CMW-DIP Gateways Overview

CMW – Controls Middleware from BE-CO

ALICE workshop, 28th June 2019

Wojciech Sliwinski
for the BE-CO CMW team
CMW – Controls Middleware from BE-CO
Our clients: CERN Accelerators
CMW System Overview

High Level Applications
- LSA/InCA
- Sequencer
- SIS
- PMA
- CESAR
- Conc.
- ...

Experiments / Tech. Infra.
- ATLAS
- CMS
- LHCb
- ALICE
- COMPASS
- REMUS
- ...

Accelerator Equipment
- Magnets
- Instruments
- Interlocks
- Kickers
- Cavities
- Collimators
- ...

CMW Services & APIs
- JMS
- Gateway
- RBAC
- Proxy
- Directory Service

DIP – Data Inter-change Protocol
JMS – Java Message Service
RBAC – Role-Based Access Control
RDA – Remote Device Access
Controls Software Architecture

Presentation Layer

Controls Middleware (CMW), RMI, JMS

Business Layer

Controls Middleware (CMW)

Front End (Back End) Layer

C++

Java

CMW Purpose:
Reliable and scalable transport of data between controls processes (Java & C++)

- 4’000 CMW servers
- 85’000 devices
- 2’000’000 IO-points
- Uses ZeroMQ

Communication

CERN GIGABIT ETHERNET TECHNICAL NETWORK

FILE SERVERS

APPLICATION SERVERS

SCADA SERVERS

TCP/IP communication services

CERN

CERN

CERN

OPTICAL FIBERS

WorldFIP SEGMENTS

TCP/IP communication services

PROFIBUS

FIPI/O

BEAM POSITION MONITORS,

QUENCH PROTECTION AGENTS,

ACTUATORS AND SENSORS

CRYOGENICS, VACUUM,

RF SYSTEMS, ETC…

SYSTEMS, ETC…

INTERLOCK

BEAM LOSS MONITORS,

POWER CONVERSION

BEAM INTERLOCKS,

FUNCTIONS GENERATORS,

BEAM POSITION MONITORS,

BEAM LOSS MONITORS,

BEAM INTERLOCKS,

RF SYSTEMS, ETC…

CMW-DIP Gateways Overview
CMW mandate & scope

• Core communication layer -> critical
  • Reliable communication in distributed system
  • Decentralized (no brokers, etc.) -> scalable

• Centrally managed middleware services
  • Directory/Naming, RBAC, Proxies, DIP Gateways, ...

• Access syntax -> Device-Property model
  • Device: addressable IO point (e.g. channel, slot, module)
  • Device: e.g. LHC.BPM.P5.B1, LHC.BPM.P5.B2
  • Property: exposed operation or method
  • Property: e.g. Acquisition, Status, Alarm, Setting

• Widely deployed for all CERN accelerators
  • Used in all Eqp. groups (3 deps: BE, EN, TE)
RDA3 – CMW core communication library

- Uses **ZeroMQ** for low-level networking
- Built upon lock-free, async (event driven) architecture
- Operations/Calls: *Get, Set, Subscribe*
  - All communication implicitly asynchronous
- Public API for developing clients & servers
- Provided for several platforms
  - C++ & Java, next Python
  - Dependencies: ZeroMQ, Boost (C++)
  - Linux (64-bit), Windows (64-bit)
- Integrated with several frameworks
  - FESA, FGCD, WinCCOA, LabVIEW, C2MON
- Provides comprehensive diagnostics
  - Admin GUI, command line tools, Kibana reports
- Exportable -> used in GSI (DE), LG (S. Korea)
## RDA3 vs DIP

<table>
<thead>
<tr>
<th>Context/Area</th>
<th>RDA3</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication model</td>
<td>Peer to peer</td>
<td>Peer to peer</td>
</tr>
<tr>
<td>Domain</td>
<td>Accelerator control system</td>
<td>Experiment control system</td>
</tr>
<tr>
<td>Access point</td>
<td>device/property</td>
<td>topic</td>
</tr>
<tr>
<td>Paradigm</td>
<td>Request/Reply Publish/Subscribe</td>
<td>Publish/Subscribe</td>
</tr>
<tr>
<td>Data type</td>
<td>Generic data or exception</td>
<td>Generic data + quality</td>
</tr>
</tbody>
</table>
CMW Gateway basic architecture

CMW

RDA3 client

DIP to CMW

RDA3 server

DIP subscriber

DIP publisher

DIP

DIP subscriber

CMW

RDA3 server

CMW to DIP

RDA3 client

DIP publisher

publication

notification

publication

publication
## CMW-DIP Gateways

Critical: Handshake, Luminosity, RunControl

<table>
<thead>
<tr>
<th>GW-ALICE</th>
<th>GW-ATLAS</th>
<th>GW-INJ-GPNDIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>GW-LHC-BPM</td>
<td>GW-LHC-CONF</td>
<td>GW-LHC-EXP-GPN</td>
</tr>
<tr>
<td>GW-LHC-INFO</td>
<td>GW-SERVICES</td>
<td>GW-BL4S</td>
</tr>
<tr>
<td>GW-LHC-BCT</td>
<td>GW-LHCB</td>
<td>GW-LHC-CRITICAL-EXP</td>
</tr>
<tr>
<td>GW-LHCF</td>
<td>GW-NORTH</td>
<td>GW-CMS</td>
</tr>
<tr>
<td>GW-GPNDIP-INJ</td>
<td>GW-LHC-BLM</td>
<td>GW-LHC-COLLM</td>
</tr>
<tr>
<td>GW-LHC-CRITICAL</td>
<td>GW-LHC-GPNDIP</td>
<td>GW-RADMON</td>
</tr>
<tr>
<td>GW-TOTEM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Resources

- General documentation: https://wikis.cern.ch/display/expcomm/
- Publications from the LHC: https://wikis.cern.ch/display/expcomm/DIP+LHC
- CMW Service: https://wikis.cern.ch/display/MW/
- DIP Service: https://readthedocs.web.cern.ch/display/ICKB/DIP+and+DIM

- Emails:
  - cmw-support@cern.ch (CMW related questions, requests, etc.)
  - dip-support@cern.ch (DIP related questions, requests, etc.)
  - icecontrols.support@cern.ch (BE-ICS support)
What are main CMW features?

- Device/Property model
- Basic operations: Get, Set, Subscribe
- Get/Set:
  - Synchronous or asynchronous
  - Grouping execution by device (default)
- Subscription:
  - First-update
  - Ordering of notifications
- Multiplexing:
  - Cycle selector for requests
  - Cycle information for acquisitions
- Rich data types (scalars, N-arrays, structures)
- Authentication and authorization
- Tracing and diagnostics
Data flow for RDA3 calls

- **Get & Set calls**
  - RDA3 client
    - ZeroMQ
  - RDA3 server
    - ZeroMQ
    - Device1/PropertyA
    - Device2/PropertyB
    - ... 
    - DeviceN/PropertyX
  - Directory Service
    - Cache (Devices & Servers)
  - Database

- **Subscribe calls**
  - RDA3 client
    - ZeroMQ
  - RDA3 server
    - ZeroMQ
    - Device11/PropertyA
    - Device22/PropertyB
    - ... 
    - DeviceNN/PropertyX
  - Directory Service
    - Cache (Devices & Servers)
  - Database

Steps:
1. Bind
2. Load devices & servers
3. [0] Lookup
5. [2 & 2'] Save
6. [3] Publish
7. [4] Subscribe
Example – sync Get call

```cpp
#include <iostream>
#include <cmw-rda3/client/service/ClientServiceBuilder.h>
...

// First create ClientService and keep it for all further communication
auto_ptr<ClientService> client = ClientServiceBuilder::newInstance()->build();

// Get AccessPoint by providing device name & property name
AccessPoint & accessPoint = client->getAccessPoint("LHC.BPM.P5.B1", "Acquisition");

// Perform sync GET call
auto_ptr<AcquiredData> acqData = accessPoint.get();

// The resulting data is obtained by calling getData() method on the AcquiredData instance
const Data & data = acqData->getData();
cout << data.toString() << endl;

// Data -> supports scalars, arrays of scalars(1D, 2D, N-D), structures (nested Data)

// Additionally AcquiredData contains also AcquiredContext, which provides meta-data about the call
const AcquiredContext & acqContext = acqData->getContext();
cout << acqContext.toString() << endl;
```
Example – async Get call with callback

```cpp
#include <iostream>
#include <cmw-rda3/client/service/ClientServiceBuilder.h>
...
class Callback : public AsyncGetCallback
{
public:
  void requestCompleted(const RequestHandle & request, auto_ptr<AcquiredData> acqData)
  {
    cout << acqData->getData().toString() << endl;
  }

  void requestFailed(const RequestHandle & request, auto_ptr<RdaException> exception)
  {
    cout << exception->what() << endl;
  }
};

auto_ptr<ClientService> client = ClientServiceBuilder::newInstance()->build();
AccessPoint & accessPoint = client->getAccessPoint("LHC.BPM.P5.B1", "Acquisition");

// Perform async GET call with callback
accessPoint.getAsynccAsyncGetCallbackSharedPtr(new Callback());

// Continue while GET is being performed
... ...
```
Example – async Get call with future

```cpp
#include <iostream>
#include <cmw-rda3/client/service/ClientServiceBuilder.h>
...
...
auto_ptr<ClientService> client = ClientServiceBuilder::newInstance()->build();
AccessPoint & accessPoint = client->getAccessPoint("LHC.BPM.P5.B1", "Acquisition");

// Perform async GET call with future
RdaGetFutureSharedPtr future = accessPoint.getAsync();
...

// Continue processing while GET is being performed
...
cout << "Is GET completed: " << future->isDone() << endl;
...

// Perform blocking call until GET data is ready
auto_ptr<AcquiredData> data = future->get();

cout << data->getData().toString() << endl;
```
# Example code – sync Set call

```cpp
#include <iostream>
#include <cmw-rda3/client/service/ClientServiceBuilder.h>
...
...
// First create ClientService and keep it for all further communication
auto_ptr<ClientService> client = ClientServiceBuilder::newInstance()->build();

// Get AccessPoint by providing device name & property name
AccessPoint & accessPoint = client->getAccessPoint("LHC.BPM.P5.B1", "Setting");

// Prepare data to be sent to server
// Data -> supports scalars, arrays of scalars(1D, 2D, N-D), structures (nested Data)
auto_ptr<Data> data = DataFactory::createData();
data->append("value", 10);
data->append("gain", 5.1234);

// Perform sync SET call
accessPoint.set(data);
```

→ Analogically async Set versions with callback & future
```cpp
#include <iostream>
#include <cmw-rda3/client/service/ClientServiceBuilder.h>

... ...

auto_ptr<ClientService> client = ClientServiceBuilder::newInstance()->build();
AccessPoint & accessPoint = client->getAccessPoint("LHC.BPM.P5.B1", "Status");

// Perform SUBSCRIBE call
SubscriptionQueueSharedPtr queue = accessPoint.subscribe();
...

// Continue processing while subscription is being established
...

// Blocking wait for a single notification with a timeout of 5000 ms
std::auto_ptr<Notification> notification = queue->poll(5000); // also with no timeout: poll()
cout << "Received notifications count: " << queue->getQueueSize() << endl;

// Process the received data
auto_ptr<AcquiredData> acqData = notification->get();
cout << acqData->getData().toString() << endl;
```

→ Analogically Subscribe version with callback
Latency stability measured by BE-CO Timing team

Setup: 500B message payload, CCC machines, 1 server/publisher, 1-1100 clients/subscribers
Observations from operational use of RDA3

- Great performance for subscriptions (thanks to async transport & internal batching)
- Good scalability for many clients (>1000)
- No more problem of slow clients (solved properly thanks to async transport)
- Much smaller footprint (CPU & memory), mainly thanks to zero-copy
- ZeroMQ - no surprises & very reliable!