Introduction to ChimeraTK.

Part 1: The DeviceAccess library and the ControlSystemAdapter

Martin Killenberg

M. Heuer, M. Hierholzer, T. Kozak, L. Petrosyan, C. Schmidt, N. Shehzad,
G. Varghese, M. Viti, *DESY, Hamburg, Germany*
S. Marsching, *aquenos GmbH, Baden-Baden, Germany*
A. Piotrowski, *FastLogic Sp. z o.o., Łódź, Poland*
C. P. Iatrou, J. Rahm, *Technische Universität Dresden, Dresden, Germany*
R. Steinbrück, M. Kuntzsch, *HZDR, Dresden-Rossendorf, Germany*
P. Prędki *Łódź University of Technology, Łódź, Poland*
A. Dworzanski, K. Czuba, *Warsaw University of Technology, Warsaw, Poland*

5. December 2017

Introduction

ChimeraTK

Control system and **H**ardware **I**nterface with **M**apped and **E**xtensible **R**egister-based device **A**bstraction **T**ool **K**it

1. **DeviceAccess**
   Register based access to (hardware) devices
2. **ControlSystemAdapter**
   Making application implementations independent from the middleware
3. **ApplicationCore** (talk by Martin Hierholzer)
   Improving abstraction of DeviceAccess and the ControlSystemAdapter, and allow more functionality

ChimeraTK

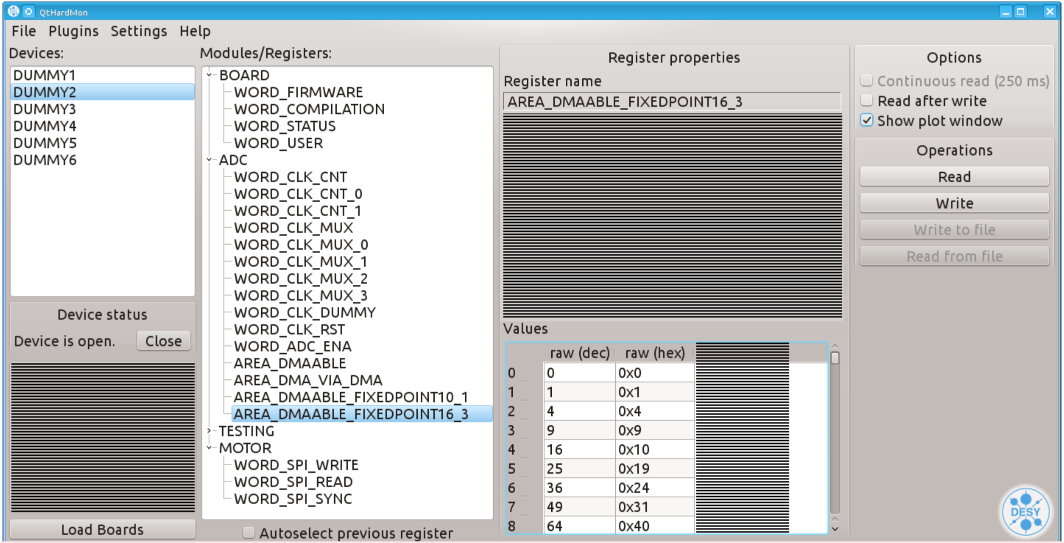
Control system and **H**ardware **I**nterface with **M**apped and
Extensible **R**egister-based device **A**bstraction **T**ool **K**it

1. **DeviceAccess**
   Register based access to (hardware) devices
2. **ControlSystemAdapter**
   Making application implementations independent from the middleware
3. **ApplicationCore** (talk by Martin Hierholzer)
   Improving abstraction of DeviceAccess and the ControlSystemAdapter, and allow more functionality

Registers

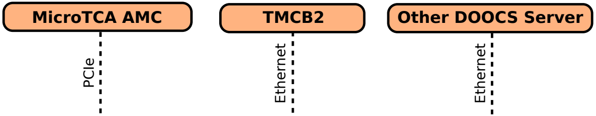
A register

- contains data (numerical or a string)
- is identified by a name
- lives on a device
- has a length ($1 \hat{=}$ scalar, $> 1 \hat{=}$ array)

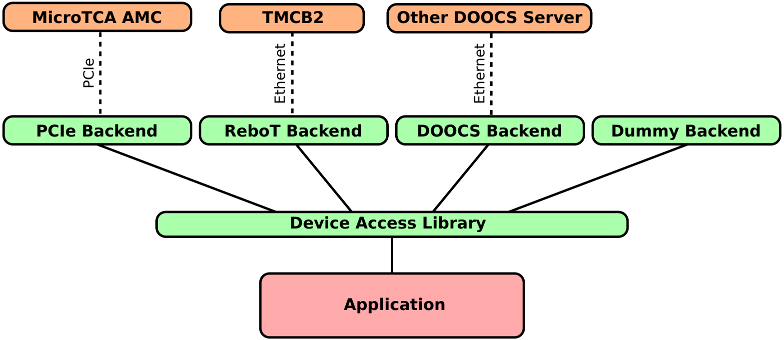

QtHardMon
File Plugins Settings Help
Devices:
DUMMY1
DUMMY2
DUMMY3
DUMMY4
DUMMY5
DUMMY6
Device status
Device is open.
Close
Load Boards
Modules/Registers:
BOARD
WORD_FIRMWARE
WORD_COMPILATION
WORD_STATUS
WORD_USER
ADC
WORD_CLK_CNT
WORD_CLK_CNT_0
WORD_CLK_CNT_1
WORD_CLK_MUX
WORD_CLK_MUX_0
WORD_CLK_MUX_1
WORD_CLK_MUX_2
WORD_CLK_MUX_3
WORD_CLK_DUMMY
WORD_CLK_RST
WORD_ADC_ENA
AREA_DMAABLE
AREA_DMA_VIA_DMA
AREA_DMAABLE_FIXEDPOINT10_1
AREA_DMAABLE_FIXEDPOINT16_3
TESTING
MOTOR
WORD_SPI_WRITE
WORD_SPI_READ
WORD_SPI_SYNC
Autoselect previous register
Register properties
Register name
AREA_DMAABLE_FIXEDPOINT16_3
Values
raw (dec)
raw (hex)
0 0 0x0
1 1 0x1
2 4 0x4
3 9 0x9
4 16 0x10
5 25 0x19
6 36 0x24
7 49 0x31
8 64 0x40
Options
Continuous read (250 ms)
Read after write
Show plot window
Operations
Read
Write
Write to file
Read from file
DESY

Live Demo (1)

Typical Devices



Typical Devices



DeviceAccess: Register Name Mapping

- DeviceAccess identifies registers by name
- PCI Express identifies registers by address in a "Base Address Range" (BAR)

⇒ We need a mapping

Example map file

```
#name                          n_words  address  n_bytes  BAR
heater.heatingCurrent                1     1024        4    2
heater.temperatureReadback           1     1028        4    2
heater.supplyVoltages                4     1032       16    2
```

- Map files are automatically created by the DESY (MSK) firmware framework
- Can easily be written manually

DeviceAccess: Register Accessors in C++

```
#include <mtca4u/Device.h>
#include <iostream>

int main(){

  mtca4u::Device d;
  d.open("sdm://./pci:pciedevs6=oven.map");









}
```

Note: ChimeraTK was previously called MicroTCA.4 User Tool Kit (MTCA4U)

```
#include <mtca4u/Device.h>
#include <iostream>

int main(){

  mtca4u::Device d;
  d.open("sdm://./pci:pciedevs6=oven.map");

  auto heatingCurrent
    = d.getScalarRegisterAccessor<int>("heater/heatingCurrent");






}
```

Note: ChimeraTK was previously called MicroTCA.4 User Tool Kit (MTCA4U)

DeviceAccess: Register Accessors in C++

```cpp
#include <mtca4u/Device.h>
#include <iostream>

int main(){

  mtca4u::Device d;
  d.open("sdm://./pci:pciedevs6=oven.map");

  auto heatingCurrent
    = d.getScalarRegisterAccessor<int>("heater/heatingCurrent");

  heatingCurrent.read();
  std::cout << "Heating current is " << heatingCurrent << std::endl;



}
```

Note: ChimeraTK was previously called MicroTCA.4 User Tool Kit (MTCA4U)

DeviceAccess: Register Accessors in C++

```
#include <mtca4u/Device.h>
#include <iostream>

int main(){

  mtca4u::Device d;
  d.open("sdm://./pci:pciedevs6=oven.map");

  auto heatingCurrent
    = d.getScalarRegisterAccessor<int>("heater/heatingCurrent");

  heatingCurrent.read();
  std::cout << "Heating current is " << heatingCurrent << std::endl;

  heatingCurrent += 3;
  heatingCurrent.write();

}
```

Note: ChimeraTK was previously called MicroTCA.4 User Tool Kit (MTCA4U)

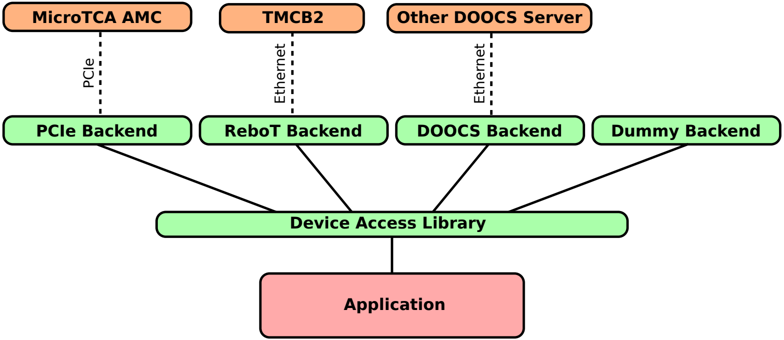
DeviceAccess: Register Accessors in C++

```cpp
#include <mtca4u/Device.h>
#include <iostream>

int main(){

  mtca4u::Device d;
  d.open("sdm://./pci:pciedevs6=oven.map");

  auto heatingCurrent
    = d.getScalarRegisterAccessor<int>("heater/heatingCurrent");

  heatingCurrent.read();
  std::cout << "Heating current is " << heatingCurrent << std::endl;

  heatingCurrent += 3;
  heatingCurrent.write();

}
```

Live Demo (2)

Note: ChimeraTK was previously called MicroTCA.4 User Tool Kit (MTCA4U)

The DeviceAccess Library



More abstraction: Identify devices by an alias name

Example device map file

```
#alias_name    URI                       map_file
oven           sdm://./pci:pciedevs6     oven.map
#oven          sdm://./dummy=oven.map    oven.map
```

- Client code identifies devices by functional name
- Actual implementation can be changed at run time

Accessing a register in C++

```
#include <mtca4u/Device.h>
#include <mtca4u/Utilities.h>
#include <iostream>

int main(){

  mtca4u::setDMapFilePath("devices.dmap");
  mtca4u::Device d;
  d.open("oven");

  auto heatingCurrent
    = d.getScalarRegisterAccessor<int>("heater/heatingCurrent");

  heatingCurrent.read();
  std::cout << "Heating current is " << heatingCurrent << std::endl;

  heatingCurrent += 3;
  heatingCurrent.write();


}
```

Live Demo (3)

Fixed Point Conversion

- Firmware often uses fixed-point arithmetic
- CPU uses floating point
- Transport layer (PCI Express) uses 32 bit words

⇒ Extend the mapping with conversion information*

Example map file

```
#name                       n_words  address  n_bytes  BAR  n_bits  n_fractionalBits  signed
heater.heatingCurrent             1      102        4    2      32                 0       0
heater.temperatureReadback        1      102        4    2      16                 3       1
heater.supplyVoltages             4      103       16    2      32                 0       0
```

* Optional, default conversion is 32 bit signed integer, no fractional bits

DeviceAccess: Float Accessor

```cpp
#include <mtca4u/Device.h>
#include <mtca4u/Utilities.h>
#include <iostream>

int main(){

  mtca4u::setDMapFilePath("devices.dmap");
  mtca4u::Device d;
  d.open("oven");

  auto temperature
    = d.getScalarRegisterAccessor<float>("heater/temperatureReadback");

  temperature.read();
  std::cout << "Readback temperature is " << temperature << std::endl;




}
```

Live Demo (4): C++ and QtHardMon

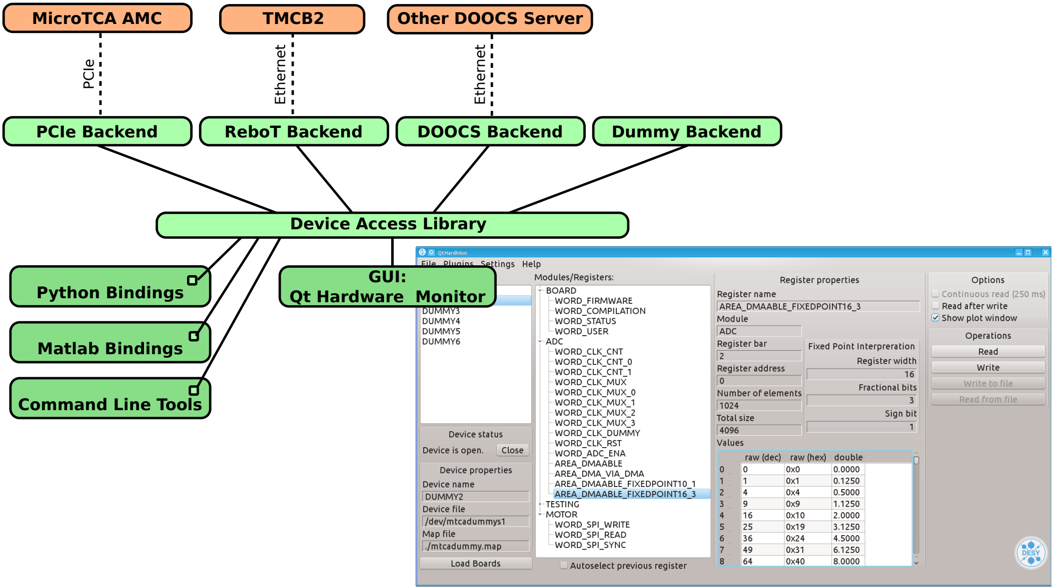
```cpp
#include <mtca4u/Device.h>
#include <mtca4u/Utilities.h>
#include <iostream>

int main(){

  mtca4u::setDMapFilePath("devices.dmap");
  mtca4u::Device d;
  d.open("oven");

  auto supplyVoltages
    = d.getOneDRegisterAccessor<int>("heater/supplyVoltages");

  supplyVoltages.read();

  std::cout << "Supply voltages are ";
  for (size_t i = 0; i < supplyVoltages.getNElements(); ++i){
    std::cout << supplyVoltages[i] << " ";
  }
  std::cout << std::endl;
}
```

DeviceAccess: 1D Accessors

```
#include <mtca4u/Device.h>
#include <mtca4u/Utilities.h>
#include <iostream>

int main(){

  mtca4u::setDMapFilePath("devices.dmap");
  mtca4u::Device d;
  d.open("oven");

  auto supplyVoltages
    = d.getOneDRegisterAccessor<int>("heater/supplyVoltages");

  supplyVoltages.read();

  std::cout << "Supply voltages are ";
  for (auto voltage : supplyVoltages){
    std::cout << voltage << " ";
  }
  std::cout << std::endl;
}
```

Live Demo (5)

The DeviceAccess Library Tools



Python bindings

C++

```cpp
#include <mtca4u/Device.h>
#include <mtca4u/Utilities.h>

int main(){
  mtca4u::setDMapFilePath("devices.dmap");
  mtca4u::Device d;
  d.open("oven");

  //"inefficient" shortcut to read a variable
  int temperature = d.read<float>("heater/temperatureReadback");
}
```

Python

```python
import mtca4u

mtca4u.set_dmap_location('devices.dmap')
d = mtca4u.Device('oven')

temperature = d.read('heater','temperatureReadback')
```

Matlab and Command line

Matlab

```
mtca4u.setDMapFilePath('devices.dmap')
d = mtca4u('oven')

temperature = d.read('heater','temperatureReadback')
```

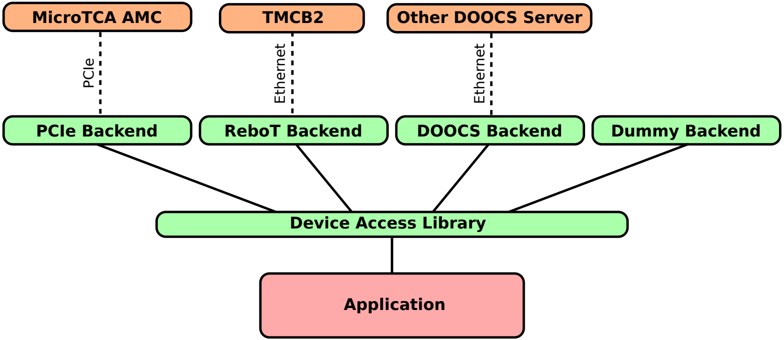
Command line

```
$ mtca4u read oven heater temperatureReadback
```

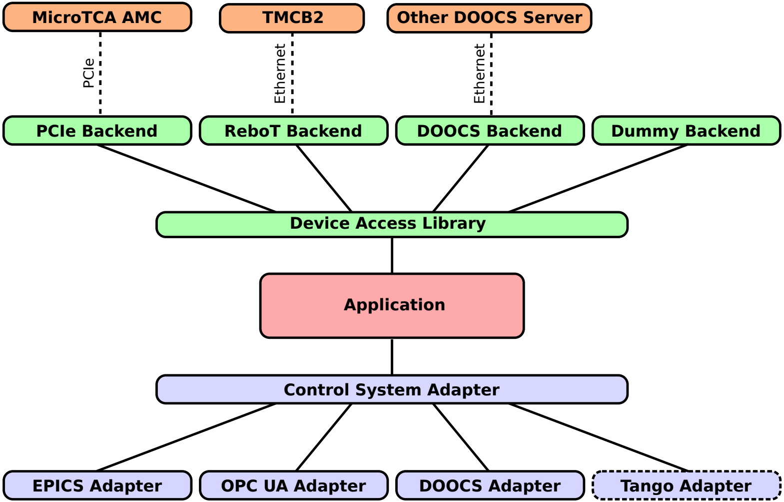
- All needed arguments in one call
- Takes the first dmap-file it finds :-O

Live Demo (6)

The ControlSystemAdapter



The ControlSystemAdapter



Typical Scenario: Integrating a small device

Integrate the device into your EPICS environment

- Just a few Process Variables

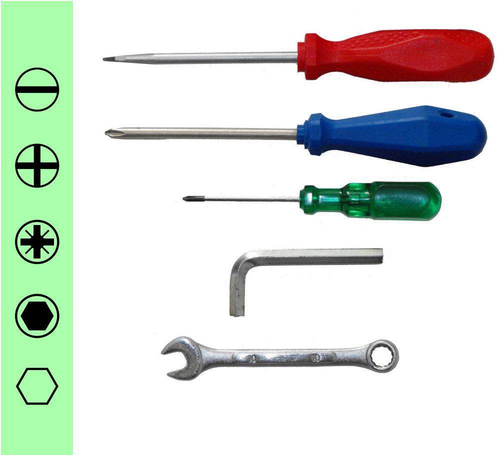
⇒ Write a new EPICS IOC, not too much work...

Integrate the same device into a DOOCS environment

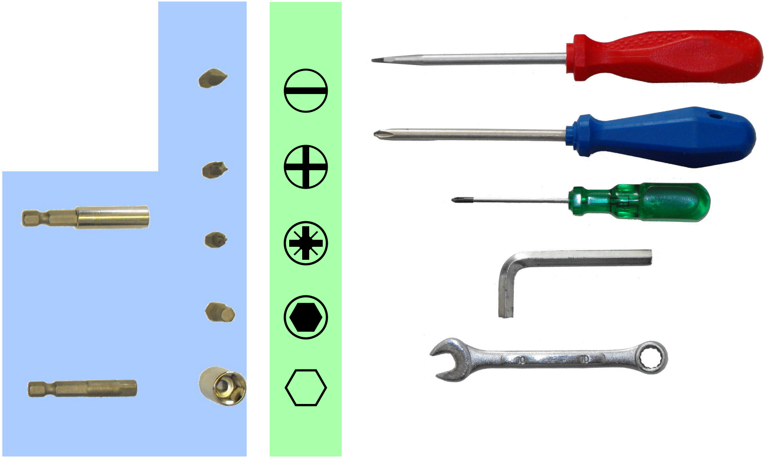
- Just a few Process Variables
- DOOCS and EPICS are very different, not much code to reuse: Better start from scratch

⇒ Write a new DOOCS device server, not too much work...

Sometimes You Need An Adapter

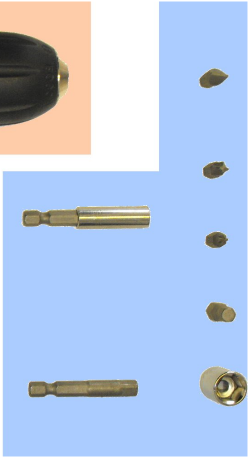


Sometimes You Need An Adapter



Sometimes You Need An Adapter

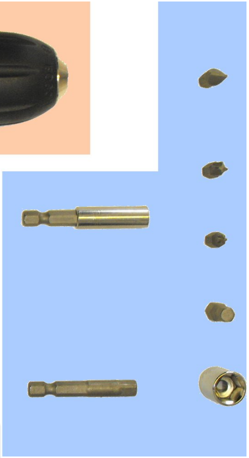
Device



Adapter

Control System

Sometimes You Need An Adapter



Adapter

Control System

EXAMPLE: LLRF Server

- $\mathcal{O}(400)$ process variables
- iterative learning algorithm
- feed forward table calculation

EXAMPLE: Target Control Systems

- DOOCS at FLASH,XFEL/DESY
- EPICS 3 at FLUTE/KIT
- WinCC/OPC UA at ELBE/HZDR
- EPICS 4 at TARLA

ControlSystemAdapter

Task

Complex control algorithms should be used with different control systems.

ControlSystemAdapter

Task

Complex control algorithms should be used with different control systems.

Requirements For Abstraction

- Keep application code control system independent
- The algorithm must interact with the control system
- Use functionality provided by the control system
- No device-dependent code on the control system side

Additional Requirements:

- Thread-safe
- Lock-free
- Must not copy large data objects (arrays)

ControlSystemAdapter

Task

Complex control algorithms should be used with different control systems.

Requirements For Abstraction

- Keep application code control system independent
- The algorithm must interact with the control system
- Use functionality provided by the control system
- No device-dependent code on the control system side

Additional Requirements:

- Thread-safe
- Lock-free
- Must not copy large data objects (arrays)

First Implementation

- Uni-directional process variables to transfer data to/from the control system

Slide by Sebastian Marsching on the 2015 Matter and Technology meeting

Comparison of Control Systems

Control System	Device Description	Device Model	Mutex
DOOCS	code based	object oriented	per group
EPICS	configuration based	channel based	per PV
TANGO	code based	object oriented	?
WinCC OA	configuration based	channel based	no (single threaded)

Plus different handling of

- limits
- alarms
- engineering units
- etc.

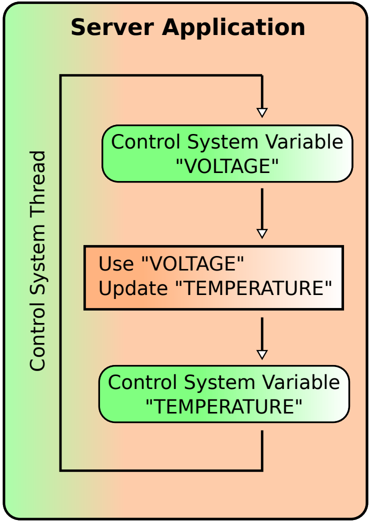
9 MTCA4U Control System Adapter 24.02.2015 aquenos

Completely different locking schemes

- Locking cannot work

$\Rightarrow$ We need a lock-free implementation!

A Typical Device Server (Without Adapter)



- Control system data types used inside the algorithm
- Control system variables can be locking/blocking
- Control system variables might not be thread safe
- Threading often handled by control system

Abstraction is needed

Required abstraction for the ControlSystemAdapter:
Separate device logic and control system integration

Application code

- Define process variables
- Implement algorithms
- Talk to hardware

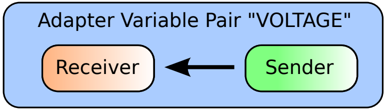
Control system "code"

- Publish process variables via middleware
- Define variable name visible in control system
- Define middleware dependent features/data types
  - Histories
  - Display properties
- Application independent, configured via config file

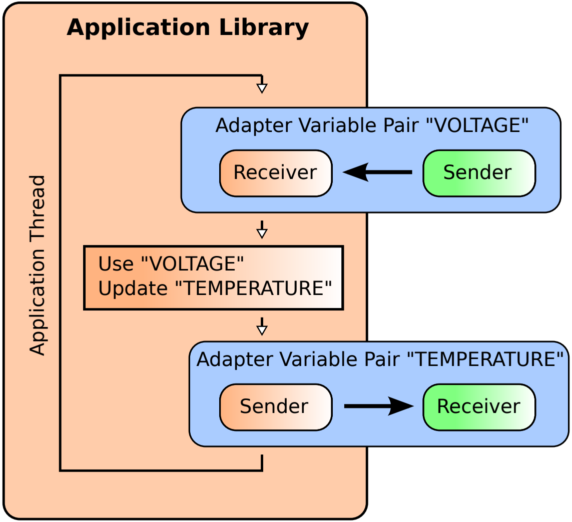
Application and control system dependent code

- Avoid it!

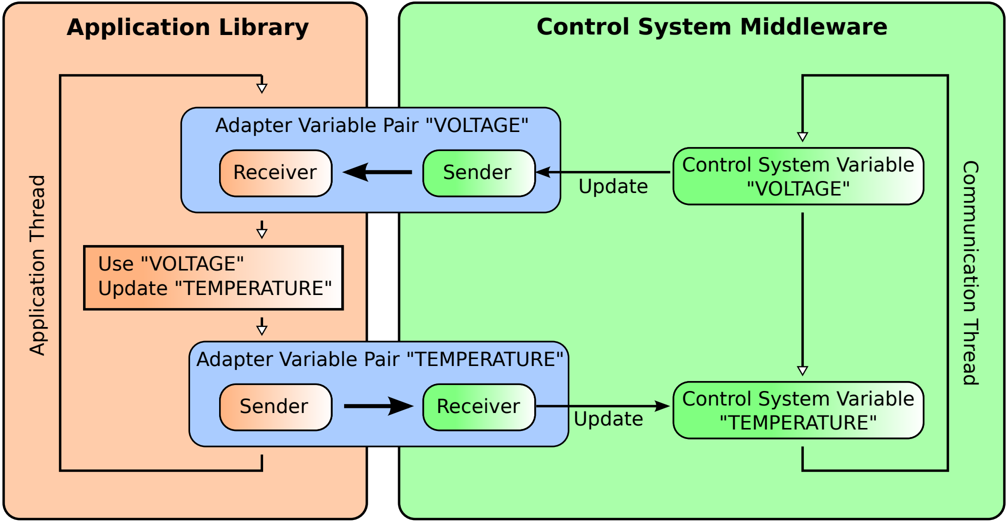
A Device Using The ControlSystemAdapter



A Device Using The ControlSystemAdapter

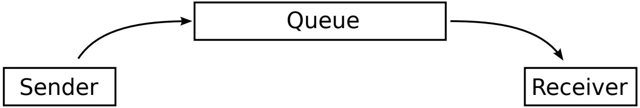


A Device Using The ControlSystemAdapter



Implementation Of The Sender/Receiver Pair

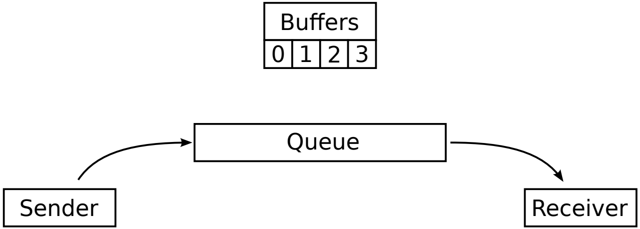
Use a queue: Allows processing a sequence of data and update notifications



- Lock-free queue

Implementation Of The Sender/Receiver Pair

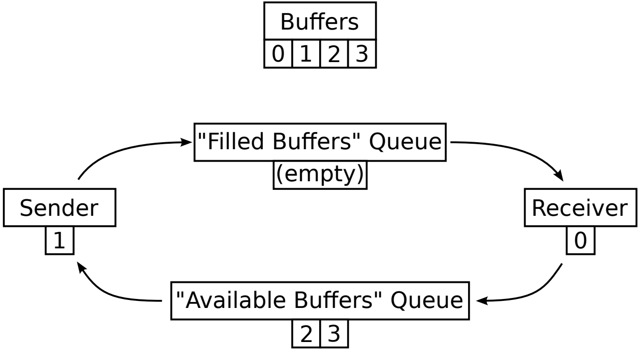
Use a queue: Allows processing a sequence of data and update notifications



- Lock-free queue
- Pre-allocated buffers for arrays

Implementation Of The Sender/Receiver Pair

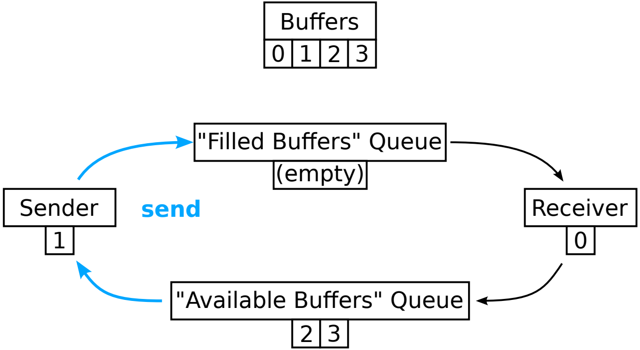
Use a queue: Allows processing a sequence of data and update notifications



- Lock-free queues
- Pre-allocated buffers for arrays
- Copy references, not buffers

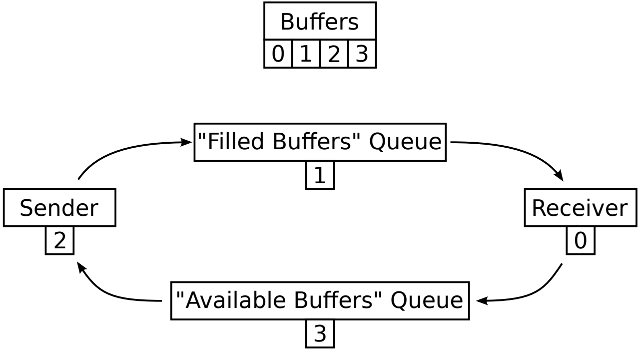
Implementation Of The Sender/Receiver Pair

Use a queue: Allows processing a sequence of data and update notifications



- Lock-free queues
- Pre-allocated buffers for arrays
- Copy references, not buffers

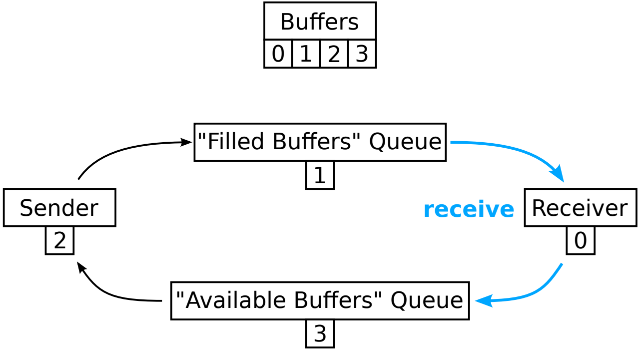
Use a queue: Allows processing a sequence of data and update notifications



- Lock-free queues
- Pre-allocated buffers for arrays
- Copy references, not buffers

Implementation Of The Sender/Receiver Pair

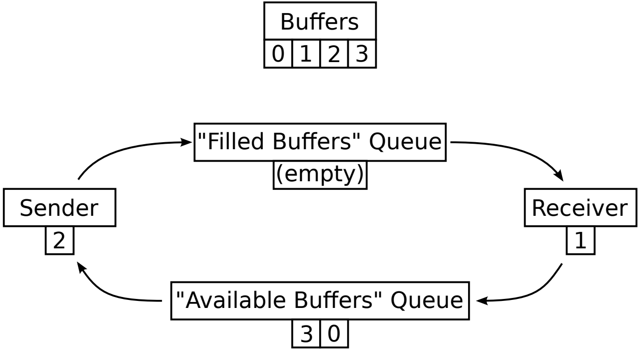
Use a queue: Allows processing a sequence of data and update notifications



- Lock-free queues
- Pre-allocated buffers for arrays
- Copy references, not buffers

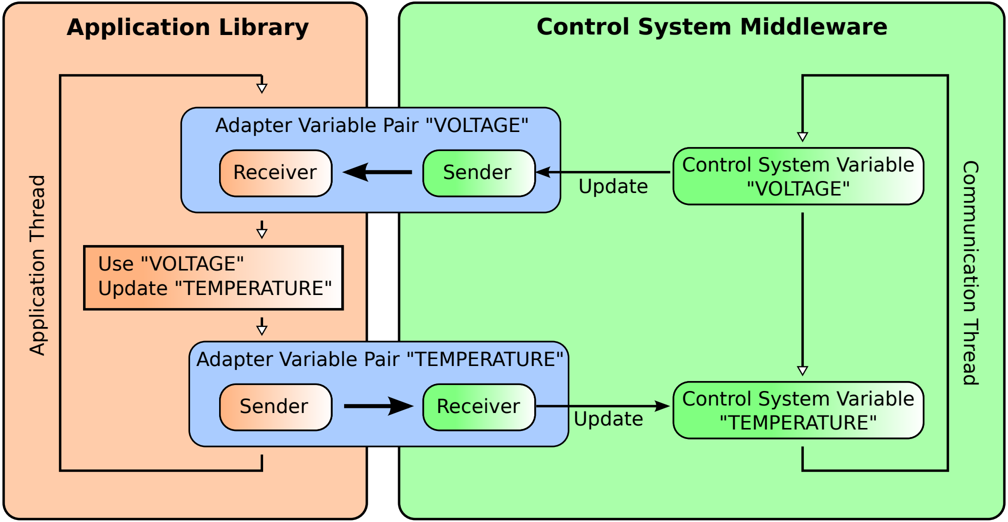
Implementation Of The Sender/Receiver Pair

Use a queue: Allows processing a sequence of data and update notifications

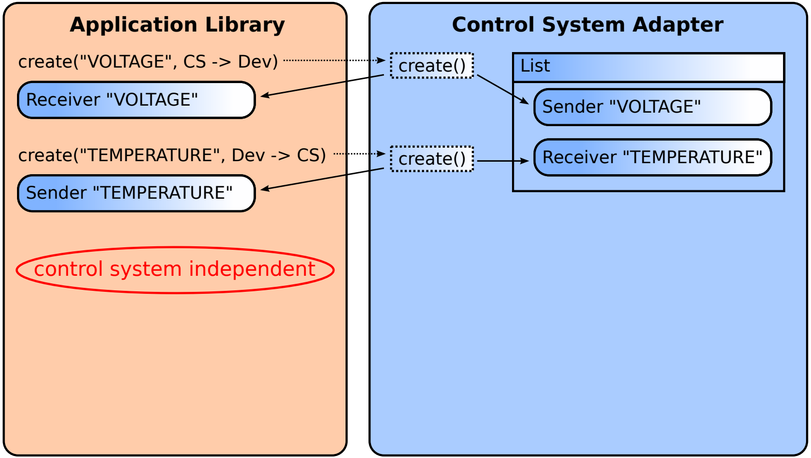


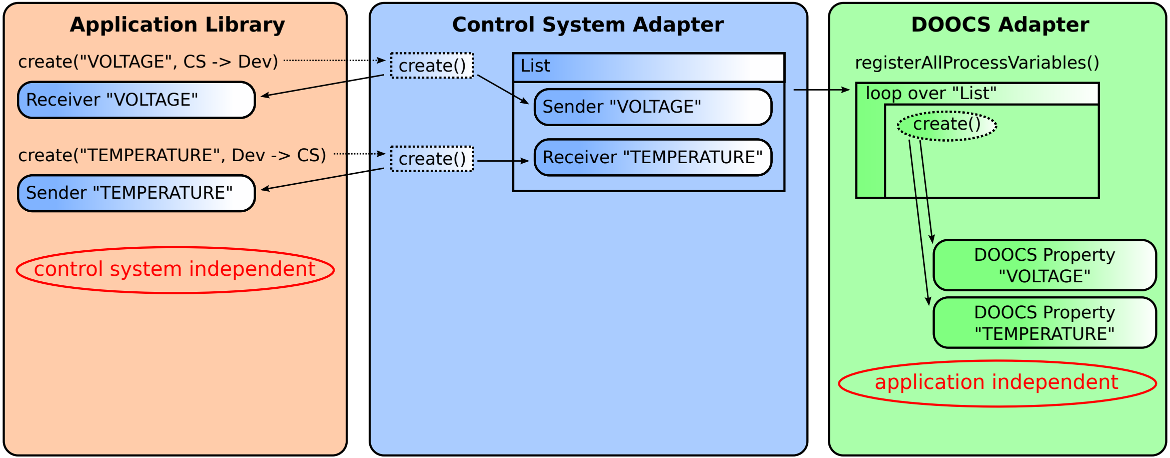
- Lock-free queues
- Pre-allocated buffers for arrays
- Copy references, not buffers

A Device Using The ControlSystemAdapter



Registering Process Variables




Application Library
create("VOLTAGE", CS -> Dev)
Receiver "VOLTAGE"
create("TEMPERATURE", Dev -> CS)
Sender "TEMPERATURE"
control system independent
Control System Adapter
create()
create()
List
Sender "VOLTAGE"
Receiver "TEMPERATURE"
DOOCS Adapter
registerAllProcessVariables()
loop over "List"
create()
DOOCS Property "VOLTAGE"
DOOCS Property "TEMPERATURE"
application independent

Porting an Application

How many lines of C++ code do I need to integrate an existing application into my control system (e.g. DOOCS)?

How many lines of C++ code do I need to integrate an existing application into my control system (e.g. DOOCS)?

Real code from the LLRF server

Porting an Application

How many lines of C++ code do I need to integrate an existing application into my control system (e.g. DOOCS)?

Real code from the LLRF server

0 lines of code are needed. Just link it!

```
$ ld myApp.o -l ChimeraTK-ControlSystemAdapter-DoocsAdapter -o myAppDoocsServer
```

(You might need config files, or at least they improve the system integration.)

Now it's time to write an application!

Control System Integration Example 1: Variable Names

Input tree

```
|-- oven1
|    |-- controller
|    |    |-- temperatureSetpoint
|    |    |-- temperatureReadback
|    |
|    |-- supplyVoltages
|
|-- oven2
     |-- controller
     |    |-- temperatureSetpoint
     |    |-- temperatureReadback
     |
     |-- supplyVoltages
```

System Integration

Control System Integration Example 1: Variable Names

Input tree

```
|-- oven1
|    |-- controller
|    |    |-- temperatureSetpoint
|    |    |-- temperatureReadback
|    |
|    |-- supplyVoltages
|
|-- oven2
     |-- controller
     |    |-- temperatureSetpoint
     |    |-- temperatureReadback
     |
     |-- supplyVoltages
```

Output tree

```
|-- breadOven
|    |-- temperature
|
|-- cookieOven
     |-- temperature
```

System Integration

- Names need to be adapted for the facility (manufacturer does not know if oven is used for bread or cookies)

⇒ Do it in system integration

Control System Integration Example 1: Variable Names

Input tree

```
|-- oven1
|    |-- controller
|    |    |-- temperatureSetpoint
|    |    |-- temperatureReadback
|    |
|    |-- supplyVoltages
|
|-- oven2
     |-- controller
     |    |-- temperatureSetpoint
     |    |-- temperatureReadback
     |
     |-- supplyVoltages
```

Output tree

```
|-- breadOven
|    |-- temperature
|
|-- cookieOven
|    |-- temperature
|
|-- expert
     |-- temperatureSetpoints
          |-- breadOven
          |-- cookieOven
```

System Integration

- Names need to be adapted for the facility (manufacturer does not know if oven is used for bread or cookies)

⇒ Do it in system integration

- Naming depends on the middleware (e.g. DOOCS only has two hierarchy levels per server)

⇒ Has to be in the middleware-specific part of the adapter

Control System Integration Example 1: Variable Names

Input tree

```
|-- oven1
|    |-- controller
|    |    |-- temperatureSetpoint
|    |    |-- temperatureReadback
|    |
|    |-- supplyVoltages
|
|-- oven2
     |-- controller
     |    |-- temperatureSetpoint
     |    |-- temperatureReadback
     |
     |-- supplyVoltages
```

Output tree

```
|-- breadOven
|    |-- temperature
|
|-- cookieOven
|    |-- temperature
|
|-- expert
|    |-- temperatureSetpoints
|         |-- breadOven
|         |-- cookieOven
|
|-- powerSupply
     |-- fuse1
     |    |-- breadOvenVoltages
     |
     |-- fuse2
          |-- cookieOvenVoltages
```

System Integration

- Names need to be adapted for the facility (manufacturer does not know if oven is used for bread or cookies)

⇒ Do it in system integration

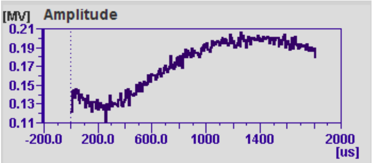
- Naming depends on the middleware (e.g. DOOCS only has two hierarchy levels per server)

⇒ Has to be in the middleware-specific part of the adapter

Control System Integration Example 2: Complex Data Types

D_spectrum: Aggregated DOOCS data type for plotting

- Main data: 1D array
- Meta data:
  - Engineering units
  - X-Axis scaling
  - . . .



- D_spectrum only known in DOOCS

⇒ Can only be configured during system integration

- Application is publishing main data and meta data(*) (example):
  - Amplitude with EGUs
  - X-axis start with EGUs
  - X-axis step width

XML code

```
<D_spectrum source="Amplitude">
   <startSource="Ampl_x_offset"/>
   <incrementSource="Ampl_x_step"/>
</D_spectrum>
```

Note: With the OPC UA adapter the published meta data is used in the panel to create the plot.

(*) Meta data can also be hard-coded in the XML config

ControlSystemAdapter Summary

Adapter for Process Variables

Decouple application logic and control system

- Generic part
- Control system specific part
  - Implementations for DOOCS and OPC UA
  - EPICS 3 adapter currently being updated

Design Goals

- Control system independent process variables ✓
- Thread safe ✓
- Lock free ✓
- Minimise copying ✓
- No device-dependent code on control system side ✓

Tools for System Integration

- Name mapping for Process Variables ✓
  (should be available in every adapter impl.)
- Save and restore settings ✓
  (default implementation in ControlSystemAdapter)
- Access to control system features:
  Availability depends on the middleware
  - Display limits
  - Engineering units ✓
  - History
  - Handle alarms

Tools for writing virtual devices, functional mocks and plant models

Virtual Timing

- Run test faster than the real time
- Simulation takes longer than real time $\rightarrow$ run application synchronously
- Test race conditions, check error handling

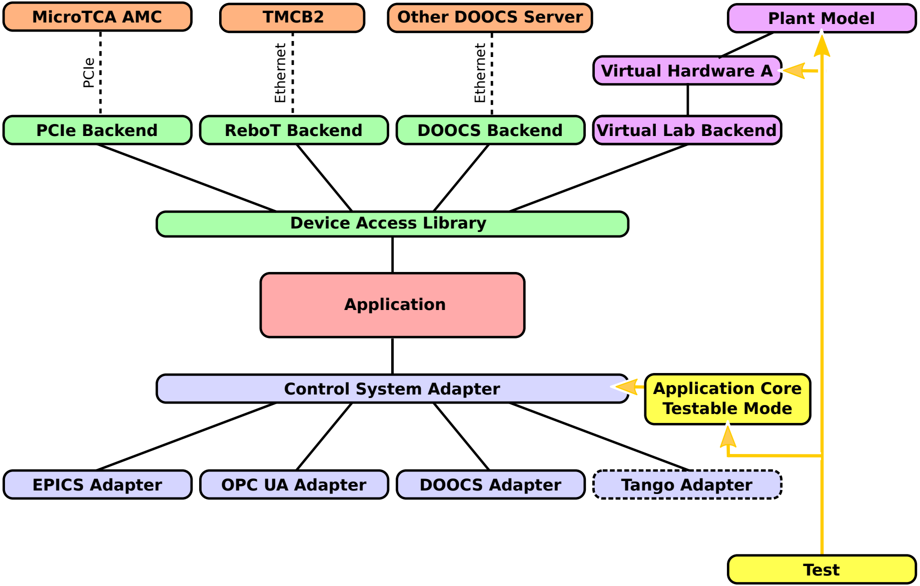
State Machine

- React on read/write to the device
- Easy implementation of firmware functionality

Signal Sinks/Sources

- Connect devices and plant models
- Modular plant models
- Planned: Share the same plant model across different applications

ChimeraTK Overview



Introduction to ChimeraTK: Summary

ControlSystemAdapter

- Use device logic with different control systems
- Implementations for DOOCS and OPC UA
- Epics 3 adapter is currently being updated

DeviceAccess Library

- Abstracted, register based hardware access
- Use real and virtual hardware, device servers
- Scripting tools and GUI

ApplicationCore Library

- Unifies DeviceAccess and ControlSystemAdapter
- Application modules
  - Input/output variables
- Hierarchical data model
  - Module and variable groups
  - Tags
- High abstraction level
  - Improves readability and reliability
  - Good maintainability

Software Repositories

All software is published under the GNU GPL or the GNU LGPL.

- ChimeraTK: https://github.com/ChimeraTK
- EPICS 3 Adapter: http://oss.aquenos.com/svnroot/epics-mtca4u/

Source code for the live demos: https://github.com/killenb/DeviceAccess_live_demo