Lab & beamtest DAQ with XDAQ tools

C.Combaret, L.Mirabito

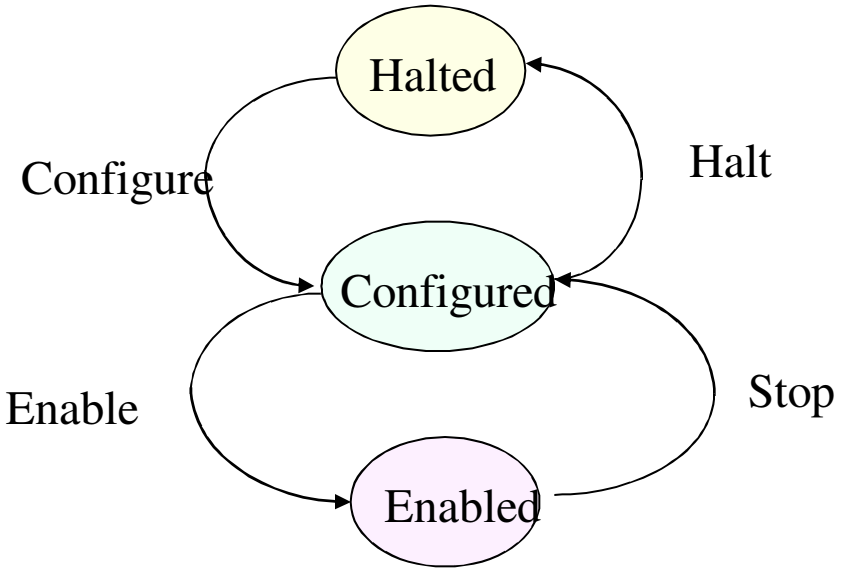
XDAQ

- Core & Powerpack: Already used for
  - Application
  - Transport
  - Web access
  - Final State Machine
- Worksuite: Not yet but …
  - Event Builder
  - Job control
  - Hardware access
  - Error managment

XDAQ basics

- Executives
  - Main XDAQ process
    - Support http: Start a web server on a given port, configuration via XML
      - Lyoac22> xdaq.exe –p 5000 –c mydaq.xml → accessible on
        - http://lyoac22:5000/
    - Additionnal transport (tcp,atcp) for binary exchange
    - Dynamic loading of shared libraries (application)
- Applications
  - Load dynamically as a shared library (XML tag in the executive configuration)
  - Each application acts as a web service
    - Embedded web page definition or Web 2 acces
      - GUI for hardware access is visible on the web
      - Curl configuration
    - Accept or emit SOAP messages
      - Remote control of command
  - It can be declared on the tcp transport and exchange binary message with other applications
    - Inside the executive: zero-copy
    - On the same PC: unix sockets
    - On the network: tcp sockets

The Final State Machine



- Binding SOAP
  - Simple manager can be either external or include in one of the application
- Use of CMS run control for larger configuration

An example: TSCSupervisor application

- Configuration

```
TSCSupervisor::TSCSupervisor(xdaq::application* stub): public WebApplication(stub)
…
deviceNumber=0;
getApplicationInfoSpace()->fireItemAvailable("device",&deviceNumber_);
…
// Define FSM
        fsm_.addState ('H', "Halted");
        fsm_.addState ('R', "Ready");
        fsm_.addState ('E', "Enabled");
        fsm_.addStateTransition ('H','R', "Configure", this, &TSCSupervisor::ConfigureAction);
        fsm_.addStateTransition ('R','E', "Enable", this, &TSCSupervisor::EnableAction);
        fsm_.addStateTransition ('E','H', "Halt", this, &TSCSupervisor::HaltAction);
        fsm_.addStateTransition ('R','H', "Halt", this, &TSCSupervisor::HaltAction);
        fsm_.setFailedStateTransitionAction( this, &TSCSupervisor::failedTransition );

        fsm_.setInitialState('H');
        fsm_.reset();

        // Bind SOAP callbacks for control messages
        xoap::bind (this, &TSCSupervisor::fireEvent, "Configure", XDAQ_NS_URI);
        xoap::bind (this, &TSCSupervisor::fireEvent, "Enable", XDAQ_NS_URI);
        xoap::bind (this, &TSCSupervisor::fireEvent, "Halt", XDAQ_NS_URI);


…

            // Bind CGI callbacks
                xgi::bind(this, &TSCSupervisor::latencyGet, "latencyGet");
```

Web access

Parameters Configurable in XML or via SOAP

FSM binded to Application methods

FSM binded to SOAP commands

Web callback definition

Example(2):Defining an HTML form

```
void TSCSupervisor::displayHaltedPage(xgi::Input * in, xgi::Output * out)
    {
     xgi::Utils::getPageHeader(*out, "Configure");

     std::string url = "/";
     url += getApplicationDescriptor()->getURN();
     url += "/supervisorConfigure »;
     *out << cgicc::form().set("method","post")
       .set("action", url)
       .set("enctype","multipart/form-data").set("style","font-size: 10pt;  font-family: arial;") << std::endl;

     *out << cgicc::fieldset().set("style","font-size: 10pt;  font-family: arial;") <<  cgicc::p() << std::endl;
     *out << cgicc::legend("Streamer configuration") << std::endl;
     *out << cgicc::label("Configuration XML url ") << std::endl;
     *out << cgicc::input().set("type","text").set("name","urlXmlConfig").set("size","60").set("value",urlXmlConfig_)
    <<  cgicc::p() <<  std::endl;

     *out << cgicc::fieldset() << std::endl;
     *out << cgicc::input().set("type", "submit")
       .set("name", "submit")
       .set("value", "Configure");

     *out << cgicc::p() << std::endl;
     *out << cgicc::form() << std::endl;
     xgi::Utils::getPageFooter(*out);
    }
```

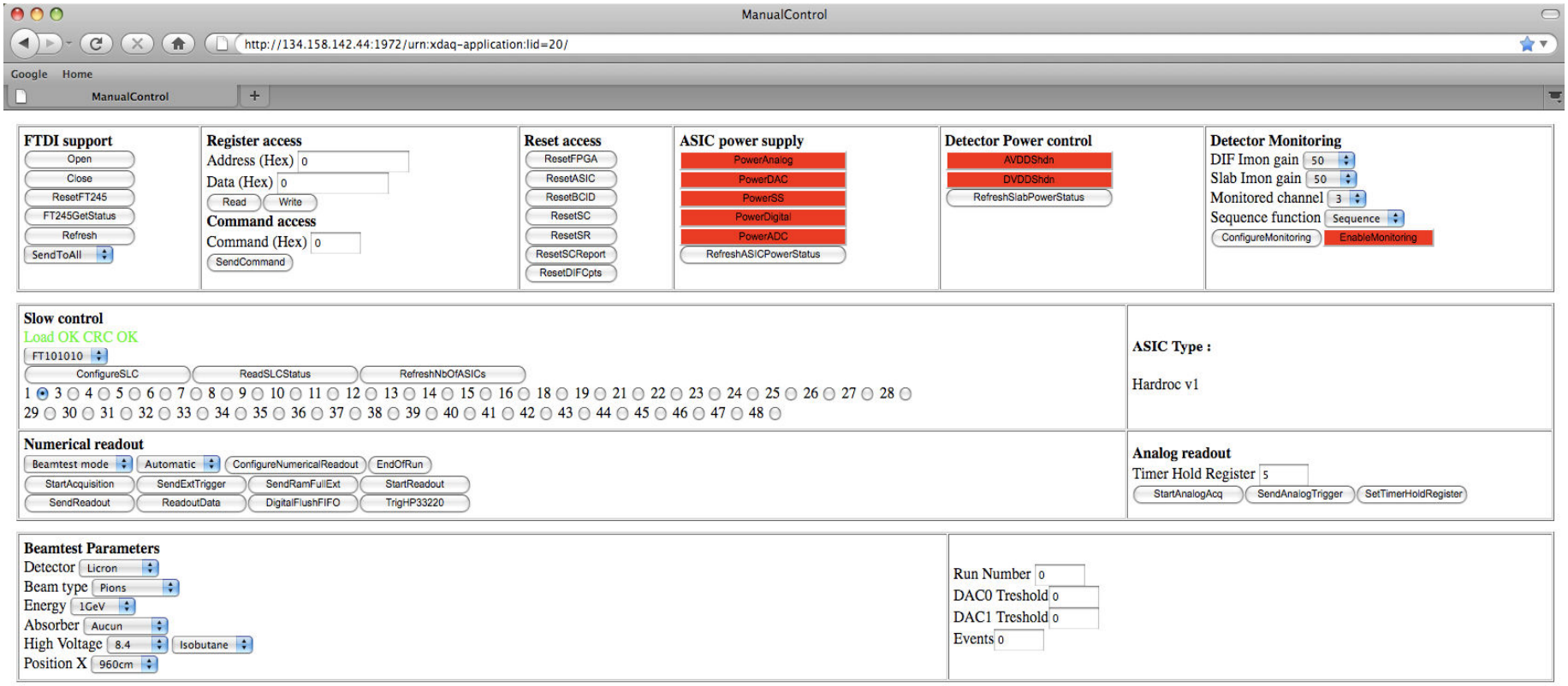
Example(3): handling the form

```
void TSCSupervisor::supervisorConfigure(xgi::Input * in, xgi::Output * out) throw (xgi::exception::Exception)
        {
         try
           {
            // Create a new Cgicc object containing all the CGI data
            cgicc::Cgicc cgi(in);

            urlXmlConfig_ = cgi["urlXmlConfig"]->getValue();
           }
         catch(const std::exception& e)
           {
            XCEPT_RAISE(xgi::exception::Exception,  e.what());
           }
         try
           {
            toolbox::Event::Reference e(new toolbox::Event("Configure",this));
            fsm_.fireEvent(e);
           }
         catch (toolbox::fsm::exception::Exception & e)
           {
            XCEPT_RETHROW(xgi::exception::Exception, "invalid command", e);
           }

         this->Default(in,out);
        }
```

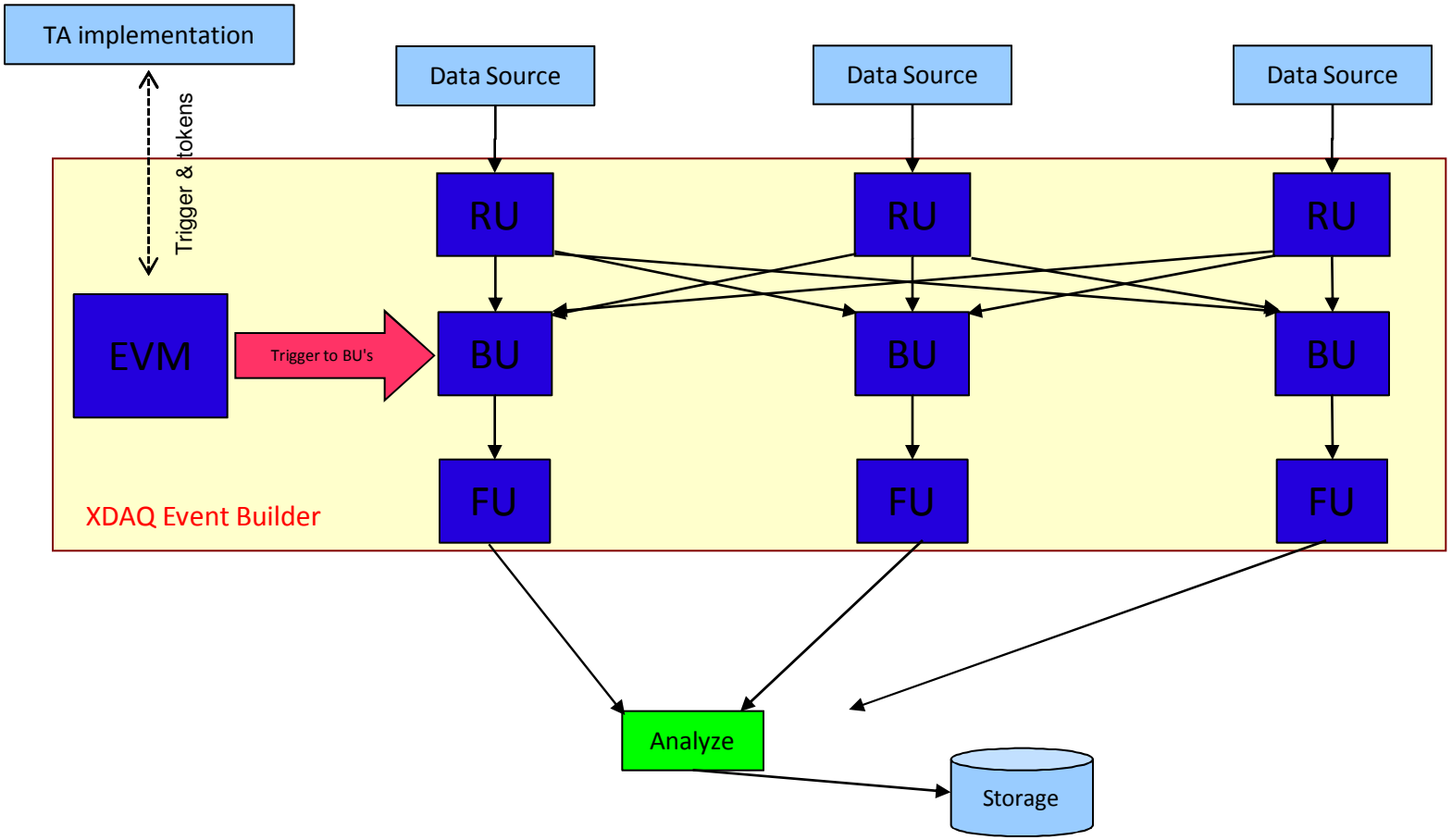
DIFSupervisor example



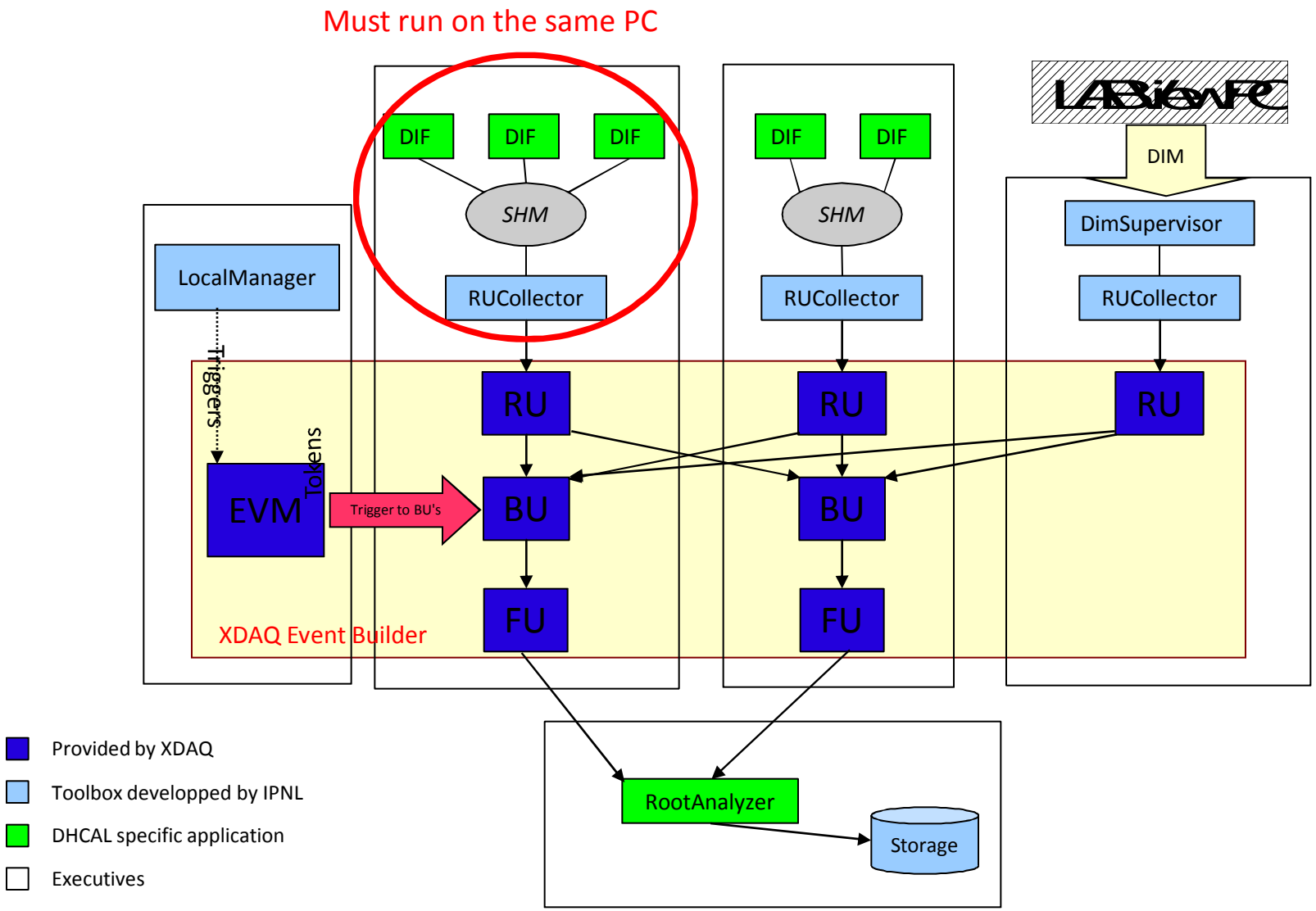
The Event Builder

- Components
  - Readout Unit (**RU**): it collects data from any source with predefined message format and buffered it in a FIFO, it will feed 1 Builder Unit on trigger request
  - Builder Unit (**BU**): it collects data from all RU's of a given partition for a given trigger. It assembles data fragment, buffers the event and feed Filter Units registered to it (tokens)
  - Filter Unit (**FU**): it processes build event. It has some processing slots and provides tokens to its associated BU according to free slots.
  - Event Manager (**EVM**): it controls the whole system. It receives trigger from an external source (Trigger Accepter) and triggers the transfer of RU buffers to BU. Once event built it receives a token from the BU it forwards to the TA to allow further triggers
  - Trigger Accepter (TA): Trigger control with predefined messages with the EVM in order to receive tokens and send triggers

The Event Builder (2)



A detailed example



Additional applications

- RUCollector: It collects data via share memories and format the message to the RU. A Share memory class is provided to be used in the DIF/LDA readout.
- LocalManager: It's the minimal implementation of the TA. It only periodically send trigger messages to the EVM if enough tokens are received from it. It crosschecks the number of event collected by each RUCollector.
- DimSupervisor: Simple interface between a DIM data source and the RUCollector. DIM is supported in LabView, PVSS…
- RootAnalyzer: DHCAL online monitoring and ROOT based data storage. Not distributed, it collects data from all FU's

Beam test DAQ

- Trigger based read out
  - It requires a real trigger board interfaced to the LocalManager:
    - Send N triggers to the DIF's on LocalManager request
    - Back pressure on DIF/LDA vetos
  - IO board, Trigger Sequencer Card developped at IPNL (CMS,ALICE) can be used. CCC ?
- Analysis has to be developped for multi-planes monitoring
- LDA supervisor application needed

To go further

- Current DAQ are smalls
  - Few PC's and processes can be described in small XML files.
  - Executives can be started manually
- Larger system
  - DB storage of applications configuration
  - Jobcontrol services on PC's to start the executives
  - Configuration and FSM handling via Run Control, TOMCAT + java servlets interfaced to XDAQ
  - Distributed analysis in FU
- Integrate the control
  - Calibration runs
    - LocalManager Loop: BlockTrigger->Change parameter->Acquire N trigger….
- Error handling
  - Several error handling framework available in XDAQ

Maintenance & deployment

- XDAQ is the CMS data acquisition framework and will be supported in the coming years. Code is free.
- Binaries are provided as RPM's on Scientific Linux CERN releases (yum installation). Re compilation is needed for other Linux platform.
- IPNL add-ons can be provided as source or RPM's if needed

Summary

- XDAQ was already used at IPNL to configure and read DIF's
  - Web access, private event building and storage
- The CMS Si-tracker acquisition software was simplified to allow the use of the Event Builder with the DIF applications
  - Other data source board can be easily added
  - Scalability of the architecture is demonstrated with CMS tracker local daq
    - 500 boards,40 PC's, 1000's of processes
- Additional applications (DIM interface, Analysis) were also developped to ease laboratory tests
- Usage in beamtests of such DAQ is straightforward
  - Trigger card needed
  - Port of other Data Source boards software to XDAQ