CHIMERATK – A SOFTWARE TOOL KIT FOR CONTROL APPLICATIONS

G. Varghese[*], M. Killenberg, M. Heuer, M. Hierholzer, L. Petrosyan, C. Schmidt, N. Shehzad, T. Kozak, M. Viti, DESY Hamburg, Germany
S. Marsching, Aquenos GmbH, Germany
A. Piotrowski, FastLogic, Poland
P. Prędki, J. Wychowaniak, Łódź University of Technology, Poland
A. Dworzanski, K. Czuba, Warsaw University of Technology, Poland
C. Iatrou, J. Rahm, TU Dresden, Germany
M. Kuntzsch, R. Steinbrueck, Helmholtz-Zentrum Dresden Rossendorf, Germany

Abstract

The presentation provides an overview of the ChimeraTK framework. The project started from a demand for software libraries that provide convenient access to PCIe bus based cards on the MicroTCA.4 platform. Previously called MTCA4U, ChimeraTK is evolving towards a set of frameworks and tools that enable users to build up control applications, while abstracting away specifics of the underlying system. Initially, the focus of the project was the DeviceAccess C++ library and its bindings for Matlab and Python, along with a Qt based client that used DeviceAccess under the hood. However, ChimeraTK has expanded to include more tools like the ControlSystemAdapter, VirtualLab and ApplicationCore. The ControlSystemAdapter framework focuses on tools that enable application code to be written in a middleware agnostic manner. VirtualLab focuses on facilitating testing of application code and providing functional mocks. The ApplicationCore library aims at unifying application interfaces to other tools in the toolkit and improving abstraction. We present an update on improvements to the project and discuss motivations and applications for these new set of tools introduced into the toolkit.

INTRODUCTION

MicroTCA.4 systems, with their ability to host powerful multi core CPUs, offer users the possibility of running computationally involved algorithms directly on hardware monitored by the facility control system. A common theme in this use case is a need for bidirectional data access with the MicroTCA Advanced Mezenine Cards (AMC) over a PCIe bus. The MTCA4U project and DeviceAccess library in its initial form was intended to provide a convenient, reusable way to achieve this purpose [1]. However, the scope of the project has shifted towards developing a more generic data access framework along with a set a components that aim at creating portable user applications across the different control systems choices. This change in scope is refected in renaming the MTCA4U project to ChimeraTK ("Control system and Hardware Interface with Mapped and Extensible Register-based device Abstraction Tool Kit") [2].

The DeviceAccess library, ControlSystemAdapter, ApplicationCore and VirtualLab are presently the main sub components of the ChimeraTK project. In addition to working with devices connected over the MicroTCA.4 PCIe backplane, DeviceAccess has expanded to accommodate data acquisition from any device for which a compatible library back end can be written. This lets users at DESY write applications that let them connect over Ethernet to devices like the TMCB boards [3] for data retrieval and control. The motivation for the ControlSystemAdapter was the requirement to port already existing Radio Frequency (RF) control applications at DESY for facilities adopting MicroTCA.4 based RF control. These facilities, however did not use the same control systems as used in DESY. Since existing code was tightly coupled with the control system middleware used at DESY, the effort required for porting it to a different control system middleware was intensive. A more generic solution was developing the ControlSystemAdapter framework [4] to let users write application code in a middleware independent manner. Currently the ControlSystemAdapter supports thread-safe, real-time capable user applications that can run on DOOCS [5], EPICS 3 [6] and OPC UA [7] middleware.

The ApplicationCore provides a unified and convenient way of using both DeviceAccess and ControlSystemAdapter frameworks together in the application code (Fig. 1). It lets users construct applications as a collection of modular components with interactions among each other while minimizing boiler plate code. The modular components make it easier for structural changes; the modules may be rearranged, interactions among them redefined or modules may be easily extracted into its own application.

The other major component of the toolkit is the VirtualLab framework that can be used for software testing during application development [8]. The framework is capable of creating mocks of actual hardware. This enables test automation and is useful for setting up a continuous integration support for the application software. The framework is also useful for simulating external faults at desired points of program execution. The Virtual lab framework was used extensively to create software tests for the low-level RF controller server for the ELBE accelerator at HZDR.

[*] geogin.varghese@desy.de

Figure 2 illustrates the components of ChimeraTK, which are described in more detail in the following sections.

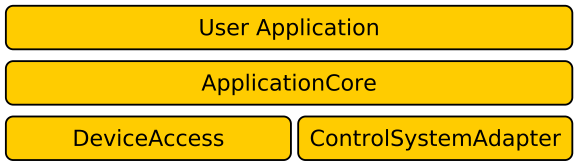

Figure 1: Structural overview of a user application written using ChimeraTK; user application can perform hardware access (using DeviceAccess) and may run as a DOOCS, EPICS 3 or OPC UA server (ControlSystemAdapter).

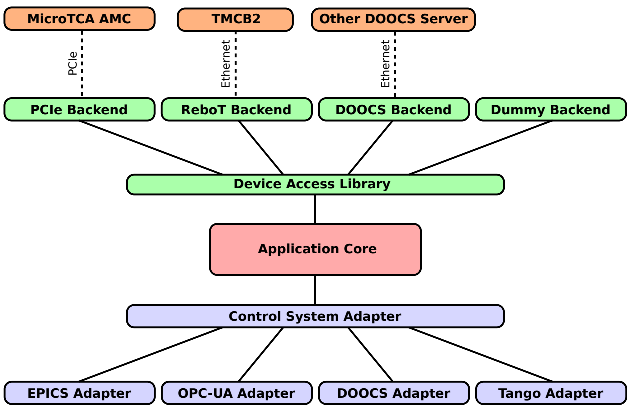

Figure 2: ChimeraTK: highlevel overview

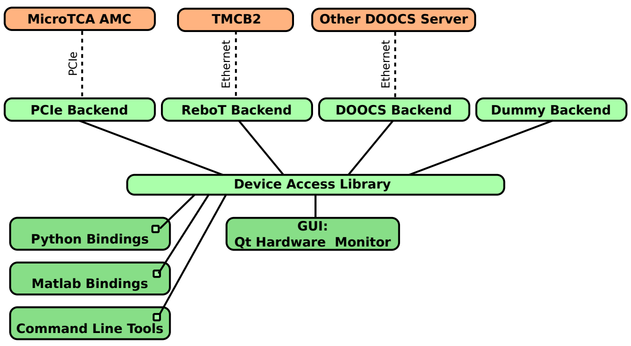

Figure 3: Components of the DeviceAccess library.

THE DEVICEACCESS LIBRARY

The DeviceAccess library provides a generic C++ interface that lets users access the I/O space of supported hardware. Figure 3 shows the subcomponents of the library. Generally, the connected device is expected to provide a register name mapping for its address space, either using an explicit mapping file or as a runtime list of registers (as is the case with DOOCS Backends). Using the Register Accessor object provided by the library, users can then refer to the contents of these registers in the device's I/O space. These accessor objects provide a convenient iterable container for the data in the accessed register, in addition to providing methods that read or write into the corresponding register address on the hardware.

Device Backends

Device Backends allow the library to support new devices. The library, by default supports a PCIe backend, ReboT Backend [3] and a DOOCS Backend which can be used to access the variables exposed by a DOOCS server. In addition a Dummy Backend is also provided; the primary use of this backend is to serve as a device mock for functional and unit testing of code. Any new device type for which the system designer can provide a device backend, can be registered with the framework using its plugin mechanism.

Logical Name Mapping

This feature of the framework lets users remap device registers into logical groups as required. This is useful for keeping conceptually connected components together when designing code. The logical name mapping for example, allows users to split up a multi element device register or group independent registers into logical registers. This may at times make for a more natural unit of grouping. Using this feature, it is also possible to group registers that span multiple hardware devices into a a single logical entity. The read and write methods of a logical register accessor object implicitly tracks the actual hardware addresses of each element of the logical register.

Language Bindings and Tools

The framework provides language bindings for Python and Matlab; these are helpful for scripting configuration actions on hardware and for algorithm development. The framework in addition bundles a Command line tool and a Qt based Hardware monitor which helps users with prototyping and testing their hardware.

CONTROL SYSTEM ADAPTER

The ControlSystemAdapter lets users write control applications that are decoupled from the underlying middleware [4]. The adapter is responsible for mapping middleware specific communication protocols, addressing schemes and other functionality needed for the user application to run. Currently the adapter supports DOOCS [5], EPICS 3 [6] and OPC UA [7] middlewares

Process Variables

Process variables are ControlSystemAdapter variables used by the user applications. The framework under the hood maps these process variables into a middleware specific data type. The process variables can represent numerical data types, strings or arrays of numerical types. To reduce complexity with thread safety and race conditions, these process variables are unidirectional. However, it is possible for the ControlSystemAdapter to modify an ingress process variable. This is useful for covering cases where the user input from the control system needs correction. The user application, in such cases can ask the ControlSystemAdapter to rectify the incorrect input by overriding the process variable with the correct value. Figure 4 illustrates the use of process variables in the user application 'Device library'

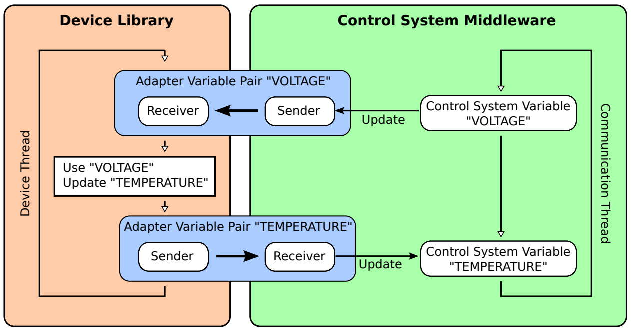
Device Library
Control System Middleware
Device Thread
Adapter Variable Pair "VOLTAGE"
Receiver
Sender
Update
Control System Variable "VOLTAGE"
Communication Thread
Use "VOLTAGE" Update "TEMPERATURE"
Adapter Variable Pair "TEMPERATURE"
Sender
Receiver
Update
Control System Variable "TEMPERATURE"

Figure 4: Process variables TEMPERATURE, VOLTAGE are mapped to the underlying control system by the ControlSystemAdapter framework. Data flow is unidirectional for these individual variables

Lock Free Queue Mechanism

In order to support real time user applications, the process variable mapping must not block on thread locking primitives. The framework uses a lock free mechanism to realize this requirement. A pool of preallocated buffers are used along with two queues - 'filled buffers' queue and the 'available buffers' queue. The sender and receiver part of the process variable are always provided with references to one of these buffers. An overview of the general mechanism is illustrated in Fig. 5. For a more detailed description on the implementation please see [4].

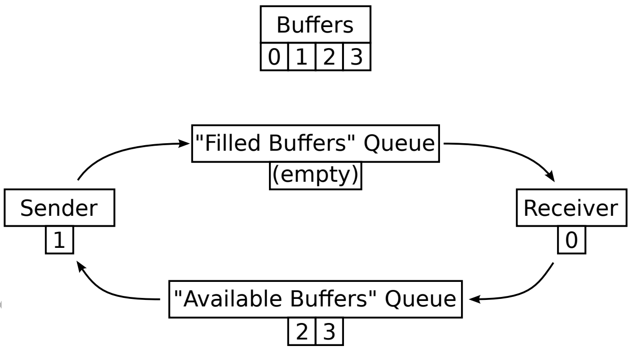

Figure 5: Lock free queue mechanism for data transfer from process variable to control system variable.

APPLICATION CORE

The ApplicationCore provides a convenient layer for integrating DeviceAccess and ControlSystemAdapter frameworks in user applications. It lets users model their applications as a set of modules with interactions defined among them. The framework keeps variable instantiation (ControlSystemAdapter and DeviceAccess variables) and application logic decoupled through the inversion of control pattern. Figure 6 illustrates the structure of a typical user application written using the ApplicationCore framework.

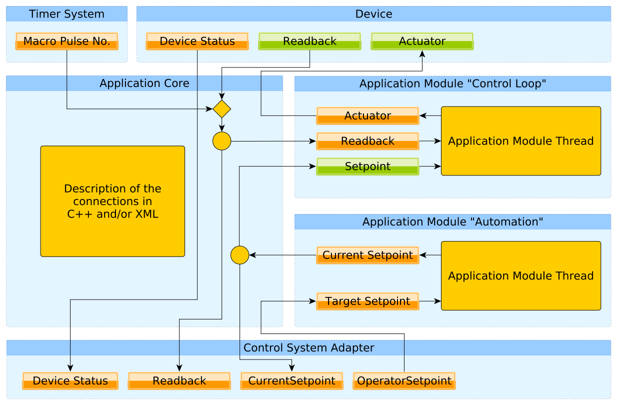

Figure 6: An example user application utilizing both ControlSystemAdapter and DeviceAccess realized using ApplicationCore

VIRTUAL LAB

The VirtualLab framework [8] can be used for functional testing of the user application. It allows users to test their software using functional mocks that replicate the minimum required behavior of the actual hardware, which is needed to test a component of the application software. This makes it possible to set up continuous integration systems that run the automated tests when changes are made to the software, thus helping verify that correctness of the user application is preserved.

Virtual Devices

Virtual Devices are the device mocks provided by the VirtualLab framework. The Virtual Devices can simulate desired functionality required for testing the user code. These can also be fully integrated with the DeviceAccess library, letting users utilize the frameworks API for access to the mock device. In addition the Virtual Devices can also be used to inject faults that test exception handling in the application program.

Virtual Time

Virtual time lets users create mocks that simulate behavior that is time dependent. The virtual time concept allows having actions or data generation on virtual devices depending on the point in time it was requested to do so. The feature can also be used for synchronizing test actions with program execution and also for simulating race condition events in order to test error handling.

CONCLUSION

ChimeraTK provides users with tools to create applications that are decoupled from specifics of hardware and middleware systems. This lets users write control logic that is portable, potentially saving effort in circumstances where the control application is needed in facilities that use a different control system middleware than the one it currently runs on. The DeviceAccess library lets user applications perform data acquisition from hardware through a generic API interface. The ControlSystemAdapter lets the user application define and use process variables that maps to variables in the underlying middleware being used. Currently, this means applications using the adapter can run on DOOCS [5], EPICS 3 [6] and OPC UA [7] middlewares. The ApplicationCore provides a convenient layer that glues together features of the DeviceAccess and ControlSystemAdapter libraries to create modular user applications. The VirtualLab framework lets users set up automated tests and continuous integration systems for the application code development.

REFERENCES

[1] N. Shehzad *et al.* “Modular Software for MicroTCA.4 Based Control Applications”, presented at the 20th Real Time Conference (IEEE RT2016).

[2] ChimeraTK Repository, `https://github.com/ChimeraTK`

[3] G. Varghese *et al.* “Implementing a ReboT server on a MicroBlaze”, presented at the 20th Real Time Conference (IEEE RT2016).

[4] M. Killenberg *et al.* “Integrating real-time control applications into different control systems”, 15th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS2015).

[5] ControlSystemAdapter DOOCS, `https://github.com/ChimeraTK/ControlSystemAdapter-DoocsAdapter`

[6] ControlSystemAdapter EPICS 3, `http://oss.aquenos.com/svnroot/epics-mtca4u/`.

[7] ControlSystemAdapter OPC UA, `https://github.com/ChimeraTK/ControlSystemAdapter-OPC-UA-Adapter`

[8] M. Hierholzer *et al.* “Software Tests and Simulations For Control Applications Based On Virtual Time”, 11th International Workshop on Personal Computers and Particle Accelerator Controls (PCaPAC’16).