Grid job management through Web services: the experience with CREAM and CEMon

Luigi Zangrando
INFN Padova
luigi.zangrando@pd.infn.it

The EGEE project

- **The Enabling Grids for E-sciencE (EGEE) project is funded by the European Commission (started from 2004)**
- **objective: to build and provide a production quality grid infrastructure spanning more than 30 countries with over 150 sites which is available to scientists 24 hours-a-day**
- **applications come from various scientific domains, including Earth Sciences, High Energy Physics, Bioinformatics and Astrophysics**
- **What we have today:**
  - > 80.000 cpu
  - > 20 PB storage
  - 250 sites
  - growing continuously
- **gLite: open source middleware developed by EGEE**
  - CREAM and CEMon: grid components developed for gLite by the grid middleware INFN Padova group
    - they provide specific functionalities
    - they share the same technology: Web Service

CREAM

- **CREAM (Computing Resource Execution And Management) service:**
  - general purpose framework for building grid services
  - functionalities/operations are pluggable
    - for example, the functionality for accessing a database can be easily added and plugged to CREAM
  - main functionalities provided: job management operations at the Computing Element (CE) level
    - allow the grid user to submit, cancel, monitor, … computational jobs
    - Computing Element: grid component acts as interface to computational resources
      - *single pc*
      - *cluster of pc handled by a LRMS (e.g. LSF, PBS/Torque, Condor)*
      - *supercomputer for High Performance Computing (HPC)*

CREAM: job manag. functionality

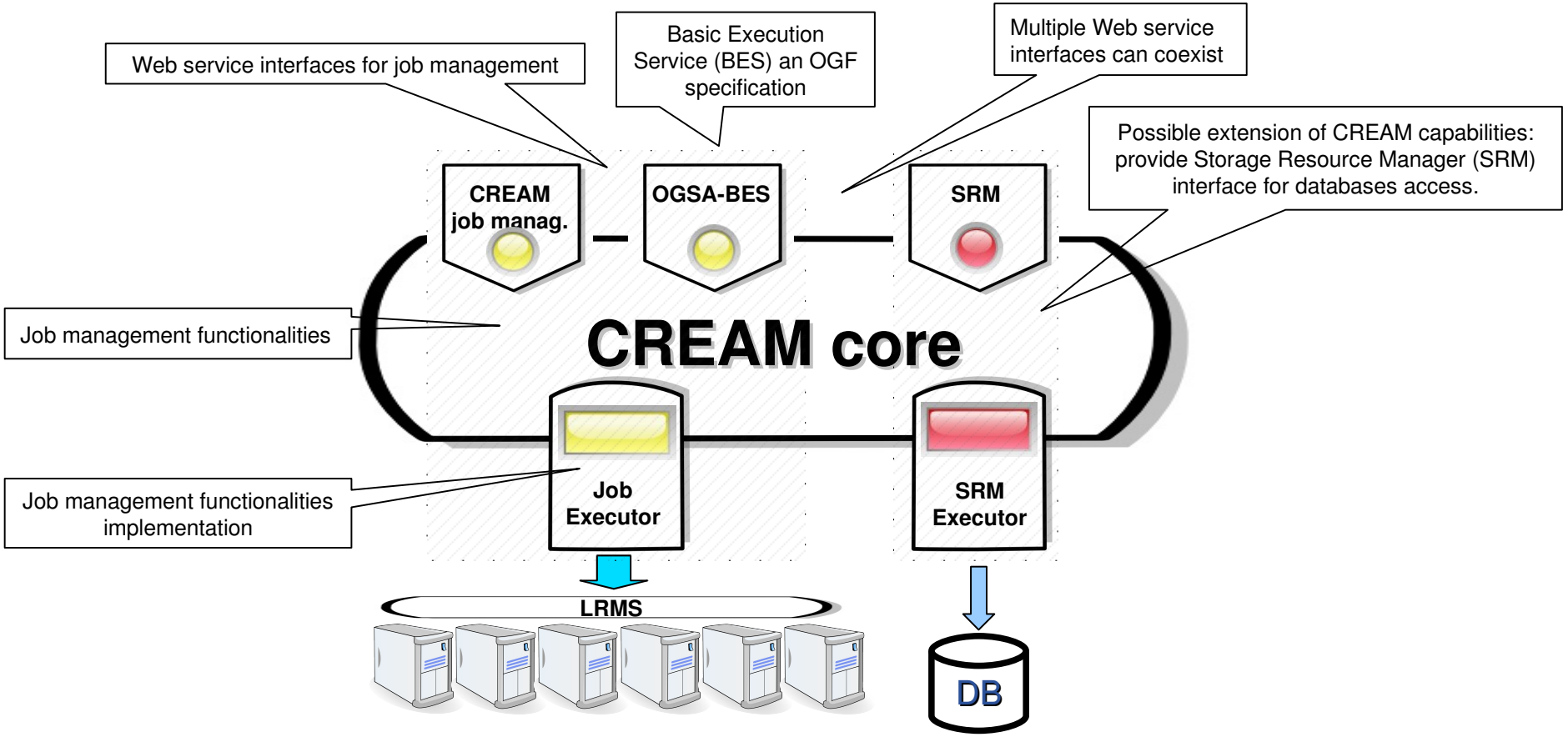
- **CREAM describes and exposes its functionality through a Web Service interface**
- **Functionality exposed:**
  - Job submission
    - Submission of computational jobs to a CREAM based CE
    - Supported job types: normal, MPI, collection/parametric
    - Job is described via a JDL (Job Description Language) expression
  - Job status
    - to get status and other info (e.g. significant timestamps, worker node, failure reason, issued commands on the job, etc.) of submitted jobs
  - Job cancellation
    - To cancel previously submitted jobs
  - Job suspension and job resume
    - To hold and then restart jobs
  - Job list
    - To get the identifiers of all your jobs submitted on a specific CREAM CE

CREAM: job manag. functionality

- Job purge
  - To clear jobs from CREAM based CE
- Proxy delegation
  - To delegate a proxy, which can be used by the job to do operations requiring security support (e.g. GridFTP file transfers)
- Proxy renewal
  - To renew proxies for previously submitted jobs
- Disable/enable new job submissions
  - Useful for example for a scheduled shutdown of the CREAM CE
  - Can be used only by CREAM CE administrators
- Check if submissions are enabled
- getServiceInfo
  - To provide information about CREAM (e.g. status, debugging messages, timestamps, properties, etc)

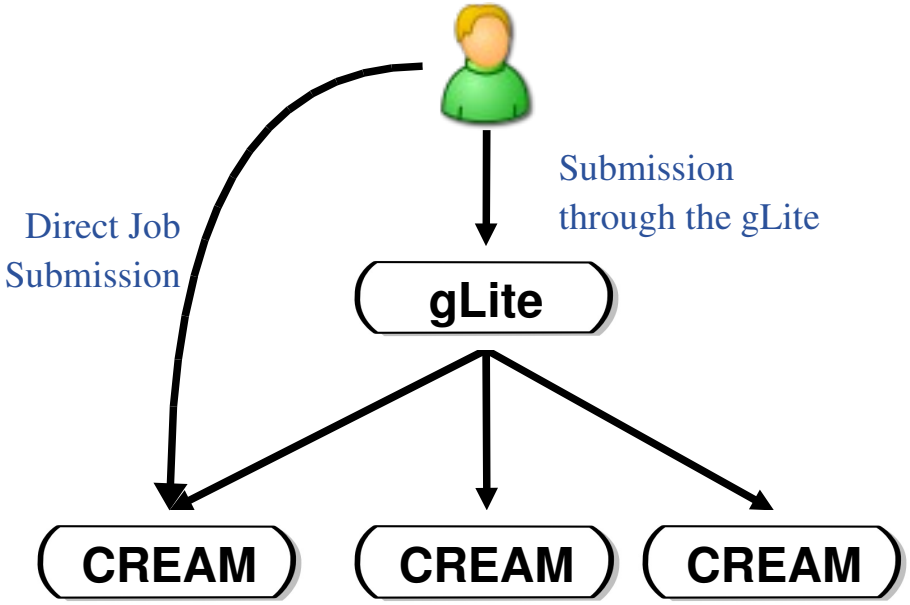
High level CREAM architecture

In the revised architecture, it is possible to plug different interfaces to CREAM (e.g. one for job management, one for database access, etc.).
The different commands (that is the operations defined on the WS interfaces) are then managed by different command executors (pluggable).
Support for both synchronous and asynchronous commands is provided.



Integration among web services

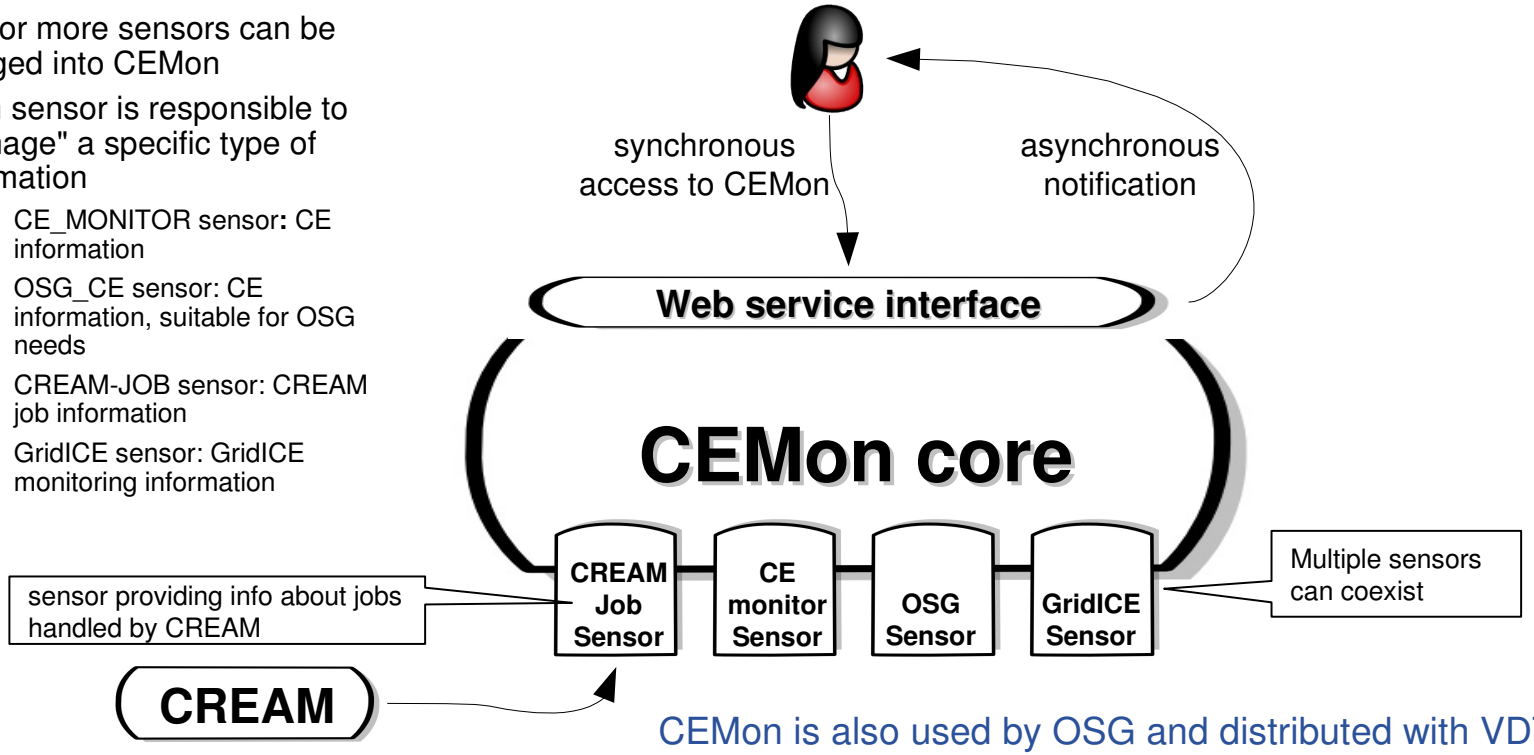
- **CREAM can be used:**
  - through the gLite components (Web Services)
  - directly by the users
    - they can build their own clients using a Web Service framework



CEMon

- **CEMon(itor) service:**
  - Web Service responsible to collect and provide information for monitoring purposes
  - general purpose notification framework working in synchronous and asynchronous mode
    - virtually it is able to support any kind of monitoring task thanks to its plug-in architecture based on sensor concept

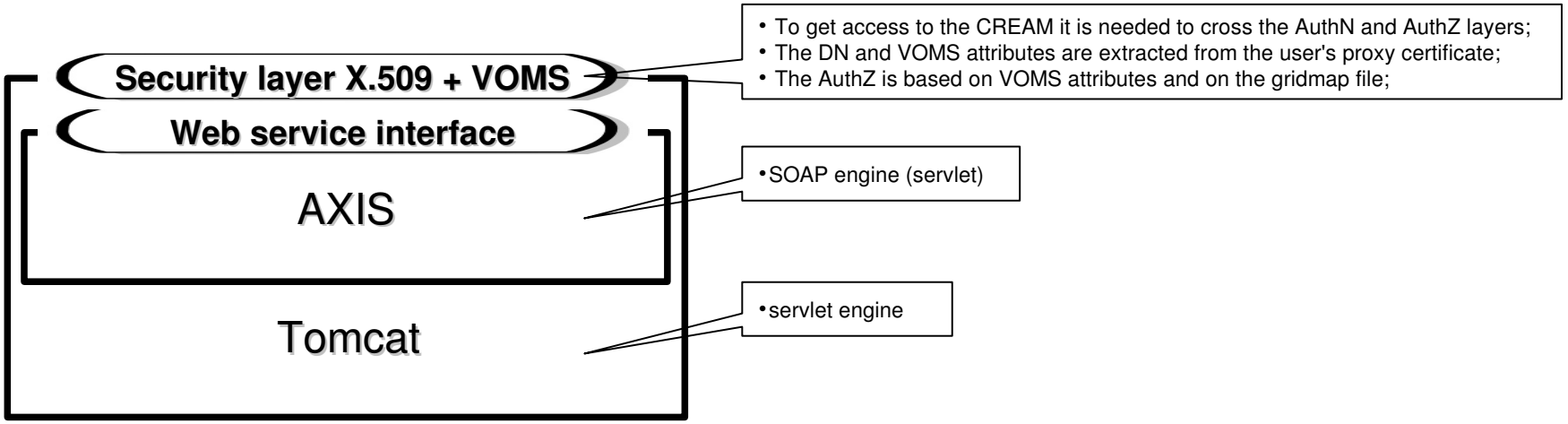
  - One or more sensors can be plugged into CEMon
  - Each sensor is responsible to "manage" a specific type of information
    - CE_MONITOR sensor**:** CE information
    - OSG_CE sensor: CE information, suitable for OSG needs
    - CREAM-JOB sensor: CREAM job information
    - GridICE sensor: GridICE monitoring information



CEMon is also used by OSG and distributed with VDT

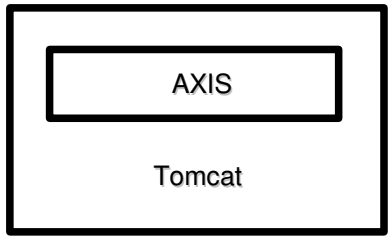
"Simplified" view of CREAM and CEMon architecture

- **Web service interface**
  - WS-I compliance
  - WSDL 1.1
  - document/literal
- **SOA (Service Oriented Architecture) paradigm adopted**
- **Fully implemented in Java**
  - developed with Apache Axis (version 1.4) framework for Java
    - http://ws.apache.org/axis/
- **CREAM and CEMon run as Java-Axis servlets on Tomcat 5.5 application server**

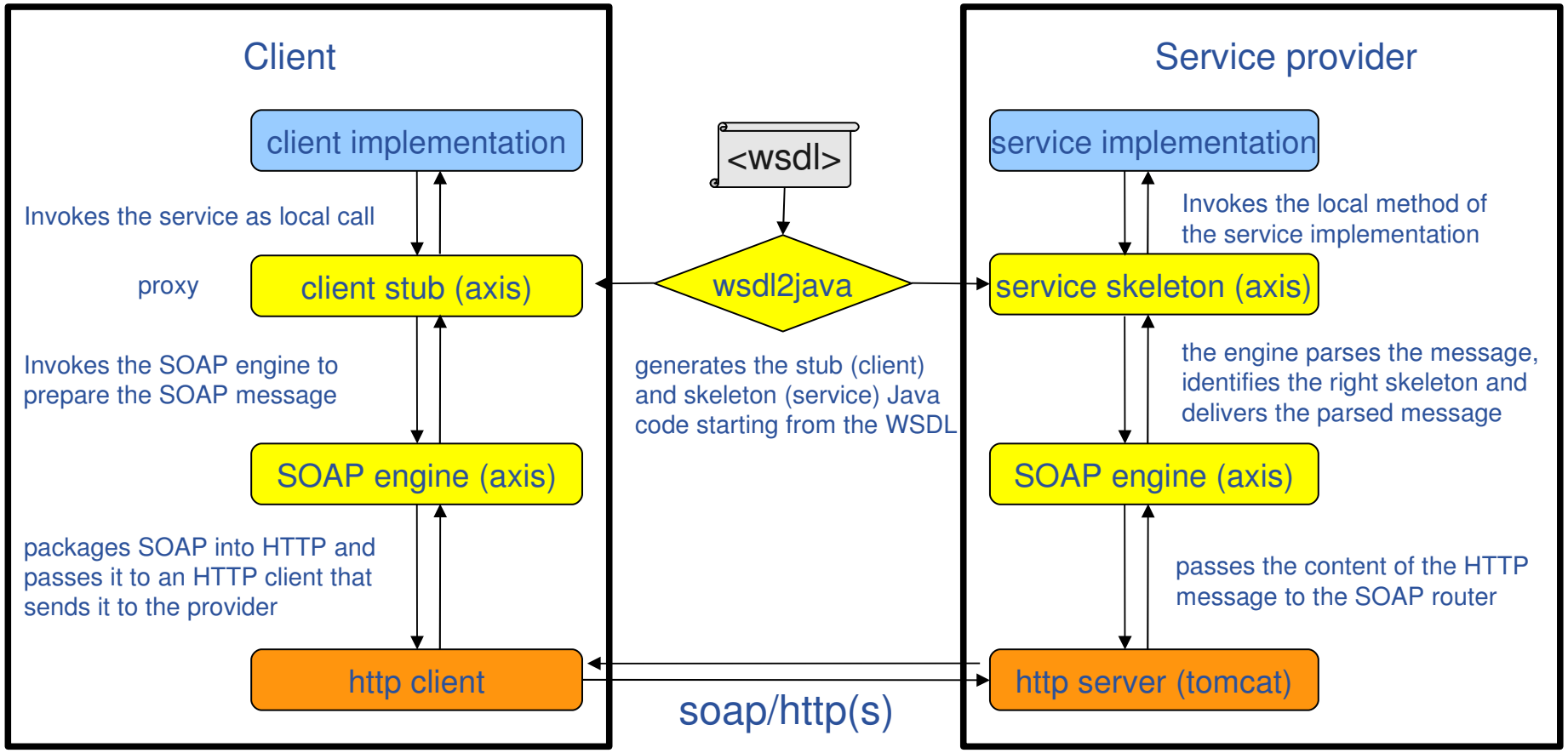


Apache Axis

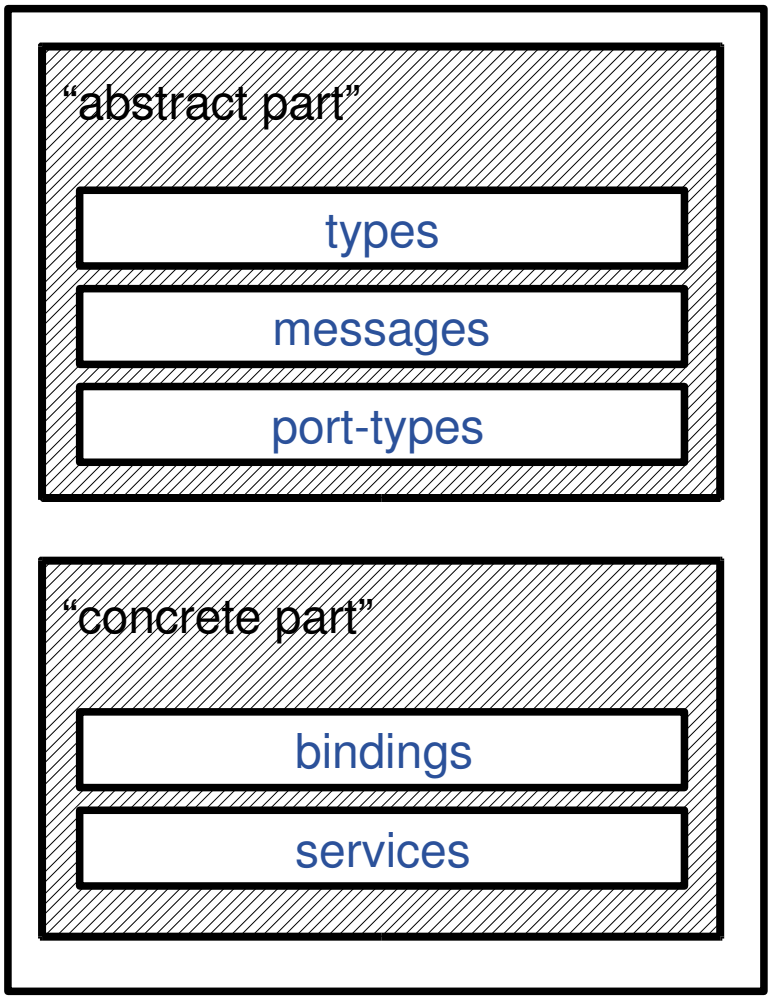
- **Axis (Apache eXtensible Interaction System) is essentially a SOAP engine (a framework for constructing SOAP processors such as clients, servers, gateways, etc)**
  - support for Java and C++ available
- **Axis also includes:**
  - a simple stand-alone server
  - a tool for monitoring TCP/IP packets.
  - a server for servlet engines such as Tomcat
  - emitter tooling that generates Java classes from WSDL and vice versa
    - Java2WSDL tool for building WSDL starting from Java classes (bottom-up approach).
    - WSDL2Java tool for building Java source code (stubs and skeletons) starting from WSDL documents (top-down approach)
      - *Stub (client side):*
        - contains the code which turns the method invocations into SOAP calls using the Axis Service and Call objects. It stands in as a proxy for the remote service, letting you call it exactly as if it were a local object. The stub hides all communication details for you.
      - *Skeleton (server side):*
        - just as a stub is the client side of a Web Service represented in Java, a skeleton is a Java framework for the server side.



Client and Service provider



WSDL structure




```
<wsdl:definitions name="nmtoken" targetNamespace="uri"?>
  <import namespace="uri" location="uri"/>*
  <wsdl:types> ?
    <xsd:schema .... />*
  </wsdl:types>
  <wsdl:message name="nmtoken"> *
    <part name="nmtoken" element="qname"? type="qname"?/> *
  </wsdl:message>
  <wsdl:portType name="nmtoken">*
    <wsdl:operation name="nmtoken">*
      <wsdl:input name="nmtoken"? message="qname"/>?
      <wsdl:output name="nmtoken"? message="qname"/>?
      <wsdl:fault name="nmtoken" message="qname"/> *
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="nmtoken" type="qname">*
    <wsdl:operation name="nmtoken">*
      <wsdl:input /> ?
      <wsdl:output /> ?
      <wsdl:fault name="nmtoken"/> *
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="nmtoken" /> *
    <wsdl:port name="nmtoken" binding="qname"> *
  </wsdl:service>
</wsdl:definitions>
```

CREAM WSDL: types (1/2)

```
<wsdl:definitions name="CREAM" ....>
  <wsdl:types>
      <xsd:schema xmlns:cream_types="http://glite.org/2007/11/ce/cream/types"
    ....  >

        <xsd:complexType name="Property">
          <xsd:sequence>
            <xsd:element name="name" type="xsd:string" />
            <xsd:element name="value" type="xsd:string" />
          </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="ServiceMessage">
          <xsd:sequence>
            <xsd:element name="type" type="xsd:string" />
            <xsd:element name="message" type="xsd:string" />
            <xsd:element name="timestamp" type="xsd:dateTime" />
          </xsd:sequence>
        </xsd:complexType>
        ...
```

CREAM WSDL: types (2/2)

```
        <xsd:complexType name="ServiceInfo">
          <xsd:sequence>
            <xsd:element name="interfaceVersion" type="xsd:string" />
            <xsd:element name="serviceVersion" type="xsd:string" />
            <xsd:element name="description" type="xsd:string" minOccurs="0" />
            <xsd:element name="startupTime" type="xsd:dateTime" />
            <xsd:element name="doesAcceptNewJobSubmissions" type="xsd:boolean" />
            <xsd:element name="status" type="xsd:string" minOccurs="0" />
            <xsd:element name="property" type="cream_types:Property" minOccurs="0"
maxOccurs="unbounded" />
            <xsd:element name="message" type="cream_types:ServiceMessage"
minOccurs="0" maxOccurs="unbounded" />
          </xsd:sequence>
        </xsd:complexType>

        <xsd:element name="ServiceInfoRequest" type="xsd:int" />
        <xsd:element name="ServiceInfoResponse" type="cream_types:ServiceInfo" />
    </wsdl:types>
```

CREAM WSDL: messages

```
<wsdl:definitions name="CREAM" ....>
  <wsdl:types>
     <xsd:schema>
  </wsdl:types>

  <wsdl:message name="ServiceInfoRequest">
    <wsdl:part name="verbosityLevel" element="cream_types:ServiceInfoRequest" />
  </wsdl:message>

  <wsdl:message name="ServiceInfoResponse">
    <wsdl:part name="ServiceInfoResponse" element="cream_types:ServiceInfoResponse" />
  </wsdl:message>
```

CREAM WSDL: portType

```
<wsdl:definitions name="CREAM" ....>
   <wsdl:types>
   <wsdl:message>

   <wsdl:portType name="CREAMPort">

    <wsdl:operation name="getServiceInfo">
      <wsdl:input message="cream:ServiceInfoRequest" />
      <wsdl:output message="cream:ServiceInfoResponse" />
      <wsdl:fault name="authorizationFault" message="cream:Authorization_Fault"/>
      <wsdl:fault name="genericFault" message="cream:Generic_Fault"/>
    </wsdl:operation>

        ...
   </wsdl:portType>
```

CREAM WSDL: binding

```
<wsdl:binding name="CreamBinding" type="cream:CREAMPort">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />

    <wsdl:operation name="getServiceInfo">
        <soap:operation
            soapAction="http://glite.org/2007/11/ce/cream/ServiceInfo"
            style="document" />
        <wsdl:input>
            <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal" />
        </wsdl:output>
        <wsdl:fault name="authorizationFault">
            <soap:fault use="literal" name="authorizationFault" />
        </wsdl:fault>
        <wsdl:fault name="genericFault">
            <soap:fault use="literal" name="genericFault" />
        </wsdl:fault>
    </wsdl:operation>
    ...
</wsdl:binding>
```

CREAM WSDL: service

```
<wsdl:definitions name="CREAM" ....>
   <wsdl:types>
   <wsdl:message>
   <wsdl:portType>
   <wsdl:binding>

   <wsdl:service name="CREAM">
      <wsdl:port name="CREAM" binding="cream:CreamBinding">
         <soap:address location="http://localhost:8080/ce-cream/services/CREAM" />
      </wsdl:port>
   </wsdl:service>

</wsdl:definitions>
```

wsdl2java: stub and skeleton

```
$ java org.apache.axis.wsdl.WSDL2Java --server-side --skeletonDeploy true org.glite.ce-cream_service.wsdl
```

- **parameters:**
  - --server-side: emit server-side bindings for web service
  - --skeletonDeploy true: deploy skeleton in deploy.wsdd
- **input: org.glite.ce-cream_service.wsdl**
- **output: java source files and deployment descriptors:**
  - CreamBindingImpl.java (service's business logic implementation)
  - CreamBindingSkeleton.java (service skeleton)
  - CreamBindingStub.java (client stub)
  - CREAMLocator.java (gives to a client access to service)
  - CREAMPort.java (java API)
  - deploy.wsdd/undeploy.wsdd (deployment descriptors)

CREAMPort.java

```
/**
 * CREAMPort.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis 1.4 Apr 22, 2006 (06:55:48 PDT) WSDL2Java emitter.
 */

package org.glite.ce.creamapi.ws.cream2;

public interface CREAMPort extends java.rmi.Remote {
    public ServiceInfo getServiceInfo(int verbosityLevel) throws java.rmi.RemoteException, GenericFault, AuthorizationFault;
    public void acceptNewJobSubmissions(boolean acceptNewJobSubmissions) throws java.rmi.RemoteException, GenericFault, AuthorizationFault;

    public JobRegisterResult[] jobRegister(JobDescription[] jobRegisterRequest) throws java.rmi.RemoteException, JobSubmissionDisabledFault,
        GenericFault, AuthorizationFault, InvalidArgumentFault;

    public Result[] jobCancel(JobFilter jobCancelRequest) throws java.rmi.RemoteException, GenericFault, AuthorizationFault, InvalidArgumentFault;

   ....

}
```

CreamBindingImpl.java

```
/**
 * CreamBindingImpl.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis 1.4 Apr 22, 2006 (06:55:48 PDT) WSDL2Java emitter.
 */

package org.glite.ce.creamapi.ws.cream2;

public class CreamBindingImpl implements org.glite.ce.creamapi.ws.cream2.CREAMPort {

    public ServiceInfo getServiceInfo(int verbosityLevel) throws java.rmi.RemoteException, GenericFault, AuthorizationFault {
        return null;

        ServiceInfo serviceInfo = new ServiceInfo();
        serviceInfo.setStartupTime(Calendar.getInstance());
        serviceInfo.setDoesAcceptNewJobSubmissions(false);
        serviceInfo.setStatus("RUNNING");
        serviceInfo.setDescription("CREAM 2");
        serviceInfo.setInterfaceVersion("2.0");
        serviceInfo.setServiceVersion("2.0");
        serviceInfo.setProperty(some_properties);
        ...
        return serviceInfo;
    }

    public void acceptNewJobSubmissions(boolean acceptNewJobSubmissions) throws java.rmi.RemoteException, GenericFault, AuthorizationFault {
    }

    public JobRegisterResult[] jobRegister(JobDescription[] jobRegisterRequest) throws java.rmi.RemoteException, JobSubmissionDisabledFault, GenericFault,
AuthorizationFault, InvalidArgumentFault {
        return null;
    }

    ....
}
```

The real implementation is actually much richer.
For brevity we just show you the basic code..

Client side: how to invoke CREAM

```
public class getServiceInfoTest {

  public static void main(String[] args) throws Exception {
    URL creamURL = new URL("https://cream-01.pd.infn.it:8443/ce-cream/services/CREAM2");
    CREAMLocator creamLocator = new CREAMLocator();
    CREAMPort cream = creamLocator.getCREAM(creamURL);

    ServiceInfo serviceInfo = cream.getServiceInfo(0);
    System.out.println("service description: " + serviceInfo.getDescription());
    System.out.println("interface version: " + serviceInfo.getInterfaceVersion());
    System.out.println("service version: " + serviceInfo.getServiceVersion());
    System.out.println("service startup time: " + serviceInfo.getStartupTime().getTime());
    System.out.println("service status: " + serviceInfo.getStatus());
    ....
  }
}
```

getServiceInfoTest

```
$ java getServiceInfoTest

service description: CREAM 2
interface version: 2.0
service version: 2.0
service startup time: Tue Dec 02 12:36:18 CET 2008
service status: RUNNING
messages: 0
properties: 1
        0) cemon_url = https://cream-14.pd.infn.it:8443/ce-monitor/services/CEMonitor
```

considerations

- **The Web service technology**
  - based on Web technologies
    - standard and protocols "open"
    - using an HTTP-based protocol move easily through firewall
  - provides an high degree of:
    - interoperability with clients written in different programming languages
    - integration among different applications running on different platforms software/hardware;
  - not based on a specific programming language: Java, .Net, C, C++, Python, Perl, …
  - not based on a programming data model: objects vs non-objects environments.
  - WS frameworks allow to developers to create, publish and use Web Service easy
    - the developer must just implement the business logic confiding on the WS framework about the low level details (WS protocols)
    - several activities automatized (deployment, SOAP protocol implementation, source code generation, etc)
  - performance poor: WS not useful for application requiring high performance

Documentation

- **Available in the CREAM web site**
  - http://grid.pd.infn.it/cream
- **Release notes**
- **Administrator guides**
  - Installation and configuration instructions for CREAM (manual and via yaim), CREAM-CLI and ICE
- **User guides**
  - CREAM User's guide
  - CREAM JDL specification
- **Info useful for troubleshooting**
  - For users and admins
  - Log files to check
  - Some FAQs
    - Relevant error messages and their meaning
    - More work needed
- **Papers and presentations**
- **Test results**
  - Actually many on them done on old CREAM implementations

That's all !

- **Contact us:**

  jra1-pd@pd.infn.it

  - Paolo Andreetto (EGEE-III SA3, formerly OMII-EUROPE and EGEE JRA1)
    - Internal release manager, Etics confs, CREAM pre-certification
  - Sara Bertocco (EGEE-III SA3, formerly EGEE-II SA1)
    - CREAM integration, yaim, CREAM pre-certification
  - Alvise Dorigo (EGEE-* JRA1)
    - CREAM and CEMon C clients, ICE
  - Eric Frizziero (e-NMR, formerly Cyclops)
    - CREAM and CEMon, CREAM-GDSE integration
  - Alessio Gianelle (EGEE-III SA3, formerly EGEE-II SA3 and EGEE JRA1)
    - CREAM pre-certification
  - Moreno Marzolla (EGEE-III JRA1, formerly OMII-EUROPE and EGEE JRA1)
    - CREAM and CEMon C clients, ICE, CREAM-BES
  - Massimo Sgaravatto (EGEE* JRA1)
    - Testing, overall coordination
  - Luigi Zangrando (EGEE* JRA1)
    - CREAM and CEMon