Web Services

CMS and Agata Run Control
The GRIDCC Project

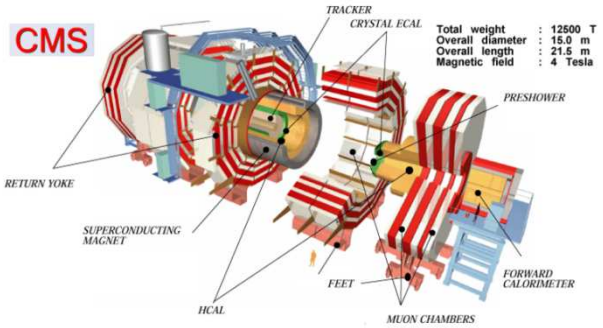
Michele Gulmini
INFN Laboratori Nazionali di Legnaro

Incontro CCR - December 10, 2008

Contents

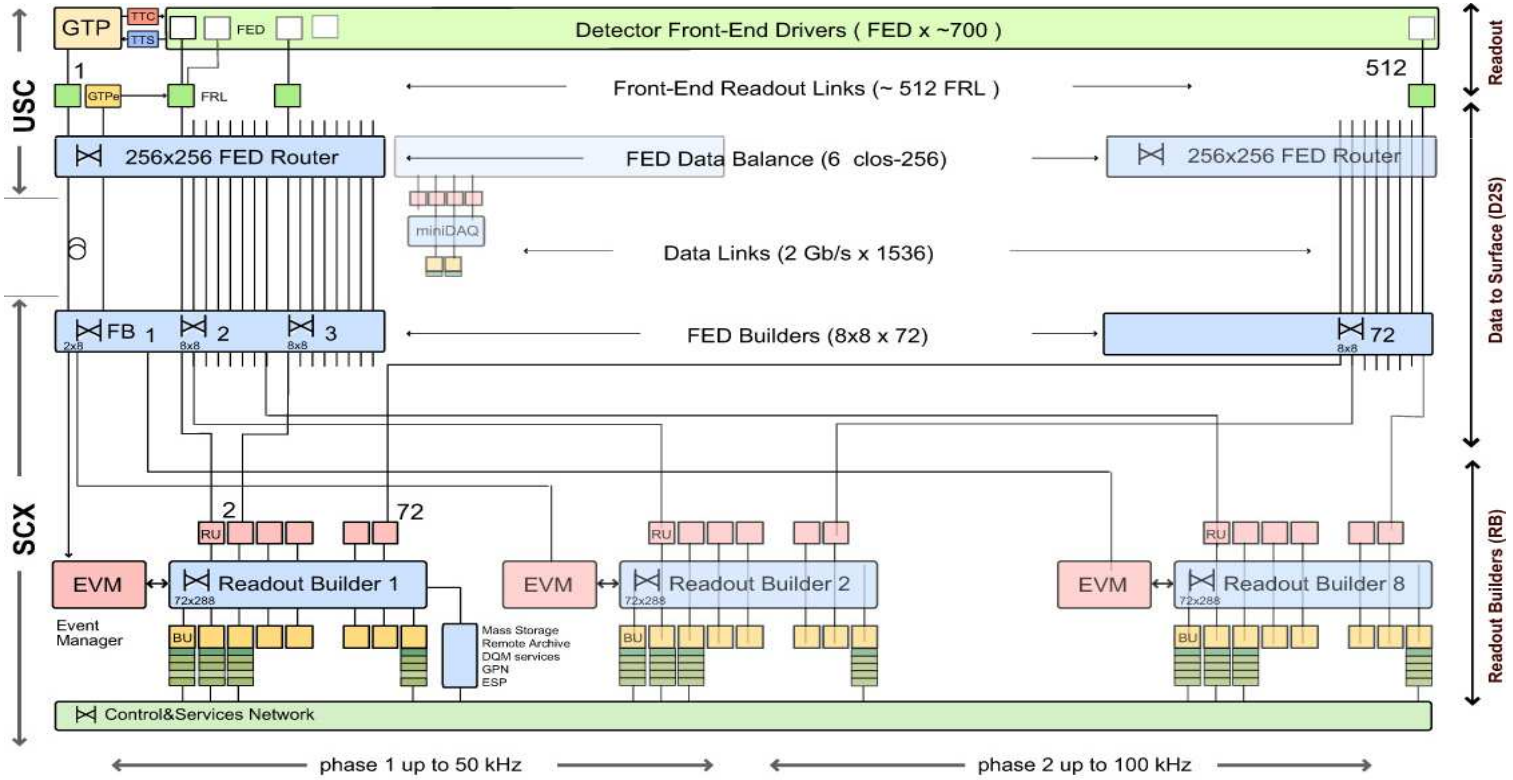
- The CMS Run Control and Monitor System
  - Service Oriented Architecture
- Developing Web Services
  - Server side experience
- The GRIDCC project
- The Agata Run Control
- Developing Web Services
  - Client side experience

CMS

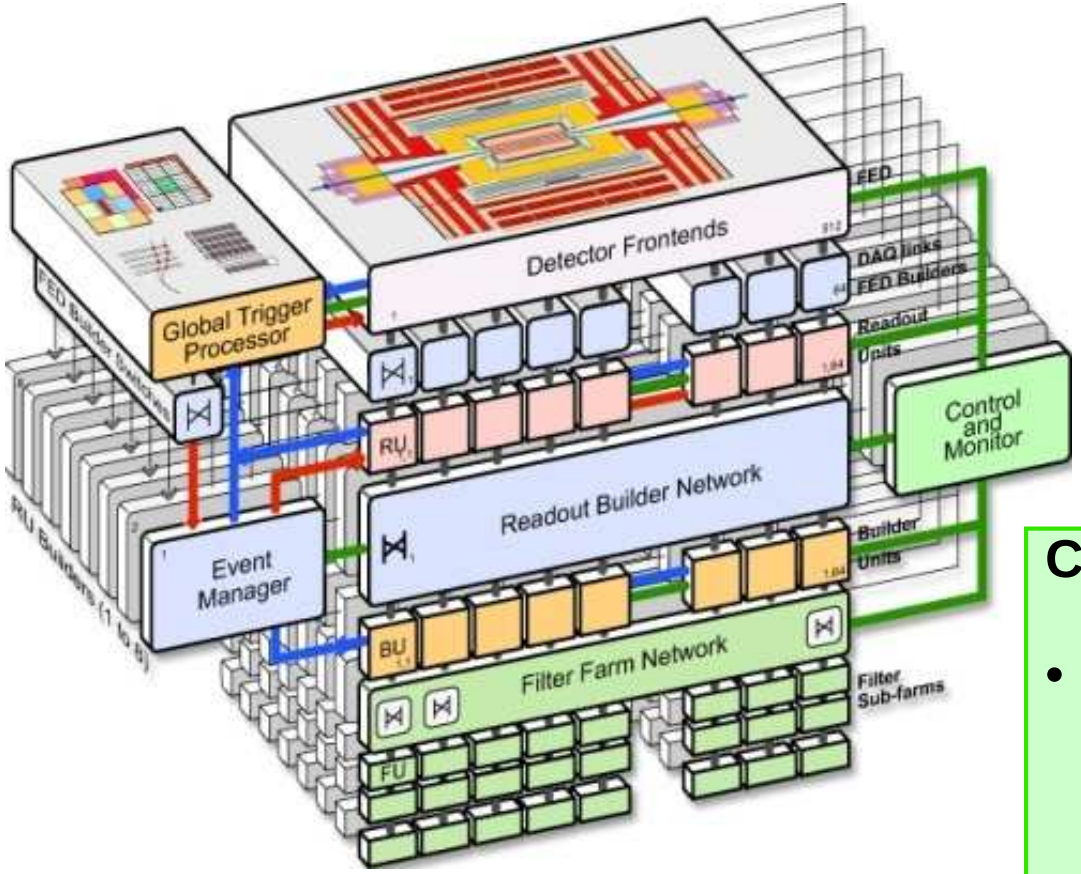


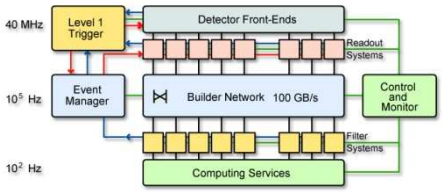
Total weight: 12500 T
Overall diameter: 15.0 m
Overall length: 21,5 m
Magnetic field: 4 Tesla

DAQ TDR Design



Control and Monitor Requirements





Baseline DAQ Configuration

- 512 inputs
- 2024 outputs

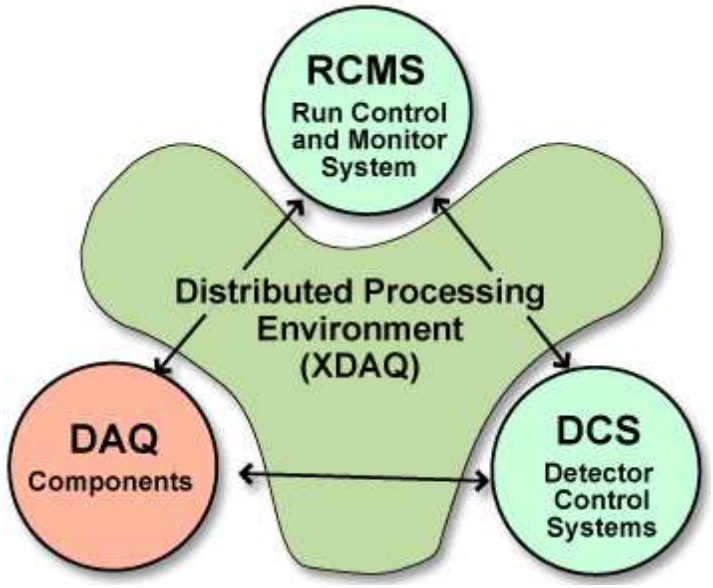
Control and Monitor requirements

- O(10^4) distributed Objects to
  - control
  - configure
  - monitor
- On-line diagnostics
- Interactive system

Run Control and Monitor System

RCMS is integrated in the CMS On-line system :

- It controls the “DAQ component”
  - Data transport
  - Event processing
- It monitors the “Detector Control System” DCS
  - manages the slow controls of the whole experiment.



The SOAP protocol and the Web Services have been adopted as the main means for communication .

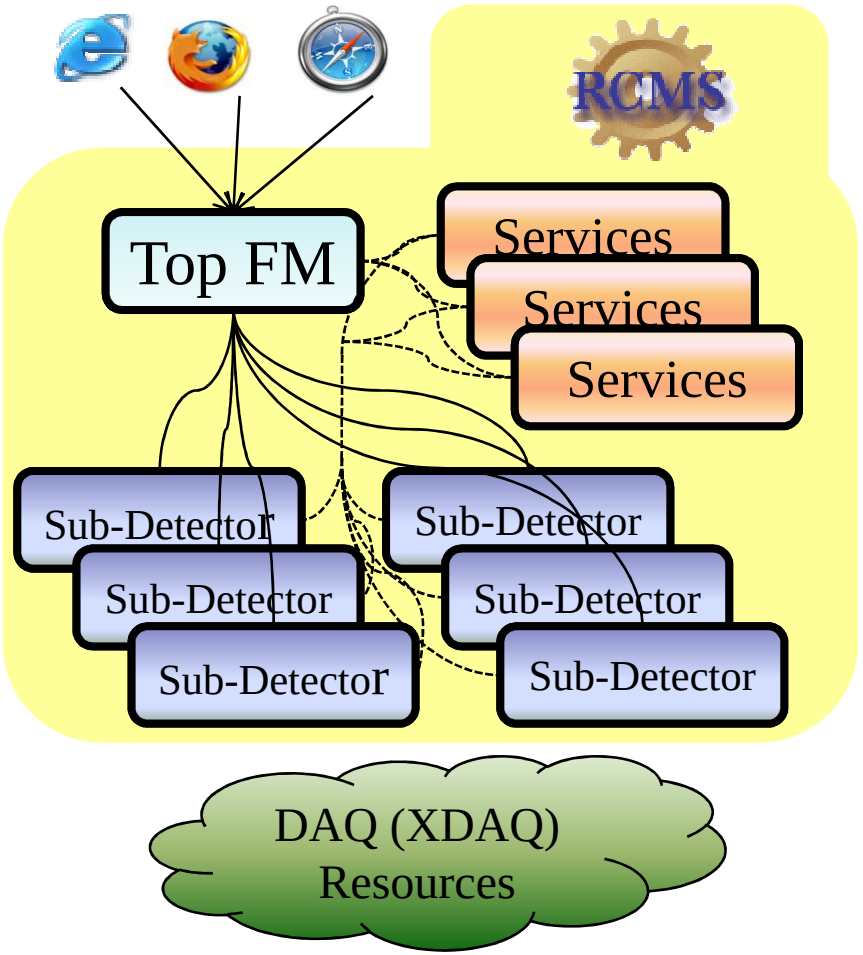
The online process environment is XDAQ, a C++ framework for a distributed Data Acquisition System.

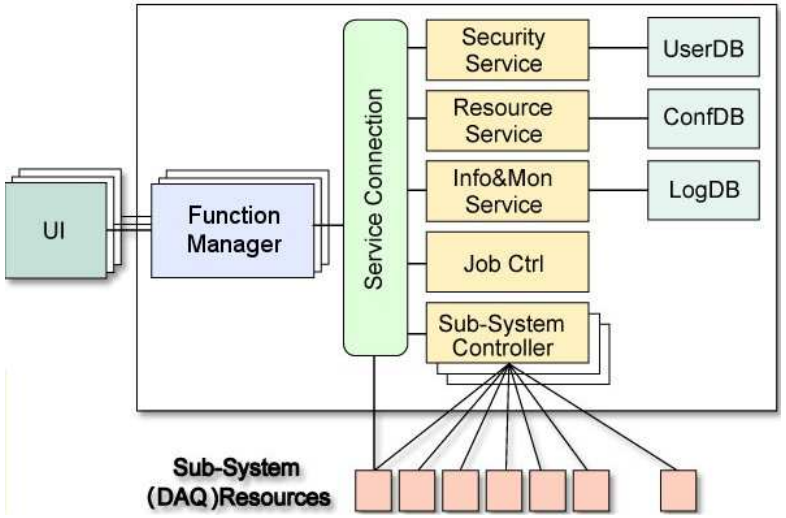
RCMS – Run Control and Monitor System

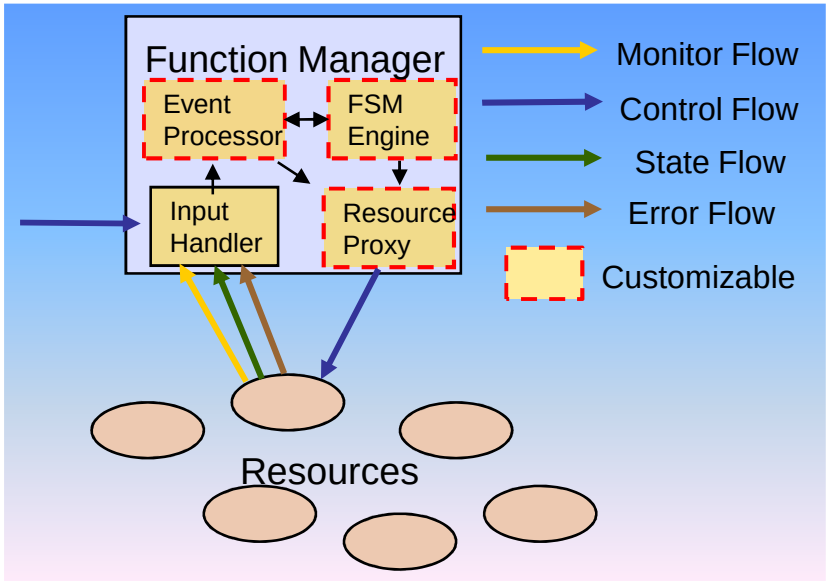
Service Oriented Architecture

- **1998**: **Java** and **CORBA** (Common Request Broker Architecture) as communication backbone
  - Used in test-beams and DT chambers production centers
- **2001**: **HTTP + XML** communication (JAXM)
  - **Tomcat** application server
  - Java applet GUI
- **2003**: Adoption of **SOAP** as control communication backbone
  - First RCMS official release (RCMS 1)
  - "DAQKit" for CMS sub-detector DAQ teams
  - Used in test-beams and production centers
- **2005**: **WSDL** and Apache **Axis**
  - RCMS 2
  - JSP based GUI
  - Sub-detector local DAQs, MTCC I&II (2006)
- **Sept 2008**: first LHC ~~collision~~ beam
  - RCMS 3: Tomcat 5.5, Axis 1.4

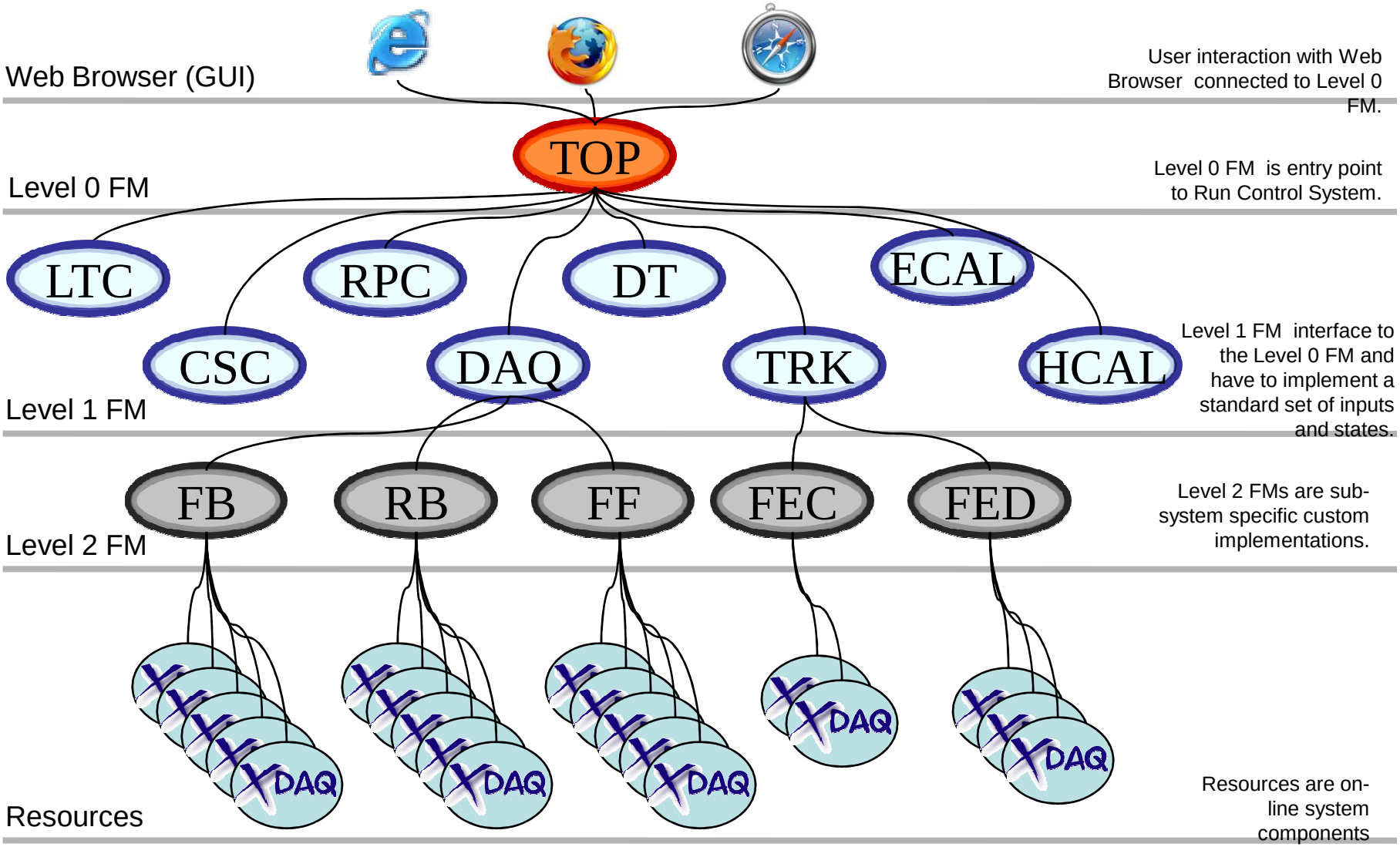
RCMS Architecture







Control Structure at MTCC I & II (2006)



RCMS & Web Services – Why?

- TDR(2002):
  - *it is planned to maximise the use of **standard** software technologies in the RCMS.*
  - ***Web technologies** and in particular **Web Services** have been chosen because they have been designed to interconnect highly heterogeneous and distributed systems, therefore implying a rich choice of available tools and solutions.*
  - *Many of these widely-spread software tools follow an **open-source** approach, thus facilitating the **integration** with other software packages.*

- Central DAQ provides tools to the sub-detector DAQ teams to develop their own local DAQ and control systems (XDAQ and RCMS).
- Adoption of XDAQ and RCMS (DAQKit) in the sub-detectors makes the integration of the overall CMS DAQ easier.
- Adoption of web services standards (WSDL and SOAP) helps sub-detector custom developments integration.

Web Services:

Server Side Development

Web Services: the beginning

- **WSDL** Experience in CMS started in 2004
  - Major revisit of all CMS DAQ software (XDAQ 3, RCMS2 …)
- Sun **J2EE** (Java 2 Enterprise Edition) tutorial is probably the best way to start learning and working with Java Web Services
  - Web Services are a sub-set of the J2EE specifications
- Apache Axis documentation is also quite good
- A (quick) investigation was done:
  - Apache Tomcat / Axis 1.2
  - Sun JWSDP 1.4 (Java Web Service Development Package)
    - Uses Tomcat
  - Sun Java System Application Server 8
- **Interoperability tests**:
  - Web Service interface defined in Java
  - WSDL generated from Java by using “Java2WSDL” tools (available both for Sun and Axis)

Apache Axis & Sun JWSDP

- Web Application Life Cycle**:**
  - 1. **Develop the** web component **code**.
  - 2. **Develop the** web application **deployment descriptor.**
  - 5. **Deploy** the application into a **web container**.

- 1) **Java Code**:
  - (almost) 100% reusable
    - If careful to use only “standard” (JAX-RPC) APIs
    - Examples provided by Axis and Sun often use “custom” classes (API implementations)
      - javax.xml.rpc.* classes – OK
      - org.apache.axis.* classes – NO (for code reuse)
- 2) **Deployment descriptors:**
  - Custom (deploy.wsdd, sun-<module-type>.xml, …)
- 5) **Web container installation, configuration, maintenance**
  - Custom

Full interoperability was demonstrated

Tomcat/Axis Java – The choice

- Had already experience with Apache Tomcat
  - Open source and free to use
  - already powering numerous large-scale, mission-critical web applications
- The learning curve of a new Application Server in not deep
  - Sun Application Server is quite complex
  - JWSDP Reference Implementation was using Tomcat
- Axis has a very active user community and was already adopted by many companies
  - implements a sub-set of the J2EE functionalities
- Migration to another J2EE Application Server does not imply java code changes
- Links: http://ws.apache.org/axis/ , http://tomcat.apache.org/

WSDL and Java Interface

- **WSDL is complex**
  - It is not easy to write it by hand.
  - It is more comfortable (easier and faster) to define the services API in Java and then generate the WSDL (*Java2WSDL* tools)

- **Java to WSDL mapping - Limitations**
  - *"Behind the scenes, JAX-RPC maps types of the Java programming language to XML/WSDL definitions. For example, JAX-RPC maps the* ***java.lang.String*** *class to the* ***xsd:string*** *XML data type. Application developers don't need to know the details of these mappings, but they should be aware that* ***not every class in the Java 2 Platform, Standard Edition (J2SE) can be used as a method parameter or return type in JAX-RPC****".*
  - Supported types:
    - *boolean, int, …*
    - *java.lang.Boolean, java.lang.Integer,…*
    - *Arrays: boolean[], int[], …*
    - *Java Beans*

WSDL and Java Exceptions

- **Exceptions** *(Axis documentation)*
  - *This is an area which causes **plenty of confusion**, and indeed, the author of this section is not entirely sure how everything works, especially from an interop perspective.*
  - **RemoteExceptions map to SOAP Faults**
    - *If the server method throws a java.rmi.RemoteException then this will be mapped into a SOAP Fault. The faultcode of this will contain the classname of the fault. …If the recipient does not know how to create an instance of the received fault, this mechanism does not work. …you can only reliably throw java.rmi.RemoteException instances, rather than subclasses.*
    - *When an implementation in another language receives such an exception, it should see the name of the class as the faultCode, but still be left to parse the body of the exception. **You need to experiment to find out what happens there**.*
  - **Exceptions are represented as wsdl:fault elements**
    - *If a method is marked as throwing an Exception that is not an instance or a subclass of java.rmi.RemoteException…The exception is no longer a SOAP Fault, but described as a wsdl:fault in the WSDL of the method. According to the JAX-RPC specification, your subclass of Exception must have accessor methods …*
    - *If your exception meets this specification, then the WSDL describing the method will describe the exception too, enabling callers to create stub implementations of the exception, regardless of platform.*
    - ***Again, to be sure of interoperability, you need to be experiment a bit**. Remember, **the calling language may not have the notion of Exceptions**, or at least not be as rigorous as Java in the rules as to how exceptions must be handled.*

Example: An RCMS Exception

- All RCMS service exceptions extend *java.rmi.RemoteException*
  - Reduce interoperability problems

```
package rcms.fm.fw.service.command;

import java.io.Serializable;
import java.rmi.RemoteException;

/**
 * Command Service Exception class.
 *
 */
public class CommandServiceException extends RemoteException implements Serializable {
…
                public CommandServiceException() {}
                public CommandServiceException(String s) {…}
                public CommandServiceException(String s, Throwable ex) {…}

                public Throwable getException() {return exception;}
                public void setException(Throwable exception) {…}
                public String getMsg() {return msg;}

                public void setMsg(String msg) {
                                this.msg = msg;
                }
}
```

Development Environment

- Eclipse Integrated Development Environment (www.eclipse.org)

- Eclipse Plugins (http://www.eclipseplugincentral.com/)
  - Sysdeo Tomcat Launcher plugin
  - Lomboz plugin (J2EE applications)

- Xdoclet to ease the deployment of the services
  - http://xdoclet.sourceforge.net
  - Mainly in the first phase we needed to change often the service API.
  - Axis deployment descriptors automatically generated
    - *XDoclet is an open source code generation engine. It enables Attribute-Oriented Programming for java. In short, this means that you can add more significance to your code by adding meta data (attributes) to your java sources. This is done in special JavaDoc tags.*

Example: an Axis service

```
package rcms.fm.ws.command;
import javax.xml.rpc.ServiceException;
…
/**
 * CommandService Web Service.
 *
 * @axis.service name="CommandService" scope="Application“ enable-remote-admin="true"
 *
 * @axis.bean name="CommandServiceException" package="rcms.fm.fw.service.command"
 * @axis.bean name="StateMachineBean“  package="rcms.statemachine.definition.bean"
…
 */
public class FMCommand implements ServiceLifecycle,CommandServiceIF {

        private CommandService commandService = null;

        /**
         * Synchronous commad execution….
         * …
         * @axis.method
         * @param command the command to execute
         * @param uriPath the URI identifying a resource
        *  @return the state after command execution
        *  @throws CommandServiceException
        */
        public StateBean execute(String[] uriPath, CommandBean command) throws
                        CommandServiceException {
        …
        }
…
}
```

Ant and XDoclet

```
<!--Build file for deploying and undeploying RCMS web services into axis and make stubs.
This file is only used by RCMS developers.-->

<project name="rcms-parameter-deploy" default="axis-deploy" basedir=".">
        <property file="build.properties" />
…
        <!-- Generate the axis deployment descriptor -->
        <target name="axisdoclet">
                <taskdef name="templatedoclet" classname="xdoclet.DocletTask"
classpathref="xdoclet.class.path" />
                <templatedoclet destdir="${output}" verbose="1">
                   <template templateFile="${deploy.xdoclet.template}" destinationFile="${deploy.wsdd}">
                        <configParam name="Xmlencoding" value="utf-8" />
                   </template>
                </templatedoclet>
        </target>
        <!-- Generate the axis undeployment descriptor -->
…
        <!-- Deploy the services into axis engine -->
        <target name="axis-deploy" depends="axisdoclet,axisdoclet-undeploy">
                <taskdef resource="axis-tasks.properties" classpathref="axis.classpath" />
                <axis-admin port="${target.port}" hostname="${target.server}" failonerror="true"
        servletpath="${target.appname}/services/AdminService" debug="true" xmlfile="${output}${deploy.wsdd}" />
        </target>
        <!-- Undeploy the services into axis -->
…
        <!-- Generate the stubs  -->
…
</project>
```

Example: Axis deployment descriptors

```
<?xml version="1.0" encoding="utf-8"?>
<deployment xmlns=http://xml.apache.org/axis/wsdd/  xmlns:java=http://xml.apache.org/axis/wsdd/providers/java
                xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
                <service name="CommandService“ provider="java:RPC">
                                <parameter name="className" value="rcms.fm.ws.command.FMCommand" />
                                <!-- check if remoteAdmin param is present -->
                                <parameter name="enableRemoteAdmin" value="true" />
                                <parameter name="allowedMethods"
                                                value="execute getState getUpdatedState getStateTree getUpdatedStateTree
                                                                getStateMachine getStateList "/>
                                <parameter name="scope" value="Application"/>

                                <beanMapping qname="ns:CommandServiceException" (Axis documentation)
                                                xmlns:ns="urn:FMCommand"
                                                languageSpecificType="java:rcms.fm.fw.service.command.CommandServiceException"
                                />
                                <beanMapping qname="ns:StateMachineBean"
                                                 xmlns:ns="urn:FMCommand"
                                                languageSpecificType="java:rcms.statemachine.definition.bean.StateMachineBean"
                                />
                                …
                </service>
</deployment>
```

```
<?xml version="1.0" encoding="utf-8"?>
<undeployment xmlns=http://xml.apache.org/axis/wsdd/ xmlns:java=http://xml.apache.org/axis/wsdd/providers/java
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">

  <service name="CommandService">
  </service>
</undeployment>
```

The GRIDCC Project

http://www.gridcc.org

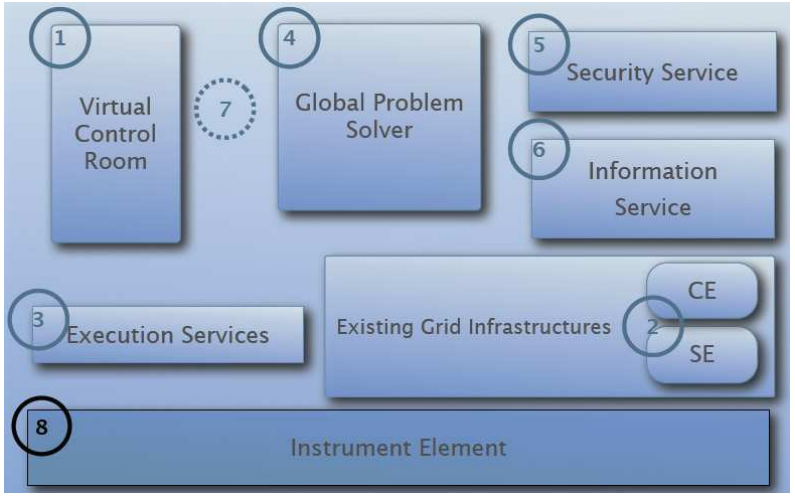
RCMS and GRIDCC

The ***G**rid enabled **R**emote **I**nstrumentation with **D**istributed **C**ontrol and **C**omputation* (**GRIDCC**) is a project funded by the European community, aimed to provide access to and control of distributed complex instrumentation.

- It is a 3-years project started in September 2004

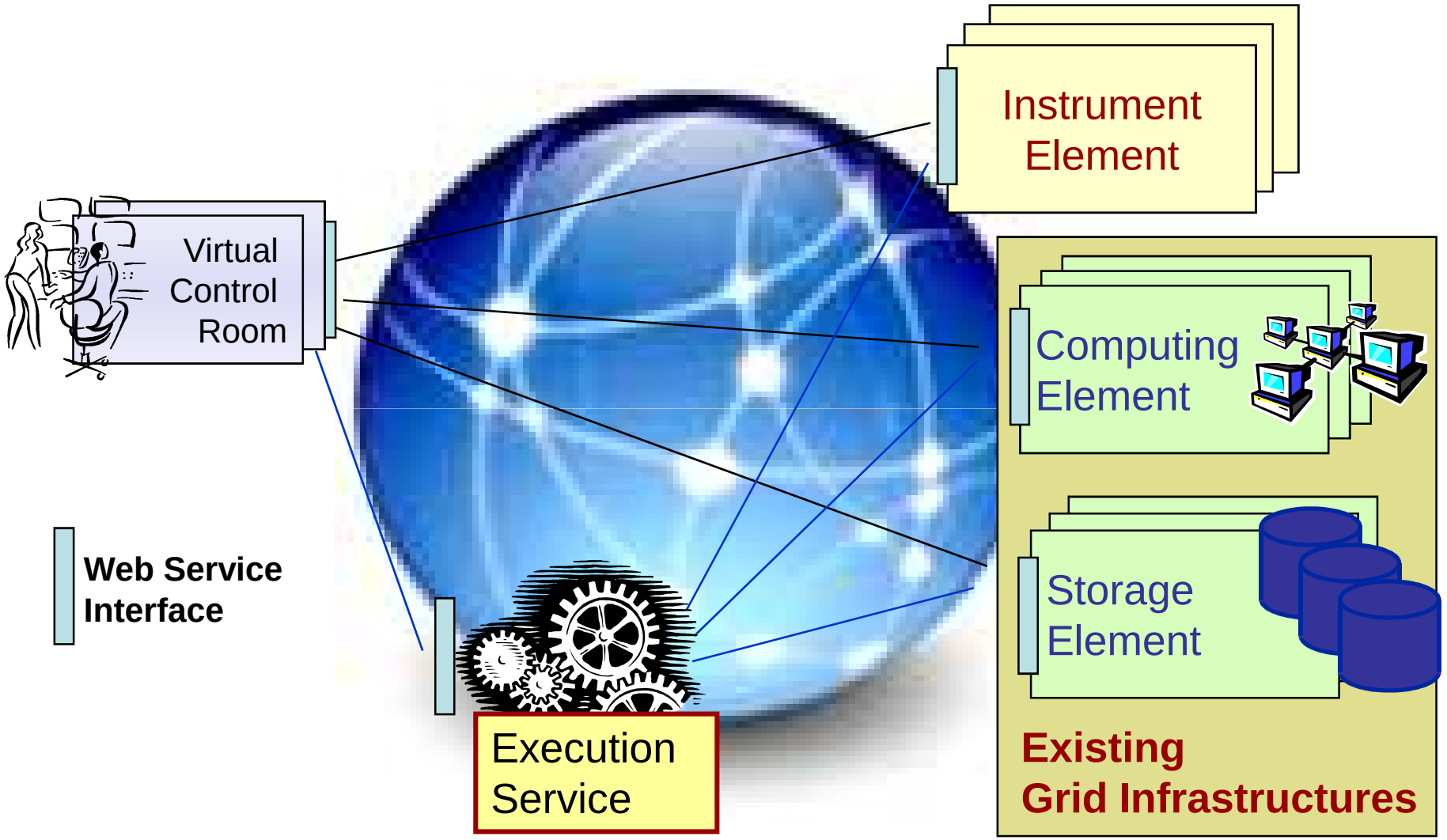
The RCMS software is the core of the Instrument Element of the GRIDCC.

The RCMS software was developed in closed collaboration with the GRIDCC.



CMS is one of the main applications for the GRIDCC project .

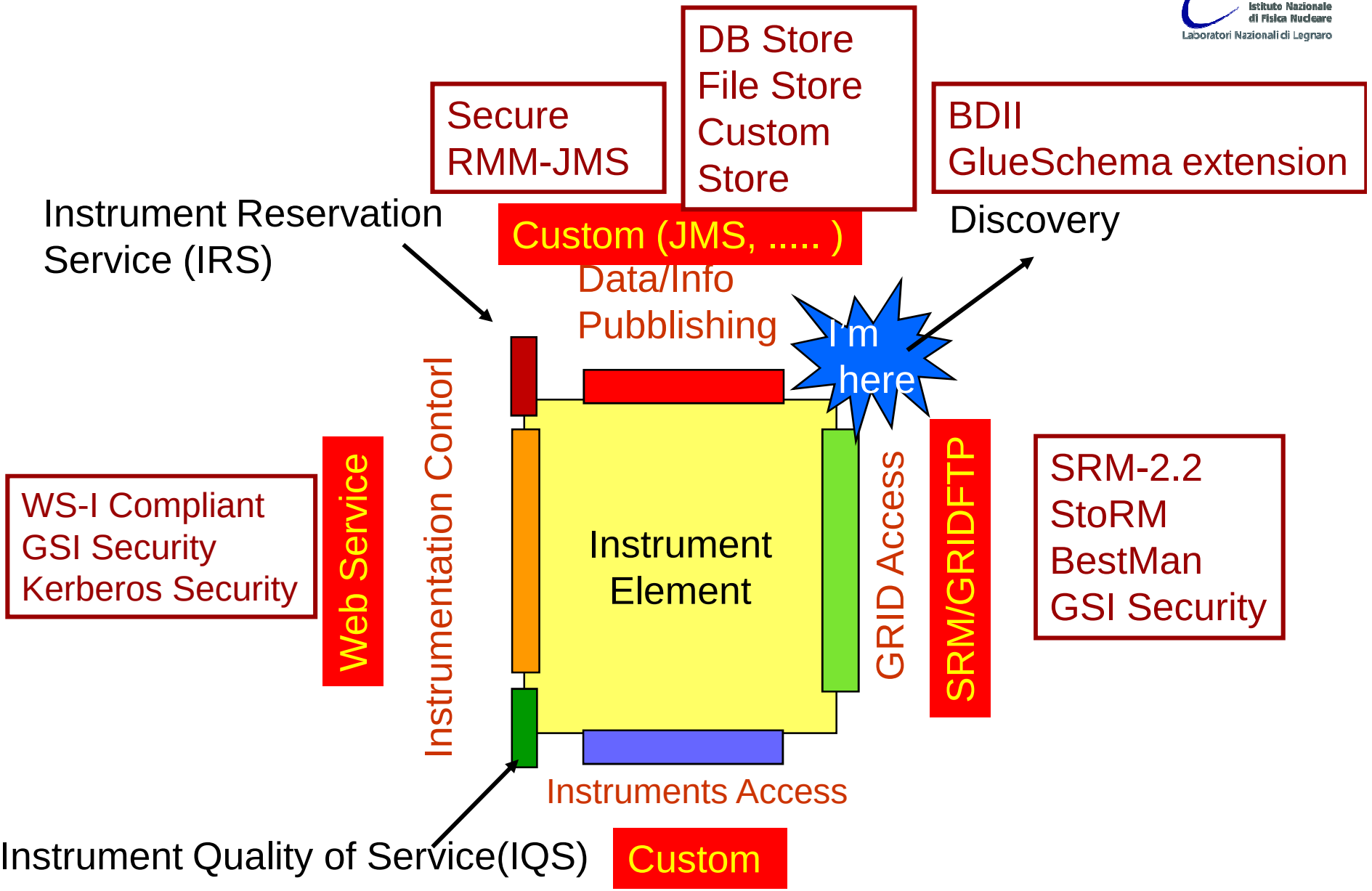
Instrument Element (IE): the basic idea



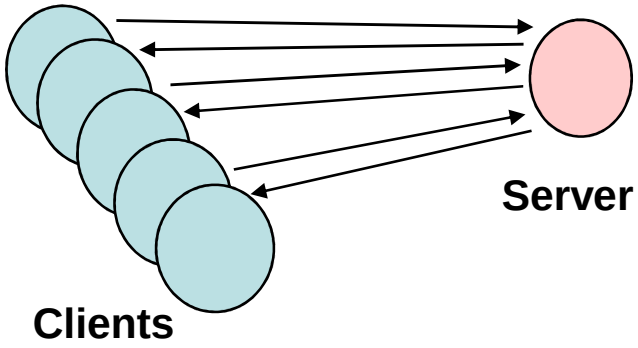
(Some) GRIDCC Developments

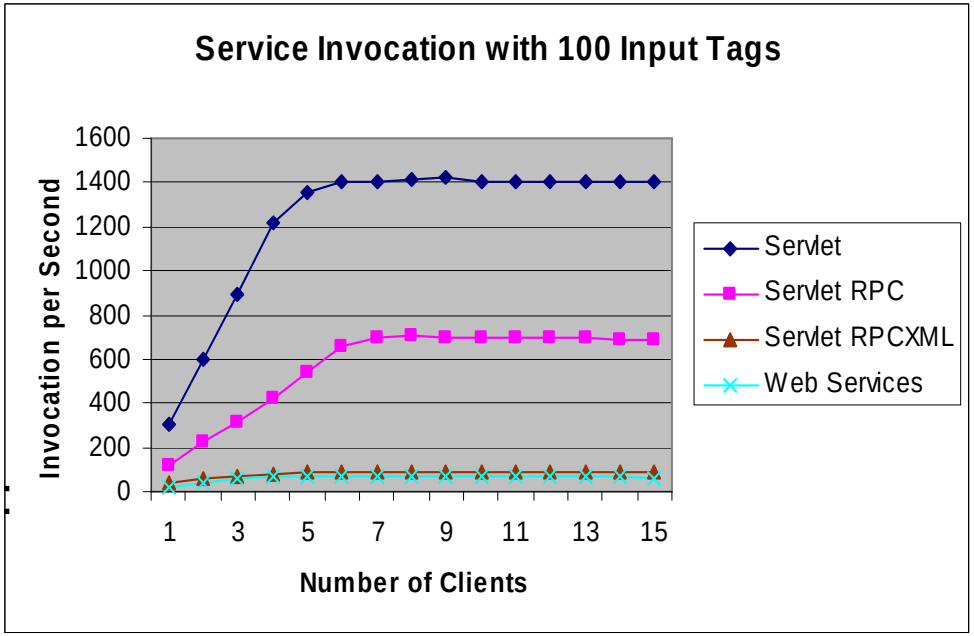
- Virtual Control Room (VCR)
- Instrument Element web service façade
  - WS-I compliance
- Instrument Reservation (Web) Service
- Problem Solver (Web) Service
- Integration with GRID (EGEE) components
- IE Security
- Quality of Service
- Automatic discovery of GRIDCC components (P2P approach)
- Grid on a CHIP
  - light version of IE to facilitate its adoption in the embedded systems, including the FPGA based – Axis standalone
- Tiny IE : light version of IE
  - Easier software deployment and installation
- JMS (Java Messaging System) experience
  - When the web services performance is not enough…
  - RMM – JMS (GRIDCC IBM)

Final Release: Instrument Element



Web Services Performance





Client/Server communications type:

- Simple HTTP
- SOAP + XML over HTTP
- Web Service

The hardware and Software:

•Application Server: Dual Xeon1.8 GHz, with 1.5 GB RAM, Ethernet 1 Gbps

Red Hat Linux Advanced Server 2.1. Tomcat 5.0.28 Axis

•Clients: 15 Pentium III 600 MHz, with 256 MB RAM, Ethernet 1 Gbps

Linux Red Hat 7.0. java 1.4 application

When the WS performance is not enough…

- **CMS DAQ Monitoring has been implemented in XDAQ (C++)**
  - O(10^6) monitorables per second
  - XML communication protocol (WS-Eventing standard) at the beginning
  - Binary protocol now

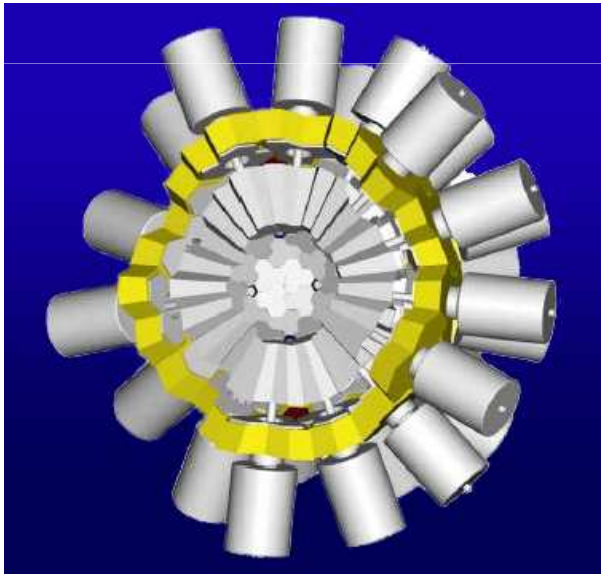
- **GRIDCC exploits JMS (Java Message System)**
  - RMM – JMS (IBM GRIDCC)

The Agata Experiment

http://www-win.gsi.de/agata/

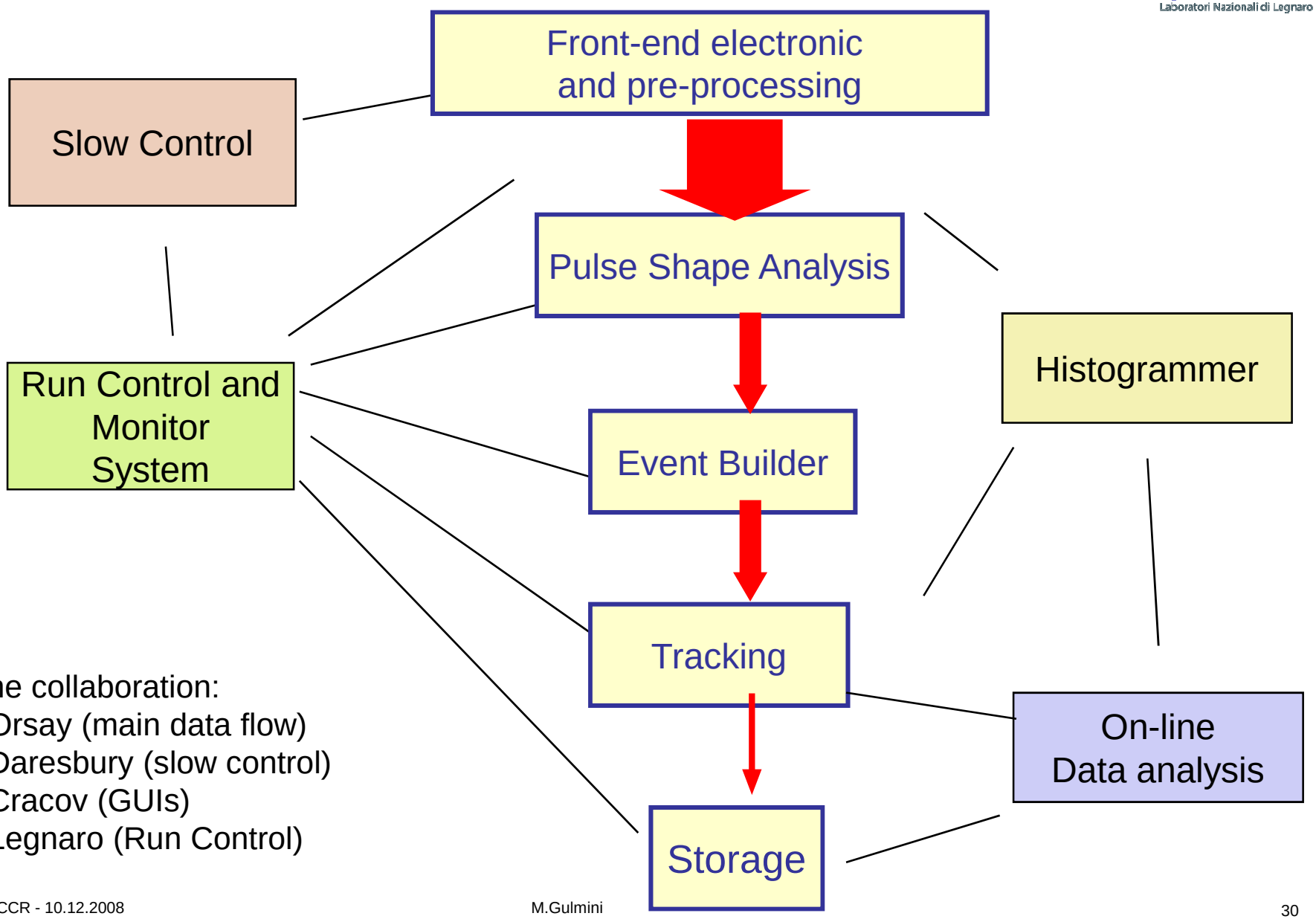
Agata Project

- The aim of Agata project is to develop, build and employ an Advanced GAmma Tracking Array, AGATA, for nuclear spectroscopy.
- **AGATA** is being realized within a European collaboration and is intended to be employed in experimental campaigns at radioactive and stable beam facilities in Europe.



- First campaign: Legnaro
- Installation at LNL in progress
- First data taking test next week

Agata DAQ

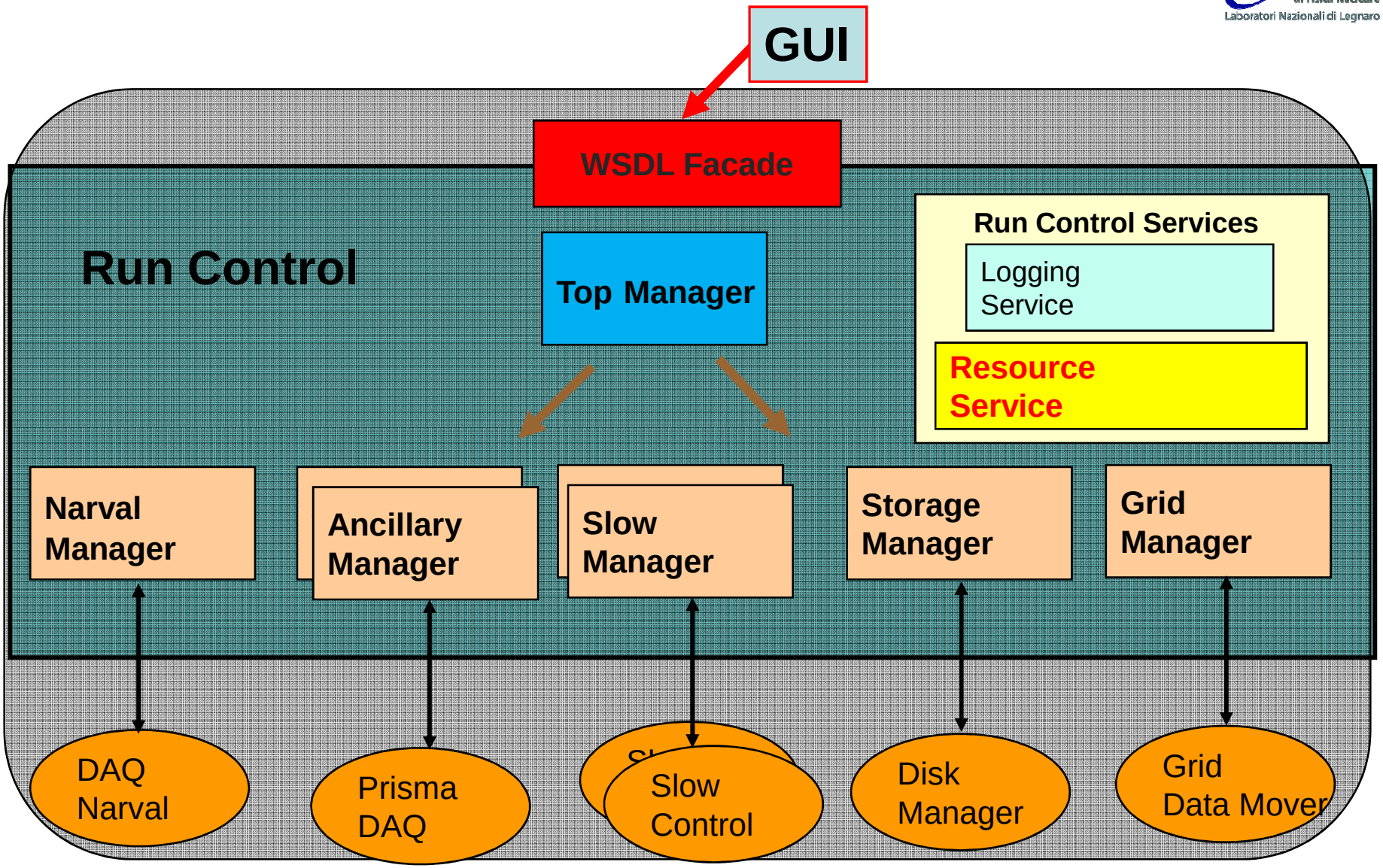


The collaboration:
- Orsay (main data flow)
- Daresbury (slow control)
- Cracov (GUIs)
- Legnaro (Run Control)

Agata DAQ & Run Control

- The DAQ is developed in Ada programming language
  - Narval (http://narval.in2p3.fr/)
    - Orsay team (IN2P3)
- LNL is developing the “Run Control”
- GRIDCC Instrument Element software is used
  - No major server side developments so far
    - Porting to Java 6 and Tomcat 6
  - Concentrate on “custom” control and monitor
- Wish (To Do) list
  - Use Modern Web Services standards and tools
    - JAX-WS APIs
    - JAXB binding
    - Axis 2

Agata Run Control Structure



Web Services

Client side Development

WSDL: Axis Java client (I)

- RCMS, GRIDCC and Agata web applications use Axis stubs
  - They communicate with other Java Axis Web Services
  - For instance CMS Function Managers (FMs) retrieve the configuration from the Resource Service WS, send commands to other FMs, receive state change notifications from other FMs
- Agata DAQ is developed in Narval (Ada programming language)
  - Narval distributed system provides a WSDL for control and monitoring purposes (Generated by *Ada2WSDL)*
  - Axis Java stubs were generated
    - no particular problems

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="narval_aws"
…
  <!-- Generated by AWS/Ada2WSDL v1.1
       on Wednesday 06 February 2008 at 10:05:01 -->
  <types>…
  </types>…
  <message name="Send_Command_With_Arguments_Request">
    <part name="Command" type="xsd:string"/>
    <part name="Arguments" type="n1:String_Array"/>
  </message>

  <message name="Send_Command_With_Arguments_Response">
    <part name="Result" type="xsd:string"/>
  </message>
…
```

WSDL: Axis Java client (II)

- Agata: Slow Control WSDL (Daresbury)
  - Defined in C language
    - **int Do-Reset (int *rc);**
      int Do-SetUp (int *rc);
      int Do-Stop (int *rc);
      int Do-Go (int *rc);
      ...
  - WSDL generated using gSOAP
  - WSDL implemented in Tcl for “digitizer electronics” (Daresbury)
    - Tcl Web Server : http://www.tcl.tk/software/tclhttpd/
  - WSDL implemented in Ada (ENX) for “segment and core mezzanines” (http://enx.in2p3.fr/)
- Run Control Java/Axis stubs generated (Legnaro)
  - First tests: no problems encountered

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="DataAcquisitionControlServer“ targetNamespace="http://npg.dl.ac.uk:8015/DataAcquisitionControl.wsdl"

<message name="Get-StateRequest">
</message>

<message name="Get-StateResponse">
 <part name="ResponseCode" type="xsd:int"/>
 <part name="Code" type="xsd:int"/>
 <part name="State" type="xsd:string"/>
 <part name="Reason" type="xsd:string"/>
</message>
```

WSDL: XDAQ Client (CMS)

- CMS XDAQ C++ framework does not provide WSDL support
- But it fully supports SOAP (Xoap library)

- Communication with RCMS web services is achieved by libraries that construct the proper SOAP messages
  - Used Mainly for asynchronous notifications (state changes, errors)
- When stub generation from the WSDL is not available
  - SOAP messages can/must be constructed by hand
  - It works fine
  - Complexity depends on WSDL complexity

In the class constructor:

```
rcmsStateNotifier_( logger_,getApplicationDescriptor(),getApplicationContext() )
```

Then:

```
try
  {
                rcmsStateNotifier_.stateChanged( "Error", xcept::stdformat_exception_history( exception ) );
  }
  catch( xcept::Exception &e )
  {
    LOG4CPLUS_ERROR( logger_, "Failed to notify state change: "
      << xcept::stdformat_exception_history(e) );
  }
```

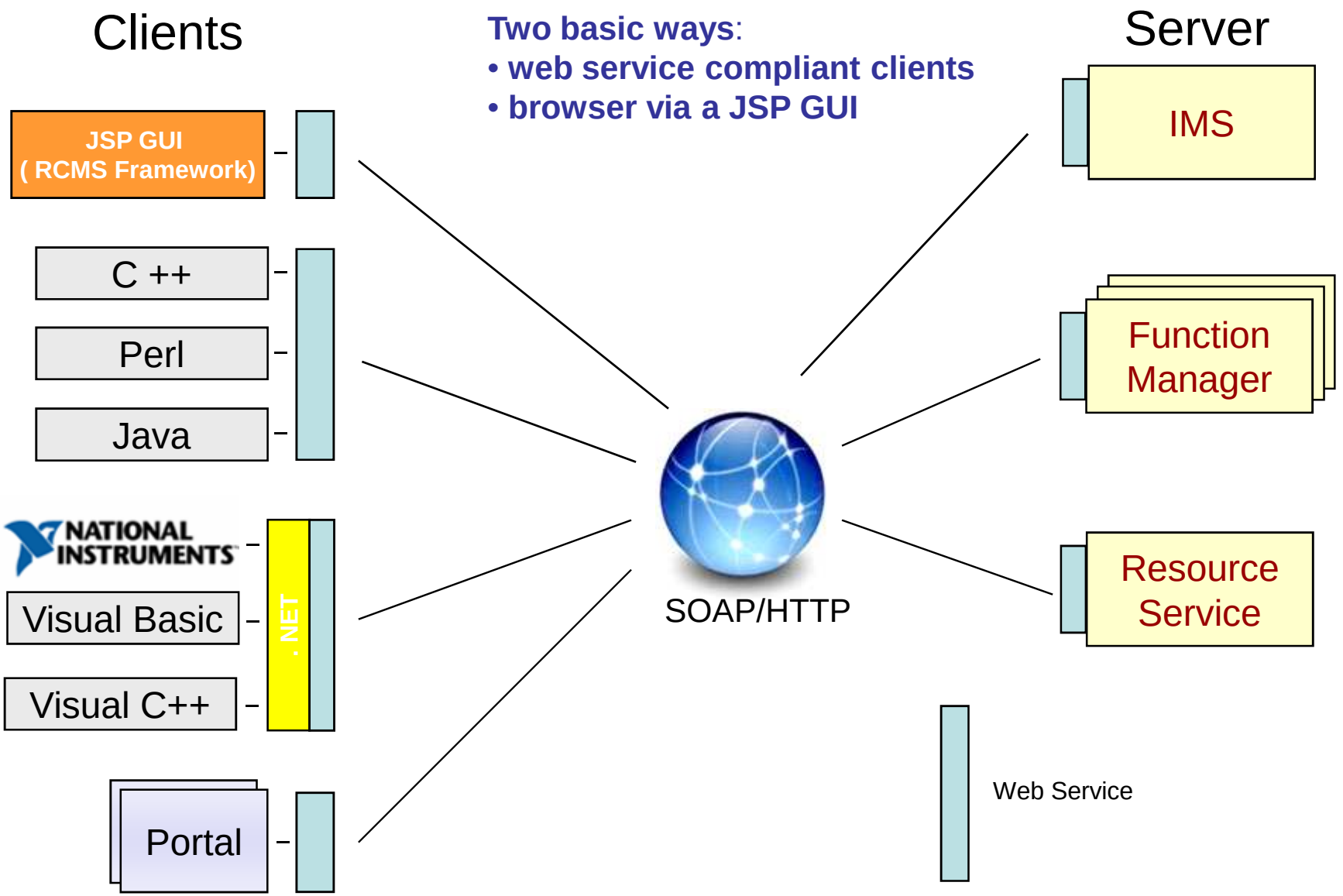
WSDL: Perl Client

- WSDL and SOAP support in Perl is provided by *SOAP::Lite* package
  - Collection of Perl modules which provides a simple and lightweight interface to the Simple Object Access Protocol
  - CERN Linux SLC 3.0.4 packages:
    - perl-MIME-Lite (3.01-1), perl-MailTools (1.62-1), perl-SOAP-Lite (0.55-3), perl-TimeDate (2.22-1)
  - DII (Dynamic Invocation Interface) approach
- Widely Used in CMS
  - Scripts to stress the control system
  - Reproduce bugs
  - Performance measurements
- Also used inside Labview for Mac (see later)

Perl Script : example

```
#!/usr/bin/perl
# getStateFM.pl
# Author: Andrea Petrucci
# date: 23/10/2005

use strict;
use SOAP::Lite;

my $WSDLCommandService="http://pccms143.cern.ch:8080/rcms/services/CommandService?wsdl";
my $NSCommandService = "urn:FMCommand";
my $HOSTCommandService = "http://pccms143.cern.ch:8080/rcms/services/CommandService";

my $CommandService = SOAP::Lite
->readable(1)
->xmlschema('http://www.w3.org/2001/XMLSchema')
->on_action( sub { return '""';} )
->proxy($HOSTCommandService)
    ->uri($NSCommandService);

my $method = SOAP::Data->name('getState')
    ->prefix('ns1')
    ->uri($NSCommandService);

my $result = $CommandService->call($method => ($uri));
if ($result->fault){
      print join ';',
          $result->faultcode,
          $result->faultstring;
}else {
   my $stateBean = $result->valueof('//getStateResponse/getStateReturn');
   print "$stateBean->{'stateString'}\n";
}
```

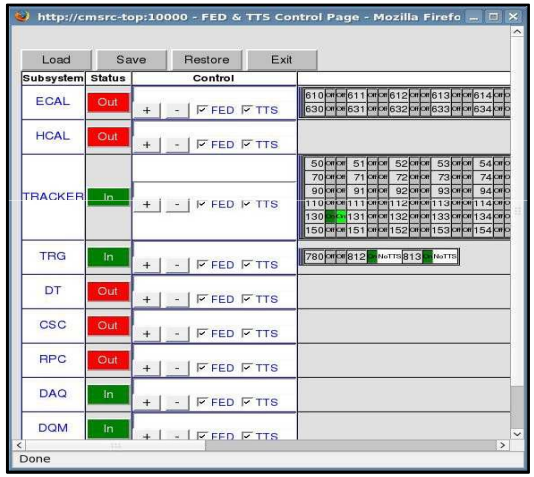
WSDL Clients: GUIs for Run Control



JSP CMS Run Control GUIs

1) RCMS GUI

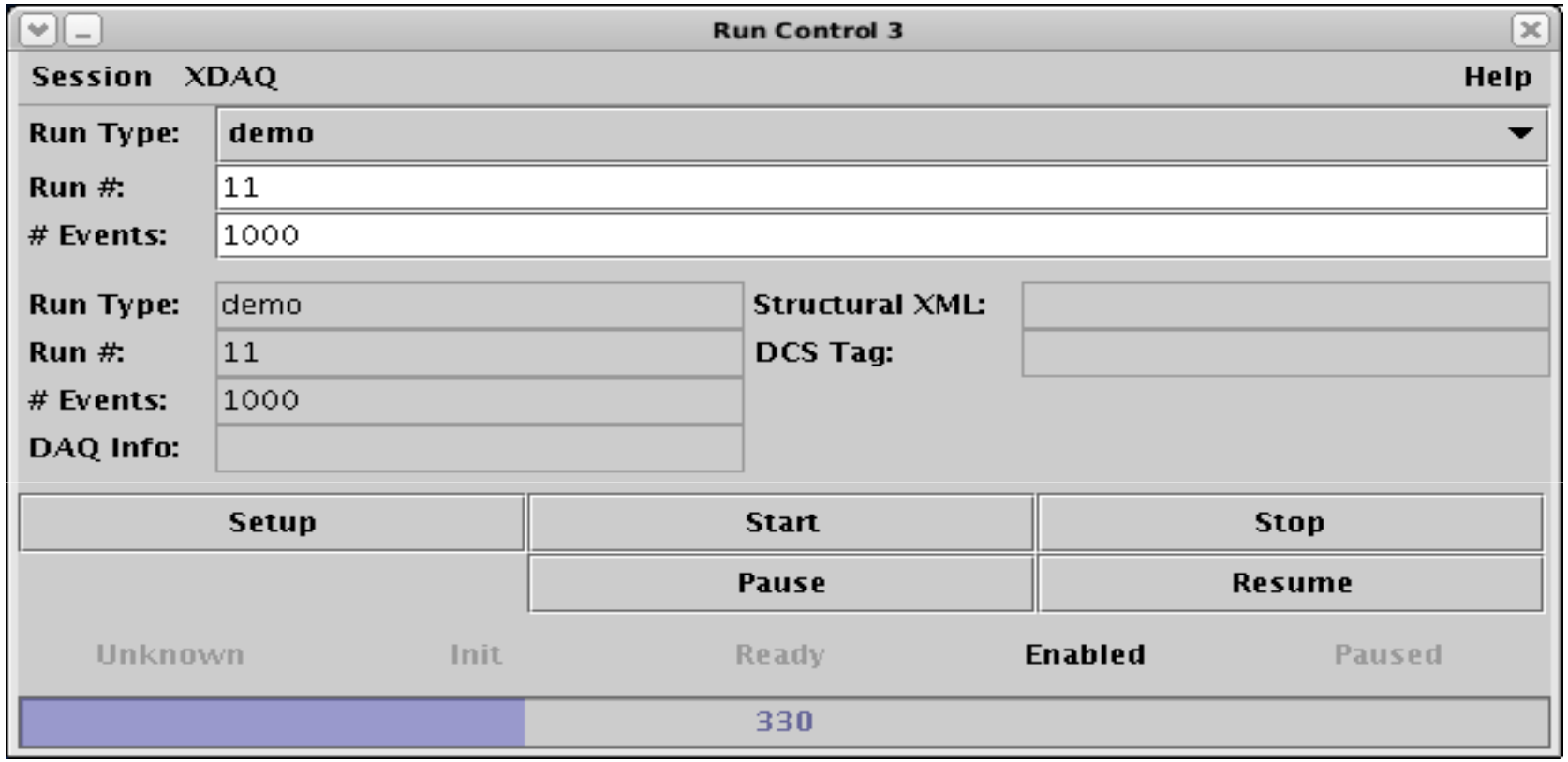
2) Function Manager Level Zero GUI



← 3) FED and TTS GUI

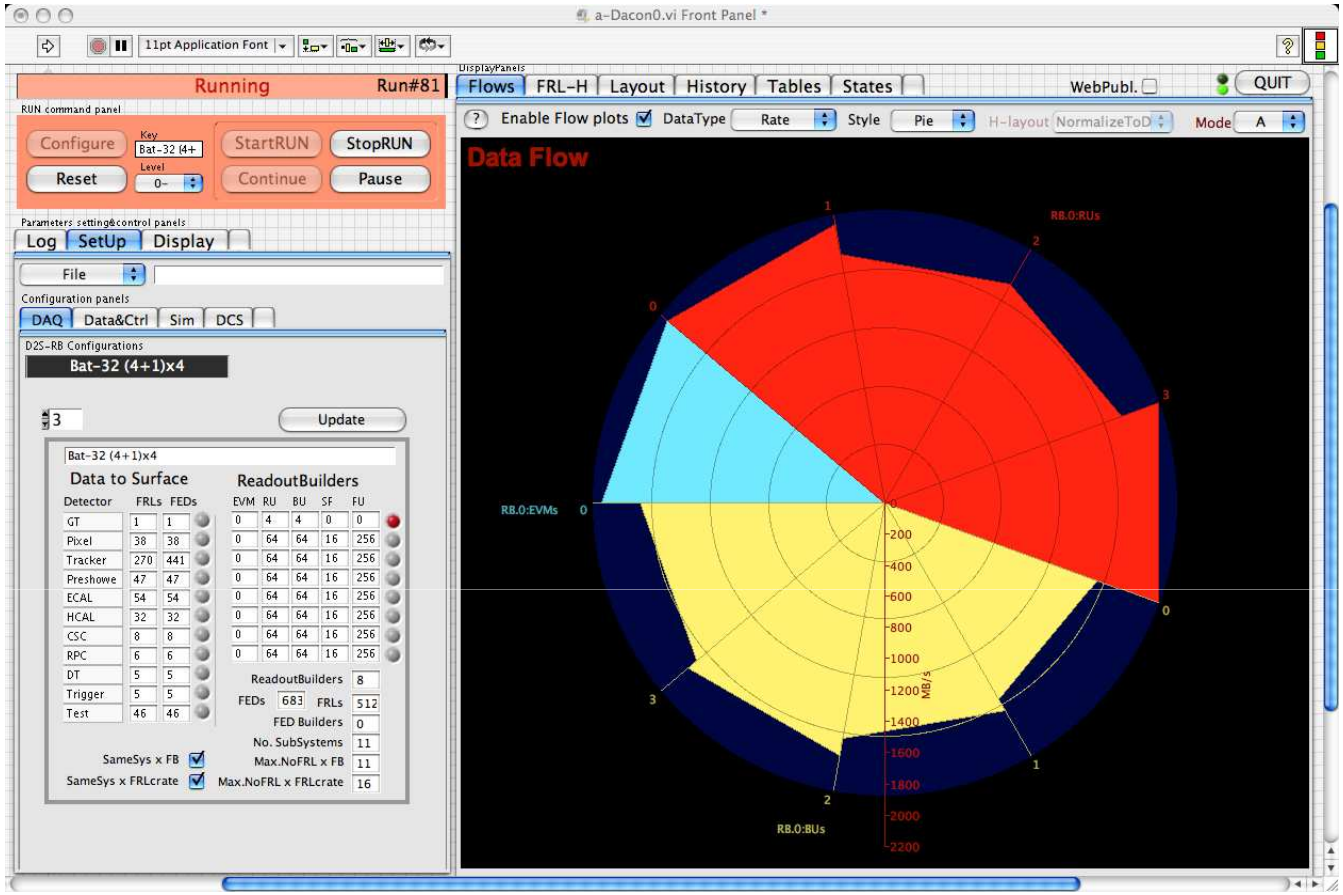
- JSP (Java Server Pages)
  - Server side code generating HTML pages
  - Use Java/Axis stubs
  - Javascript and AJAX technology

Java Applet GUI example



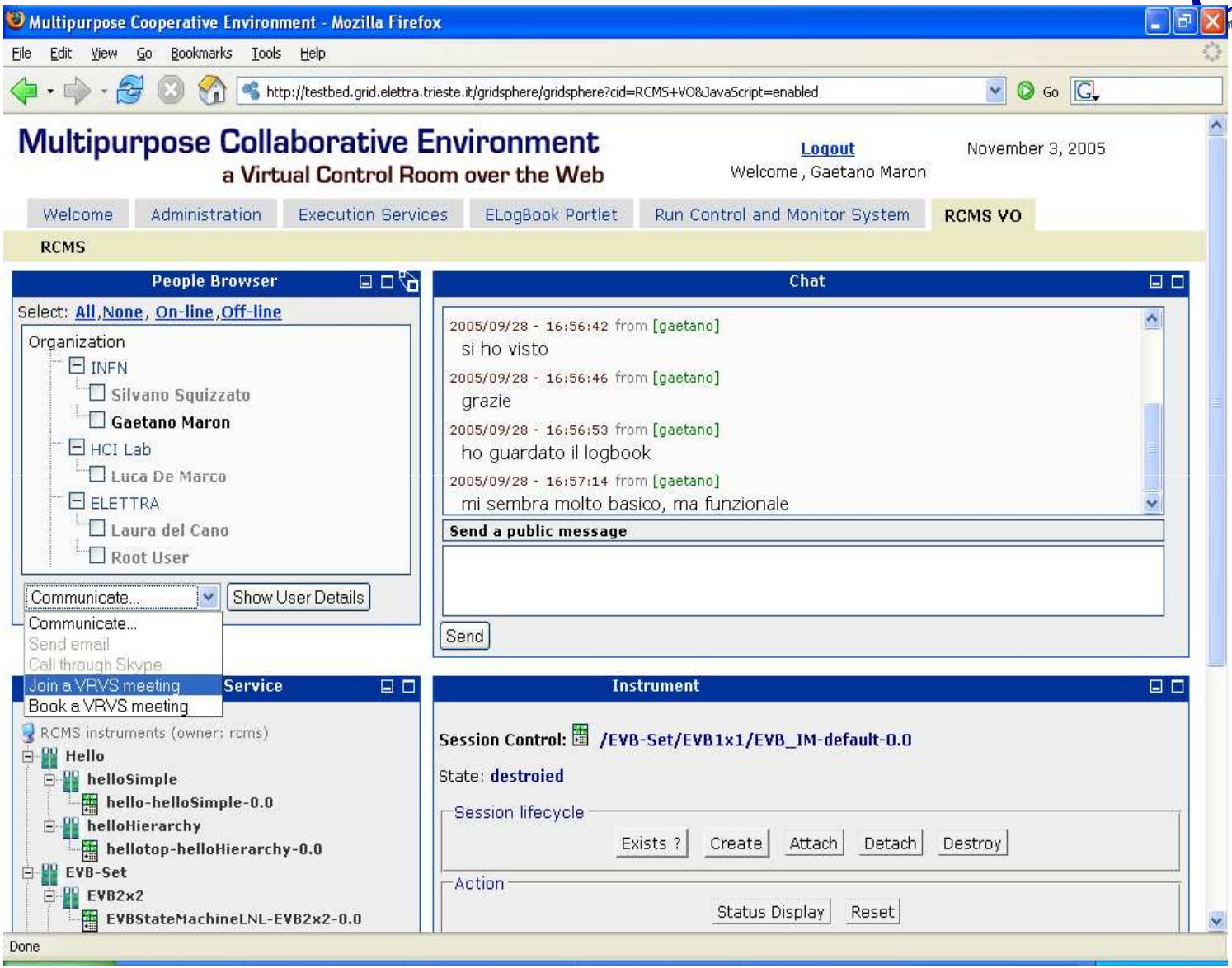
- CMS test beams and production centers

LabView GUI

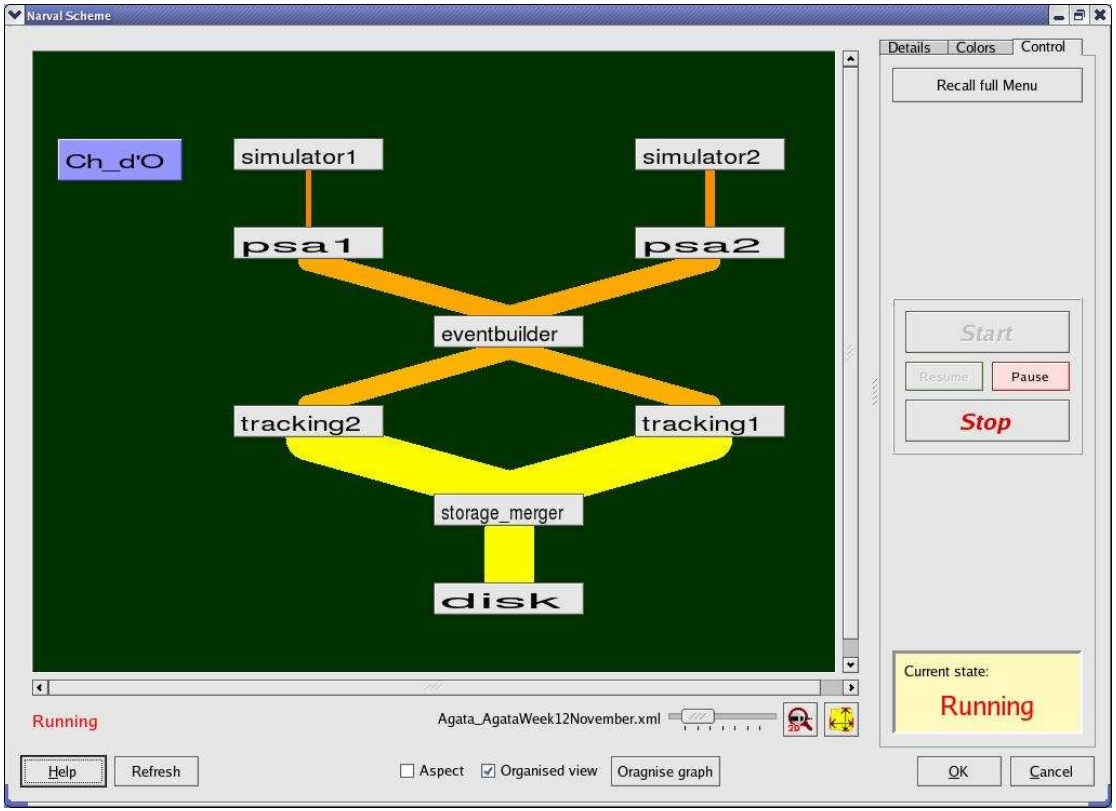


- WSDL client tests both with Mac and Windows
  - Windows: .NET support (OK)
    - Some troubles with complex data structures
  - Mac: (KO)
    - Perl scripting language as a workaround

GRIDCC VCR – Portal and Portlet



Agata GUI (Cracov)



- Prototype available, Developed in C++, Qt based
- Communicates to the Agata Run Control through the GRIDCC WSDL facade
- Agata Week November 2007:
  - *“Lucky the WSDL files given to me are fully compatibile with my gSOAP standard” (Jurek Grebosz)*

Considerations

- Web Services are a wonderful paradigm for…
  - **Interoperation** of software running on different platforms and developed with different programming languages
  - Quick development of clients
    - Allow to concentrate on the business part of the application
- Many many useful “Web Services tools” available
- Suitable for “online” systems when performance is not a big issue
  - Run Control systems

Questions

People

- List of people working on the contents of this presentation (not exhaustive, not ordered):
  - Michele Gulmini
  - Gaetano Maron
  - Pietro Molini
  - Andrea Petrucci
  - Alexander Oh
  - Francesco Lelli
  - Eric Frizziero
  - Sergio Traldi
  - Luigi Zangrando
  - Silvano Squizzato

Istituto Nazionale di Fisica Nucleare
Laboratori Nazionali di Legnaro

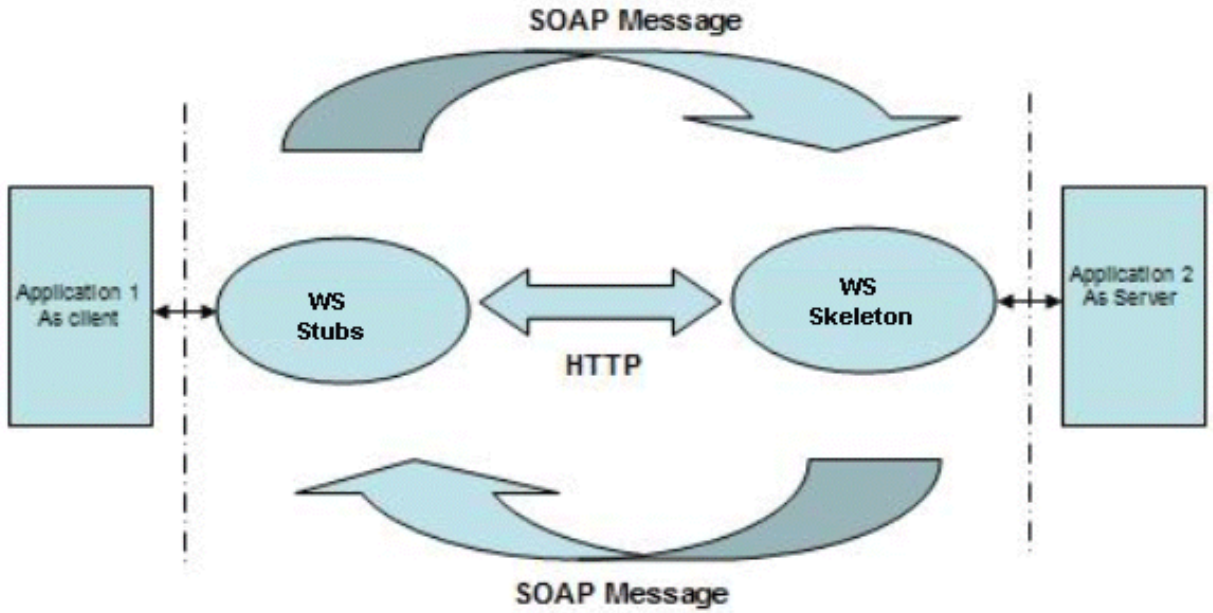
SPARE SLIDES

JAX-RPC

JAX-RPC provides an easy to develop programming model for development of SOAP based Web services.

The figure elaborates the normal Web service invocation architecture for the synchronous request-response mode :

- The client uses runtime libraries to serialize Java objects to a SOAP message and sends it to the Web service end point, using HTTP transport.
- As the Web service side that is deployed on Apache Tomcat receives this request, the service-side JAX-RPC runtime deserializes the SOAP message in to Java types and invokes the method on the Web service and in turn makes a call to Application 2.
- The Web service, after processing the request, sends response back to the client in a similar.

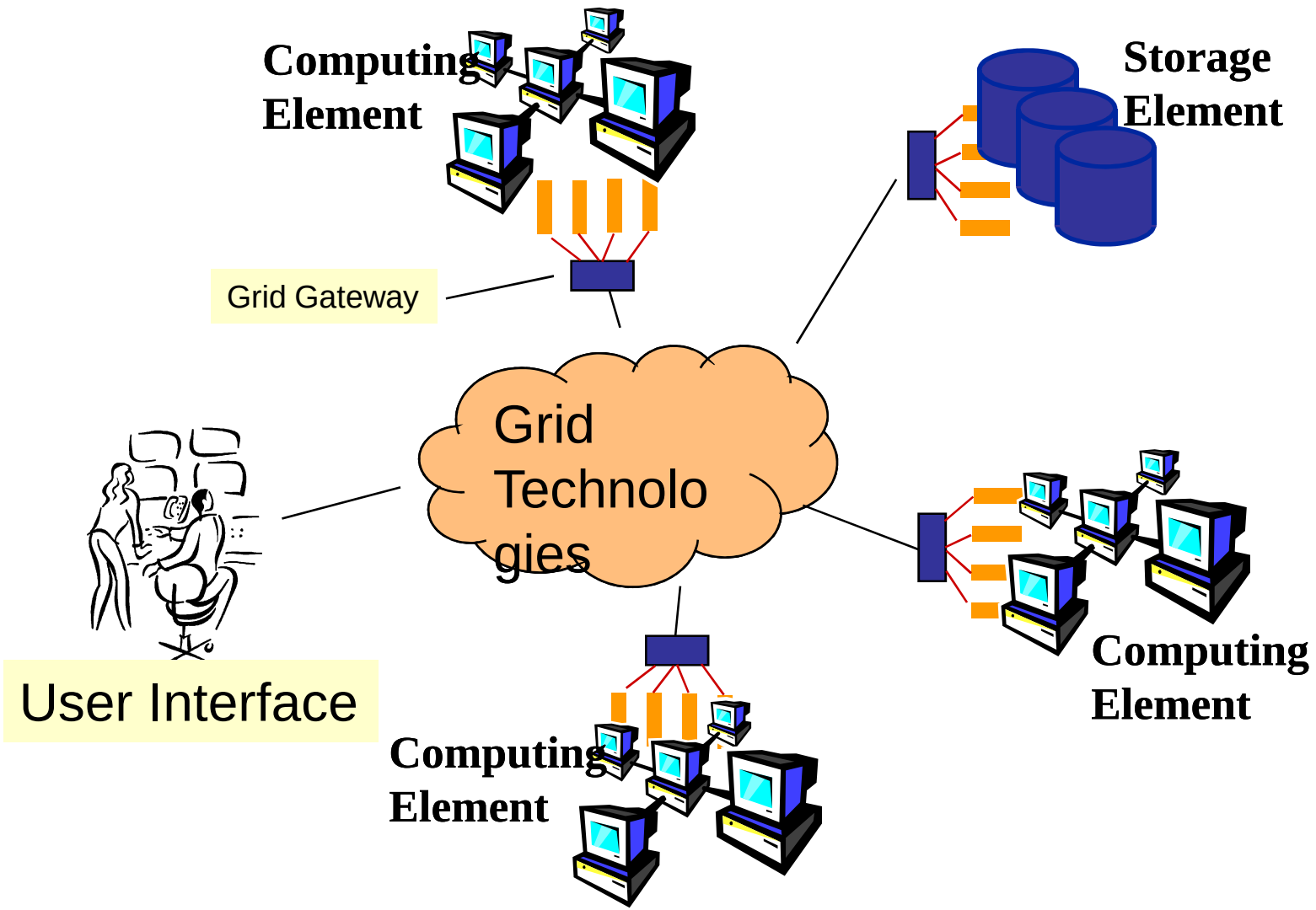


References

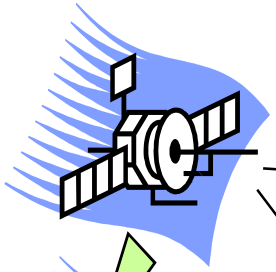
- Web service : Web Services Architecture (http://www.w3.org/TR/ws-arch/)
- WSDL : Web Service Description Language (http://www.w3.org/TR/wsdl)

  Basic Profile Version 1.0 (http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html)
- Apache Tomcat : The Apache Jakarta Tomcat 5 Servlet/JSP Container (http://tomcat.apache.org/tomcat-5.0-doc/index.html)
- Apache Axis : Axis User's Guide (http://ws.apache.org/axis/java/user-guide.html)
- J2ee : Java 2 Platform, Enterprise Edition (http://java.sun.com/j2ee/1.4/)
- JAX-RPC : Java API for XML-Based RPC (http://java.sun.com/webservices/jaxrpc/index.jsp)

The Grid Technologies to extend the limit of a single computer (center)



Extending the Grid Concepts



Grid Gateway

Grid Technologies

Satellite views
to monitor the volcano

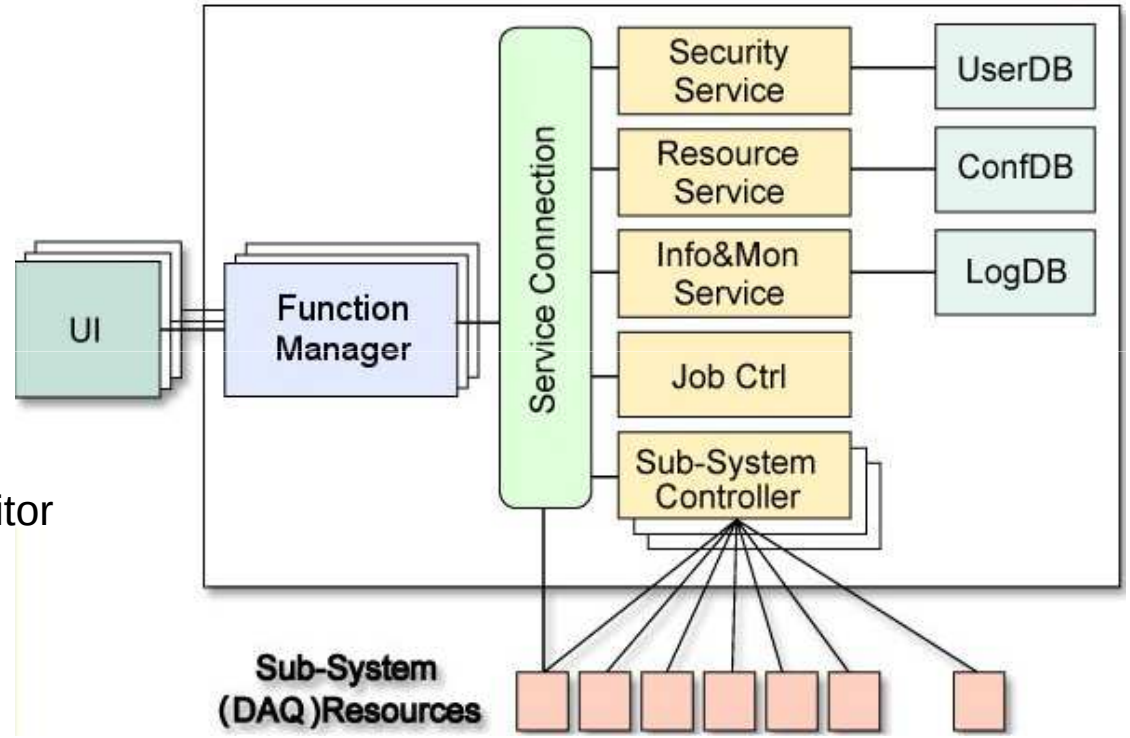
Terrestrial probes to monitor
The volcano activities

Control and Monitor Room

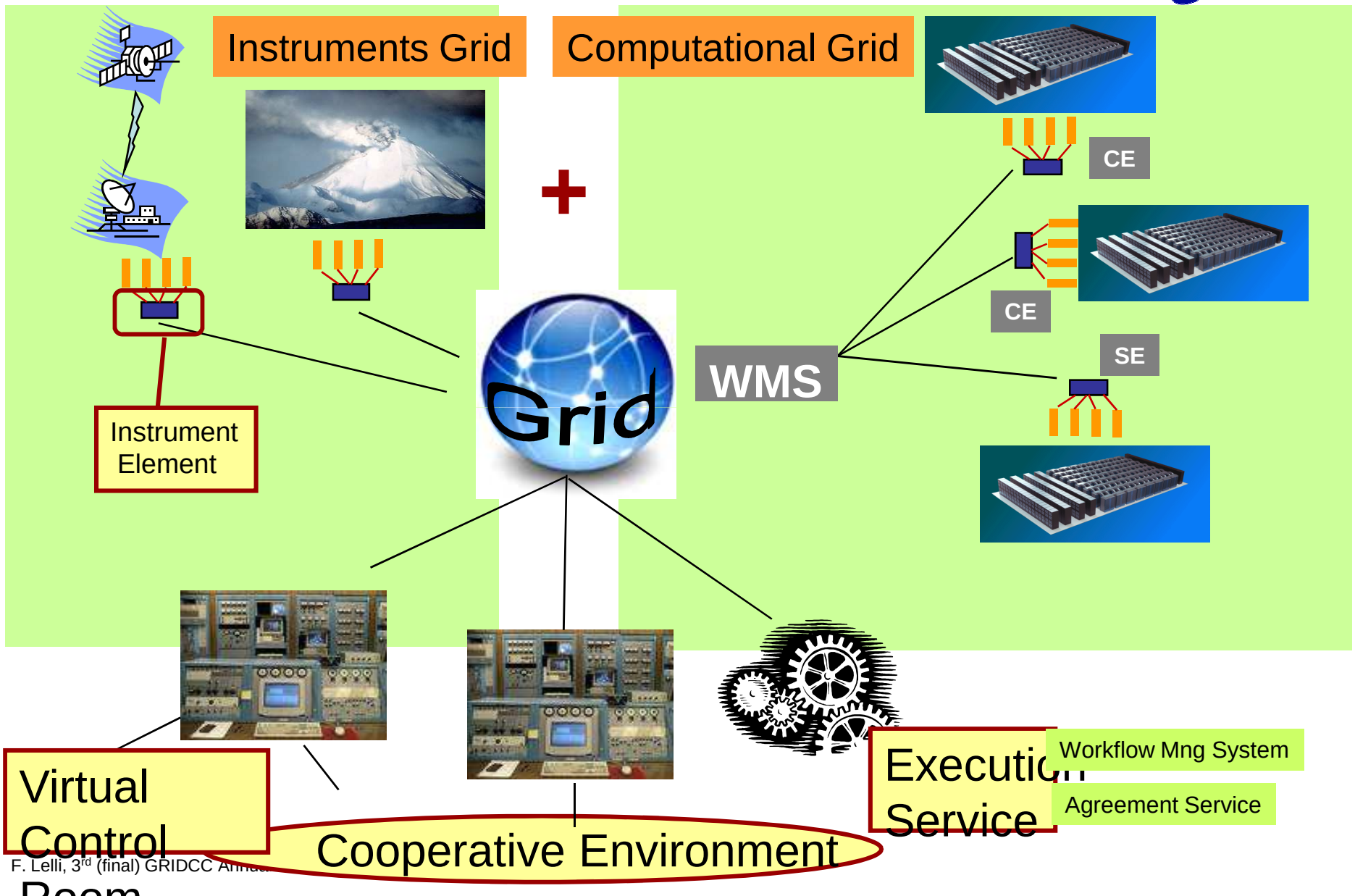
To model calculations and disaster predictio

RCMS Services

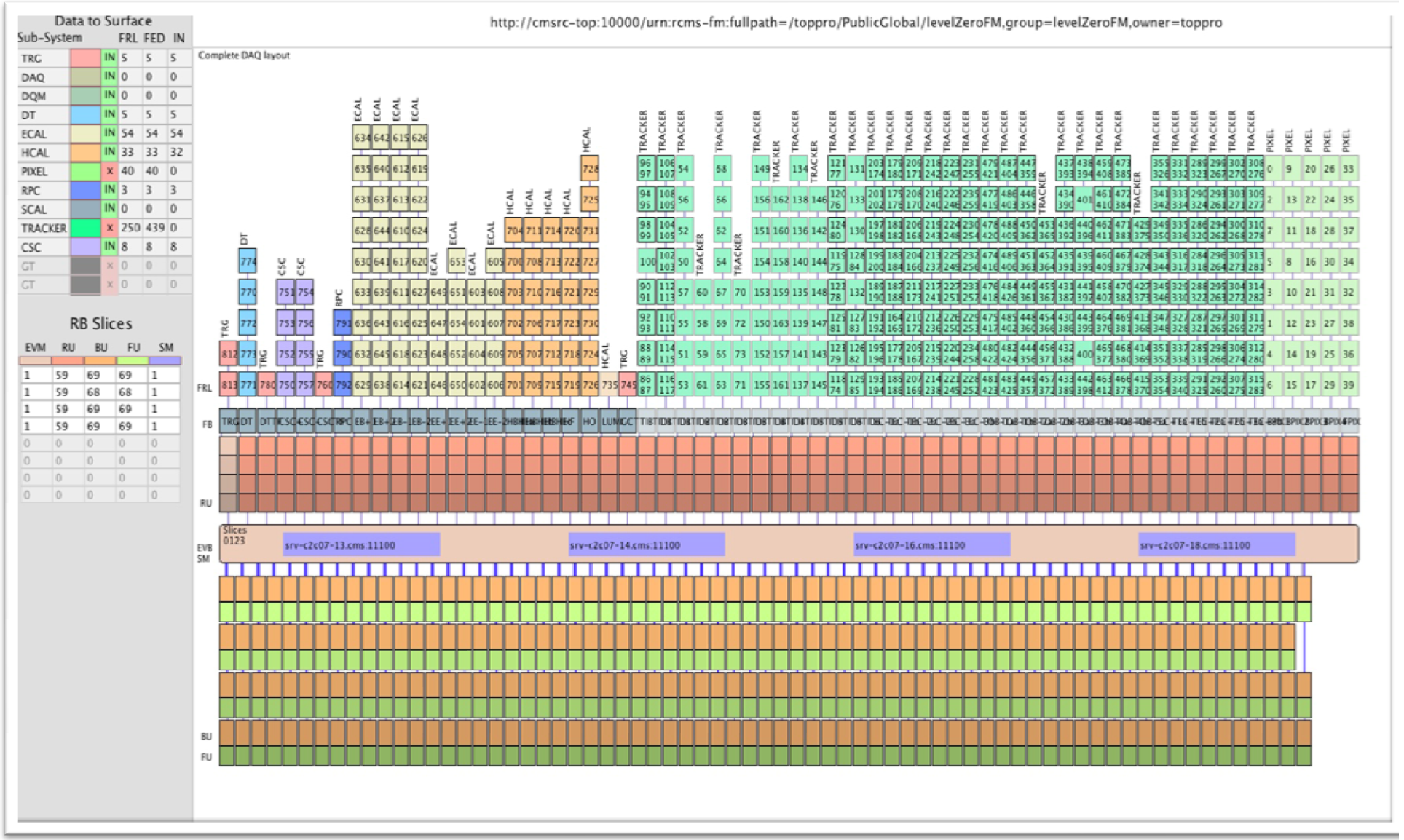
- SECURITY SERVICE
  - login and user account management;
- RESOURCE SERVICE (RS)
  - information about DAQ resources and partitions;
- INFORMATION AND MONITOR SERVICE (IMS)
  - Collects messages and monitor data; distributes them to the subscribers;
- JOB CONTROL
  - Starts, monitors and stops the software elements of RCMS, including the DAQ components;



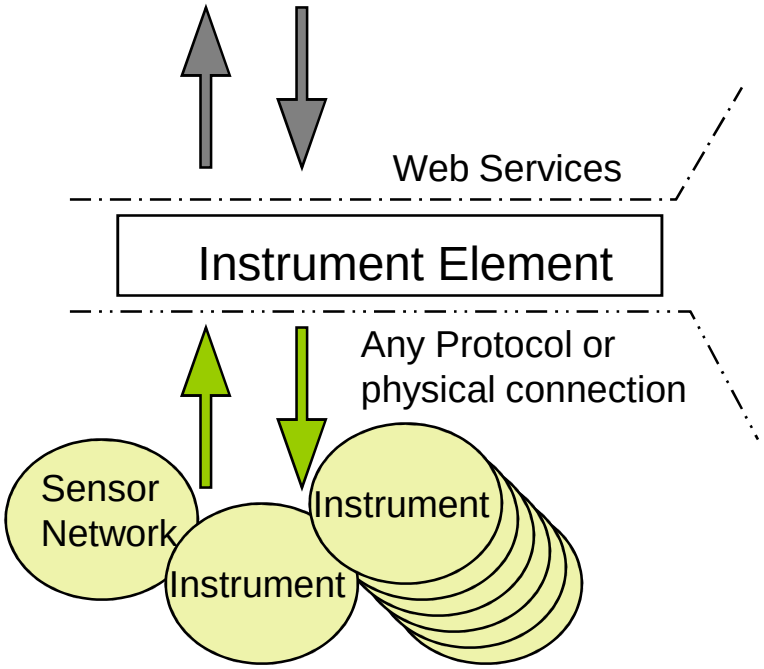
The GRIDCC Architecture

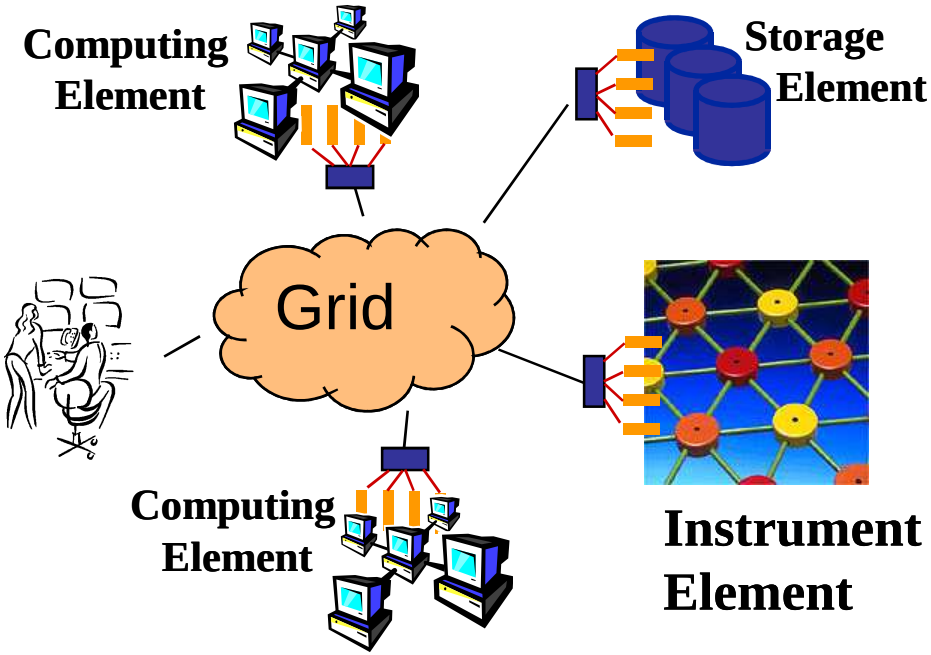


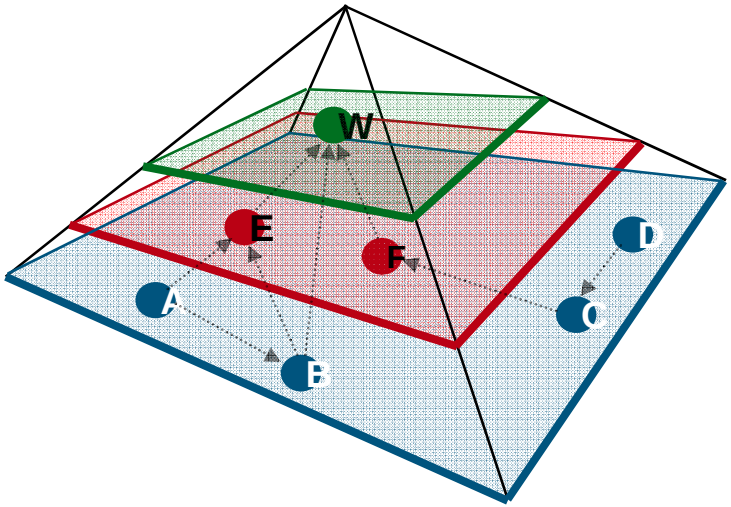
First ~~collision~~ beam configuration. Sept 08



Instrument Element Requirements







1: Provide a uniform access to the physical devices

2: Allow a standard grid access to the instruments

3: Allow the cooperation between different instruments that belong to different VOs

GRIDCC – IE technologies

- Web Service compliant (WS-I)
- Tomcat + Axis (and Java) and Axis standalone are the main technologies of the IE
- All the services are deployed on a single or multiple instances of Tomcat, according to the needs of the application
- Message oriented middleware (Pub/Sub) is based on the Java Messaging System (JMS). The following implementations are used in the project
  - RMM - JMS (GridCC IBM)
- MySQL and Oracle are used as Data Base for the RS
- STORM vers. 1.2 is the used SE that exposes a SRM interface version 2.2