LHCCP Workshop

Controls Middleware (CMW) Status and Deployment

June 12, 2003

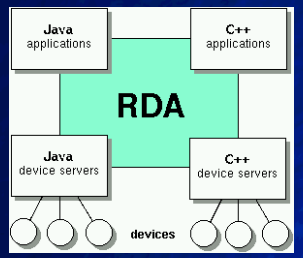
Kris Kostro, AB/CO/FC
For the CMW team

Presentation layout

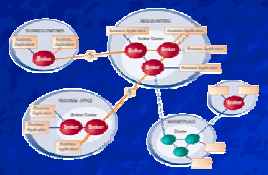
- Introduction
- CMW guided tour
- Deployment status
- Future work
- Conclusions

Device Access model versus Topic model

Remote Device Access



JMS - SonicMQ



- Device Access model and RDA are the mainstream of CMW
- SonicMQ is used via J2EE, for Alarms, candidate for DIP

Milestones recall

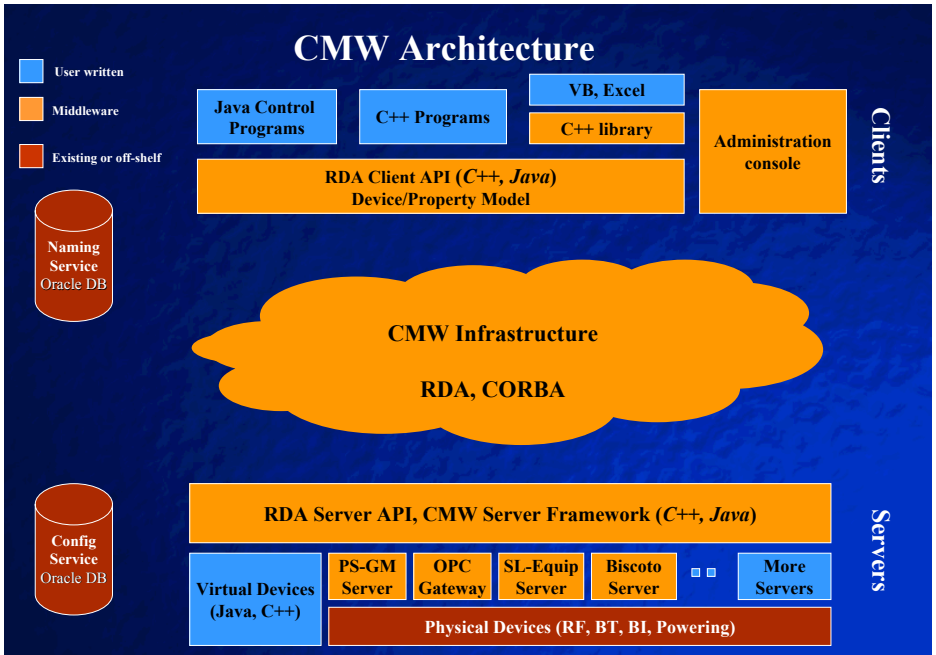
- August 1999 - User Requirement Document published
- September 2000 – First version of RDA
- March 2002 – CMW servers deployed on all PS front-ends
- 2003 – Stable APIs. Infrastructure fully developed

From project to support

- Since the AB division has been created, the CMW responsibility is fully within the AB/CO/FC section:
  - Kris Kostro (overall support, servers)
  - Steen Jensen (servers, operation)
  - Nikolai Trofimov (protocol, development)
- CMW has been designated by the group as the standard equipment access method
- Every piece of equipment that has to be remotely operated from AB/CO back-end or console computers shall ultimately be interfaced with CMW and accessible via a unique API

Presentation layout

- Introduction
- CMW guided tour
- Deployment status
- Future work
- Conclusions



Devices and Properties

- A *device* is a named entity within the control system, which corresponds to a physical device (Beam Position Monitor, Power Converter) or to a virtual controls entity (e.g. transfer line)
- The state of a device is accessed via *properties* and can be read or modified by the *get* and *set* access methods. (synchronous and asynchronous)
- Property can be *monitored* (publish/subscribe)
  - A *cycleSelector* or a polling period can be specified
  - Optional on-change mode: client will be notified only when property has changed (server criteria).

CMW Data Format

- Property values delivered to the client or set by the client have the form of ***Data*** objects.
- Data object is a container for one or many ***DataEntry*** objects.
- Each *DataEntry* has a **tag** (a string) and a **value**, which can be a scalar, a string or an array of these.
- Device classes can implement many properties of simple type (PS, SPS, OPC) or few properties of composite type (Biscoto)
  - On-change update can be optimized for simple types
  - Composite properties make sense for data acquisition

BCT Acquisition as example of composite property

Standard Entries

Data object for the Acquisition property	
tag	**value type**
cycleDescriptor	String
cycleStamp	Double
timeStamp	Double
numberOfMonitors	Integer
numberOfBunch	Integer
maximumBunchIntensity	Float
bunchIntensity	Float[NbOfBunch]
minimumBunchIntensity	Float
bunchSpreadSigma	Float
statusTag	Long

Cycle Selector

- A cycle selector specifies an accelerator condition for an I/O operation.
- Currently specified as a String, to be interpreted by the equipment servers via timing services.
- PS semantics : *telegram.group.line*
  - CPS.USER.SFTPRO
- SPS semantics : *sequence.cycle.cycle-instance*
  - SPS_production_FT_CNGS.CNGS.2
- With monitoring, cycle selector implies the moment when polling should occur.
- With monitoring a polling period can be specified instead of cycle selector.

Acquisition Stamping

- All data sent to the client is accompanied by a ***timeStamp***. The meaning of this timestamp can depend on the device but it usually describes the best approximation of time at which the acquisition was made (or a setting was changed).
- In the PS complex, timestamps are replaced by ***cyclestamps*** to facilitate the correlation of data belonging to the same cycle. A ***cycleStamp*** is the timestamp at begin of the cycle for which the acquisition was made and it uniquely identifies the cycle.
- Timestamps (and cycle stamps) are currently expressed as milliseconds from POSIX origin (Jan 1970) and transmitted as seconds in a double.
- 64bits double have a 52 bits mantissa. 32 bits for seconds (same as Posix) leaves 20 bits for fractions. Micro-seconds resolution is supported.
- Alternate choice: 64 bit integer expressed in nano-seconds. (need `long long int` support)
- Proposition: stay with double (could be changed later if required).

CMW Java clients

- CMW provides the low-level RDA API
- AB/CO/AP provides the high-level API

ASC or JAPC API

↓

RDA Client API

```
try  {
        RDAService rda = RDAService.init();
        deviceName = "BCTFI_LABO10";
        property = "BCTFI.Shared.Actions.acquisition";
        DeviceHandle device = rda.getDeviceHandle(deviceName);

        result = device.get(property, "21000101");
        System.out.println(result.toString());
 }
```

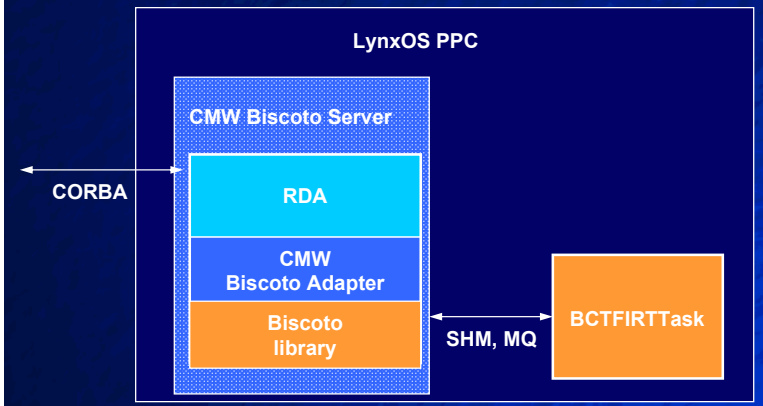
```
tag   : cycleDescriptor
type  : byte [64]
value : 72
        73
        95
...
tag   : statusTag
type  : int [10]
value : 1212768075
        1380537137
        838860800
...
tag   : minimumBunchIntensity
type  : float [10]
value : 206205.17
        2.16171037E11
        7.4505806E-9
...
tag   : timeStamp
type  : double
value : 1.7267453847708817E40

tag   : maximumBunchIntensity
type  : float [10]
value : 206205.17
        2.16171037E11
        7.4505806E-9
...
tag   : bunchSpreadSigma
type  : float [10]
value : 206205.17
        2.16171037E11
        7.4505806E-9
...
```


CMW C++ servers

- Libraries available for LynxOS, Linux and Windows
- Servers can be built with RDA C++ API and with CMW Server Framework
- CMW server has to be built once for each category of equipment access architecture:
  - PS GM server
  - SPS SL-Equip server
  - OPC GW
  - Biscoto server
- Waiting for the “ultimate” F-E architecture delivered by the FECOMSA project.
- Reduced number of architectures will ease maintenance

Server example: Biscoto



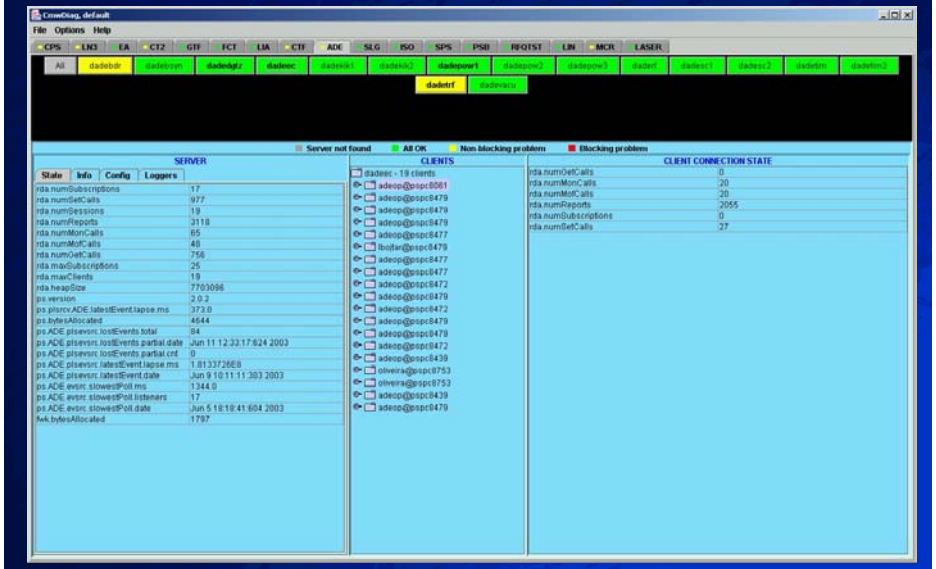
Naming and Configuration

- Resolution of device name to CMW server is supported by information in the Oracle DB
  - Two separate databases and methods are used now
  - Ultimately this service will be assured by CO/DM for controls equipment
- CMW offers a service of dynamic server configuration at start-up. This service is maintained by CMW team but the configuration is maintained by CO/DM.

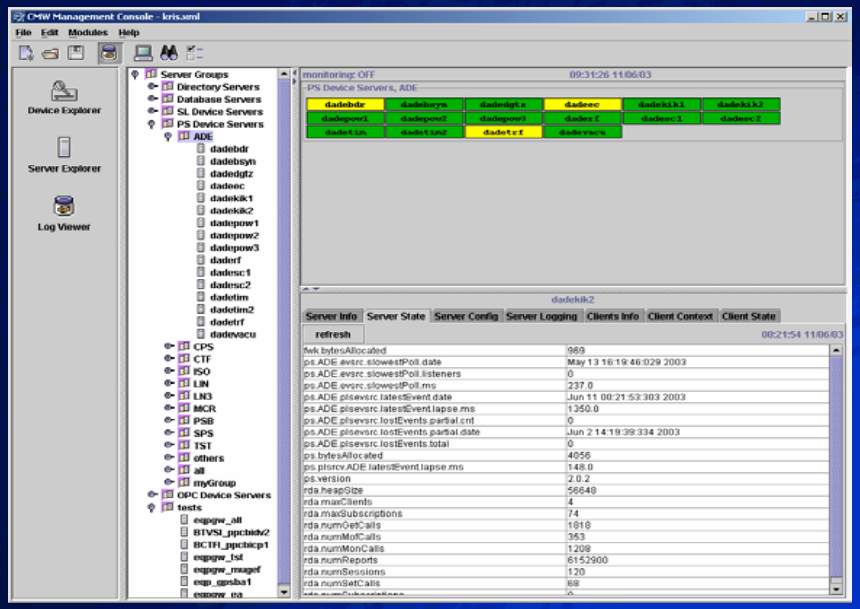
CMW Admin facilities

- Each CMW server implements administration interface
- Administration interface can be remotely used by administration tools
  - General server status (green, red, yellow)
  - Generic server information (start date, name, pid, etc.)
  - Server statistics (RDA and server-specific)
  - Connected clients
  - Server configuration
  - Control of logging levels
  - Restart, shut-down

CMW Diagnostics Console



CMW Admin Console



Some performance figures

Client	Server	Synch. get() 1 value + 1 timestamp
800 MHz Windows, Java	LynxOS 3.1.0, Rio 2 CMW server only	2.1 ms
800 MHz Windows, Java	PS GM server LynxOS 3.1.0, Rio 2	4.5 ms
800 MHz Windows, Java	CMW OPC GW 850 Mhz Windows	1.8 ms

- Monitoring on PS servers - 1000 updates in 200ms

Presentation layout

- Introduction
- CMW guided tour
- Deployment status
- Future work
- Conclusions

Deployment status

- ✓ CMW Base Products
  - ✓ RDA (Java, libraries for Linux & Windows)
  - ✓ Naming and configuration services
  - ✓ Administration tools
- ✓ Access to existing PS & SPS equipment via CMW
  - ✓ Deployed in PS since 2002
  - ✓ Deployed in SPS as SL-Equip gateway for Excel passerelle and RF, can also be deployed natively on LynxOS F-E
  - ✓ Gateway for CESAR (EA renovation) to access all EA equipment

CMW for TT40 commissioning

- Generic Biscoto CMW server developed and tested with BCTFI
- Other Biscoto instruments: BPMI, BTVP
- Via CMW Equip servers: beam loss (BLR), SPS orbit (MOPOS), other?
- Currently helping to set up “shot-by-shot” logging using subscription.
- Collaboration with AB/BDI and AB/CO/AP
- Beam Interlock System (BIC), currently under development, is using PS-type CMW server.

LHC RF (HW commissioning)

- RF Klystron tests during 2002/2003 (finished)
- RF 400Mhz cavity tests in SM18 to begin soon
- Standard CMW OPC gateway used to access Schneider PLCs
- Java GUI with RDA API
- OPC gateway can only run on Windows
- RF is looking forward for a solution to interface PLCs via a standard AB/CO platform

LHC Systems

- LHC Power Converters
  - First approach abandoned End of 2001
  - Second approach End of 2002 with help of CMW team
  - CMW server for LHC Power Converters ready, including subscription (S. Page, AB/PO).
- Quench Protection System
  - PS CMW server available
  - May need CMW accessibility from PVSS

Presentation layout

- Introduction
- CMW guided tour
- Deployment status
- Future work
- Conclusions

Future Work around CMW

- Standardize usage of CMW across various servers
  - Timestamps
  - Filters
  - Generic Contracts, contract and property description
- Exploit full potential of CMW servers
  - Access control, device reservation etc.
- Develop CMW server for FECOMSA standard front-end server framework
- Interface CMW with PVSS

Interfacing CMW and PVSS

- Number of LHC systems will require connection between PVSS and CMW (Vacuum, QPS)
- Half-gateway has been developed by ST for the GTPM project.
  - It is a “PVSS driver” for CMW
  - Allows subscription to CMW devices
  - Has to run on Windows since no working Linux version could be generated.
  - May not be a workable solution for Linux – mixing of gcc versions is known to be difficult with PVSS.
- Will address this problem during 2003

Conclusions

- CMW in The AB/CO standard for remote equipment access
- The infrastructure is finalized
- Existing PS/SPS equipment is accessible
- Requirements for TT40 commissioning, HW commissioning, LHC, as far as known, have been addressed. Working closely with equipment groups, CO/AP and operation.

- The upcoming equipment server framework has to be integrated.
- PVSS has to be interfaced with CMW