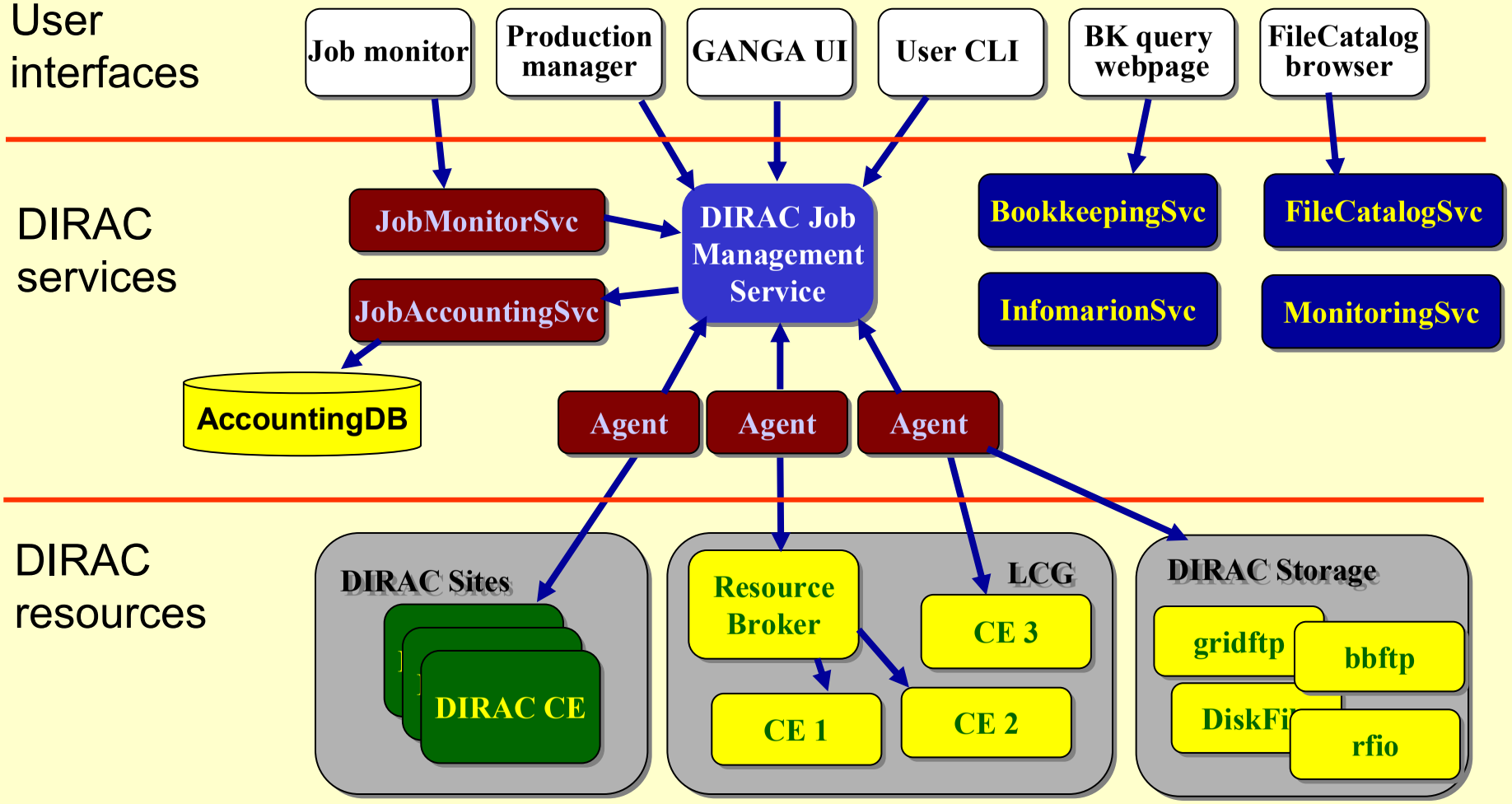
Using services in DIRAC

A.Tsaregorodtsev,
CPPM, Marseille

2nd ARDA Workshop, 21-23 June 2004, CERN

DIRAC Services and Resources



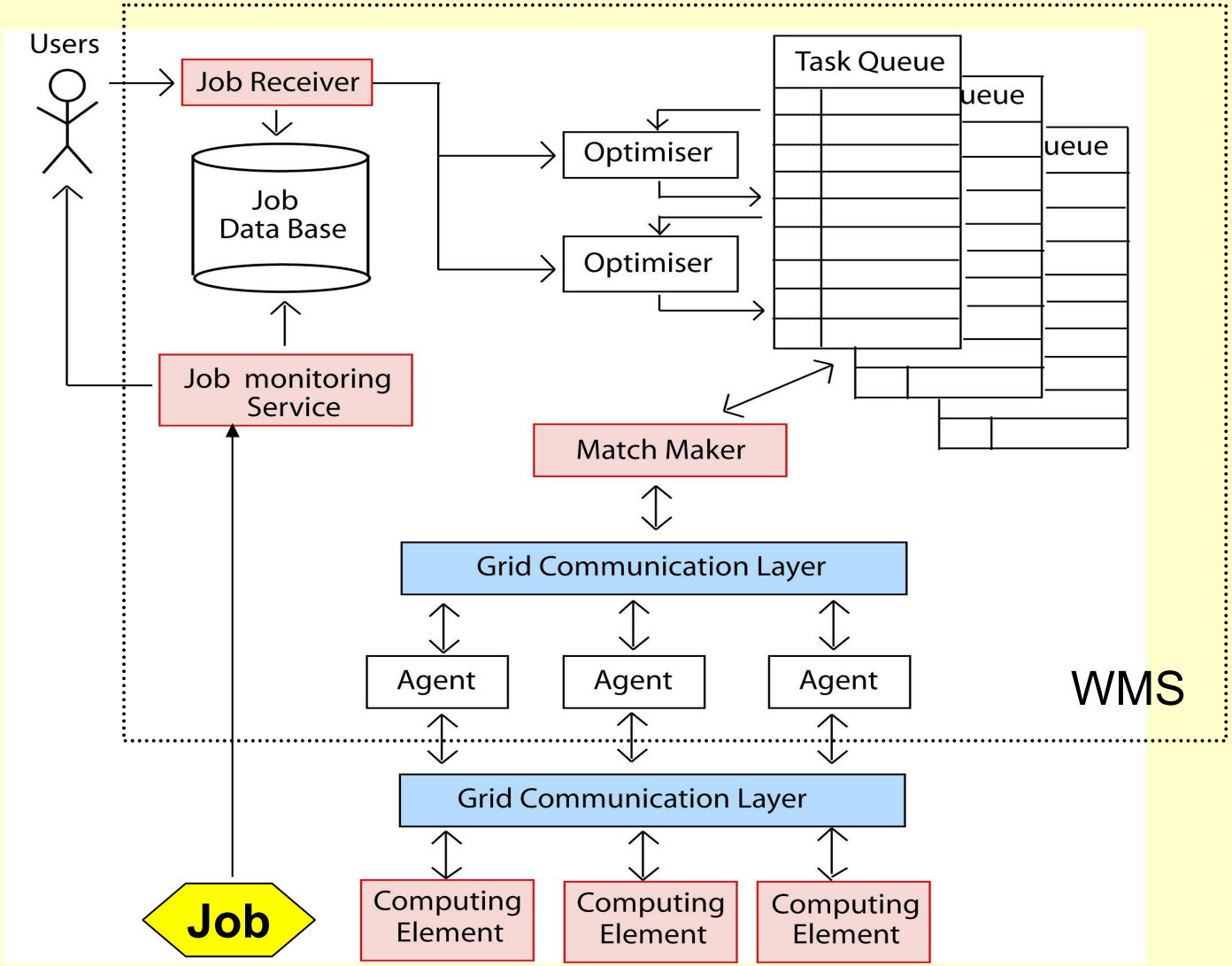
GT3 attempt

- JobReceiver prototype Grid Service was done
  - Service gets jobs, checks its integrity and stores in the Job DB (MySQL)
  - Notifies Optimizers via Jabber
- Uses Tomcat and GT3

DIRAC WM architecture

-Submit (JDL)

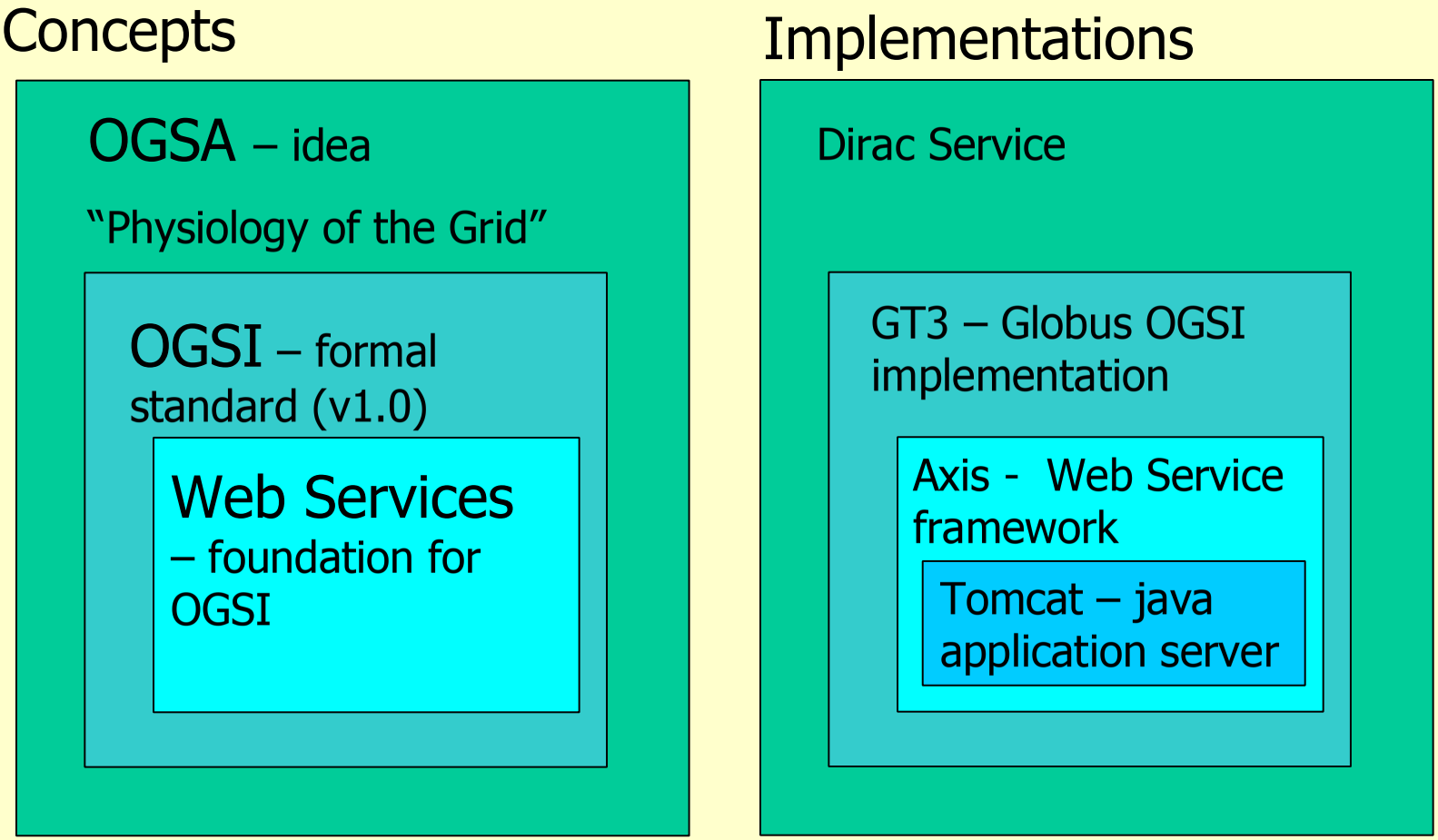
-Status job



Globus Toolkit 3 (aka GT3)

- What is GT3? The Globus implementation of OGSI
  - Really just the new version of Globus, prepared predominantly by Argonne National Labs (**ANL**) and **IBM**
  - Re-implements much of Globus Toolkit v2.4, but in **Java**
  - Based on Web Services, and runs from Tomcat Java Application Server

OGSI vs. GT3


Concepts
OGSA – idea
"Physiology of the Grid"
OGSI – formal standard (v1.0)
Web Services – foundation for OGSI
Implementations
Dirac Service
GT3 – Globus OGSI implementation
Axis - Web Service framework
Tomcat – java application server

GT3 attempt results

- GT3 is currently difficult to install, configure, and administer
  - took 2+ weeks to get GT3 fully installed and running properly
  - discuss@globus.org mailing list is full of similar stories
- Service development and deployment is not straight forward
  - Must define GWSDL, WSDD, XML Schemas, etc. (don't worry about acronyms)
- Service performance is very low:
  - More than a minute to submit one job
- Good news: Client side is not too complicated, and should be language independent.

Other OGSI Implementations

- OGSI::Lite
  - Perl version developed at Manchester
- pyGridWare
  - 100% python OGSI implementation
  - Currently: client maybe ready, server isn't
  - We are interested because
    - Possibility of Ganga OGSI client
    - Expect it to be lightweight, easy to deploy, robust, and high performance

      MUST CONFIRM THESE EXPECTATIONS

XML-RPC protocol

- Standard, simple, available out of the box in the standard python library
  - Both server and client
  - Using expat XML parser
- Server
  - Very simple socket based
  - Multithreaded
- Client
  - Dynamically built service proxy

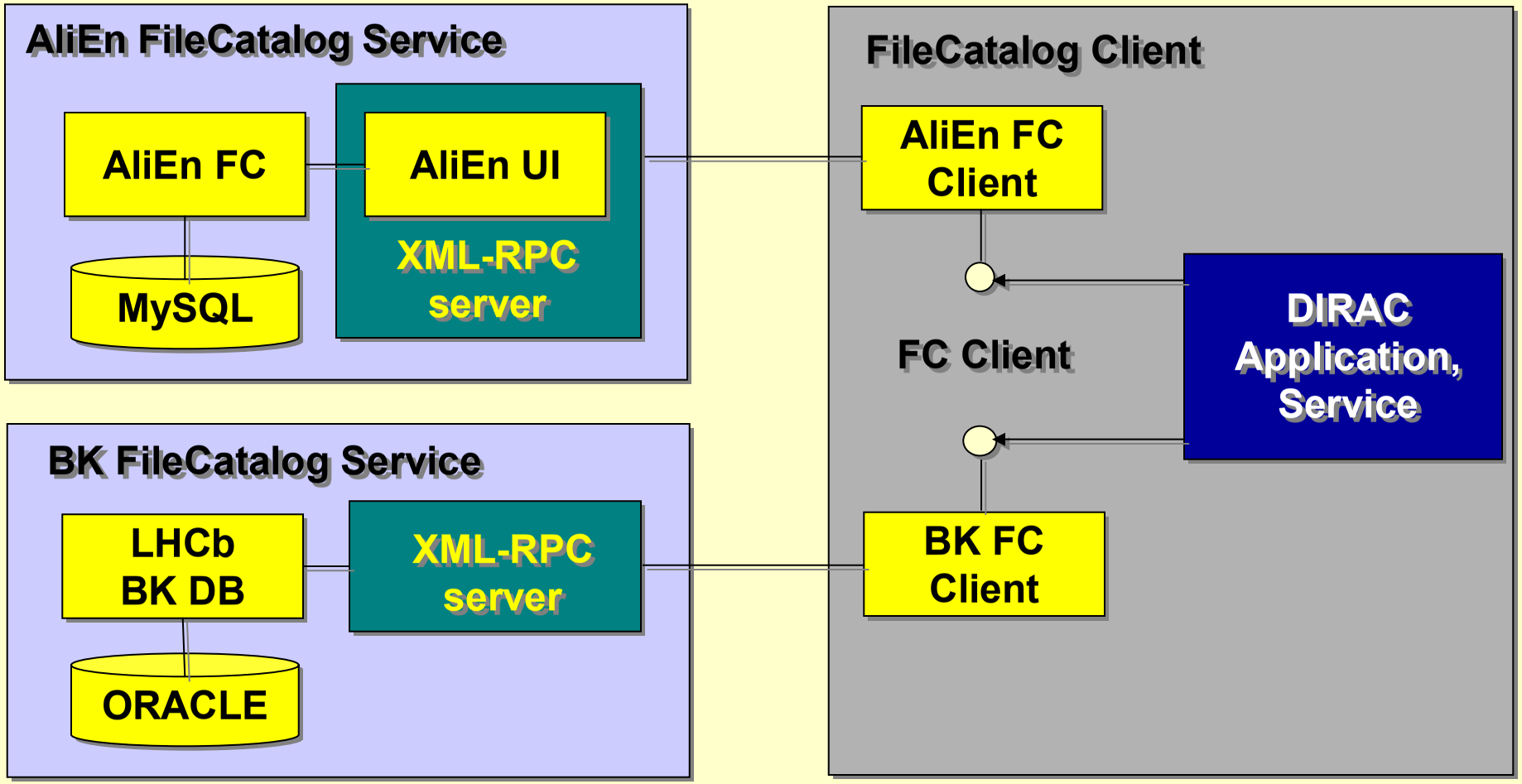
File catalog service

- The LHCb Bookkeeping was not meant to be used as a File (Replica) Catalog
  - Main use as Metadata and Job Provenance database
  - Replica catalog based on specially built views
- AliEn File Catalog was chosen to get a (full) set of the necessary functionality:
  - Hierarchical structure:
    - Logical organization of data – optimized queries;
    - ACL by directory;
    - Metadata by directory;
    - File system paradigm;
  - Robust, proven implementation
  - Easy to wrap as an independent service:
    - Inspired by the ARDA RTAG work

AliEn FileCatalog in DIRAC

- AliEn FC SOAP interface was not ready in the beginning of 2004
  - Had to provide our own XML-RPC wrapper
    - Compatible with XML-RPC BK File Catalog
- Using AliEn command line “alien –exec”
  - Ugly, but works
- Building service on top of AliEn which is run by the lhcbprod AliEn user
  - Not really using the AliEn security mechanisms
- Using AliEn version 1.32
- So far in DC2004:
  - >100’000 files with >250’000 replicas
  - Very stable performance

File catalogs



FileCatalog common interface

```
addFile(lfn,guid,size,SE<opt>,pfn<opt>,poolType<opt>)
rmFile(lfn)
rmFileByGuid(guid)
addPfn(lfn,pfn,SE<opt>)
addPfnByGuid(guid,pfn)
removePfn(lfn,pfn)
getPfnsByLfn(lfn)
getPfnsByGuid(guid)
getGuidByLfn(lfn)
getLfnByGuid(guid)
exists(lfn)
existsGuid(guid)
getFileSize(lfn)
```

Reliability issues

- Regular back-up of underlying database
- Journaling of all the write operations
- Running more than one instance of the service
  - Configuration service running at CERN and in Oxford
- Running services reliably:
  - Runit set of tools

Runit service management tool

http://smarden.org/runit/

- Runit package features
  - Automatic restart on failure
  - Automatic logging of standard output with time/date stamps
  - Automatic log rotation
  - Cleanup scripts on process completion
  - Simple interface to pause, stop, and send control signals to services
  - Runs service in daemon mode
  - Can be used entirely in user space
- Light-weight service management

Using Instant Messaging

- Mechanism to pass messages across the network (ICQ, MSN, etc):
  - Asynchronous, robust
  - Client needs only outbound connectivity for bidirectional message exchange
- Jabber as IM protocol
  - Open source, mature server implementation
  - Using XML for message formatting
  - Possibility to build a network of Jabber servers
    - Load balancing, proxies
  - Client API exists in many languages:
    - Python included

Using Instant Messaging (2)

- Initially used for WMS components communication:
  - JobReceiver notifies Optimizers using Jabber
- Monitoring of Agents
  - Using Chat Room paradigm where agents are updating their “presence” status
  - Possibility of broadcasting messages to agents
- Monitoring Jobs
  - Jabber client running near the job
  - Access to the job std.out/err
  - Interactive control channel to the job

Using Instant Messaging (3)

- Services communicating via IM protocol
  - XML-RPC over Jabber works just fine
  - Agents deployed on the fly on a computing resource can work as a fully functional service
    - With outbound connectivity only (no firewall problems)
    - Using this paradigm to run jobs on LCG now
  - Proxy for message passing is just a matter of yet another Jabber server
  - Client and server are keeping connection open
    - Authenticate once
    - Scalability issues
      - ➡ Problems when monitoring thousands of jobs

Conclusions

- Building services from the existing components is rather easy;
  - Secure services is not easy
- Having more than one implementation of services is a **must;**
- Instant Messaging is very promising for creating dynamically deployed, light-weight services – agents.