Taller de Sakai

Creación de herramientas

Alexandre Ballesté
ASIC – UdL
17 de Junio 2008

Universitat de Lleida

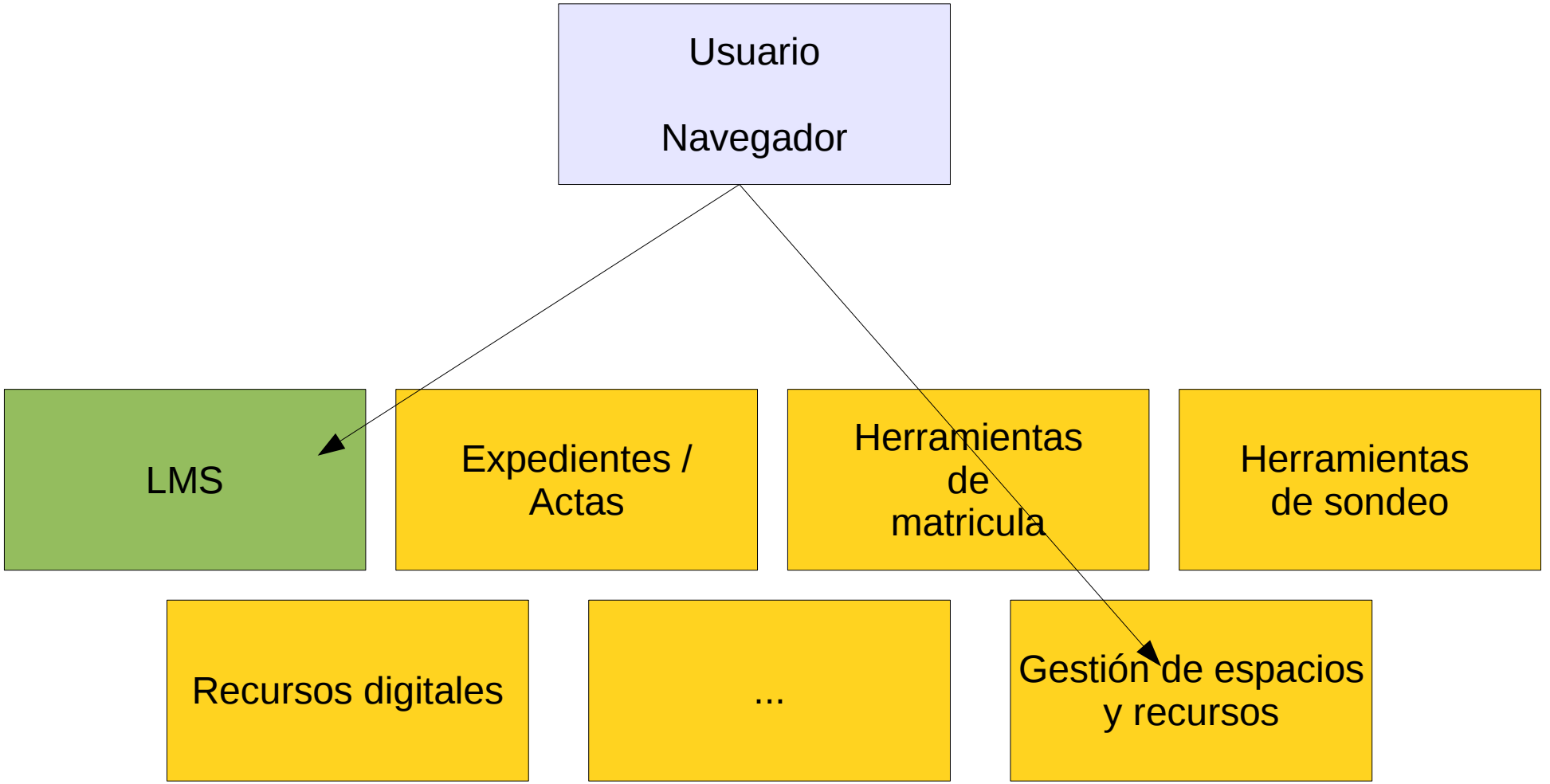
Guía de la presentación

- LMS como portal
- El *framework* Sakai
- Construcción de una herramienta
- Servicios Web en Sakai
- Herramientas remotas

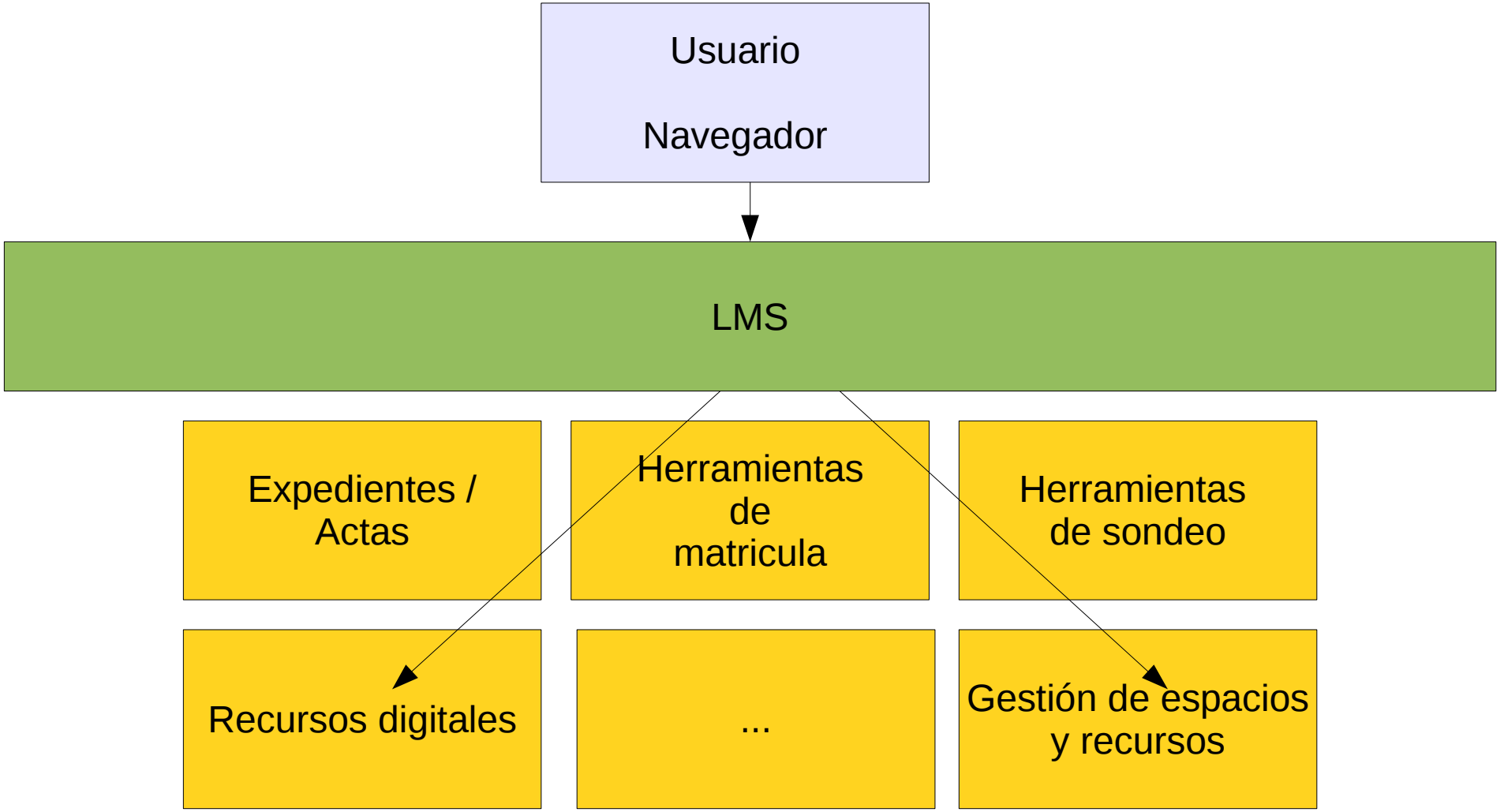
LMS como portal

- El papel que juega el *LMS* dentro de las universidades es cada vez más importante.
- Cada vez existen más fuentes de información y han de ser visibles para los miembros de la comunidad Universitaria.
- Los problemas que nos encontramos normalmente son:
  - Dispersión en el acceso de la información.
  - Limitaciones técnicas para adaptar soluciones existentes.

LMS como portal



LMS como portal



El framework Sakai

- Sakai es un *framework* dirigido a la enseñanza colaborativa.
- Las funciones principales de este son las que conocemos como las de Campus Virtual
- Dispone de un sistema central y de un conjunto de herramientas conectadas a este.

El framework Sakai

- La estructura del LMS Sakai, le permite actuar como figura central del sistema de información
- Por su naturaleza modular, Sakai permite aumentar su ámbito de acción fácilmente
- Sakai se distribuye bajo licencia *ECL* que nos permite usar, estudiar, modificar y redistribuir el producto.

El framework Sakai

- El principal beneficio de este tipo de licencia no es el económico, sino que nos ofrece “libertad” para:
  - Adaptar nuestro modelo de información.
  - Extender funcionalidades base.
  - Mejorar las funcionalidades existentes.

El framework Sakai

- Sakai nos permite desarrollar/integrar:
  - Aplicaciones web
  - Servicios y componentes
  - Servicios web
  - Herramientas remotas

El framework Sakai

- La Udl ha desarrollado herramientas como:
  - Cualificación Actas
  - WS sincronización con *UXXi*
  - Tabla de planificación docente
  - *Provider* OpenLdap
  - Evaluación del profesorado
  - Gestión de cuentas de directorio LDAP
  - *HelpDesk*

El framework Sakai

- Sakai integra una conjunto de tecnologías que se agrupan en funcionalidades y modelos.
- Las funcionalidades ayudan a hacer el proceso de desarrollo menos costoso.
- Los modelos de información proporcionan acceso a información de uso común

El framework Sakai

- Algunas de las funcionalidades mas importantes:
  - *Universal Components*
  - *Collaborative Tool*
  - *DB*
  - *Email Out*
  - *Memory Cache*
  - *Presence*
  - *JSF, RSF, Velocity, etc...*
  - *Util*

El framework Sakai

- Agunos de los modelos de información mas importantes son:
  - *Entity Bus*
  - *User*
  - *Authorization Groups*
  - *Context*
  - *Usage Events*
  - *Course Management*
  - *Alias*
  - *Site*

Construcción de una herramienta

- Aplicaciones internas de Sakai:
  - Se puede integrar a Sakai herramientas web basadas en *JSP/Servlets*.
  - Cada herramienta ocupa un directorio en la raíz de código de Sakai.
  - Se separa en cuatro partes:
    - Aplicación
    - Modelo o *API* del servicio
    - Implementación
    - Definición del componente o *package*

Construcción de una herramienta

- Ejemplo de creación de una herramienta básica de Sakai.
  - Creación de una aplicación
  - Uso de los servicios de Sakai
  - Creación de un servicio propio y su uso

Construcción de una herramienta

- Antes de nada necesitamos tener configurados y funcionando:
  - *Java sdk 1.5*
  - *Maven 2.0.5*
  - *Sakai 2.5.0*

Construcción de una herramienta

Pasos:

1) Crear un directorio en la raíz de *sakai-src* con el nombre de la herramienta

2) Crear dentro de este un fichero *pom.xml* (descriptor general del proyecto)

Construcción de una herramienta

- La definición del proyecto seria algo así:

```
<?xml version="1.0"?>
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <artifactId>base</artifactId>
    <groupId>org.sakaiproject</groupId>
    <version>2.5.0</version>
    <relativePath>../pom.xml</relativePath>
  </parent>
  <name>Ejemplo herramienta Sakai para Taller UdL</name>
  <groupId>cat.udl.taller</groupId>
  <artifactId>taller-sakai-tool</artifactId>
  <packaging>pom</packaging>
  <modules>
        <module>taller-app</module>
 </modules>
```

Construcción de una herramienta

3) Crear el directorio del módulo (*taller-app*)

4) Crear el fichero *pom.xml* (descriptor del módulo de aplicación web):

```
<?xml version="1.0"?>
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <artifactId>taller-sakai-tool</artifactId>
    <groupId>cat.udl.taller</groupId>
    <version>2.5.0</version>
    <relativePath>../pom.xml</relativePath>
  </parent>
```

Construcción de una herramienta

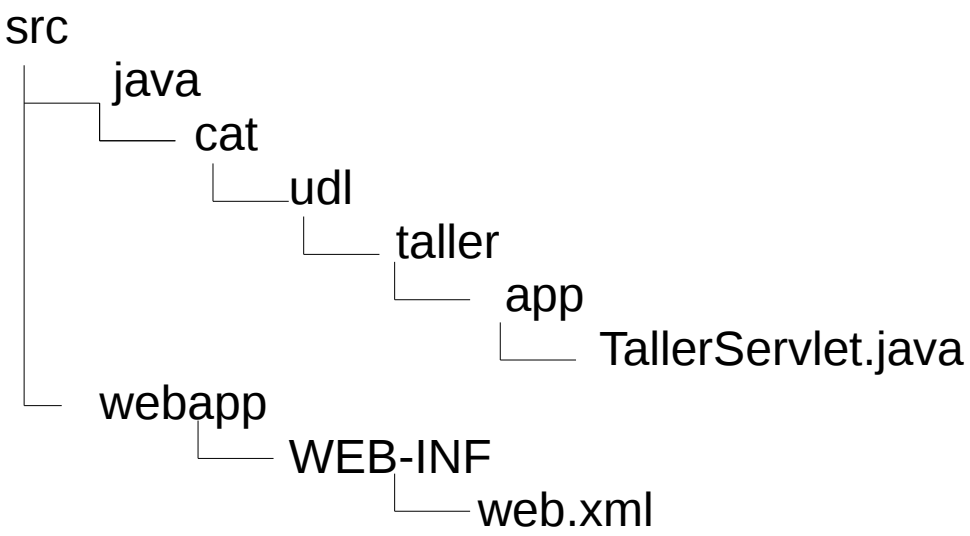
```
<name>taller-app-sample</name>
  <groupId>cat.udl.taller</groupId>
  <artifactId>taller-app-sample</artifactId>
  <organization>
    <name>Universitat de Lleida</name>
    <url>http://www.udl.cat/</url>
  </organization>
  <inceptionYear>2008</inceptionYear>
  <packaging>war</packaging>
  <dependencies>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>${sakai.servletapi.version}</version>
    </dependency>
```

Construcción de una herramienta

```
    <dependency>
      <groupId>commons-logging</groupId>
      <artifactId>commons-logging</artifactId>
      <version>1.0.4</version>
    </dependency>
    <dependency>
      <groupId>org.sakaiproject</groupId>
      <artifactId>sakai-util</artifactId>
      <version>${sakai.version}</version>
    </dependency>
  </dependencies>
  <build>
    <resources/>
  </build>
</project>
```

Construcción de una herramienta

5) Crear la estructura de directorios y ficheros de la aplicación web



Construcción de una herramienta

6) Crear la carpeta “*tools*” dentro de *webapp*

7) Crear el fichero descriptor de aplicación de sakai. (*sakai.taller.xm*l)

```
<?xml version="1.0"?>

<registration>

    <tool       id="sakai.taller" title="Herramienta del Taller"
                description="Herramienta de ejemplo para la construcción de aplicaciones
                 en sakai" >

        <category name="course" />
        <configuration name="parametro1" value="valor por defecto" />
    </tool>

</registration>
```

Construcción de una herramienta

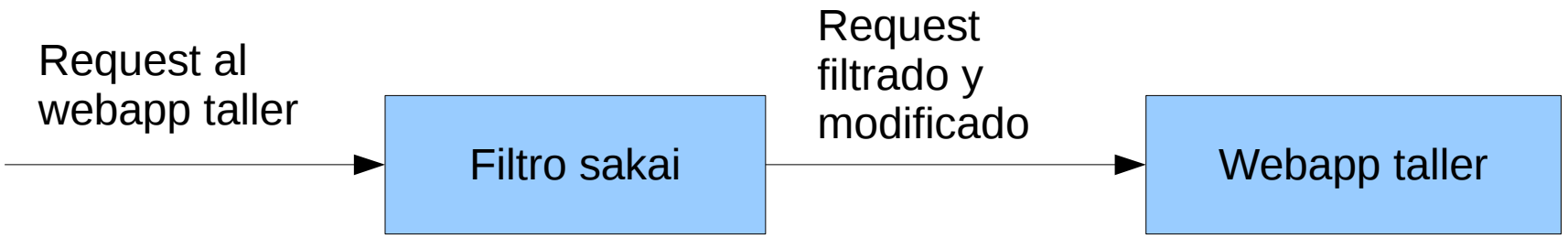
8) Modificar el fichero *web.xml*

- Añadir el filtro:

```
<filter>
   <filter-name>sakai.request</filter-name>
   <filter-class>org.sakaiproject.util.RequestFilter</filter-class>
</filter>
<filter-mapping>
   <filter-name>sakai.request</filter-name>
   <servlet-name>sakai.taller</servlet-name>
   <dispatcher>REQUEST</dispatcher>
   <dispatcher>FORWARD</dispatcher>
   <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
```

Construcción de una herramienta

Los “*requests*” serán procesados primero por sakai haciendo posible la integración de una webapp como parte de la plataforma de Sakai.



Construcción de una herramienta

- Añadir el *Listener* de registro de aplicaciones

```
<listener>
   <listener-class>org.sakaiproject.util.ToolListener</listener-class>
</listener>
```

El *ToolListener* es el encargado de registrar las aplicaciones en Sakai buscando los ficheros *xml* que están dentro de la carpeta *tools* de *webapp*.

Construcción de una herramienta

9) Compilar y desplegar

- *mvn clean install sakai:deploy*

La aplicación ya esta lista para ser usada.

Construcción de una herramienta

- Para probar la aplicación podemos ejecutarla:
  - Entorno de Sakai (Administracion sites)
    - Crear un Site
    - Crear una nueva página
    - Insertar la herramienta
  - Entorno *mercury*
    - Agregar la opción *mercury.enabled* = *true* en *sakai.properties*
    - Ir a la Url http://sakaiserver:8080/mercury

Construcción de una herramienta

- Dos modos de utilizar los servicios de Sakai:
  - Utilizar el *ComponentManager*
  - Delegar la llamada a los servicios tipo *cover*

Construcción de una herramienta

- El *ComponentManager* utiliza *Spring* para gestionar la definición e implementación de Servicios (componentes)
- Para obtener un componente solo hace falta llamar a *ComponentManager.get (id-del-componente")*

Construcción de una herramienta

- La mejor alternativa es usar Servicios *cover*
- Los servicios estáticos que utilizan la implementación por defecto definida en Sakai
- Ex: *SiteService.getSite(“my-site-id”);*

Construcción de una herramienta

Para usar los servicios del ejemplo necesitamos:

1) Añadir los módulos en los que se encuentran definidos como dependencias de nuestra aplicación (taller-app/pom.xml).

```
<dependency>
  <groupId>org.sakaiproject</groupId>
  <artifactId>sakai-tool-api</artifactId>
  <version>${sakai.version}</version>
</dependency>
```

Construcción de una herramienta

```
 <dependency>
  <groupId>org.sakaiproject</groupId>
  <artifactId>sakai-site-api</artifactId>
  <version>${sakai.version}</version>
</dependency>
<dependency>
  <groupId>org.sakaiproject</groupId>
  <artifactId>sakai-authz-api</artifactId>
  <version>${sakai.version}</version>
</dependency>
<dependency>
  <groupId>org.sakaiproject</groupId>
  <artifactId>sakai-entity-api</artifactId>
  <version>${sakai.version}</version>
</dependency>
 <dependency>
  <groupId>org.sakaiproject</groupId>
  <artifactId>sakai-util-api</artifactId>
  <version>${sakai.version}</version>
</dependency>
```

Construcción de una herramienta

2) Importar los paquetes *cover* de cada uno de los Servicios y los elementos de las APIs implicadas

```
import org.sakaiproject.site.api.Site;
import org.sakaiproject.site.cover.SiteService;
import org.sakaiproject.tool.cover.ToolManager;
import org.sakaiproject.tool.api.Placement;
import org.sakaiproject.authz.api.Member;
```

Construcción de una herramienta

3) Utilizarlos mediante los métodos estáticos de los servicios *cover*:

```
Placement currentPlacement = ToolManager.getCurrentPlacement();
Site currentSite = SiteService.getSite(currentPlacement.getContext());
Set <Member> memberSet = currentSite.getMembers();
```

Construcción de una herramienta

- Para la creación de servicios necesitamos:
  - API del servicio
  - Implementación de esta
  - Definición del componente para que Sakai lo registre

Construcción de una herramienta

Los pasos a seguir para la creación del servicio:

1) Crear el directorio para albergar la api (*taller-api*)

2) Definir el *pom.xml* (Descripción del módulo de API)

Construcción de una herramienta

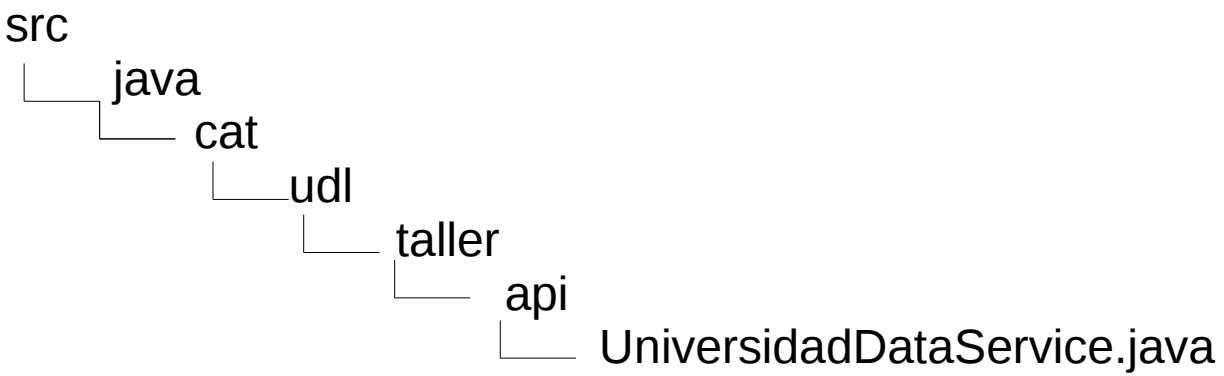
```
<?xml version="1.0"?>
<project xmlns="http://maven.apache.org/POM/4.0.0">

  <modelVersion>4.0.0</modelVersion>

  <parent>
    <artifactId>taller-sakai-tool</artifactId>
    <groupId>cat.udl.taller</groupId>
    <version>2.5.0</version>
      <relativePath>../pom.xml</relativePath>
  </parent>

  <name>taller-api</name>
  <artifactId>taller-api</artifactId>
  <groupId>cat.udl.taller</groupId>

  <organization>
    <name>Universitat de Lleida</name>
    <url>http://www.udl.cat</url>
  </organization>
    <inceptionYear>2008</inceptionYear>
```

Construcción de una herramienta

```
<packaging>jar</packaging>
 <properties>
  <deploy.target>shared</deploy.target>
</properties>


      <dependencies/>

<build>
     <resources/>
</build>

</project>
```

Construcción de una herramienta

3) Crear la estructura de la API y el servicio (interficie java) :



El servicio proporcionará 2 métodos:

```
public String getUrl();
public String getNombre();
```

Construcción de una herramienta

4) Modificar el fichero *pom.xml* del proyecto taller para que incluya el módulo taller API

<module>taller-api</module>

Construcción de una herramienta

Para la creación de la implementación y definición del componente:

1) Crear el directorio *taller-impl*

2) Crear el directorio para la implementación (*taller-impl/impl*)

3) Crear el directorio para la descripción del componente (*taller-impl/pack*)

4) Crear en cada directorio los descriptores de los módulos.

Construcción de una herramienta

- *impl/pom.xml*

```
<?xml version="1.0"?>
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <artifactId>taller-sakai-tool</artifactId>
    <groupId>cat.udl.taller</groupId>
    <version>2.5.0</version>
    <relativePath>../../pom.xml</relativePath>
  </parent>
  <name>taller-impl</name>
  <groupId>cat.udl.taller</groupId>
  <artifactId>taller-impl</artifactId>
  <organization>
    <name>Universitat de Lleida</name>
    <url>http://www.udl.cat/</url>
  </organization>
  <inceptionYear>2008</inceptionYear>
```

Construcción de una herramienta

```
<packaging>jar</packaging>
  <properties>
    <deploy.target/>
  </properties>
  <dependencies>
    <dependency>
      <groupId>cat.udl.taller</groupId>
      <artifactId>taller-api</artifactId>
      <version>${sakai.version}</version>
    </dependency>
    <dependency>
 <dependency>
      <groupId>org.sakaiproject</groupId>
      <artifactId>sakai-component</artifactId>
      <version>${sakai.version}</version>
    </dependency>
    <dependency>
      <groupId>org.sakaiproject</groupId>
      <artifactId>sakai-component-api</artifactId>
      <version>${sakai.version}</version>
    </dependency>
  </dependencies>
  <build/>
</project>
```

Construcción de una herramienta

- *pack/pom.xml*

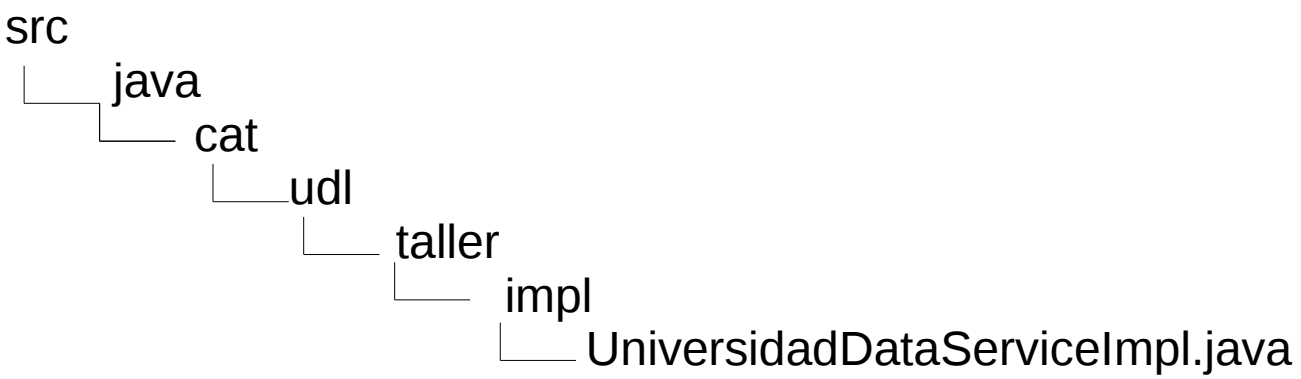
```
<?xml version="1.0"?>
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <artifactId>taller-sakai-tool</artifactId>
    <groupId>cat.udl.taller</groupId>
    <version>2.5.0</version>
    <relativePath>../../pom.xml</relativePath>
  </parent>
  <name>taller-pack</name>
  <groupId>org.sakaiproject</groupId>
  <artifactId>taller-pack</artifactId>
  <organization>
    <name>Unversitat de Lleida</name>
    <url>http://www.udl.cat/</url>
  </organization>
  <inceptionYear>2008</inceptionYear>
```

Construcción de una herramienta

```
 <packaging>sakai-component</packaging>
  <properties>
    <deploy.target>components</deploy.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>cat.udl.taller</groupId>
      <artifactId>taller-impl</artifactId>
      <version>${sakai.version}</version>
    </dependency>
  </dependencies>
  <build>
    <resources/>
  </build>
</project>
```

Construcción de una herramienta

5) En el directorio *impl* creamos la estructura y ficheros de implementación



Construcción de una herramienta

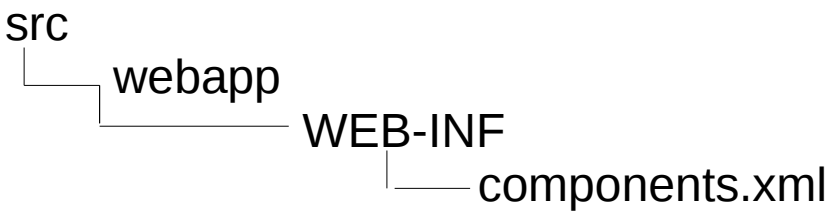
- La clase *UnivesidadDataServiceImpl* implementará los dos métodos definidos por la interficie más dos métodos “setter” para dos propiedades. De estas dos propiedades obtendrán el valor de *url* y nombre.
- Los métodos “setter” los utilizaremos posteriormente para inyectarle valor con el *ComponentManager*.

Construcción de una herramienta

```
String institucionUrl = null;
String institucionNombre = null;

/*Setter para la inversrion de control */

public void setInstitucionNombre (String nombre){
	institucionNombre = nombre;
}
public void setInstitucionUrl (String url) {
	institucionUrl = url;
}


/*Getters para implementar los metodos de la interficie*/

public String getUrl(){
	return institucionUrl;
}
public String getNombre (){
	return institucionNombre;
}
```

Construcción de una herramienta

- 6) En el directorio *pack* creamos la estructura necesaria y el fichero *components.xml*



  - Es el descriptor de los componentes que implementamos.
  - Utiliza el formato xml de *Spring* para definir *Beans*

Construcción de una herramienta

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
        "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
<bean id="cat.udl.taller.api.UniversidadDataService" class="cat.udl.taller.impl.UniversidadDataServiceImpl" >
        <property name="institucionNombre"><value>Universitat de Lleida</value></property>
        <property name="institucionUrl"><value>http://www.udl.cat</value></property>
</bean>
</beans>
```

- La estructura se desplegará en el directorio *components* de Tomcat
- Sakai registrará el componente con el id indicado y pasará a estar disponible a través del *ComponentManager*

Construcción de una herramienta

7) Modificar el fichero *pom.xml* del proyecto taller para que incluya el módulo *taller-impl/impl* y *taller-impl/pack*

8) Compilar el proyecto con *mvn clean install sakai:deploy*

9) Rebotar Tomcat

El servicio ya está disponible para las herramientas.

Construcción de una herramienta

Para probar el servicio seguiremos los siguientes pasos:

1) Modificaremos *taller-app/pom.xml* para incluir las dependencias (*ComponentManager* y API de taller)

```
<dependency>
     <groupId>org.sakaiproject</groupId>
     <artifactId>sakai-component</artifactId>
     <version>${sakai.version}</version>
   </dependency>

 <dependency>
     <groupId>cat.udl.taller</groupId>
     <artifactId>taller-api</artifactId>
       <scope>provided</scope>
     <version>${sakai.version}</version>
   </dependency>
```

Construcción de una herramienta

2) Incluir en la aplicación el *ComponentManger* y la interficie del servicio

```
import org.sakaiproject.component.cover.ComponentManager;
import cat.udl.taller.api.UniversidadDataService;
```

3) Solicitar al *ComponentManager* el componente

```
UniversidadDataService uds = (UniversidadDataService)
ComponentManager.get("cat.udl.taller.api.UniversidadDataService");
```

4) Ejecutar los métodos del servicio

```
uds.getUrl()
```

Construcción de una herramienta

5) Compilar y ejecutar la prueba

Construcción de una herramienta

- Para hacer un buen servicio es recomendable proporcionar un servicio tipo *cover*.
- Las herramientas podrán utilizarlo sin necesidad de recurrir a los mecanismos que proporciona el *ComponentManager*

Construcción de una herramienta

Para crearlo tenemos que:

1) Añadir la dependencia a taller-api/pom.xml

```
<dependency>
    <groupId>org.sakaiproject</groupId>
    <artifactId>sakai-component</artifactId>
    <version>${sakai.version}</version>
  </dependency>
```

2) Crear una clase con los métodos estáticos correspondientes para obtener la información deseada.

Construcción de una herramienta

3) Proporcionar a la clase un método *getInstance* para que los métodos mencionados puedan cargar la implementación con *ComponentManager* y ejecutar el mismo método en la implementación.

Construcción de una herramienta

```
public static cat.udl.taller.api.UniversidadDataService getInstance(){
        return (cat.udl.taller.api.UniversidadDataService)
            ComponentManager.get(cat.udl.taller.api.UniversidadDataService.class);
}

public static String getNombre(){
        cat.udl.taller.api.UniversidadDataService cs = getInstance();
        return cs.getNombre();
}

public static String  getUrl(){
        cat.udl.taller.api.UniversidadDataService cs = getInstance();
        return cs.getUrl();
}
```

Construcción de una herramienta

4) Utilizar en la aplicación el servicio tipo *cover* de forma estática

Ej: *UniversidadDataService.getUrl()*

Construcción de una herramienta

- Tecnologías más usadas para la construcción de aplicaciones web:
  - *JSF*
  - *MyFaces*
  - *RSF*
  - *Portlets*
  - *Velocity*
  - ...

Servicios web en Sakai

Servicios Web:

- Sakai ofrece una serie de recursos que permite crear fácilmente servicios web *(WS)* que interactuen con la *API* de Sakai
- Basados en *JWS*

Servicios web en Sakai

Pasos a seguir:

1) Activar la propiedad *webservices.allowlogin=true* en *sakai.properties*

2) Crear un fixero jws dentro de *sakai-src-2.5.0/webservices/axis/src/webapp*

   Ejemplo *MyService.jws*

Servicios web en Sakai

3) Importar todos los servicios *cover* i *API* que necesitéis:

4) Crear un método *establisSession* para poderlo llamar en cada método del *jws*

```
private Session establishSession(String id) throws AxisFault {
        Session s = SessionManager.getSession(id);

        if (s == null)
        {
                throw new AxisFault("Session "+id+" is not active");
        }
        s.setActive();
        SessionManager.setCurrentSession(s);
        return s;
    }
```

Servicios web en Sakai

5) Crear los métodos necesarios y empezar siempre estableciendo la sesión.

```
public String getNumeroUsuarios(String sessionid){
        Session session = establishSession(sessionid);
        return "" + UserDirectoryService.countUsers();
    }
```

6) Compilar el proyecto webservice con el comando *mvn clean install sakai:deploy*

7) Crear un *script* en *python, perl*, ... para probar el *ws.*

Servicios web en Sakai

- El script siempre ha de ejecutar el *SakaiLogin.jws* antes de empezar

```
import os
import sys
from SOAPpy import WSDL

username = "admin"
password = "admin"

server_url = "http://localhost:8080"
login_url = server_url + "/sakai-axis/SakaiLogin.jws?wsdl"
script_url = server_url + "/sakai-axis/MyService.jws?wsdl"

login_proxy = WSDL.SOAPProxy(login_url)
script_proxy = WSDL.SOAPProxy(script_url)

loginsoap = WSDL.SOAPProxy(login_url)
sessionid = loginsoap.login(username, password)

scriptsoap = WSDL.SOAPProxy(script_url)

print "El numero de usuarios es " +scriptsoap.getNumeroUsuarios(sessionid);
```

Herramientas remotas

- Ventajas
  - Interoperabilidad
  - Escalabilidad

Herramientas remotas

- Herramientas remotas:
  - LinkTool
  - IMS TI
  - Projecte Campus Tool

Herramientas remotas

LinkTool

- Nos permite ejecutar aplicaciones remotas
- Estas pueden estar escritas en en otro lenguaje
- Utiliza WS para la comunicación con la plataforma.
- Fácil de utilizar
- Solo para sakai

Herramientas remotas

IMS TI

- Existen dos herramientas:
  - Sakai IMS Tool Interoperability Tool (Cumple 100% de la especificación IMS TI 1.0)
  - Sakai IMS Tool Interoperability Portlet (no cumple el 100% de la especificación IMS TI) Existen mejoras para probar funcionalidades deseadas en IMS TI 2.0

- Ejecutan herramientas que cumplen IMS TI.

Herramientas remotas

Proyecto Campus

- Ejecuta herramientas remotas basada en el estándar *OSID* de *OKI*.
- Existen todo un conjunto de herramientas para esta plataforma.
- Las herramientas se pueden usar tanto en Moodle como en Sakai.
- La implementación de las herramientas se debe hacer en Java o php.
- Aún está en fase de test

Herramientas remotas

- Futuro
  - IMS TI 2.0 + Proyecto Campus

Preguntas