Status of mTCA Slow Control development at CMS

Petr Volkov

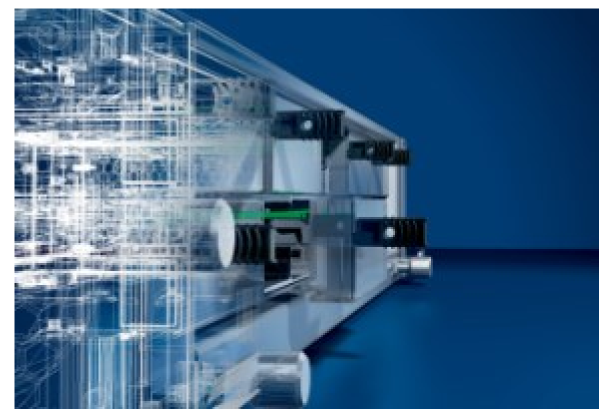
Lomonosov Moscow State University Skobeltsyn Institute of Nuclear Physics, Russia

uTCA: Introduction

MicroTCA® - open standard for fabric computer systems.
MicroTCA systems are both physically small and non-expensive.
Nevertheless their internal architectures are large.

MicroTCA design goals:

- Favorable cost, size, and modularity
- Target low start-up costs
- Scalable Backplane bandwidth
- Modular and serviceable
- Standardized Shelf management implementation
- 300 mm nominal equipment depth
- 19 in. nominal equipment width
- Cooling: 20–80 W/Card
- Extended temperatures (–40 to +65 degrees)
- Power: 12 V
- Life span: at least eight years



uTCA: CERN

Major upgrades of the LHC experiments at CERN are foreseen over > 10 years

- aligned with LHC upgrade long shutdowns: 2013/14, 2018, 2023

Off-detector electronics of the LHC experiments mostly based on VME

- “old” technology and doubts about long-term availability

Experiments planning to use MicroTCA & ATCA for upgrades of their back-end electronics

- MicroTCA: CMS
- ATCA: LHCb & ATLAS

xTCA advantages

- choice of form factors
- backplane bandwidth and protocols
- cooling and power supply
- redundancy (PSU, cooling)
- infrastructure monitoring features

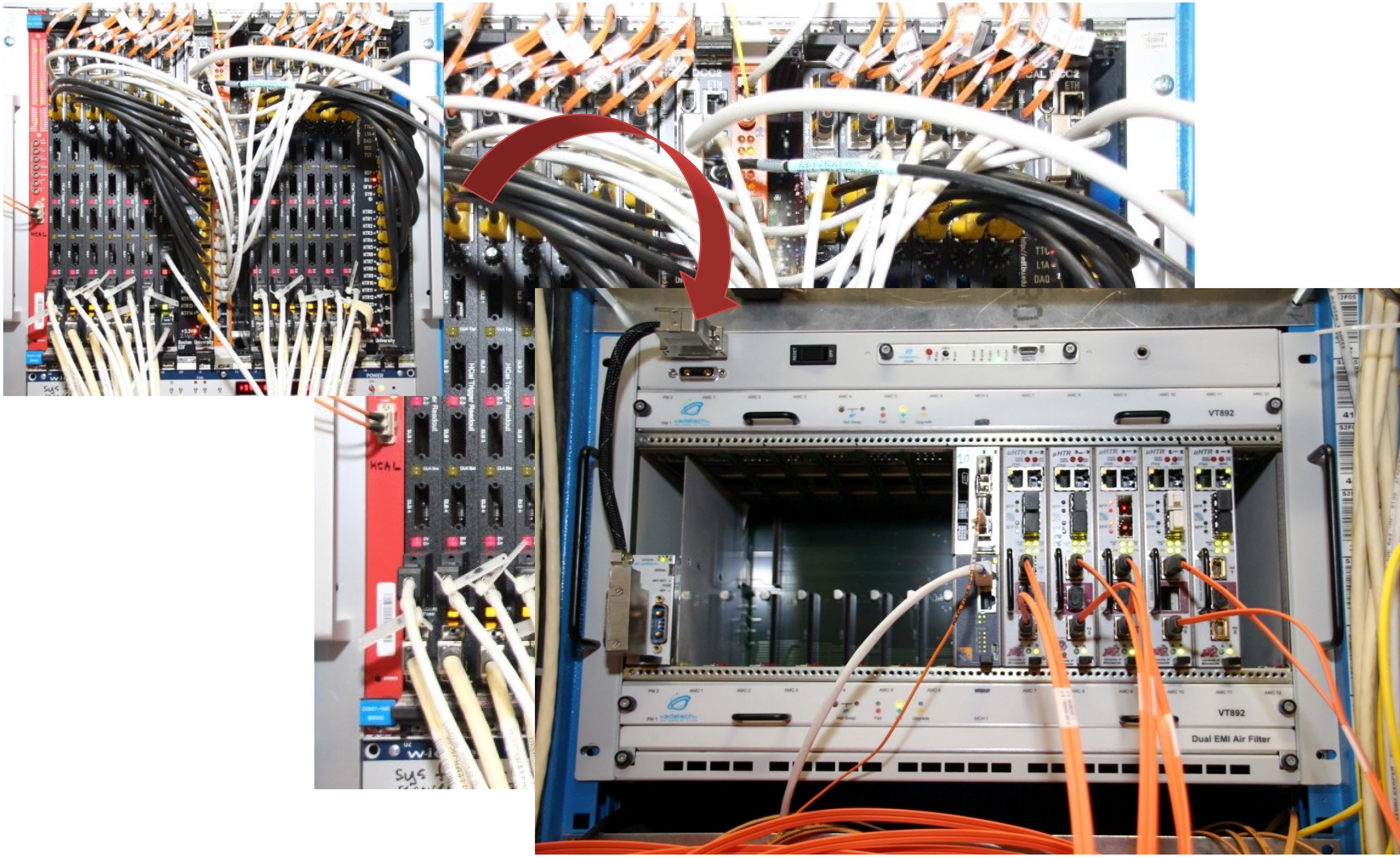
MicroTCA and ATCA developments already on-going at CERN and collaborating institutes

Accelerator sector is also investigating MicroTCA

uTCA: CMS

- CMS is upgrading back end electronics. Why?
  - Performance
    - be ready for 4 times more data
    - be ready for much higher luminosity / occupancy
    - be ready for filter / trigger processes needing much more data
    - current system not able to cope with this
  - Maintainability
    - avoid legacy support for 15 years or more
      - current design based on pre 2000 technology
    - reduce complexity
      - reduce number of cables for crate internal data transfer
      - reduce number of different boards
      - get rid of many mezzanine cards

uTCA: Reduce complexity – from VME to mTCA



CMS: Moving to mTCA

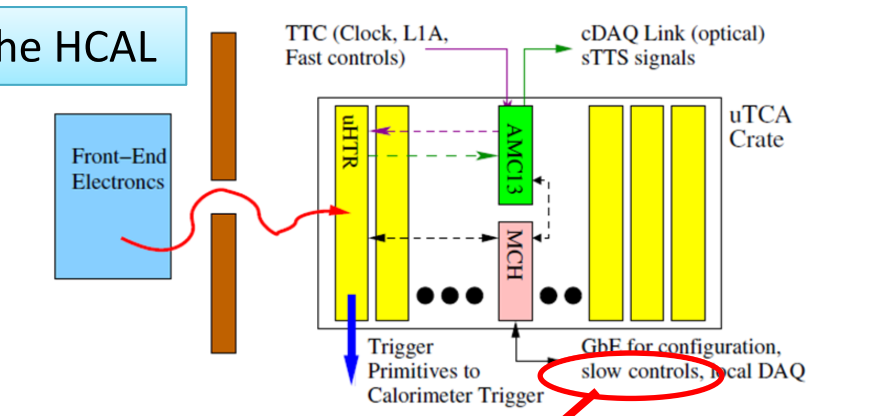
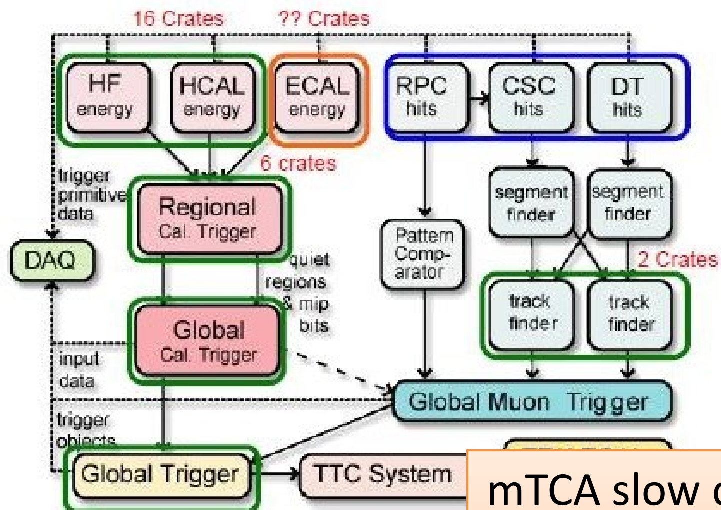

6.3. Back-end Architecture 117

Figure 6.1: Crate layout structure of the μTCA-based back-end electronics, showing the data links from the front-end electronics to the uHTRs, the connections from the uHTR cards to the calorimeter trigger, and the fast control and DAQ connections which connect to the AMC13 and use the μTCA backplane.

CMS TDR for the phase 1 upgrade of the HCAL

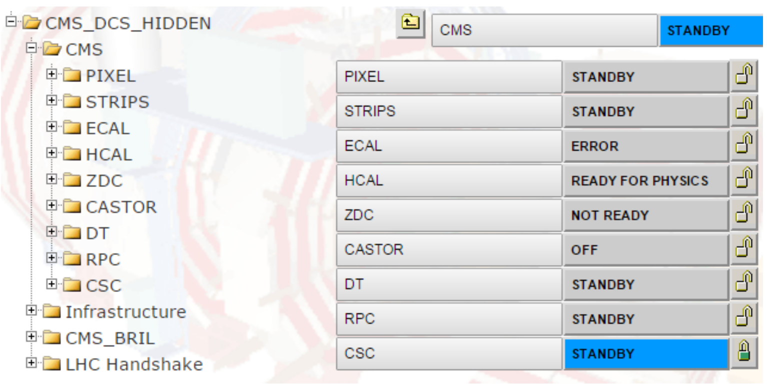


Moving VME to microTCA

mTCA slow control tasks:

- to bring into any desired operational state
- to signal any abnormal behavior to the operator
- and to allow manual or automatic actions to be taken
- to monitor and archive the operational parameters such as voltages, currents, temperatures

CMS: Detector Control System



CMS Detector Control System (DCS) handles

- configuration
- monitoring
- operation

of all experimental equipment
and provides an interface between the user and the physics setup

The CMS DCS **mandatory** rules:

1. Use PVSSII and the JCOP Framework to develop your control applications. These are the official CMS DCS developing tools.

2. Follow the CMS DCS naming conventions.

3. Create the FSM detector trees following the CMS FSM conventions.

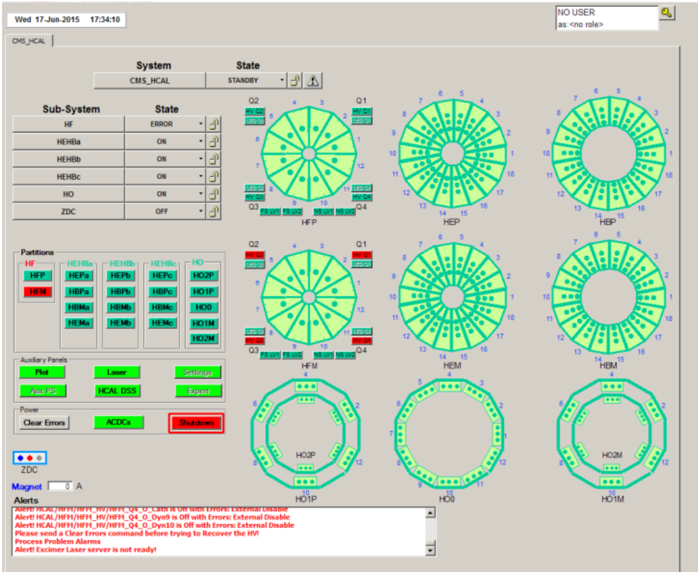
4. Create Detector Framework components out of your control applications.

5. Install only one PVSSII project in each production system.

6. Use CMS central software repository for your Detector and JCOP framework components.

7. Integrate access control in your DCS applications.

8. Follow CMS DCS alarm handling policies.



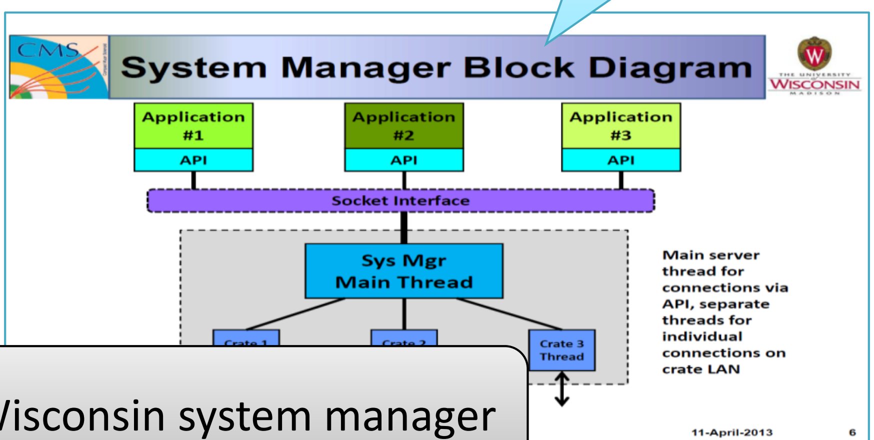
uTCA Slow Control: Software architecture

Configuration database

Condition database

PVSS (WinCC OA)
control, monitoring, visualization

Interface?



Wisconsin system manager

Control and monitoring:
WinCC OA project - Win7 (64 bit)

Low level interface:
Wisconsin System Manager – Linux

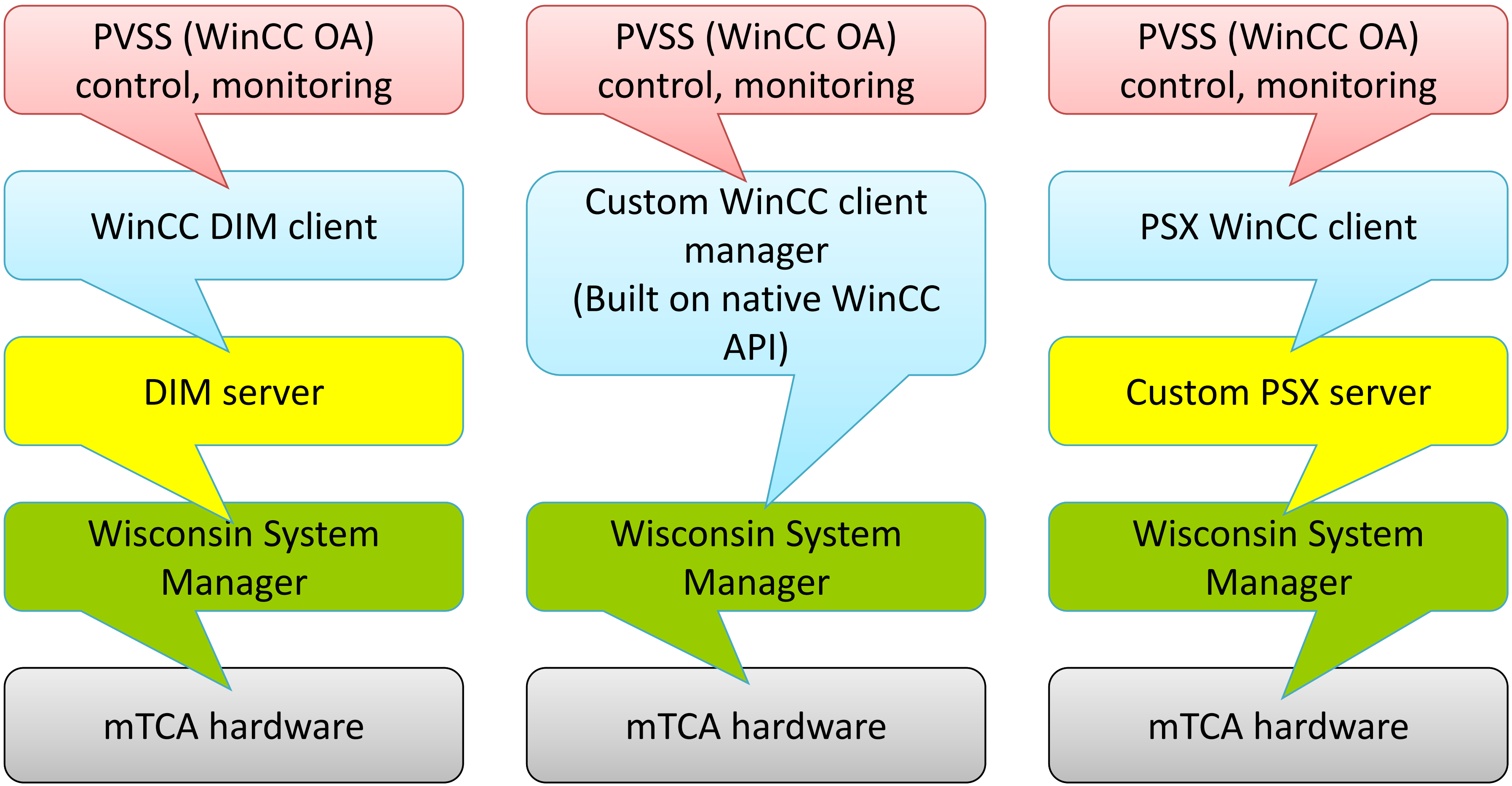
Databases:
Condition DB - Oracle
Configuration DB - Oracle

Questions were to decide:

How to connect WSM to WinCC OA?

How to describe in WinCC a variable hot-swap mTCA configuration?

uTCA SlowControl: Interface – 3 possible solutions



DIM server | Native PVSS API | Custom PSX server

uTCA SlowControl: 3 possible solutions - comparison

Interface	Pro	Contra
DIM Server	Well known in CERN and CMS Good performance Already tested with mTCA and WSM Good integrated development environment – Visual Studio 2010 Supported by Linux and Windows	Additional level in the line of data transfer Complexity in configuring Very hard coded data structure Non-industrial, home made in CERN
Custom PVSS client on native PVSS API	Siemens (ETM) industrial standard Best performance Already tested with mTCA and WSM Direct connection to PVSS datapoints Very simple set of functions (dpSet, dpGet, dpConnect) Good IDE – Visual Studio 2010 Supported by Linux and Windows	Custom PVSS client however built on the industrial standard
Custom server on PSX software	Well known in CMS Good performance Uses the same simple set of functions as native PVSS API Direct (via PSX client) connection to PVSS datapoints	Non-industrial, home made in CMS, supported only by Linux Additional level of data transfer Additional obligatory Linux node Requires xDAQ to be installed

uTCA SlowControl: Data transferring

Roadmap of the data:

mTCA ->System Manager->
DIM server->WinCC OA

```
typedef struct
{
bool enabled;
bool on;
bool hotswap;
bool empty;
float temp1;
float temp2;
float temp3;
float temp4;
float current;
float current1;
bool telcoAlarm;
bool
powerSwitch;
bool ipmbLink;
bool empty1;
} CU_DATA;
```

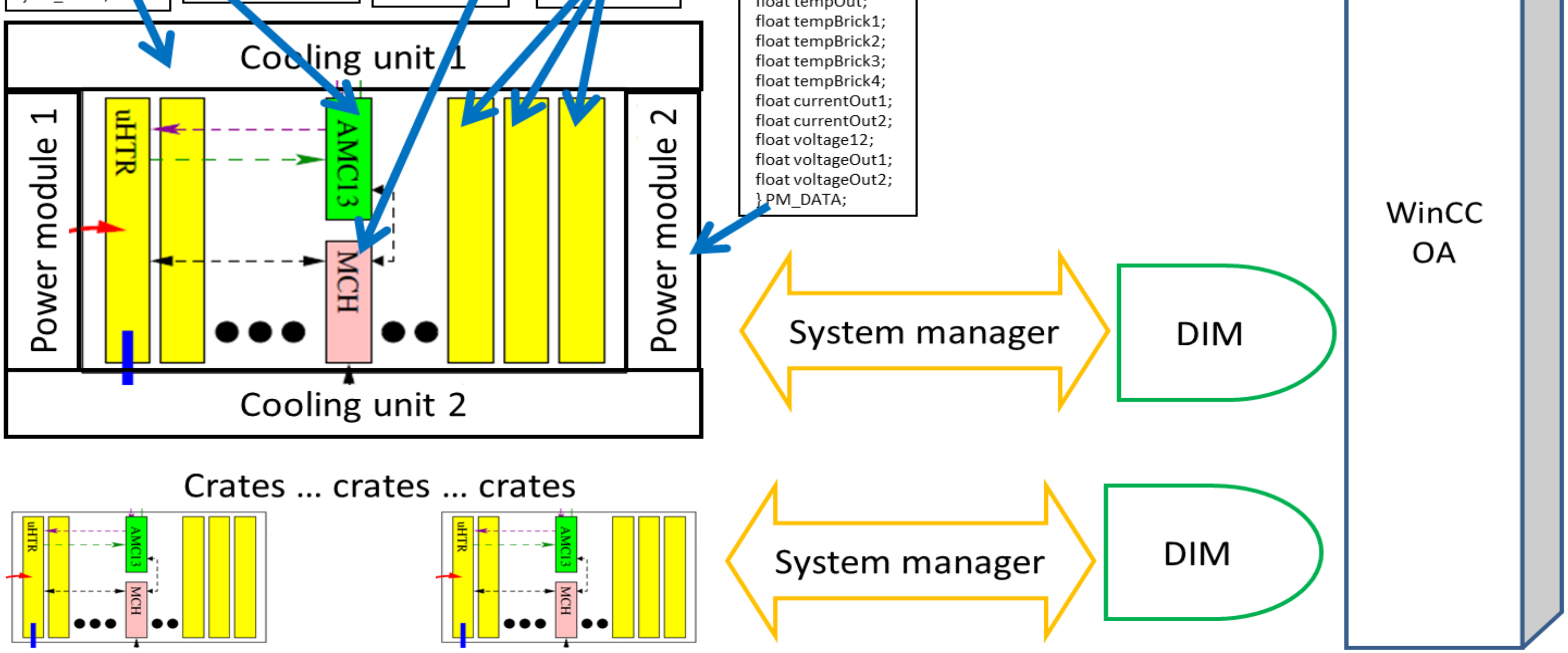
```
typedef struct
{
bool enabled;
bool on;
bool hotswap;
bool empty;
float current;
float current1;
float voltage12;
float tempT2;
float voltage3_3;
float voltage1_2;
bool alarmLevel;
bool powerGood;
bool fpgaConfig;
bool empty1;
} AMC13_DATA;
```

```
typedef struct
{
bool enabled;
bool on;
bool hotswap;
bool empty;
float current;
float current1;
} MCH_DATA;
```

```
typedef struct
{
char name[15];
bool enabled;
bool on;
bool hotswap;
bool empty;
float sensor1;
float sensor2;
float sensor3;
float sensor4;
float sensor5;
float sensor6;

} AMC_DATA;
```

```
typedef struct
{
bool enabled;
bool on;
bool hotswap;
bool empty;
float tempFet;
float tempIn;
float tempOut;
float tempBrick1;
float tempBrick2;
float tempBrick3;
float tempBrick4;
float currentOut1;
float currentOut2;
float voltage12;
float voltageOut1;
float voltageOut2;
} PM_DATA;
```



uTCA Slow Control: Crate data structures

```
typedef struct
{
bool enabled;
bool on;
bool hotswap;
bool empty;
float tempFet;
float tempIn;
float tempOut;
float tempBrick1;
float tempBrick2;
float tempBrick3;
float tempBrick4;
float currentOut1;
float currentOut2;
float voltage12;
float voltageOut1;
float voltageOut2;
} PM_DATA;
```

```
typedef struct
{
bool enabled;
bool on;
bool hotswap;
bool empty;
float temp1;
float temp2;
float temp3;
float temp4;
float current;
float current1;
bool telcoAlarm;
bool powerSwitch;
bool ipmbLink;
bool empty1;
} CU_DATA;
```

```
typedef struct
{
bool enabled;
bool on;
bool hotswap;
bool empty;
float current;
float current1;
float voltage12;
float tempT2;
float voltage3_3;
float voltage1_2;
bool alarmLevel;
bool powerGood;
bool fpgaConfig;
bool empty1;
} AMC13_DATA;
```

```
typedef struct
{
char name[15];
bool enabled;
bool on;
bool hotswap;
bool empty;
float sensor1;
float sensor2;
float sensor3;
float sensor4;
float sensor5;
float sensor6;
} AMC_DATA;
```

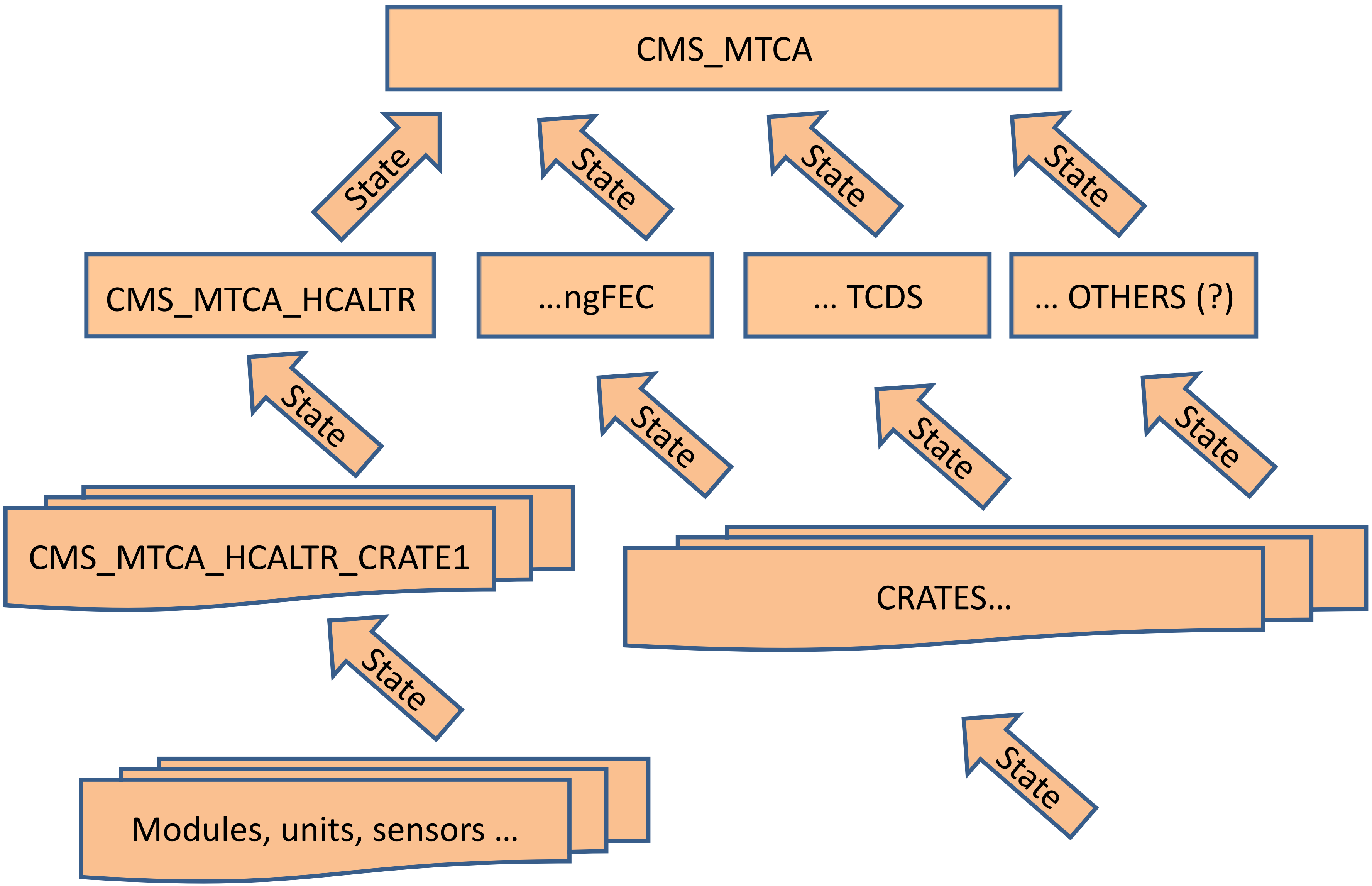
Depends on AMC type

```
typedef struct
{
bool enabled;
bool on;
bool hotswap;
bool empty;
float current;
float current1;
} MCH_DATA;
```

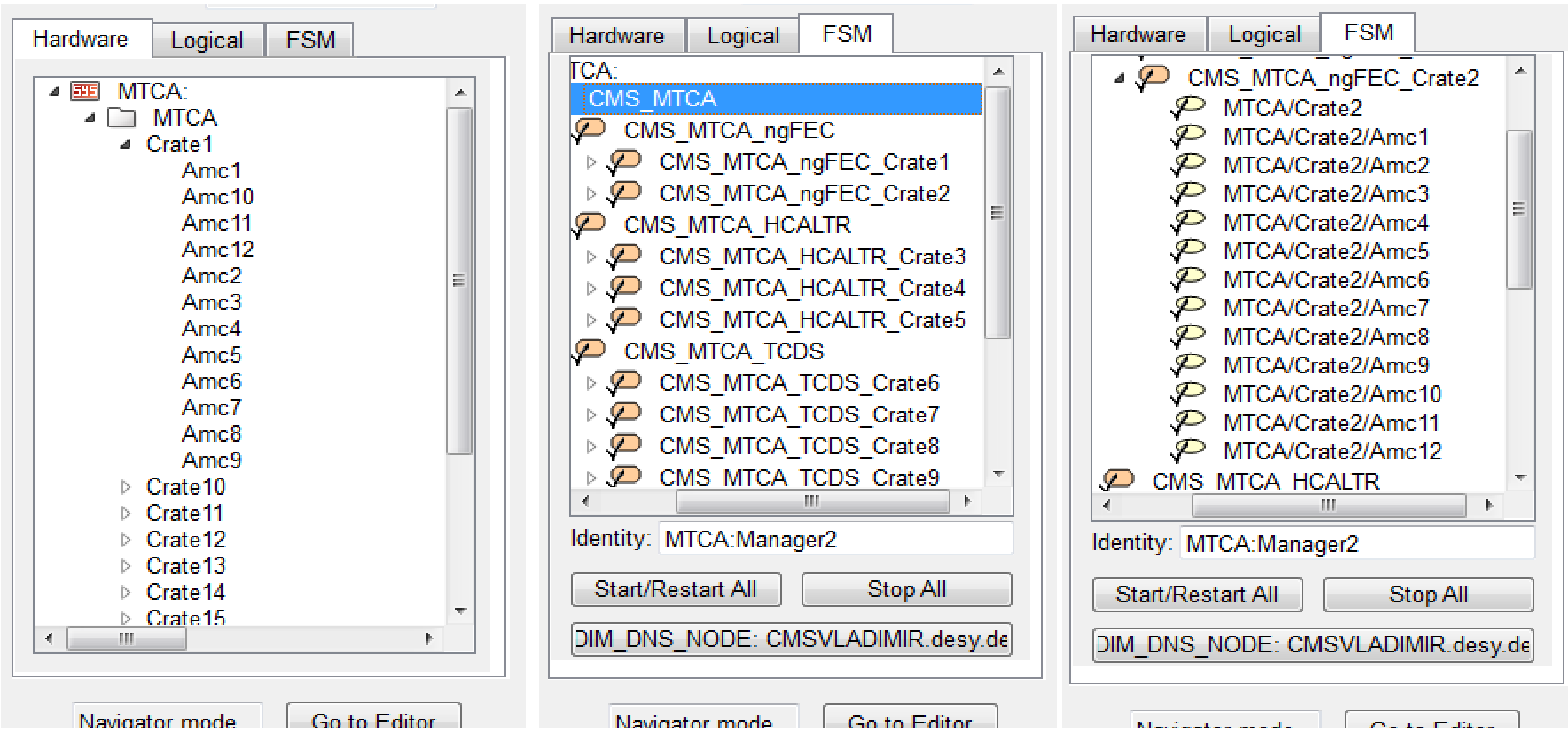
```
typedef struct
{
int number;
MCH_DATA mch;
CU_DATA cu1;
CU_DATA cu2;
PM_DATA pm1;
PM_DATA pm2;
AMC13_DATA amc13;
AMC_DATA amcs[12];
} CRATE_DATA;
```

Information from one crate is joined in one memory block which is transferred to WinCC OA via a subscribed DIM service

uTCA Slow Control: Logical view



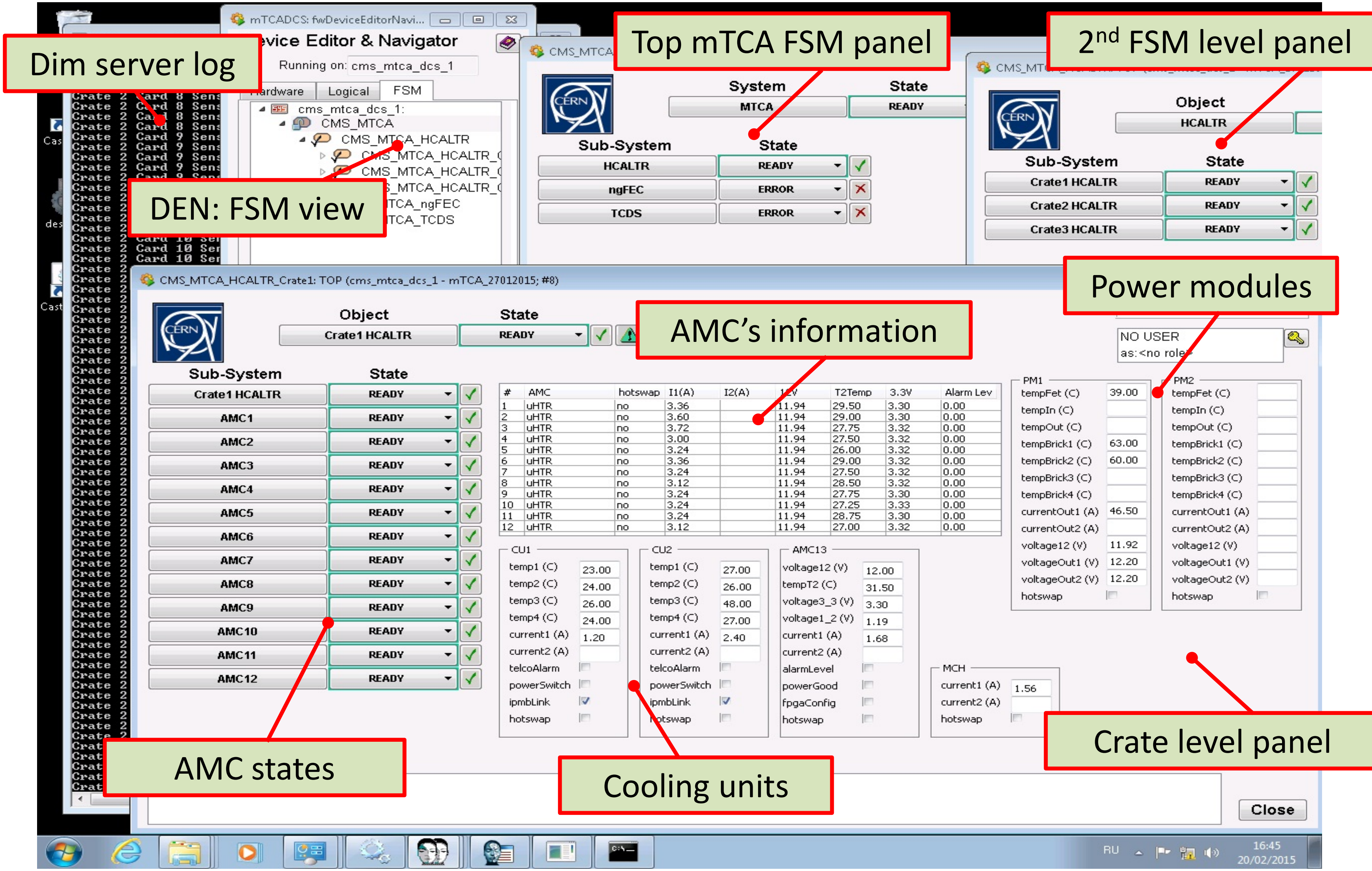
uTCA: Introduction



Hardware tree consists of 15 “crate” datapoints with common crate information (PM, CU, MCH, AMC13) + 12 individual AMC datapoints per crate

FSM tree consists of three **basic branches**: ngFEC, HCALTR, TCDS
Basic branches are divided on **crate branches** with numbering corresponding to the HWTree
Crate branches are also subdivided on a **common crate device unit and individual AMC device unit nodes**
Naming convention is kept according to the CMS DCS guideline

uTCA Slow Control: Examples of visualization panels



uTCA Slow Control: Current status

First production version of mTCA DCS is installed on the CMS DCS production system

- Current configuration:
  FSM tree: HCALTR branch (up to 3 crates)
  ngFEC branch (up to 2 crates)
  TCDS branch (up to 10 crates)
- HCALTR branch (3 crates) already connected to mTCA hardware via System manager on hcalutca01.cms
- Condition DB is connected, data archiving is implemented
- The history of every sensor is recorded in Cond DB
- Dim server – under Linux & Win - is located in regular position
- Transfer rate of sensors information ~ 10 sec/crate
- Beta-version is ready and works - feedback is welcome

uTCA Slow Control: Moving forward…

In order to move the system in full production operation for 2015 – 2016 runs one need:

- To define a final configuration of branches, crates and AMCs
- To connect to TCDS, ngFEC and, maybe, other subsystems
- To realize links from subdetectors FSM panels to mTCA FSM tree for those who need mTCA visual information
- To bring the control of the system to the Central DCS shifter
- To implement alert signals and messages for three levels of severities:
  - Warning
  - Error
  - Fatal

Next step will be to prepare the system for highly increased number of mTCA equipments (up to 50 crates) in future LHC Runs