Glite 3.1 Release(s)

Laurence Field

Introduction

- The current state of the middleware
- Maturing software and production services
- The new challenges of release management
- Building and Integration
- Requirements on SA3 and JRA1

Heritage

- **Roots of the middleware**
  - European Data Grid Project Project
    - Spring 2001 – Spring 2004
- **Initial Middleware composition**
  - Globus (Gatekeeper, MDS GridFTP)
  - Resource Broker
- **Node Types**
  - SE (GridFTP server, MDS GRIS)
  - CE (Gatekeeper, MDS GRIS and GIIS)
  - RB (Resource Broker)
  - UI (RB clients and GridFTP clients)
- **“Production” Infrastructure with 3-5 sites**
  - A few workers nodes per site

EGEE Today

- **Current size of the grid (Deployment)**
  - 230+ sites in 49 Countries
  - 40K CPUs (Worker Nodes)
  - 50+ VOs
  - 1000s of users (User Interfaces)
  - 100K jobs per day (Feb 2007)
- **Middleware Stack (Release Preparation)**
  - 18 Node Types
    - Not including flavours eg mysql/oracle, torque/lsf etc.
  - More services per node type
    - Eg APEL, DGAS, BDII, CE MON, GSIFTP
      - *All just on the glite CE!*
- **And expanding!!**

The Journey

- **Initial prototyping phase**
  - Full of discovery and innovation
    - Experimentation of ideas
  - Rapidly changing interfaces
    - Clients and server versions coupled
- **Re-design and re-engineering**
  - Solidifying ideas and moving to proven methods
  - Settling on interfaces
    - Clients and server versions more loosely coupled
- **Maturing software**
  - Standardized interfaces
    - Clients and servers coupled via specification
      - *Backwards compatibility between version specifications*
  - Interoperation is a high priority for grid infrastructures
    - Driving the need for standards

Middleware Distribution

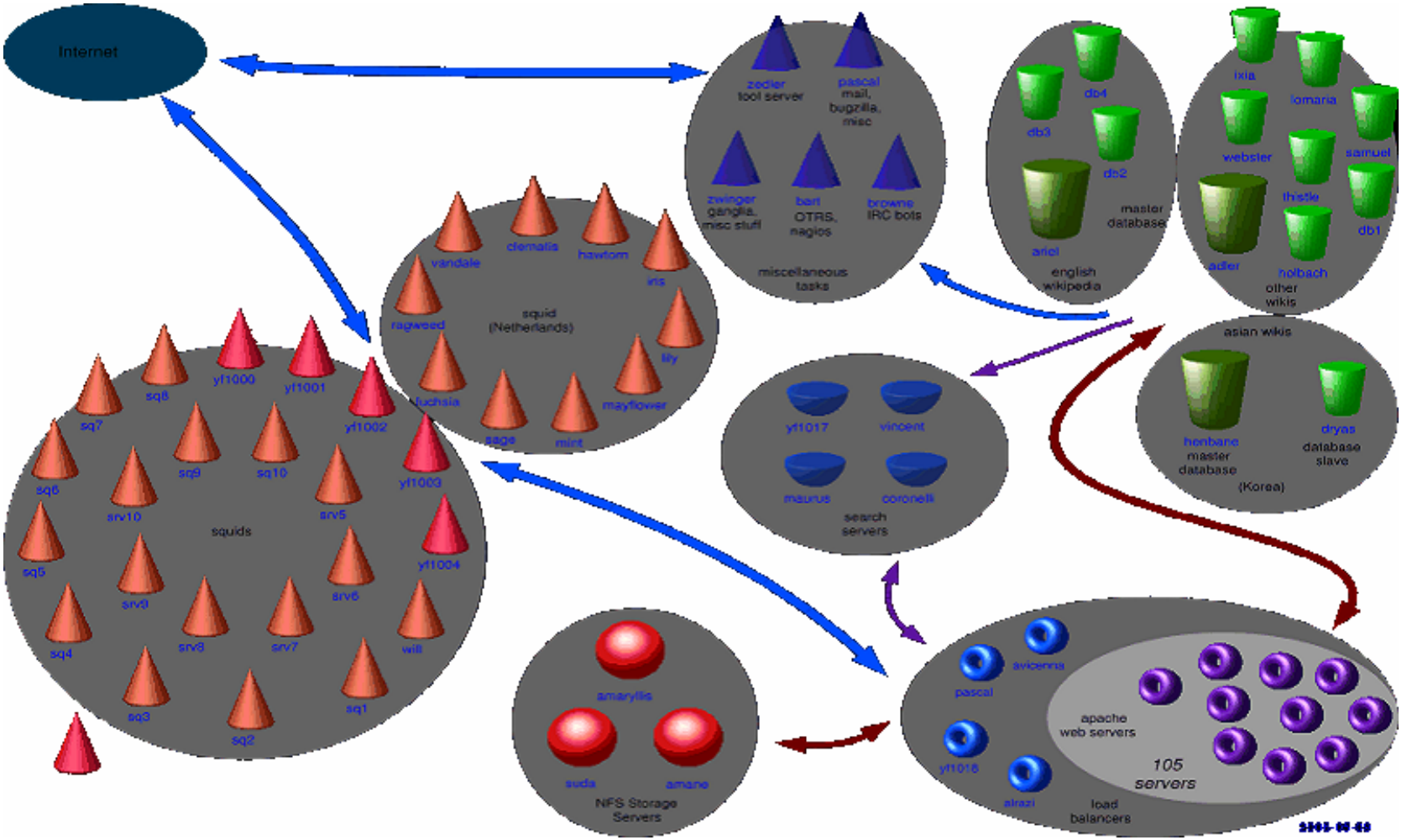
- **Initially few services and tightly coupled interfaces**
  - Release a big blob of middleware
    - To ensure it all works together
    - Requires simultaneous deployment
      - *Break in service*
- **Now a much bigger blob!**
  - Non-related components are tied together
    - One component fails to build, blob can't be released
      - *Probability of this grows bigger with number of components*
      - *Non-critical component affects an update to a critical component*
    - All components given the same priority
      - *Worker Node clients to be deployed on 40K machines*
      - *Same priority as WMS, deployed a few sites*
- **Need to manage components individually**

Apache and Firefox

- **Examples of mature projects**
  - Using a client/server model
- **Each project managed separately**
  - By different distributed development communities
  - Different release schedules and priorities
- **They have different deployment scenarios**
  - Apache: scaleable service, usually Linux, run by administrators
  - Firefox: single machine, many OSs, run by users
- **Each Implements the same standard**
  - All versions work together
    - Upgrade of Apache does not require and upgrade of Firefox!
  - And they are backwards compatible!

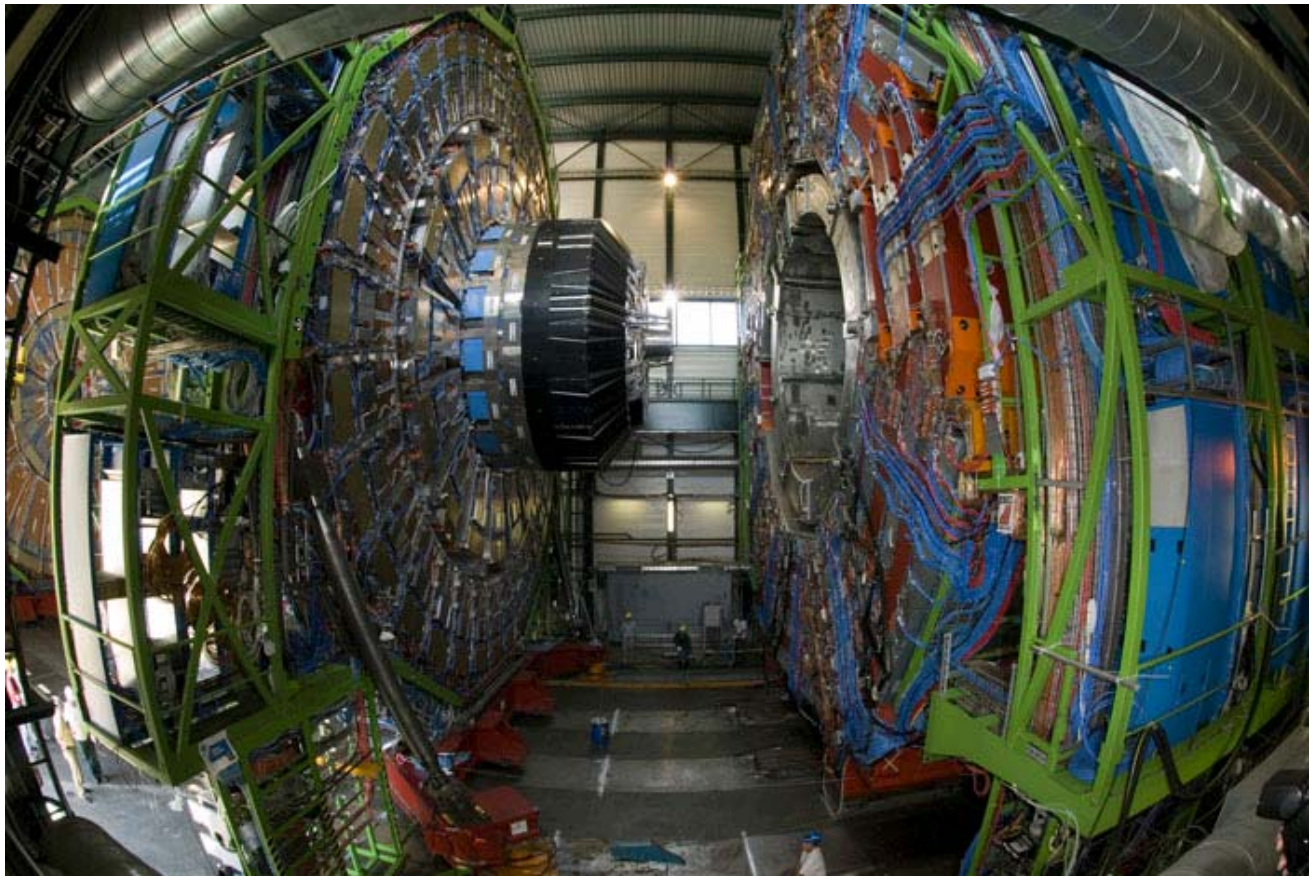
Example Service Deployment

- **Serving an online encyclopedia**



The Real Problem

- **How do we manage the data from this?!**



  - 1,500,000 sets of the Encyclopaedia Britannica per year!
- **Are we ready for September?**

Release Management

- **Updated Software Process seems to be working**
  - Problems: Recorded in Bugs
  - Solutions: Recorded in Patches
- **Improved communication and tracking**
  - Created transparency
    - Developers can check the progress of their patch
  - All results in change!
- **Change creates workload**
  - Workload the same for accepted and rejected patches
  - The amount of change is growing
    - Due to increased services
    - Service ramp-up
- **Need to efficiently manage change**
  - On the component level

The Software Stack

- **Need to break it up into more manageable parts**
  - Manage Individual components
    - Independently
      - *Client should not be tied to server etc.*
      - *Interfaces and APIs should be stable*
        - Ability to rebuild against any library version
  - Release components independently
    - Different releases for each node type
- **Slim down heavy clients**
  - Avoid common libs used by the client including server only functions
- **Cut down on exotic third party dependencies**
  - All add to portability and maintenance problems
    - Is the dependency really required?
      - *Can it not be replaced with something else?*

Repository Management

- **Three package repositories**
  - Certification
  - Pre-production
  - Production
- **Repositories update to include the “patch”**
  - Updated packages
  - Tracked in Savannah
- **Updates bunched for Pre-production and Production**
  - For improved efficiency
  - Documentation produced for each update
- **Management decoupled from the build system**
  - The etics package repository is the interface
    - All packages are considered external
      - *Integrate at the package level*

Build System

- **Packages need to be made available in the repository**
  - So that they can be integrated and tested (Certification)
- **All components should be built against the reference**
  - Essentially what is currently in the Certification repository
- **Ideally the component will be build before patch submission**
  - Or automatically shortly after the patch has been submitted
- **The supported build system is ETICS**
  - It must be used directly following any project conventions
- **Avoid using the glite build system underneath**
  - It will create twice as many problems
  - It is impossible to make any improvements
    - While the glite build system layer is being used underneath
  - It makes the maintenance more difficult
  - Difficult to find where the problems actually are
- **Need to be able to rebuild packages from source**
  - For the porting to other platforms

Avoid Things Like This!

```
init:printf "os.platform=${platformName}
    os.compatible.platform=${platformName}
    platform=${platformName}
    glite.gcc.version=${gcc.version}
    build.type=I
    offline.repository=true
    repository=${repositoryDir}/externals
    bootstrap=true
    etics.dist=true
    quick.build=true" > $HOME/.glite.build.properties
```

- **The Build System is currently like a “house of cards”.**
  - It will all fall down if one piece is removed!

The Problem With Subsystems

- **The use of sub-systems is creating many problems**
  - We need to stop using them!
- **Remove all dependencies on subsystems**
  - Hides the real dependencies
    - This was a huge problem in the glite build system
- **Subsystem grouping is not well defined**
  - Organizational grouping has nothing to do with dependencies
  - Current sub-systems have no overlap with deployment reality
    - eg data vs glite-templates-latex-style
- **Creates a level of indirection**
  - Deployed package with name and version
  - To find any important information need to
    - Travers arbitrary subsystem names and versions
- **Currently investigating removing subsystems**
  - Will probably have to remove the glite build system first
- **The focus should be on components and component dependencies**

Requirements on JRA1

- **Ensure that the ETICS configuration is correct for the component**
  - Build and Runtime dependencies especially
  - And ensure that it will builds against the reference
- **Ensure that one component = one package**
  - Required granularity
- **Ensure that the package contains only the required function**
  - Eg common libs used by the client including server only functions
- **Respond to and fix Savannah bugs**
  - Especially EMT “Tracked” bugs
    - Which should be given special priority
- **Submit a patch for the fix to the bug or bugs**
  - Specify both the ETICS configuration
    - And the package name and versions

Configuration

- **Yaim requires refactoring**
  - For node type based releases
- **Spliting per node type**
  - Including configuration file
  - yaim-core, yaim-gliteCE, yaim-WMS, etc.
- **Will enable us to update the configuration**
  - Independently for separate services
- **Should remove problems associated with patch rejection**
  - Having to undo the configuration for the rejected component
- **Gain improvements in certification efficiency**
  - Processing patches in parallel and independently
- **Enable more contributors**
  - Experts providing configuration functions
    - Eg dcache, condor batch system etc.

Summary

- **The software stack is growing**
  - And so are the number of changes!
- **Feedback from deployment is driving the releases**
  - Need to start managing the software at the component level
    - Should not be a problem for mature software
- **New approach to building and integration is required**
  - Building must be done against the reference
- **Managing the repositories independently of the build system**
  - The interface is the etics repository.
- **Ensure that each component is in ETICS**
  - And the meta-data is correct
    - Especially the dependencies
- **Remove the glite-build system layer ASAP**