FACULTAD DE
INFORMÁTICA

UNIVERSIDAD DE
MURCIA

Universidad de MURCIA
FACULTAD DE INFORMÁTICA

SINCRONIZACIÓN DE SITIOS EN SAKAI

Grado en Ingeniería en Informática
Trabajo Fin de Grado

Autor:
Juan Arcadio Martínez Cárceles

Dirigido por:
Ginés García Mateos
Juan José Meroño Sánchez

Murcia, septiembre de 2011

Agradecimientos

El resultado de este trabajo no hubiera sido posible sin la ayuda prestada por muchas personas. Empezando por mis directores de proyecto, Juan José Meroño, es mi *Sakai fellow*, todos los días me enseña algo nuevo, Ginés García Mateos, por su gran capacidad para motivarme y por guiarme en esta travesía. La gente de Ática, Juanmi Bernal y Javi Quirante que me ayudaron a resolver algunos problemas de JSF y JPA. Eduardo Rey y Jesús Méndez expertos CSS y JavaScript. El excelente equipo de Campus Virtual apoyándome siempre, incluso el día que iban a invadir Polonia. José Rabal que me apadrinó al llegar allí y me enseñó muchísimo, es el que más me ha tenido que aguantar. José Mariano Luján, *the master of permissions*. Sabina, Lorena, Raúl, Juan Rodríguez, Manolo y Ramón, entre todos generan un ambiente ideal para abordar proyectos como el desarrollado. Jesús Pérez, que alguna noche me ha dejado las llaves para que cierre. Daniel Merino por sus sugerencias. Gabriel Sánchez, que me ha contagiado el perfeccionismo. Juan de la Cruz Maiquez, tan brillante como siempre aportándome algo de luz. También he contado con la ayuda de Antonio Javier Martínez, Sergio Murcia y José Quiroga. La música, la cual he abandonado temporalmente, pero ella nunca me ha abandonado. Y finalmente, mis amigos y mi familia, por entenderme en todo momento. Sin todos ellos no hubiese sido posible este proyecto, mi más sincero agradecimiento.

Índice general

Índice de figuras

Resumen

Este proyecto es un proyecto de desarrollo basado en software libre sobre la plataforma Sakai.

Sakai es un Sistema de Gestión del Aprendizaje (también conocido por las siglas LMS, *Learning Management System*), un sistema diseñado para facilitar las tareas relacionadas con el aprendizaje. El trabajo ha sido realizado sobre Sakai CLE (*Collaboration and Learning Environment*), concretamente con la versión 2.7.1 de Sakai CLE.

En Sakai la información está accesible desde los sitios, todas las páginas que se consultan desde la plataforma son sitios. Los sitios pueden tener distinto fin, como por ejemplo para una asignatura o para un proyecto colaborativo.

Los sitios tienen herramientas, cada herramienta define un conjunto de permisos que pueden servir para acceder a cierta información, o realizar ciertas acciones.

Un concepto clave en Sakai son los *realms*, estamos interesados concretamente en los *realms* que están asociados a un sitio. Los usuarios que acceden a un sitio lo hacen con un rol, los roles están definidos en el *realm*. Cada rol tiene asociado un conjunto de permisos que otorgan ciertos privilegios sobre las herramientas a los usuarios que tenga dicho rol en un sitio.

Al crear un sitio en Sakai también se crea el *realm* asociado a ese sitio. La creación de este *realm* de sitio se hace copiando desde un *realm* plantilla para cada tipo de sitio, de modo que cada sitio tiene su propio *realm* y es independiente de los demás.

Hay situaciones en las que es deseable modificar un permiso a todos los sitios de un tipo. Las soluciones que tiene un administrador de Sakai para realizar los cambios sobre los *realms* de sitios son:

- Modificar los *realms* de todos los sitios uno a uno; es inviable cuando tenemos una gran cantidad de sitios.
- Ejecutar las modificaciones sobre la base de datos; es peligroso porque puede dejar la base de datos inconsistente.
- Usar el *realm* especial *!role.sitehelper;* se hace una disyunción lógica de los permisos con los de este *realm* especial, no permite distinguir para qué tipo de sitios.

Con el propósito de mejorar estas soluciones y crear un método mucho más eficiente de gestión de los permisos, en este proyecto vamos a abarcar el

problema de la sincronización de sitios detectada en la administración de la plataforma Sakai. Para el desarrollo se han utilizado las siguientes tecnologías de código abierto:

- Apache Maven.
- Apache Tomcat.
- Hibernate.
- JavaEE 5 – JPA.
- Java Server Faces.
- Mockito.
- Quartz.
- Spring Framework.
- Subversion.

Como primera solución al problema se desarrolló un trabajo de Quartz. Se trata de un planificador para procesos Java similar al comando *cron* de Unix.

Este primer desarrollo cubre la carencia detectada en la administración de sitios en Sakai, pero la solución no es ágil. Se desarrolla una interfaz web que permite actualizar fácilmente las tareas de sincronización. Para este propósito necesitaremos almacenar en una base de datos la información de las tareas a realizar. Se incluye un permiso en la herramienta de modo que solo sea accesible por los administradores. Además se desarrollan pruebas unitarias para comprobar el correcto funcionamiento de la herramienta. Por último se escribe una ayuda de la herramienta integrada dentro de la ayuda de Sakai explicando el funcionamiento de la herramienta.

En la versión 2.7 apareció el concepto de estructura Indie, que hace la herramienta independiente del núcleo de Sakai. Por otra parte, nuestra herramienta también estaba ligada a la base de datos de la Universidad de Murcia. Realizamos un proceso de adaptación para hacerla independiente, de manera que pueda ser usada por cualquier institución que use Sakai.

Hemos usado un complemento de Apache Maven para gestionar los lanzamientos de nuevas versiones o *releases*, este complemento genera los *tags* en el servidor de Subversion y modifica todos los ficheros con las versiones del proyecto en desarrollo.

Como conclusión del proyecto, hemos publicado tanto el código como los compilados en el repositorio oficial de Sakai, dentro de un espacio que dispone la Universidad de Murcia. Además se ha creado una página web[1] en la que se explica cómo instalar la herramienta desarrollada, así como su funcionamiento y configuración. Nos hemos puesto en contacto con los responsables del proyecto para mover esta página a *confluence* (wikipedia de los desarrolladores

[1] Página web: https://aulavirtual.um.es/access/content/user/juanarcadio%40um.es/umusync/index.html

de Sakai), y para enviar un anuncio a la comunidad para que los usuarios de Sakai estén al corriente de la existencia de dicha herramienta, puedan probarla y enviarnos las sugerencias que crean oportunas.

Extended Abstract

This project is a free software development project based on Sakai e-learning platform.

Sakai is a Learning Management System (LMS), a system designed to facilitate works related to teaching and learning.

The project has been done on Sakai CLE (Collaboration and Learning Environment). Nowadays, Sakai CLE is used for different purposes: learning management, research projects, collaborative projects, or as an ePortfolio, in which students can publish their work, teachers have more flexibility to guide students, and organizations can assess their student's learning process. In particular we have used the 2.7.1 version of Sakai CLE.

There are other Sakai products, Sakai OAE (Open Academic Environment) and Sakai hybrid (Sakai hybrid integrates a portion of Sakai CLE into Sakai OAE), but these products are not in an stable version yet, on the 1st of September 2011, the latest version of Sakai OAE is 1 RC6, acronym *RC* means release candidate, and it is not an stable release.

In Sakai, information is available from sites, all pages that are visited on the platform are sites. Sites can have different purposes, for example we could have course sites or collaborative sites.

Each site have a set of tools, the most common tools that can be found within an LMS platform are: resources, announcements, assignments, chats, forums, wiki, exams and grade books.

In Sakai, each tool defines a set of permissions, those permissions allow users to have access to certain information or perform some actions.

A key concept in Sakai is the term *realm*. For this project we are especially interested in the *realms* that are connected to a site. When a user access a site, the user has a role, roles are defined in the *realm*. The relationship between one user and one role within one site can be direct or through a provider. Each role has an associated set of permissions that grants certain privileges to users.

When you create a site in Sakai, the system creates the *realm* associated with that site. That new *realm* is created by coping from the specific *realm* template depending on the site's type. There is a realm template for each type of site. Therefore, each site has its own *realm* and it is independent from others.

When adding a new tool to Sakai you might need to set a new permission, also,

if the administrator has to modify a set of permissions for an specific role in a concrete type of site then, the admin would need to modify the template for that type of site, so that new sites created copy the right set of permissions. Those changes made to the realm template will not affect all sites that were created before introducing these new changes to the template.

The possible solutions that a Sakai administrator has to change the *realms* of a set of existing sites are:

- To modify the *realms* of all the sites one by one using the *realm* admin tool; this is impractical when we have a great deal of sites.
- To modify the *realms* in the database, this is not recommended, it is too dangerous because it can bring inconsistencies to the database.
- To use the special *realm* called *!role.sitehelper*. In a site, when the system checks if a user has one permission, it firstly checks if the site *realm* for that user role has the permission and then it also checks if in that special *realm* mentioned the user role has the permission.

When institutions carry out new deployments of Sakai, it is very common to have to tweak the set of permissions a lot until they get the configuration they want. Other similar situation is when adding a new tool within Sakai platform. If you just add a new tool, then none of the sites created before the new permissions are set, in the right realm template, will have the permissions needed to use this tool.

Sakai platform currently lacks a management tool that is able to modify permissions or tools en masse, there are some solutions to satisfy those needs but they are not good enough. Therefore, the main goal of this project will be to develop a tool to fill this gap.

In the project we have used the following open source technologies:

- Apache Maven.
- Apache Tomcat.
- Hibernate.
- JavaEE 5 – JPA
- Java Server Faces.
- Mockito.
- Quartz.
- Spring Framework.
- Subversion.

As a first approach we developed a Quartz job. Among the Sakai management tools we can find a Java job scheduler based on Quartz. A Quartz job is like a schedule made with *cron*, the Unix command, also the temporal expressions are identical.

With Spring Framework, we can use an adapter from the Sakai code that allows us to schedule and execute the job in Java, which consists of a synchronization tasks list.

This first development provided us a utility that covers the lack identified in the management of Sakai sites, but it causes one inconvenience, the jobs defined in Quartz are static. When a Sakai administrator wants to sync sites, he/she probably wants to change only the sites of a particular type; from the Sakai user interface, the Quartz tool does not allow jobs to be parameterized. Therefore, there is no possibility to configure job properties from the Sakai user interface. To make changes in sync tasks, we should edit an XML file and place it in the Apache Tomcat server and then restart it. Restarting is necessary because the properties are set "on boot", when initializing the Spring context.

In the Quartz job that syncs sites, it is not easy to change the job configuration, so the next goal was to modify it including a web interface which allows us to setup synchronization jobs easily. For this purpose, it is necessary to store the information related to all tasks in database, so it is not necessary to reboot the server to adopt the changes; now synchronization jobs can be modified at any time. From this new tool it will be possible to run a synchronization task and the Quartz job is modified. When the job is triggered, it synchronizes all tasks stored in the database that are available at the moment. The tool includes one permission so that it is only accessible by administrators. The tool also includes unit tests to verify a proper operation of the tool. Finally I wrote some help pages that explain how to use this tool; this pages are integrated within the Sakai help tool.

This tool has been developed according to the classic structure of a Sakai tool. Sakai code from version 2.7 is divided into three parts: Kernel, Sakai (core) and Indie. Our tool has been developed similar to core tools from Sakai; this tool has a dependency with some components that we have included within the Kernel. In version 2.7, it appeared the new Indie structure, which makes the tool independent from the core of Sakai. In subsequent versions the developers of Sakai have been moving some tools to the new structure. Moreover, our tool was also connected to the University of Murcia database so we removed this link to make our tool independent and now it can be used by any institution using Sakai.

Our next goal was to adapt the tool to the Indie structure. The advantages of having an independent tool are that it can be easily installed by others. To install the tool developed, people only have to include a small module in Sakai and deploy it into the server, this module will download the compiled code of our tool, for this reason it is not necessary to compile all our code. Also, they can download the code, compile it, and use the module mentioned to deploy it on the server. Both, the code and the compiled version will be stored into Sakai's official repository, under an available space for the University of Murcia

so that it can be downloaded by anyone.

Another advantage of making this tool independent is that it will be easier to handle when migrating a new Sakai version. To deploy the tool in another version of Sakai you just have to modify some variables in our project; these variables indicate the versions of the components we use from Sakai.

For the future maintenance of the project, the development will be done in the main branch of subversion (trunk) and by using merge operations we could move the changes to the branch of this project. We have used an Apache Maven plug-in to manage releases of new versions. This plug-in accesses subversion repository, adds tags, and modifies all files with the versions of the project development.

Related to the future maintenance of the tool, we have also included two profiles that allow us to compile for the version of Sakai you are using. One profile is for 2.7.1 version and one for 2.8.0 version. Probably, in September 2011, versions 2.7.2 and 2.8.1 will appear; we could easily create a new profile for each version. A migration to a new version of Sakai will not involve any modification of code in our tool.

To conclude this project, I have created a web page hosted on *aulavirtual.um.es*, the Sakai platform implanted at University of Murcia. I have made public the webcontent that is hosted in my workspace. This page explains how to install the tool, how the tool works and how to set up; I have contacted with the Sakai webmaster to move this page to *confluence* (the Sakai wiki) and to send an announcement to the community to notify the existence of this new tool.

Capítulo 1

Introducción y referencias históricas

1.1. Introducción a Sakai

Sakai es un Sistema de Gestión del Aprendizaje (también conocido por las siglas LMS - *Learning Management System*), un sistema diseñado para facilitar las tareas relacionadas con el aprendizaje, también permite desarrollar cursos a distancia. Universidades y otras instituciones de educación superior están poniendo cada vez más interés en estos sistemas.

Las herramientas más habituales que se pueden encontrar en un LMS son: recursos, anuncios, tareas, chat, foros, wiki, exámenes y calificaciones.

La plataforma Sakai tiene su origen en Estados Unidos, en el año 2004 se lanzó la versión 1.0, las universidades precursoras fueron las de Míchigan, Indiana, Standford, Massachusetts y Virginia, apoyadas por el consorcio uPortal y la iniciativa de conocimiento abierto (*OKI*). Posteriormente se creó la fundación Sakai para gestionar este proyecto, durante el último año se ha hecho gran hincapié en la internacionalización de la plataforma lo cual está fomentando su globalización, ya es usada por más de 350 instituciones, por lo que habrá que tener en cuenta este aspecto cuando queramos aportar el trabajo a la comunidad.

El proyecto SAKAI es de código abierto, está basado principalmente en *Java*, aunque engloba otras tecnologías como por ejemplo: *Maven, Hibernate, Spring, JSF, RSF, Velocity, JSP*, y otras.

Desde la fundación Sakai[2] hablan de tres productos:

- Sakai CLE (Collaboration and Learning Environment): El que se usa actualmente con distintos fines, para gestión del aprendizaje, para proyectos de investigación, para proyectos colaborativos, o como ePortfolio, en el que el alumnado puede presentar su trabajo, el profesorado tiene mayor flexibilidad para guiar al alumno, y las organizaciones pueden evaluar el aprendizaje de los alumnos.

[2] Conferencia de Ian Dolphin, director ejecutivo de la fundación Sakai, en la Universidad Politécnica de Valencia, Marzo de 2011: https://polimedia.upv.es/visor/?id=bfb42b31-80c6-444e-a3c2-3cdba31d6952

- Sakai OAE (Open Academic Environment): Es otro concepto, redes sociales en un entorno educativo, te conecta con personas que puedan resolver tus dudas, emplea sistemas de enseñanza abierta, permite adaptar el entorno a lo que necesites, este producto está aún en desarrollo y no se utiliza en ninguna institución.
- Sakai Hibrido: Es una combinación de ambos, como la versión OAE aún no dispone de herramientas, mientras se desarrollan las nuevas herramientas para esta versión, la opción híbrida es reutilizar algunas funcionalidades de la versión CLE dentro de la OAE.

1.2. Organización del código de Sakai

La última versión de Sakai CLE en el momento de comenzar este proyecto es la versión 2.7.1, que fue lanzada el 25 de agosto de 2010.

El código de Sakai se divide en tres partes:

- **Kernel**: Donde se encuentran los servicios básicos de la plataforma, como pueden ser la gestión de usuarios, de permisos, de recursos, etc.
- **Sakai**: Incluye las herramientas *core*, las que componen la distribución oficial de Sakai.
- **Indie**: Aparece en la versión 2.7.0, aquí encontramos herramientas como las que hay en la parte de Sakai, la ventaja de los "*Indie Projects*" respecto a las herramientas de Sakai es que no interfieren al lanzamiento de nuevas versiones de Sakai, y también al contrario, Sakai no interfiere en el lanzamiento de nuevas versiones de estas herramientas, que pueden lanzar nuevas versiones antes de que Sakai cambie de versión, la tendencia es incorporar más herramientas desde el *core* de Sakai a *Indie*.

Además de la distribución oficial de Sakai, algunos miembros de la comunidad donan sus proyectos como contribuciones, para albergar todas estas aportaciones hay un espacio denominado ***contrib***[3] en el repositorio oficial.

1.3. Conceptos sobre Sakai

Algunos conceptos importantes en el entorno Sakai son los siguientes.

1.3.1. Sitios

Un sitio es un punto de reunión de participantes, un espacio de trabajo que tiene un contenido relacionado con él. Puede ser público, de modo que

[3] Ruta de acceso al espacio contrib: https://source.sakaiproject.org/contrib/

cualquier usuario no autenticado acceda a él, o privado, en este caso existe un moderador del sitio. En Sakai toda la información que se muestra está relacionada con un sitio, por ejemplo, la página inicial que aparece antes de autenticarte en Sakai es un sitio.

Existen distintos tipos de sitios, los principales tipos de sitio que hay por defecto son:

- ***My workspace***: espacio personal asociado a cada usuario.
- ***Course***: espacios asociados a asignaturas.
- ***Project***: espacios colaborativos.

También existen otros tipos de sitio, como por ejemplo: ***gateway*** para la página de inicio, sitios ***porfolio*** para uso de herramientas de ese tipo o ***help*** que contiene la ayuda.

Además se pueden definir tipos de sitio personalizados, indicando que herramientas estarán disponibles para cada tipo de sitio.
Los sitios se pueden dividir en **grupos**, de modo que las personas que pertenecen a un sitio además pueden pertenecer a uno, a varios, o a ningún grupo.

Los sitios tienen un ***site id***, es una cadena que representa al sitio, es un identificador único.

Un sitio puede tener asociadas una serie de propiedades tipo diccionario, en formato texto, como por ejemplo, a que curso académico pertenece el sitio, o quien es el sitio padre.

Se puede establecer jerarquía entre sitios, esta relación es usada por temas de accesibilidad, desde un sitio acceder a cualquiera de sus hijos, o desde un hijo, acceder con un simple clic al sitio padre. Herramientas como por ejemplo el calendario también usa esta propiedad para poder mostrar en el calendario del sitio padre los eventos de los calendarios de los sitios hijos.

1.3.2. Herramientas

Una herramienta proporciona una funcionalidad que es usada desde un sitio a través de una interfaz web.

Una herramienta define una serie de **permisos**, de modo que los usuarios que tengan cierto permiso pueden acceder a cierta información, o realizar una acción concreta.

Relacionado con los permisos se puede hablar de herramientas que son "***section aware***", esto significa que la herramienta se puede comportar de distinta forma

para cada grupo del sitio, por ejemplo, en un herramienta foro mostrando información distinta, los miembros del grupo A no pueden ver los hilos que han publicado miembros del grupo B.

La herramienta tiene un título que desde la versión 2.7 está internacionalizado, de modo que en las versiones anteriores al cambiar las preferencias del idioma seguías viendo el título de la herramienta en el idioma en el que se creó.

Las herramientas además tienen una descripción, una breve definición de su utilidad, este texto está internacionalizado, y está junto con el título de la herramienta en el siguiente fichero de configuración:

```
{Sakai}/config/localization/bundles/src/bundle/org/sakaiproject/localization/bundle/tool
/tools.properties
```

En el directorio donde está el fichero de configuración hay distintas versiones, cada versión está en un idioma o idioma/país, como por ejemplo los ficheros `tools_pt_BR.properties` y `tools_pt_PT.properties`, los dos son en portugués para el primero para Brasil y el segundo para Portugal.

Las herramientas tienen un identificador único ***tool id***, como por ejemplo: sakai.chat, sakai.announcements, sakai.discussion, etc.

1.3.3. Páginas

Las herramientas están alojadas dentro de páginas, la mayoría de las páginas contienen una única herramienta, pero hay otros casos en los que una página alberga más de una.

La página además de tener herramientas tiene un título, en caso de tener una única herramienta Sakai mostrará como título de la página el título de dicha herramienta, en este caso estará internacionalizado, la plataforma modifica el título de la página según las preferencias idiomáticas de usuario, sin embargo, cuando la página tiene más de una herramienta, el usuario que crea dicha página le debe de poner un título y siempre mostrará el mismo título independientemente de las preferencias de idioma que tengan los usuarios.

Desde un sitio se puede acceder a las páginas usando el menú que aparece en la parte izquierda. Allí se identifica cada página por el título, y al poner el ratón sobre la página, muestra en un *tooltip* la descripción que tiene la herramienta.

Para páginas multiherramienta, es interesante conocer que las páginas tiene una configuración de disposición que puede ser *layout* en una columna o en dos columnas, y las herramientas dentro de las páginas tienen una posición indicada por coordenadas, siendo la coordenada (0,0) la herramienta que aparecerá arriba a la izquierda.

El caso más habitual de una página con varias herramientas es la página de inicio. La página de inicio tiene la propiedad *IS_HOME* con valor *true*, lo que hace que ignore el título de la página, y le ponga el título de Inicio, internacionalizado.

Las herramientas que hay presentes en la página de inicio para cada tipo de sitio se definen en unas propiedades dentro del fichero de configuración ***sakai.properties***. De modo que al añadir o quitar herramientas de un sitio, la página de inicio se actualizará según los valores que indique dichas propiedades de configuración.

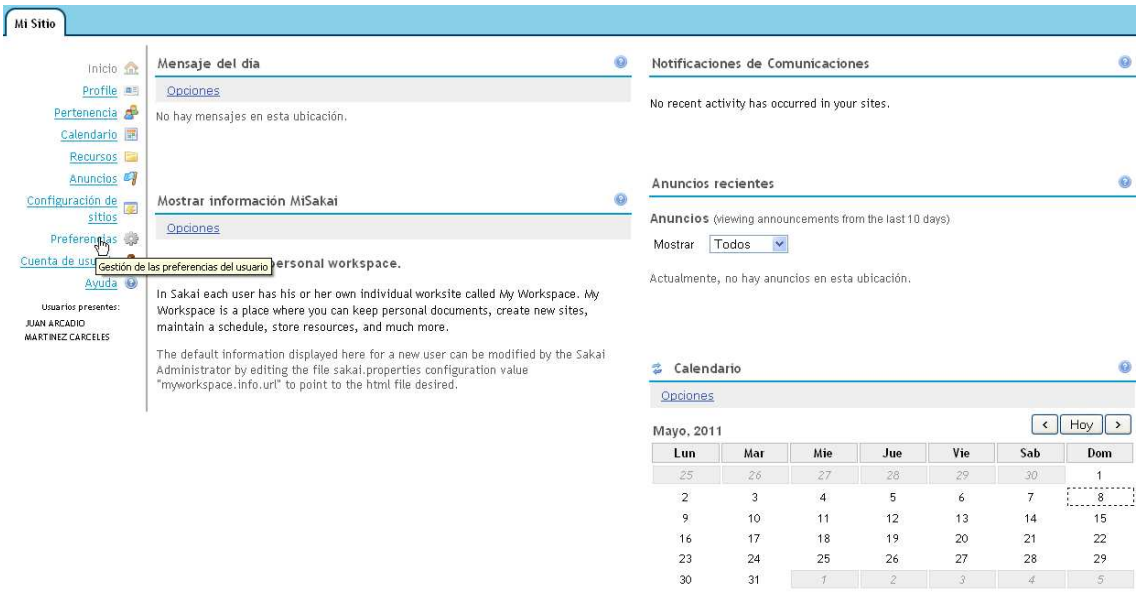
Las páginas dentro del sitio también tienen un orden, en el siguiente fichero de configuración del kernel:

```
{kernel}/component-manager/src/main/bundle/org/sakaiproject/config/toolOrder.xml
```

En este fichero se indica el orden de las páginas para cada tipo de sitio según las herramientas que contengan.

Al acceder a un sitio por primera vez, se muestra la primera página que según la configuración del orden de herramientas va en primer lugar, generalmente suele ser la página de inicio.

La imagen 1-1 muestra cómo es la interfaz de Sakai, se puede observar en el menú de la izquierda las páginas presentes dentro del sitio (*Inicio, Profile, Pertenencia, etc.*), al poner el ratón sobre una de ellas aparece la descripción de la herramienta, la página que está seleccionada es la de Inicio, se muestra a la derecha, se puede observar que contiene varias herramientas, con una distribución de dos columnas, cada herramienta tiene su título como se puede observar (*Mensaje del día, Notificación de Comunicaciones*, etc.).



1-1 Captura de pantalla: Páginas y Herramientas dentro de un Sitio

1.3.4. Realms

Los *realms*, llamados perfiles en la traducción de la herramienta *sakai.realm*, es el elemento central del sistema de seguridad de Sakai.

Un *realm* tiene una agrupación de **roles**, y una lista de **usuarios**, a cada usuario le asigna un rol y un estado (activo, inactivo). Los usuarios pueden ser añadidos al *realm* de forma manual, o a través de un proveedor de usuarios.

Uno de los roles presentes en el *realm* debe ser de **mantenimiento**, lo que le dará permisos especiales sobre la entidad que esté asociada al *realm*.

Un *realm* puede estar asociado a distintos tipos de entidades:

- A un **sitio**, todos los sitios tienen un *realm* asociado, dicho *realm* se identifica con: */site/site_id*
- A un **grupo**, todos los grupos de un sitio tienen un *realm* asociado, dicho *realm* se identifica con: */site/site_id/group/group_id*
- A un **recurso**, desde la herramienta de recursos de Sakai podemos indicar que un recurso esté accesible durante un periodo de tiempo, o solo para un grupo de usuarios, esto genera un *realm* asociado al recurso almacenado en Sakai, estos *realms* se identifican con: */content/xxx*
- A un **tipo de usuario**, los usuarios de Sakai se le puede asignar un tipo, cada tipo de usuario tiene un *realm*, no existe un *realm* para cada usuario concreto, si no para todos los del tipo, este *realm* se identifica con: *!user.template.tipousuario*, en caso de no existir dicho *realm* plantilla para el tipo de un usuario, o en caso de que el usuario no tenga tipo, se le asociará el *realm*: *!user.template*

- A un **tipo de sitio**, existe un *realm* plantilla asociado a un tipo de sitio, de modo que aparecen los roles presentes en los sitios de ese tipo, este *realm* se identifica con: *!site.template.tipositio*, en caso de no existir dicho *realm* plantilla para el tipo de sitio, utilizará el *realm*: *!site.template*
- A **grupos de un tipo de sitio**, del mismo modo que pasa con los sitios, existe un *realm* plantilla asociado a los grupos de un tipo de sitio, este *realm* se identifica con: *!group.template.tipositio*. Si no existe dicho *realm* plantilla para el tipo de sitio, utilizará un *realm* genérico para grupos de sitio identificado como: *!group.template*

Los *realms* plantilla que hemos visto, son especiales, se distinguen de los demás *realms* porque todos empiezan con el símbolo admiración, y tienen la particularidad de que no tienen usuarios definidos, únicamente definen roles.

El servicio del kernel de Sakai que se encarga de la gestión de los *realms* se llama **AuthGroupService**.

1.3.5. Roles

Como hemos visto los usuarios que pertenecen a un sitio tienen asociado un rol, ese rol tiene un nombre que lo identifica **role id** y un listado con todos los permisos de Sakai. Cada permiso tiene dos posibles valores, activo o inactivo.

Los roles que hay por defecto en Sakai son:

- **.auth/.anon**: Diferencia entre usuarios autenticados y usuarios que no han iniciado sesión, el rol *.auth* actúa como rol de mantenimiento, estos roles se utilizan en *realms* de sitios como en *gateway* (la página inicial), en el sitio de ayuda, o en las plantillas para los *realms* de tipo de usuario.
- **access/maintain**: El rol *maintain* es el rol de mantenimiento, es el administrador del sitio, mientras que *access* es un rol de acceso al sitio, que pueden ver el sitio pero no pueden configurarlo, añadir herramientas, participantes, etc. Estos roles se utilizan en los sitios colaborativos de tipo *project*, o en los sitios personales de usuario.
- **Instructor/Student/Teaching Assistant**: Son los roles para sitios docentes de tipo *course*. El rol de mantenimiento en estos sitios es el rol *Instructor*.

Se pueden personalizar los *realms*, creando nuevos roles o modificando los existentes.

1.3.6. Permisos

Un permiso sirve para permitir realizar cierta acción o poder visualizar cierta información, por ejemplo, una herramienta puede definir un permiso *tool.view*

que sirva para ser visible a los usuarios que tienen un rol con dicho permiso activo dentro del sitio donde está la herramienta (en *realm* asociado al sitio), y no mostrarse a los usuarios con un rol que no lo tiene activo.

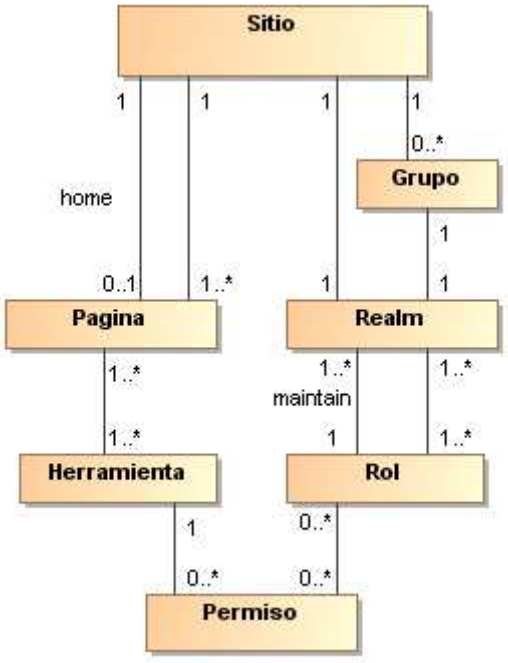
En una herramienta generalmente el sistema comprobará los permisos en el *realm* del sitio.

Si la herramienta es "*section aware*" se comprobarán los permisos en la unión del *realm* de sitio con el *realm* de grupo.

En herramientas que usan recursos, la herramienta de recursos y otras que la utilizan, como por ejemplo para añadir un anexo a un mensaje, se comprobará en primer lugar los permisos que se tiene en el *realm* del sitio, los permisos de *realm* de grupo, y por último los permisos que se tiene en el recurso sobre el que se quiere realizar cierta acción.

Cada herramienta define sus propios permisos, cuando se despliega una nueva herramienta en Sakai, automáticamente en todos los roles aparecerán los permisos que lleve asociados dicha herramienta (los nuevos permisos aparecen con valor inactivo).

En la imagen 1-2 podemos observar el modelo conceptual del sistema de permisos que utiliza Sakai, es un resumen de lo expuesto hasta el momento.



1-2 Modelo conceptual: sistema de seguridad de Sakai

1.3.7. Creación de sitios

El permiso para poder crear sitios en Sakai es el permiso *site.add*, este es un permiso especial, no se comprueba en ningún *realm* asociado a un sitio, se comprueba de forma dinámica en el *realm* plantilla para el tipo de usuario, los *realms* del tipo *!user.template.tipo* son los que definen que usuarios pueden crear un sitio, también se comprueba en el *realm* de todos los usuarios *!user.template*.

Si tenemos el permiso adecuado y realizamos la acción de crear un nuevo sitio de tipo *project, course, portfolio, etc.*, en ese momento también se creará el *realm* del sitio, que será una copia del *realm* plantilla para ese tipo de sitio en ese momento, de modo que si en el futuro el administrador de Sakai decide cambiar algún permiso en la plantilla para ese tipo de sitio, el sitio creado mantendrá su plantilla estática, no se actualizará.

Los sitios de tipo *my workspace*, se crean al conectarse el usuario por primera vez a la plataforma, o si es la primera vez que accede el usuario tras la eliminación de su sitio personal.

Existe un *realm* especial con identificador: ***!site.helper***, en este *realm* se pueden hacer cambios temporales que afectan a todos los roles en todos los *realms*, no hace ninguna distinción, un rol ajustado de esa manera comprobará si el permiso está presente en el *realm* del sitio o en el *realm !site.helper*.

Capítulo 2

Análisis de objetivos y metodología

Con lo que hemos visto en la introducción, hay un claro problema con los *realms* de los sitios. Cuando se crean se hace una copia de la plantilla del momento de la creación y no se actualizan con nuevos ajustes que se pueden hacer, estos *realms* se quedan anticuados.

En las nuevas implantaciones de Sakai es muy habitual tener que retocar las plantillas de permisos. Otras ocasiones en las que es necesario retocar las plantillas de permisos es al añadir una nueva herramienta a Sakai, nadie tiene los permisos en dicha herramienta.

Las soluciones que hay ahora mismo no son suficientemente buenas, cuando hay una gran proliferación de sitios, tener que modificar manualmente los *realms* de los sitios es inviable.

Usando el *realm !site.helper*, no se hace ninguna distinción, los permisos se añaden a todos los roles, por ejemplo si tenemos que distinguir entre dos tipos de sitios para cursos, unos presenciales y otros *online*, y queremos que una nueva herramienta con la que los alumnos hacen videollamada a los profesores solo esté disponible para los del curso *online*, con la solución de usar *!site.helper*, al añadir el permiso se lo dará a todo el que tenga rol alumno, sin importar el sitio en el que esté.

Además, usando el *realm !site.helper* nunca se puede eliminar un permiso, siempre añadir.

Otra solución habitual es ejecutar scripts de SQL directamente sobre la base de datos de Sakai para modificar los permisos, pero el proceso de realizar modificaciones sobre la base de datos es un trabajo delicado.

2.1. Objetivos del proyecto

El objetivo fundamental de este proyecto fin de grado es facilitar el engorroso proceso para el administrador de Sakai, tal y como hemos analizado previamente.

El permiso **site.upd** además de otras funcionalidades, proporciona la posibilidad de añadir o eliminar herramientas sobre el sitio que se tenga dicho permiso, por lo que los usuarios no siempre tendrán la posibilidad de modificar las herramientas del sitio, y en esos casos es el administrador el que añade o elimina las herramientas.

Si se decide añadir una nueva herramienta a Sakai puede ser interesante añadirla automáticamente a ciertos sitios de un determinado tipo, o que cumplan unas características. Dado que vamos a sincronizar los permisos de un sitio, vemos adecuado añadir la opción de sincronizar herramientas a este trabajo.

Por lo tanto, el objetivo principal del proyecto es desarrollar la funcionalidad de sincronización de sitios de forma masiva para Sakai. El trabajo desarrollado deberá tener las siguientes características:

1. Utilizar una estructura similar a las herramientas de Sakai.
2. Estar integrado como trabajo de Quartz dentro de la plataforma Sakai.
3. Sincronizar tanto permisos como herramientas.

Una vez obtenida la funcionalidad básica del proyecto, se establecen los siguientes objetivos secundarios:

1. Proveer de una interfaz web a la herramienta.
2. Garantizar el correcto funcionamiento de la herramienta.
3. Hacer la herramienta independiente (tipo *Indie).*
4. Permitir que se pueda instalar sin mucho esfuerzo descargando los artefactos ya compilados.
5. Administrar versiones finales del proyecto.
6. Adaptar la herramienta a distintas versiones de Sakai.
7. Internacionalizar la herramienta.
8. Aportar a la comunidad Sakai nuestro trabajo.

2.2. Metodología de trabajo y herramientas

En cuanto a la metodología de trabajo, podemos decir que he aplicado un desarrollo ágil de software. Me he ido marcando pequeños objetivos, haciendo primero un análisis y un diseño, a continuación el desarrollo correspondiente al objetivo marcado, y finalmente pruebas para comprobar que todo funciona correctamente.

La primera tarea fue de investigación, sobre funcionamiento de Sakai y de cómo utilizar su API, concretamente analizando el código del servicio *SiteService* que es el que se utiliza para la gestión de sitios.

Las herramientas utilizadas para poder llevar a cabo este proyecto son las siguientes:

- **Eclipse**: Es un entorno de desarrollo de código abierto, en él hemos desarrollado todo el proyecto. Sobre este entorno hemos utilizado los siguientes complementos:

    - **JEE**: Incluye las funcionalidades para trabajar con Java dentro de un servidor de aplicaciones.
    - **Subclipse**: Complemento para integrar un cliente de Subversion dentro de Eclipse.
    - **m2clipse:** Complemento para integrar Maven 2 dentro de Eclipse.
    - **JBoss Tools RichFaces:** Complemento para facilitar la edición de ficheros de JSF con la librería RichFaces.

- **Apache Maven:** Hemos utilizado Maven para la gestión y construcción del proyecto. Además nos sirve para ejecutar las pruebas unitarias sobre nuestro código. Sobre esta herramienta hemos utilizado los siguientes complementos:

    - **Javadoc:** Para generar documentación en formato HTML a partir de código fuente Java.
    - **Sakai**: Para desplegar los artefactos en el servidor Apache Tomcat.
    - **Release**: Lo hemos utilizado junto a un cliente de Subversion para cerrar versiones de nuestra herramienta.
    - **Surefire/Cobertura**: Para generar informes con los resultados de la ejecución de las pruebas unitarias.

Capítulo 3

UmuSync como trabajo de Quartz

Como solución al problema planteado vamos a desarrollar un trabajo que actualice las herramientas y permisos de los sitios ya creados. Aplicaremos un filtro para que sincronice solo los sitios que cumplan ciertas condiciones.

Este trabajo será generalmente lanzado bajo de demanda, al definirlo como trabajo se puede diferir su ejecución y planificar su ejecución en horas en las que haya menos carga en el servidor.

3.1. Quartz Enterprise Scheduler Job

Dentro de Sakai podemos encontrar un planificador de trabajos para Java. Este planificador de trabajos es Quartz, un proyecto de código abierto que está integrado en una herramienta de administración de Sakai denominada "Planificador de trabajos basado en Quartz" (sakai.scheduler).

Desde esta herramienta tenemos una lista con todos los trabajos de Quartz definidos en Sakai, seleccionando uno de ellos podemos ejecutarlo en ese mismo momento, o planificar su ejecución. Para planificarlo debemos de introducir una expresión temporal cuya sintaxis es muy similar a la utilizada por el comando *cron* de Linux.

La expresión temporal está formada por 6 o 7 campos separados por espacios en blanco. Los campos son los siguientes:

Campo	Obligatorio	Valores permitidos	Caracteres especiales
Segundos	SI	0-59	, - * /
Minutos	SI	0-59	, - * /
Horas	SI	0-23	, - * /
Día del mes	SI	1-31	, - * ?/ L W
Mes	SI	1-12 o JAN-DEC	, - * /
Día de la semana	SI	1-7 o SUN-SAT	, - * ? / L #
Año	NO	Vacío, 1970-2099	, - * /

3-1 Tabla: campos de una expresión temporal de Quartz

Significado de los caracteres especiales:

- ***** (todos los valores): Para usar todos los valores permitidos.
- **?** (valor no especificado): Para indicar cualquier valor.
- **-** (rangos): Para indicar rangos de valores.
- **,** (valor múltiple): Para indicar varios valores.
- **/** (incrementos): Para indicar incrementos, por ejemplo 0/20 en el campo Minutos indica los valores 00, 20 y 40.
- **L** (último): Para indicar el último día del mes o la semana.
- **W** (día de la semana): Para indicar días entre semana, de lunes a viernes.
    - **LW**: Para indicar el último día entre semana del mes.
- **#** (cardinales): Para indicar cardinalidad, por ejemplo 2#1 indica el primer lunes del mes.

Un ejemplo de expresión temporal:

0 15 7 ? * MON-FRI Se ejecuta todos los días de lunes a viernes a las 7:15:00

En Sakai podemos encontrar un fichero de propiedades con la configuración de Quartz, una de estas propiedades por ejemplo indica si estamos en un cluster, este fichero de configuración está en la siguiente ruta:

```
{indie}/scheduler/scheduler-component-shared/src/java/org/sakaiproject/component/app/
scheduler/quartz.properties
```

3.2. Proyecto umujobs

En *umujobs* estarán declarados los trabajos de Quartz que desarrolle la Universidad de Murcia, estos trabajos podrán ser ejecutados desde la herramienta de planificación de trabajos en Sakai, además incluiremos el código que es común a todos los trabajos que hagamos.

Sakai dispone de un adaptador para ejecutar los trabajos, para esto al adaptador le indicamos un nombre de trabajo y el código que queremos ejecutar, Sakai se encargará de lanzarlo cuando sea requerido por disparadores de las planificaciones que hagamos, o por petición manual del usuario.

Para registrar un trabajo en Sakai debemos de definir un componente de *Spring* que sea de la clase *SpringJobBeanWrapper*, para el trabajo de sincronización de sitios el componente definido en el fichero *component.xml* dentro del proyecto *umujobs* es el siguiente:

```
<bean id="org.sakaiproject.api.app.scheduler.JobBeanWrapper.DoJobsSync"
      class="org.sakaiproject.component.app.scheduler.jobs.SpringJobBeanWrapper"
      singleton="true" init-method="init">

       <property name="beanId">
              <value>DoJobsSync</value>
       </property>
       <property name="jobName">
              <value>Sync Sites Job</value>
```

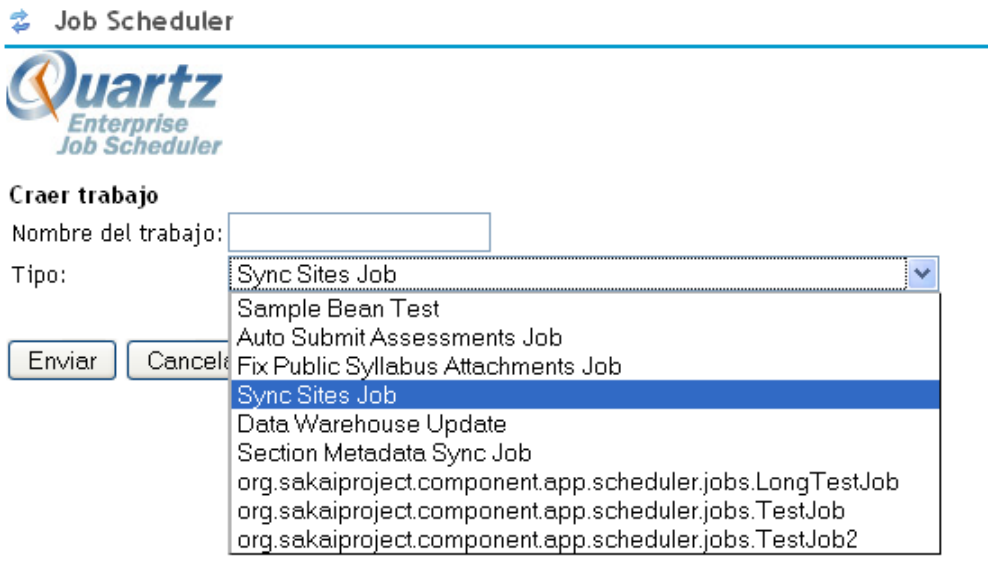
```
        </property>
        <property name="schedulerManager">
                <ref bean="org.sakaiproject.api.app.scheduler.SchedulerManager"/>
        </property>
</bean>
```

Donde las propiedades indicadas son:

- **beanId** : Identificador del componente que se invocará cuando se lance la ejecución del trabajo, debe de implementar la interface *org.quartz.Job.*
- **jobName** : Es el nombre que aparecerá en el listado de trabajos de la herramienta Quartz de Sakai.
- **schedulerManager** : El componente planificador de trabajos de Quartz.

Desplegando este fichero en el servidor, tras su arranque podremos observar que en la herramienta de planificación de trabajos basados en Quartz el trabajo que hemos declarado ya está disponible para planificar o lanzar su ejecución desde Sakai, aunque si se ejecuta, debido a que aún no hemos declarado el componente *DoJobsSync*, el resultado de la ejecución arrojará un error.

Podemos comprobar que se ha desplegado correctamente al acceder a la herramienta de planificación de trabajos basados en Quartz, que se muestra en la figura 3-2.



3-2 Captura de pantalla: trabajos disponibles en Quartz

El siguiente componente que debemos declarar es *DoJobsSync*, lo haremos en el mismo fichero components.xml:

```
<bean id="DoJobsSync" class="umu.sakai.umujobs.DoJobs">
        <property name="jobs" ref="umusync-jobs"/>
        <property name="authzGroupService">
                <ref bean="org.sakaiproject.authz.api.AuthzGroupService"/>
        </property>
```

```
        <property name="sessionManager">
                <ref bean="org.sakaiproject.tool.api.SessionManager"/>
        </property>
</bean>
```

El *bean* con identificador ***DoJobsSync***, es de la clase *DoJobs*, que como se dijo anteriormente debe implementar la interface *org.quartz.Job*, dicha interface tiene un método de ejecución que es el invocado por el planificador:

```
void execute(JobExecutionContext context) throws JobExecutionException
```

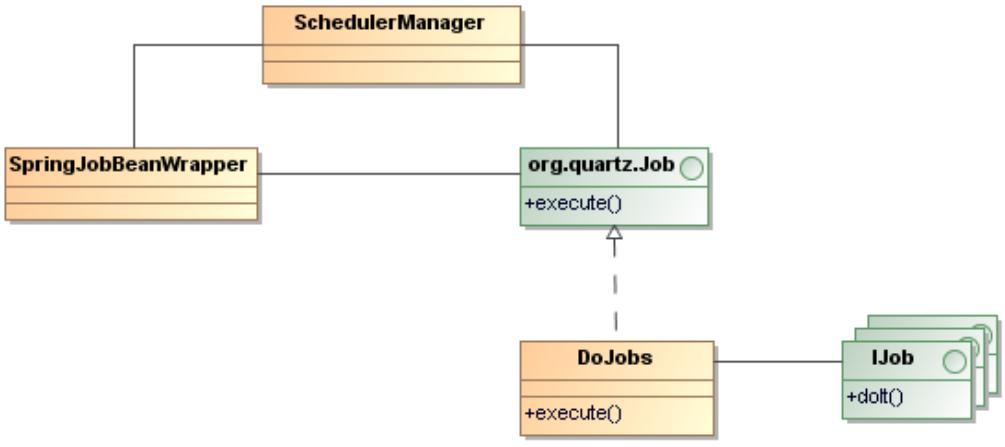
En este método hemos incluido las tareas comunes a todos los trabajos. En general todos los trabajos realizan tareas de administración, y para que su ejecución sea exitosa deben ejecutarse con una sesión de administrador, este método lo primero que hace es abrir la sesión del usuario administrador y al finalizar el proceso cierra dicha sesión.

Si un trabajo requiere ejecutarse con otro usuario se le puede indicar que usuario, estableciendo la propiedad ***user*** en este *bean*, por omisión toma el usuario *admin*. Para poder trabajar con sesiones necesita de los servicios de Sakai ***authzGroupService*** y ***sessionManager*** que son inyectados en la inicialización de la clase desde Spring.

La propiedad ***jobs*** indicada en el *bean* es una lista de objetos de la interface *IJob*, la interface se define en esta aplicación, y tiene un método de ejecución, de modo que el método de ejecución de *DoJobs* invoca al método de ejecución de todos los objetos de la lista.

Esto nos permite por un lado tener un trabajo como una lista de trabajos, un trabajo de Sakai es la invocación de uno o varios trabajos que hemos definido nosotros en *umujobs*, y por otra parte nos permite definir nuestros trabajos en otros proyectos, pero estos deben de tener una dependencia con *umujobs*.

En el diagrama 3-3 están las clases involucradas en este proceso.



3-3 Diagrama de clases: trabajos registrados e invocados por Quartz

3.3. Proyecto umusync

En *umusync* se definen todos los trabajos de sincronización que invocaremos desde el Quartz a través de *umujobs*.

A dichos trabajos le especificamos unos *realms* plantilla, una lista de herramientas a añadir, una lista de páginas a añadir, una lista de elementos que queremos eliminar (páginas y/o herramientas), además de unos criterios de selección.

Al ejecutar uno de los trabajos, primero obtiene todos los sitios creados, después comprueba cuales de ellos cumplen los criterios de selección, una vez que un sitio es seleccionado, se le modifican los permisos del *realm* asociado al sitio y a los grupos con las plantillas indicadas. Esta sincronización no elimina roles, si el sitio tiene un rol no presente en el *realm* plantilla, dicho rol se seguirá manteniendo en el sitio después de la ejecución del trabajo.

Cuando una herramienta de la lista a añadir no está presente en el sitio seleccionado se procede a crear una página con dicha herramienta, si la herramienta aparece dos veces en la lista del trabajo, se creará dos páginas con la misma herramienta, pero si estaba inicialmente en el sitio no se añade ninguna, por otra parte, si la herramienta no está permitida[4] para ese tipo de sitio, se ignora.

En el trabajo también se especifica las páginas con varias herramientas que se desean crear en el sitio, se comprueba que no estén creadas previamente, y en ese caso, se crea una nueva página con la información que se añade en su configuración, el título, su disposición (una o dos columnas), y las herramientas que estarán presentes.

En el sitio seleccionado, se le eliminarán todas las páginas que contengan solamente una herramienta y dicha herramientas sea de la lista a eliminar, o bien que la página tenga por título el especificado en la lista a eliminar.

En el fichero ***components.xml*** del proyecto *umusync* está especificada la lista de trabajos. Los elementos de esta lista deben ser objetos que implementen *IJob* para que puedan ser invocados desde *umujobs* cuando se dispare la ejecución:

```
<!-- Lista de IJobs que se van a ejecutar en esta tarea -->
<util:list id="umusync-jobs">
        <ref bean="umu.sakai.umusync.api.ISyncManager.asignaturaGrado"/>
        <ref bean="umu.sakai.umusync.api.ISyncManager.asignaturaPosgrado"/>
        <ref bean="umu.sakai.umusync.api.ISyncManager.titulacionGrado"/>
        <ref bean="umu.sakai.umusync.api.ISyncManager.myWorkSpaceOficial"/>
        <ref bean="umu.sakai.umusync.api.ISyncManager.myWorkSpaceExterno"/>
</util:list>
```

[4] En el fichero de configuración de cada herramienta se especifica en que sitios está permitida dicha herramienta.

La clase *SyncManager* es la que implementa la interface *IJob* para el trabajo de sincronización de sitios, es la que tiene toda la lógica de la ejecución del trabajo, las propiedades de cada sincronización están definidas en el mencionado fichero ***components.xml.*** A continuación se muestra un ejemplo de definición de este *IJob* incluyendo comentarios:

```
<bean id="umu.sakai.umusync.api.ISyncManager.coursePrimero2009" class="umu.sakai.umusync.impl.SyncManager">

        # Tipo de sitios a sincronizar, si no se indica ninguno sincronizará todos los sitios
        <property name="type" value="course"/>

        # Realm que copiaremos en el sitio
        <property name="authzBase" value="!site.template.course"/>

        # Realm que copiaremos en los grupos
        <property name="authzSection" value="!group.template.course"/>

        # Servicios[5] de Sakai necesarios para gestionar paginas y permisos
        <property name="siteService"><ref bean="org.sakaiproject.site.api.SiteService"/></property>
        <property name="authzGroupService"><ref bean="org.sakaiproject.authz.api.AuthzGroupService"/></property>

        # No tener en cuenta los sitios con los ids indicados
        <property name="ignoreById" value="idSitio1:idSitio2"/>

        # Propiedades que deben de cumplir los sitios (cualquier propiedad que definamos en el sitio)
        <property name="criteria">
            <map>
                # Solo tener en cuenta los sitios con curso académico indicado
                <entry key="term" value="2009"/>
                # Solo tener en cuenta los sitios con el curso dentro de la titulación indicado
                <entry key="curso" value="PRIMERO"/>
            </map>
        </property>

        # Lista de tools a añadir (en caso de que no estén, no se comprueban páginas con varias tools)
        <property name="addTool">
            <list>
                <value>Id de tool</value>
                <value>sakai.schedule</value>
            </list>
        </property>

        # Lista de tools/páginas a eliminar
        # Se eliminan las paginas con una tool, y el Id de la tool coincida con uno de la lista
        # Se eliminan las paginas con más de una tool, y el titulo de la pagina coincida con uno de la lista
        <property name="delTool">
           <list>
                <value>Titulo de pagina</value>
                <value>Id de tool</value>
           </list>
        </property>

        # Añadir varias herramientas dentro de una misma página
        <property name="multiTool">
                <map>
                        <entry key="TITULO DE LA PAGINA">
                        <list>
                                <value>ID DE TOOL 1</value>
                                <value>ID DE TOOL 2</value>
                                <value>ID DE TOOL 3</value>
                        </list>
                        </entry>
                </map>
        </property>

        # Si no se indica ningún layout toma por defecto single column
        <property name="columna" value="doble"/>
</bean>
```

[5] Sin una sesión de administrador no se podrán usar algunos métodos de los servicios de Sakai que requieren ciertos privilegios.

Para que un sitio cumpla las condiciones de selección debe de cumplir todos los criterios enumerados a continuación:

1. Es del **tipo**[6] correspondiente: Se comprueba que el sitio es del tipo especificado por el trabajo, si no se define esta propiedad en el trabajo indica que es para todo tipo de sitios.
2. El identificador del sitio no está en la lista ***ignoredById***: los sitios incluidos en esta propiedad son ignorados para ese trabajo.
3. El sitio no tiene la propiedad ***synchronized***, o la tiene y su valor no es ***off***. Una propiedad del sitio que la añade el administrador para que nunca sea modificado por ningún trabajo.
4. Se cumplen todas las propiedades de ***criteria***: Las propiedades definidas en esta variable tipo diccionario deben existir entre las propiedades asociadas al sitio, y deben de tener el mismo valor para que se proceda a ejecutar la sincronización.

En versiones anteriores de Sakai hubiéramos tenido serios problemas al establecer títulos en páginas y herramientas, ya que estos títulos no se internacionalizaban, para el caso de la ejecución de este trabajo en una versión anterior las páginas nuevas y herramientas que se crean estarían todas según las preferencias idiomáticas del administrador, y usuarios que tuviesen en sus preferencias de idiomas distintas al administrador tendrían dentro de un mismo sitio herramientas con el título en distintos idiomas, aunque el contenido de la herramienta sería en el idioma correcto.

[6] La propiedad tipo para los sitios de usuario por defecto (*myworkspace)* es *null*, podemos comprobar si es un sitio de usuario usando el servicio de Sakai *SiteService*, concretamente invocando al método *isUserSite*.

Capítulo 4

UmuSync como herramienta

Con el trabajo de Quartz realizado disponemos de una utilidad que cubre la carencia detectada en la administración de sitios en Sakai.

Tiene un inconveniente, ya que los trabajos de sincronización no son estáticos, no siempre queremos sincronizar los mismos sitios, y dado que los trabajos ejecutados en Sakai por *Quartz* no se pueden parametrizar, no hay ninguna opción de poder realizar la configuración de las propiedades de los trabajos desde la interfaz de Sakai.

Con el sistema actual, para realizar cambios en los trabajos que se ejecutan es necesario acceder al servidor, modificar el fichero y reiniciar el servidor. El reinicio es necesario porque las propiedades se establecen en el arranque, cuando se inicializa el contexto de *Spring*.

Por lo tanto, no es fácil realizar un cambio de configuración, y sería muy adecuado realizar una herramienta con una **interfaz web** para poder realizar más fácilmente la configuración de los trabajos de sincronización.

Para este propósito necesitaremos almacenar en **base de dato**s la información de las tareas a realizar, de modo que ya no sea necesario reiniciar para que el servidor reciba los cambios, ahora los trabajos de sincronización se podrán modificar en cualquier momento. Hasta el momento solo habíamos hecho uso de la base de datos a través de servicios de Sakai, ahora necesitaremos crear tablas para la herramienta.

El trabajo de Quartz se modificará y solo tendrá un trabajo en la lista, que será, el de realizar la sincronización según la configuración de la base de datos. Con el nuevo diseño el trabajo de Quartz recorre la lista de tareas almacenadas en base de datos y ejecuta aquellas están en estado activo.

Por otra parte, se ha detectado alguna carencia y se han añadido nuevas funcionalidades a las tareas de sincronización:

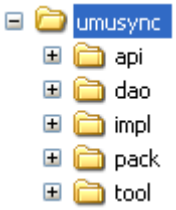
- **Tipos de sitio**: Se han incluido comprobaciones para los sitios de usuario que tiene Sakai por defecto, y para todos los tipos de sitio de usuario.

- **Configuración avanzada de filtros**: en el trabajo anterior solo se comprobaba que una propiedad tenía un valor, solo usábamos el operador de igualdad, se han incluido nuevos comparadores: distinto, existe, no existe, menor que, mayor que y coincidencia con expresión regular.
- **Sincronizar páginas**: Se añade la opción de poder comprobar si una herramienta existe en páginas multiherramienta.
- **Páginas de inicio**: Antes no se tenía en cuenta esta página especial.
- **Ignorar permisos**: Una nueva opción de añadir excepciones a la sincronización de permisos.
- **Elimina roles**: En anterior trabajo los roles que estaban en el sitio y no en la plantilla se mantenían en el sitio, ahora se sincroniza completamente eliminando aquellos roles que no aparezcan en *realm* plantilla.
- **Mejoras en el rendimiento**: Se comprueba que el sitio ha sido alterado para guardar sus cambios.

4.1. Estructura de herramienta en Sakai

En Sakai no existe una estructura predefinida para una herramienta, la organización de la misma se deja a criterio del programador. Dado que el código de Sakai es analizado y modificado por multitud de desarrolladores, es conveniente ajustar el código a una estructura familiar.

Aaron Zeckoski[7] y su equipo crearon en 2006 un complemento para el entorno de desarrollo *Eclipse*, este complemento crea una estructura de ficheros muy parecida a la de la imagen 4-1, que es similar a la que podemos encontrar en la mayoría de las herramientas de Sakai.



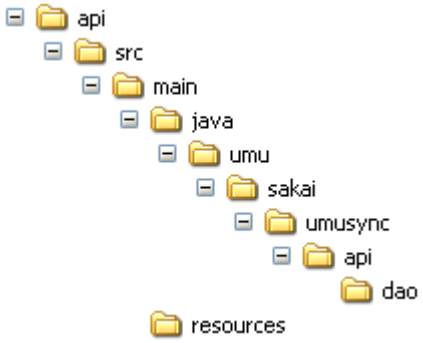
4-1 Estructura de herramienta

Con *Maven* también se puede definir el esqueleto básico de la herramienta, se puede crear un ***arquetipo***, y a partir de este arquetipo se pueden crear diferentes herramientas con la misma estructura.

[7] **Aaron Zeckoski** recibió en 2006-07 una distinción de la fundación, fue nombrado ***Sakai Fellow*** por sus aportes a la comunidad. ([2], capítulo 18).

Accediendo a cada módulo tenemos lo siguiente:

- **api**: contiene interfaces, modelos de objetos y ficheros hbm de *Hibernate*. Se despliega como ***jar*** en el directorio *shared/lib* del servidor *Tomcat*.

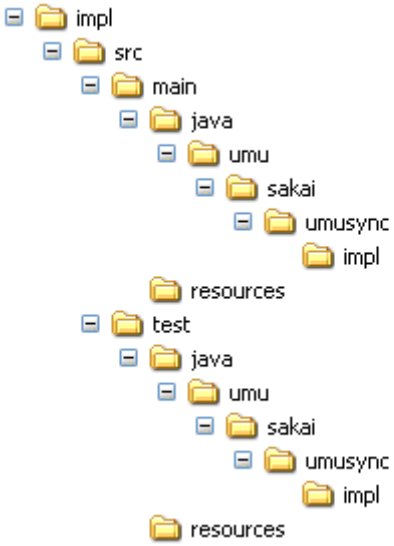


4-2 Estructura de herramienta: api

- **dao:** contiene la implementación de las clases *api/dao*, entidades, objetos que son almacenados en base de datos y consultas que se ejecutarán sobre estos objetos. Al igual que el módulo *api* este módulo también se despliega como ***jar*** en el directorio *shared/lib* del servidor *Tomcat*.

4-3 Estructura de herramienta: dao

- **impl:** contiene la implementación de las clases *api*, también podemos encontrar un directorio *test* donde realizar pruebas unitarias sobre las clases *Java* presentes en este módulo. Se despliega como **jar** dentro de la carpeta *\WEB-INF\lib* del componente que lo usa.



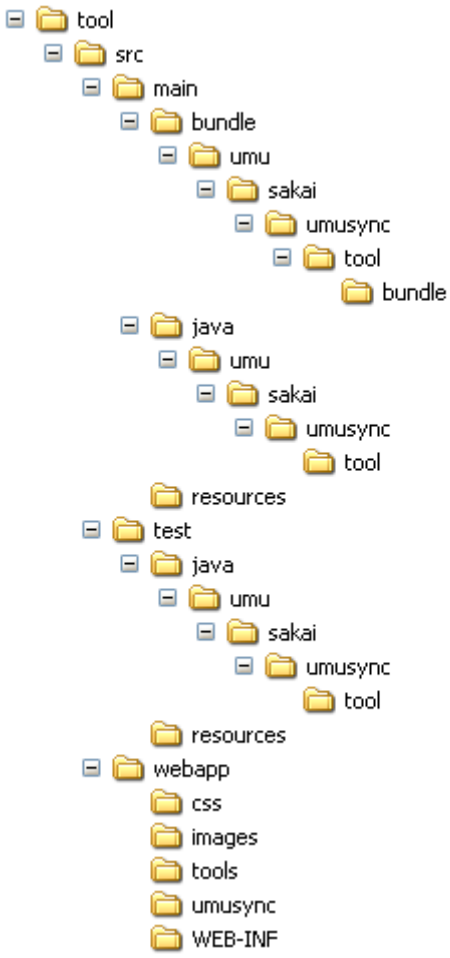
4-4 Estructura de herramienta: impl

- **pack:** contiene los componentes de *Spring*. Se despliega en el directorio *components* del servidor *Tomcat*.

4-5 Estructura de herramienta: pack

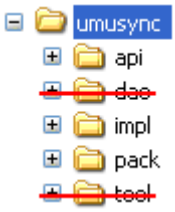
- **tool:** contiene la interfaz de usuario se despliega como ***war*** en el directorio *webapps* del servidor *Tomcat*. Dentro del módulo *tool* podemos encontrar:

    - **bundle:** contiene recursos de traducción que permite realizar una herramienta internacionalizada.
    - **java:** contiene código Java empleado por la herramienta.
    - **test:** es similar al que aparece en el módulo *impl,* es para comprobar el correcto funcionamiento de las clases Java de este módulo.
    - **webapp:** contiene la aplicación web.



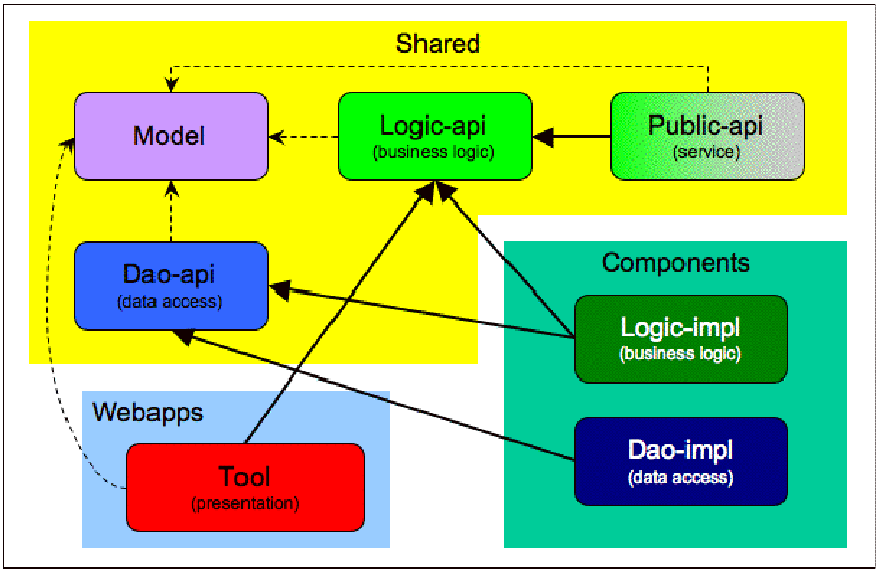
4-6 Estructura de herramienta: tool

En el primer desarrollo de umusync como trabajo de Quartz, no teníamos ni interfaz web, ni acceso a base de datos, teníamos la misma estructura pero mucho más simplificada, solo teníamos los módulos **api**, **impl** y **pack**.



4-7 Estructura como trabajo de Quartz

En la documentación de Sakai ([1], En la página *Sakai application (tool) structure*), indican que la estructura de la herramienta desplegada en el servidor debería de ser como se muestra en la siguiente imagen:



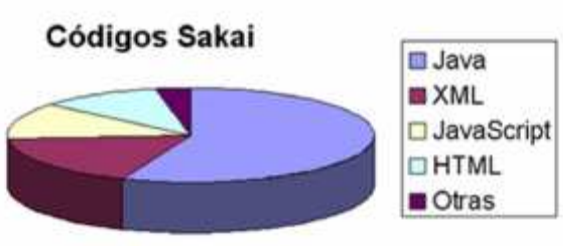
4-8 Estructura de herramienta: desplegada en el servidor

La diferencia del resultado que hemos obtenido nosotros con el que proponen desde Sakai, es que tenemos la implementación de DAO en el directorio *shared/lib* en vez de en el directorio *components*, esto es debido a que nuestra estructura de DAO tiene fusionado el modelo con su implementación.

En las herramientas de Sakai se usa *Hibernate* directamente, mientras que nosotros hemos añadido una capa de abstracción usando *JPA* sobre *Hibernate*, todo esto está detallado más adelante.

4.2. Marco tecnológico

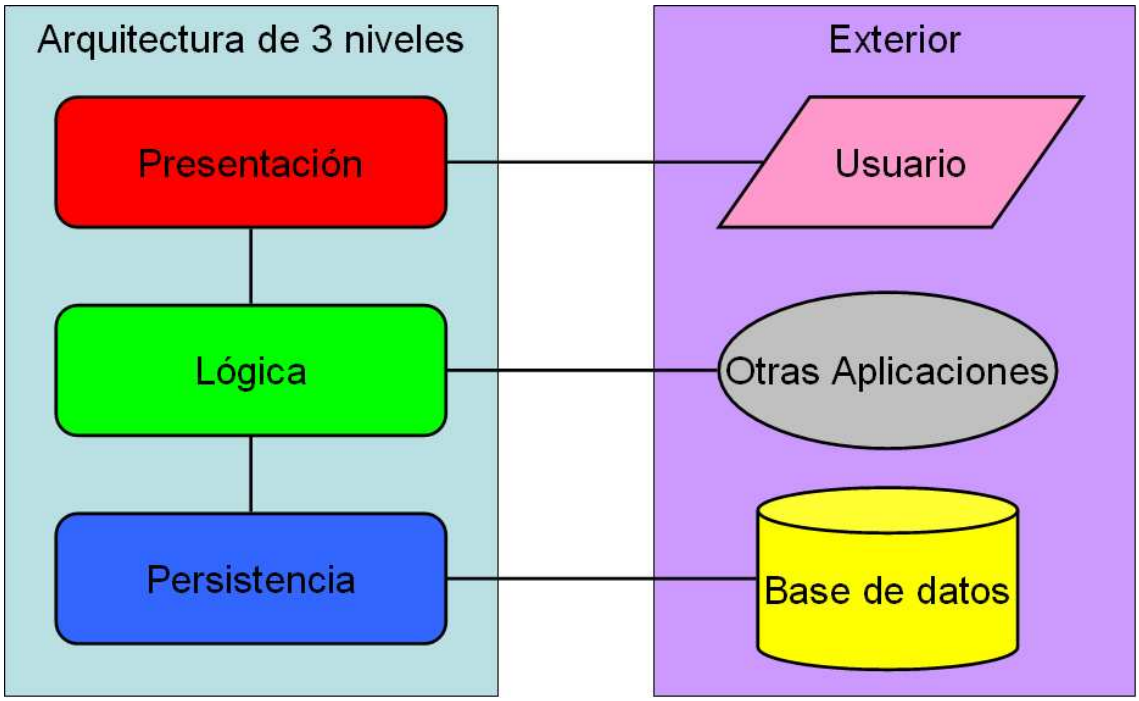
El lenguaje más utilizado en Sakai es Java, aunque existen otros lenguajes como XML, JavaScript y HTML, en el gráfico 4-9 se puede observar cual es el porcentaje de uso de cada lenguaje.



4-9 Gráfico: Lenguajes más utilizados en Sakai

A grandes rasgos podemos afirmar que Sakai utiliza una arquitectura de tres niveles, son los siguientes:

- El nivel de **presentación** es con la que interactúa el usuario, incluye toda la parte del diseño web.
- El nivel de **lógica** es donde se ejecutan las peticiones del usuario, ofrece una interface por la que recibe peticiones de la capa de presentación y le devuelve los resultados, para alguna de estas peticiones puede ser necesario acceder al nivel de persistencia. También puede recibir peticiones desde otras aplicaciones, como pueden ser servicios web, otras herramientas o trabajos de Quartz.
- El nivel de **persistencia** es desde donde se accede a la base de datos, aquí se gestiona el almacenamiento de los datos. Recibe peticiones de acceso a base de datos desde el nivel de lógica.



4-10 Arquitectura de 3 niveles

En los siguientes apartados vamos a realizar un análisis en profundidad los niveles de presentación y persistencia, y también como hace uso Sakai de *Spring* para la comunicación entre estos niveles.

En el nivel de lógica no es muy interesante en cuanto al marco tecnológico se refiere, utiliza *Java* para realizar sus propósitos. Por lo que no se ha dedicado un apartado a este nivel.

4.2.1. Nivel de presentación

En **Sakai** la capa de presentación es la más heterogénea de todas, se emplean

tecnologías muy diferentes como pueden ser *Java Servlets, Velocity Templates, JSF, RSF, Apache Wicket, Apache POI* o *Google Web Toolkit,* apoyadas por *JavaScript, AJAX o XML*.

David Roldán, Raúl E. Mengod y Daniel Merino hacen una comparación entre las tecnologías más usadas para la capa de presentación ([3], capítulo 6):

Tecnología	Facilidad de desarrollo	Facilidad de diseño	Integración con Sakai	Integración con AJAX/JavaScript
Java Servlets	Difícil.	Difícil.	No hay.	Media.
Velocity	Relativamente sencilla.	Difícil.	Buena.	Media.
JSP	Sencilla.	Difícil.	No hay.	Media.
JSF	Curva de aprendizaje elevada.	Difícil.	Muy buena.	Pobre.
RSF	Curva de aprendizaje moderada.	Excelente.	Media.	Buena.

4-11 Tabla: Tecnologías en capa de presentación

Para **nuestro desarrollo** hemos elegido *JSF*, que es la que mejor se integra con Sakai, es la más utilizada después de *Velocity Templates.*

Aunque califica esta tecnología de tener una curva de aprendizaje elevada, está bien documentada, lo que no es un gran problema, el principal problema del uso de esta tecnología, es que en Sakai usan una versión muy antigua, la versión 1.1.01, sobre ella tienen construida una librería de etiquetas para Sakai que es usada en todas las herramientas.

Podemos usar otras implementaciones como *myFaces*, que nos ha servido para incluir *richFaces*, esto nos ofrece nuevos componentes que mejoran el diseño y la integración con *AJAX* y *JavaScript*.

4.2.2. Nivel de persistencia

En Sakai hay dos formas de acceso a base de datos, a través de un componente de Sakai o mediante la generación de una capa DAO (*Data Acess Object*).

Los componentes de Sakai que dan acceso a base de datos están incluidos en el *kernel*, uno de estos componentes es *SqlService*, incluyendo este servicio a nuestra herramienta podemos ser capaces de acceder a base de datos. Esta opción es más habitual cuando se accede a tablas del funcionamiento básico de Sakai.

La opción de generar una capa *DAO* es más habitual cuando se accede a tablas particulares de una herramienta, y para ello en Sakai usan el *framework* de persistencia ***Hibernate***, que nos sirve para relacionar los objetos de *Java* con tablas en base de datos.

Hibernate hace el mapeo objeto-relacional usando unos ficheros *XML* en los que se especifica la tabla de base de datos que se mapea, las columnas de dicha tabla indicando con que atributos del objeto *Java* se corresponde cada una y las relaciones con otras tablas.

La comunicación con *Hibernate* se hace a través del objeto *Session*, con este objeto podemos abrir transacciones, almacenar objetos, hacer consultas, etc. además hace de caché de los objetos cargados, los objetos pueden ser de dos tipos:

- Tipo **persistent:** Es cuando ya está almacenado en base de datos.
- Tipo **transient:** Es cuando existe en memoria y no está en base de datos.

Además *Hibernate* ofrece su propio lenguaje para hacer consultas, **HQL** (*Hibernate Query Language).*

En nuestro desarrollo, sobre *Hibernate* hemos añadido otra capa de abstracción haciendo uso de ***JPA*** (*Java Persistence API*). De modo que no interactuamos directamente con *Hibernate*, el mapeo objeto-relacional (relaciones entre entidades Java y tablas de bases de datos, consultas, etc.), se realiza con anotaciones en la clase entidad, no es necesario usar ficheros descriptores *XML.*

JPA está incluido en *JavaEE 5* ([8], capítulo 24) nosotros usamos la versión 1.0, para usar la versión 2 de *JPA* es necesario utilizar una versión igual o superior a la versión 3.5 de *Hibernate*, pero en Sakai actualmente se usa la versión 3.2.7.

En *JPA* definimos la unidad de persistencia en un fichero XML en el que se indican diversos parámetros de configuración de base de datos, los objetos Java que utilizará para el mapeo objeto-relacional, y entre otras cosas, indicaremos el nombre de un fichero externo en el que escribiremos las consultas que vayamos a hacer sobre la base de datos. En este fichero distinguimos dos tipos de consultas:

- **Named Query**: Están escritas en *JPQL*, el lenguaje de consultas de *JPA*, este lenguaje tiene la ventaja de no depender de la base de datos que se utilice, la propia API hace las transformaciones necesarias cuando va a realizar las consultas.
- **Named Native Query**: Se ejecutan directamente sobre la base de datos.

Por cada unidad de persistencia *JPA* generará un contexto de persistencia. El contexto de persistencia es donde se manejan todas las entidades, nos garantiza que por cada identidad habrá una única instancia en el contexto, ya que un fila en base de datos solo será cargada una vez en un contexto. Podría considerarse como una memoria caché de la base de datos, acelerando el proceso de acceso a la base de datos, ya que minimiza el tráfico a la misma.

La clase **EntityManager** es la que tiene acceso a un contexto de persistencia, es

la que utiliza el programador para realizar las operaciones de base de datos.

Las operaciones básicas que la clase *EntityManager* puede hacer sobre la base de datos son las siguientes:

- **persist:** Insertar una nueva entidad.
- **find:** Realizar una consulta.
- **remove:** Eliminar una entidad.
- **merge:** Actualiza la información de base de datos con la de la entidad. En caso de no existir en base da datos hace lo mismo que *persist*.
- **refresh:** Actualiza la entidad con la información que hay en base de datos.
- **getReference:** Carga un objeto accediendo por su identificador.
- **clear:** Realizar un *rollback*.
- **flush:** Realizar un *commit*.

En JPA las entidades tienen cuatro estados, son los siguientes:

- **Transient:** Una entidad recién creada que no ha sido enlazada con el contexto de persistencia, solo existe en Java.
- **Persistent:** Una entidad sincronizada con el contexto de persistencia, puede llegar a este estado ejecutando una consulta, una inserción o una actualización con la clase *EntityManager.*
- **Detached:** Una entidad que sigue estando en Java después de finalizar una transacción, está desligada del contexto de persistencia, existe en Java y en la base de datos.
- **Removed:** Una entidad marcada para eliminar, existe en Java y se borrará de la base de datos al finalizar la transacción.

Las clases del módulo *dao* son clases Java en las que utilizamos anotaciones para definirlas como entidades, podemos usar tanto anotaciones de *Hibernate* como de *JPA*. Las principales son las siguientes:

Sobre la clase:

- **Entity**: Para indicar que se trata de una entidad.
- **Table**: Especifica el nombre de la tabla en la que se almacenarán los objetos de esa clase.

Sobre los atributos de la clase:

- **Id**: Sobre un atributo primitivo indica que es clave primaria.
- **SequenceGenerator/GeneratedValue**: Para definir una secuencia que genere automáticamente valores para el atributo.
- **EmbeddedId**: Sobre un atributo no-primitivo, indica que la clase representa una clave compuesta.
- **Column**: Especifica el nombre de la columna donde se almacenarán los datos del atributo sobre el que está la anotación, se le pueden especificar restricciones de integridad como atributos de esta anotación, algunos ejemplos pueden ser *nullable* y *length*, que también se pueden poner con

otras anotaciones como las que se indican a continuación.

- **NotNull:** Indica que el atributo no puede ser nulo.
- **Length:** Restricción de integridad especificando el tamaño que debe tener el valor de la columna, se puede especificar un máximo, un mínimo, un rango o un valor exacto.
- **OneToOne:** Relación uno a uno con otra entidad.
- **OneToMany/ManyToOne:** Relación uno a muchos.
- **ManyToMany:** Relación muchos a muchos.

Las anotaciones de relaciones tienen los siguientes atributos:

- Fetch: Indica la forma de cargar los objetos en las relaciones.
    - Lazy: Al cargar una entidad no carga las entidades relacionadas con ella hasta que no sean accedidas, en el momento de acceder a ellas es cuando se cargan en el contexto de persistencia.
    - Eager: Al cargar una entidad también se carga en el contexto de persistencia todas aquellas relacionadas con esta entidad, si las entidades relacionadas que estamos cargando también tienen relaciones de tipo *Eager*, también se cargarán todas las relacionadas con estas, por lo que hay que ser cautelosos con usar esta opción. En *JPA* es la opción por defecto.

- Cascade: Indica que operaciones se ejecutarán en cascada, puede tener los siguientes valores:
    - Persist: Propaga las operaciones de inserción a las entidades relacionadas.
    - Remove: Propaga las operaciones de eliminación a las entidades relacionadas.
    - Merge: Propaga las operaciones de actualización de base de datos a las entidades relacionadas.
    - Refresh: Propaga las operaciones de actualización de la entidad con el contenido de base de datos a las entidades relacionadas.
    - All: Propaga todas las operaciones anteriormente definidas.

Además en relaciones *OneToMany*, tenemos el atributo *mappedBy* para especificar el atributo de la otra entidad participante en la relación es la clave ajena. En el otro lado de la relación, aparecerá la anotación *ManyToOne* utilizando la anotación *JoinColumn* se indica la clave ajena.

En las relaciones *ManyToMany* podemos especificar una tabla intermedia para mapear esta relación, para ello se utiliza la anotación *JoinTable*.

Dentro de la anotación *JoinTable* se usa la anotación *JoinColumn* para indicar las columnas que forman la clave primaria de cada entidad participante en la relación, almacenándose como claves ajenas en la tabla intermedia que mapea la relación.

En el siguiente apartado veremos como combinando la tecnología *JPA* con *Spring* conseguimos grandes mejoras. Por ejemplo, la gestión de transacciones que en *Hibernate* se haría manualmente, utilizando *Spring* será mucho más sencillo para el programador.

4.2.3. Spring Framework

Spring Framework es usado en Sakai para la integración entre todas sus partes. Gestiona objetos como componentes, permitiendo crear objetos, configurar los objetos, enlazarlos con otros, controlar su ciclo de vida. Facilitan el uso de patrones de diseño.

En Sakai se crean dos tipos de contextos *Spring*, por un lado tenemos el que engloba lo desplegado en la carpeta ***components*** de *Apache Tomcat*, en la que están referenciados los niveles de lógica y persistencia, en este contexto se cargan todos los servicios que ofrece Sakai, en el servidor solo hay un contexto de este tipo.

El otro tipo de contexto engloba lo desplegado en la carpeta ***webapps*** de *Apache Tomcat*, todo lo referente al nivel de presentación, hay un contexto diferente por cada aplicación web, y no es accesible por otros contextos, aunque si se le pueden inyectar servicios del contexto de *components*.

La definición del contexto Spring para *components* está en los documentos *XML* del módulo *pack*, vamos a ver el caso concreto de los componentes que hemos definido en nuestra herramienta.

A cada componente generalmente se le pone el mismo nombre que la *API*. *Spring* usa *FactoryBean* como patrón factoría para crear componentes según la configuración indicada en el documento *XML*, nosotros lo declaramos como un objeto de la clase *ProxyFactoryBean* que implementa la citada interface, la definición es la siguiente:

```
<bean id="umu.sakai.umusync.api.ISyncManager"
      class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces" ref="umusync.proxyInterfaces"/>
    <property name="interceptorNames">
           <list>
                  <value>umu.sakai.umusync.secure.UMUSecureInterceptor</value>
           </list>
    </property>
    <property name="target">
           <ref bean="umu.sakai.umusync.api.ISyncManagerTarget"/>
   </property>
</bean>
```

En sus propiedades incluye una referencia a una lista de *interfaces*, que es la siguiente:

```
<!-- Interfaces -->
<util:list id="umusync.proxyInterfaces">
   <value>umu.sakai.umusync.api.ISyncManager</value>
```

```
        <value>umu.sakai.umujobs.api.IJob</value>
    </util:list>
```

Esta lista indica que la clase definida implementa esas interfaces y por lo tanto puede ser utilizada como objetos de ambos tipos. Por ejemplo para la ejecución desde *umujob* se tendrá en cuenta la interface *IJob*.

Otra propiedad es un interceptor de seguridad, que es el siguiente:

```
    <bean id="umu.sakai.umusync.secure.UMUSecureInterceptor"
          parent="umu.sakai.kernel.api.IUMUSecureInterceptor" init-method="init">
        <property name="toolPrefix" value="umusync."/>
        <property name="toolPermissions" ref="umusync.permissionList"/>
    </bean>
```

El interceptor de seguridad registra los permisos de la herramienta en Sakai, los permisos para nuestra herramienta son los siguientes:

```
    <!-- Permisos para esta tool -->
    <util:list id="umusync.permissionList">
              <value>VIEW</value>
        </util:list>
```

En el momento de iniciar el contexto *Spring*, cuando se crea este componente hace que en la lista de permisos de Sakai aparezca un nuevo permiso, el permiso *umusync.VIEW*. En la clase de la implementación *SyncManager*, podemos añadir anotaciones de seguridad indicando si es necesario algún permiso para ejecutar un método. Si un método público invocado a través de la API y dicho método tiene una anotación que requiere un permiso y no tenemos dicho permiso el interceptor lanzará una excepción y no nos permitirá ejecutar dicho método.

Por último en la propiedad *target* indica que este componente está referenciando a otro, que es el siguiente:

```
  <bean id="umu.sakai.umusync.api.ISyncManagerTarget"
        class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
    <property name="transactionManager" ref="umusync-transactionManager"/>
    <property name="target">
        <bean class="umu.sakai.umusync.impl.SyncManager">
            <property name="siteService">
                <ref bean="org.sakaiproject.site.api.SiteService"/>
            </property>
            <property name="authzGroupService">
                <ref bean="org.sakaiproject.authz.api.AuthzGroupService"/>
            </property>
            <property name="toolManager">
                <ref bean="org.sakaiproject.tool.api.ToolManager"/>
            </property>
            <property name="functionManager">
                <ref bean="org.sakaiproject.authz.api.FunctionManager"/>
            </property>
            <property name="serverConfigurationService">
                <ref bean="org.sakaiproject.component.api.ServerConfigurationService"/>
            </property>
        </bean>
    </property>
    <property name="transactionAttributes">
        <props>
            <prop key="add*">PROPAGATION_REQUIRED, -Throwable</prop>
            <prop key="del*">PROPAGATION_REQUIRED</prop>
```

```
            <prop key="get*">PROPAGATION_REQUIRED,readOnly</prop>
            <prop key="syncSites">PROPAGATION_REQUIRED,readOnly</prop>
            <prop key="doit">PROPAGATION_REQUIRED,readOnly</prop>
        </props>
    </property>
</bean>
```

Este componente ha sido creado con la misma clase factoría que el anterior componente, en sus propiedades tenemos en primer lugar *transactionManager*, que se trata de un gestor de transacciones de *JPA*, tiene una referencia al siguiente componente:

```
    <bean id="umusync-transactionManager"
          class="org.springframework.orm.jpa.JpaTransactionManager">
     <property name="entityManagerFactory" ref="umusync-entityManagerFactory"/>
     <property name="dataSource" ref="javax.sql.DataSource"/>
</bean>
```

Este objeto es de la clase *JpaTransactionManager* con la que se pueden gestionar las transacciones desde *Spring*, sus propiedades referencian a una factoría de *entityManager* y a la fuente de datos. El componente *javax.sql.DataSource* es la fuente de datos que lleva Sakai por defecto y está definida en el *kernel*. El componente *umusync-entityManagerFactory* es el siguiente:

```
    <bean id="umusync-entityManagerFactory"
          class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
           <property name="dataSource" ref="javax.sql.DataSource"/>
           <property name="persistenceXmlLocation">
                  <value>classpath:/META-INF/umusync-persistence.xml</value>
           </property>
           <property name="persistenceUnitName" value="umusync-jpa"/>
    </bean>
```

Las propiedades de este componente son, *dataSource* que referencia a la fuente de datos, también estaba incluida en el componente anterior, la propiedad *persistenceXmlLocation* que indica la localización del fichero persistence.xml, el cual tenemos en la carpeta *META-INF* dentro del módulo *dao*, y por último la propiedad *persistenceUnitName* para indicar el nombre de la unidad de persistencia.

Volviendo al componente *umu.sakai.umusync.api.ISyncManagerTarget* que estábamos desglosando, la propiedad *target* indica que es de la clase de nuestra implementación *umu.sakai.umusync.impl.SyncManager*, y las propiedades especificadas son referencias a los servicios de Sakai que utiliza nuestra clase, por cada propiedad definida en este componente existe un método en la clase *Java* llamado *setNombreDeLaPropiedad* que se ejecuta al crear el objeto.

Y por último la propiedad *transactionAtributes*, aquí se especifica que métodos de los que invoquemos desde la API iniciarán una transacción de base de datos, y unas opciones adicionales, nosotros hemos usado las siguientes:

- **PROPAGATION_REQUIRED**: Con esta opción la transacción seguirá activa al realizar invocaciones a otros métodos desde el método que inició la transacción.

- **readOnly**: Para aquellas transacciones que sean de solo lectura.
- **+/- Exception**: Si el método lanza una excepción de la clase indicada (o una subclase de la indicada) con el + se realizará un *commit* cuando esto suceda, mientras que con el – se realizará un *rollback*. Si no se indica nada el comportamiento por defecto es hacer *rollback* si se trata de una *RunTimeException*, en cualquier otro caso hace *commit* al finalizar el método.

Si el método no está indicado en esta propiedad no tendrá iniciada la transacción y por lo tanto no tendrá acceso a base de datos.

En las pruebas unitarias realizadas *Spring* carga un contexto como el del *components*, pero está limitado, no inicializa todos los servicios de Sakai, estos servicios son *"mockeados"* de modo que en las propias pruebas unitarias se define su comportamiento.

El contexto Spring de la aplicación web está definido en el fichero *local.xml*, allí se inyectan los servicios que deseamos utilizar en la presentación:

```
<util:map id="umusyncServices">
    <entry key="umu.sakai.umusync.api.ISyncManager">
        <ref bean="umu.sakai.umusync.api.ISyncManager"/>
    </entry>
    <entry key="umu.sakai.kernel.secure.UMUSecureJSP">
        <ref bean="umu.sakai.umusync.UMUSecureJSP"/>
    </entry>
    <entry key="org.sakaiproject.util.ResourceLoader">
        <ref bean="umu.sakai.umusync.ResourceLoader"/>
    </entry>
</util:map>
```

El componente *ISyncManager* es el que acabamos de explicar, el responsable de acceder a base de datos y de ejecutar toda la lógica de la herramienta.

El componente *UMUSecureJSP* sirve para consultar los permisos desde la herramienta.

```
<bean id="umu.sakai.umusync.UMUSecureJSP"
      class="umu.sakai.jsf.beans.UMUSecureJspMap">
       <property name="permissionService"
                 ref="umu.sakai.umusync.secure.UMUSecureInterceptor"/>
</bean>
```

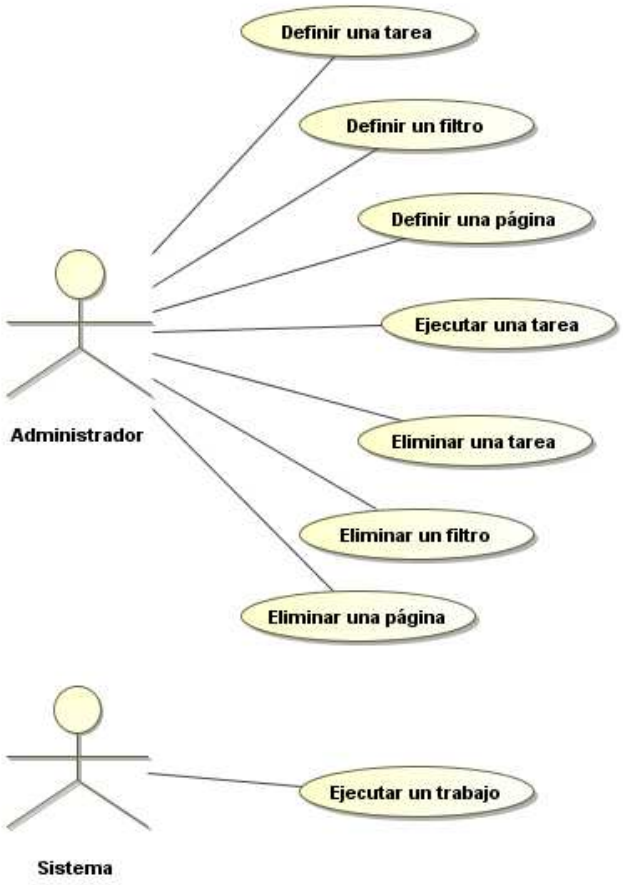
El componente *ResourceLoader* se usa para la internacionalización de textos según las preferencias idiomáticas del usuario.

```
<bean id="umu.sakai.umusync.ResourceLoader"
      class="org.sakaiproject.util.ResourceLoader">
      <constructor-arg index="0" value="umu.sakai.umusync.tool.bundle.Messages"/>
</bean>
```

4.3. Casos de uso

En el desarrollo de la herramienta hemos identificado los casos de uso que se

muestran en el siguiente diagrama:



4-12 Diagrama de casos de uso

El administrador planifica un trabajo para que posteriormente lo ejecute el sistema, pero ese caso de uso *"Planificar un trabajo"* está fuera de nuestra herramienta. Para los casos de uso identificados tenemos su descripción en los siguientes subapartados.

4.3.1. Caso de uso CU-1: Definir tarea

Objetivo: El usuario con rol administrador desea definir un filtro.

Actor Principal: Administrador

Precondiciones: Haber iniciado sesión con un usuario que tenga rol administrador[8].

[8] Decimos que un usuario tiene rol administrador en Sakai si es mienbro del sitio *Administration Workspace* y en el *realm* de ese sitio tiene asociado el rol admin.

Postcondiciones: La tarea con las opciones elegidas por el usuario se registra en el sistema.

Flujo Básico:

1. El administrador accede a la herramienta de sincronización de sitios.
2. El sistema muestra el listado de tareas definidas, si las hubiese.
3. El administrador selecciona en el menú superior la opción *Nueva Tarea.*
4. El sistema muestra el formulario de edición de tareas.
5. El administrador rellena los detalles de la tarea: Al finalizar pulsa sobre *Guardar*.
6. El sistema verifica los datos introducidos.
7. El sistema registra la tarea definida por el usuario en el catálogo de tareas.

Flujo alternativo:

*. En cualquier momento el sistema falla:
    1. El sistema muestra una página de error finalizando el proceso sin éxito.

3.a. El administrador desea modificar una tarea creada previamente:
    1. En la lista de tareas creadas selecciona aquella que desea modificar.
    2. Continúa por el paso 4 del flujo principal.

5.a. El administrador no desea definir la tarea.
    1. El administrador selecciona en el menú superior la opción *Volver*, o bien hace clic sobre el botón *Volver*.
    2. El sistema descarta los formularios rellenados por el usuario y muestra el listado de tareas.

6.a. El sistema comprueba que hay datos incorrectos (comentarios de longitud mayor a 276 caracteres, ignorar un sitio que no existe).
    1. El sistema muestra el formulario de edición de tareas indicando aquellos que tienen un valor incorrecto.
    2. Continúa por el paso 5 del flujo principal.

4.3.2. Caso de uso CU-2: Definir filtro

Objetivo: El usuario con rol administrador desea definir un filtro.

Actor Principal: Administrador

Precondiciones: Haber iniciado sesión con un usuario que tenga rol administrador.

Postcondiciones: El filtro con las opciones elegidas por el usuario se registra en el sistema.

Flujo Básico:

1. El administrador accede a la herramienta de sincronización de sitios.
2. El sistema muestra el listado de tareas definidas, si las hubiese.
3. El administrador selecciona en el menú superior la opción *Filtros*
4. El sistema muestra el listado de filtros definidos, si los hubiese
5. El administrador selecciona en el menú superior la opción *Nuevo Filtro*
6. El sistema muestra el formulario de edición de filtros.
7. El administrador rellena los detalles del filtro, nombre y condiciones, cada condición tiene una propiedad, un operador y en caso de que el operador sea binario se debe de indicar un valor. Al finalizar pulsa sobre *Añadir Filtro*.
8. El sistema verifica los datos introducidos.
9. El sistema registra el filtro definido por el usuario en el catálogo de filtros.

Flujo alternativo:

*. En cualquier momento el sistema falla:
    1. El sistema muestra una página de error finalizando el proceso sin éxito.

5.a. El administrador desea modificar un filtro creado previamente:
    1. En la lista de filtros creados selecciona aquel que desea modificar.
    2. Continúa por el paso 4 del flujo principal.

7.a. El administrador no desea definir el filtro.
    1. El administrador selecciona en el menú superior la opción *Volver*, o bien hace clic sobre el botón *Volver*.
    2. El sistema descarta los formularios rellenados por el usuario y muestra el listado de filtros.

8.a. El sistema comprueba que hay datos incorrectos (nombre de filtro ya utilizado, condición que tenga propiedad o valor sin datos).
    1. El sistema muestra el formulario de edición de filtros indicando aquellos que tienen un valor incorrecto.
    2. Continúa por el paso 7 del flujo principal.

4.3.3. Caso de uso CU-3: Definir página

Objetivo: El usuario con rol administrador desea definir una página.

Actor Principal: Administrador

Precondiciones: Haber iniciado sesión con un usuario que tenga rol administrador.

Postcondiciones: La página con las opciones elegidas por el usuario se registra en el sistema.

Flujo Básico:

1. El administrador accede a la herramienta de sincronización de sitios.
2. El sistema muestra el listado de tareas definidas, si las hubiese.
3. El administrador selecciona en el menú superior la opción *Páginas*
4. El sistema muestra el listado de páginas definidas, si las hubiese.
5. El administrador selecciona en el menú superior la opción *Nueva Página*
6. El sistema muestra el formulario de edición de páginas.
7. El administrador rellena los detalles de la página, nombre, disposición y herramientas. Ordena las herramientas arrastrándolas a la posición deseada, y al finalizar pulsa sobre *Guardar*.
8. El sistema verifica los datos introducidos.
9. El sistema registra la página definida por el usuario en el catálogo de páginas.

Flujo alternativo:

*. En cualquier momento el sistema falla:
    1. El sistema muestra una página de error finalizando el proceso sin éxito.

5.a. El administrador desea modificar una página creada previamente:
    1. En la lista de páginas creadas selecciona aquella que desea modificar.
    2. Continúa por el paso 4 del flujo principal.

7.a. El administrador no desea definir la página.
    1. El administrador selecciona en el menú superior la opción *Volver*, o bien hace clic sobre el botón *Volver*.
    2. El sistema descarta los formularios rellenados por el usuario y muestra el listado de páginas.

8.a. El sistema comprueba que hay datos incorrectos (nombre de página coincide con otra creada previamente, con el de una herramienta de Sakai, o su longitud es superior a los 50 caracteres).
    1. El sistema muestra el formulario de edición de páginas indicando que tiene un valor incorrecto.
    2. Continúa por el paso 7 del flujo principal.

4.3.4. Caso de uso CU-4: Ejecutar una tarea

Objetivo: El usuario con rol administrador desea ejecutar una tarea.

Actor Principal: Administrador

Precondiciones: Haber iniciado sesión con un usuario que tenga rol administrador. La tarea que se desea ejecutar debe de estar registrada en el catálogo de tareas.

Postcondiciones: Los sitios que cumplen las condiciones de la tarea de sincronización se actualizan según el propósito de la tarea.

Flujo Básico:
1. El administrador accede a la herramienta de sincronización de sitios.
2. El sistema muestra el listado de tareas definidas.
3. El administrador selecciona la opción de ejecutar sobre la tarea que desea ejecutar.
4. El sistema muestra toda la información de la tarea y solicita confirmación.
5. El administrador confirma la ejecución de la tarea.
6. El sistema lanza la ejecución de la tarea y muestra cuantos sitios se han modificado y cuantos sitios cumplen los requisitos de la tarea.

Flujo alternativo:
1-5. En cualquier momento el sistema falla:
    1. El sistema muestra una página de error finalizando el proceso sin éxito.

5.a. El administrador no desea ejecutar la tarea:
    1. Selecciona la opción *Volver*.
    2. El sistema muestra el listado de tareas.

6.a. El sistema falla durante la ejecución de la tarea:
    1. El sistema muestra una página de error, la tarea se puede haber ejecutado parcialmente, algunos sitios se habrán modificado y otros no.

4.3.5. Caso de uso CU-5: Ejecutar un trabajo

Objetivo: El sistema ejecuta un trabajo de Quartz para sincronizar sitios.

Actor Principal: Sistema

Precondiciones: Se cumple la condición para que se lance el disparador (*trigger*) del trabajo de Quartz de sincronización de sitios.

Postcondiciones: Los sitios que cumplen las condiciones de las tareas de sincronización habilitadas se actualizan según el propósito de cada tarea.

Flujo Básico:
1. El sistema obtiene la lista de tareas con estado activo y ejecuta todas las tareas de sincronización sobre los sitios que cumplen las condiciones.

Flujo alternativo:

1.a. La ejecución del trabajo falla por problemas con el planificador de trabajos basado en Quartz:

1. El sistema lanza la ejecución del trabajo cuando se recupera el planificador de trabajos.

1.b. El sistema falla durante la ejecución del trabajo:

1. Algunas tareas se pueden haber completado, otras puede que no se hayan iniciado, y alguna puede que se haya ejecutado parcialmente, por lo que algunos sitios se habrán sincronizado y otros no.
2. El proceso termina sin éxito.

4.3.6. Caso de uso CU-6: Eliminar una tarea

Objetivo: El usuario con rol administrador desea eliminar una tarea creada previamente.

Actor Principal: Administrador

Precondiciones: Haber iniciado sesión con un usuario que tenga rol administrador.

Postcondiciones: La tarea seleccionada por el usuario se elimina del sistema.

Flujo Básico:

1. El administrador accede a la herramienta de sincronización de sitios.
2. El sistema muestra el listado de páginas definidas.
3. El administrador selecciona la opción de eliminar sobre la tarea que desea eliminar.
4. El sistema pide confirmación al usuario.
5. El administrador confirma la eliminación pulsando en *Aceptar*.
6. El sistema elimina esa tarea del catálogo de tareas.

Flujo alternativo:

*. En cualquier momento el sistema falla:

1. El sistema muestra una página de error finalizando el proceso sin éxito.

5.a. El administrador no desea eliminar la tarea.

1. El administrador no confirma la eliminación de la tarea y pulsa en *Cancelar*.
2. El sistema muestra el listado de tareas.

4.3.7. Caso de uso CU-7: Eliminar un filtro

Objetivo: El usuario con rol administrador desea eliminar un filtro creado previamente.

Actor Principal: Administrador

Precondiciones: Haber iniciado sesión con un usuario que tenga rol administrador.

Postcondiciones: El filtro elegido por el usuario se elimina del sistema.

Flujo Básico:
1. El administrador accede a la herramienta de sincronización de sitios.
2. El sistema muestra el listado de tareas definidas, si las hubiese.
3. El administrador selecciona en el menú superior la opción *Filtros*
4. El sistema muestra el listado de filtros definidos.
5. El administrador selecciona la opción de eliminar sobre el filtro que desea eliminar.
6. El sistema pide confirmación al usuario.
7. El administrador confirma la eliminación pulsando en *Aceptar*.
8. El sistema elimina ese filtro del catálogo de filtros.

Flujo alternativo:
*. En cualquier momento el sistema falla:
    1. El sistema muestra una página de error finalizando el proceso sin éxito.

7.a. El administrador no desea eliminar el filtro.
    1. El administrador no confirma la eliminación del filtro.
    2. El sistema muestra el listado de filtros.

4.3.8. Caso de uso CU-8: Eliminar una página

Objetivo: El usuario con rol administrador desea eliminar una página creada previamente.

Actor Principal: Administrador

Precondiciones: Haber iniciado sesión con un usuario que tenga rol administrador.

Postcondiciones: La página seleccionada por el usuario se elimina del sistema.

Flujo Básico:
1. El administrador accede a la herramienta de sincronización de sitios.
2. El sistema muestra el listado de tareas definidas, si las hubiese.

3. El administrador selecciona en el menú superior la opción *Páginas*
4. El sistema muestra el listado de páginas definidas.
5. El administrador selecciona la opción de eliminar sobre la página que desea eliminar.
6. El sistema pide confirmación al usuario.
7. El administrador confirma la eliminación pulsando en *Aceptar*.
8. El sistema elimina esa página del catálogo de páginas.

Flujo alternativo:
*. En cualquier momento el sistema falla:
1. El sistema muestra una página de error finalizando el proceso sin éxito.

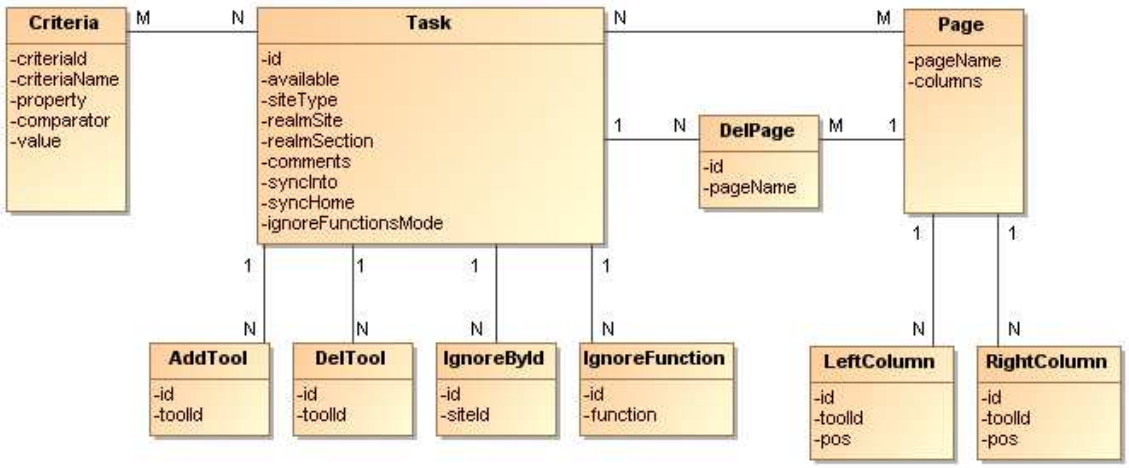
7.a. El administrador no desea eliminar la página.
1. El administrador no confirma la eliminación de la página.
2. El sistema muestra el listado de páginas.

4.4. Almacenamiento en base de datos

Las tareas de sincronización ahora van a ser persistentes, vamos a almacenar la información necesaria para ejecutar una tarea de sincronización en base de datos, de modo que se pueda modificar en cualquier momento desde la herramienta de sincronización de sitios.

El módulo ***dao*** tiene la información de base de datos, en este módulo se utiliza la tecnología de persistencia anteriormente explicada (*JPA* sobre *Hibernate*). Todas las clases *Java* de este módulo son entidades, y cada una se corresponde con una tabla de base de datos.

En el diagrama de clases 4-13 podemos ver todas las entidades definidas, así como sus relaciones.



4-13 Diagrama de clases: dao

En este diseño tenemos tres entidades **fuertes**:

- **Task**: Es la entidad principal del diseño, representa una tarea de sincronización. Tiene los siguientes atributos:
  - **id**: Número generado por una secuencia, es la clave primaria.
  - **avaliable**: Indica si está activada la tarea o no, tiene valor *S* o *N*.
  - **siteType**: Tipo de sitio.
  - **realmSite**: Plantilla de permisos para el sitio.
  - **realmSection**: Plantilla de permisos para grupos.
  - **comments**: Comentarios para identificar a la tarea.
  - **syncInto**: Indica si hay que tener en cuenta páginas multiherramienta, tiene valor *S* o *N*.
  - **syncHome**: Acciones a realizar sobre la página de inicio, tiene tres valores posibles, definidos como constantes:
    - **(cadena vacía o null)**: No cambiar nada.
    - **umusync.homepage.remove**: Eliminar la página de inicio.
    - **umusync.homepage.properties**: Según las propiedades del fichero sakai.properties para el tipo de sitio en cuestión.
  - **ignoreFuncionsMode**: Indica si hay que ignorar algún permiso, también tiene tres valores definidos en constantes, son los siguientes:
    - **N**: No ignorar ningún permiso
    - **P**: Según las propiedades del fichero sakai.properties
    - **C**: Personalizado

- **Criteria**: Representa una condición, sus atributos son:
  - **criteriaId**: Número generado por una secuencia, clave primaria.
  - **criteriaName**: Nombre por el que se agrupará la condición.
  - **property**: Propiedad a comprobar en el filtro.
  - **comparator**: Operador de comparación utilizado, será uno de los siguientes:
    - **=** : Igual.
    - ! : Distinto.
    - < : Menor alfabéticamente.
    - > : Mayor alfabéticamente.
    - E : Existe.
    - N : No existe.
    - M : Expresión regular.
  - **value**: Valor con el que comparar.

- **Page**: Representa una página de Sakai, tiene los siguientes atributos:
  - **pageName:** Nombre de la página, es la clave primaria, además no puede coincidir con el identificador de una herramienta de Sakai.
  - **columns:** Disposición de la herramienta puede ser 1 o 2.

Hemos utilizado una serie de entidades **débiles** para representar colecciones, son dependientes de las entidades *Task* o *Page*. Hemos organizado el código de la siguiente forma, por un lado tenemos las entidades fuertes en el paquete *Java* que hemos denominado ***umu.sakai.umusync.dao***, por otro lados las entidades que dependen de *Task* en ***umu.sakai.umusync.dao.task***, y por último, las que dependen de la entidad *Page* en ***umu.sakai.umusync.dao.page***.

Las entidades débiles definidas son las siguientes:

- **AddTool**: Representa las herramientas que se añaden en caso de no estar presentes en el sitio por la tarea de sincronización.
    - **id**: Número generado por una secuencia, clave primaria.
    - **tool:** Identificador de la herramienta.
- **DelTool**: Representa las herramientas que serán eliminadas del sitio por la tarea de sincronización en caso de estar presentes.
    - **id**: Número generado por una secuencia, clave primaria.
    - **tool:** Identificador de la herramienta.
- **IgnoreById**: Representa los sitios que no se deben de tener en cuenta para el trabajo de sincronización. Sus atributos son:
    - **id**: Número generado por una secuencia, clave primaria.
    - **siteId:** Identificador de sitio.
- **IgnoreFunction**: Representa aquellos permisos que no deseamos sincronizar en una tarea.
    - **id**: Número generado por una secuencia, clave primaria.
    - **function:** Identificador del permiso.

- **DelPage:** Representa las páginas definidas por nosotros que queremos eliminar de un sitio.
    - **id**: Número generado por una secuencia, clave primaria.
    - **pageName:** Identificador de la página.

- **LeftColumn:** Representa las herramientas que pertenecen a la primera o única columna de una página.
    - **id**: Número generado por una secuencia, clave primaria.
    - **pos**: Posición en la columna de la herramienta.
    - **tool:** Identificador de la herramienta.
- **RightColumn:** Representa las herramientas que pertenecen a la segunda columna en una disposición de dos columnas dentro de una página.
    - **id**: Número generado por una secuencia, clave primaria.
    - **pos**: Posición en la columna de la herramienta.
    - **tool:** Identificador de la herramienta.

Todas estas entidades, tienen una relación *ManyToOne* con la entidad de la que dependen, relación *OneToMany* visto desde la entidad fuerte, y representan listas. Estas entidades tienen una clave ajena de la entidad fuerte con la que está relacionada.

Cuando eliminamos una de estas relaciones, la entidad débil debería de dejar de existir, al utilizar JPA 1.0, no disponemos de un mecanismo automático de **eliminación de huérfanos**, que si aparece en JPA 2.0, lo hemos simulado almacenando aquellas entidades que se desligan para cuando se almacenen los cambios en base de datos se eliminen completamente de base de datos.

Además hemos definido las siguientes relaciones **ManyToMany**:

- **Task-Criteria**: Representa las condiciones que debe de cumplir una tarea para que se proceda a realizar su sincronización. Las condiciones siempre las utilizaremos agrupadas por nombre, todas las condiciones que tengan el mismo nombre representan al filtro con ese nombre. Podemos definir un filtro como una colección de objetos de la entidad *Criteria*. De modo que una tarea puede estar relacionada con un filtro. Mientras que un filtro se puede usar en muchas tareas de sincronización, o bien no usarse.
- **Task-Page**: Representa las páginas que se añaden en caso de no estar presentes en el sitio.

Para estas relaciones *ManyToMany*, se crea una tabla intermedia que contiene como claves ajenas las de las entidades que está relacionando, JPA la gestiona automáticamente.

Nosotros tenemos la entidad **DelPage** que tiene relación *OneToMany* con *Task* y *Page*, es otra forma de representar la relación *ManyToMany*, la tabla *DelPage*, es la tabla intermedia que se genera en ese tipo de relaciones. La razón de utilizar la relación *OneToMany* en vez de *ManyToMany* es que cuando se utiliza esta tabla solo necesitamos conocer su clave primaria, que es el nombre de la página.

En el fichero **umusync-orm.xml**, tenemos escritas las consultas que se harán sobre estas entidades:

```
<named-query name="listTasks">
	<query>select obj from umu.sakai.umusync.dao.Task obj</query>
</named-query>

<named-query name="listCriteria">
	<query>select obj from umu.sakai.umusync.dao.Criteria obj</query>
</named-query>

<named-query name="listPages">
	<query>select obj from umu.sakai.umusync.dao.Page obj</query>
</named-query>
```

Solo necesitamos obtener una lista con todas las tareas, otra lista con todas las condiciones y una última lista con todas las páginas, para ello hemos utilizado **named-query**, la gran ventaja de utilizar este tipo de consulta es que es totalmente independiente de la fuente de datos. Están escritas en el lenguaje *JPQL*.

En las entidades cuando se definen las relaciones con otras entidades, uno de los parámetros que se especifica es la forma de cargarla, en todas nuestras relaciones hemos indicado el modo perezoso ***Lazy***. Al utilizar esto las consultas que hemos visto solo obtienen la información de la tabla principal, y en el momento que se accede a información que no tienen disponible realizan la carga desde base de datos, sin nosotros tener que especificarle nada.

Todo lo que hemos visto hasta ahora está en el módulo *dao*, el modelo, las consultas, pero es desde el módulo *impl* donde se utiliza. En el fichero de *Spring* **db-umsync.xml** está definido el contexto JPA, que es posteriormente utilizado por *SyncManager* desde Java accediendo a la instancia que ha inicializado *Spring* de la clase **EntityManager**:

```
@PersistenceContext(unitName = "umusync-jpa")
protected EntityManager entityManager;
```

En la inicialización del contexto se indica donde está el fichero de persistencia, en nuestro caso es **umusync-persistence.xml** y en este fichero se especifica entre otras cosas cuales son las clases *dao* que utilizamos, cual es el fichero *orm* o que dialecto usa nuestra base de datos (Oracle, MySQL, etc.).

En la inicialización del contexto JPA, también se especifica un interceptor, que controla las transacciones, es decir, según al método de Java que se invoque puedo acceder a base de datos, y al finalizar dicho método según sea el resultado, lanza un tipo de excepción, finaliza normalmente, etc., se especifica si hace *commit* o *rollback*.

Este interceptor nos provocó un pequeño problema, por ejemplo, desde la herramienta (módulo *tool*) accedemos a la lista de tareas, se inicia una transacción y el método correspondiente nos proporciona el listado de objetos tipo *Task*, finalizando la transacción al devolverlo, todo correcto. Ahora de un elemento cualquiera de esta lista accedo a la lista de herramientas para añadir, como habíamos definido el acceso en modo perezoso a esta entidad obtenemos un error, ya que no hay iniciada ninguna transacción.

Para solucionar este problema se optó por crear un método que inicie transacción y cargue el elemento pasado por parámetros completamente, para realizar la carga completa simplemente hay que acceder a los atributos, forzando así que los consulte de base de datos.

La otra opción es eliminar el modo perezoso, y al obtener un listado de tareas cargaría los elementos relacionados con cada tarea, y a su vez los elementos relacionados con estos, etc., con mucha probabilidad acabaría recorriendo todas las tablas definidas en nuestra herramienta. Así que el modo perezoso bajo demanda que hemos implementado es mucho mejor.

4.5. Seguridad

En la declaración del componente Spring especificábamos un interceptor de seguridad, *UmuSecureInterceptor,* está definido en *umukernel*, tiene las siguientes funciones:

- **Registrar los permisos:** Cuando se crea el objeto registra los permisos definidos en la herramienta en Sakai, a través del servicio *functionManager*.

- **Impedir invocaciones externas**: Cualquier persona puede acceder a las herramientas que hay en la carpeta *webapp* del servidor, el interceptor de seguridad nos sirve para saber si la persona que está accediendo está conectada a Sakai y si tiene los permisos necesarios. En la configuración de nuestra herramienta hemos indicado que el permiso *umusync.VIEW* sea obligatorio para acceder a la herramienta. De modo que si alguien que no tenga ese permiso intenta acceder genere una excepción y no pueda obtener información de nuestro servicio de sincronización. La configuración de la herramienta es la siguiente:

```
<tool id="sakai.umusync" title="Sincronización de sitios" description="umusync">
       <category name="sakai.admin"/>
       <configuration name="functions.require" value="umusync.VIEW"/>
</tool>
```

- **Impedir invocaciones sin permisos**: Además del permiso para poder acceder a la herramienta, algunas operaciones dentro de la herramienta pueden estar limitadas a los usuarios que tengan un cierto permiso, en nuestra herramienta no hemos usado el permiso para limitar, en las ejecuciones hemos incluido la anotación `@HasPermission(isAdmin=true)` sobre los métodos de ejecución, de modo que solo el administrador puede ejecutar esos métodos.

Por otra parte, en el nivel de presentación al utilizar *JSF* tenemos protección ante amenazas de *XSS* (*Cross-site scripting*) en texto almacenado, al mostrarlo en la web siempre está escapado. En primeras versiones mostrábamos por pantalla texto en HTML que no era escapado para que sea interpretador por el navegador, pero ya está solventado ese problema de seguridad.

En los menús desplegables que ofrecen una lista de opciones, si modificamos el código HTML para enviar en el formulario una opción no ofrecida por el navegador originalmente nos da un error de validación.

Las consultas que se hacen en la herramienta son todas almacenadas, por lo que tampoco hay posibilidad de inyección SQL.

No hemos detectado ninguna vulnerabilidad en la aplicación web, los

analizadores de código FindBugs y PMD han dado buenos resultados.

4.6. Ayuda de la herramienta

La ayuda de nuestra herramienta es desplegada en el directorio *shared/lib* del servidor *Apache Tomcat*, desde la herramienta de ayuda de Sakai se obtienen todos los documentos de ayuda registrados para todas las herramientas.

Nuestra herramienta tiene el identificador *sakai.umusync*, por lo que para realizar la ayuda de esta herramienta debemos de crearla en el directorio ***sakai_umusync***.

La herramienta de ayuda de Sakai buscará el fichero ***help.xml***, en este fichero definimos unos componentes de la clase:

```
org.sakaiproject.component.app.help.model.ResourceBean
```

Como por ejemplo:

```
<bean id="umusyncOverview"
   class="org.sakaiproject.component.app.help.model.ResourceBean">
   <property name="docId"><value>umusyncoverview</value></property>
   <property name="name"><value>umusync Overview</value></property>
   <property name="location"><value>/sakai_umusync/overview.html</value></property>
   <property name="defaultForTool"><value>sakai.umusync</value></property>
</bean>
```

Cada componente de este tipo sirve para registrar las páginas de ayuda de la herramienta, son páginas en lenguaje *HTML*, en el ejemplo mostrado tiene activada la propiedad de página por defecto para la ayuda de nuestra herramienta, esto quiere decir que cuando se acceda a la ayuda en el icono que aparece en nuestra herramienta saldrá la página *overview.html*.

En el fichero *help.xml* para la ayuda también hemos definido otro tipo de componente, usa la clase:

```
org.sakaiproject.component.app.help.model.TableOfContentsBean
```

la cual tiene como componente interno la clase:

```
org.sakaiproject.component.app.help.model.CategoryBean
```

En nuestro caso tenemos el siguiente componente:

```
<bean id="org.sakaiproject.api.app.help.TableOfContents"
      class="org.sakaiproject.component.app.help.model.TableOfContentsBean">
     <property name="name">
            <value>root</value>
     </property>
     <property name="categories">
            <list>
                   <bean id="umusyncCategory"
                      class="org.sakaiproject.component.app.help.model.CategoryBean">
```

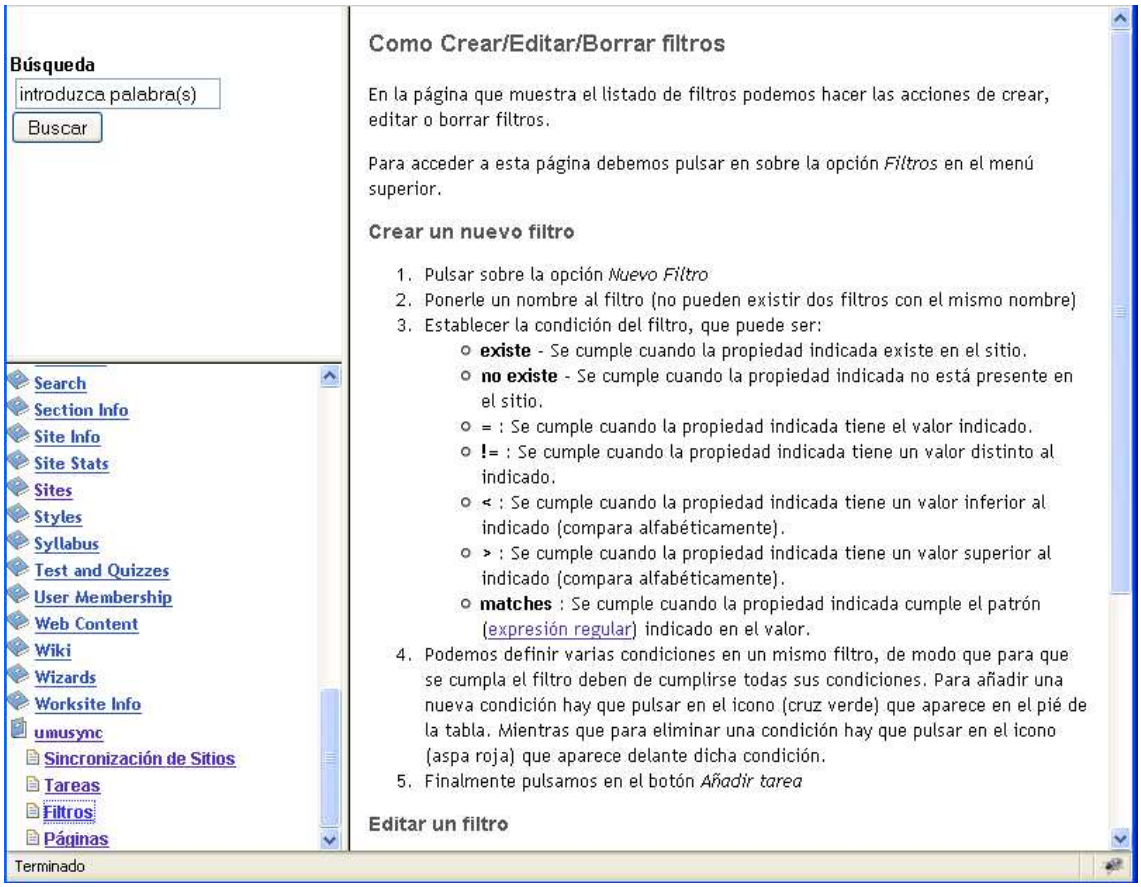
```
                        <property name="name">
                                <value>umusync</value>
                        </property>
                        <property name="resources">
                                <list>
                                        <ref bean="umusyncOverview" />
                                        <ref bean="umusyncTasks" />
                                        <ref bean="umusyncFilters" />
                                        <ref bean="umusyncPages" />
                                </list>
                        </property>
                </bean>
        </list>
    </property>
</bean>
```

Sirve para definir la tabla de contenidos para nuestra herramienta, que subpáginas aparecerán en la herramienta de ayuda.

La ayuda se puede definir en varios lenguajes, para ellos solo hay que crear documentos *help.xml* incluyendo en el nombre un identificador de la configuración regional, por ejemplo nosotros hemos creado el fichero *help_es.xml* que incluye la ayuda en español.

En Sakai la ayuda no se inicializa con el contexto *Spring* de componentes al arrancar el servidor, solo se inicializa la herramienta de ayuda, pero no su contenido. En el momento en el que una persona accede a al ayuda es cuando el sistema carga la ayuda de todas las herramientas, y en todos los lenguajes.

En la imagen 4-14 se muestra la visualización de la tabla de contenidos y de las páginas de la ayuda de nuestra herramienta dentro de la herramienta de ayuda de Sakai.



4-14 Captura de pantalla: Ayuda de la herramienta

4.7. Pruebas unitarias

Existe una técnica de programación que es desarrollo guiado por pruebas también conocido por las siglas TDD (*Test-driven development*), en el que inicialmente se definen los requerimientos, por cada uno se crea un código de prueba (que inicialmente fallará), y posteriormente se hace la implementación necesaria para que la prueba funcione. Esta técnica de desarrollo genera un código que está probado en el que rara vez el desarrollador utilizará herramientas de depuración.

Nosotros no hemos aplicado esa técnica, pero si hemos utilizado las pruebas unitarias durante el desarrollo para probar el correcto funcionamiento del módulo *impl*. La hemos aplicado únicamente a este módulo porque es en el que se realizan toda la lógica de la herramienta. La utilización de pruebas unitarias nos ha servido para detectar algunos errores de programación.

En el módulo *impl* del proyecto *umusync* hay un directorio *test* en el que se encuentra la configuración, el fichero *testng.xml* indica las clases para este propósito que hay definidas.

```
<suite name="UmuSyncImplSuite" verbose="4">
    <test name="SyncManagerTest">
            <classes>
                <class name="umu.sakai.umusync.impl.TestSyncManager"/>
            </classes>
    </test>
</suite>
```

Para el desarrollo de las pruebas unitarias se ha usado TestNG que es un Framework para realizar pruebas, está inspirado en JUnit y NUnit, utilizando un complemento de Maven con el resultado de las pruebas unitarias realizadas se generan informes de cobertura de código, los hemos incluido en este proyecto como *Anexo B*.

En el proyecto *umutests* tenemos cierta configuración que es solamente deseable cuando se están ejecutando las pruebas unitarias. En el desarrollo de pruebas unitarias se quiere comprobar el funcionamiento de la herramienta por separado, de modo que para que no interfieran otros elementos externos se utilizan *mocks*, como por ejemplo para los servicios de Sakai.

Un *mock* es un objeto falso que sustituye al original, y es el programador el que define su comportamiento, para realizar esto hemos usado Mockito, que es un proyecto de código abierto.

Dentro del proyecto *umutests* hay un fichero de componentes de Spring que define un contexto más limitado, en este contexto hay un *mock* por cada servicio de Sakai que utilizamos, este contexto de Spring se inicializa antes de la ejecución de las pruebas unitarias.

Un ejemplo de como hemos usado Mockito para las pruebas unitarias es el siguiente:

```
SiteService ss = (SiteService) this.getApplicationContext().
                                getBean("org.sakaiproject.site.api.SiteService");
```

En esta línea se carga el servicio *SiteService* de Sakai, pero como este servicio no es el original de Sakai, en *umutests* podemos encontrar la definición del componente como sigue:

```
<bean id="org.sakaiproject.site.api.SiteService"
       class="umu.sakai.umutests.SakaiMockService" factory-method="createMock">
       <constructor-arg>
              <value> org.sakaiproject.site.api.SiteService </value>
       </constructor-arg>
</bean>
```

De modo que cuando solicitamos el servicio *SiterService*, obtenemos un objeto *mock* que implementa la interface de *SiteService*, pero que todos sus métodos no hacen nada, si tienen un valor de retorno todos por defecto devuelven *null*.

Posteriormente tenemos lo siguiente:

```
Site propertyOff = Mockito.mock(Site.class);
```

Hemos creado el *mock* de la interface que representa los sitios de Sakai, este *mock* va a simular un sitio concreto. Podemos modificar este comportamiento con lo siguiente:

```
Mockito.when(propertyOff.addPage()).thenReturn(mockSitePage);
Mockito.when(propertyOff.getType()).thenReturn("asignaturaGrado");
Mockito.when(propertyOff.getId()).thenReturn("propertyOff");
```

Cuando se invoque a los métodos *addPage, getType* o *getId*, obtendremos los resultados que allí se indican, si se invocan a métodos que no hemos especificado cual es el resultado, devolverá *null*. Si es necesario también se puede decir que la invocación de un método devuelva una determinada excepción.

```
Mockito.when(ss.getSite("propertyOff")).thenReturn(propertyOff);
```

Con esta línea se define el siguiente comportamiento, cuando desde nuestro código original se realice una invocación al servicio *SiteService*, solicitando el sitio con identificador *propertyOff*, en el código legítimo que queremos comprobar su funcionamiento se obtendrá el *mock* de la clase *Site* de Sakai que hemos definido anteriormente, cuyo comportamiento ya hemos definido.

Por ejemplo, las propiedades que simulará tener ese sitio falso serán las siguientes:

```
ResourceProperties rpPropertyOff = Mockito.mock(ResourceProperties.class);
Mockito.when(rpPropertyOff.get("synchronize")).thenReturn("off");
Mockito.when(rpPropertyOff.get("id")).thenReturn("propertyOff");
Mockito.when(propertyOff.getProperties()).thenReturn(rpPropertyOff);
```

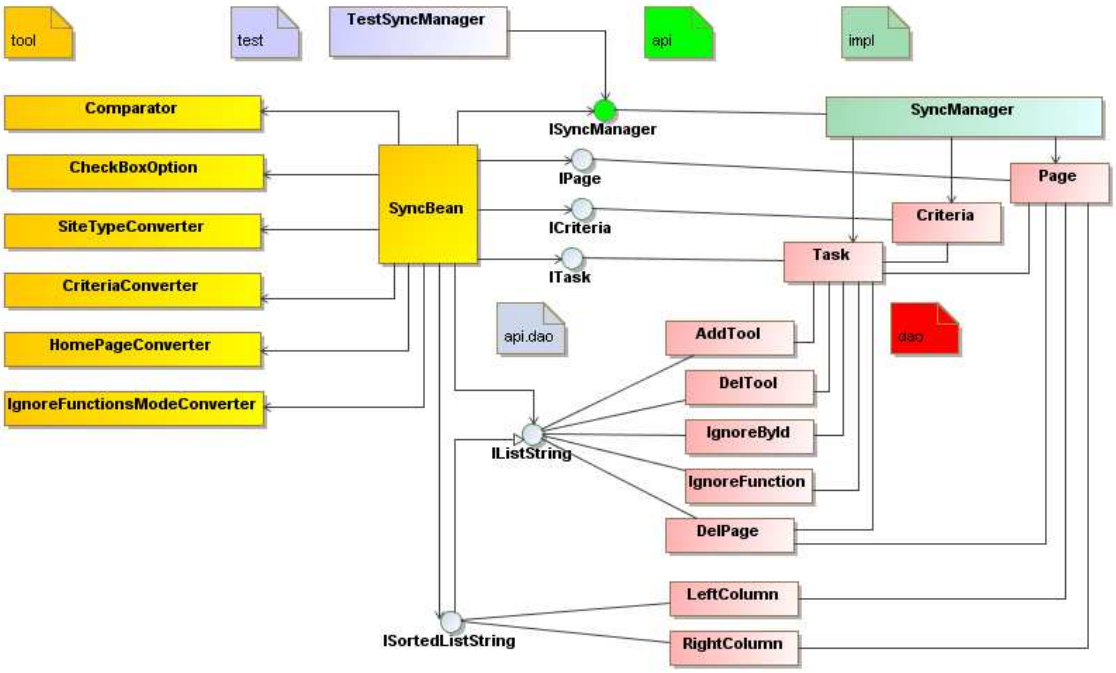
En la clase *TestSyncManager* se realiza la programación del comportamiento de los objetos *mock*, además tiene las pruebas unitarias que debe ejecutar para comprobar el correcto funcionamiento del módulo *impl*, cada prueba es un método precedido por la anotación *@Test.* Para verificar la correcta ejecución se utilizan los asertos de Java y algunas verificaciones que nos ofrece Mockito, las verificaciones se pueden ejecutar sobre objetos *mock* o sobre objetos reales que son *espiados* por Mockito, algunas posibles verificaciones son que un método no produzca excepciones, que se cumpla un determinado número de invocaciones, o en que orden se han ejecutado los métodos sobre un objeto.

La clase *TestSyncManager* definida dentro del módulo *impl* hereda de la clase *UMUTest*, la cual está definida dentro del proyecto *umutests,* que a su vez hereda de la clase de Spring *AbstractTransactionalSpringContextTests,* la cual nos proporciona los métodos *setUp* y *tearDown*, que se ejecutarán antes y después de cada prueba que realicemos, sirven para que la base de datos quede inalterada tras la ejecución de las pruebas unitarias, al finalizar cada método de

test se hace un rollback.

4.8. Diagrama de clases

Vamos a ver en primer lugar las clases que hay dentro del proyecto *umusync*, son las que intervienen en la herramienta desarrollada:



4-15 Diagrama de clases: Proyecto umusync

Los módulos que tenemos coinciden con la estructura de herramienta que habíamos definido, en el módulo pack no se desarrollada ninguna clase por lo que es el único que no aparece.

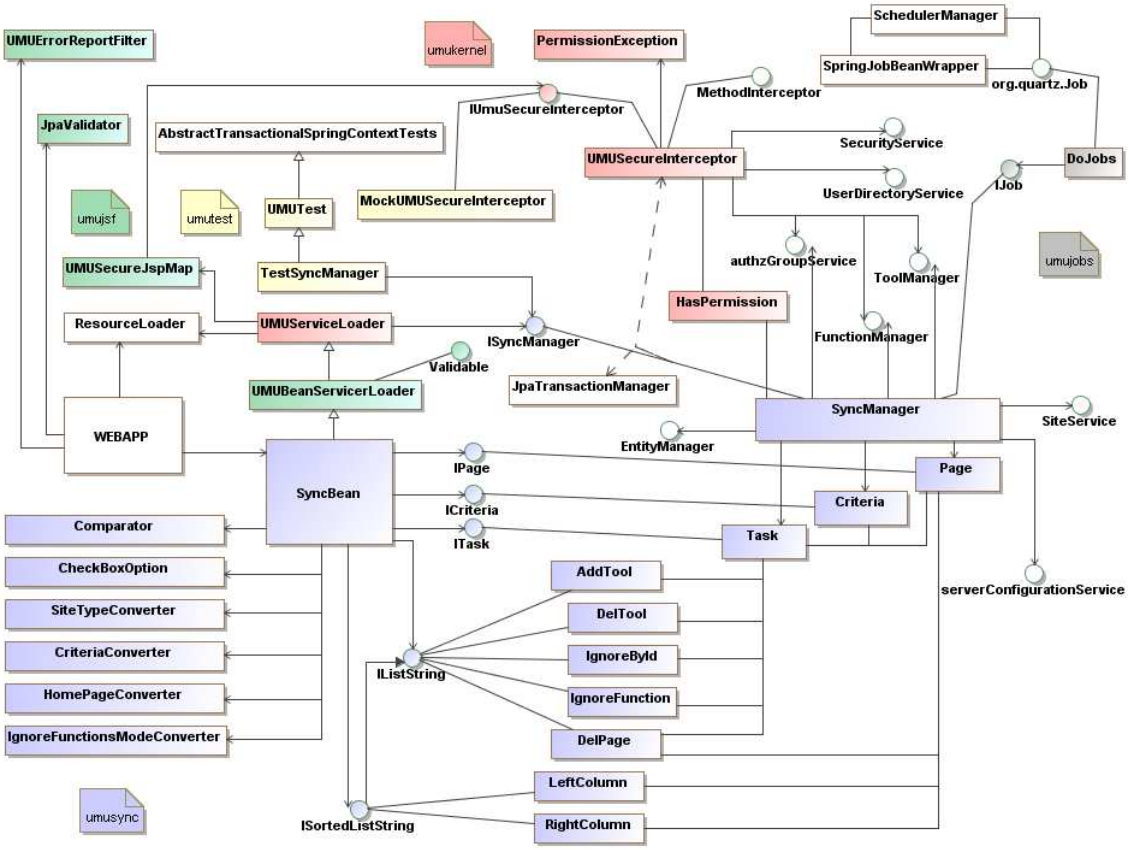
Los módulos que tenemos son los siguientes:

- **tool**: Desde la interfaz web se realizan peticiones a la clase *SyncBean* que es la que se encarga de comunicarse con el nivel de lógica, utilizando la *api* que nos provee, también se comunica con otras clases de este mismo módulo. El resultado final de la presentación, lo que aparece en el navegador, es texto, las clases conversoras sirven para representar los objetos en formato texto, y viceversa, de una cadena de texto obtener el objeto que representa. La clase *Comparator* representa a los operadores de los filtros, y la clase *CheckBoxOption* gestiona las casillas de verificación de que herramienta o página añadir/eliminar. En algunos casos es posible que la interfaz web realice peticiones directamente al conversor, en este caso no es posible y realiza una petición a SyncBean para que le

diga que conversor usar, esto es debido a que necesitan inicialización.

- **api**: Es el punto de acceso a la lógica, tiene un submódulo **api.dao** que sirve de punto de acceso a los elementos que se obtienen de base de datos.
- **impl**: La clase *SyncManager* ofrece toda la lógica necesaria para la ejecución de tareas. Obtiene información de base de datos utilizando la clase E*ntityManager* que hemos visto al explicar *JPA* en el nivel de persistencia, esta clase nos devuelve clases del módulo *dao*.
- **test**: No se corresponde con ningún módulo, los *tests* están dentro del *impl*, los hemos distinguido porque no se ejecutan en el servidor, solamente al compilar comprueban que todo funciona como debería, la clase *TestSyncManager* se encarga de comprobar el funcionamiento de la clase de lógica *SyncManager* para ello realiza peticiones a la *api* de esta clase, la interface *ISyncManager* y comprueba que el resultado de la petición es el esperado. La información sobre los tests realizados está ampliada en el *Anexo B*.
- **dao**: Las clases de este módulo son entidades que representan las tablas de base de datos, en el nivel de persistencia se corresponden con el modelo de los datos.

A continuación vamos a ver las clases que interactúan con las clases de la herramienta *umusync*:



4-16 Diagrama de clases: Proyecto umusync junto a proyectos dependientes

En el diagrama se puede observar como interactúan las clases de los proyectos *umujobs, umujsf, umukernel, umusync* y *umutest*. Las clases o interfaces que no tienen color no pertenecen a ninguno de estos proyectos, son de Sakai.

Vamos a explicar la aportación que realiza cada proyecto para el funcionamiento de la herramienta de sincronización de sitios.

- **umujobs:** En este proyecto tenemos la clase *DoJobs* que es el trabajo que se registra en el planificador de trabajos de *Quartz*, las clases relacionadas con él son las que registran y ejecutan un trabajo de *Quartz*. Anteriormente la clase *DoJobs* estaba relacionada con varios elementos *IJob* cada uno representaba una tarea de sincronización, tras los cambios realizados, en la versión actual tenemos un único elemento *IJob* en la lista, que es la clase *SyncManager*. Realiza el trabajo recorriendo la lista de tareas que tengan el estado activo.
- **umujsf:** En este proyecto están las clases relacionadas con la presentación, tenemos las siguientes clases:
    - **UMUErrorReportFilter:** Es un filtro que detecta si hay algún error para redirigir a una página de error controlada.

    - **JpaValidator:** Es un validador de los objetos *DAO*, los objetos que tienen definidas restricciones de integridad con anotaciones, por ejemplo si un objeto tiene limitada la longitud del campo, en el formulario donde se rellena el campo que está limitado, al poner el validador comprobará que no incumpla dichas restricciones, evita enviar formularios erróneos.
    - **UMUSecureJspMap:** Está relacionada con el interceptor de seguridad, comprueba si el usuario tiene un cierto permiso en el sitio, adicionalmente se le puede activar una caché para que no realice estas consultas al interceptor cada vez que aparezca referenciado un permiso y el acceso sea mucho más rápido.
    - **UMUBeanServiceLoader:** Da acceso desde la web a la consulta de permisos. Hereda de la clase de *umukernel UMUServiceLoader* la cual ofrece los servicios inyectados desde *Spring*, también implementa la interface **Validable** que está relacionada con la validación de objetos DAO dentro su clase hija de *umusync*, la clase *SyncBean*.
- **umukernel:** En el proyecto se usa la clase *UMUServiceLoader* que como hemos dicho ofrece acceso a los servicios inyectados desde *Spring*, en este caso son *ResourceLoader*, para acceso a recursos internacionalizados según preferencias idiomáticas de usuario, el citado *UMUSecureJspMap* con acceso a los permisos del usuario y da acceso a la interface *ISyncManager*, pero para llegar a su implementación tenemos que pasar una clase de este proyecto, *UMUSecureInterceptor.* Esta clase utiliza *HasPermission* que es una anotación, se pone sobre los métodos de *SyncManager* para indicar que nivel de permisos es necesario para invocarlos la cual permite o bloquea el acceso a la implementación dependiendo de los permisos que tengamos, en el caso de la herramienta diseñada solo dejará acceder al usuario con rol administrador. En caso de no tener permisos suficientes se generará una excepción del tipo *PermissionException*.
- **umusync:** Ya hemos comentado lo que hacen sus clases con detalle, de las clases nuevas que aparecen en este diagrama de clases tenemos por un lado la inyección de servicios de Sakai a la clase *SyncManager*, el interceptor de seguridad *UMUSecureInterceptor* junto con las anotaciones *HasPermission,* la gestión de transacciones usando la clase de *Spring JpaTransactionManager* y el acceso al contexto de persistencia a través de la clase *EntityManager*.
- **umutest:** En la clase *UMUTest* hay unos métodos que se ejecutan antes y después de cada *test*, que se encargan de deshacer los cambios en base de datos. En la clase *TestSyncManager* están los métodos para probar el funcionamiento de la clase *SyncManager*, solo se tiene acceso a los métodos públicos desde la *api*. Esta clase de pruebas carga un contexto de *Spring* más limitado, modifica los servicios inyectados por *mocks* que en la propia clase *TestSyncManager* define su comportamiento, cambia el interceptor de seguridad por una clase *MockUMUSecureInterceptor, el cual da acceso a todo el mundo.*

Capítulo 5

Adaptación de la herramienta

Nuestra herramienta inicialmente ha sido desarrollada con semejanza a las herramientas de Sakai de la versión 2.6.

Cuando se publicó la versión de Sakai 2.7 apareció el concepto de herramientas *Indie,* en los cambios respecto a la versión anterior algunas herramientas fueron trasladadas a este nuevo formato. En la reciente versión 2.8 ya hay una gran cantidad de las herramientas que son del tipo *Indie*.

Nuestro próximo objetivo es hacer nuestra herramienta independiente. Tendrá las ventajas de que podrá ser instalada fácilmente por otras personas, solo tendrían que incluir un pequeño módulo en su Sakai que al desplegarlo en el servidor se descargará el código compilado de nuestra herramienta, evitando la obligación de tener que compilar todo el código. También se podrán descargar el código, compilarlo y utilizar el mencionado módulo para desplegarlo en el servidor. Otra ventaja de hacer esta herramienta independiente es en el caso de las migraciones de versión de Sakai, para desplegar la herramienta en otra versión de Sakai tendremos que modificar unas variables de nuestros proyectos que indican las versiones de los componentes de Sakai que utilizamos.

5.1. Apache Maven

Apache Maven es una herramienta que automatiza el proceso de construcción de un proyecto *Java*, se puede utilizar para compilar, generar documentación o realizar pruebas.

Además su funcionalidad es extensible incluyendo complementos. Otra funcionalidad vista anteriormente es la de generar *arquetipos*, como se indicó el esqueleto de nuestra herramienta se generó con *Maven*.

Esta herramienta utiliza repositorios de artefactos, que nos permite fácilmente encontrar las librerías que pueda utilizar nuestro proyecto. Cuando instalamos *Apache Maven* tenemos un repositorio local en el que estarán los artefactos que necesite nuestro proyecto que pueden ser instalados por nosotros, descargados de un repositorio de *Maven*, o compilados por nosotros en el caso de disponer del código fuente.

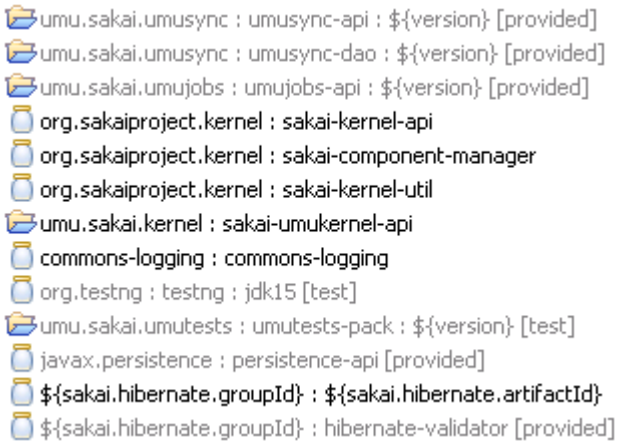
Para la definición del proyecto Maven utiliza el modelo POM (*Project Object Model*), es un documento XML, puede tener herencia, de modo que un POM hijo tiene acceso a la información que hay en su POM padre (solo puede tener un padre, en caso de tenerlo).

Los elementos clave de un POM de Maven son los siguientes:

- **project:** es el primer elemento de cada archivo pom.xml.
- **modelVersion:** este elemento obligatorio, indica la versión del modelo de objeto que el POM está usando.
- **groupId**: indica el identificador único de la organización o del grupo de elementos creados para el proyecto.
- **artifactId:** indica el nombre base único del componente o artefacto primario generado para este proyecto.
- **packaging**: indica el tipo de empaquetamiento (JAR, WAR, EAR, etc.).
- **version:** indica la versión del componente o artefacto generado por el proyecto.
- **name:** indica el nombre de despliegue usado para el proyecto.
- **url:** indica la dirección del sitio del proyecto.
- **description**: proporciona una descripción del proyecto.

La gestión de dependencias es una de las claves de Maven cuando tenemos una gran cantidad de módulos dentro de un mismo proyecto, como es el caso de Sakai.

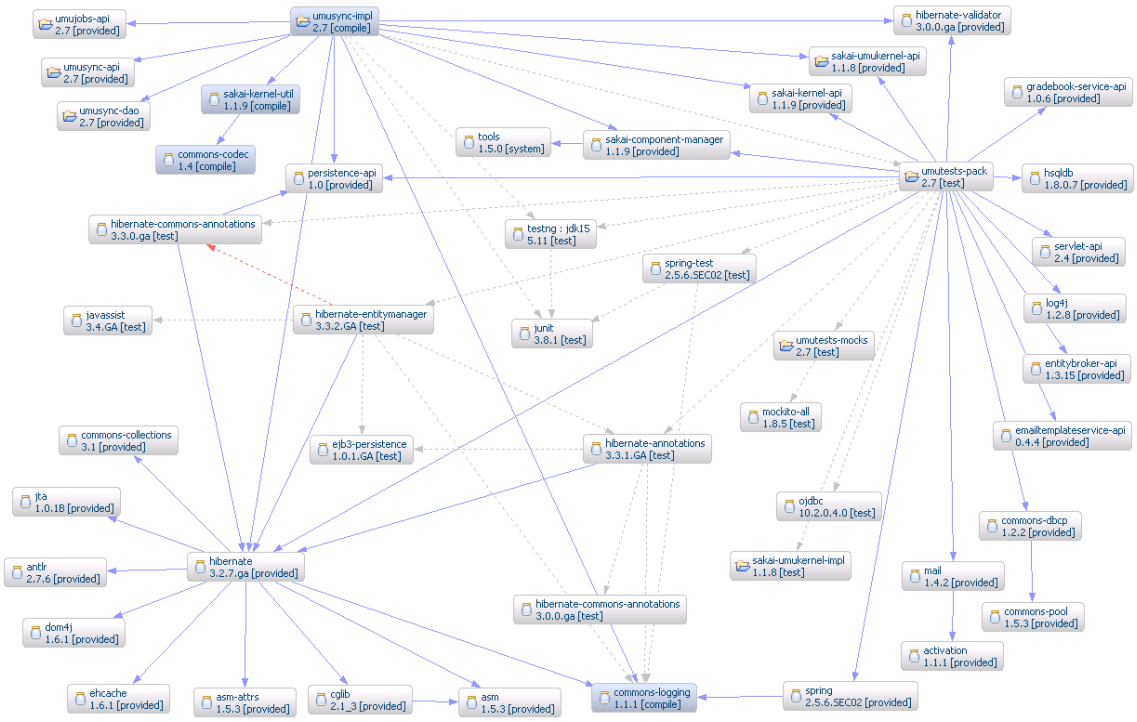
Maven nos ayuda a mantener un alto grado de control y escalabilidad. Por ejemplo, la imagen 5-1 muestra las dependencias que hemos incluido en el artefacto *umusync-impl*.



5-1 Dependencias de artefacto umusync-impl

Pero el artefacto tiene otras dependencias que vienen heredadas de sus

dependencias, ejecutando el comando **mvn dependency:resolve** podemos comprobar todas las que son. Otra opción para comprobarlas es en el IDE Eclipse utilizando el complemento de integración con Maven, una de las vistas del documento *pom.xml* es un grafo de dependencias. En la imagen 5-2 se muestran todas las dependencias que tiene el artefacto *umusync-impl*.



5-2 Grafo de dependencias del artefacto umusync-impl

En un proyecto si necesitamos usar un artefacto incluimos una dependencia con él, automáticamente estaremos incluyendo también todo lo que necesite dicho artefacto. Es muy cómodo ya que no nos tenemos que preocupar de incluir más dependencias de las que utilizamos directamente.

Podemos limitar la inclusión automática de dependencias, de modo que al incluir una dependencia a nuestro proyecto podemos indicarle, con la opción ***exclusions***, que dependencias de esta no queremos incluir. Un ejemplo de esto lo encontramos al incluir la librería *richfaces* en la herramienta, por defecto incluye *jsf*, pero la excluimos para usar en su lugar la implementación de de *jsf* que hace *myfaces*, a continuación se muestra esta dependencia en el artefacto *umusync-tool*:

```
<dependency>
	<groupId>org.richfaces.framework</groupId>
	<artifactId>richfaces-impl</artifactId>
	<version>${umu.richfaces.version}</version>
	<exclusions>
		<exclusion>
			<!-- using myfaces instead -->
			<groupId>javax.faces</groupId>
			<artifactId>jsf-api</artifactId>
		</exclusion>
```

```
                <exclusion>
                        <!-- using myfaces instead -->
                        <groupId>javax.faces</groupId>
                        <artifactId>jsf-impl</artifactId>
                </exclusion>
        </exclusions>
</dependency>
```

Al incluir una dependencia también definimos el ámbito que tiene, indicando así en que momento la necesitamos, hemos utilizado los siguientes ámbitos:

- **compile**: es el ámbito por defecto, indica que la dependencia es necesaria para compilar.
- **provided**: indica que la dependencia no es necesaria incluirla al empaquetar el proyecto, ya que estará en el servidor, por ejemplo si tenemos dependencias con servicios de Sakai el ámbito es provided porque ya están desplegados por Sakai.
- **runtime:** indica que la dependencia es necesaria durante la ejecución y no antes.
- **test:** indica que la dependencia es necesaria para ejecutar tests, no se necesita para la compilación, ni para la ejecución.

Sakai usa *Maven* para la construcción de todo el proyecto, además hay un ***plugin* de Sakai** para Maven que permite desplegar los artefactos generados (JAR, WAR, etc.) en un servidor.

5.2. Jerarquía de *POMs* en Sakai

En el primer capítulo indicábamos que el código de Sakai está dividido en tres partes, si queremos compilar todo el código tendríamos que hacerlo en el siguiente orden:

1. Kernel
2. Indie
3. Sakai

Y por cada una de las partes del proyecto tenemos un POM padre, son los que se muestran a continuación:

El POM padre de **Sakai** es *master*:

```
<groupId>org.sakaiproject</groupId>
<artifactId>master</artifactId>
```

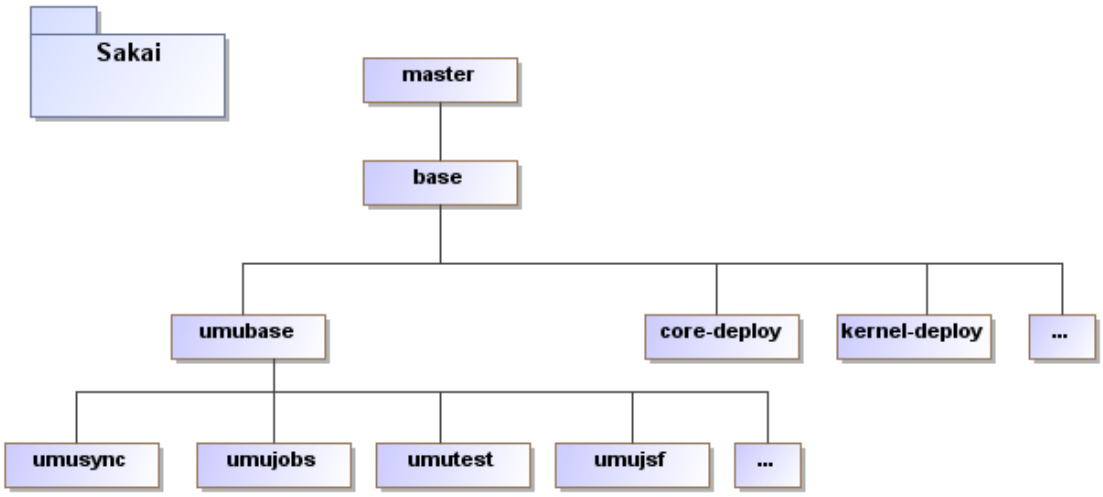
De *master* hereda el artefacto *base*:

```
<name>Sakai Core Project </name>
<groupId>org.sakaiproject</groupId>
<artifactId>base</artifactId>
```

De *base* heredan todas las herramientas de Sakai que aún no han sido adaptadas a *Indie*, las herramientas desarrolladas por la Universidad de Murcia heredan del POM umubase:

```
<name>Sakai UmuBase</name>
<groupId>umu.sakai</groupId>
<artifactId>umubase</artifactId>
```

El diagrama 5-3 representa la jerarquía de POMs en Sakai.



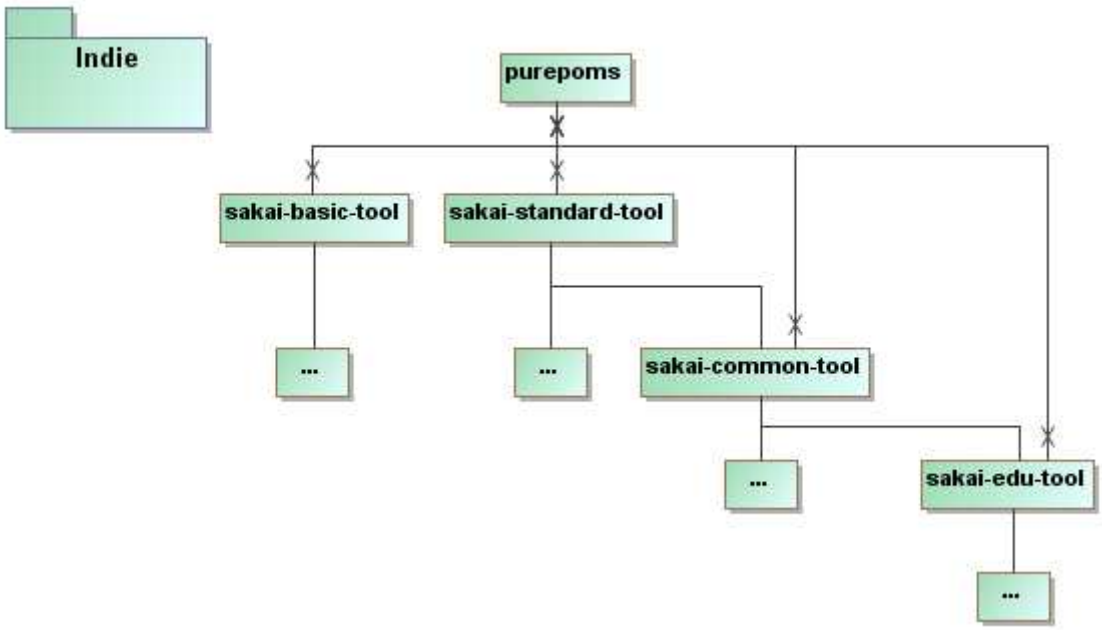
5-3 Jerarquía de POMs en Sakai

El POM padre de **Indie** es *purepoms* dentro de este proyecto encontramos varios módulos *sakai-basic-tool*, *sakai-common-tool*, *sakai-edu-tool* y *sakai-standard-tool*. Todas las herramientas que hay en *Indie*, son hijas de alguno de estos artefactos.

```
<name>Sakai Pure Poms project</name>
<groupId>org.sakaiproject.purepoms</groupId>
<artifactId>sakai-standard-tool</artifactId>
<packaging>pom</packaging>
<version>2.7.8</version>
```

El artefacto *sakai-standard-tool* es del que heredan la mayoría de las herramientas *Indie*, de él también hereda uno de módulos de *purepoms*, el artefacto *sakai-common-tool* y de este a su vez hereda *sakai-edu-tool*. Los únicos que no tienen padre son *sakai-standard-tool* y *sakai-basic-tool*.

La jerarquía de POMs en *Indie* está representada en el diagrama 5-4.



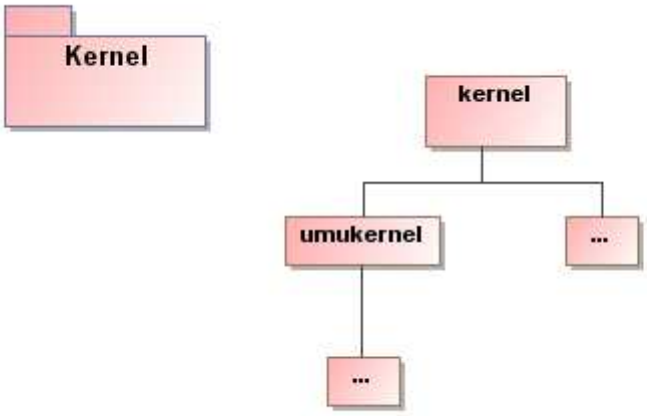
5-4 Jerarquía de POMs en Indie

Y por último el POM padre de **Kernel** es *kernel* es padre de todos los módulos que hay aquí.

```
<groupId>org.sakaiproject</groupId>
<artifactId>kernel</artifactId>
<packaging>pom</packaging>
<name>Sakai Kernel</name>
<version>1.1.9</version>
```

El desarrollo que hace la Universidad de Murcia sobre el kernel está en el proyecto *umukernel*, que es hijo de *kernel*.

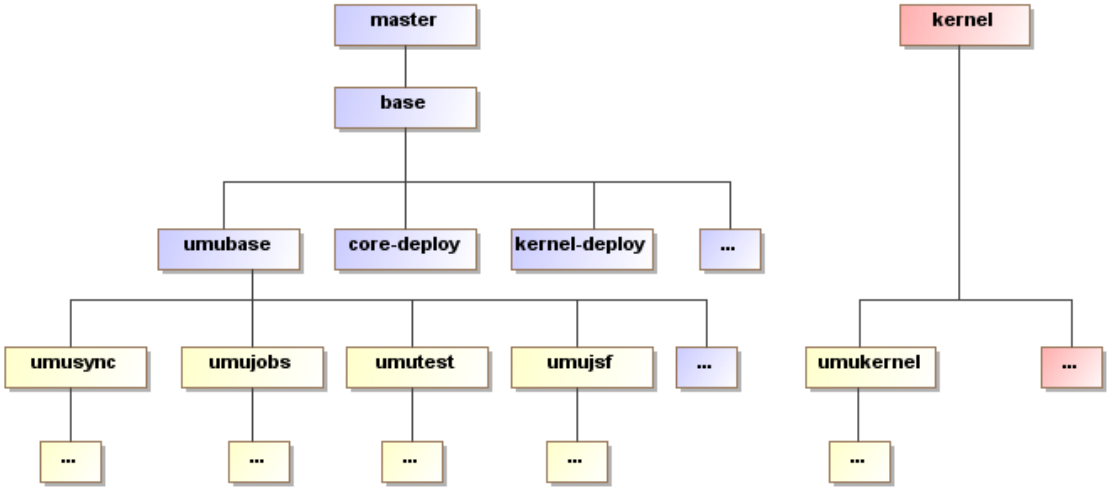
La jerarquía de POMs en Kernel está representada en el diagrama 5-5.



5-5 Jerarquía de POMs en Kernel

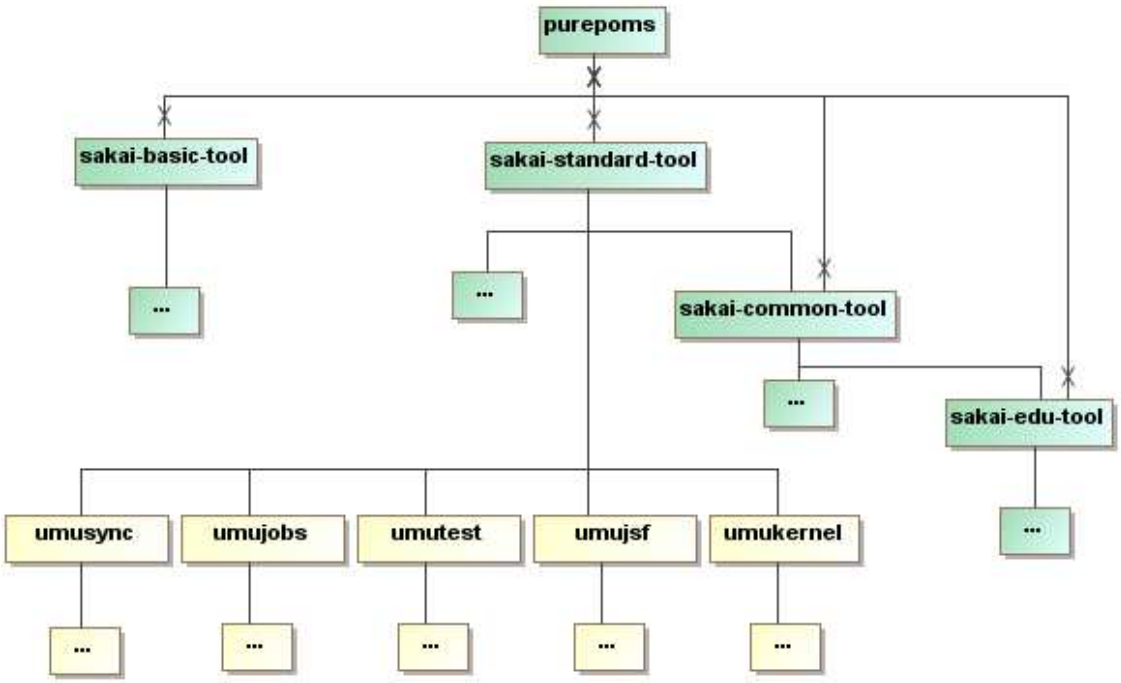
El código creado originalmente para nuestra herramienta es por una parte hijo

de *umukernel* y por otra parte de *umubase*.



5-6 Distribución de POMs en la herramienta original

Tras las modificaciones oportunas hacemos que cada artefacto sea hijo de *sakai-standard-tool*, ahora cada elemento es independiente, lo hacemos así para que puedan ser utilizados en otros proyectos.



5-7 Distribución de POMs en la herramienta independiente

Por otra parte hemos creado un nuevo POM llamado **umusync-indie-project** que incluye como módulos los artefactos que acabamos de hacer independientes, de modo que podamos compilar todos los módulos a la vez y

no tengamos que ir uno a uno.

5.3. Despliegue de Kernel/Indie en el servidor

El paso de construcción indicado anteriormente no será siempre así, el caso habitual será en el que el administrador de Sakai no tenga modificaciones en el código de Kernel o Indie. Según la configuración de *Maven* utilizada, no es necesario tener que compilar Kernel o Indie, incluso sin tener el código de Kernel o Indie podemos realizar el despliegue del proyecto Sakai completo.

En Sakai hay un módulo **kernel-deploy** por el que podemos obtener los artefactos del kernel ya compilados, y otro módulo **core-deploy**, que hace lo mismo con los elementos del Indie.

Si estamos en el caso de que hayamos modificado parte de Kernel o Indie, cuando compilamos ese código, el proceso de construcción sobrescribe los artefactos en el repositorio local de *Maven*, de modo que al desplegarlo en el servidor donde tenemos Sakai estamos incluyendo esta nueva información.

Tanto si modificamos el código de Kernel o Indie, como si no, para el despliegue de estas partes de código siempre hay que hacerlo con los módulos de Sakai mencionados anteriormente (*kernel-deploy* y *core-deploy*).

El complemento de *Maven* que realiza el despliegue en el servidor solo se usa en la parte de Sakai. En la construcción de Kernel o Indie se hace uso de un complemento de *Maven* que genera ensamblados.

5.4. Creando un *assembly*

En los módulos de Kernel e Indie podemos hallar un submódulo que es un ensamblado, incluye un plugin de Maven que genera un archivo comprimido con toda la información que desplegaría en el servidor, distribuye los ficheros en los directorios en los que copiará en el servidor: shared/lib, components, webapp, common lib, etc., de modo que cuando compilamos el código se genera este ensamblado (*assembly*).

Los módulos *kernel-deploy* y *core-deploy* son los encargados de copiar el ensamblado en el servidor, el fichero POM que define estos módulos encontramos en la propiedad ***clean.targets*** la información que tiene que limpiar en el servidor, y tienen dependencias con aquellos ensamblados que queremos desplegar en el servidor.

En un principio se creó un único módulo *assembly*. Al ser todos nuestros módulos independientes tiene el inconveniente de no tener acceso a las

propiedades, para incluir en el ensamblado las librerías que son necesarias para nuestro proyecto, no podemos acceder a la propiedad de la versión.

Por lo que se creó un submódulo *assembly* en aquellos módulos que generan artefactos de nuestro proyecto, es decir los módulos: *kernel*, *jobs* y *sync*. Tenerlo separado tiene la ventaja de poder generar el ensamblado de cada módulo por separado.

El módulo *assembly* es muy sencillo, además del POM que lo define contiene un fichero **deploy.xml** en la que se indica los artefactos necesarios y donde se alojaran en el servidor, el ejemplo más completo es el del módulo *sync*, ya que tiene elementos de *shared/lib*, *components* y *webapp*, el fichero es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<assembly>
        <id>tomcat-overlay</id>
        <formats>
                <format>zip</format>
        </formats>
        <includeBaseDirectory>false</includeBaseDirectory>
        <dependencySets>

                <!-- stuff that goes into shared -->
                <dependencySet>
                        <outputDirectory>shared/lib</outputDirectory>
                        <useTransitiveDependencies>false</useTransitiveDependencies>
                        <includes>
                            <include>umu.sakai-indie.umusync:umusync-api:jar:*</include>
                            <include>umu.sakai-indie.umusync:umusync-dao:jar:*</include>
                            <include>umu.sakai-indie.umusync:umusync-help:jar:*</include>
                        </includes>
                </dependencySet>

                <!-- stuff that goes into components -->
                <dependencySet>
                        <outputDirectory>components/umusync-pack</outputDirectory>
                        <useTransitiveDependencies>false</useTransitiveDependencies>
                        <includes>
                            <include>umu.sakai-indie.umusync:umusync-pack:war:*</include>
                        </includes>
                        <unpack>true</unpack>
                </dependencySet>

                <!-- stuff that goes into webapps -->
                <dependencySet>
                        <outputDirectory>webapps</outputDirectory>
                        <useTransitiveDependencies>false</useTransitiveDependencies>
                        <includes>
                            <include>umu.sakai-indie.umusync:umusync-tool:war:*</include>
                        </includes>
                        <unpack>false</unpack>
                </dependencySet>

        </dependencySets>
</assembly>
```

Por ahora Sakai se despliega sobre el servidor de aplicaciones **Apache Tomcat**, si se utilizase otro servidor, se podría definir otro fichero *deploy* indicando la estructura del ensamblado en ese servidor.

5.5. Despliegue de nuestra herramienta en el servidor

Para el despliegue de la herramienta hemos creado un nuevo proyecto llamado *umusync-deploy* cuyo contenido únicamente es un fichero *POM*, está en Sakai y hereda de base. En este fichero solo incluye los *assembly* generados, servirá para desplegar nuestra herramienta Indie en el servidor. Similar a lo que hace el módulo *core-deploy*.

Por último, en este *POM* que realiza el despliegue, incluimos la dirección del repositorio de *Maven* público donde publicaremos los artefactos compilados que requiere (los ensamblados).

De modo que cualquier usuario que quiera incluir la herramienta desarrollada en este proyecto dentro de la configuración de su servidor Sakai solamente tiene que descargarse el documento POM que se muestra a continuación y ejecutar el despliegue en el servidor, este es el documento *POM* del proyecto *umusync-deploy*:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=http://www.w3.org/2001/XMLSchema-
instance xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <parent>
        <artifactId>base</artifactId>
        <groupId>org.sakaiproject</groupId>
        <version>2.7.1</version>
        <relativePath>../pom.xml</relativePath>
    </parent>

    <name>Sakai Umusync Deploy</name>
    <groupId>umu.sakai-indie</groupId>
    <artifactId>umusync-deploy</artifactId>
    <packaging>pom</packaging>
    <version>2.7.1</version>

    <profiles>
        <profile>
            <id>full</id>
            <activation>
                <activeByDefault>true</activeByDefault>
            </activation>
            <properties>
                <clean.targets>
                    components/sakai-umukernel-component;
                    components/umujobs-pack;
                    components/umusync-pack;
                    webapps/umusync-tool
                </clean.targets>
                <deploy.target>tomcat-overlay</deploy.target>
            </properties>
            <dependencies>
                <dependency>
                      <groupId>umu.sakai-indie.umujobs</groupId>
                      <artifactId>umujobs-assembly</artifactId>
                      <version>1.0-SNAPSHOT</version>
                      <classifier>tomcat-overlay</classifier>
                      <type>zip</type>
                </dependency>
                <dependency>
                      <groupId>umu.sakai-indie.kernel</groupId>
                      <artifactId>umukernel-assembly</artifactId>
                       <version>1.0-SNAPSHOT</version>
                      <classifier>tomcat-overlay</classifier>
                      <type>zip</type>
```

```
                </dependency>
                <dependency>
                        <groupId>umu.sakai-indie.umusync</groupId>
                        <artifactId>umusync-assembly</artifactId>
                        <version>1.0-SNAPSHOT</version>
                        <classifier>tomcat-overlay</classifier>
                        <type>zip</type>
                </dependency>
            </dependencies>
        </profile>
    </profiles>

        <repositories>
                <repository>
                        <id>RepoUniversidadMurcia</id>
                        <name>Repository of University of Murcia</name>
                        <url> http://source.sakaiproject.org/svn/msub/um.es/maven2</url>
                </repository>
        </repositories>
</project>
```

Estando en el directorio de ese fichero solo tenemos que ejecutar el comando de despliegue con el complemento de Sakai para *Maven*:

```
mvn clean install sakai:deploy
```

Y todo nuestro desarrollo será desplegado en el servidor *Apache Tomcat.*

5.6. Base de datos adaptable

En nuestra herramienta usamos consultas con JPQL, al usar solo estos tipos de consultas y no utilizar ninguna consulta nativa podemos modificar el tipo de base de datos MySQL, Oracle, etc., y no tendremos ningún problema en la ejecución de estas consultas.

En la configuración que lleva por defecto Sakai usa como base de datos **HSQLDB** [9], almacenando la información en ficheros de texto plano en el propio servidor *Apache Tomcat*.

Hemos incluido esta base de datos para los *tests*, de modo que se almacene la información únicamente en memoria, para ello en el fichero de componentes de Spring que se carga en el caso de estar ejecutando un test hemos incluido:

```
<context:property-override location="classpath:umu/sakai/umutests/override.properties"/>
```

Esto modifica las propiedades definidas en el fichero de propiedades, cuyo contenido es:

```
umusync-entityManagerFactory.persistenceXmlLocation=classpath:/META-INF/umusync-persistence-test.xml
```

Modifica la propiedad *persistenceXmlLocation* del *bean* con identificador *umusync-entityManagerFactory*, de modo que utiliza el fichero de persistencia

[9] HSQLDB: Es un sistema gestor de base de datos relacional libre, que almacena la información en ficheros de texto plano en el propio servidor

definido para los tests.

En la configuración del archivo de persistencia se especifican una serie de propiedades, en la propiedad dialecto para una base de datos HSQLDB debemos de poner el valor *org.hibernate.dialect.HSQLDialect,* para la propiedad *hibernate.hbm2ddl.auto*, hemos indicado el valor *create-drop* para que utilice esta base de datos solo en memoria, de modo que al acceder a una base datos totalmente vacía también podemos comprobar la creación de las tablas a partir de las anotaciones de las entidades definidas en el módulo *dao*.

La herramienta inicialmente se realizó con la configuración de base de datos que hay para Sakai en la Universidad de Murcia, en esta configuración hay dos bases de datos:

- Una en la que está toda la información de la plataforma Sakai, los servicios y herramientas propias de Sakai acceden únicamente a esta base de datos.
- Y otra en la que está la información referente a la Universidad de Murcia, como pueden ser profesorado, alumnado, titulaciones, asignaturas, etc., y además aquí se almacena la información de las herramientas de Sakai desarrolladas por la Universidad de Murcia.

Al realizar el desarrollo de la interfaz web de la herramienta se proporcionó acceso a la base de datos para almacenar la información referente a las tareas de sincronización, esta información se almacena en la base de datos de la Universidad de Murcia.

Al arrancar el servidor, en la inicialización del contexto *Spring*, se cargan algunos ficheros de propiedades como pueden ser *sakai.properties*, o *security.properties*, en estos ficheros se definen unas variables con la configuración de base de datos que deseamos utilizar. Para tener una base de datos adaptable a la configuración del usuario modificamos el fichero que define la unidad de persistencia *JPA* incluyendo las variables de los ficheros de propiedades dentro de las propiedades de la base de datos en la unidad de persistencia, son las siguientes:

```
<properties>
       <property name="generateDdl" value="${auto.ddl}"/>
       <property name="hibernate.dialect" value="${hibernate.dialect}"/>
       <property name="hibernate.hbm2ddl.auto" value="update"/>
       <property name="showSql" value="${hibernate.show_sql}"/>
 </properties>
```

- La propiedad ***generateDdl*** tiene un valor de verdadero o falso, indica si se desea generar el script de creación del esquema de base de datos.
- La propiedad ***hibernate.dialect*** indica el dialecto de la fuente de datos, por ejemplo para una base de datos Oracle10g deberíamos de especificar el valor *org.hibernate.dialect.Oracle10gDialect.*
- La propiedad ***hibernate.hbm2ddl.auto*** no está entre las propiedades de

Sakai para la configuración de base de datos, nosotros le asignamos el valor *update,* sirve para la modificación del esquema de base de datos según lo especificado en el módulo *dao*, por ejemplo si en una nueva versión de la herramienta se añade una nueva columna a una de las tablas existentes, al cargar este fichero se ejecutará el comando *ALTER TABLE* correspondiente sobre la base de datos. Los valores posibles para la propiedad *hibernate.hbm2ddl.auto* son:

  - **validate**: Valida el esquema sin realizar ninguna modificación.
  - **update**: Realiza las modificaciones del esquema de base de datos que sean necesarias.
  - **create**: Crea de nuevo el esquema de base de datos, eliminando los datos anteriores.
  - **create-drop:** Elimina el esquema de base de datos al finalizar la sesión.

- Y por último la propiedad ***showSql***, que sirve para mostrar los accesos a base de datos en los ficheros de auditoría en lenguaje SQL.

Cuando nombramos el esquema de base de datos nos referimos únicamente a las tablas que se especifican en los objetos *dao* dentro de la unidad de persistencia.

De modo que hemos conseguido que en el momento de ejecución, cuando se construyen todas las clases en el arranque del servidor, al inicializar el contexto *JPA* desde Spring, nuestra herramienta creará las tablas necesarias según la configuración de base de datos especificada en los ficheros de propiedades del servidor. Siendo adaptable a cualquier base de datos utilizada en Sakai, siempre que permita crear/modificar las tablas para nuestra herramienta.

Capítulo 6

Mantenimiento de la herramienta

En este capítulo vamos a exponer técnicas para mantener la herramienta, como actualizar los módulos del proyecto sin mucho esfuerzo, utilizaremos un método para lanzar nuevas versiones del proyecto.

Por otra parte se han incluido perfiles de *Maven*, para poder adaptar la herramienta a futuras versiones de Sakai.

6.1. Subversion

Subversion es una herramienta de código abierto para el control de versiones de ficheros. Se basa en un repositorio central que actúa como servidor de ficheros, y almacena todos los cambios tanto en ficheros como en la estructura de directorios.

En Subversion son importantes los siguientes conceptos:

- **Revisión**: Es un número único, que representa el estado del repositorio al completo en un instante determinado.
- **HEAD**: Es el número de la última revisión conocida.
- **Rama**: Es una copia de una parte del repositorio en un instante concreto.

Haciendo un uso exhaustivo de Subversion distinguimos tres tipos de ramas:

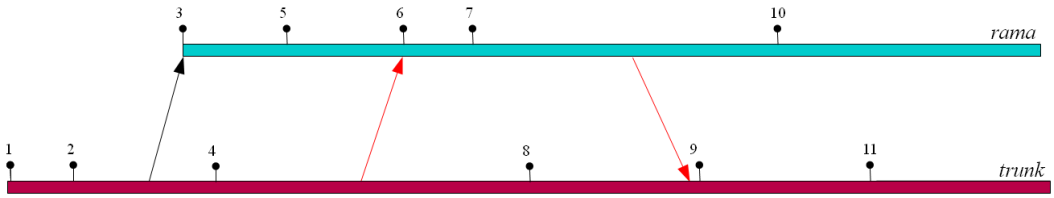
- **trunk:** Es la rama del proyecto vivo, donde está el trabajo diario.
- **branches:** Se crean ramas con diferentes motivos, mantener versiones antiguas del proyecto, realizar un desarrollo de larga duración, etc.
- **tags:** Estas ramas nunca se modifican, se usan para obtener versiones cerradas del producto o *releases*.

Las principales operaciones que hemos realizado sobre el servidor de Subversion son las siguientes:

- **Checkout:** Obtiene una copia local sincronizada con el repositorio.
- **Export:** Obtiene una copia del repositorio sin sincronización.

- **Update:** Actualiza una copia local sincronizada.
- **Commit:** Almacena en el repositorio los cambios realizados sobre una copia local sincronizada, actualizando el número de revisión del repositorio completo.
- **Revert:** Deshace los cambios realizados sobre una copia local sincronizada.
- **Merge:** Mezcla de dos ramas del repositorio idénticas.

Vamos a ilustrar con un ejemplo como funciona la operación de *merge*.



6-1 Ejemplo de operación merge

El número de revisión indica el estado del repositorio completo en un instante, cada cambio en el repositorio aumenta ese número de revisión, vamos a explicar este ejemplo siguiendo estos números de revisión:

1. Se crea el repositorio y tenemos el directorio *trunk* vacío.
2. Se modifica el contenido de *trunk*, por ejemplo creando el fichero A y B.
3. Se crea una rama llamada *rama* desde el *trunk*, es una copia idéntica de todo el contenido de *trunk,* por lo tanto tiene los ficheros A y B.
4. En *trunk* se modifica el fichero A, lo llamaremos A^t.
5. En *rama* se crea un nuevo fichero C.
6. Se realiza una operación de merge desde el *trunk* hacia la rama, al realizar una operación de merge debes indicar desde que revisión si se quieren copiar todos los cambios se indica el número de revisión siguiente al del último merge o al de la creación de la rama, en nuestro caso haremos *merge* desde la revisión 4, por lo que copiará todos los cambios que hemos hecho en *trunk* desde ese momento, que es la modificación A^t sobre el fichero A.
7. En *rama* se modifican todos los ficheros, tenemos por lo tanto A^{tr}, B^r y C^r.
8. En *trunk* se modifica el fichero B, obteniendo así B^t.
9. Operación de *merge* desde *rama* al *trunk,* es la primera vez que se realiza en esta dirección, así que para obtener todos los cambios que se han hecho en la rama debemos hacerlo desde la revisión 4 (la siguiente a la creación de la rama), esta operación nos puede dar un conflicto, hemos modificado el fichero B en las dos ramas, si es en distinta línea la operación se hará correctamente y el contenido de las dos ramas será de nuevo idéntico, si hemos modificado la misma línea tenemos que

resolver el conflicto, tenemos varias opciones:

a. Resolver conflicto con la **versión entrante**: Como resultado de la operación *merge* se obtiene las modificaciones que se hicieron en la rama.
b. Resolver conflicto con **mi versión**: En este caso prevalecen los cambios realizados en el *trunk*.
c. Resolver el conflicto **manualmente**: Quizás nos interesen los dos cambios, o solo uno de ellos, en cualquier caso tenemos que editar la línea problemática y una vez terminado debemos marcar el conflicto como resuelto.

Los directorios del repositorio Subversion pueden tener propiedades, hemos usado las siguientes:

- **svn:ignore.** Para evitar que ficheros no deseados estén en el sistema de control de versiones, por ejemplo, ficheros que genera el *IDE Eclipse*.
- **svn:externals.** Para vincular directorios de otra parte del repositorio con el directorio que contiene la propiedad.

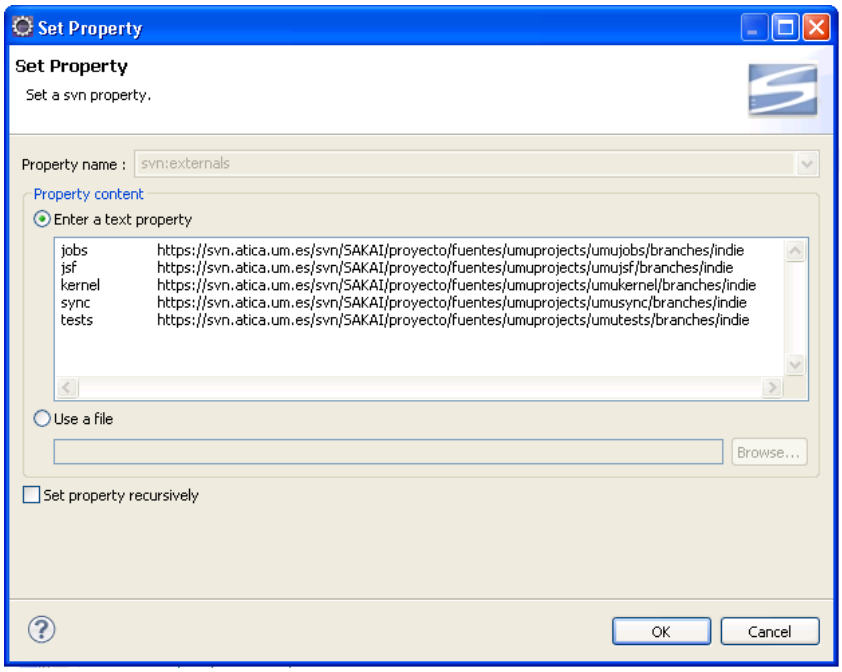
Con la propiedad *externals* conseguimos que la distribución física de directorios dentro del repositorio pueda ser distinta de la distribución del proyecto.

El manejo de *subversion* que hemos hecho en este proyecto ha sido a través del *IDE Eclipse*, para ello se ha utilizando el complemento *subclipse*, que integra *subversion* dentro de *Eclipse*.

6.2. Rama del proyecto

Por cada módulo utilizado para este proyecto he creado una rama llamada ***indie***. En esta rama he aplicado los cambios de adaptación necesarios para que pueda ser independiente, los cambios están especificados en el capítulo anterior. Estos cambios afectan a todos los POMs de todos los módulos, y al componente del módulo *sync* relacionado con la base de datos.

También he creado un nuevo directorio para el proyecto, y en su propiedad *svn:externals* tiene referencias a todas las ramas *indie* dentro del repositorio.



6-2 Captura de pantalla: Propiedad svn:externals

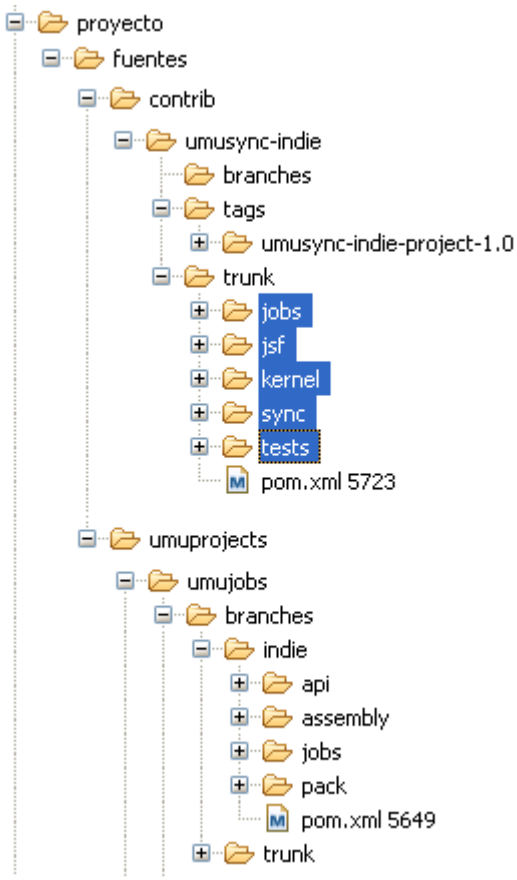
Si accedemos al directorio en el repositorio de *Subversion* solo estará el fichero *POM* que genera el proyecto, además de la propiedad *externals* anteriormente mencionada.

A partir de ahora, los cambios que se hagan en el *trunk* de estos módulos se pueden aplicar fácilmente al proyecto, solo tenemos que aplicar el proceso de *merge* entre la rama *indie* y el *trunk*. Cuando los cambios no afecten a ficheros *POM* no tendremos conflictos en el proceso. Cuando las modificaciones de alguno de estos ficheros coincide que son en las mismas líneas, entonces tendremos un conflicto y será necesario resolverlo manualmente para poder realizar la operación de *merge*.

Esta opción de tener los módulos en *externals* no nos permite cerrar versiones del proyecto correctamente, las razones están detalladas en la siguiente sección (*Release*). Por lo que tenemos que buscar una solución sin usar esta propiedad.

Eliminamos la propiedad *svn:externals* del directorio de nuestro *POM* base, los módulos de nuestro proyecto ahora están desligados del repositorio, eliminamos todo el contenido y recuperamos ese contenido creando una rama desde la rama *indie* por cada módulo.

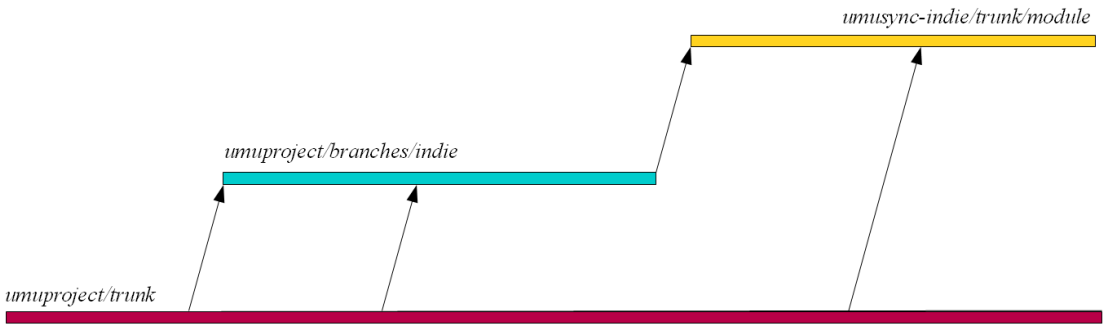
La estructura del repositorio de *subversion* queda como sigue:



6-3 Captura de pantalla: Estructura del repositorio de subversion

Los módulos que hay seleccionados *jobs, jsf, kernel, sync y tests* que están dentro del *trunk* de *umusync-indie,* son ramas de los respectivos proyectos *umujobs, umujsf, umukernel, umusync* y *umutests*.

De modo que el desarrollo que se haga en el *trunk* de *umuprojects* se trasladaría al *trunk* de *umusync-indie* haciendo la operación *merge* que se detallaba para el caso de la rama *indie*. En la imagen 6-4 se ilustran los procesos de merge realizados.



6-4 Ramas de desarrollo del proyecto

Después de crear las ramas como módulos de *umusync-indie* se realizó una operación de merge desde el *trunk* de *umuprojects* para aquellos módulos que habían sufrido cambios

desde el último *merge* con la rama *indie*, para no tener que mirar en esa rama cuando queramos volver a hacer un *merge* desde el *trunk*.

Por el momento la rama *indie* la hemos dejado abandonada, ya que ahora mismo solo se usa para este proyecto, y como ahora hemos creado una nueva rama integrada dentro del proyecto pues solo actualizamos la nueva, la anterior rama en el futuro puede ser útil si se decide independizar todos los subproyectos de Sakai, o si se realiza otro proyecto que necesite de estos subproyectos (kernel, jobs, jsf, sync o tests).

6.3. Release

Una de las virtudes de *Maven* es que puede ser fácilmente extensible con complementos, hemos usado el complemento *release*, que sirve para cerrar versiones del proyecto.

Para utilizar este complemento lo primero que debemos hacer es cambiar la versión de todos los *POMs* de la rama *indie* de nuestro proyecto, debemos ponerle una versión *SNAPSHOT*, después de esto ya no nos tendremos que volver a hacer ningún cambio de versión ya que el complemento hará los cambios automáticamente.

En el *POM* base de nuestro proyecto indicamos que vamos a usar el complemento *release* y *scm,* este último lo utiliza para conectar al servidor de *subversion* y hacer los cambios oportunos.

En la configuración del complemento **scm** debemos indicar la conexión al directorio del repositorio donde está el *POM* base del proyecto.

Para la configuración del complemento **release** lo que se le indica es la dirección del repositorio donde almacenar las versiones cerradas (*tags*) del proyecto. Como queremos que todos los módulos tengan la misma versión que el *POM* padre incluimos la opción *autoVersionSubmodules*, si no incluimos esta opción, al ejecutar el complemento nos preguntará por la versión de cada módulo, esto permitiría tener distintas versiones en los módulos para una misma versión del proyecto. Y la opción *goal* se usa para indicar que objetivos de *Maven* se ejecutarán al finalizar el proceso, si no se le indica nada tiene como objetivos por defecto *deploy site:deploy*, lo cual no nos interesa ya que nuestro proyecto independiente no se despliega directamente en el servidor y si lo hiciese usaría el complemento de Sakai.

```
<scm>
  <connection>
        scm:svn:http://xp-dev.com/svn/umusync/proyecto/umusync/trunk
  </connection>
  <developerConnection>
        scm:svn:http://xp-dev.com/svn/umusync/proyecto/umusync/trunk
  </developerConnection>
</scm>
```

```
<build>
  <plugins>
    <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-scm-plugin</artifactId>
       <version>1.2</version>
       <configuration>
              <connectionType>developerconnection</connectionType>
       </configuration>
    </plugin>
    <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-release-plugin</artifactId>
       <version>2.2.1</version>
       <configuration>
         <tagBase>
              http://xp-dev.com/svn/umusync/proyecto/umusync/tags
         </tagBase>
          <autoVersionSubmodules>true</autoVersionSubmodules>
         <goals>clean install</goals>
       </configuration>
    </plugin>
  </plugins>
</build>
```

El complemento *release* ofrece varias posibilidades de uso, son las siguientes:

- **Pre-release**: A veces es deseable lanzar una *pre-release* una versión de para realizar pruebas sobre ella antes de lanzar la versión final, el complemento tiene la opción de seguir esta estrategia, puede publicar las versiones candidatas en un servidor de pruebas, de modo que no sean públicas y después la versión estable final en el servidor público.

En nuestro caso, las versiones que se sacarán estarán probadas, por lo que no necesitamos usar esta opción.

- **Planificado**: Otra opción que ofrece el complemento es automatizar el proceso de etiquetado y hacerlo periódicamente, por ejemplo, que todas las noches cambie la versión del proyecto y lance una versión del producto.

No estamos ante este escenario, haremos cambios en *trunk*, cuando creamos conveniente lanzar una nueva versión haremos una operación de *merge* desde el *trunk*, llevando los cambios a la rama *indie*, probaremos que todo funciona correctamente en la rama y desde la rama se sacará una versión del proyecto.

- **Ensayo:** Al hacer un ensayo te muestra como quedarían los *POMs*, las versiones que le pondría a cada uno, de modo que puedas verificar que todo es correcto.

Esta opción la vemos más adecuada a nuestras necesidades. Esto es opcional, de modo que si queremos realizar un ensayo y comprobar como quedaría el proyecto antes de realizar el proceso definitivo debemos ejecutar lo siguiente:

```
mvn release:prepare -DdryRun=true
```

El ensayo no modifica ningún fichero, ni de la copia local ni del repositorio, solamente crea unas copias de los ficheros *POM* indicando como serían al cerrar la versión del proyecto y como serían en el próximo desarrollo.

Una vez comprobado que todo está correctamente, ejecutamos el siguiente comando:

```
mvn release:clean
```

Esto deshace los cambios, eliminando las copias de los *POMs*, y deja el proyecto preparado para lanzar la nueva versión, que es el proceso descrito a continuación.

Primero ejecutamos el comando de preparar la nueva versión:

```
mvn release:prepare
```

Al preparar el lanzamiento de una nueva versión el complemento realiza las siguientes tareas:

- Comprueba que no haya cambios locales en el proyecto.
- Comprueba que no haya dependencias con versiones *SNAPSHOT*.
- Pregunta por lo siguiente:
    - La versión que estás lanzando, si partimos del primer desarrollo tenemos en la rama de desarrollo la versión *1.0-SNAPSHOT* te propondrá como versión final 1.0.
    - El nombre de la *tag*, es el nombre que le pondrá al directorio que tendrá la versión, estará dentro de la ruta indicada en la configuración del complemento. El nombre propuesto para el directorio es nombre del *POM* seguido de número de versión, en nuestro caso sería *umusync-indie-project-1.0*.
    - La versión de desarrollo, cual es la versión que tendremos en nuestra rama de desarrollo, la que nos propondría al lanzar la primera versión sería *1.1-SNAPSHOT*.
- Cambia la versión *SNAPSHOT* de todos los *POMs* del proyecto por la versión de lanzamiento elegida, vamos a seguir los pasos con los valores del primer desarrollo, en este caso sería la versión 1.0.
- Modifica la información de la etiqueta *scm*, de modo que apunte a la ruta dentro del repositorio para el destino final, que es:
    - tagBase /umusync-indie-project-1.0
- Ejecuta los *tests* unitarios sobre la nueva versión, para comprobar que todo funciona correctamente.
- Realiza la operación de *commit* sobre todos los *POMs* modificados. El comentario asociado al cambio realizado es:
    - [maven-release-plugin] prepare release umusync-indie-project-1.0
- Genera la *tag* a partir de los cambios almacenados en el repositorio. El comentario asociado al cambio realizado es:

- [maven-release-plugin] copy for tag umusync-indie-project-1.0

- Cambia de nuevo la version de todos los POMs, poniéndole ahora la del siguiente desarrollo, *1.1-SNAPSHOT*, y restaura el valor de la propiedad *scm* que había modificado para generar la *tag*.
- Realiza la operación de *commit* sobre todos los *POMs* modificados. El comentario asociado al cambio realizado es:

- [maven-release-plugin] prepare for next development iteration

Si algún error hiciese que este proceso se interrumpiera y no termina de hacer todos los cambios podemos encontrarnos con algunos cambios en el repositorio de subversion no deseados, para solucionar este problema tenemos el siguiente comando:

```
mvn release:rollback
```

Y por último, para finalizar el proceso de lanzamiento tenemos que ejecutar:

```
mvn release:perform
```

Esta tarea descarga la nueva versión del proyecto y ejecuta los objetivos de *Maven* que le hemos puesto en la configuración del complemento, si no le hemos indicado ninguno ejecutará *deploy site:deploy*. Le hemos puesto que ejecute los objetivos *clean install -DcreateChecksum=true*, de modo que generará los ensamblados de la nueva versión, y además incluirá en el repositorio local los ficheros *MD5* y *SHA1* que servirán para comprobar la integridad de nuestros ficheros cuando sean publicados en el servidor.

Si no hemos finalizado el proceso ejecutando este último comando no podremos iniciar el lanzamiento de una nueva versión, si eso sucediese también tenemos la opción de ignorar el comando anterior desactivando la opción *resume*.

```
mvn release:prepare -Dresume=false
```

Para poder utilizar este complemento hemos tenido que deshacernos de las propiedades *externals*, ahora mismo no está preparado para usar esta propiedad, los resultados obtenidos no eran los esperados, ya que la *tag* mantenía la propiedad *svn:externals* apuntando a las ramas, por lo que se modificaba la versión que era distinta de la del proyecto padre.

A los desarrolladores de este complemento se les propuso[10] hacer la mejora de tener en cuenta esta propiedad. Esta petición no es de las más populares, por lo tanto no parece que este problema vaya a estar solucionado a corto plazo. En este sistema de incidencias, los usuarios votan por las mejoras o problemas que más les interesa, de modo que los desarrolladores pueden priorizar las tareas.

[10] Propuesta de mejora: https://jira.codehaus.org/browse/MRELEASE-549

6.4. Publicación en *msub*

Dentro del repositorio oficial de Sakai, existe un espacio público denominado *msub* (*massively-inclusive Subversion*) este repositorio está hospedado por la Universidad de Indiana. Sirve para proporcionar a cada institución un espacio compartido donde pueden poner sus implementaciones de Sakai, y realizar aportaciones a la comunidad, de modo que alguna funcionalidad pueda ser incluida en próximas versiones de Sakai, hay algunas restricciones sobre que es lo que se puede publicar, por ejemplo no podemos incluir software privativo. En la documentación de Sakai ([1], En la página *Deployment Source Code Repository (mSub))*, se puede encontrar más información sobre este repositorio de código.

Vamos a utilizar este espacio para publicar nuestro código, publicaremos el código de todas las versiones que cerremos con el complemento de Maven *release*.

El proyecto que incluye el código desarrollado en este proyecto se encuentra alojado en la siguiente dirección:

https://source.sakaiproject.org/svn/msub/um.es/contrib/umusync/umusync-indie/

Y además, publicaremos los compilados, es decir, vamos a usar el repositorio de Subversion como un repositorio de Maven, de modo que la persona interesada en incluir nuestra herramienta en su versión de Sakai no necesite compilarlo.

Esto lo haremos publicando los artefactos necesarios para el despliegue, estos artefactos los podemos encontrar en nuestro repositorio local de Maven. La dirección del repositorio de Maven donde publicamos los artefactos es la siguiente:

https://source.sakaiproject.org/svn/msub/um.es/maven2/

En el proceso de cierre de la versión incluíamos el parámetro *createChecksum* para generar un *hash* de los artefactos que Maven incluye en nuestro repositorio local, pero esto sólo se aplica a los que se compactan en un tipo ZIP, JAR, WAR o EAR. Los artefactos compactados en un POM no tienen el *hash* generado, hemos generado el *hash* de estos ficheros para que cuando Maven se descargue dichos artefactos no generen ningún fallo en la verificación del fichero.

La dirección donde hemos publicado los artefactos está incluida como repositorio de Maven en el proyecto que realiza el despliegue, el cual también hemos publicado, está disponible en esta dirección:

https://source.sakaiproject.org/svn/msub/um.es/contrib/umusync/umusync-deploy/

Hemos comprobado que al compilar el proyecto de despliegue utilizando el repositorio creado para obtener los compilados, además de descargarse los elementos con los que tiene dependencia (el *POM* del módulo *assembly* y el propio fichero *ZIP* del *assembly*) También se intenta descargar aquellas dependencias incluidas en el *POM* del módulo *assembly*. Esto es debido a que estas dependencias se necesitan para compilar dicho *POM*.

Por lo tanto modificamos el proyecto de despliegue, excluyendo aquellas dependencias de los *assembly* que no son necesarias para el despliegue, con esta modificación únicamente se descargará los ficheros inicialmente esperados.

6.5. Herramienta multiversión

Nuestro desarrollo ha sido sobre la versión 2.7.1, utilizando dependencias con los servicios de esa versión. Si queremos que nuestra herramienta funcione en la versión 2.8.0 debemos de modificar todas las versiones de esas dependencias en los ficheros POM de cada módulo de nuestro proyecto.

Maven ofrece la posibilidad de definir perfiles; esto nos sirve para indicar el valor de las propiedades de la versión de las dependencias en cada perfil. Creamos los perfiles ***sakai2.7.1*** y ***sakai2.8.0***, en el futuro cuando aparezcan nuevas versiones de Sakai definiremos el perfil correspondiente a esa nueva versión.

Por ejemplo, en el módulo *umujobs* tenemos una dependencia con el artefacto *sakai-component*, tiene una variable estática con la versión de Sakai.

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

Con los perfiles se resuelve esta dependencia dinámicamente, tomando un valor distinto según el perfil activo en el momento de compilar, la declaración de los perfiles de este módulo es la siguiente:

```
<profiles>
	<profile>
		<id>sakai2.7.1</id>
		<activation>
			<activeByDefault>true</activeByDefault>
		</activation>
		<properties>
			<sakai.version>2.7.1</sakai.version>
		</properties>
	</profile>
	<profile>
		<id>sakai2.8.0</id>
		<activation>
			<property>
				<name>sakai2.8.0</name>
			</property>
		</activation>
```

```
                <properties>
                        <sakai.version>2.8.0</sakai.version>
                </properties>
        </profile>
</profiles>
```

El perfil *sakai2.7.1* se activa por defecto, al activarse le asigna a la variable *${sakai.version}* el valor 2.7.1, el perfil *sakai2.8.0* no está activado por defecto, para activarlo en el momento de compilar debemos de indicárselo explícitamente con la opción ***–Psakai2.8.0***. Los perfiles también pueden ser activados en la configuración de *Maven* en el fichero *settings.xml*, o desde variables de entorno del sistema operativo.

Al compilar nuestro proyecto generamos un ensamblado, ahora mismo no tenemos la opción de distinguir si ese ensamblado es para la versión 2.7.1 o para la 2.8.0.

Los artefactos de *Maven* pueden tener un ***classifier***, este elemento sirve para distinguir un artefacto con el mismo identificador y misma versión, pero que tenga distinto contenido.

Un ejemplo habitual en el que se usa este elemento es cuando se ofrece un artefacto ejecutable con JRE 1.5, pero al mismo tiempo se ofrece el mismo artefacto que soporta la versión JRE 1.4. Para distinguirlos al primer artefacto se le añade el *classifier* jdk15, y al segundo jdk14, el usuario final de este artefacto elige que *classifier* usar. Otros ejemplos, para distinguir del sistema operativo, o el entorno, si estamos desplegando el artefacto en servidor concreto.

Al crear el *assembly* de cada módulo del proyecto en el fichero *deploy.xml*, indicábamos como identificador *tomcat-overlay*, indicando que el ensamblado es para un servidor *Apache Tomcat*. Como solo podemos tener un *classifier* por artefacto, hemos añadido el perfil seleccionado al *classifier* existente:

```
<id>tomcat-overlay-${profileName}</id>
```

De modo que ahora los *assembly* que genera nuestro proyecto al compilar tendrán el siguiente formato:

```
umukernel-assembly-1.1-tomcat-overlay-Sakai-2.7.1.zip
```

Esto nos obliga a modificar el proyecto de despliegue para que tenga en cuenta el nuevo *classifier*. En este proyecto tenemos acceso a la versión de Sakai ya que esa propiedad está definida en el *POM* master de Sakai así que mantenemos el perfil de despliegue en el servidor *Tomcat* modificando el elemento *classifier*:

```
<properties>
  <clean.targets>
        components/sakai-umukernel-component;
        components/umujobs-pack;
        components/umusync-pack;
        webapps/umusync-tool
  </clean.targets>
```

```
  <deploy.target>tomcat-overlay</deploy.target>
   <umu.sakai-indie.classifier>Sakai-${sakai.version}</umu.sakai-indie.classifier>
   <umu.sakai-indie.version>1.1-SNAPSHOT</umu.sakai-indie.version>
</properties>
<dependencies>
  <dependency>
        <groupId>umu.sakai-indie.umujobs</groupId>
        <artifactId>umujobs-assembly</artifactId>
        <version>${umu.sakai-indie.version}</version>
        <classifier>tomcat-overlay-${umu.sakai-indie.classifier}</classifier>
        <type>zip</type>
   </dependency>
  <dependency>
        <groupId>umu.sakai-indie.kernel</groupId>
        <artifactId>umukernel-assembly</artifactId>
        <version>${umu.sakai-indie.version}</version>
        <classifier>tomcat-overlay-${umu.sakai-indie.classifier}</classifier>
        <type>zip</type>
   </dependency>
  <dependency>
        <groupId>umu.sakai-indie.umusync</groupId>
        <artifactId>umusync-assembly</artifactId>
        <version>${umu.sakai-indie.version}</version>
        <classifier>tomcat-overlay-${umu.sakai-indie.classifier}</classifier>
        <type>zip</type>
   </dependency>
</dependencies>
```

Ante una migración de versión de Sakai tenemos que actualizar el POM del proyecto de despliegue, ya que al desplegar se descargará los compilados de la nueva versión.

Si para la migración de versión de Sakai queremos compilar el código de *umusync-indie* hay que modificar la versión del padre en todos los *POMs* de sus módulos hijos. La versión del *POM* padre no puede ser una variable, por lo que no podemos incluirla en los perfiles, por lo tanto hay que modificarlo manualmente.

Así que para el perfil de Sakai 2.7.1 el padre de nuestros proyectos debe de tener la versión 2.7.8, como se muestra a continuación:

```
<parent>
        <groupId>org.sakaiproject.purepoms</groupId>
        <artifactId>sakai-standard-tool</artifactId>
        <version>2.7.8</version>
</parent>
```

Y para el perfil de Sakai 2.8.0 el padre de nuestros proyectos debe de tener la versión 2.8.1.

```
<parent>
        <groupId>org.sakaiproject.purepoms</groupId>
        <artifactId>sakai-standard-tool</artifactId>
        <version>2.8.1</version>
</parent>
```

Capítulo 7

Conclusiones

En el desarrollo de este proyecto fin de grado, se ha tratado el problema de la sincronización de sitios en la plataforma Sakai, analizando, diseñando e implementando una herramienta capaz de realizar esta tarea de forma efectiva.

Estimamos que la herramienta obtenida como resultado de este proyecto cumple todos los objetivos marcados al comienzo del mismo.

La herramienta desarrollada actualiza roles-permisos y páginas-herramientas, según las configuraciones de las tareas de sincronización, por lo que cubre la funcionalidad básica.

Las tareas de sincronización se editan desde la web. Pueden ser ejecutadas desde la edición de tareas, o diferir su ejecución usando el planificador de procesos basado en Quartz.

La herramienta funciona correctamente, se han desarrollado pruebas unitarias que cubren la mayoría del código, además está siendo utilizada por la Univerisad de Murcia en un entorno de producción, sincronizando más de 2000 sitios, y no se ha detectado ningún problema en la ejecución.

Con el fin de poder aportar el trabajo a la comunidad Sakai, la herramienta desarrollada está totalmente internacionalizada. Hemos incluido los idiomas español e inglés, pero es fácilmente ampliable a otros idiomas añadiendo ficheros de propiedades de traducción a los que encontramos en el directorio *bundles* del módulo *tool*.

Además hemos conseguido hacerla independiente, es decir, la hemos adaptado a la estructura Indie, logrando una independencia de la versión de Sakai. Además hicimos la herramienta independiente de la base de datos utilizada.

Hacer la herramienta independiente ha sido un gran avance. Por una parte facilita la adaptación a las nuevas versiones de Sakai; por otra parte, permite realizar el control de versiones usando el complemento *release*.

Hemos utilizado el repositorio de Sakai además de para publicar el código, para almacenar la herramienta compilada, con esto hacemos que pueda ser incluida por cualquier persona del mundo en Sakai muy fácilmente.

La estructura de nuestra herramienta es muy similar a las herramientas de Sakai, por lo que nuestro código puede ser fácilmente entendible por los miembros de la comunidad.

El desarrollo realizado permite usar la sincronización de sitios como un servicio. Si en otro desarrollo surgiese la necesidad de tener que realizar tareas de sincronización de herramientas o de permisos, se puede disponer del servicio utilizando el componente de *Spring*. A través de la API se puede usar desde fuera de nuestra herramienta. Los escenarios para usar esta herramienta pueden ser: desde un trabajo de Quartz (en el que hemos desarrollado nosotros no accedemos por la API de *umusync,* usamos la *api* de *umujobs*), desde un servicio web integrado en Sakai o desde otra herramienta.

Apache Maven ha sido una herramienta muy útil para el desarrollo del proyecto, sin él es inviable manejar un proyecto tan grande como es Sakai.

Por último, hemos publicado en una zona del repositorio oficial de Sakai todo nuestro código, así como una pequeña documentación[11] en la que se explica cómo instalar la herramienta, su funcionamiento, como modificarla, y nos hemos puesto en contacto con los responsables de Sakai para poder incluir esta información en la página *confluence* (wikipedia de los desarrolladores de Sakai) y enviar un anuncio a la comunidad para que los usuarios de Sakai estén al corriente de la existencia de dicha herramienta, puedan probarla y enviarnos las sugerencias que crean oportunas.

[11] Documentación: https://aulavirtual.um.es/access/content/user/juanarcadio%40um.es/umusync/index.html

Capítulo 8

Vías Futuras

Una vía de futuro es la del **mantenimiento**. Dentro del proyecto se da una solución para facilitar el mantenimiento. Nuestras intenciones son utilizar esta solución para ir lanzando nuevas versiones de la herramienta. Sacaremos nuevas versiones cuando solucionemos aquellos problemas que surjan en el futuro, cuando se incluyan nuevas funcionalidades, o incluyamos perfiles para poder desplegarla en nuevas versiones de Sakai. Dado que la herramienta es de código abierto, nos pueden hacer llegar parches sobre nuestro proyecto para incluirlos en posteriores versiones.

Otra vía de futuro es la de **realizar estudios**. Durante el desarrollo de la herramienta, se ha observado que el rendimiento de las tareas de sincronización empeora ligeramente en sitios donde los usuarios participantes están asignados por un proveedor de cursos, respecto a los sitios cuyos usuarios están asignados directamente en el *realm*. Este retraso se provoca fuera del código de nuestra herramienta, una vez realizadas las modificaciones en el sitio se invoca al servicio *SiteService* de Sakai para almacenar los cambios y finalizamos la tarea de sincronización.

Sería interesante tener dos entornos similares, cuya única diferencia sea el proveedor de usuarios, y ver en qué medida afecta al rendimiento de este proceso, y así realizar estudios de rendimiento con proveedor de usuarios y sin proveedor de usuarios.

En las ejecuciones de tareas de sincronización realizadas sobre el entorno de producción de la Universidad de Murcia, en el que se han llegado a sincronizar hasta 2000 sitios en una tarea, obtenemos el siguiente rendimiento:

- Sincronizando herramientas: la tarea se ejecuta con una velocidad aproximada de 3,5 sitios/segundo.
- Sincronizando permisos: la tarea se ejecuta con una velocidad aproximada de 2 sitios/segundo.
- Sincronizando permisos y herramientas: la velocidad se asemeja a la de realizar únicamente la sincronización de permisos.

Como vía futura se podría realizar un estudio de tiempos de ejecución viendo qué variables afectan al proceso de sincronización, como pueden ser: número de usuarios, número de sitios, número de roles, si se sincronizan solo permisos, solo herramientas, tanto permisos como herramientas simultáneamente.

Una vez realizado este análisis, se podría abordar una vía futura de mejorar el **rendimiento**, habría que ver de qué modo se podría mejorar la ejecución en entornos con muchos sitios y muchos usuarios. La Universidad de Indiana tiene más de 135.000 usuarios, y más de 90.000 sitios[12] en su plataforma Sakai.

En la ejecución de los trabajos de *Quartz*, el sistema permite una configuración en *cluster*, pero cuando los nodos ejecutan los trabajos solo uno de ellos lo ejecuta. La ejecución de una tarea desde la herramienta se ejecuta sólo en el nodo donde se invocó. Se podría intentar aprovechar todos los nodos, o segmentar el trabajo de sincronización.

Otra opción de cara al futuro es **potenciar la herramienta y aumentar su ámbito de actuación**, que no solo sincronice permisos y herramientas en los sitios; incluir nuevas condiciones en los filtros, por ejemplo comprobando espacio usado en recursos o el número de usuarios pertenecientes al sitio; incluir nuevas acciones como pueden ser establecer propiedades, publicar/despublicar un sitio, hacerlo suscribible o eliminarlo

Estas nuevas características se podrían completar utilizando un *query language* similar al utilizado en la plataforma *JIRA*[13]. Obteniendo una herramienta de administración de sitios mucho más completa de lo que ofrece ahora mismo Sakai.

Y como última vía futura, **asegurar la calidad** del proyecto. En Sakai se usan servidores de QA (*Quality Assurance*), que todos los días compilan el código ejecutan los *tests* que hay programados. En nuestro caso solo hemos desarrollado pruebas unitarias que cubren una parte del módulo *impl*. Los servidores de QA mencionados, además las pruebas unitarias, suelen ejecutar otro tipo de pruebas realizadas con *Selenium*, estas pruebas sirven para comprobar el funcionamiento desde la web y además evalúan el rendimiento. Se podría proponer a los administradores de los servidores de QA de Sakai para que incluyan nuestra herramienta, y sea probada en esos servidores diariamente. Aunque lo normal es que solo tengan las herramientas que vienen con la versión cerrada de Sakai, por lo que tendríamos que configurar nuestro propio servidor de QA para que realice estas tareas.

[12] Datos oficiales de la Universidad Indiana: https://jira.sakaiproject.org/browse/PROD-3
[13] Video explicativo de *query language* de JIRA: http://www.atlassian.com/tv/episode?id=nmpviw8u7g95

Bibliografía

[1] Documentación del proyecto Sakai
http://confluence.sakaiproject.org/

[2] Alan Berg, Michael Korcuska.
Sakai Courseware Management. The Official Guide
PACKT, 2009

[3] David Roldán, Raúl E. Mengod, Daniel Merino
Sakai. Administración, configuración y desarrollo de aplicaciones.
Ra-Ma, 2011

[4] Documentación de Spring 2.5.6
http://static.springsource.org/spring/docs/2.5.6/reference

[5] Bill Dudney, Jonathan Lehr, Bill Willis, LeRoy Mattingly
Mastering JavaServer Faces
Wiley, 2004

[6] Documentación de RichFaces
http://docs.jboss.org/richfaces/

[7] Documentación de Hibernate
http://www.hibernate.org/docs

[8] Documentación de JavaEE
http://download.oracle.com/javaee/5/tutorial/doc/index.html

[9] Documentación de Maven 2
http://maven.apache.org/guides

[10] Control de versiones con Subversion
http://svnbook.red-bean.com/nightly/es/index.html

[11] Complemento Release para Maven
http://docs.codehaus.org/display/MAVENUSER/Release+Plugin

[12] Documentación del proyecto Mockito
http://mockito.org/

Anexo A
Javadoc

Si alguien desea utilizar la herramienta desarrollada como un servicio externo (servicio web, un trabajo de Quartz, otra herramienta, etc.), debe de utilizar la API, y para que esto no le resulte muy complicado, se ha documentado en inglés la interface ISyncManager con Javadoc.

Para generar la documentación hemos usado el *plug-in* ***javadoc*** de *Maven*, de modo que podemos obtener la documentación ejecutando el siguiente comando:

```
mvn javadoc:javadoc
```

La documentación se muestra a continuación:

umu.sakai.umusync.api
Interface ISyncManager

```
public interface ISyncManager
```

Method Summary	
long	**addCriteria**(ICriteria nueva) Persist criterion in database.
void	**addNewPage**(IPage p) Persist a new page in database.
void	**addTask**(ITask nueva) Persist in database the task.
void	**addUpdPage**(IPage p) Update tools of an existing page in database.
boolean	**checkSiteId**(String siteId) Check if the site with id siteId exists.
ICriteria	**createCriteria**() Create an empty criterion.
IPage	**createPage**() Create an empty page.
ITask	**createTask**() Create an empty task.
void	**delCriteria**(long pk) Remove the criterion whose identifier is the parameter received.

void	**delPage**(String pk) Remove the page whose name is the parameter received.
void	**delTask**(long pk) Remove the task whose identifier is the parameter received.
List<ICriteria>	**getCriteria**() Get all criteria from database.
IPage	**getPage**(String name) Lazy Load: Load fully the page with identifier name.
List<IPage>	**getPages**() Get all pages from database.
List<String>	**getRegisteredFunctions**()
Collection<String>	**getSectionRealms**() Get all template realms for groups registered in Sakai services.
Collection<String>	**getSiteRealms**() Get all template realms for sites registered in Sakai services.
Collection<String>	**getSiteTypes**() Get all types of site from Sakai services.
ITask	**getTask**(long taskId) Lazy Load: Load fully the task with identifier taskId.
ITask	**getTaskAndRelatedPages**(long taskId) Lazy Load: Load fully the task with identifier taskId.
List<ITask>	**getTasks**() Get all task from database.
Collection<String>	**getTools**() Get all tools from Sakai services.
String	**syncSites**(ITask tarea) Do the sync process of one task.

Method Detail

syncSites

```
String syncSites(ITask tarea)
                 throws Throwable
```

Do the sync process of one task.

Parameters:

`tarea` - sync task, it can contains filter, realms, tools, pages, etc.

Returns:

String with nm/ns where: nm: number of sites saved after modify it. ns: number of sites that matches criteria to sync.

Throws:

`Throwable` - Throws an exception if any problem occurs.

---

getTasks

`List<ITask> getTasks()`

Get all task from database.

Returns:

Collection of registered tasks.

---

getCriteria

`List<ICriteria> getCriteria()`

Get all criteria from database.

Returns:

Collection of registered criterion.

---

getPages

`List<IPage> getPages()`

Get all pages from database.

Returns:

Collection of registered pages.

---

getTask

`ITask getTask(long taskId)`

Lazy Load: Load fully the task with identifier taskId.

Parameters:

`taskId` - Identifier for tasks.

Returns:

The task linked in database.

---

getPage

`IPage getPage(String name)`

Lazy Load: Load fully the page with identifier name.

Parameters:

`name` - Identifier for pages.

Returns:

The page linked in database.

---

getTaskAndRelatedPages

`ITask getTaskAndRelatedPages(long taskId)`

Lazy Load: Load fully the task with identifier taskId. and includes the load fully of related pages.

Parameters:

`taskId` - Identifier for tasks.

Returns:

The task linked in database.

---

getTools

```
Collection<String> getTools()
```

Get all tools from Sakai services.

Returns:

Collection of registered tools.

---

getSiteTypes

```
Collection<String> getSiteTypes()
```

Get all types of site from Sakai services.

Returns:

Collection of registered site type.

---

getRegisteredFunctions

```
List<String> getRegisteredFunctions()
```

Returns:

List with the name of all registered functions.

---

getSiteRealms

```
Collection<String> getSiteRealms()
```

Get all template realms for sites registered in Sakai services.

Returns:

realms that starts with "!site.template"

---

getSectionRealms

```
Collection<String> getSectionRealms()
```

Get all template realms for groups registered in Sakai services.

Returns:

realms that starts with "!group.template"

---

createTask

```
ITask createTask()
```

Create an empty task.

Returns:

empty task

---

addTask

```
void addTask(ITask nueva)
             throws Throwable
```

Persist in database the task. If task exists do update, else do insert.

Parameters:

`nueva` - Task to persist.

Throws:

`Throwable`

---

delTask

```
void delTask(long pk)
```

Remove the task whose identifier is the parameter received. If that task doesn't exist, do nothing.

Parameters:

`pk` - Task identifier.

---

createCriteria

```
ICriteria createCriteria()
```

Create an empty criterion.

Returns:

empty criterion.

---

addCriteria

```
long addCriteria(ICriteria nueva)
```

Persist criterion in database. If criterion exists do update, else do insert.

Parameters:

`nueva` - Criterion to persist

Returns:

id Identifier of persistent object.

---

delCriteria

```
void delCriteria(long pk)
```

Remove the criterion whose identifier is the parameter received. If that criterion doesn't exist, do nothing.

Parameters:

`pk` - Criterion identifier.

---

createPage

```
IPage createPage()
```

Create an empty page.

Returns:

empty page.

---

addNewPage

```
void addNewPage(IPage p)
                throws Throwable
```

Persist a new page in database.

Parameters:

`p` - New page to persist.

Throws:

`Throwable` - Throws an exception if any problem occurs.

---

addUpdPage

```
void addUpdPage(IPage p)
                throws Throwable
```

Update tools of an existing page in database.

Parameters:

`p` - Modified page to persist.

Throws:

`Throwable` - Throws an exception if any problem occurs.

---

delPage

```
void delPage(String pk)
```

Remove the page whose name is the parameter received. If that page doesn't exist, do nothing.

Parameters:

`pk` - Name of the page.

---

checkSiteId

```
boolean checkSiteId(String siteId)
```

Check if the site with id siteId exists.

Parameters:

`siteId` - identifier of the site that we want check.

Returns:

true if site exists, false if site doesn't exist.

En el módulo **api**, tenemos el submódulo **dao**. A continuación mostramos los métodos y campos que tienen las interfaces de estos elementos DAO (Data Access Object), que son los siguientes:

- **ITask**: Para representar tareas.
- **ICriteria**: Para representar condiciones.
- **IPage**: Para representar páginas.
- **IListString**: Para representar una lista de cadenas relacionadas con otro objeto, es usado por ejemplo para herramientas, identificadores de sitio o permisos que dependen de una tarea.
- **ISortedListString**: Para representar una lista de cadenas relacionadas con otro objeto, pero nos importa en que orden estén, es usado para relacionar las herramientas de columnas que depende de una página.

umu.sakai.umusync.api.dao

Interface ITask

All Superinterfaces:

Comparable<ITask>

```
public interface ITask
extends Comparable<ITask>
```

Field Summary	
static String	**ALL_USER_SITE**
static String	**CUSTOMLIST_FUNCTION_MODE**
static String	**DEFAULT_USER_SITE**
static String	**NOT_CHANGE_FUNCTION_MODE**
static String	**NOT_CHANGE_HOME_PAGE**
static String	**PROPERTIES_FUNCTION_MODE**
static String	**PROPERTIES_HOME_PAGE**
static String	**REMOVE_THE_HOME_PAGE**

Method Summary	
void	**addFunctionToIgnore**(String functionName)
void	**addPageToAdd**(IPage page)
void	**addPageToDel**(String pageName)
void	**addToolToAdd**(String toolId)
void	**addToolToDel**(String toolId)
void	**changeAvailable**()
boolean	**delFunctionToIgnore**(IListString function)
boolean	**delPageToAdd**(IPage page)
boolean	**delPageToDel**(IListString pageName)
boolean	**delToolToAdd**(IListString toolId)
boolean	**delToolToDel**(IListString toolId)
List<IListString>	**flushOrphanRemovalEntities**()
boolean	**getAvailable**()
String	**getComments**()
List<ICriteria>	**getCriteria**()
long	**getId**()
List<IListString>	**getIgnoreById**()
List<IListString>	**getIgnoredFunctions**()
String	**getIgnoreFunctionsMode**()
String	**getIgnoreSitesById**()
List<IPage>	**getPagesToAdd**()
List<IListString>	**getPagesToDel**()

String	**getRealmSection**()
String	**getRealmSite**()
String	**getSyncHome**()
boolean	**getSyncInto**()
String	**getTipo**()
List<IListString>	**getToolsToAdd**()
List<IListString>	**getToolsToDel**()
void	**setComments**(String comments)
void	**setCriteria**(List<ICriteria> criteria)
void	**setId**(long id)
void	**setIgnoreFunctionsMode**(String ignoreFunctions)
void	**setIgnoreSitesById**(String ignore)
void	**setRealmSection**(String realmSection)
void	**setRealmSite**(String realmSite)
void	**setSyncHome**(String syncHome)
void	**setSyncInto**(boolean syncInto)
void	**setTipo**(String tipo)

Methods inherited from interface java.lang.Comparable
compareTo

umu.sakai.umusync.api.dao

Interface IPage

```
public interface IPage
```

Method Summary

void	**addToolInLeftColumn**(Integer pos, String toolElegida)
void	**addToolInRightColumn**(Integer pos, String toolId)
boolean	**delToolFromLeftColumn**(IListString tool)
boolean	**delToolFromRightColumn**(IListString tool)
Collection<IListString>	**flushOrphanRemovalEntities**()
String	**getColumns**()
List<IListString>	**getLeftColumn**()
String	**getName**()
int	**getPos**(IListString item)
List<IListString>	**getRightColumn**()
void	**numberingAfterLoad**()
void	**numberingBeforeSave**()
void	**setColumns**(String columns)
void	**setName**(String name)

umu.sakai.umusync.api.dao

Interface ICriteria

```
public interface ICriteria
```

Method Summary

String	**getComparador**()
long	**getId**()
String	**getName**()
String	**getProperty**()
String	**getValor**()
void	**setComparador**(String comparador)
void	**setId**(long id)
void	**setName**(String name)
void	**setProperty**(String property)
void	**setValor**(String valor)

umu.sakai.umusync.api.dao

Interface IListString

All Known Subinterfaces:

ISortedListString

```
public interface IListString
```

Method Summary	
long	**getId**()
String	**getString**()
void	**setMaster**(Object obj)
void	**setString**(String str)

umu.sakai.umusync.api.dao

Interface ISortedListString

All Superinterfaces:

IListString

```
public interface ISortedListString
extends IListString
```

Method Summary	
int	**getPos**()
void	**setPos**(int pos)

Methods inherited from interface umu.sakai.umusync.api.dao.IListString
getId, getString, setMaster, setString

Anexo B
Cobertura de código

Maven incluye un complemento que sirve para realizar estudios de cobertura sobre el código. Nuestro código realiza pruebas unitarias sobre el módulo *impl* que únicamente contiene la clase *SyncManager*, en la tabla B-1 se muestra el resultado del informe de cobertura.

Nombre	Clases	Concicionales	Líneas	Métodos
SyncManager.java	100% 1/1	57% 138/242	77% 308/401	98% 45/46

B-1 Informe de cobertura: Módulo impl

Llegar al 100% de cobertura es una utopía, hay código que no se puede llegar a comprobar con *mocks*. Por ejemplo, *Mockito* no deja simular el método *equals*. En nuestro código usamos la clase *CollectionUtils* de la librería Apache Commons, intermante utiliza el método *equals* para realizar las operaciones necesarias. En nuestro caso estamos manejando colecciones de objetos "*mockeados*", con lo que se tiene en cuenta el método equal de la clase *Mock* y no de la clase real, obteniendo resultados que no son esperados.

Además, he comprobado que algunos resultados no son correctos, por ejemplo en esta línea:

```
if (addTool.isEmpty() &&                                    C1
        delTool.isEmpty() &&                                C2
        addPage.isEmpty() &&                                C3
        delPage.isEmpty() &&                                C4
        ITask.NOT_CHANGE_HOME_PAGE.equals(syncHome))        C5
```

El informe de cobertura dice que estamos cubriendo 60% de condicionales, 6/10. Las posibilidades existentes aparecen en la tabla B-2. Por lo que realmente nuestro código está cumpliendo el 100% de los casos.

CASO	C1	C2	C3	C4	C5	**Resultado**
1	V	V	V	V	V	V
2	V	V	V	V	F	F
3	V	V	V	F	-	F
4	V	V	F	-	-	F
5	V	F	-	-	-	F
6	F	-	-	-	-	F

B-2 Tabla: Posibilidades de ejecución

Donde:

- C1, C2,.. C5 expresan las 5 condiciones de la línea código.
- V: Indica que se cumple la condición.
- F: Indica que no se cumple la condición.
- - : Indica que no será comprobado.
- Resultado: Indica el valor de la línea condicional *if*.

Además, Maven incluye otro complemento, *surefire,* que genera otro tipo de informes. En los informes de *surefire* aparecen todas las pruebas ejecutadas, el tiempo que tardan, los datos de entrada proporcionados a cada prueba, si se espera una excepción muestra la pila de la excepción obtenida. A continuación se muestran algunos ejemplos:

Este es el resultado de la ejecución de todas nuestras pruebas unitarias:

SyncManagerTest

Tests passed/Failed/Skipped:	53/0/0
Started on:	Thu Sep 01 17:27:28 CEST 2011
Total time:	7 seconds (7359 ms)

En la prueba unitaria *testAddTaskWithInvalidValues*, se espera una excepción, el complemento muestra el resultado completo de la excepción.

umu.sakai.umusync.impl.TestSyncManager:testAddTaskWithInvalidValues

tipo: la longitud debe ser entre 0 y 30
umu.sakai.umusync.impl.SyncManager.addTask(SyncManager.java:183)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
at java.lang.reflect.Method.invoke(Method.java:597)
at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:307)
at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:182)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:149)
at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:106)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:171)
at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:204)
at $Proxy29.addTask(Unknown Source)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
at java.lang.reflect.Method.invoke(Method.java:597)
at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:307)
at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:182)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:149)
at umu.sakai.umutests.MockUMUSecureInterceptor.invoke(MockUMUSecureInterceptor.java:19)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:171)
at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:204)
at $Proxy29.addTask(Unknown Source)
at umu.sakai.umusync.impl.TestSyncManager.testAddTaskWithInvalidValues(TestSyncManager.java:993)
… 30 lines not shown

En la prueba unitaria *testCheckSiteIdEncontrado*, se incluye un proveedor de datos, se muestran los datos con los que se ha ejecutado dicha prueba.

umu.sakai.umusync.impl.TestSyncManager:testCheckSiteIdEncontrado

Parameter #1
~admin
Existente

Anexo C
Código fuente

Todo el código fuente desarrollado para este proyecto está alojado en la siguiente dirección:

https://source.sakaiproject.org/svn/msub/um.es/contrib/umusync/umusync-indie/

De esa dirección se accede a los siguientes directorios:

- **branches**: Por el momento vacío, en el futuro puede incluir algún desarrollo.
- **tags**: Encontramos ***umusync-project-1.0*** que es la versión cerrada del proyecto en Septiembre de 2011, no se modificará, en el futuro aparecerán nuevas versiones, pero esta versión nunca será modificada.
- **trunk**: Desarrollo vivo del proyecto, sobre el que se harán las modificaciones.

El proyecto que realiza el despliegue es muy sencillo, únicamente tiene un fichero *POM*, de cualquier modo sobre él hemos puesto los directorios típicos de Subversion, y están en la dirección:

https://source.sakaiproject.org/svn/msub/um.es/contrib/umusync/umusync-deploy/

Encontramos el *tag* ***umusync-deploy-1.0*** que es la encargada de desplegar en el servidor la versión 1.0 de nuestra herramienta.

El código fuente de la versión 1.0, cerrada el 4 de Septiembre de 2011 lo podemos descargar de las siguientes direcciones:

- **Código del proyecto independiente:**

https://source.sakaiproject.org/svn/msub/um.es/contrib/umusync/umusync-indie/tags/umusync-indie-project-1.0

- **Código del proyecto para desplegar:**

https://source.sakaiproject.org/svn/msub/um.es/contrib/umusync/umusync-deploy/tags/umusync-deploy-1.0