INTEGRATING CONTROL APPLICATIONS INTO DIFFERENT CONTROL SYSTEMS*

M. Killenberg†, M. Hierholzer, C. Schmidt, DESY, Hamburg, Germany
S. Marsching, aquenos GmbH, Baden-Baden, Germany
J. Wychowaniak, Łódź University of Technology, Łódź, Poland

Abstract

Porting complex device servers from one control system to another is often a major effort due to the strong code coupling of the business logic to control system data structures. Together with its partners from the Helmholtz Association and from industry, DESY is developing a control system adapter. It allows writing applications in a control system independent way, while still being able to update the process variables and react on control system triggers. We report on the status of the project and the experience we gained trying to write portable device servers.

INTRODUCTION

With embedded systems becoming more and more powerful, the algorithms in the devices which are accessed via control systems are becoming more and more advanced. Especially on MicroTCA [1] systems the hardware usually features a powerful multi-core CPU with several GB of RAM. The MicroTCA.4 [2] extension brings trigger and clock lines, as well as large rear transition modules which can be used for demanding analogue control applications.

Many particle accelerators which are currently being build are using or will use MicroTCA.4 for control of the radio frequency (RF) in the accelerator, for instance FLASH [3] and the European XFEL [4] hosted at DESY, Hamburg, ELBE [5] at Helmholtz-Zentrum Dresden-Rossendorf, or FLUTE [6] at KIT, Karlsruhe. The complex RF control applications shall be reused across the different accelerators, while all the facilities are using different control systems. It turned out that porting the software to a different control system is a major effort because the code is strongly coupled to the original control system. This lead to the idea to have an adapter layer between the device library, which implements the algorithms, and the control system, which provides the communication protocols and integration into the facility's control infrastructure. This adapter shall be part of the MicroTCA.4 User Tool Kit (MTCA4U) [7, 8], a collection of libraries to facilitate the implementation of control applications.

REQUIREMENTS

The main task of the adapter is to allow application code to access process variables which are communicated to the outside world in a control system independent way. For this, the adapter has to use the functionality which is provided by the control system, like communication protocols or the addressing scheme.

The part of code which is device *and* control system dependent has to be minimal, zero if possible. This type of code is causing the huge workload when porting and maintaining applications for multiple control systems.

Abstraction is easy to achieve if data is simply copied back and forth between two domains, but this comes with a performance penalty. So an additional requirement to the adapter is to avoid unnecessary copying, especially of large data structures like arrays. In addition, the adapter should be thread safe and usable in real time application.

As a starting point and to check if the abstraction is working, the adapter is tested with two control systems: DOOCS [9] and EPICS [10]. DOOCS (used at DESY for FLASH and the European XFEL), has an object-oriented data model written in C++. EPICS 3, one the most widely used control systems for particle accelerators and used at FLUTE, has a channel-based C API. We intentionally used two conceptually different control systems, hoping that the abstraction needed to work with these two should also allow other control systems to be used without too much modifications in the design.

DESIGN CONCEPT

The first implementation of the adapter focuses on process variables. It provides data structures for scalars (8, 16 and 32 bit signed and unsigned integers, single and double precision floating point), strings, and arrays of the numerical data types. Each process variable is identified by a unique name which describes its function inside the device code ("TEMPERATURE" for instance for a device with a temperature sensor). The name does not contain information where the device is installed and in which context it is used. This part depends on the control system and the facility, and is added in the control system specific part of the adapter, not in the device part.

The original idea to avoid copying, especially for large arrays, was to have a single instance of the data. This would be stored in a control system dependent type, an instance which always has to be there to work with the particular control system. The adapter would provide a wrapper, which would be used inside the business logic. But it turned out that this approach is not viable. DOOCS and EPICS have different locking schemes for their variables, and only the control system side could know when it is safe to access

* This work is supported by the Helmholtz Validation Fund HVF-0016 "MTCA.4 for Industry".
† martin.killenberg@desy.de

ISBN 978-3-95450-148-9

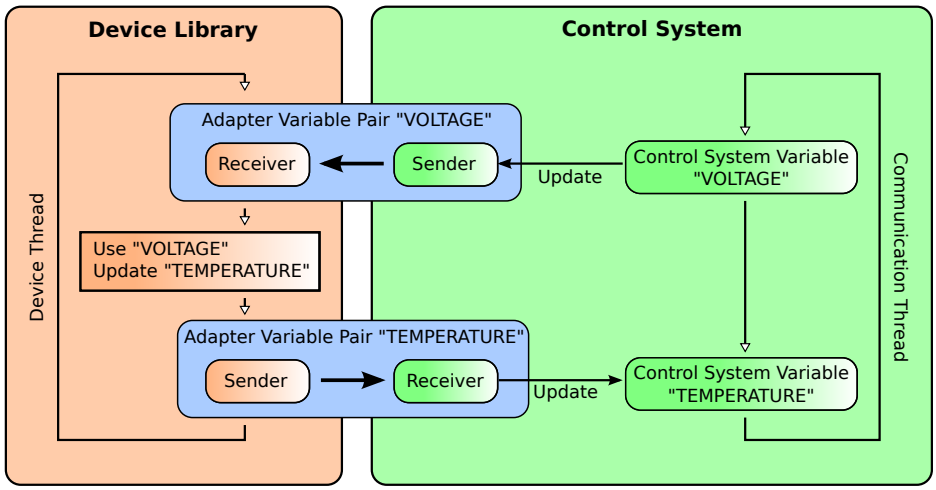

Figure 1: The update flow using the control system adapter.

the variables, but not the control system independent device part.

The solution is to have a control system independent instance on the device side which can be accessed at any time, and the control system variable on the other side, which can have a control system lock (Fig. 1). These variables have to be synchronised, which means that the abstraction at this point requires one copy which cannot be avoided.[1] As not all control systems allow a variables to be input and output at the same time, it was decided to restrict process variables to be unidirectional ("control system to device" or "device to control system").

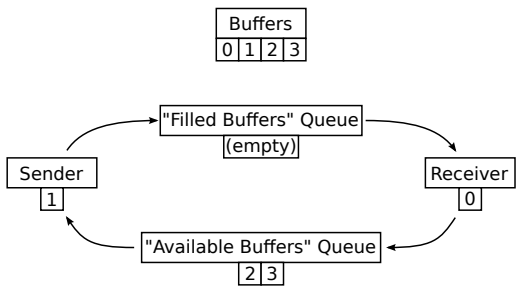

Figure 2: For arrays a process variable pair with sender and receiver features two lock-free queues and at least four pre-allocated buffers.

As one of the requirements for the adapter is to allow real time threads in the device library, the variable on the device side has to be lock-free. To implement this, the adapter's process variable is always a sender-receiver pair (Fig. 2), using lock-free queues for transfer. As dynamic memory allocation is not allowed in a real time thread, the mechanism is working with a pool of pre-allocated buffers for arrays. Sender and receiver each hold the reference to one buffer at all times, which allows the business logic to access and modify the data at will, except while sending or receiving. The data being transferred is always the reference to a buffer, not the buffer itself, which avoids unnecessary copying of large data structures.[2]

When receiving, the receiver will pop the head of the "filled buffers" queue. If it could get (the reference to) a new buffer, it will push the now outdated buffer to the "available buffers" queue, so the sender can reuse it. In case there are no updated buffers available, the receiver will hold on to the current buffer, as it contains the most up-to-date information that is available on the receiver side.

Before actually sending, the sender will pop the head of the available buffers queue to be sure it has a new buffer which it can fill after sending (at all times there must be at least one buffer on each the sender and the receiver side). After that, the buffer to be send is pushed to the "filled buffers" queue. Both receiver and sender first pop the head of the queue where they retrieve the next buffer, and then push the buffer which has been processed for use by the other side. As this can happen at the same time, it means there have to be at least four buffers.

In contrast to a triple buffer, which is a common scheme in real time applications, this approach does not require a dirty flag for the buffer, and the receiver does not have to swap back to keep the latest available buffer in case no updated buffer is found. As a further advantage, the number of buffers can simply be increased, allowing a longer queue in case there are fluctuations on the receiver side, but it is fast enough to catch up with a certain backlog.

If the receiver is too slow to process data at the rate at which the sender is producing it, data will be lost. This cannot be avoided. In our implementation with a fixed number of buffers it means the "available buffers" queue is empty. In a naive implementation, the sender would keep its current buffer to overwrite it (buffer 3 in Fig. 3). However, it is not a good solution to stop sending because this would result

[1] A copy can be avoided if the control system type allows swapping of the internal buffer.

[2] For scalars copying the value is not more expensive than copying the pointer. In this case the values are directly copied and no additional buffers are needed.

ISBN 978-3-95450-148-9

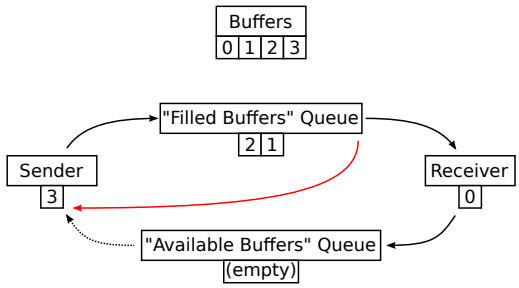

Figure 3: If the queue of available buffers is empty, a further send call would pop the head of the "filled buffers" queue to be overwritten (buffer index 1 in this picture), and perform the send operation (buffer index 3 goes to the queue). Like this, the information in the queue can be updated, whether the receiver is active or not.

in newer data being discarded in favour of older data that is already in the queue. If for instance the receiver was down for several minutes, the information in the queue would be several minutes old when it is received, while newer data has been overwritten. Instead, the oldest information which has not yet been received should be dropped. So in case no free buffers are available, the sender will pop the head of the "filled buffers" queue (buffer 1 in Fig. 3) to be overwritten, and send buffer 3, which has just been filled. Like this, the data in the queue is being updated even if the receiver is not active.

Creation of a Process Variable

Not only the device business logic has to be independent from the control system side, also the amount of device-specific control system code should be minimised. For this reason the creation of process variables is automated as much as possible in the adapter (Fig. 4). The device is calling the create function of the adapter, giving the name and the direction ("device to control system" or "control system to device"). Depending on the direction, a sender or a receiver is returned to the device side. The other partner of the process variable pair is stored in a list. After all process variables have been created by the device, the control system calls a function which creates the instances of the control system variables for all process variables known to the adapter. This function does not depend on the device logic, which improves the decoupling. It is, however part of the control system specific part of the adapter and looks different for each control system. The level of abstraction which can be achieved here may vary.

STATUS AND OUTLOOK

The current implementation of the control system adapter provides process variables in a common, control system independent part. A control system specific, but device independent part has to be added to the adapter for each target control system. Bindings for DOOCS and EPICS have been written and are ready to use.

As "proof of concept", some example devices with a couple of scalars and arrays have been created and tested using either DOOCS or EPICS, and have afterwards been ported to the other control system. This porting worked smoothly in both directions. As expected, the device code itself did not have to be modified. The amount of device-specific code is very small on the control system side. The example devices are down to three lines of device-dependent control system code, for DOOCS as well as for EPICS, independent from the number of process variables.[3]

To make use of additional control system features like variable limits or histories, these currently have to be implemented as device-specific code on the control system side. This introduces additional work when porting and maintaining the code for multiple control systems. It is currently being discussed how the adapter can be extended to allow the definition of the features on the device side, but use the functionality from the control system. Especially features which are expensive to implement like archiving and history should not be duplicated in the adapter.

CONCLUSIONS

The MTCA4U control system adapter provides an interface to use process variables in a device library without introducing a coupling to a particular control system. The implementation is lock free and transfers pre-allocated buffers without copying. This step was necessary to allow operation in control systems with different locking mechanisms. It also enables the device logic to use process variables in a real time thread.

The adapter consists of a common part, which implements the decoupling, and a control system specific part, which provides the particular bindings. As a proof of concept, adapters for DOOCS and EPICS have been implemented. An extension for OPC-UA is planned. As next steps, the abstraction of additional control system features like variable limits, engineering units and histories are intended.

All software is published under the GNU General Public License and available in the respective software repositories [11–13].

REFERENCES

[1] PICMG®, "Micro Telecommunications Computing Architecture, MicroTCA.0 R1.0" (2006).

[2] PICMG®, "MicroTCA® Enhancements for Rear I/O and Precision Timing, MicroTCA.4 R1.0" (2011/2012).

[3] C. Schmidt et al., "Real time control of RF fields using a MicroTCA.4 based LLRF system at FLASH", 19th IEEE Real-Time Conference, Nara, Japan (2014).

[4] M. Altarelli et al., "XFEL : The European X-Ray Free-Electron Laser : Technical Design Report", DESY-2006-097, DESY, Hamburg (2007).

[3] The device configuration files for the control system had to be written in addition.

ISBN 978-3-95450-148-9

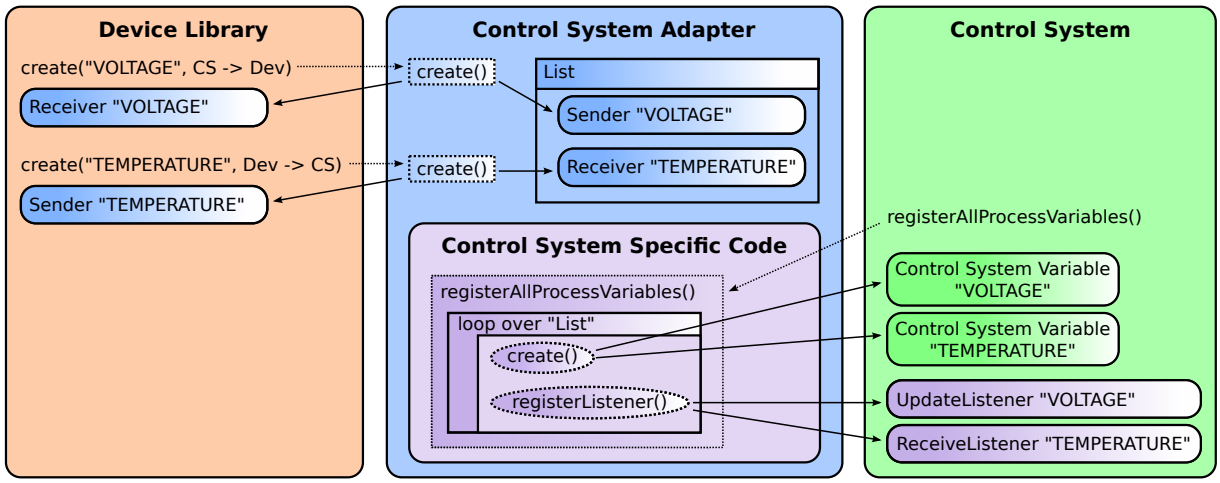

Figure 4: The creation of process variables is requested on the device side. The instantiation on the control system side is automated as much as possible inside the adapter.

[5] F. Gabriel et. al., "The Rossendorf radiation source ELBE and its FEL projects", Nucl. Instr. Meth. B 161-163, 1143 (2000), http://dx.doi.org/10.1016/S0168-583X(99)00909-X

[6] S. Marsching et al., "Status of the FLUTE Control System", WPO013, PCaPAC2014, Karlsruhe, Germany (2014).

[7] M. Killenberg et al., "Drivers and Software for MicroTCA.4", WEPGF015, *These Proceedings*, ICALEPCS'15, Melbourne, Australia (2015).

[8] MTCA4U—The DESY MicroTCA.4 User Tool Kit, Subversion Repository, https://svnsrv.desy.de/public/mtca4u

[9] The Distributed Object Oriented Control System (DOOCS), http://doocs.desy.de/

[10] Experimental Physics and Industrial Control System (EPICS), http://www.aps.anl.gov/epics/index.php

[11] MTCA4U Control System Adapter, Subversion Repository, https://svnsrv.desy.de/public/mtca4u/ControlSystemTools/

[12] MTCA4U EPICS Adapter, Subversion Repository, http://oss.aquenos.com/svnroot/epics-mtca4u/

[13] MTCA4U DOOCS Adapter, Subversion Repository, https://svnsrv.desy.de/public/mtca4u_applications/DOOCS_Adapter/

ISBN 978-3-95450-148-9