However, a need to interact with a different control system framework arises sooner or later in real world. This is often solved via "gateways" — dedicated software, translating one protocol to another (Fig. 2). This approach has its disadvantages besides a necessity for an additional software component. Not only does such gateway have to convert between protocols, but sometimes between different data paradigms, it convering in a very inconvenient place.

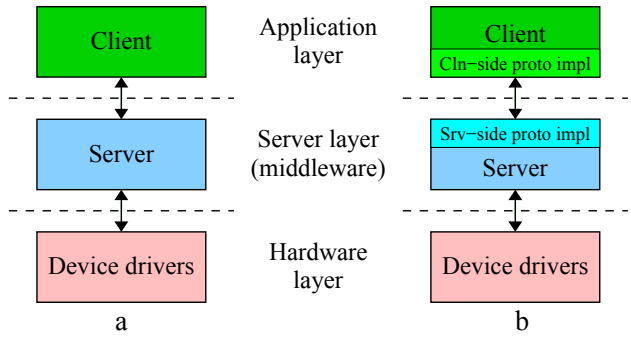

Figure 1: 3-layer architecture. (a) General layout; (b) Client-server communication.

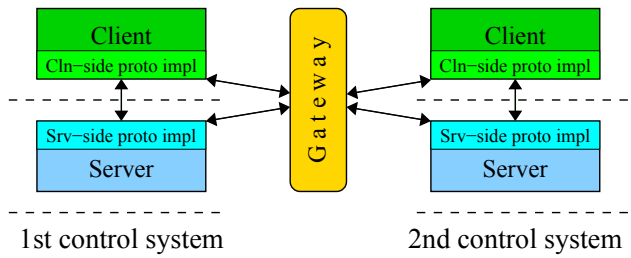

Figure 2: Interaction between different control systems via a gateway.

SOLUTION

In authors' opinion [9, p.3], the key to solution is separation of network protocol specifics and implementation from both client libraries and server. That was done in CX version 4 (see Fig. 3).

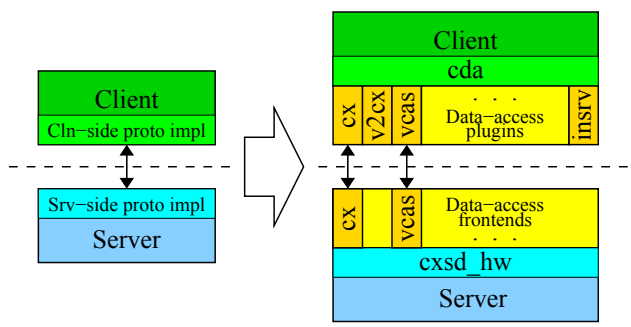

Figure 3: CXv4 modular structure.

Client library doesn't interact with server directly but rather via plugins. The library core provides clients with

ISBN 978-3-95450-148-9

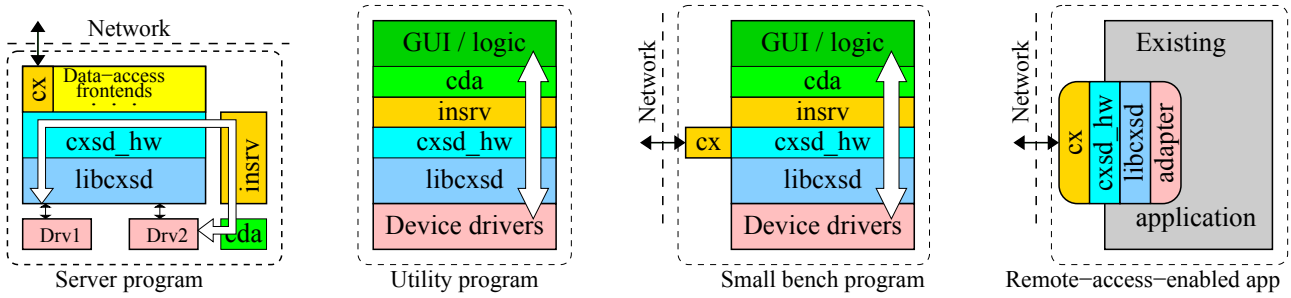

Figure 4: Possible combinations of CX modules.

protocol-agnostic access to data, hence its name "CDA" being Cx Data Access. Data-access plugins are free to implement any communication protocol; this enabled to support old CX versions seamlessly via "v2cx" plugin.

The situation is mirrored on a server side. Server core isn't engaged in network interaction, but just maintains an operating environment for data (channels) and drivers. External access to data is provided by plugins so-called "data frontends". Frontends access data via a simple API provided by `cxsd_hw` module, and are also free to implement any communication protocol.

Server code is implemented as a library (libcxsd), so that the server binary (cxsd) should be just a "conductor" that controls an orchestra of software modules (libcxsd, frontends, drivers, libraries, etc.).

Since most components of client and server sides are modules, these modules can be combined in several interesting ways.

- Device drivers can access channels from neighbor drivers in the same way as clients via "insrv" plugin/frontend (Fig. 4a). I.e., a unified API is used for both local and remote access.
  This gives freedom to place parts of control system (such as calculations or control logic) at either client or server side.
  Besides, data from other servers (and control systems) is accessed in exactly the same way (albeit via a different protocol(s)).
- The same `insrv:` "null-link" can be used to combine GUI and server with drivers into a single simple application without any need for network interaction (Fig. 4b).
  This is handy for small tasks such as a test bench for some type of device.
- A similar setup with a network frontend added (Fig. 4c) is suitable for small facilities where a full-scale 3-layer control system would be an overkill.
  This setup combines simplicity of a monolythic application with ability to access control channels remotely (which is useful for various tools like archivers).
- Implanting `libcxsd` with `cx` frontend to an existing application (such as an "all-in-one" test bench program) is a cheap way to add it remote access abilities (Fig. 4d).

IMPLEMENTATION CHALLENGES

Different Data Paradigms

Approach to data management differs between control systems.[1] Generally the concept of "channels" is used. A channel is a named data entity which can be atomically read and/or written to. However, more variants, such as properties, commands, methods, etc. exist.

To minimize interoperation problems CX uses as simple approach as possible:

- Any object addressable in a control system is a channel.[2] Channels can be either read-only or read-write.
- Commands are presented as channels. Writing to "command channel" triggers an action; value written can be used as parameter.
  This convention is implemented at device driver level and doesn't require any specific handling at other levels.
- Channels are addressed by names at the client side. A name can begin with a "PROTOCOL::" prefix which selects a way to access data and is usually the name of a corresponding protocol or control system (missing prefix taken to be "CX::").
  What goes after PROTOCOL:: is up to data-access-plugin and is passed to it verbatim.

Main Loop Integration

Any non-trivial application or application framework implements some kind of a main loop. Interaction with the main loop can use different models: callbacks in Xt/Motif, signals/slots in Qt, events in LabWindows, select()/poll() in console programs, etc. This diversity of main-loop paradigms presents serious problems when making code which should be able to "live" in any environment.

However, let's look what is required for vital activity of an I/O library or a device driver.

1. To watch for events on a file descriptor (typically readyness for read and/or write).
2. To request a timeout (either after some period or at exact time).

[1] Probably some "ideal way" to operate data in control systems do exist (as well as "idea of data", in platonian sense), but authors are unaware of any definitive work on this subject.

[2] This is similar to *NIX approach "everything is a file".

ISBN 978-3-95450-148-9

This functionality is present in any main loop implementation. And since this functionality is very limited and simple it is possible to create a formal API with different implementations for different main loops.

Such API was created for CX-server circa 2005. It it is called CXscheduler [10] in a "native" console form and implements the event-driven main loop based on `select()` syscall. API implementations (adapters) for Xt/Motif and Qt exist, libev and LabWindows implementations are under development.

Since all levels and components of CX employ CXscheduler all of them can be used in either console or GUI applications in a uniform way.

Modularity

Since the early days of CX device drivers aren't statically linked into CX server binary but are loaded at run-time, via `dlopen()`.

This approach was extended in v4: data-access modules, frontends, screen instruments in GUI programs are implemented as plugins. Even various configuration-file readers (such as hardware configuration for server, screen description for screen manager) are plugins; this allows us to use external macro-processors (currently m4) or some database instead of files.

Plugins can be either dynamically loaded or linked in statically (for platforms without `dlopen()`); care is taken of this by a dedicated "cxldr" component.

PERFORMANCE

Additional functionality was expected to come at some performance cost.

- Plug-ins management and data routing between plugins and core libraries obviously require some CPU time, which was hard to estimate in advance.
- Initial versions of CX had a limitation for scalar data at device drivers' level to be 32-bit integers only, since this fits most control hardware and significantly simplifies programming. CXv4 removes this limitation, supporting data of various formats (int, float, text) and of different sizes; conversion is performed automatically, if required. This was also expected to cost some performance loss.
- Performance can be especially critical for the most resource-starved pieces of control system computer hardware — CAMAC and CAN intelligent controllers [11] running Linux on 50MHz PowerPC CPU.

However, simple benchmarking shows that neither additional modularity nor more flexible data model cause any performance degradation. And software in intelligent controllers even got several percent performance gain due to optimizations in remote-driver-environment library.

CXv4 DEPLOYMENT

This part of work has been supported by Russian Science Foundation (project N 14-50-00080).

VEPP-5

Drivers for a whole range of hardware used at VEPP-5 were ported from CXv2 to CXv4 during 2014–2015. Control system had switched from v2 to v4 during summer'2015 maintenance stop.

Besides regular data-access plugins, a special "formula scripting" CDA plugin is used. It allows us to perform simple calculations like "return value of $channel1 + channel2 * channel3$" (for read channels), as well as simple sequences like *"put user-input values to channel1 and channel2; pause for 2s; put 1 to channel3"* (for write channels). Availability of this simple scripting in a screen-manager application gives freedom of where to place such macro-actions either at a server level or at a client side, and allows us to easily debug it first in the former case.

VEPP-2000 Interaction

VCAS support has been implemented and tested. Further works on interoperation between VEPP-5 and VEPP-2000 control systems will continue when VEPP-2000 resumes operation in 2016.

Electron Beam Welding Facility

BINP electron beam welding facility is of small scale. It employs a small (10 devices, depending on experiment), but diverse set of control hardware (including CAMAC, CANbus and RS485). Thus, 3-layer control system was used from the very beginning.

However, a full 3-layer software stack is a bit excessive for a small facility. Modular nature of CXv4 enabled to unite GUI and server parts into a single application, as shown on Fig. 4c (and there are several variants depending on experiment being carried out). As this application includes a CX network data-access frontend, benefits of a 3-layer control system are retained.

LIA-2

LIA-2 still runs CXv2 but all required device drivers are ready, and high-level software is under development; switch to CXv4 is scheduled to 2016.

FUTURE AND PROSPECTIVE WORKS

As the key point of CXv4 is its flexibility and ability to interact with other control systems, the main direction of development is expanding the repertoire of data-access plugins and frontends.

EPICS Integration

The nearest target is interaction with EPICS. At present time, client-side access to EPICS is implemented at Python client library; it will be moved to a CDA plugin. EPICS/CA

ISBN 978-3-95450-148-9

frontend for CX-server is also planned, its implementation will enable both-sided communication with VEPP-4 control system.

NI Integration

BINP uses NI software (LabWindows) and hardware (such as CompactRIO). When interoperation between NI products and CX was required, it was usually implemented via some simple custom protocols.

However, with CXv4 modular structure it seems possible to use CDA in LabWindows and to mate a slightly modified version of CX-server with CompactRIO internals (as shown on Fig. 4d). These projects are currently under development.

CONCLUSION

CX version 4 employs a modular approach with client-server communication separated from both client and server cores and implemented in a plugin fashion.

This model proved to be flexible enabling direct communication with other control system frameworks. Additional flexibility makes CXv4 more suitable for control systems of various scales from small test stands to large facilities.

REFERENCES

[1] A.A.Starostenko et al., "Status of Injection Complex VEPP-5: machine commissioning and first experience of positron storage", IPAC2014, Dresden, Germany, MOPME073 https://inspirehep.net/record/1314333/files/mopme073.pdf

[2] D.A.Starostenko et al., "Results of operating LIA-2 in radiograph mode", Physics of Particles and Nuclei Letters, September 2014, Volume 11, Issue 5, pp 660-664.

[3] D.Bolkhovityanov et al., "Design and Development of a Control System for Intense Source of Radioactive Ions prototype", ICALEPCS'2005, 10-14 November, 2005, Geneva, Switzerland, P1-091.

[4] Yu.I.Semenov et al., "60 KEV 30 KW electron beam facility for electron beam technology", EPAC'08, June 23-27, 2008, Genoa, Italy, TUPP161.

[5] V.E.Blinov, et al., "The status of VEPP-4", Physics of Particles and Nuclei Letters, 2014, Vol.11, No.5, pp. 620-631.

[6] "VEPP-2000 Project", http://vepp2k.inp.nsk.su/

[7] A.Bogomyagkov, et al., "Automation of operations on the VEPP-4 Control System", ICALEPCS-05, 10-14 November, 2005, Geneva, Switzerland.

[8] A.Senchenko et al., "VEPP-2000 Collider Control System", PCaPAC-2012, Kolkata, India, FRCB04 (2012) http://epaper.kek.jp/pcapac2012/papers/frcb04.pdf

[9] D.Yu.Bolkhovityanov et al, "Present Status of VEPP-5 Control System", ICALEPCS2007, October 2007, Oak Ridge, USA, TPPB18.

[10] "CXscheduler: a simple scheduler for event-driven console applications" http://cxscheduler.sourceforge.net/

[11] D.Yu.Bolkhovityanov et al, "PowerPC-based CAMAC and CAN-bus Controllers in VEPP-5 Control System", PCaPAC2005, 22-25 March, 2005, Hayama, Japan, WEB4 http://conference.kek.jp/PCaPAC2005/paper/WEB4.pdf

ISBN 978-3-95450-148-9