Enabling Grids for E-science in Europe

www.eu-egee.org

JRA1 all-hands-meeting, Padova 15.11.2004

Configuring & Managing Web Services for gLite

Joachim Flammer
Integration Team

EGEE is a project funded by the European Union under contract IST-2003-508833

Contents

- Management for web services
- Configuration
  - … the Tomcat approach
  - … the JMX approach
- JMX in a nutshell
- JMX for gLite
- gLiteService and gLiteManager
- gLiteService in action (Demo)
- Summary

Managing Web Services

Web services have several management functionalities that are common to all of them

- **Control of the web service**
  - Configuration
  - change web service dynamically
- **Lifecycle-specific requirements**
  - start & stop
  - check if service is alive (pinging)
  - Produce load statistics
- **Request of service information**
  - to describe how many messages it is processing at a given time
  - to display its identification, its current version number
  - to display its current set of dependencies
- **Manage the performance of web services (goes together with testing)**
  - response time
  - uptime
  - management tool should take the quality of service as input
  - Metering the usage of web services: log number of messages from different users
- **Debugging of services**
  - Make internals visible for debugging, finding of bottlenecks
  - request that the service sends any error message to a named target or file
  - SOAP message monitoring: see incoming/outgoing messages
  - SOAP message logging

Requirements for Web Services

- Several common functionalities are provided by container (e.g. tomcat)
  - Some of the functionality are nice to use
    - e.g. starting/stopping a web service via the tomcat manager
  - Some of the functionality is not enough for us
    - configuration is only static
  - Some functionalities might not be provided at all

- We have to provide
  - a common approach to the management of web services
  - use available techniques where applicable
  - extend techniques where necessary

Configuration – the Tomcat approach

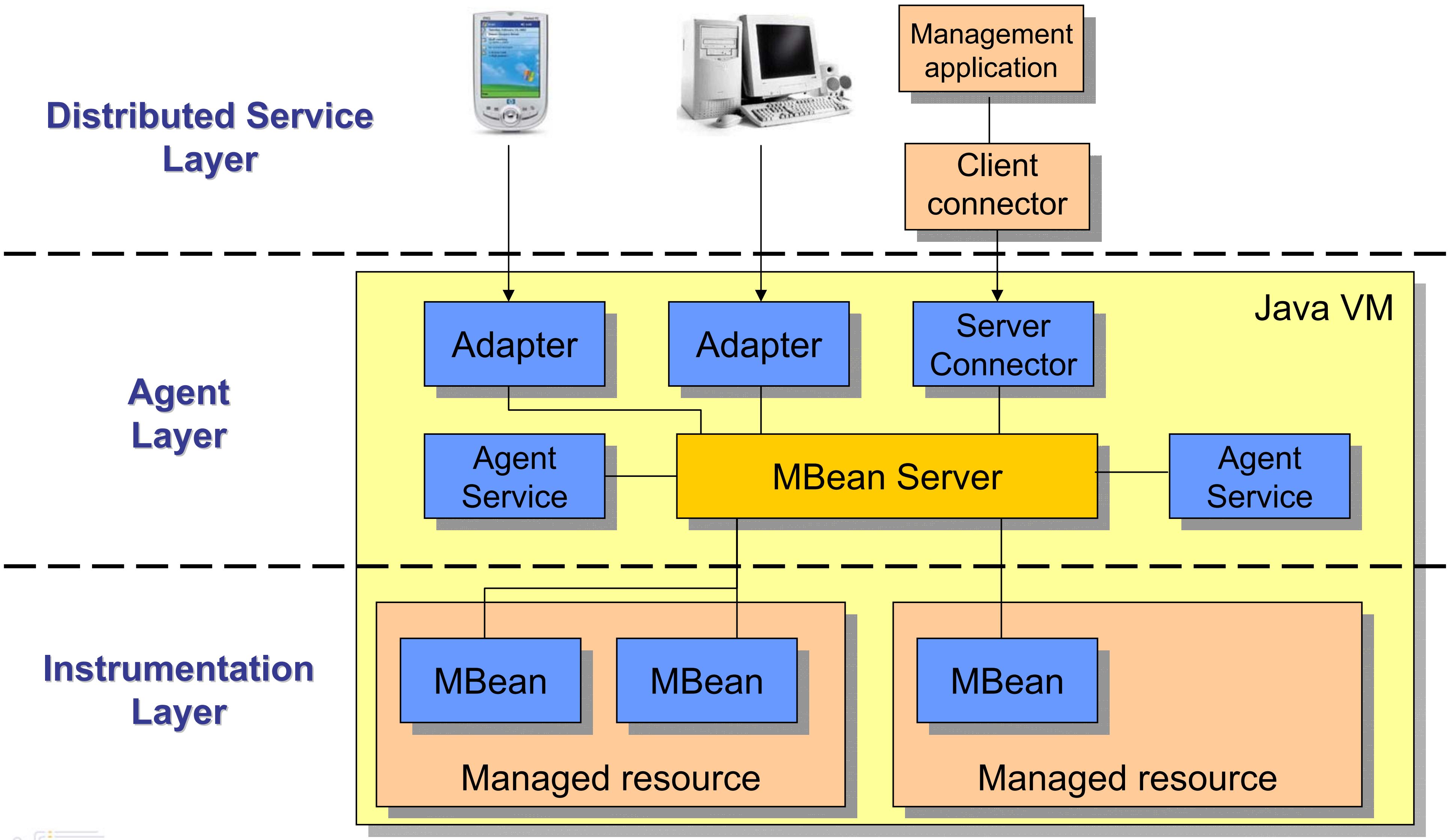
- For Tomcat configuration is done
  - via the context of each web service
  - application can get information via JNDI
- The pros and cons are
  - standard approach ☺
  - pre-configuration is done by tomcat ☺
  - Tomcat JNDI is read only (Tomcat emulates JNDI) ☹
  - No dynamic configuration ☹
  - You cannot get the configuration information from service ☹
  - You cannot get the configuration information from central places ☹

Configuration – the JMX approach

- JMX – Java Management extensions
  - Standard designed for enabling resource management of/with Java applications
  - Extension of Java following standard specification
    - Specification within the Java Community Process (JCP) as a Java Submission Request (JSR3, JSR77, JSR xx Remote)
  - First implementations from 1998
  - Several active implementations – commercial and open source
    - Sun JDMK
    - MX4J
    - …
  - Each implementation follows the standard and gives some extras
  - Integrated in SUN Java 1.5
  - Accepted standard in industry - used in several commercial products
    - HP openview
    - IBM Websphere
    - ….
  - Enables you to do dynamic configuration
  - Enables you to retrieve configuration information remotely
  - Enables you to read configuration from different places
  - …. much more like monitoring etc. …

… and also TOMCAT uses it for its internal configuration …

JMX in a nutshell: Overview



The instrumentation layer: MBeans

```
class Service implements ServiceMBean {

   […]

   protected String name;

   public String getName(){
        return Name;
   }

   public void setName(String name){
        this.name = name;
    }

    public bool updateService(){
         // do something
         return true
    }

    […]
}
```

```
public interface ServiceMBean{

       String getName();
       void setName(String name);
       bool updateService();

}
```

ce

re to MBeans:

eir interface at runtime)

c.

The agent layer: MBeanServer

- **MBeanServer**

```
// creating the MBeanServer
MBeanServer mbs = MBeanServerFactory.createMBeanServer(“glite”);

// querying for an existing MBeanServer in the JVM
List srvList = MBeanServerFactory.findMBeanServer(null);
MBeanServer mbs2 = (MBeanServer) srvList.get(0);

// registering your MBean
Service myService = new Service()
ObjectName myServiceON = new ObjectName(“glite:type=service,port=8080”);
mbs.registerMBean(myService, myServiceON);
mbs.registerMBean(new Service(), new ObjectName(“glite:type=service,port=8090”);

// manipulating MBeans in a server
String name = mbs.getAttribute(myServiceON,”name”);
Attribute attribute = new Attribute(“name”, new String(“gliteService”));
mbs.setAttribute(myServiceON, attribute);
mbs.invoke(myServiceON, “updateService”, null, null);
```

- Monitor MBeans

The distributed layer: Adaptors & Connectors

- All MBeanServer methods are nice – but how do you connect from outside the JVM?
  - Adaptors & Connectors

- **Adaptors**
  - Adaptor is an MBean that listens on a particular port and speaks a particular protocol
  - Example: HTTP adaptor (see Demo later on)

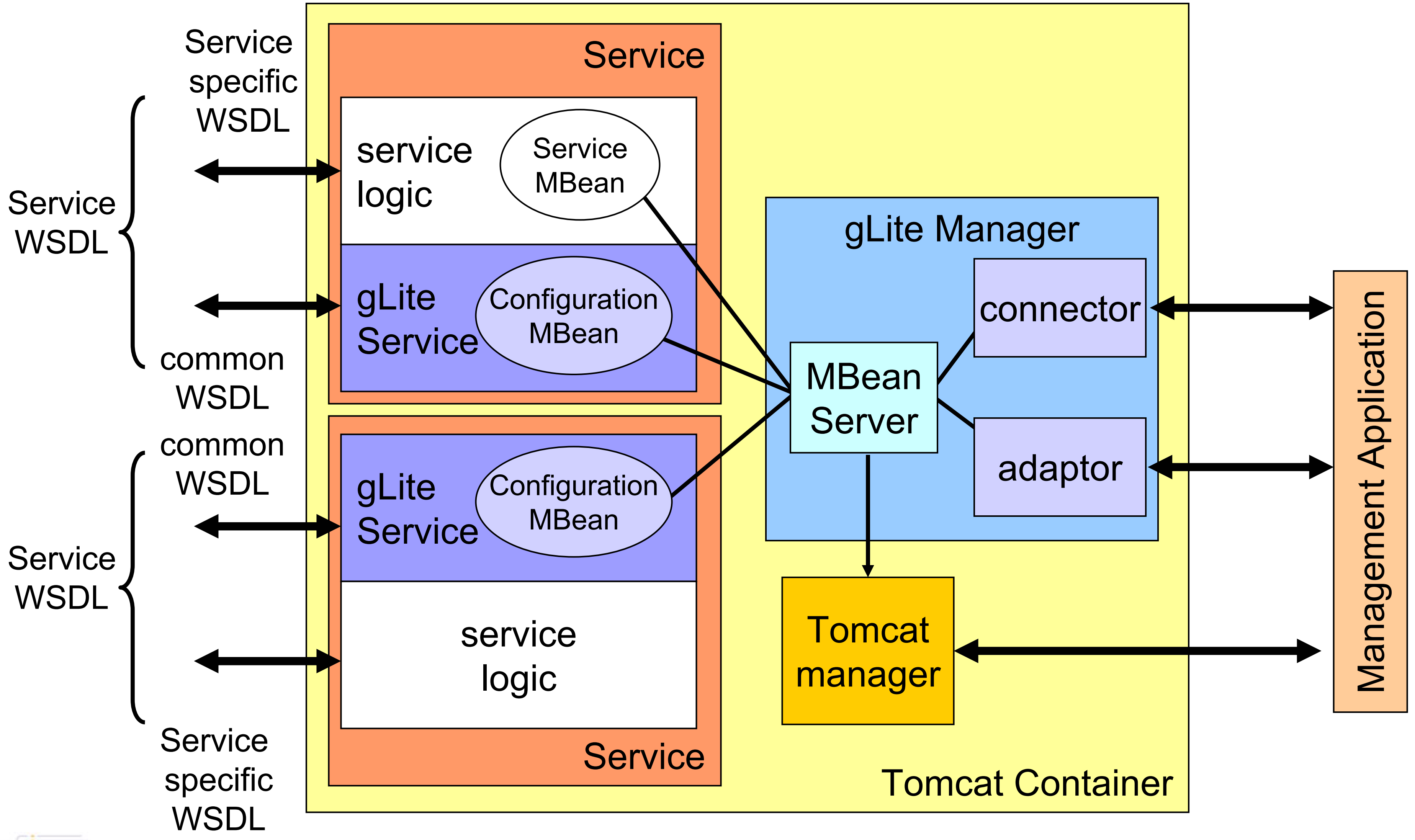
- **Connectors**
  - Connector is an MBean that can co-operate with a peer on a client machine
  - Example: RMI connector (see Demo later on)

- You can register the adaptors/connectors you need/want to support
- All adaptors/connectors are MBeans and can be manipulated like other MBeans

Stay informed: notifications

- You can be informed when MBeans are changed
- MBean can be a source for notifications
  - listen to changes on MBeans by subscribing to notification
  - you can apply filters to notifications
- Information stored in each notification
  - Type (a String) used for filtering
  - SequenceNumber (integer)
  - TimeStamp
  - UserData and Message
  - Source (to identify the generating MBean)

JMX for gLite



Implementation - gLiteService and gLiteManager

- We propose:
  - **gLiteService**
    - Implements the common aspects we want to have for each web service
      - Generic WSDL interface for
        - Version number
        - Ping interface
        - ….
      - Common handling of configuration
    - Each gLite web service will extend this base gLiteService class to implement its functionality
    - gLiteService can reuse/extend functionalities provided by container
  - **gLiteManager**
    - one (lightweight) instance per web server to handle generic stuff
    - contains MBeanServer

Implementation - Some practical details

- What do you have to do to implement it?
  1. Your service extends abstract gLiteService
     - Implement the abstract functions
       – String getServiceName()
       – void reconfigureDynamically()
         (if notifications are included – this will probably go away)
       – …
     - Other methods depend on which common functionalities we want to see
     - **Interface needs to be finalized !!!**
  2. Implement retrieval of configuration values to configure your values
     - see next slide
  3. Implement Reconfiguration
     - dynamic reconfiguration via gLiteService method or via notification
     - static reconfiguration via gLiteManager (nothing to be done for you)
     - **Put as much as possible to dynamic reconfiguration**
  4. Add management to your classes (if you want …)
     - if you want to have more control over your applications: add your own MBeans
     - void registerMBean(Object object, String name);
     - see next slide

Example - Configuring a service

```
// get the "basic" DataSource from JNDI
try {
        Context initCtx = new InitialContext();
        Context envCtx = (Context) initCtx.lookup("glite");
        m_dataSource = (DataSource) envCtx.lookup(m_db_pool_name);
} catch (NamingException e) {
        m_log.error("Got naming error trying to fetch pool: " + pool, e);
        throw new DBException();
}

// configure the DataSource with JMX
try{
      List srvList = MBeanServerFactory.findMBeanServer(null);
      for (int i=0; i<srvList.siz(); i++){
           if (((MBeanServer) srvList.get(i)).getDefaultDomain().compareTo("glite") == 0) {
                mbeanServer = (MBeanServer) srvList.get(i);
                break;
           }
      }
} catch (Exception e) {
    m_log.error("Error in querying for MBeanServer: ", e);
}

try{
    ObjectName configMBeanName = new ObjectName("myService:type:Configuration");
    ((BasicDataSource) m_dataSource).setPassword((String) mbeanServer.getAttribute(configMBeanName,
      "password"));
            […]
} catch (Exception e) {
    m_log.error("Error while configuring DataSource: " , e);
}
```

Example - put manageability to your classes

```
class DbConnection implements DbConnectionMBean{
   …. // see MBean slide
}
```

```
class MyService extends gLiteService{
   […]
   DbConnection dbConnection= new DbConnection;
   registerMBean(dbConnection, “DatabaseConnection”);
   […]
}
```

Next steps

- Agree on implementation details
  - where to put the MBeanServer
  - general methods for each web service
- Choose adaptors, connectors …
  - How do we want to connect to the MBeanServer from outside
    - HTTP
    - RMI
    - SOAP
    - ….
- Security
  - How to make sure that only WE change the settings…
  - There exists security implementations for the different adaptors, connectors
  - Discussion with JRA3
- Discussion needs
  - Present implementation uses “application” scope for axis
    - ↳ is that acceptable ?
- Prepare a detailed description paper with interfaces etc.

gLite Configuration in action

- A little demo
  - data-catalog-service-meta web service
- Demo contains
  - Reading configuration values from configuration files
  - Configuring the database connection
  - Dynamic reconfiguration
  - Static reconfiguration
  - Accessing the configuration from outside via different connectors
  - Monitoring
- Demo contains simplified version
  - everything in one service
  - no notification included yet

Summary

- Management and configuration are very important aspects for web services
- Tomcat offers way to manage/configure service
  - We can (re)use part of the functionality
  - not enough functionality (dynamic, central reconfiguration, ….)
- Java Management Extensions (JMX) is the Java standard for management/configuration/control
- JMX offers
  - easy way to control our applications
  - the developers an easy way to understand what is going on in their application
- Next steps:
  - Agreement
  - Implementation details

Links

- JMX@sun http://java.sun.com/products/JavaManagement/
- Open source JMX implementation

  MX4J http://mx4j.sourceforge.net/

- JMX books
  - JMX in action

    http://www.manning.com/sullins
  - JMX: Managing J2EE with Java Management extensions

    http://www.samspublishing.com/title/0672322889
  - Java Management Extensions

    http://www.oreilly.com/catalog/javamngext/
  - Java and JMX – Building manageable applications

    http://www.awprofessional.com/title/0672324083