ChimeraTK: a control system independent software framework.

Martin Hierholzer

M. Killenberg, T. Kozak, L. Petrosyan, C. Schmidt, N. Shehzad,
G. Varghese, J. Georg, C. Kampmeyer, *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*
P. Prędki *Łódź University of Technology, Łódź, Poland*
A. Dworzanski, K. Czuba, *Warsaw University of Technology, Warsaw, Poland*

13. June 2018
4th annual MT meeting, Berlin

Overview on ChimeraTK

ControlSystemAdapter

Support multiple control systems without changing source code!

DOOCS OPC UA EPICS

ApplicationCore

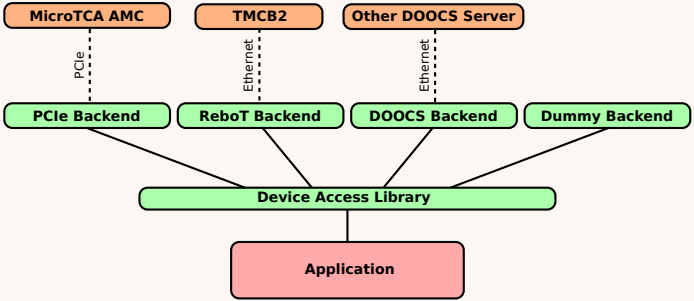
- Modular framework for control applications
- Unifies ControlSystemAdapter and DeviceAccess
- Helps writing applications which integrate well in different control systems

DeviceAccess

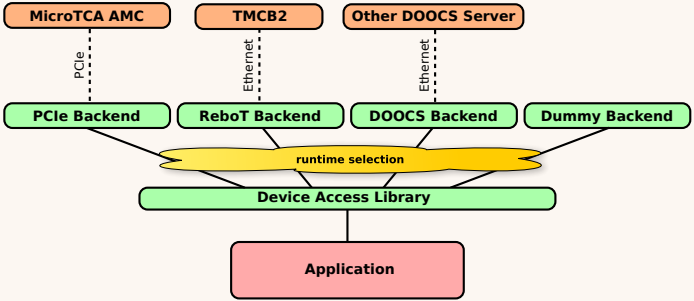
Access to hardware and other control system applications

PCI EXPRESS Ethernet DOOCS

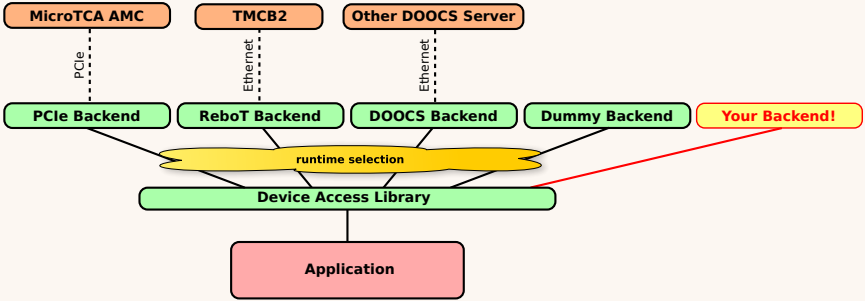
DeviceAccess



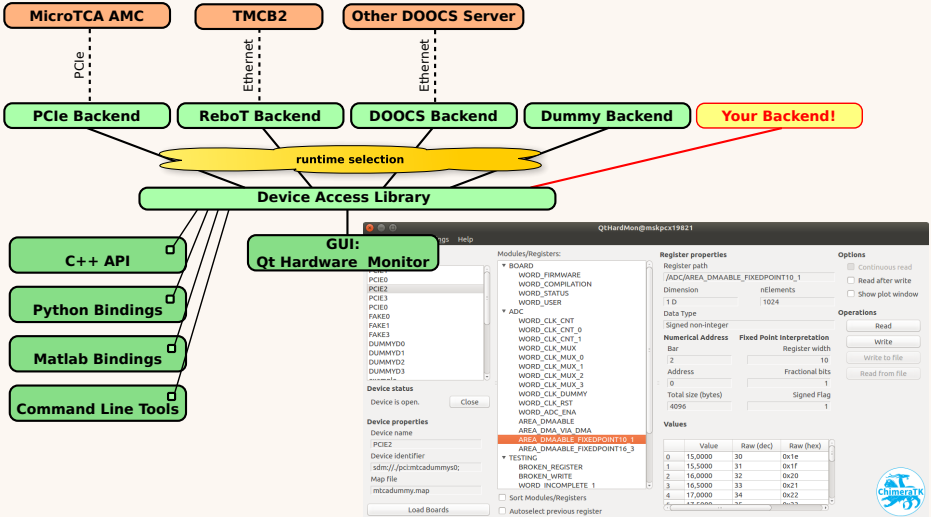
DeviceAccess



DeviceAccess



DeviceAccess



The DeviceAccess C++ API

Special emphasis on easy-to-use C++ API:

```
ChimeraTK::setDMapFilePath("devices.dmap");
ChimeraTK::Device dev;
dev.open("MyDevice");

auto current = dev.getScalarRegisterAccessor<double>("CURRENT");

current.read();
std::cout << current << std::endl;
current = 42;
current.write();
```

The DeviceAccess C++ API

Special emphasis on easy-to-use C++ API:

```
# Alias       Device URI                  MAP file (if needed)
MyDevice      sdm://./dummy=dummy.map     dummy.map
OtherDevice   sdm://./pci:pcieunis4       realDevice.map
```

```
ChimeraTK::setDMapFilePath("devices.dmap");
ChimeraTK::Device dev;
dev.open("MyDevice");

auto current = dev.getScalarRegisterAccessor<double>("CURRENT");

current.read();
std::cout << current << std::endl;
current = 42;
current.write();
```

The DeviceAccess C++ API

PCIe Backend | RebooT Backend | DOOCS Backend | Dummy Backend

Special emphasis on easy-to-use C++ API:

```
# Alias       Device URI                  MAP file (if needed)
MyDevice      sdm://./dummy=dummy.map     dummy.map
OtherDevice   sdm://./pci:pcieunis4       realDevice.map
```

```
ChimeraTK::setDMapFilePath("devices.dmap");
ChimeraTK::Device dev;
dev.open("MyDevice");

auto current = dev.getScalarRegisterAccessor<double>("CURRENT");

current.read();
std::cout << current << std::endl;
current = 42;
current.write();
```

The DeviceAccess C++ API

Special emphasis on easy-to-use C++ API:

```
# Alias       Device URI                  MAP file (if needed)
MyDevice      sdm://./dummy=dummy.map     dummy.map
OtherDevice   sdm://./pci:pcieunis4       realDevice.map
```

```
# name   nr of elements   address   size   bar
CURRENT  1                0         4      0
VOLTAGE  2                4         8      0
```

```
ChimeraTK::setDMapFilePath("devices.dmap");
ChimeraTK::Device dev;
dev.open("MyDevice");

auto current = dev.getScalarRegisterAccessor<double>("CURRENT");

current.read();
std::cout << current << std::endl;
current = 42;
current.write();
```

The DeviceAccess C++ API

Special emphasis on easy-to-use C++ API:

```
# Alias       Device URI                  MAP file (if needed)
MyDevice      sdm://./dummy=dummy.map     dummy.map
OtherDevice   sdm://./pci:pcieunis4       realDevice.map
```

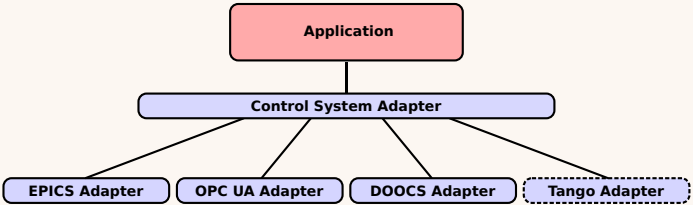
```
# name   nr of elements   address   size   bar
CURRENT  1                0         4      0
VOLTAGE  2                4         8      0
```

```
ChimeraTK::setDMapFilePath("devices.dmap");
ChimeraTK::Device dev;
dev.open("MyDevice");

auto current = dev.getScalarRegisterAccessor<double>("CURRENT");

current.read();
std::cout << current << std::endl;
current = 42;
current.write();
```

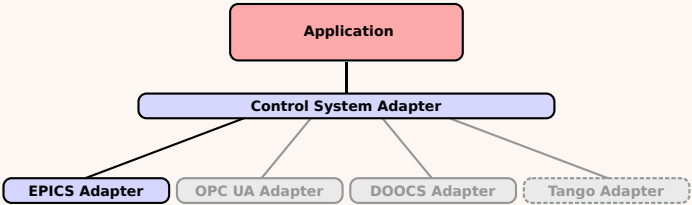
Abstraction

- make source code independent of hardware details
- Code is more readable and more generic

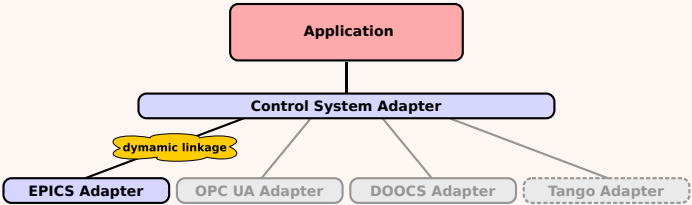
ControlSystemAdapter



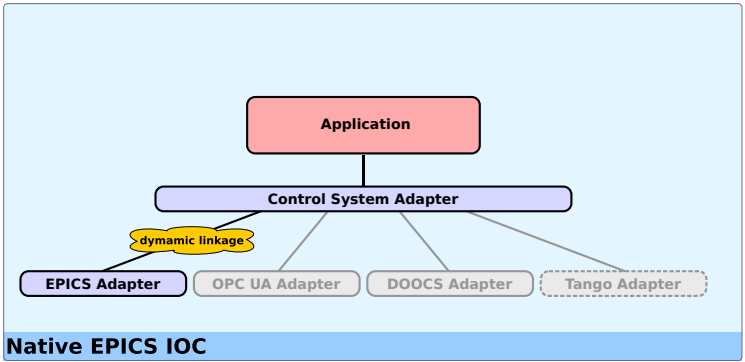
ControlSystemAdapter



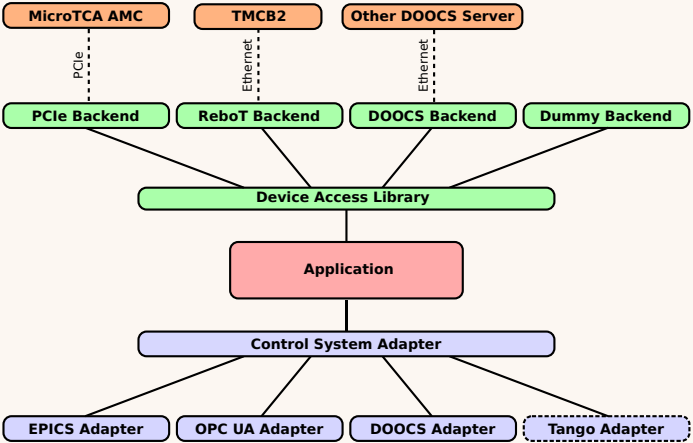
ControlSystemAdapter



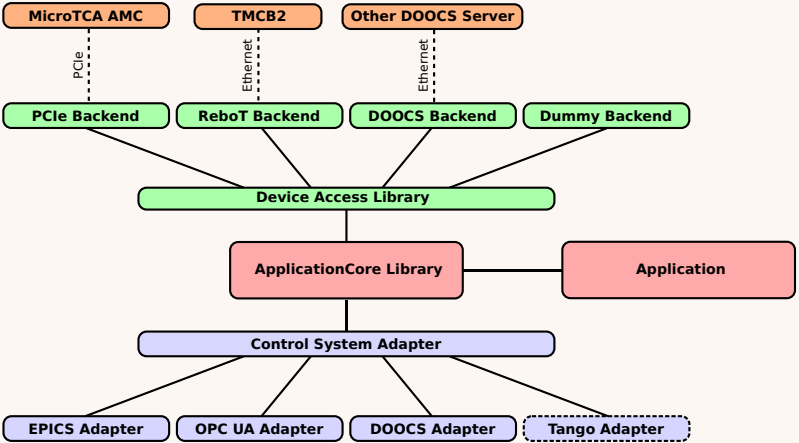
ControlSystemAdapter



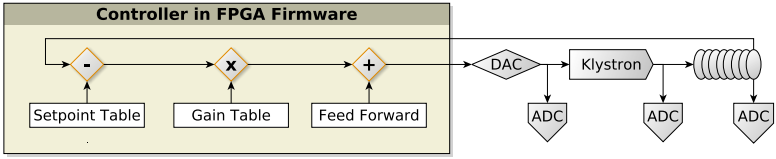
Introducing ApplicationCore



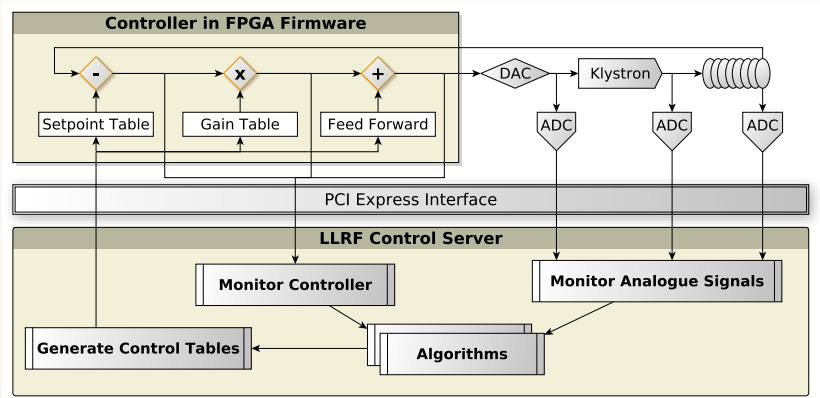
Introducing ApplicationCore



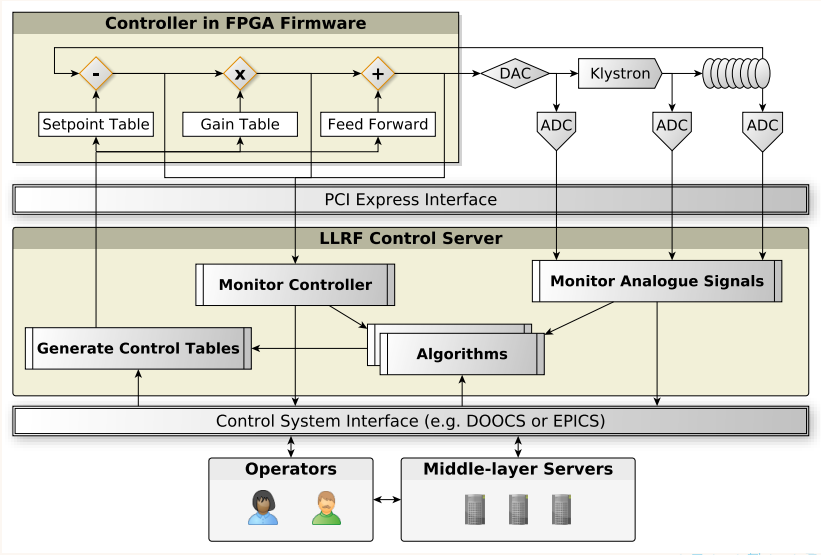
Example: Low-level RF controller server



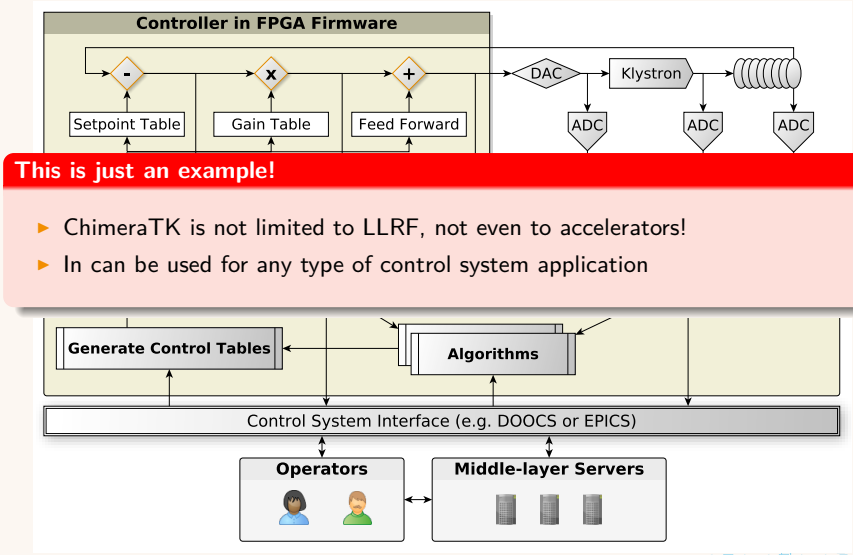
Example: Low-level RF controller server



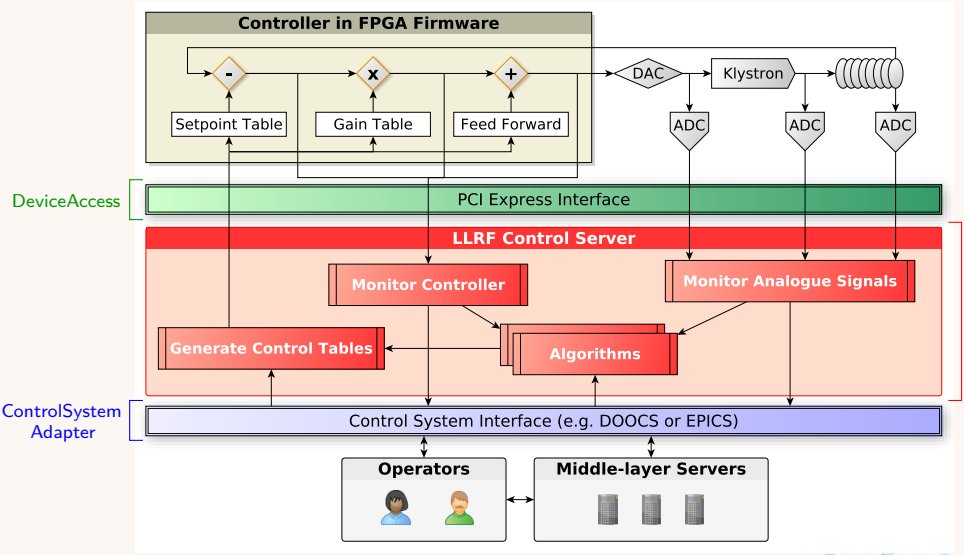
Example: Low-level RF controller server



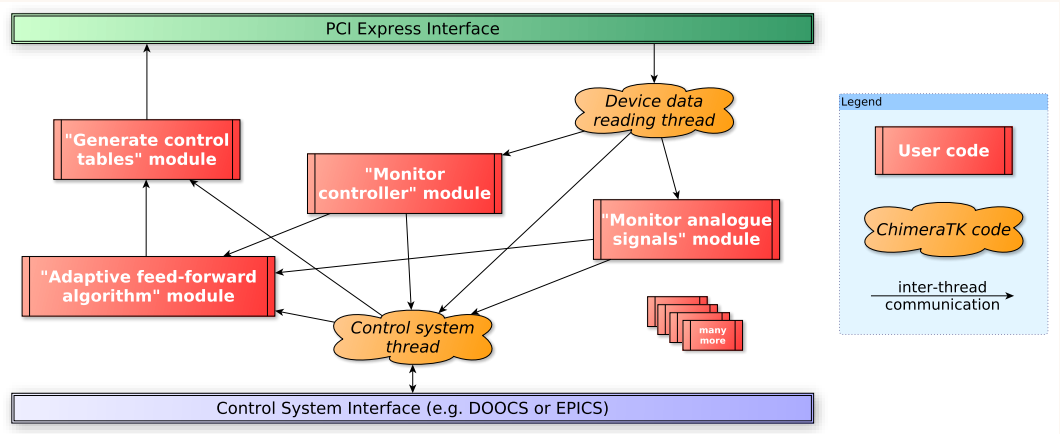
Example: Low-level RF controller server



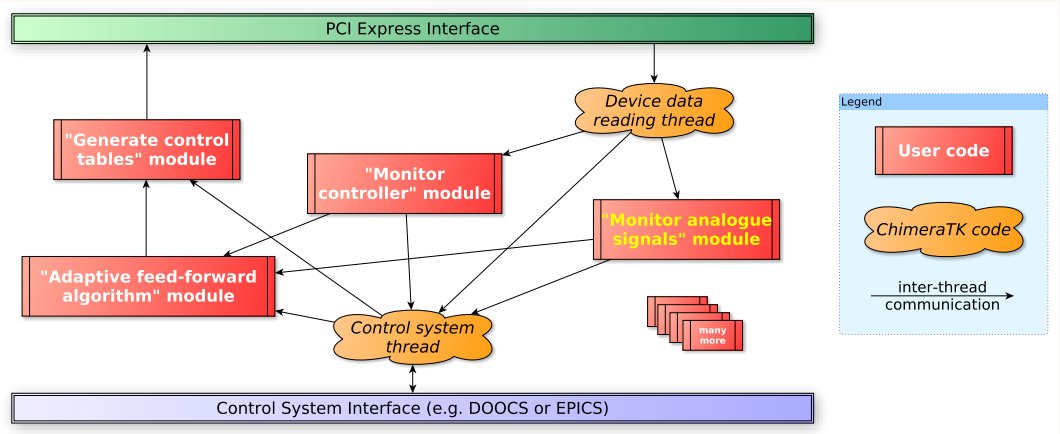
Example: Low-level RF controller server



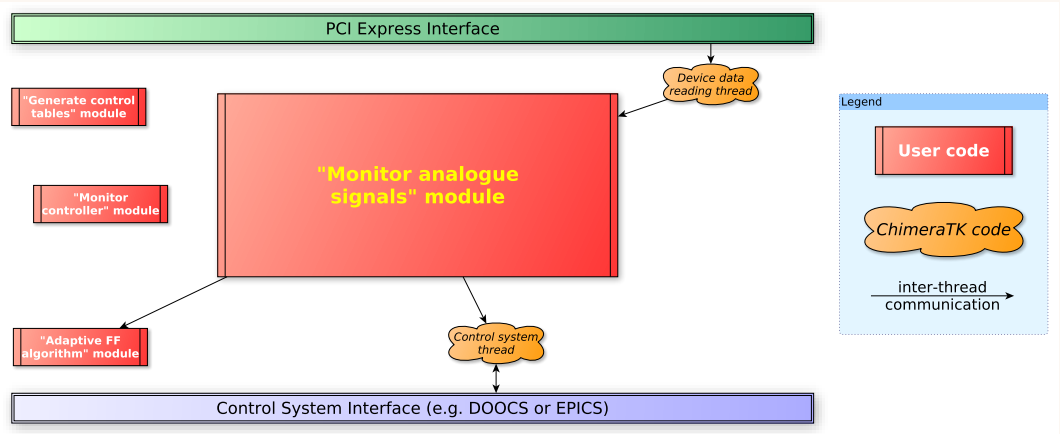
ApplicationCore modularity and multi-threading



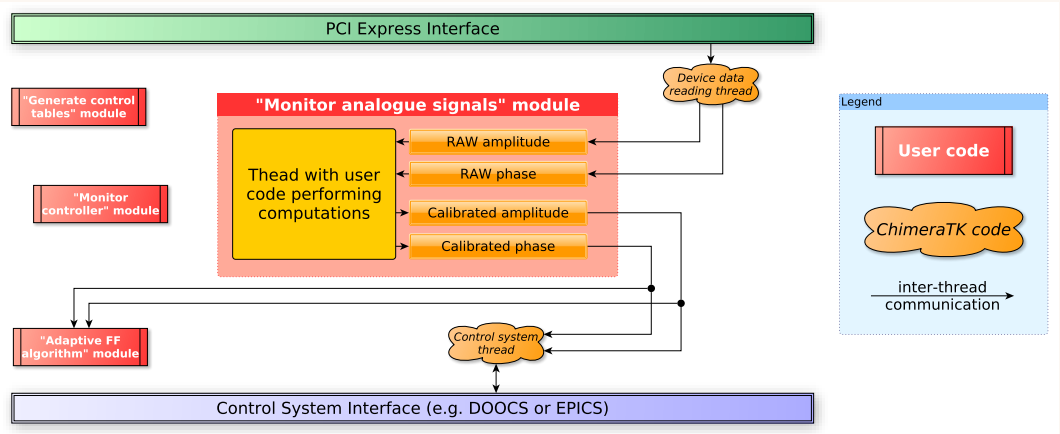
ApplicationCore modularity and multi-threading



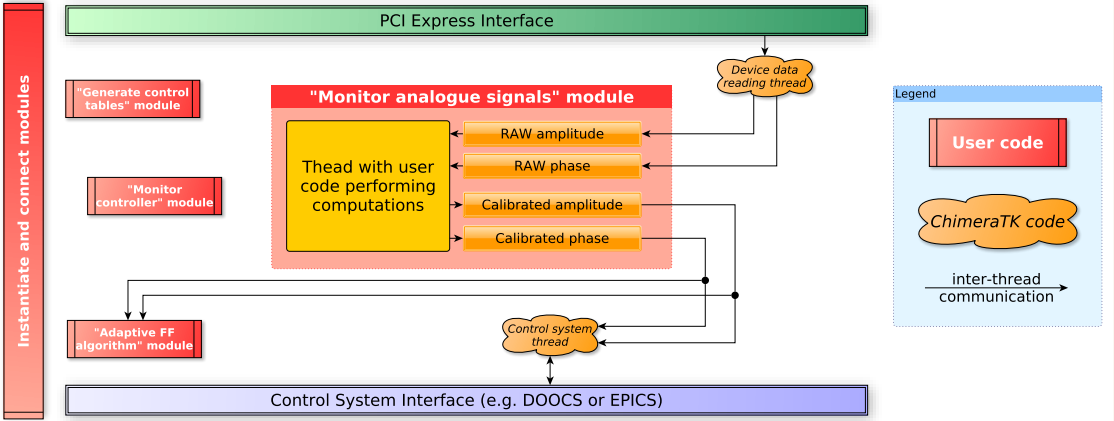
ApplicationCore modularity and multi-threading



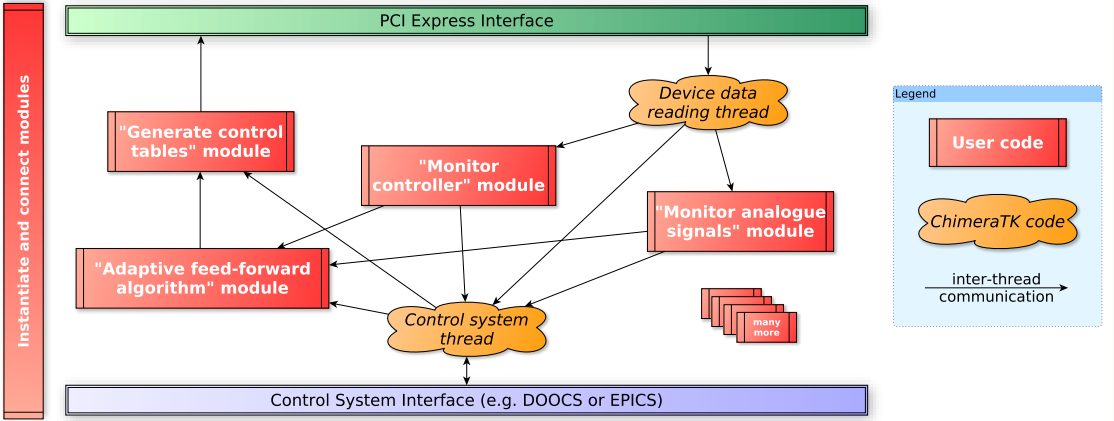
ApplicationCore modularity and multi-threading



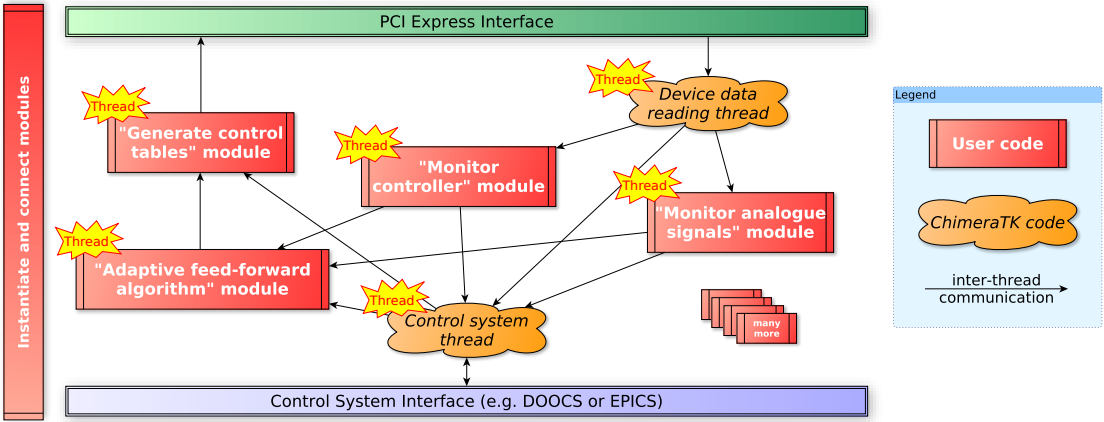
ApplicationCore modularity and multi-threading



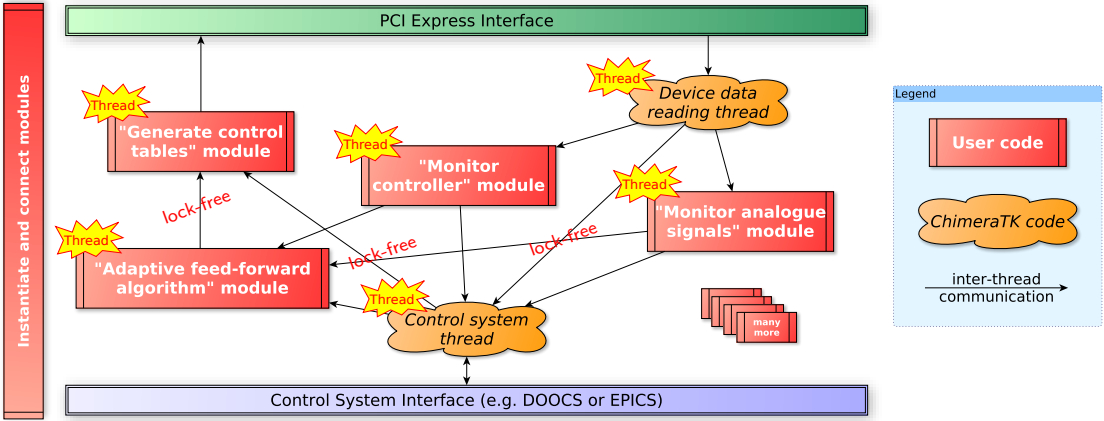
ApplicationCore modularity and multi-threading



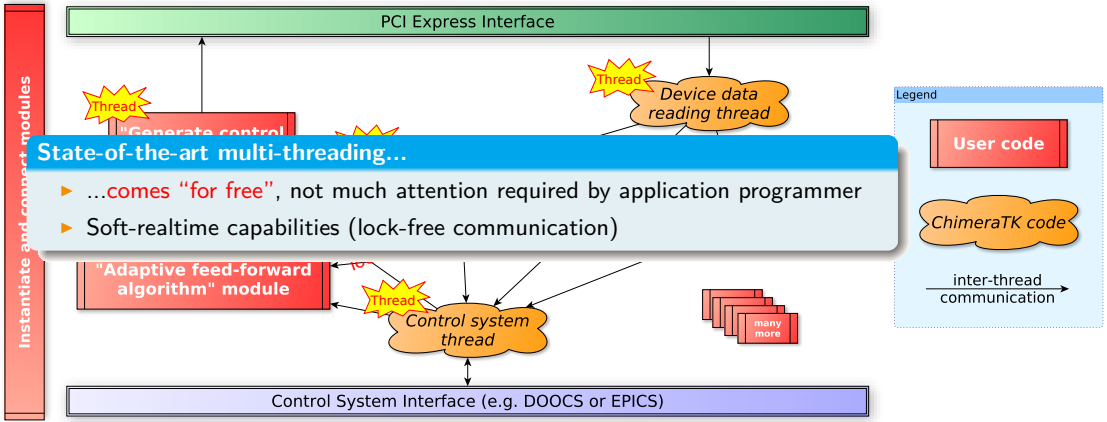
ApplicationCore modularity and multi-threading



ApplicationCore modularity and multi-threading



ApplicationCore modularity and multi-threading

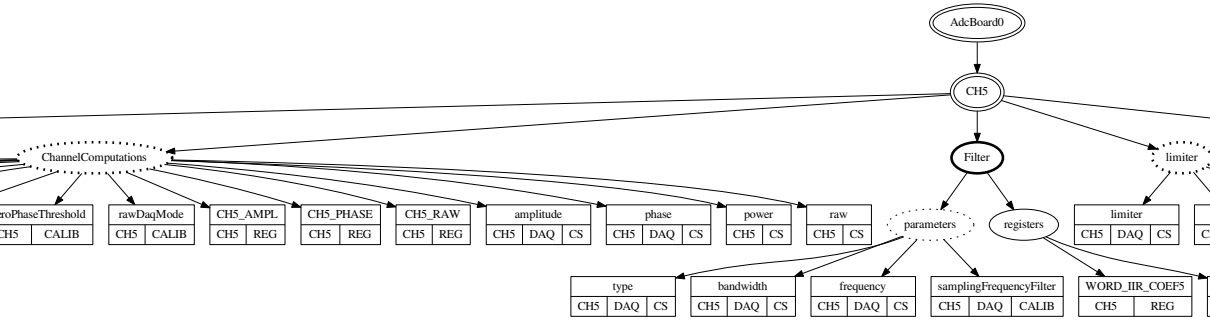


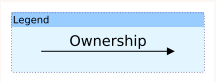
State-of-the-art multi-threading...

- ...comes "for free", not much attention required by application programmer
- Soft-realtime capabilities (lock-free communication)

Application data model

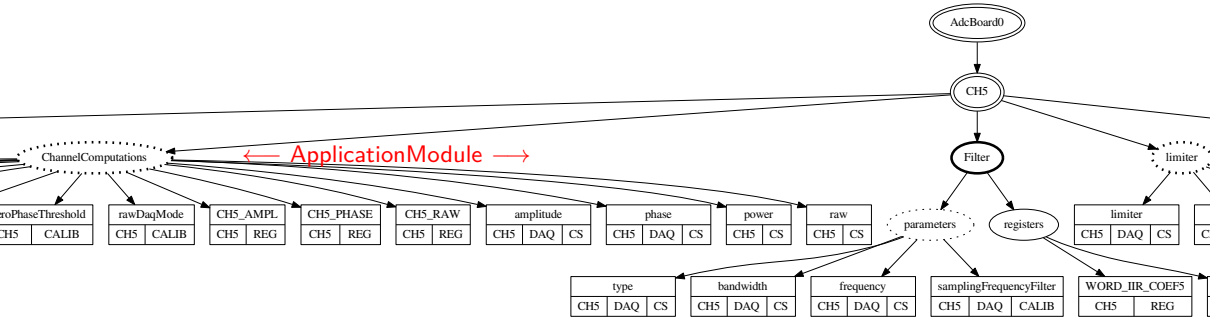
Example: “Monitor analogue signals” module for channel 5 of the ADC board in the LLRF system

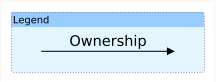




Application data model

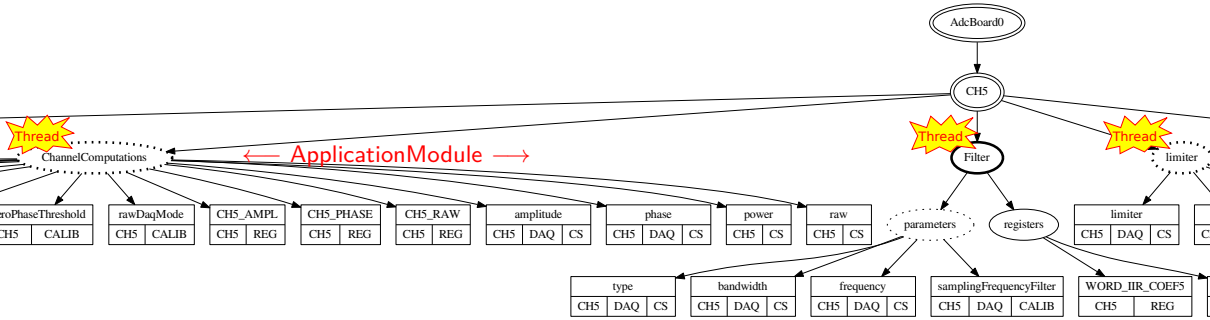
Example: “Monitor analogue signals” module for channel 5 of the ADC board in the LLRF system

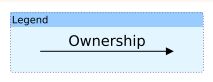




Application data model

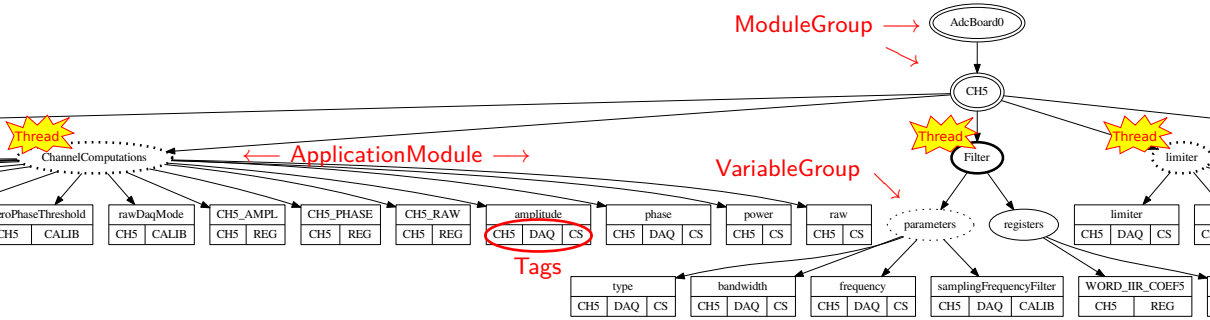
Example: "Monitor analogue signals" module for channel 5 of the ADC board in the LLRF system

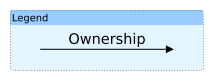




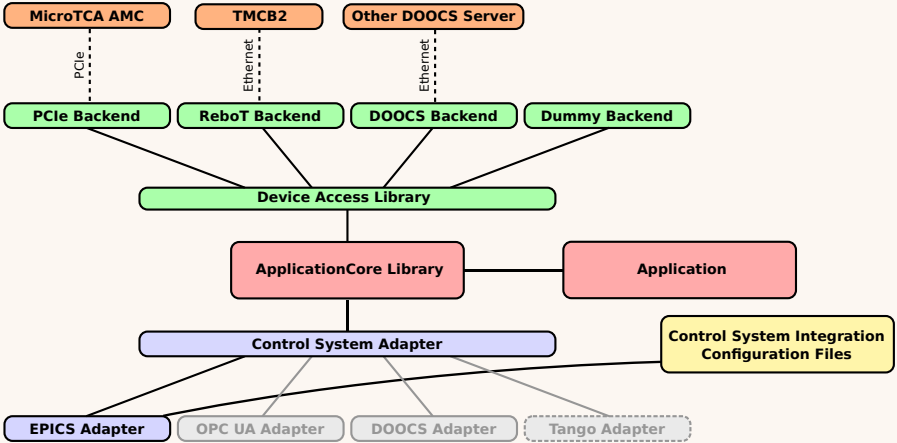
Application data model

Example: “Monitor analogue signals” module for channel 5 of the ADC board in the LLRF system

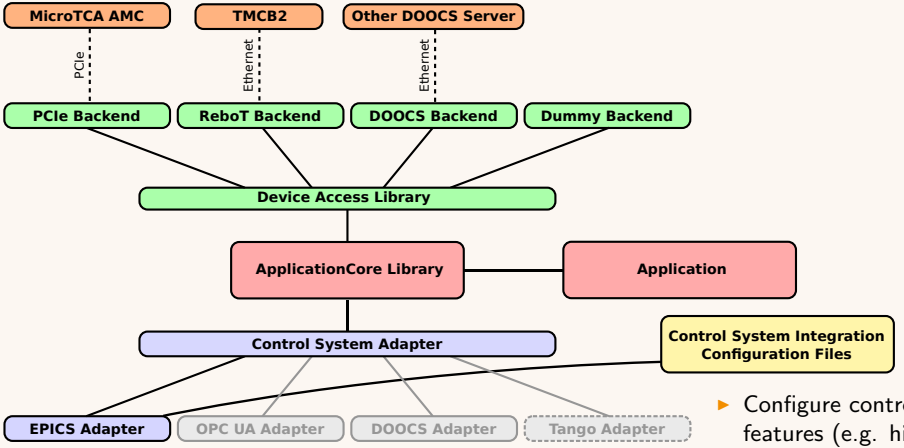




System Integration

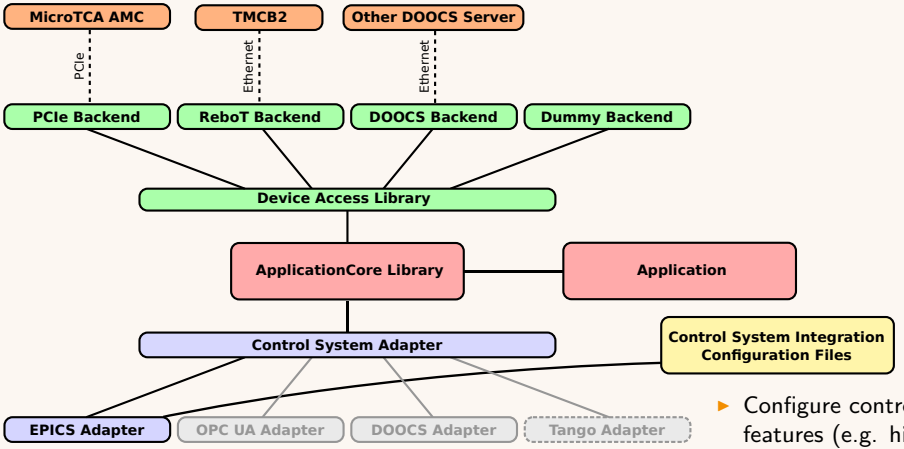


System Integration



- Configure control system specific features (e.g. history logging)
- Rename variables
- EPICS: just st.cmd and db files

System Integration



- Configure control system specific features (e.g. history logging)
- Rename variables
- EPICS: just st.cmd and db files

Conclusions and outlook

- ChimeraTK core libraries: DeviceAccess, ControlSystemAdapter, ApplicationCore
- Control system independent LLRF server in use at multiple facilities:
  - REGAE at DESY (with DOOCS)
  - ELBE at HZDR (with OPC UA / Siemens WinCC)
  - FLUTE at KIT (for now DOOCS, soon with EPICS)
  - TARLA at Ankara University (under construction, EPICS)
  - soon: bERLinPro at HZB (EPICS)
- Other (also non-LLRF) applications in development

Outlook:

- DeviceAccess: More backends, e.g. EPICS channel access client
- ControlSystemAdapter: More adapters, e.g. Tango
- ApplicationCore: Automatic exception handling (e.g. connection broken)

Conclusions and outlook

- ChimeraTK core libraries: DeviceAccess, ControlSystemAdapter, ApplicationCore
- Control system independent LLRF server in use at multiple facilities:
  - REGA
  - ELBE
  - FLUT
  - TARL
  - soon:
- Other (als

Availability

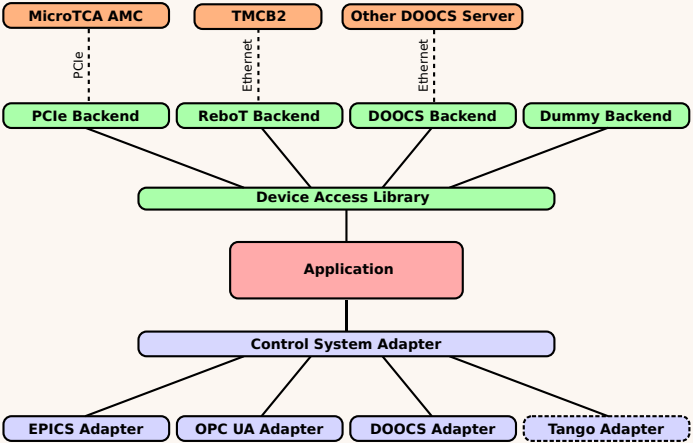
- Open source under LGPL: https://github.com/ChimeraTK
- Documentation, tutorials and support: https://chimeratk.github.io
- Ubuntu 16.04 binary packages available

Outlook:

- DeviceAccess: More backends, e.g. EPICS channel access client
- ControlSystemAdapter: More adapters, e.g. Tango
- ApplicationCore: Automatic exception handling (e.g. connection broken)

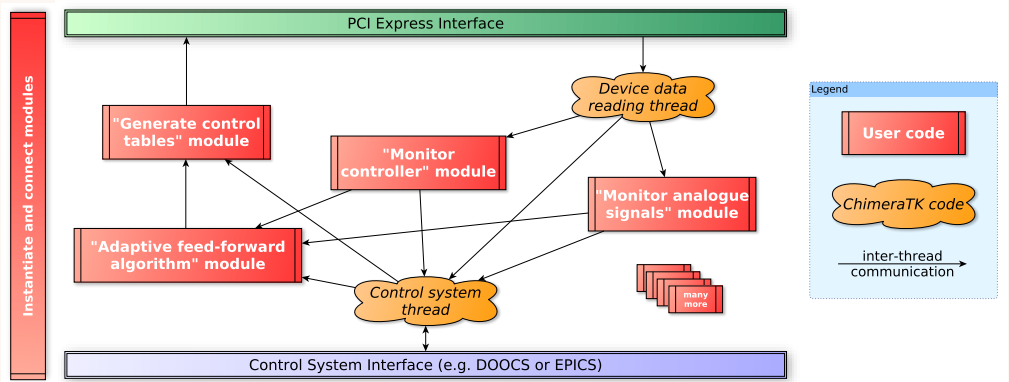
(backup)

ControlSystemAdapter



- Build application against ControlSystemAdapter to make it independent from control system middleware
- Just linking against e.g. EPICS adapter makes the application a native EPICS IOC
- No source code change necessary
- Works also for future adapters

ApplicationCore modularity and multi-threading



- One permanent thread per module
- Modern and efficient synchronisation with lockfree mechanism
- No dynamic memory allocation (soft realtime)
- Efficient multi-threading comes “for free”, not much attention required by application programmer