ChimeraTK
Hardware Access and Control System Integration (not only) for MicroTCA

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

6th MicroTCA Workshop
DESY, Hamburg
7. December 2017
ChimeraTK
Control system and Hardware Interface with Mapped and Extensible Register-based device Abstraction Tool Kit

DeviceAccess Library
Last year’s presentation
- Register based devices
- Abstracted interface with backends
ChimeraTK

Control system and Hardware Interface with Mapped and Extensible Register-based device Abstraction Tool Kit

DeviceAccess Library

- Register based devices
- Abstracted interface with backends

ControlSystemAdapter

- Making application implementations independent from the middleware

ApplicationCore

- Unify concepts of DeviceAccess and ControlSystemAdapter
- Enhance functionality

Last year’s presentation

Martin Killenberg (DESY)
Complex control algorithms should be used with different control systems.

<table>
<thead>
<tr>
<th>LLRF server</th>
<th>Facilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{O}(400) ) process variables</td>
<td>DOOCS at XFEL/FLASH (DESY)</td>
</tr>
<tr>
<td>iterative learning algorithms</td>
<td>WinCC/OPC UA at ELBE (HZDR)</td>
</tr>
<tr>
<td>feed forward table calculations</td>
<td>EPICS 3 at FLUTE (KIT)</td>
</tr>
</tbody>
</table>

⇒ EPICS 4 at TARLA (Ankara)
Complex control algorithms should be used with different control systems.

**LLRF server**
- \(\mathcal{O}(400)\) process variables
- iterative learning algorithms
- feed forward table calculations

**Facilities**
- DOOCS at XFEL/FLASH (DESY)
- WinCC/OPC UA at ELBE (HZDR)
- EPICS 3 at FLUTE (KIT)
- EPICS 4 at TARLA (Ankara)

**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
A Typical Device Server (Without Adapter)

- Control system data types used inside the algorithm
- Control system variables might not be thread safe
- Control system variables can be locking/blocking
- 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

Adapter Variable Pair "VOLTAGE"

Sender
Receiver
A Device Using The ControlSystemAdapter

- Adapter Variable Pair "VOLTAGE"
  - Receiver
  - Sender

- Use "VOLTAGE"
  - Update "TEMPERATURE"

- Adapter Variable Pair "TEMPERATURE"
  - Sender
  - Receiver

Application Library
Application Thread
A Device Using The ControlSystemAdapter

Adapter Variable Pair "VOLTAGE"
Sender
Receiver
Control System Variable "VOLTAGE"
Update

Use "VOLTAGE"
Update "TEMPERATURE"

Adapter Variable Pair "TEMPERATURE"
Sender
Receiver
Control System Variable "TEMPERATURE"
Update

Application Library

Control System Middleware Application Library
Control System Variable "VOLTAGE"
Application Thread
Communication Thread

ChimeraTK 6
Martin Killenberg (DESY)
Description of connections in C++

Modules
- Input/output variables
- Application Modules
  - One thread per module
- Special modules
  - Device module
  - Control system module

Connection Code
- Connect application modules
- Triggering
  - Read multiple variables synchronously
  - Synchronise application modules to HW trigger
**Threading system** with efficient implementation:
Sender-receiver pairs based on lock-free queues and futures
- Message driven design: sender pushes data into the queue
- No dead-locks possible
- Threads are not unintentionally blocked
- Wait for data without using CPU time

API fosters designs with **small modules**
- Clean design
- Improved performance on multi-core CPUs
- High scalability

Use **C++11** to avoid boiler plate code where possible
Automatically generated graph for LLRF server, module AdcBoard, channel 5 (part)
Automatically generated graph for LLRF server, module AdcBoard, channel 5 (part)
Automatically generated graph for LLRF server, module AdcBoard, channel 5 (part)

- Module groups and variable groups: Add hierarchies
Automatically generated graph for LLRF server, module AdcBoard, channel 5 (part)

- Module groups and variable groups: Add hierarchies
- Eliminate hierarchies
Information modelling

Automatically generated graph for LLRF server, module AdcBoard, channel 5 (part)

- Module groups and variable groups: Add hierarchies
- Eliminate hierarchies
- Tags: Simplify the connection code

ModuleGroup → ApplicationModule → VariableGroup

Tags
Control System Integration Example: Variable Names

Input tree

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

System Integration

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

Configure middleware-dependent features (display limits, . . . )

⇒ Do it in system integration via config file
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)
- Configure middleware-dependent features (display limits, ...)

⇒ Do it in system integration via config file
Control System Integration Example: 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)
- Configure middleware-dependent features (display limits, ...)

⇒ Do it in system integration via config file
What about the LLRF server...?

- LLRF server developed using the DOOCS adapter at DESY
- Re-linked with the OPC UA adapter and integrated at ELBE/HZDR
  ⇒ Works without changing a single line of C++ code!
- Server and OPC UA adapter need some refinement for production system

Presentation:
“ChimeraTK OPC UA Adapter for the Integration of MicroTCA.4 based digital LLRF” by Reinhard Steinbrück

Poster:
“ApplicationCore: A Framework for Modern Control Applications at the Example of a Facility Independent LLRF Server” by Martin Hierholzer
### Introduction to ChimeraTK: Summary

<table>
<thead>
<tr>
<th>ControlSystemAdapter</th>
<th>DeviceAccess Library</th>
<th>ApplicationCore Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use device logic with different control systems</td>
<td>Abstracted, register based hardware access</td>
<td>Unifies DeviceAccess and ControlSystemAdapter</td>
</tr>
<tr>
<td>Implementations for DOOCS and OPC UA</td>
<td>Use real and virtual hardware, device servers</td>
<td>Application modules</td>
</tr>
<tr>
<td>Epics 3 adapter is currently being updated</td>
<td>Scripting tools and GUI</td>
<td>Input/output variables</td>
</tr>
</tbody>
</table>

#### Software Repositories

- ChimeraTK: [https://github.com/ChimeraTK](https://github.com/ChimeraTK)

#### Slides and Source Code from Tuesday’s Tutorial

- [https://github.com/mhier/ApplicationCoreLiveDemo](https://github.com/mhier/ApplicationCoreLiveDemo)