GridPP
UK Computing for Particle Physics

http://dirac.cern.ch

Grid Information and Monitoring System using XML-RPC and Instant Messaging for DIRAC

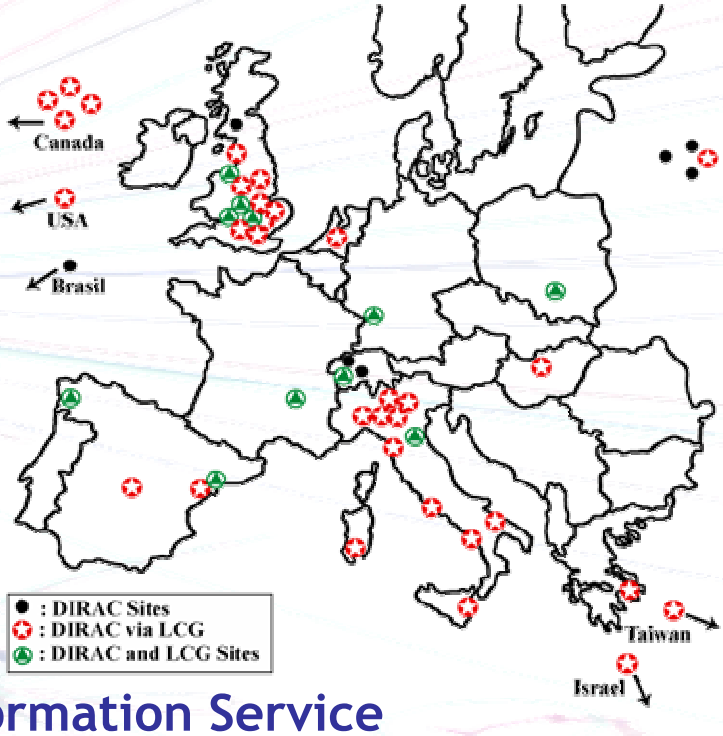
Ian STOKES-REES
University of Oxford

Andrei TSAREGORODTSEV and
Vincent GARONNE
Centre de Physique des
Particules de Marseille

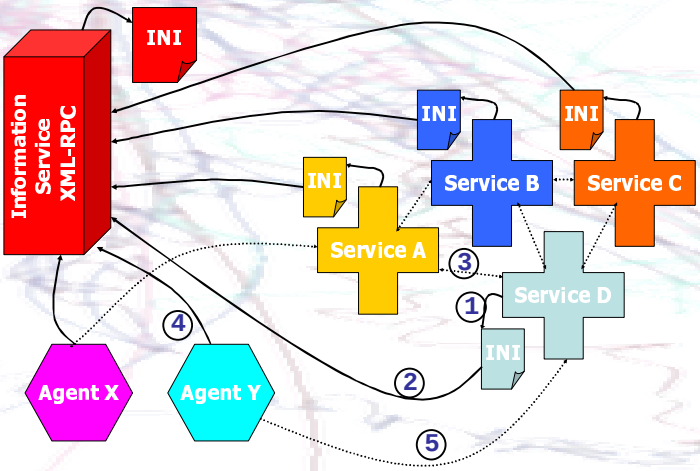
Background

The DIRAC system developed for the CERN LHCb experiment is a grid infrastructure for managing generic simulation and analysis jobs. It enables jobs to be distributed across a variety of computing resources, such as PBS, LSF, BQS, Condor, Globus, LCG, and individual workstations.

A key challenge of distributed service architectures is that there is no single point of control over all components. DIRAC addresses this via two complementary features: a distributed Information System, and an XMPP (Extensible Messaging and Presence Protocol) Instant Messaging framework.



Lightweight Information Service



Problem

Every service and agent needs a combination of:

- local/custom configuration
- global shared configuration
- knowledge of other services configuration

However services and agents can be started, stopped, added, removed, and reconfigured randomly. How does one component learn about the other components, and share its own configuration? This is the classic "Name Service" problem, addressed in other contexts by DNS, LDAP, and UDDI, to name a few.

Solution

A simple information model provides grouped name/value pairs. This is accessed via an API to a single in-memory Information Service object. This object transparently searches for a requested item until found, starting from the local settings (loaded from a file), then trying in turn a list of alternative information sources, which may be remote services or local files.

The local Information Service may cache results from requests to remote sources, or perform batch fetches to improve performance. As well, a fail-over mechanism provides redundancy.

```
[lbnts2]                    <- Section
Modules      = JobAgent     <- Option

[Agent]
LogFile      = agent.log
LogOutput    = stdout,file
LogLevel     = DEBUG

[JobAgent]                  <- Value
CEUniqueIds  = in2p3.fr/pbs-short
AgentName    = TestModularAgent
```

Backdrop *Particle Paintings* created by artist **Tom Kemp** September 2004 *http://www.tomkemp.com*

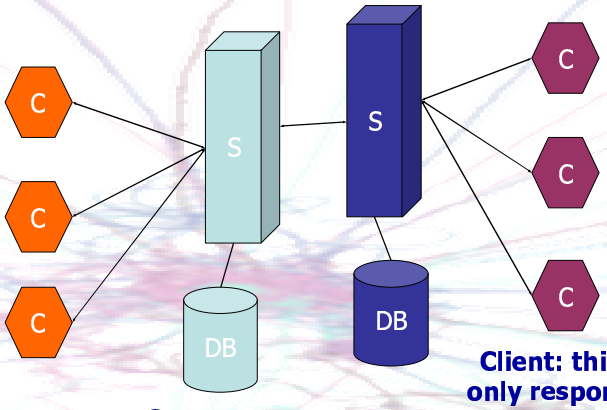
GridPP

UK Computing for Particle Physics

http://dirac.cern.ch

Server: routes and buffers messages, handles presence probes

XMPP-Core IETF Draft
XMPP-IM IETF Draft



DB: Stores user profiles and rosters

Client: thin client, only responsible for processing incoming messages

Problem

In a grid environment it is difficult to locate remote services and running jobs, as the grid topology is dynamic. Agents, Services, Users, and Jobs are constantly coming online or going offline. Furthermore, many grid enabled nodes are protected by firewalls, making "inbound" access difficult or impossible.

A level of indirection is required to enable components to form *ad hoc* networks, allow dynamic addressing for secure remote access, and buffer communication to increase robustness.

Jabber

Grid Monitoring with Instant Messaging

Solution

XMPP (Extensible Messaging and Presence Protocol), the IETF standardisation of the Jabber Instant Messaging Protocol, addresses many of these problems.

***Ad hoc* networks** can be formed by creating a "chat room" which then acts as a message broadcast hub. Components connect to the chat room and then listen to broadcast messages or post messages. Users can also connect to the chat room to track messages and provide monitoring of component state, with only the knowledge of the messaging hub (chat room), rather than the names of specific components.

Roster lists show the names of "subscribed" components, either tracked specifically by a User, or who are members of a chat room. The **presence** mechanism is utilised to report the state of a component.

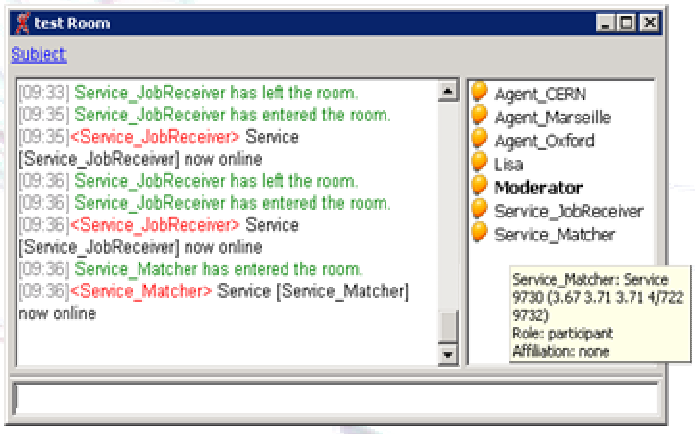
XMPP servers allow for **message logging** which can be utilised for **provenance, accounting, and debugging**.

Dynamic addressing is handled by JIDs (Jabber IDs), allowing a component to be contacted regardless of where it "actually" exists. This enables components to migrate between hosts, and for late binding of a component to a host but early binding of a component to a JID.

Message buffering by the XMPP server provides **fault tolerance** in the presence of network, system, or service timeouts, failures, or restarts. **Asynchronous messaging** also improves service decoupling by removing blocking which occurs with synchronous procedure calls.

Message Types

<presence> availability and status
<iq> pull based RPC mechanism
<message> push based general communication
- Instant message
- One to one chat
- Group chat (broadcast)



Interaction

Users can communicate with components (Services, Agents, or Jobs) **using standard "chat messages"** which are parsed and interpreted by the component XMPP message handler.

The XMPP **IQ (Information/Query) messages** provide a rich RPC mechanism allowing XMPP enabled components to expose a programmatic interface.