VEPP-5 INJECTION COMPLEX CONTROL SYSTEM BASE SOFTWARE UPGRADE

D. Bolkhovityanov*, F. Emanov, BINP SB RAS

Abstract

VEPP-5 Injection Complex control system software is based on CX framework. In the course of 2015-2016 it was upgraded to version 4. K500 transport channel (delivering e+/e- beams to VEPP-2000 and VEPP-4) has switched to CXv4 in 2017-2018. CXv4 is a redesign from the ground up, built in a modular fashion with maximum flexibility in mind. CXv4 server is easily configurable via plaintext files. Additionally, server configuration can be autogenerated from a database containing high-level information on the facility. CXv4 server supports artificial "mailbox" channels, which are used by high-level facility management software for inter-communication. For client-level access, a high-performance binding for Python exists, and visual programming tools are being developed.

BACKGROUND

VEPP-5 Injection Complex [1] operates at the Budker Institute of Nuclear Physics in Novosibirsk, Russia. It feeds BINP colliders VEPP-4M and VEPP-2000 with e+ and e- beams via K500 transport channel.

CX control system framework was developed at BINP in 1990s specifically for VEPP-5 [2]. Later it was also employed at several small- and middle-size facilities.

Its initial task was to serve CAMAC hardware. Later CAN-bus hardware support was added. Since CAMAC controllers used at VEPP-5 had changed several times during CX first years, its driver architecture was made modular since early days.

The 32-bit integer was hardwired as a datatype and it was adequate for most hardware. Digital oscilloscopes, CCD cameras and similar "vector-data" hardware were served in a special way. However, in mid-2000s new hardware appeared at BINP facilities, including that which natively operates with floating-point data. Thus, it became obvious that a broader datatype support is required.

Taking into account the accumulated experience of CX operation, including its benefits and shortcomings, a decision was made to create a new version from the ground up, reusing fragments of existing code, and this work started in late 2000s.

In 2015 the new CX version 4 (CXv4) was deployed at VEPP-5 for basic controls (magnetic and vacuum system, thermostabilization etc.) [3]. Digital oscilloscopes and CCDs followed the suit in 2016. Finally, K500 transport channel has switched to CXv4 in 2017–2018, as well as VEPP-5 RF-synchronization and modulators' controls.

* D.Yu.Bolkhovityanov@inp.nsk.su

CXv4 GENERAL STRUCTURE

CX is based on a classic 3-layer model (Fig.1a).

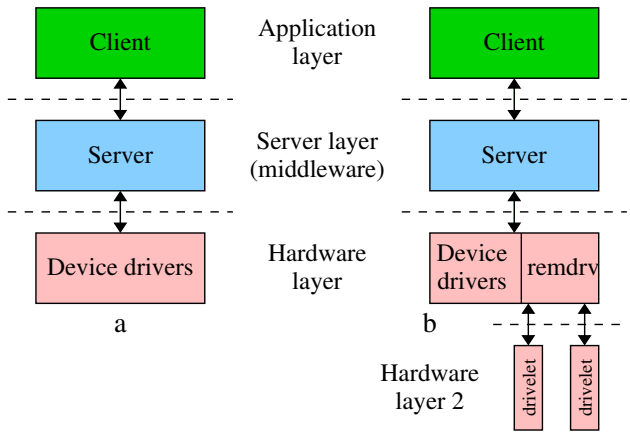

Figure 1: (a) Classic 3-layer architecture; (b) CX with 2-part hardware layer.

However, the lower layer can be split into two parts, if peripherial low-performance intelligent controllers are used (such as for CAMAC, CANbus or VME). In this case only a "low-weight" driver (called "drivelet" in CX) runs in such a controller and communicates with CX server (running on a more productive node) via network (Fig.1b).

Data Exchange Paradigm

As opposed to many other control systems, CX uses completely asynchronous data interchange at all levels — from device drivers up to operator screens.

The *"Read"* and *"Write"* requests don't suppose immediate result, but are rather treated as messages. Thus, a read request can be left without any answer at all. On the other hand, data can arrive without any request, on the initiative of the device (which is usual in case of externally-triggered devices). So, this is more of a publish/subscribe model rather than a client/server one.

CXv4 NEW FEATURES

Modularity

From the very beginning CXv4 was designed in a modular fashion [3]. Technology of "plugins" is used instead of a fixed implementation (see Fig.2).

Modules can be either statically linked or loaded at runtime (via `dlopen()`; this is taken care of by a dedicated "cxldr" component).

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

ISBN 978-3-95450-197-7

with protocol-agnostic access to data, hence its name "CDA" is Cx Data Access.
- External access to channels is provided by "data frontends" on the server side.
- The server itself is a library.
- Configuration file readers are plugins too.
- Screen instruments in GUI applications are implemented as plugins, thus making generic "screen manager" application extensible and capable of performing complicated control tasks.

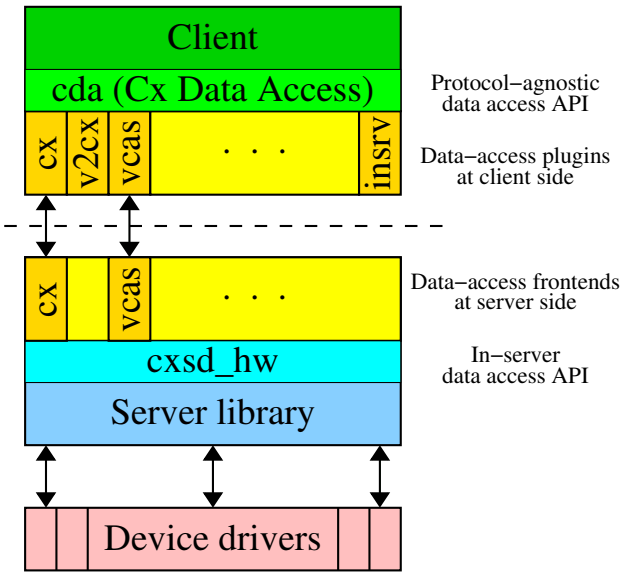

Figure 2: CXv4 core modular structure.

Inter-driver Data Access

Modularity of data-access protocols allowed us to create an `insrv::` "loopback" protocol for access from inside the CX-server. It is used by "virtual hardware" drivers to connect to underlying hardware. As access is performed via standard cda API, those virtual devices can be seamlessly pointed to hardware in other servers.

Data types

CXv4 supports integer (8-, 16, 32 and 64-bit), floating point (single- and double-precision) and character (byte and 32-bit (Unicode)) channels. Type and size conversion is performed automatically when required.

Any channel can be either a scalar or a vector. Scalars are channels "of fixed length of 1", while vectors' content can vary in length from 0 to a specified maximum. Character vectors (used for text strings) are treated specially: an additional NUL character is added at the end of data, thus simplifying use in C programs.

Python binding

A high-performance Python binding was created for CX. It lowers the barrier of entry into CX programming and greatly simplifies development of complex software, such as machine mode manipulation, automatic control and data analysis applications.

Hostnameless Addressing

CX channel naming convention supposes hierarchical names, consisting of level names separated with dots; such as

`Facility.System.Element.Channel`

Fully-qualified CX channel references have the following format:

`[PROTO::]HOST:N.path.to.channel`

i.e., an optional protocol spec followed by hostname with server instance number and, finally, a channel name inside the server.

Thus, one has to know which host/server a channel belongs to. This is definitely an extra burden, since this information reflects only low-level technical decisions and can change with time.

To overcome this problem, CX now allows a "hostnameless" addressing. If only a channel name is specified — e.g., just

`path.to.channel`

then CX data access layer performs a search to locate a containing server for requested channel. If later the channel moves to another server, resolving is repeated invisibly to an application.

Besides convenience, this feature gives ability to host adjacent points of hierarchy on *different* servers and to augment existing (hardware) hierarchies with additional calculated "artificial" channels.

For example, linac's `QL27` power supply is controlled by `canhw:11` server and, thus, its channels have full names `canhw:11.QL27.`*NNN*. And there's a separate server `softhw:1` for software-generated channels, such as `QL27.failure` (whose value is calculated based on several heuristics). With "hostnameless" addressing these servers can be maintained independently, however, their channels are still named adjacently.

Cpoints

Hardware channel names in CX consist of 2 parts: device instance name and a channel name inside device. E.g., `ctl_ql12.adc5` means *"ADC channel 5 in QL12 supply's controller"*. These names are created automatically upon device instantiation[1].

However, CX has a mechanism named *cpoints*[2], somewhat similar to Unix symlinks, which allows us to extend hardware namespace and create virtual hierarchies of channels.

In a simplest form like

```
cpoint ic.mag.QL27.set QL27.Iset
```

it creates an alias name. A whole device can also be aliased. Intermediate cpoint-"containers" (directories) are created automatically when utilized.

[1] CX behaves similarly to Algol/C-like programming languages: when a struct-typed variable is instantiated, all of its fields are created.
[2] The term "cpoint" is an abbreviation of "control point".

ISBN 978-3-95450-197-7

As virtual hierarchy has precedence over hardware one, it is possible to augment, replace and "redirect" hardware channels.

CONFIGURATION

Prior CX versions included a very simple configuration mechanism: just a textfile with a list of devices to be controlled, with per-device parameters (such as bus addresses).

However, as large experimental facilities (such as VEPP-5) require lots of configuration information of various kind, several attempts to create a generic centralized software and hardware configuration tool were made [4,5]. Unfortunately, none of these had gained reception. Main problem is that time expenditure to master and maintain these systems is higher than gained time savings (at least at facility of VEPP-5 scale). Thus, another solution was required.

First, CX configuration language was extended. Now it includes a notion of "device types" (which are first declared and than instantiated) and an instrument to create extensive virtual hierarchies of channels (see section *"Cpoints"* above). On the one hand, configuration textfiles are piped via M4 preprocessor, which gives rich programming abilities. On the other hand, core syntax is simple enough for configuration files to be software-generated (from some other sources).

Second, a VEPP-5-specific software was created. It includes a database with high-level information and configuration tools [6]. This software also includes a machine mode manipulation system, automatic control and data analysis programs.

Third, CX-server supports artificial "mailbox" channels, which aren't connected to any hardware and just hold values written, notifying subscribed clients about modifications. CX operates just as a software data bus here, allowing system configuration and management software intercommunication via the same way they access hardrare.

PROSPECTIVE DEVELOPMENT

First, a per-channel locking mechanism is being developed to enable exclusive access to sensitive controls and atomic modification of groups of related channels.

Second, `cda_d_epics` and `cda_d_tango` data access plugins are planned for CX client applications to be able to directly access respective "foreign" control systems used at peer BINP facilities.

Similarly, `cxsd_fe_epics` and `cxsd_fe_tango` frontends are being considered. This would allow peers a "native" access to VEPP-5 control system. Plus, these frontends would enable to use CSS and Taurus for CX.

CONCLUSION

In the course of 2015–2018 VEPP-5 Injection Complex control system was successfully upgraded. New base software allows easy data interchange with VEPP-4 and VEPP-2000 colliders.

A high-performance Python binding was created. System configuration and whole-facility mode saving and restoring software is Python-based. Some operator applications are also written in Python now, and visual programming tools for Python are being developed.

REFERENCES

[1] D. Berkaev *et al.*, "VEPP-5 Injection Complex: Two Colliders Operation Experience", Proc. IPAC2017, Copenhagen, Denmark, May 2017, paper WEPIK026, `http://accelconf.web.cern.ch/AccelConf/ipac2017/papers/wepik026.pdf`

[2] D. Bolkhovityanov, "VEPP-5 Injection Complex Control System Software", 2007, Ph.D. thesis (in russian) `http://www.inp.nsk.su/~bolkhov/publs/bolkhov_phd_final.pdf`

[3] D. Bolkhovityanov *et al.*, "CXv4, a Modular Control System", Proc. ICALEPCS2015, Melbourne, Australia, October 2015, paper WEPGF093, `http://accelconf.web.cern.ch/AccelConf/ICALEPCS2015/papers/wepgf093.pdf`

[4] A. Makeev *et al.*, "Centralized Software and Hardware Configuration Tool for Large and Small Experimental Physics Facilities", Proc. ICALEPCS2013, San Francisco, CA, USA, October 2013, paper TUPPC022, `http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/tuppc022.pdf`

[5] M.A. Ilina, P.B. Cheblakov, "Applying Ontological Approach to Storing Configuration Data", Proc. ICALEPCS2017, Barcelona, Spain, October 2017, paper THMPL05, `http://accelconf.web.cern.ch/AccelConf/icalepcs2017/papers/thmpl05.pdf`

[6] F. Emanov *et al.*, "Upgrade of Application-Level Software of VEPP-5 Injection Complex", this conference, paper THPSC12.

ISBN 978-3-95450-197-7