OdA J2EE

Aplicación del Metamodelo Relacional para el Soporte de Metadatos Específicos del Dominio en Repositorios de Objetos de Aprendizaje

Omar Ruiz Rodríguez
Andrea Seco Peña
Jaime Velay Valor

GRADO EN INGENIERÍA DEL SOFTWARE
FACULTAD DE INFORMÁTICA
DEPARTAMENTO DE INGENIERÍA DEL SOFTWARE E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE GRADO EN INGENIERÍA DEL SOFTWARE
13 de septiembre de 2013
Director: Antonio Navarro Martín

Autorización de difusión y utilización

13 de septiembre de 2013,

Los alumnos abajo firmantes autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

OMAR RUÍZ RODRÍGUEZ ANDREA SECO PEÑA JAIME VELAY VALOR

Resumen

Los objetos de aprendizaje electrónico (LOs) se conciben comúnmente como unidades digitales de información utilizadas para la enseñanza y el aprendizaje. Para facilitar su clasificación para propósitos pedagógicos, los objetos de aprendizaje electrónico (LOs) se complementan con metadatos (por ejemplo, el autor del LO) (Navarro et al. 2013).

Tal y como se describe en (Navarro et al., 2013), ciertos objetos de aprendizaje (LOs) complejos, necesitan ser complementados con diferentes tipos de información dependiente del dominio:

1. Metadatos de clasificación para mejorar la contextualización, búsqueda y recuperación (por ejemplo, la estructura de etiquetado de un sitio arqueológico donde se ha encontrado un objeto arqueológico).

2. Datos adicionales que puede enriquecer el LO (por ejemplo, el peso y otras dimensiones de un artefacto arqueológico descrito en un *podcast*).

Nos referimos a objetos de aprendizaje (LOs) mejorados con información de dominio dependiente, como *Objetos de Aprendizaje Híbridos* (HLOs).

Sin embargo, la mayoría de Repositorios de Objetos de Aprendizaje (LORs) sólo permiten que un conjunto predeterminado de metadatos sea utilizado en la clasificación de Objetos de Aprendizaje (LOs).

Esta rigidez es inapropiada cuando la información depende de esquemas de información dependiente del dominio que pueden utilizarse para la navegación, recuperación y secuenciación pedagógica de HLOS.

Por lo tanto, si se quiere gestionar LOs LOs que necesitan ser etiquetados con información perteneciente a dominios específicos, es necesaria la construcción de repositorios software personalizados por dominios.

Este proyecto presenta un solo LOR para clasificar y enriquecer LOs de acuerdo con esquemas de información dependientes del dominio, los cuales pueden ser cambiados dinámicamente después de su definición. La cuestión clave en nuestro enfoque es la presencia de un metamodelo relacional para la definición dinámica del esquema de la base de datos relacional dependiente del dominio utilizado para la clasificación y el enriquecimiento de los LOs.

Palabras clave:

Repositorio de objetos de aprendizaje, metadatos, información de esquema de dominio dependiente, clasificación, Oda (Objeto de Aprendizaje).

Abstract

Electronic learning objects (LOs) are commonly conceived of as digital units of information used for teaching and learning. To facilitate their classification for pedagogical planning and retrieval purposes, LOs are complemented with metadata (e.g., the author).

These metadata are usually restricted by a set of predetermined tags to which the classification schema must conform (e.g., IEEE LOM).

In our experience, certain complex LOs need to be complemented with different types of domain dependent information for their pedagogical planning and retrieval: (i) classification metadata for enhancing contextualisation, search and retrieval (e.g., the tagged structure of an archaeological site where an archeological object has been found) and (ii) additional data that can enrich the LO (e.g., the weight and other dimensions of an archaeological artifact described in a podcast). We refer to LOs enhanced with domain-dependent information as *Hybrid Learning Objects* (HLOs).

However, most learning object repositories (LORs) only permit a predetermined, fixed set of metadata attributes to be used in the classification of LOs.

This rigidity is inappropriate when domain-dependent information schemas are used for the browsing, retrieval and domain-specific pedagogical sequencing of HLOs.

This custom software applications are needed to manage LOs that must be tagged with information belonging to specific domains.

This paper presents a theoretical approach that permits the use of a single LOR for classifying and enriching LOs according to domain-dependent information schemas, which can be dynamically changed after their definition.

The key issue in our approach is the presence of ameta-relational model for the dynamic definition of specific domain-dependent relational database schemas used for classifying and enriching LOs.

Keywords:

Learning Object Repository, Metadata, Domain-dependent information schema, Classification, LO (Learning Object).

Índice

Índice de figuras

Índice de ilustraciones

Índice de abreviaturas

- LO: Learning Object (Objeto de Aprendizaje)
- HLO: Hybrid Learning Object (Objeto de Aprendizaje Híbrido)
- LOR: Learning Object Repository (Repositorio de Objetos de Aprendizaje)
- BBDD: Base de datos
- DAO: Data Access Object
- JSF: Java Server Faces
- JPA: Java Persistence API
- JAAS: Java Authentication and Authorization
- JPQL: Java Persistence Query Language
- OdA: Objeto de Aprendizaje
- J2EE: Java Platform Enterprise Edition

Capítulo 1. Introducción

1.1 Motivación

El paradigma de los Learning Objects (LOs) ha dominado el panorama teórico del e-learning en los últimos años. El término de Learning Object fue introducido por Wayne Hodgins en 1992. A partir de esa fecha, han sido muchos los autores que han definido el concepto sin obtener un consenso en su definición, llevando el a la utilización de múltiples términos: *learning object, objetos de aprendizaje reutilizables, objeto de conocimiento reutilizable, cápsula de conocimiento*, etc.

David Willey (2001) propone la siguiente definición: "Los objetos de aprendizaje son los elementos de un nuevo tipo de instrucción basada en el computador y fundamentada en el paradigma computacional de 'orientación al objeto'. Se valora sobre todo la creación de componentes (llamados objetos) que pueden ser reutilizados en múltiples contextos. Esta es la idea fundamental que se esconde tras los objetos de aprendizaje: los diseñadores instruccionales pueden construir pequeños componentes de instrucción (en relación con el tamaño de un curso entero) que pueden ser reutilizados varias veces en contextos de estudio diferentes."

La falta de acuerdo en las definiciones básicas adoptadas por diferentes autores evidencia la necesidad de una concordancia de planteamientos y criterios que actualmente está siendo abordada por diferentes organizaciones e iniciativas, entre ellas IEEE, IMS, AICC, CEN/ISSS, que han desarrollado especificaciones y estándares para la descripción común de recursos. Si los objetos de aprendizaje se diseñan y se construyen con base en una metodología, definiendo su ciclo de vida, y se catalogan y almacenan convenientemente, podrán ser aprovechados con facilidad en múltiples contextos, es decir, podrán ser reutilizados.

En este sentido se manifiesta Polsani (2003), que incluye la reutilización en su definición: "Un objeto de aprendizaje es una unidad independiente de aprendizaje en donde su contenido está predispuesto para reutilizarse en contextos instruccionales múltiples"

Este concepto de reutilización de LO introduce un nuevo planteamiento: los repositorios de objetos de aprendizaje (LORs) (Tzikopoulos, 2009). Para facilitar su uso, los LO se almacenan en repositorios de objetos de aprendizaje, que a menudo utilizan un conjunto fijo de metadatos para la clasificación, recuperación y construcción de secuencias pedagógicas de LOs (Neven y Duval, 2002).

Las definiciones, en su sentido general no difieren mucho entre sí, y dejan ver claramente que estos repositorios, sean bases de datos o catálogos, están creados para ser utilizados en un proceso de enseñanza, lo cual lleva a que los LORs se vean como facilitadores claves para incrementar el valor de los recursos de aprendizaje dando la oportunidad a reutilizar, reorientar y hacer reingeniería para cubrir las necesidades del usuario final.

Sin embargo, estos repositorios no son adecuados para la creación de colecciones LO altamente especializadas (Navarro et al., 2013). Este tipo de colecciones de LO normalmente son utilizados en los campus virtuales de la universidad donde profesores y estudiantes trabajan en un dominio de conocimientos y necesidades educativas y de investigación específicos. Estas comunidades usan su propia terminología común para representar sus LOs, con la necesidad de describir sus características específicas, y prefieren definir sus propios esquemas de organización y de navegación para ser compatibles con sus modelos didácticos (Soergel, 2002).

En este contexto, los objetos de aprendizaje deben ser complementados con la información dependiente del dominio, que se puede utilizar de dos maneras:

- Los metadatos de clasificación para mejorar la contextualización, búsqueda y recuperación (por ejemplo, la estructura de etiquetado de un sitio arqueológico donde se ha encontrado un objeto arqueológico).

- Los datos adicionales que puede enriquecer el LO (por ejemplo, el peso y otras dimensiones de un artefacto arqueológico descrito en un *podcast*).

Estos nuevos objetos de aprendizaje los denominamos Objetos de Aprendizaje Híbridos, *Hybrid Learning Objects* (HLO). Esta información dependiente del dominio permite al usuario tanto buscar un LO basado en información no intrínseca y datos, como enriquecer LOs con información intrínseca sobre el tema al que ser refiere. Por lo tanto, para HLOs, utilizamos el término información dependiente del dominio en lugar de datos o metadatos. La información dependiente del dominio se estructura mediante esquemas de información de dependiente del dominio.

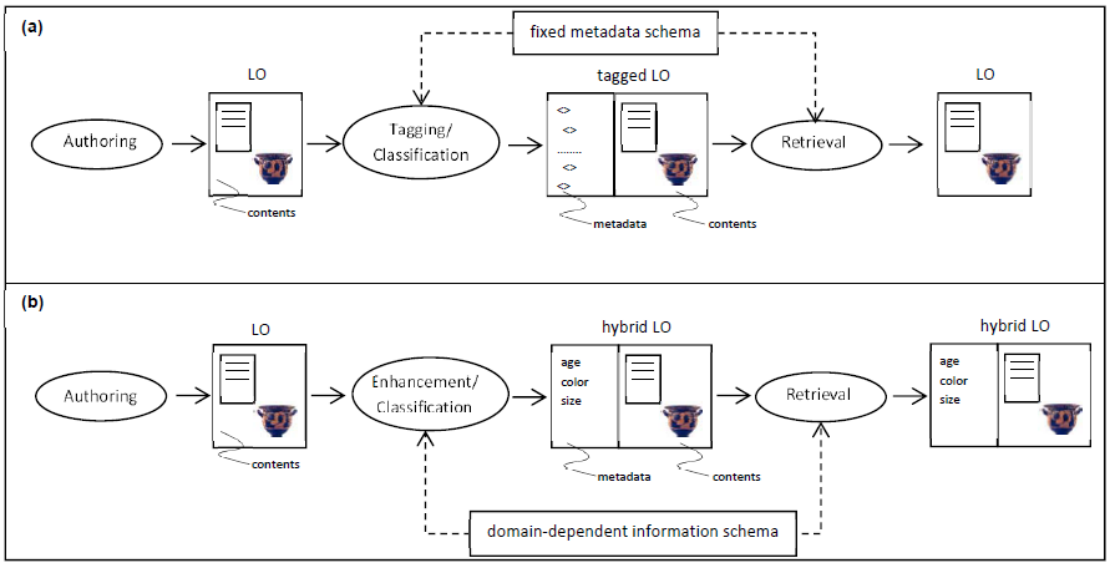


Figura 1.1.1 Diferencias entre LO clásico y un HLO

La figura representa la diferencia entre un LO clásico y un HLO. El LO clásico tiene un proceso de autoría y se marca a continuación, con un conjunto fijo de metadatos que se utilizan para recuperarlo (figura 1.a). Una vez recuperado, no se utilizan estos metadatos. (Esquemas de metadatos fijos). Sin embargo, el proceso de creación de un HLO incluye la definición de un esquema de información dependiente del dominio

utilizado para mejorar el LO con conocimientos y datos contextuales específicos (Figura 1.b). El motor de recuperación tiene en cuenta esta variable de información dependiente del dominio al buscar el HLO. Además, cuando se recupera, la información dependiente del dominio se utiliza como contenido del LO.

Esta situación presenta un inconveniente importante: debido a que los esquemas de información de dependiente del dominio varían según el dominio, repositorios específicos de HLOs han de ser construidos para cada caso. Esto representa una considerable desventaja con respecto a los repositorios de LOs clásicos, donde no se permite un esquema con la información dependiente del dominio, pero si se puede reutilizar el mismo LOR.

Para superar este problema, (Navarro et al., 2013) define un enfoque meta-relacional que permite el uso de un repositorio único para los diferentes dominios. Con este enfoque, un solo repositorio puede ser utilizado para gestionar HLOs pertenecientes a diferentes dominios y enriquecerlos con diferentes esquemas de información de dependiente del dominio. La necesidad de esquemas individuales de información para dominios específicos y el uso de diferentes esquemas de información para el mismo dominio, son temas complejos. Por tanto, no entramos en la definición de uno o varios esquemas de información para la estructuración de un dominio específico, sino en la cuestión de cómo codificar cualquier esquema de información, el cual puede ser utilizado para un dominio específico, en una única herramienta que puede ser reutilizada en diferentes dominios.

Además estos esquemas, una vez definidos, pueden cambiar dinámicamente con el tiempo. Es deseable disponer de un método que permita la modificación dinámica del esquema de información dependiente del dominio.

1.2 Objetivos

Partiendo de la motivación expuesta anteriormente, podemos establecer como objetivo principal el diseño de una herramienta como repositorio de objetos de aprendizaje híbridos (HLOs) que contemple el soporte de metadatos específicos del domino. Para ello, se establecen las siguientes metas:

- Realizar un estudio acerca de los repositorios de objetos de aprendizaje ya existentes, tengan información dependiente del dominio o no. Analizar sus características y requisitos principales para establecer los nuestros propios.
- Definir una arquitectura para nuestro repositorio que permita el manejo de objetos de aprendizaje con información dependiente del dominio estructurada en un esquema de datos relacional.
- Definir un enfoque meta-relacional que permita el uso de un repositorio único para los diferentes dominios.
- Contemplar en nuestra solución la modificación dinámica de los esquemas que contienen la información dependiente del dominio.
- Replicar un caso concreto con un metamodelo de un dominio ya conocido en el repositorio.

- Aplicar los marcos J2EE [1] que determinan el estado del arte para la construcción de aplicaciones empresariales (Fowler et al., 2002).

1.3 Organización de la memoria

El resto del documento se divide en cuatro capítulos más:

- Capítulo 2: En este capítulo se realiza una revisión del estado del arte, el cual fundamenta esta propuesta, señalando los aspectos teóricos y tecnológicos más relevantes. En su primera sección se estudia los principales repositorios de objetos de aprendizaje existentes y sus características diferenciales. Como punto final de la sección, se realiza una comparación conjunta de las herramientas estudiadas y nuestra propia herramienta. En la segunda sección, se introduce las tecnologías de implementación y patrones arquitectónicos utilizados, incidiendo en los frameworks J2EE más relevantes.

- Capítulo 3: Este capítulo profundiza en el diseño de la solución adoptado para el planteamiento anterior. Primero se hace un acercamiento a las necesidades de la herramienta por medio de casos de uso. Se repasa la arquitectura software y sus componentes, así como la especificación del metamodelo, parte clave de nuestro proyecto. A través de diagramas de secuencia, clase y los modelos de objetos y relacional, se revisan todos los detalles del diseño. Finalmente, se realiza un diagrama de despliegue de la aplicación.

- Capítulo 4: A través de un caso de prueba intensivo, se recorre toda la funcionalidad real de nuestra aplicación. En este capítulo se observa cómo se muestra y funciona la aplicación desde el punto de vista de los distintos usuarios, y los resultados que producen sus acciones, mostrando también en paralelo el estado de la BBDD.

- Capítulo 5: En este último capítulo se expone las conclusiones finales del trabajo, poniendo de manifiesto hasta qué punto se han logrado los objetivos, y repasa los conocimientos adquiridos a través de la realización del mismo. Finalmente, se analizan las posibles líneas futuras para la continuidad del proyecto.

Capítulo 2. Estado del arte

Este capítulo recoge el estado del arte del proyecto, centrándose en dos aspectos: los repositorios de objetos de aprendizaje y la tecnología J2EE.

2.1 Repositorio de objetos de aprendizaje

Esta sección analiza algunos de los repositorios de aprendiza más relevantes considerados para realizar el trabajo (Navarro et al., 2013).

- *Ariadne* [2] es una asociación europea que ha desarrollado una arquitectura abierta y escalable basada en normas para la gestión de redes de enlace en repositorios distribuidos (Ariadne, 2011). Actualmente, más de 620.757 HLOs (Objetos de Aprendizaje Híbridos) están contenidos en el repositorio Ariadne (Imagen 2.1.1). Cuando se incluyen los proveedores de GLOBE, este número es de aproximadamente 900.000.

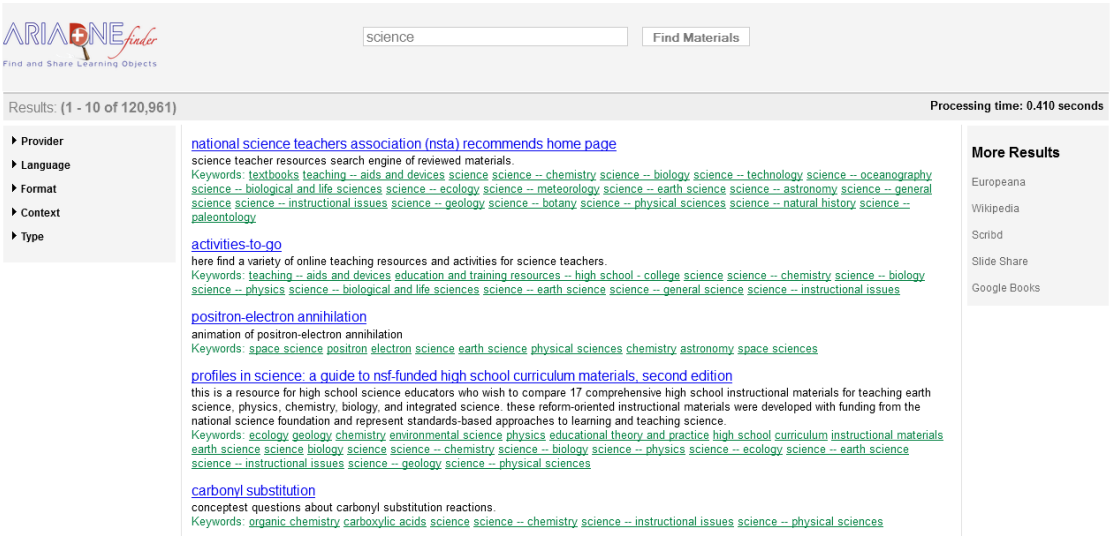


Imagen 2.1.1. Web del repositorio Ariadne

- *MERLOT* (Recursos Educativos Multimedia para el Aprendizaje y Enseñanza en línea) [3] (Merlot, 2011) es un LOR (Repositorios de Objetos de Aprendizaje) internacional muy conocido y reconocido. Almacena los materiales de aprendizaje, ejercicios de aprendizaje, comentarios, colecciones personales y contenido de las páginas Web Builder, diseñados para mejorar la experiencia de uso de materiales de aprendizaje (Cechinel et al., 2010).(Imagen 2.1.2). Actualmente Merlot, un miembro del consorcio GLOBE, contiene más de 30.650 LOs (Objetos de Aprendizaje). Estas características hacen de MERLOT un ejemplo excelente LOR.

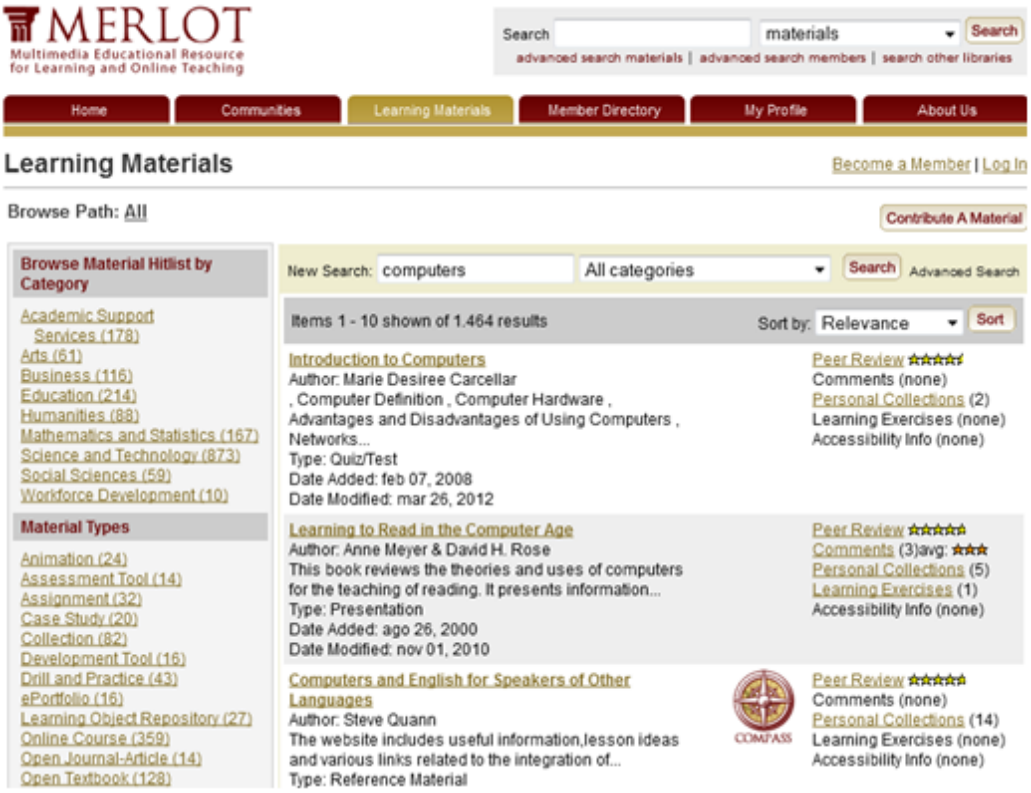


Imagen 2.1.2. Web del repositorio Merlot

- *DOOR* (Digital Open Object Repository) [4] (DOOR, 2011) es una aplicación de código abierto para la creación de LORs. DOOR permite a los usuarios buscar, recuperar e incluir LORs en cursos o unidades de instrucción.(Imagen 2.1.3). DOOR se distribuye bajo licencia GPL.

Imagen 2.1.3. Web del repositorio DOOR

- *DSpace*, un repositorio digital abierto (DSpace, 2011), es, junto con Fedora, una de las mayores aplicaciones de software de código abierto para la gestión y el acceso a los contenidos digitales.(Imagen 2.1.4).Aunque DSpace no es un repositorio de LOs, se ha utilizado con éxito como tal (Waller y Strunz, 2010).

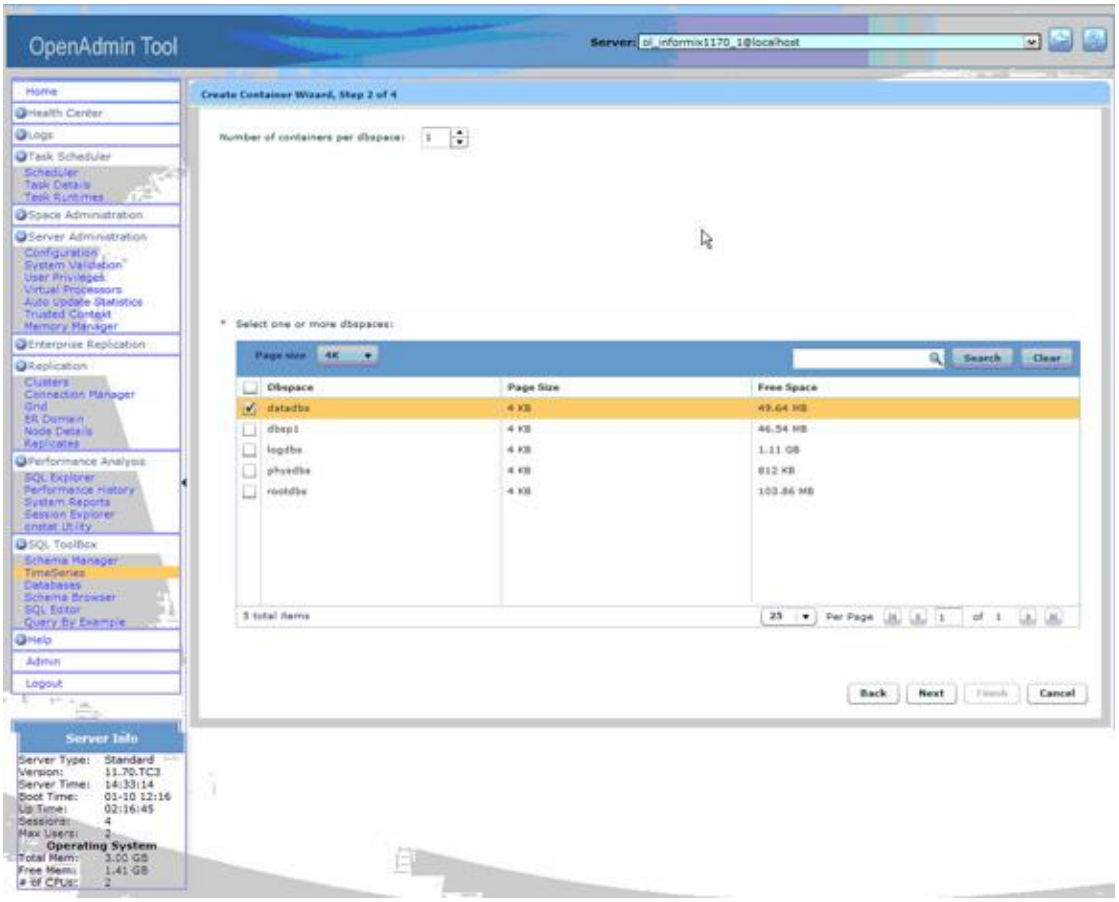


Imagen 2.1.4. Web del repositorio DSpace

- *Fedora* (Flexible Extensible Digital Object and Repository Architecture) [5] fue diseñado originalmente en 2001 en la Universidad de Cornell y la Universidad de Virginia como un proyecto de código abierto (Staples et al., 2003). Fedora tiene una gran comunidad internacional de usuarios y se instala en todo el mundo. Es unsistema de gestión de contenidos digitales en general que apoya la creación de LORs.(Imagen 2.1.5). Fedora proporciona un modelo de objetos digitales complejos y un repositorio XML basado en estándares para gestionar y acceder a ellos (Lagore et al., 2005). El repositorio tiene una arquitectura de software bien definida, y la mayoría de sus servicios se presentan como los servicios web de SOAP (Hansen, 2007).

Imagen 2.1.5. Web del repositorio Fedora

Imagen 2.1.6. Fedora Digital Object

El LOR propuesto en este trabajo, OdA J2EEE, es una aplicación de base de datos en línea diseñado para la definición y gestión de HLOS. OdA proporciona a los profesores, investigadores y estudiantes una herramienta sencilla y flexible para difundir sus materiales educativos y de investigación.

Su objetivo principal es tratar la dinámica de definición de esquemas de información de dominio dependientes utilizadas para enriquecer LOs tradicionales y por lo tanto transformarlos en HLOS.

Los LORs mencionados en esta memoria son herramientas poderosas para el manejo de LOs tradicionales, pero, a excepción de OdA, se han encontrado que son inapropiados para la definición dinámica de esquemas de información de dominio específico para la navegación y enriquecimiento de los LOs. No son compatibles con la definición de estos esquemas, y la mayoría de estas herramientas utilizan los esquemas de metadatos estándar (por ejemplo, IMS LOM) que no pueden ser redefinidos por el usuario. Fedora es la única excepción, ya que permite la definición de esquemas de información de dominio específico. Sin embargo, a diferencia de OdA, no permite la modificación dinámica de estos esquemas.

La Tabla 1, compara OdAcon las aplicaciones mencionadas anteriormente, basados en cinco características:

(I). Esquema de metadatos fijo. Presencia de un esquema de metadatos fijo para la clasificación de los LOs.

(II). Esquema de información de dependiente del dominio. Capacidad para definir dinámicamente esquemas de información dependiente del dominio, tanto para la navegación como para el enriquecimiento de los LOs.

(III). Redefinición dinámica del esquema de información. Capacidad para la redefinición de un dominio dependiente de esquema de información.

(IV). Los objetos compuestos. La presencia de LOs complejos, formado por la agregación de otros LOs definidos en el LOR.

(V). Import / Export. Capacidades de importación / exportación de acuerdo con una norma reconocida (por ejemplo, IMS CP).

(VI). Integración de LMS. Los usuarios pueden buscar directamente la colección LO. En algunos casos, los usuarios pueden importar el LO en el LMS y publicarlo.

La tabla muestra que los LORs más utilizados, Merlot y Ariadne, se basan en un sencillo y fijo modelo de LO. Por consiguiente, su capacidad para representar y gestionar HLOS altamente especializados es compleja y limitada. Fedora utiliza estándares propietarios, mientras que Oda puede utilizar cualquier modelo de metadatos, cuando se define como un esquema de información depende del dominio.

Entre las aplicaciones de software que se utilizan para crear LORs, Fedora y OdA son las únicas herramientas que pueden definir esquemas de clasificación depende del dominio de la gestión de HLOS. Sin embargo, en Fedora, el modelo de LOno se puede cambiar dinámicamente una vez que el repositorio ha sido creado. Además, Fedora requiere una cantidad sustancial de instalación y soporte de mantenimiento, ya que está diseñado para crear grandes repositorios de objetos digitales institucionales.

Por el contrario, Oda ha sido diseñado para la creación dinámica y mantenimiento de colecciones HLO. Oda puede hacer frente a los diferentes esquemas de información dependientes del dominio, lo que permite su redefinición dinámica como sea necesario. Por lo tanto, el mismo repositorio puede manejar diferentes LOs que pertenecen a diferentes dominios.

La tabla 2.1.1 compara las características de estos LORs.

Repositorio	(i) Fijo	(ii) Dominio	(iii) Dinámico	(iv) Compuesto	(v) Importar/ Exportar	(vi) LMS
Ariadne	✓	×	×	×	✓	✓
Merlot	✓	×	×	×	✓	✓
DOOR	✓	×	×	×	✓	✓
Dspace	✓	×	×	✓	✓	×
Fedora	✓/×	✓	×	✓	✓/×	×
OdA	✓/×	✓	✓	✓	✓/×	×

(i) Esquema de metadatosfijo.
(ii) Información de esquema de dominio dependiente.
(iii) Redefinición dinámica de un esquema de información.
(iv) Objetos compuestos.
(v) Importar / Exportar. Estas capacidades acordes con algún estándar (ej. IMS CP).
(vi) Integración LMS.

Tabla 2.1.1. Comparación de Repositorios de LO y sistemas de gestión de contenidos digitales

A pesar de que está contemplada la capacidad de importación y exportación en OdA, no ha sido posible incluirla en la aplicación desarrollada en este trabajo.

Nótese que sólo DSpace, Fedora y OdApermiten la presencia de LOs compuesto formados por la agregación de otros LOs. Éste es un tema clave de OdA porque los HLOs a menudo referencian a otros HLOs que se han desplegado.

2.2 Tecnologías J2EE

Hemos creado un buen sistema de información desde el punto de vista arquitectónico, aplicando los principios de la arquitectura multicapa y los patrones de diseño. El sistema es capaz de:

- Manejar una gran cantidad de datos persistentes
- Acceder a datos concurrentemente
- Representar la funcionalidad de la aplicación mediante una gran cantidad de lógica del negocio
- Acceder a datos a través de interfaces de usuario

También se han utilizado los frameworks JSF, JPA y JAAS.

A continuación se detallarán las tecnologías utilizadas para el proyecto.

2.2.1 Arquitectura Multicapa

A pesar de no ser una tecnología J2EE, sí que es cierto que el principal catálogo de patrones de arquitectura multicapa (Alur et al., 2003) sí que está definido para J2EE.

La arquitectura multicapa considera una capa de presentación, otra de negocio, y otra de integración.

La *capa de presentación* encapsula toda la lógica de presentación, necesaria para dar servicio a los clientes que acceden al sistema.

La *capa de negocio* proporciona los servicios del sistema.

La *capa de integración* es responsable de la comunicación con recursos y sistemas externos.

Además, por encima y deba de estas capas están las capas de usuarios del sistema, y de recursos del mismo.

Aunque no estrictamente necesarios, sí que hay una serie de patrones extraídos del catálogo de patrones de Gamma et al. [6] que son útiles a la hora de aplicar los patrones de arquitectura multicapa (Navarro et al., 2011):

- El patrón *fachada* proporciona una interfaz unificada para un conjunto de interfaces de un subsistema.

- Oculta a los clientes los componentes del subsistema, reduciendo así el número de objetos con los que tratan los clientes. De esta forma el subsistema es más fácil de utilizar.

- El patrón *factoría abstracta* proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas. Desliga la creación de nuevos objetos de la clase que se refiere a dichos objetos a través de la interfaz que implementan.

- El patrón *singleton* garantiza que sólo hay una instancia de una clase, proporcionando un único punto de acceso a ella. Esta instancia podría ser redefinida mediante herencia. Los clientes deberían ser capaces de utilizar estas subclases sin modificar su código.

El catálogo de Alur et al. contiene veintiún patrones de arquitectura multicapa. De ellos, los utilizados en este proyecto son:

- Controlador frontal y de aplicación: También conocido como *Front Controller*. Proporciona un punto de acceso para el manejo de las peticiones de la capa de presentación para evitar lógica de control duplicada, aplicar una lógica común a distintas peticiones, separar la lógica del procesamiento del sistema de la vista y/o tener puntos de acceso centralizados y controlados del sistema.

- Ayudante de vista: También conocido como *ViewHelper*. Encapsula la lógica de acceso a datos en beneficio de los componentes de la presentación. Por ejemplo, los JavaBeans pueden ser usados como patrón *ViewHelper* para las páginas JSP. Varios clientes, como controladores y vistas, podrían utilizar el mismo Helper para recuperar y adaptar estados del modelo similares para su presentación en varias vistas, eliminando la redundancia en el código.

- Servicio de aplicación: También conocido como *Application Service*. Centraliza la lógica de negocio y se aplica cuando se quiera representar una lógica del negocio que actúe sobre distintos servicios u objetos del negocio, agrupar funcionalidades relacionadas y/o encapsular lógica no representada por objetos del negocio.

- Objeto del negocio: También conocido como *Business Object*. Sirve para representar la lógica del negocio y el modelo del dominio en términos orientados a objetos.

- Transferencia: También conocido como *Transfer*. Independiza el intercambio de datos entre capas. Se aplica cuando no se desea conocer la representación interna de una entidad dentro de una capa. Se aplica cuando se disponga de un modelo conceptual con reglas de validación y lógica del negocio avanzadas, se desee separar la lógica del negocio del resto de la aplicación, se desee centralizar la lógica del negocio y/o se desee incrementar la reusabilidad del código.

- DAO: También conocido como *Data Access Object*. Permite acceder a la capa de datos (recursos, en general), proporcionando representaciones orientadas a objetos (e.g. objetos transferencia) a sus clientes. Se aplica cuando se quiere independizar la representación y el acceso a los datos de su procesamiento.

- Almacén del dominio: También conocido como *Domain Store*. Separa la persistencia del modelo de objetos. Se aplica cuando se deseen omitir detalles de persistencia en los objetos del negocio, no se desee utilizar EJBs de entidad, la aplicación podría ejecutarse en un contenedor web y/o el modelo de objetos utiliza herencia y relaciones complejas.

2.2.2 JSF (Java Server Faces)

Java Server Faces es un *framework* para aplicaciones java J2EE con arquitectura multicapa y basadas en web, que simplifica el desarrollo de interfaces de usuario.

Las características principales de JSF son:

- Está basado en componentes y eventos del lado del servidor. Mantiene del lado del servidor una representación del interfaz de usuario presentado en el cliente.

- Está orientado a componentes y objetos.

- Es extensible, y con arquitectura de rending.

- Se puede modificar el comportamiento del framework mediante APIs que controlan su funcionamiento (PrimeFaces, MyFaces...).

- JSF forma parte del estándar J2EE, mientras que otras tecnologías para creación de vistas de las aplicaciones no lo forman, como por ejemplo Struts.

Elementos:

La principal función del frameworkJSF es asociar a las pantallas, clases Java que recogen la información introducida y que disponen de métodos que responden a las acciones del usuario. Para ello, las aplicaciones JSF están formadas por los siguientes elementos principales:

Vistas:

Conjunto de ficheros JSP con las tag libraries de JSF, Facelets (xhtml) y otros PDL (*Page Declaration Language*). A través de la vista se describe la jerarquía de componentes JSF que conforman cada una de las páginas del interfaz del usuario. Estos componentes quedan vinculados con sus controladores en java: los *managed beans*.

Managed Beans:

Los managed beans son las clases java que están detrás de los formularios JSF, empleando EL (*Expression Language*). Una página JSF lee los valores de las propiedades del bean que tiene asociada, y cuando se hace un post de un formulario, se guardan sus valores en el bean. Además de poder lanzar acciones. Estos beans se referencian en el fichero de configuración de JSF.

Cada managed bean debe ir definido con su ciclo de vida:

- *Application Scoped*: Objetos disponibles para todas las peticiones de cualquier usuario en todas las vistas de la aplicación.
- *Session Scoped*: Objetos disponibles para todas las peticiones que formen parte de la misma sesión de un cliente.
- *View Scoped*: Objetos disponibles para todas las peticiones que se realicen sobre la misma vista (página JSF).
- *Request Scoped*: Objetos disponibles desde que se recibe la petición del cliente hasta que la respuesta es enviada.

A continuación se describen algunos objetos de especial relevancia en el marco.

Controlador:

El controlador de JSF se denomina *Faces Servlet* y queda configurado por el fichero faces-config.xml. Todas las peticiones HTTP del usuario pasan por él. *Faces Servlet* examina las peticiones recibidas, actualiza la representación del interfaz del cliente y los datos de los *Managed Beans*, e invoca los manejadores de eventos.

Renderizadores:

Es el objeto responsable de generar la representación de los componentes JSF a mostrar en los clientes y de recuperar las entradas de usuario recibidas para actualizar los valores vinculados a los componentes.

Convertidores:

Tienen dos principales funciones principales:

1. Al generar la presentación, transforman de forma conveniente los valores de los objetos Java vinculados a los componentes JSF en Strings.
2. Al recuperar las entradas de usuario, transforman los Strings enviados por el cliente en los objetos Java vinculados a los componentes JSF según el tipo que corresponda.

Validadores:

Son los objetos responsables de comprobar que los valores recibidos y almacenados en los componentes JSF cumplen las restricciones especificadas. Esta comprobación se realiza antes de actualizar los atributos de los *Managed Beans*, y queda del lado del servidor.

Navegación:

Cuando se ejecuta una petición que incluye una acción se ejecuta el mecanismo de navegación de JSF. Tras la ejecución de la acción, el controlador determina cómo se debe mostrar al usuario el resultado de la petición.

Este mecanismo de navegación se implementa de manera sencilla en la página JSF. Cuando el controlador JSF llama al método asociado a la acción, este devuelve un valor de tipo String. Este valor es utilizado junto con las reglas de navegación creadas en el fichero de configuración de JSF para determinar la página que se debe enviar como respuesta al usuario.

Las reglas de navegación definen la página de origen, la etiqueta de destino, la página de destino asociada a cada etiqueta, y si el envío es directo o una redirección externa.

Proceso de una petición:

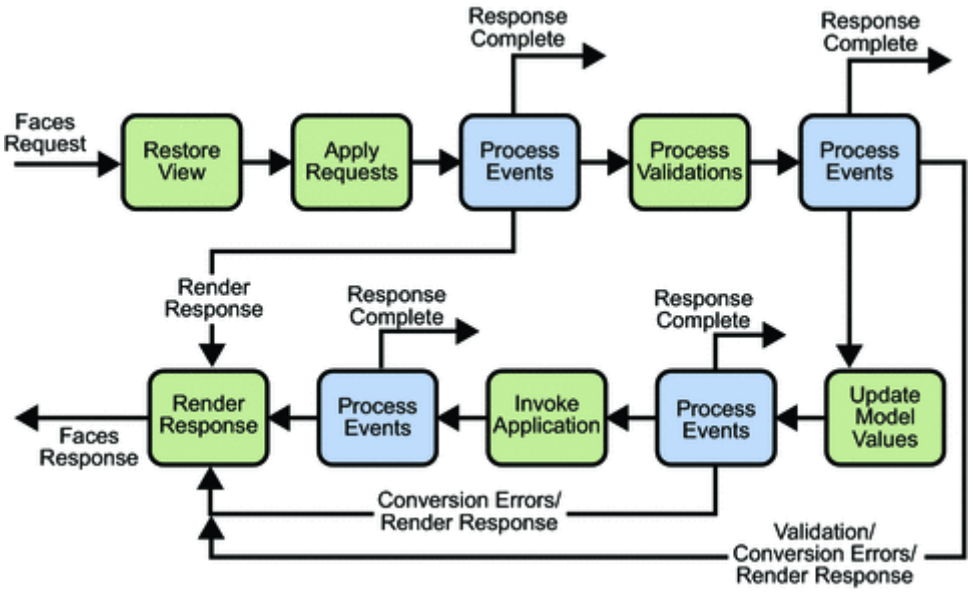


Figura 2.2.2.1 Esquema del recorrido de una petición para JSF [6]

Durante el procesamiento de una petición el controlador JSF realiza las siguientes etapas:

1. Restaurar los componentes de la vista *(restore view*). En esta etapa el controlador construye en memoria la estructura de componentes de la página.
2. Aplicar los valores de la petición (*apply request values*). En esta etapa se recuperan los valores de la *request* y se asignan a los beans de la página.
3. Procesamiento de las validaciones (*process validations*). Se verifican los parámetros de entrada según un conjunto de reglas definidas en un fichero de configuración.
4. Actualizar los valores del modelo (*update model values*). Los valores leídos y validados son cargados en los *beans*.
5. Invocación a la aplicación (*invoke application*). Se ejecutan las acciones y eventos solicitados para la página. Si es necesario se realiza la navegación.
6. Generación de la página (*render response*). En esta fase se genera la página que será enviada al usuario con todos sus elementos y valores actualizados.

2.2.3 JPA (Java Persistence API)

Implementa al marco de almacén del dominio y a los objetos del dominio. Éstos últimos proporcionan una orientación OO plena.

La manipulación de objetos persistentes desde la capa de negocio requiere la utilización de un API que permite al programador realizar las operaciones típicas de creación, actualización, eliminación y recuperación de objetos.

Entidades

Una entidad es un objeto del negocio persistente. Sus instancias tienen un identificador único, son transaccionales con respecto al almacén persistente, no son transaccionales en memoria y son equivalentes a clases.

Las entidades tienen metadatos asociados que describen su relación entre la representación en memoria y en el almacén persistente. Éstas se pueden definir mediante anotaciones o ficheros XML.

Entity Manager

Los entity manager gestionan contextos de persistencia, que es un conjunto de instancias de entidad, y por ende, las instancias allí contenidas. Cada contexto de persistencia está asociado con una unidad de persistencia, restringiendo las instancias a las clases de la unidad.

Operaciones fundamentales que proporcionan los entity managers:

- Persist() Acepta una nueva instancia de entidad y la gestiona, si no lo estaba. Está pensado para instancias nuevas que no existen en la base de datos.
- Find() Localiza instancias de entidades en base a su clave primaria. Devuelve una instancia gestionada si se invoca en el contexto de una transacción y una instancia desvinculada si se invoca fuera de una transacción.
- Remove() Elimina una instancia de entidad de la base de datos, una vez hecho commit. Se puede invocar fuera de una transacción, pero no surte efecto hasta que una transacción del entity manager se compromete.
- Clear y detach() Clear limpia el entity manager, dejando a las entidades que gestionaba desvinculadas de la base de datos. Detach desvincula selectivamente una entidad del entity manager.
- Flush() Actualiza la base de datos con las entidades creads, modificadas o eliminadas en el entity manager.
- Merge() Fusiona una instancia de entidad desvinculada con su entity manager. Ésta ha podido producirse por cierre del contexto de persistencia, invocación del método clear() o detach(), invocación de un rollback() o serialización de la instancia de la entidad.
- Refresh() Refresca una entidad gestionada, acutalizando el estado en memotia con el actual en la base de datos.

Concurrencia

Una entidad gestionada pertenece a un único contexto de persistencia. Por defecto JPA funciona con un nivel de aislamiento read commited. Así, los cambios comprometidos por una transacción pueden ser vistos durante la ejecución de una transacción paralela.

JPA implementa fácilmente el bloqueo optimista. Hay dos tipos de bloqueos optimistas:

- Optimista: sólo aumenta el número de versión si hay modificaciones en el objeto.
- De incremento forzado: aumenta el número de versión al hacer la lectura, sin necesidad de modificar el objeto.

Java PersistenceQueryLanguage (JPQL)

A través del lenguaje Java *PersistenceQueryLanguage* (JPQL) es posible generar consultas complejas para extraer instancias de una entidad.

2.2.4 JAAS

JAAS son las siglas de *Java Authentication and Authorization* [7] Se trata de un framework de seguridad para aplicaciones java cuya finalidad es la de definir un estándar para los procesos de autentificación y autorización. Permite separar el sistema de autenticación de la aplicación J2EE.

Características:

Supongamos que un usuario desea controlar el acceso a unos recursos. Normalmente ante esta situación se pasa por dos fases: autenticación y autorización.

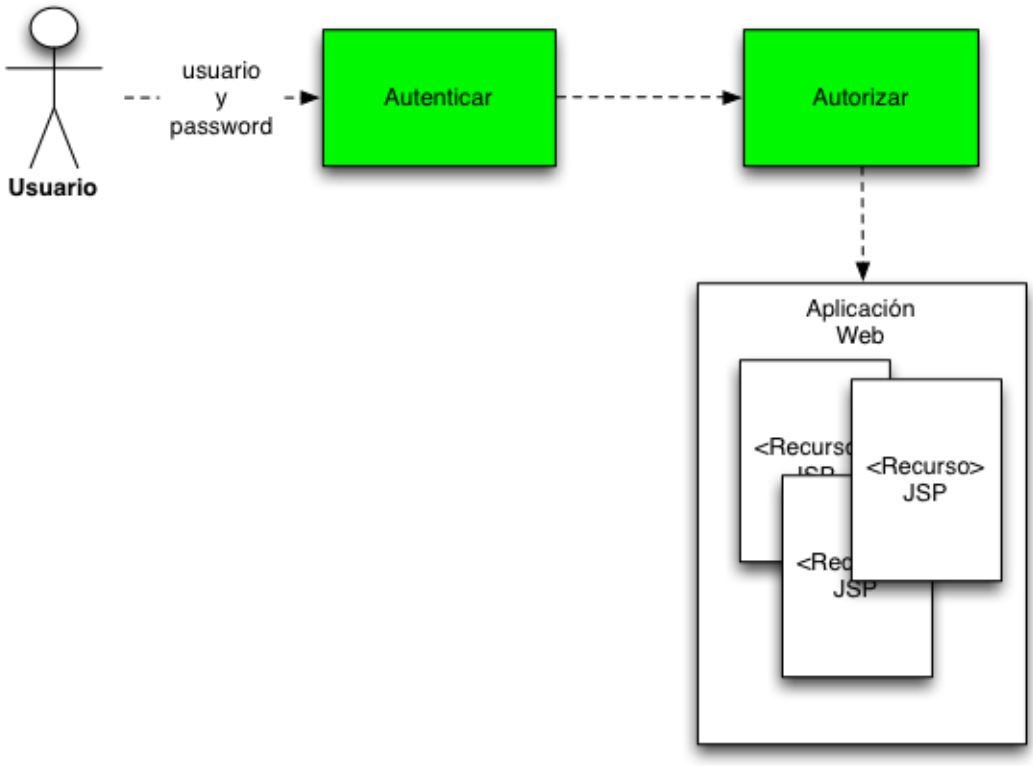


Figura 2.2.4.1 Autentificación y autorización típica de un usuario

1. Autenticación: El usuario envía al servidor su nombre y su contraseña y el servidor comprueba contra algún tipo de repositorio si el usuario es válido o no.

2. Autorización: Si el usuario ha sido validado pasamos a la fase de autorización en la que se comprueba que el usuario tiene permisos para acceder a estos recursos.

Sin embargo, con JAAS la fase de autorización está sujeta a la gestión de roles y no a usuarios. No tenemos que asignar permisos a los recursos por cada usuario individualmente, sino que se realiza un mapeo entre usuarios y roles, donde un conjunto de usuarios comparten un mismo rol.

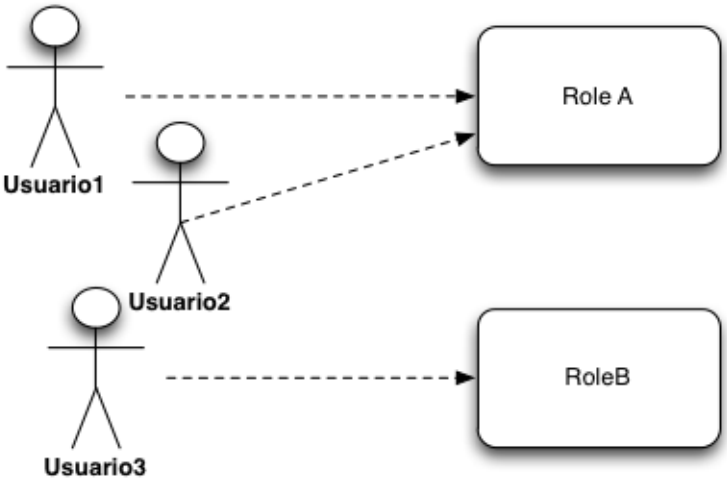


Figura 2.2.4.2 Usuarios asignados a roles

Una vez definidos los roles y usuarios, JAAS determinará cuáles son los recursos a los que cada usuario, a través de su rol o roles, puede acceder.

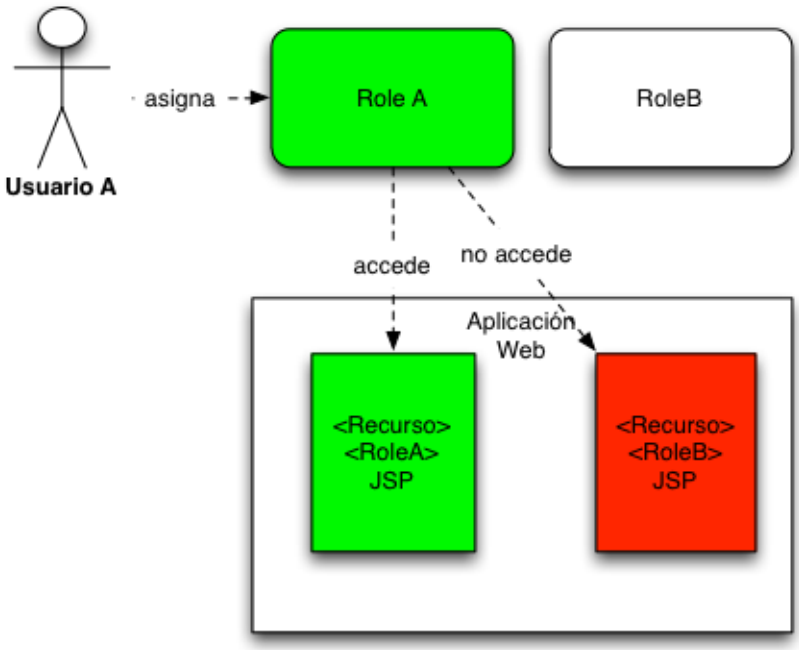


Figura 2.2.4.3 Usuario asignado a un rol que controla el acceso a los recursos

Funcionamiento:

Mediante un archivo de configuración, determinaremos los módulos de autenticación a utilizar y su forma de uso. Se puede optar por módulos de Login ya implementados (*Solaris Login Module, NT Login Module, JNDI Login Module)*, o definir el nuestro propio. Estos módulos deben implementar los siguientes métodos: *Initialize*, *login*, *commit*, *abort* y *logout*.

Para la autenticación, disponemos de varios tipos según se especifique en el descriptor de despliegue:

1. BASIC: Autenticación básica de usuario/contraseña. Utiliza esquema de autenticación en la cabecera HTTP.

2. DIGEST: Similar a la anterior, pero con la contraseña cifrada.

3. CLIENT-CERT: Maneja certificados.

4. FORM: Basada en formulario web.

La autenticación se realiza mediante *realms*: Repositorios de usuarios y roles para validar los usuarios de una aplicación y controlados por la misma política de autenticación. Existen diferentes *realms*, como *JDBC Realm, DtaSourceRealm, JNDI Realm, MemoryRealm y JAAS Realm.*

Una vez definido el *realm*, debemos declararlo en nuestro servidor y contexto.

Las restricciones de acceso se realizan de manera declarativa y programática. Para ello deberemos modificar el fichero web.xml y añadirle nuestras etiquetas de seguridad:

- <login-config> Define el mecanismo de autenticación.
- <auth-metod> Especifica tipo de autenticación.
- <login-config> Mecanismo de autenticación.
- <form-login-config> Define los formularios webs para la autenticación.
- <security-constraint> Define los privilegios de acceso a una colección de recursos.
- <web-resource-collection> Define los componentes de la aplicación Web que aplican para este constraint.
- <web-resource-name> El nombre de la colección de recursos web.
- <url-pattern>. Describe un patrón para identificar los elementos que aplican al constraint de seguridad.
- <Auth-constraint> Define que roles tienen acceso a esta colección de recursos Web definidos en el constraint.
- <rol-name> Debe corresponder con los roles definidos en el descriptor.
- <security-role> Define un tipo de rol de usuario.
- <rol-name> Define el nombre del rol.
- <description> Descripción del tipo de rol.

Capítulo 3. Diseño de la solución

Este capítulo describe el diseño de la solución empezando por los requisitos del mismo caracterizado como casos de uso descritos en lenguaje natural estructurado, tal y como aparecerían en un documento de especificación de requisitos redactado según el IEEE Std. 830-1998 [8]. Después se describe la arquitectura software que ha guiado la solución propuesta. También se incluye el metamodelo utilizado para soportar el modelo de datos de la aplicación. A continuación se proporciona algunos elementos clave del diseño. Finalmente se muestra el despliegue de la solución.

3.1 Casos de uso

Definición de los actores

La aplicación cuenta con tres actores básicos: administrador, usuario registrado y usuario no registrado.

ACT-01	Administrador
Descripción	Se encarga de la gestión de metadatos, generación de learningobjects y la gestión de usuarios
Núm. Usuarios	Uno

Actor ACT-01: Administrador

ACT-02	Usuario registrado
Descripción	Encargado de gestionar las instancias de metadatos, generar las instancias de HLO, gestionar los Recursos Externos, las referencias de los HLOs con otros HLOs y vincular y desvincular HLO con Recursos Externos
Núm. Usuarios	Tantos como hayan sido dados de alta por admin

Actor ACT-02: Usuario registrado

ACT-03	Usuario no registrado
Descripción	Puede consultar las instancias de HLO. Serán aquellos actores que accedan a la aplicación sin necesidad de registro
Núm. Usuarios	Tantos como accedan a la aplicación

Actor ACT-03: Usuario no registrado

Casos de uso

Los requisitos de usuario de la aplicación, brevemente descritos son:

GF-1: Gestión de metadatatos

- GF-1.1: Esquemas relacionales
  - RF-1.1.1 Crear esquema relacional
  - RF-1.1.2 Mostrar esquema relacional
  - RF-1.1.3 Eliminar esquema relacional
- GF-1.2: Tablas
  - RF-1.2.1 Crear tabla
  - RF-1.2.2 Mostrar tabla
  - RF-1.2.3 Modificar tabla
  - RF-1.2.4 Eliminar tabla
- GF-1.3: Columnas pertenecientes a las tablas
  - RF-1.3.1 Crear columna
  - RF-1.3.2 Mostrar columna
  - RF-1.3.3 Modificar columna
  - RF-1.3.4 Eliminar columna
- GF-1.4: Claves primarias de las tablas
  - RF-1.4.1 Crear clave primaria
  - RF-1.4.2 Mostrar clave primaria
  - R.F. 1.4.3 Modificar clave primaria
  - RF-1.4.4 Eliminar clave primaria
- GF-1.5: Claves extranjeras de las tablas
  - RF-1.5.1 Crear clave extranjera
  - RF-1.2.2 Mostrar clave extranjera
  - RF-1.2.3 Modificar clave extranjera
  - R.F. 1.5.4 Eliminar clave extranjera

GF-2: Gestión de instancias de metadatos

- RF-2.1 Crear instancia de metadatos

RF-2.2 Mostrar instancia de metadatos

RF-2.3 Modificar instancia de metadatos

RF-2.4 Eliminar instancia de metadatos

GF-3: Gestión de estructura de HLO

RF-3.1 Crear estructura de HLO

RF-3.2 Mostrar estructura de HLO

RF-3.3 Eliminar estructura de HLO

GF-4: Gestión de contenido de los HLO

RF-4.1 Generar instancias de HLO

GF-4.1: Gestión de recursos (imágenes, texto, audio, etc.) de los HLO

RF-4.1.1 Subir recurso externo

RF-4.1.2 Mostrar recurso externo

RF-4.1.3 Modificar recurso externo

RF-4.1.4 Eliminar recurso externo

GF-4.2: Gestión de referencias a otros HLO

RF-4.2.1 Crear referencia

RF-4.2.2 Mostrar referencia

RF-4.2.3 Eliminar referencia

GF-4.3: Gestión de vínculos de HLO con sus recursos externos

RF-4.3.1 Vincular

RF-4.3.2 Desvincular

GF-5: Gestión de usuarios

RF-5.1 Crear usuario

RF-5.2 Mostrar usuario

RF-5.3 Modificar usuario

RF-5.4 Eliminar usuario

GF-1: Gestión de metadatatos

GF-1.1: Esquemas relacionales

RF-1.1.1 Crear esquema relacional	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de alta un esquema relacional en la BBDD
Entrada	Nombre del esquema relacional
Salida	ID del esquema relacional
Origen	Teclado
Destino	BBDD schema
Necesita	BBDD schema
Acción	Dar de alta un esquema relacional
Precondición	No exista un esquema relacional con el mismo nombre en la BBDD
Postcondición	Esquema relacional creado y almacenado en la BBDD schema
Efectos Laterales	N/a

RF-1.1.2 Mostrar esquema relacional	
Prioridad	Alta
Estabilidad	Alta
Descripción	Muestra un esquema relacional
Entrada	ID del esquema relacional
Salida	Nombre del esquema relacional
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD schema
Acción	Mostrar un esquema relacional
Precondición	El esquema relacional existe en la BBDD schema
Postcondición	Nombre del esquema relacional mostrado mediante interfaz gráfica

Efectos Laterales	N/a

RF-1.1.3 Eliminar esquema relacional	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de baja un esquema relacional en la BBDD
Entrada	ID del esquema relacional
Salida	
Origen	Teclado
Destino	BBDD esquema relacional
Necesita	BBDD esquema relacional
Acción	Dar de baja un esquema relacional
Precondición	El esquema relacional exista en la BBDD schema
Postcondición	Esquema relacional borrado lógicamente de la BBDD schema
Efectos Laterales	Al borrar un esquema relacional, se eliminará en cascada todas las entidades que contenga y que dependan de él

GF-1.2: Tablas

RF-1.2.1 Crear tabla	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de alta una tabla en la BBDD
Entrada	Nombre de la tabla e ID del esquema relacional al que pertenece
Salida	ID de la tabla
Origen	Teclado
Destino	BBDD table
Necesita	BBDD table
Acción	Dar de alta una tabla
Precondición	El esquema relacional existe en BBDD y no existe una tabla en el mismo esquema relacional con el mismo nombre
Postcondición	Tabla creada y almacenada en la BBDD table
Efectos Laterales	N/a

RF-1.2.2 Mostrar tabla	
Prioridad	Alta
Estabilidad	Alta
Descripción	Muestra una tabla
Entrada	ID del esquema relacional al que pertenece e ID de la tabla
Salida	Nombre de la tabla
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD table y schema
Acción	Mostrar una tabla
Precondición	La tabla y el esquema relacional existen en la BBDD
Postcondición	Nombre de la tabla mostrado mediante interfaz gráfica
Efectos Laterales	N/a

RF-1.2.3 Modificar tabla	
Prioridad	Alta
Estabilidad	Alta
Descripción	Modifica el nombre de una tabla
Entrada	ID de la tabla e ID del esquema relacional al que pertenece
Salida	Notificación mediante interfaz gráfica de la información actualizada
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD table y schema
Acción	Modificar el nombre de una tabla
Precondición	La tabla y el esquema relacional existen en la BBDD
Postcondición	Nombre de la tabla modificado y almacenado en la BBDD table
Efectos Laterales	N/a

RF-1.2.4 Eliminar tabla	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de baja una tabla en la BBDD
Entrada	ID de la tabla e ID del esquema relacional al que pertenece
Salida	
Origen	Teclado
Destino	BBDD table
Necesita	BBDD table y schema
Acción	Dar de baja una tabla
Precondición	La tabla y el esquema relacional existan en la BBDD
Postcondición	Tabla borrada lógicamente de la BBDD table
Efectos Laterales	

GF-1.3: Columnas

RF-1.3.1 Crear columna	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de alta una columna en la BBDD
Entrada	Nombre de la columna, tipo de datos, longitud, ID de la tabla e ID del esquema relacional al que pertenece
Salida	ID de la columna
Origen	Teclado
Destino	BBDD column
Necesita	BBDD column, table y schema
Acción	Dar de alta una columna
Precondición	No exista una columna en la misma tabla con el mismo nombre en la BBDD
Postcondición	Columna creada y almacenada en la BBDD columna
Efectos Laterales	N/a

RF-1.3.2 Mostrar columna	
Prioridad	Alta
Estabilidad	Alta
Descripción	Muestra una columna
Entrada	ID de la tabla a la que pertenece e ID de la columna
Salida	Nombre de la columna, tipo y longitud
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD column, table y schema
Acción	Mostrar una columna
Precondición	La columna, la tabla y el esquema relacional existen en la BBDD
Postcondición	Nombre y tipo de la columna mostrado mediante interfaz gráfica
Efectos Laterales	N/a

RF-1.3.3 Modificar columna	
Prioridad	Alta
Estabilidad	Alta
Descripción	Modifica el nombre de una columna
Entrada	ID de la columna e ID de la tabla a la que pertenece
Salida	Notificación mediante interfaz gráfica de la información actualizada
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD columna, tabla y esquema relacional
Acción	Modificar el nombre de una columna
Precondición	La columna, la tabla y el esquema relacional existen en la BBDD
Postcondición	Nombre de la columna modificado y almacenado en la BBDD columna
Efectos Laterales	N/a

RF-1.3.4 Eliminar columna	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de baja una columna en la BBDD
Entrada	ID de la columna e ID de la tabla a la que pertenece
Salida	
Origen	Teclado
Destino	BBDD column
Necesita	BBDD columna,tabla y esquema relacional
Acción	Dar de baja una columna
Precondición	La columna, la tabla y el esquema relaciona lexistan en la BBDD
Postcondición	Columna borrada lógicamente de la BBDD column
Efectos Laterales	N/a

GF-1.4: Claves primarias de las tablas

RF-1.4.1 Crear clave primaria	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de alta una clave primaria en la BBDD
Entrada	Nombre de la clave primaria, ID de la tabla e ID del esquema relacional al que pertenece e IDs de las columnas que formarán la clave primaria
Salida	ID de la clave primaria
Origen	Teclado
Destino	BBDD key
Necesita	BBDD key, column, table y schema
Acción	Dar de alta una clave primaria
Precondición	No exista una clave primaria en la misma tabla con el mismo nombre en la BBDD. Las columnas de la clave primaria pertenecen a la misma tabla
Postcondición	Clave primaria creada y almacenada en la BBDD key
Efectos Laterales	

RF-1.4.2 Mostrar clave primaria	
Prioridad	Alta
Estabilidad	Alta
Descripción	Mostrar las columnas, nombre, tabla y esquema relacional al que pertenece una clave primaria
Entrada	ID de la clave primaria
Salida	Columnas, nombre, tabla y esquema relacional al que pertenece una clave primaria
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD key, column, table y schema
Acción	Muestra información de una clave primaria

Precondición	La clave primaria, la columna, la tabla y el esquema relacional existen en la BBDD
Postcondición	Información de clave primaria mostrado mediante interfaz gráfica
Efectos Laterales	N/a

R.F. 1.4.3 Modificar clave primaria	
Prioridad	Alta
Estabilidad	Alta
Descripción	Modificar el nombre y/o las columnas que forman la clave primaria
Entrada	ID de la clave primaria, nombre e IDs de las columnas que forman parte de la clave primaria
Salida	Notificación mediante interfaz gráfica de la información actualizada
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD key, column, table y schema
Acción	Modifica el nombre de una clave primaria y/o las columnas que la componen
Precondición	La clave primaria, columnas que lo componen, la tabla y el esquema relacional existen en la BBDD. Las columnas pertenecen a la misma tabla. No existe una clave primaria con el mismo nombre en la tabla
Postcondición	Nombre de la clave primaria y/o columnas que la componen modificados y almacenados en la BBDD key
Efectos Laterales	N/a

RF-1.4.4 Eliminar clave primaria	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de baja una clave primaria en la BBDD
Entrada	ID de la clave primaria e ID de la tabla a la que pertenece
Salida	
Origen	Teclado
Destino	BBDD key, table

Necesita	BBDD clave primaria, columna, tabla y esquema relacional
Acción	Da de baja una clave primaria
Precondición	La clave primaria, las columnas que la componen, la tabla y el esquema relacional existan en la BBDD
Postcondición	Clave primaria borrada lógicamente de la BBDD clave primaria
Efectos Laterales	N/a

GF-1.5: Claves extranjeras de las tablas

RF-1.5.1 Crear clave extranjera	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de alta una clave extranjera en la BBDD
Entrada	Nombre de la clave extranjera, nombre de la tabla y nombre del esquema relacional al que pertenece y columnas que formarán la clave primaria
Salida	ID de la clave primaria
Origen	Teclado
Destino	BBDD clave primaria
Necesita	BBDD clave primaria, columna, tabla y esquema relacional
Acción	Da de alta una clave extranjera
Precondición	No exista una clave primaria en la misma tabla con el mismo nombre en la BBDD. Las columnas de la clave primaria son de la misma tabla
Postcondición	Clave primaria almacenada en la BBDD clave primaria
Efectos Laterales	

RF-1.2.2 Mostrar clave extranjera	
Prioridad	Alta
Estabilidad	Alta
Descripción	Muestra una tabla
Entrada	ID del esquema relacional al que pertenece e ID de la tabla
Salida	Nombre de la tabla
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD tabla y esquema relacional
Acción	Mostrar una tabla
Precondición	La tabla y el esquema relacional existen en la BBDD
Postcondición	Nombre de la tabla mostrado mediante interfaz gráfica

Efectos Laterales	N/a

RF-1.2.3 Modificar clave extranjera	
Prioridad	Alta
Estabilidad	Alta
Descripción	Modifica el nombre de una tabla
Entrada	ID de la tabla e ID del esquema relacional al que pertenece
Salida	Notificación mediante interfaz gráfica de la información actualizada
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD tabla y esquema relacional
Acción	Modificar el nombre de una tabla
Precondición	La tabla y el esquema relacional existen en la BBDD
Postcondición	Nombre de la tabla modificado y almacenado en la BBDD tabla
Efectos Laterales	N/a

R.F. 1.5.4 Eliminar clave extranjera	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de baja una clave primaria en la BBDD
Entrada	ID de la clave primaria e ID de la tabla a la que pertenece
Salida	
Origen	Teclado
Destino	BBDD clave primaria
Necesita	BBDD clave primaria, columna,tabla y esquema relacional
Acción	Dar de baja una clave primaria
Precondición	La clave primaria, las columnas que la componen, la tabla y el esquema relacionalexistan en la BBDD
Postcondición	Clave primaria borrada lógicamente de la BBDD clave primaria

Efectos Laterales	N/a

GF-2: Gestión de instancias de metadatos
Este grupo funcional contiene los casos de uso crear, mostrar, modificar y eliminar tuplas que pueblan las columnas de las tablas para esquemas relacionales de metadatos soportando las claves de dichas tablas

RF-2.1 Crear instancia de metadatos	
Prioridad	Alta
Estabilidad	Alta
Descripción	Crear una instancia de metadatos para una tabla explícita
Entrada	Nombre del esquema, nombre de la tabla y valores de las columnas de la nueva instancia
Salida	ID de instancia
Origen	Teclado
Destino	BBDD explícita en la que se quiera crear una instancia
Necesita	BBDD explícita en la que se quiera crear una instancia
Acción	Dar de alta una instancia de metadatos
Precondición	Restricciones de integridad: los valores de la instancia corresponderán con los tipos de las columnas de la tabla que se quiere crear una nueva instancia. No es posible valores repetidos para aquellas columnas que formen una clave primaria y si alguna columna es clave extranjera deberá corresponder con algún valor de la clave primaria de la otra tabla
Postcondición	Instancia de metadatos almacenada en la BBDD que se quiera crear una instancia
Efectos Laterales	N/a

RF-2.2 Mostrar instancia de metadatos	
Prioridad	Alta
Estabilidad	Alta
Descripción	Mostrar una instancia de metadatos de una tabla
Entrada	ID del esquema, ID de la tabla
Salida	Instancia de metadatos de una tabla concreta
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD schema, table y tabla explícita que se quiera consultar

Acción	Muestra una instancia de metadatos de una tabla
Precondición	El esquema relacional, la tabla y la tabla explícita existen en la BBDD
Postcondición	Valor de cada columna que forma la instancia de metadatos mostrada mediante interfaz gráfica
Efectos Laterales	N/a

RF-2.3 Modificar instancia de metadatos	
Prioridad	Alta
Estabilidad	Alta
Descripción	Modificar los valores de las columnas de una instancia de metadatos para una tabla explícita
Entrada	ID de la tabla, ID de la instancia de metadatos y los valores de las columnas
Salida	Notificación mediante interfaz gráfica de la información actualizada
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD table y tabla explícita y valores de las columnas de la instancia de metadatos
Acción	Modifica los valores de las columnas de una instancia de metadatos
Precondición	La tabla existe en la BBDD. Restricciones de integridad: los valores de la instancia corresponderán con los tipos de las columnas de la tabla que se quiere crear una nueva instancia. No es posible valores repetidos para aquellas columnas que formen una clave primaria y si alguna columna es clave extranjera deberá corresponder con algún valor de la clave primaria de la otra tabla
Postcondición	Valores de las columnas de la instancia de metadatos modificados y almacenados en la BBDD tabla
Efectos Laterales	N/a

RF-2.4 Eliminar instancia de metadatos	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de baja una instancia de metadatos de una tabla explícita en la BBDD

Entrada	ID de la tabla, ID de la instancia
Salida	
Origen	Teclado
Destino	BBDD tabla explícita
Necesita	BBDD tabla explícita
Acción	Da de baja una instancia de metadatos
Precondición	La instancia de metadatos y la tabla existen en la BBDD
Postcondición	Instancia de metadatos borrada lógicamente de la BBDD
Efectos Laterales	N/a

GF-3: Gestión de estructura de HLOs

RF-3.1 Crear estructura de HLO	
Prioridad	Alta
Estabilidad	Alta
Descripción	Crear estructura de HLO
Entrada	Nombre del HLO, descripción, esquema relacional, columnas que forman el HLO, columnas de join y columna directora
Salida	ID del HLO
Origen	Teclado
Destino	BBDD HLO
Necesita	BBDD schema, column, table y learning_object
Acción	Crea la estructura de un HLO
Precondición	El esquema relacional y las columnas que forman el HLO existen en la BBDD y el usuario selecciona correctamente las columnas de join y la columna directora
Postcondición	Estructura de HLO creada y almacenada en la BBDD
Efectos Laterales	N/a

RF-3.2 Mostrar estructura de HLO	
Prioridad	Alta
Estabilidad	Alta
Descripción	Muestra las columnas que forman un HLO y las tablas a las que pertenecen
Entrada	ID del HLO
Salida	Columnas que forman el HLO
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD learning_object, column y table
Acción	Muestra la información de un HLO
Precondición	El HLO y las columnas que lo forman existen en la BBDD learning_object y column respectivamente

Postcondición	Información de un HLO mostrada mediante interfaz gráfica
Efectos Laterales	N/a

RF-3.3 Eliminar estructura de HLO	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de baja un HLO en la BBDD
Entrada	ID del HLO
Salida	
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD learning_object
Acción	Da de baja un HLO
Precondición	El HLO existe en la BBDD learning_object
Postcondición	HLO borrado lógicamente de la BBDD
Efectos Laterales	N/a

GF-4: Gestión de recursosde los HLO

RF-4.1 Generar instancias de HLO	
Prioridad	Alta
Estabilidad	Alta
Descripción	Genera automáticamente las instancias de un HLO y la tabla tupla asociada a ese HLO
Entrada	ID del HLO
Salida	Número de instancias creadas
Origen	Teclado
Destino	BBDD tuple_X, donde X se corresponde con el ID del HLO y BBDD learning_object_instance
Necesita	BBDD learning_object, learning_object_instance y tuple_X
Acción	Genera las instancias de un HLO y la tabla tuple asociada a éste
Precondición	El usuario ha definido las columnas que forman el HLO, columna directora y columnas de join correctamente y éste existe en la BBDD
Postcondición	Instancias de HLO y tabla tupla creadas y almacenada en la BBDD learning_object_instance y tuple_X respectivamente
Efectos Laterales	N/a

GF-4.1 Gestión de recursos(ficheros de imágenes, texto, audio, etc.) de los HLO

RF-4.1.1 Subirrecursoexterno	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de alta un Recurso Externo en la BBDD
Entrada	Dirección en la que se almacena y descripción
Salida	ID del Recurso Externo
Origen	Teclado
Destino	BBDD external_resource
Necesita	BBDD external_resource
Acción	Dar de alta un Recurso Externo (ficheros de imágenes, texto, audio, etc.)
Precondición	El Recurso Externo sea un fichero legible por la aplicación
Postcondición	Recurso Externo almacenado en la BBDD external_resource
Efectos Laterales	N/a

RF-4.1.2 Mostrar recursoexterno	
Prioridad	Alta
Estabilidad	Alta
Descripción	Muestra el contenido de un Recurso Externo
Entrada	ID del Recurso Externo
Salida	Información del recurso externo
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD external_resource
Acción	Mostrar información de un Recurso Externo
Precondición	El Recurso Externo existe en la BBDD
Postcondición	Información del Recurso Externo mostrado mediante interfaz gráfica
Efectos Laterales	N/a

RF-4.1.3 Modificar recursoexterno	
Prioridad	Alta
Estabilidad	Alta
Descripción	Modifica la descripción de un Recurso Externo
Entrada	ID del Recurso Externo y la descripción
Salida	Notificación mediante interfaz gráfica de la información actualizada
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD external_resource
Acción	Modifica la descripción de un Recurso Externo
Precondición	El Recurso Externo existe la BBDD external_resource
Postcondición	Descripción del Recurso Externo modificado y almacenado en la BBDD external_resource
Efectos Laterales	N/a

RF-4.1.4 Eliminar recursoexterno	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de baja un Recurso Externo en la BBDD
Entrada	ID del Recurso Externo
Salida	
Origen	Teclado
Destino	BBDD external_resource
Necesita	BBDD external_resource
Acción	Da de baja un Recurso Externo
Precondición	El Recurso Externo existe en la BBDD external_resource
Postcondición	Recurso Externo borrado lógicamente de la BBDD external_resource
Efectos Laterales	N/a

GF-4.2 Gestión de referencias a otros HLO

RF-4.2.1 Crear referencia	
Prioridad	Alta
Estabilidad	Alta
Descripción	Vincula un HLO con uno o varios HLO creados
Entrada	ID del HLO e ID del HLO con el que se vincula
Salida	
Origen	Teclado
Destino	BBDD learning_object_instance
Necesita	BBDD learning_object_instance
Acción	Vincula dos HLO
Precondición	Ambos ID existen en la BBDD learning_object_instance
Postcondición	Referencia a otro HLO creada y almacenada en la BBDD learning_object_instance
Efectos Laterales	N/a

RF-4.2.2 Mostrar referencia	
Prioridad	Alta
Estabilidad	Alta
Descripción	Mostrar los HLOs con los que está relacionado el HLO consultado
Entrada	ID del HLO
Salida	HLOs vinculados
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD learning_object_instance
Acción	Muestra los HLO con los que está relacionado el HLO consultado
Precondición	El HLO existe en la BBDD learning_object_instance y tienevinvulado algún HLO
Postcondición	HLO vinculados mostrados mediante interfaz gráfica
Efectos Laterales	N/a

RF-4.2.3 Eliminar referencia	
Prioridad	Alta
Estabilidad	Alta
Descripción	Desvincular un HLO con otro HLO creado
Entrada	ID del HLO e ID del HLO con el que se desvincula
Salida	
Origen	Teclado
Destino	BBDD learning_object_instance
Necesita	BBDD learning_object_instance
Acción	Desvincula un HLO de otro HLO
Precondición	Ambos ID existen en la BBDD learning_object_instance y ambos HLO están vinculados
Postcondición	Referencia a otro HLO eliminada lógicamente de la BBDD learning_object_instance
Efectos Laterales	N/a

GF-4.3 Gestión de vínculos de HLO con Recursos Externos

RF-4.3.1 Vincular	
Prioridad	Alta
Estabilidad	Alta
Descripción	Vincular un HLO con un Recurso Externo
Entrada	ID del HLO e ID del Recurso Externo
Salida	Notificación mediante interfaz gráfica de la información vinculada
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD learning_object_instancey external_resource
Acción	Vincula un HLO con un Recurso Externo
Precondición	El HLO y el Recurso Externo existen en la BBDD learning_object_instance y external_resource respectivamente
Postcondición	Relación entre HLO y Recurso Externo creada y almacenada en la BBDD
Efectos Laterales	N/a

RF-4.3.2 Desvincular	
Prioridad	Alta
Estabilidad	Alta
Descripción	Desvincular un HLO de un Recurso Externo
Entrada	ID del HLO e ID del Recurso Externo
Salida	Notificación mediante interfaz gráfica de la información vinculada
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD learning_object_instancey external_resource
Acción	Desvincula un HLO de un Recurso Externo
Precondición	El HLO y el Recurso Externo existen en la BBDD learning_object_instance y external_resource respectivamente
Postcondición	Relación entre HLO y Recurso Externo eliminada lógicamente de la BBDD
Efectos Laterales	N/a

GF-5: Gestión de usuarios.

RF-5.1 Crear usuario	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de alta un usuario en la BBDD
Entrada	Nombre del usuario, contraseña e ID del rol
Salida	ID del usuario
Origen	Teclado
Destino	BBDD usuarios
Necesita	BBDD usuarios
Acción	Da de alta un usuario
Precondición	El rol exista en la BBDD
Postcondición	Usuario almacenado en la BBDD usuario
Efectos Laterales	N/a

RF-5.2 Mostrar usuario	
Prioridad	Alta
Estabilidad	Alta
Descripción	Mostrar un usuario
Entrada	ID del usuario
Salida	Información de usuario
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD usuario
Acción	Muestra información de un usuario
Precondición	El usuario existe en la BBDD
Postcondición	Información del usuario mostrado mediante interfaz gráfica
Efectos Laterales	N/a

RF-5.3 Modificar usuario	
Prioridad	Alta
Estabilidad	Alta
Descripción	Modificar información de usuario
Entrada	ID del usuario, nombre y contraseña
Salida	Notificación mediante interfaz gráfica de la información actualizada
Origen	Teclado
Destino	Interfaz gráfica
Necesita	BBDD usuario
Acción	Modifica la información de un usuario
Precondición	El usuario existe en la BBDD usuario
Postcondición	Usuario modificado y almacenado en la BBDD usuario
Efectos Laterales	N/a

RF-5.4 Eliminar usuario	
Prioridad	Alta
Estabilidad	Alta
Descripción	Dar de baja un usuario en la BBDD
Entrada	ID del usuario
Salida	
Origen	Teclado
Destino	BBDD usuario
Necesita	BBDD usuario
Acción	Da de baja un usuario
Precondición	El usuario existe en la BBDD usuario
Postcondición	Usuario borrado lógicamente de la BBDD usuario
Efectos Laterales	N/a

Diagramas de casos de uso

Los diagramas de casos de uso representan de forma gráfica los requisitos y actores que intervienen en cada uno de ellos.

Diagrama de Casos de Uso del actor ACT-01: Administrador

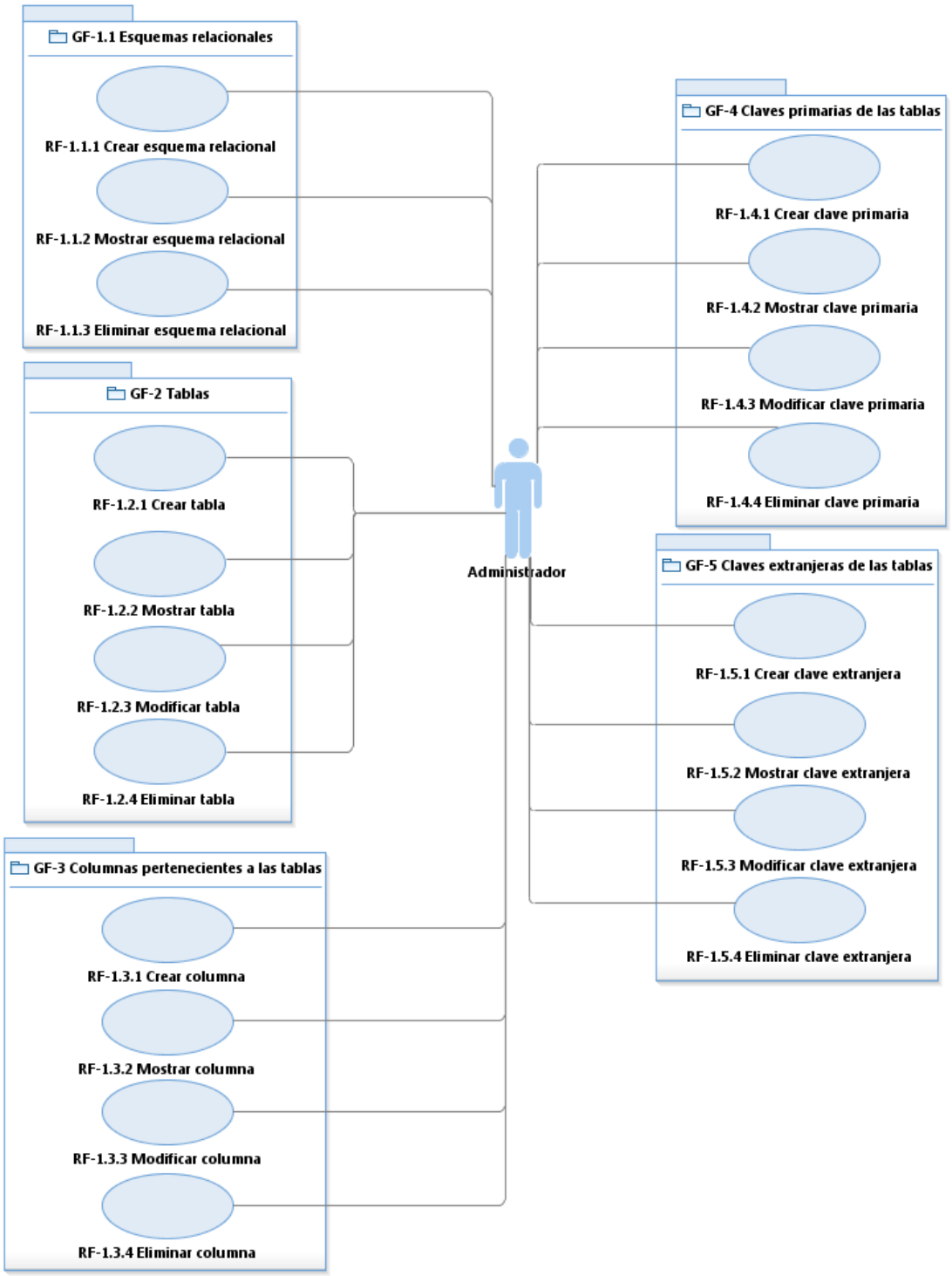


Figura 3.1.1 Diagrama de Casos de Uso del actor ACT-01: Administrador (I)

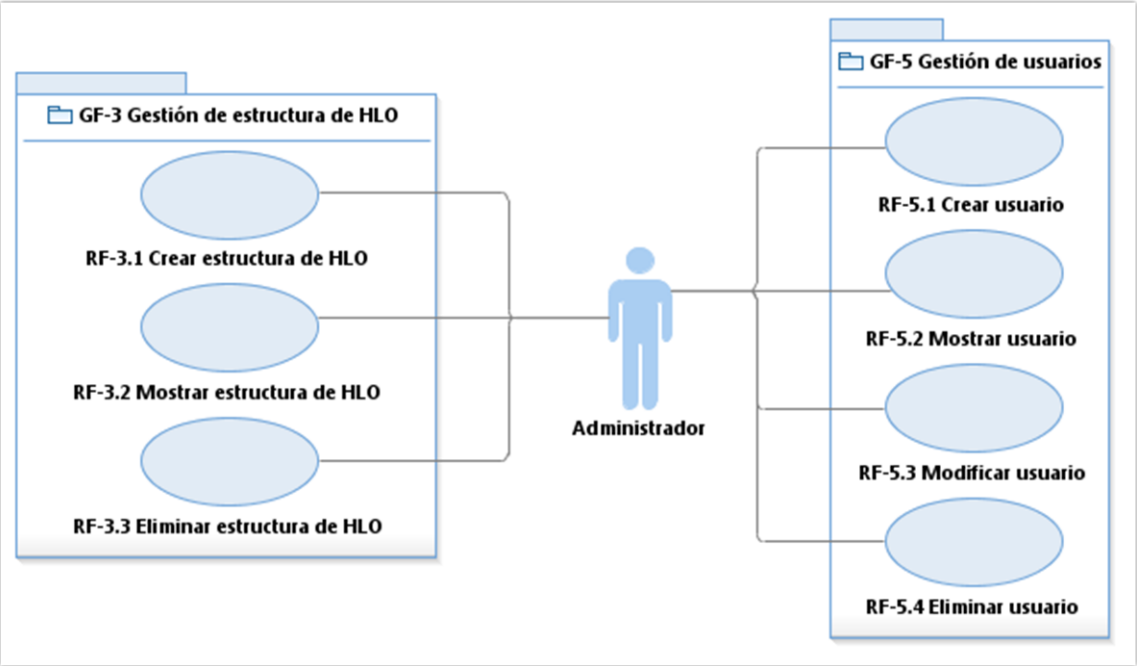


Figura 3.1.2 Diagrama de Casos de Uso del actor ACT-01: Administrador (II)

Diagrama de Casos de Uso del actor ACT-02: Usuario registrado

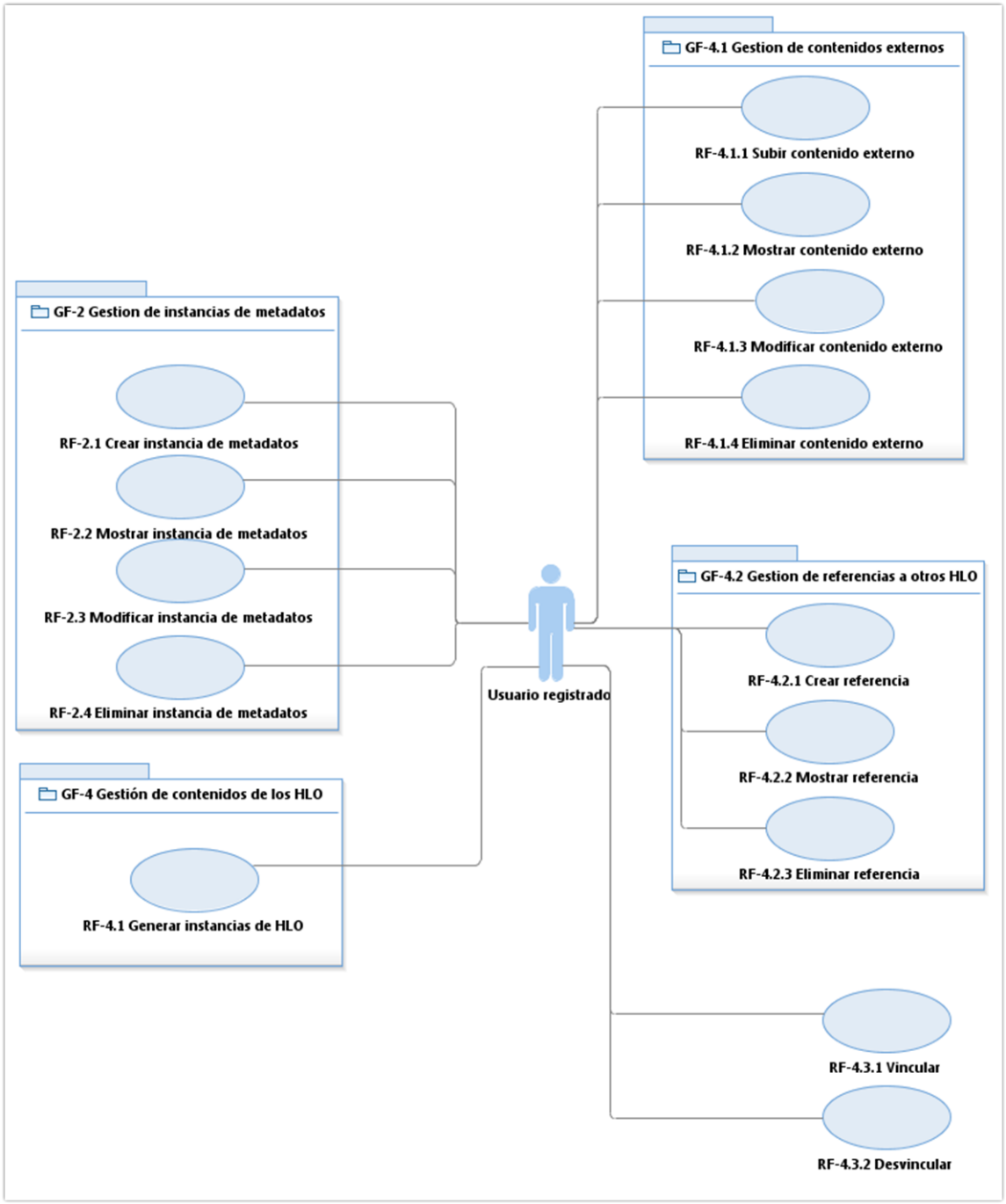


Figura 3.1.3 Diagrama de Casos de Uso del actor ACT-02: Usuario registrado

Especialización de actores

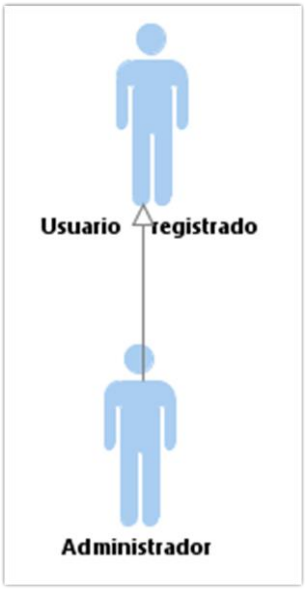


Figura 3.1.4 Herencia entre actores ACT-01 Administrador y ACT-02 Usuario registrado

3.2 Arquitectura Software

El diseño de la aplicación está basado en una arquitectura multicapa tal y como muestra la Figura x. Como puede apreciarse en esta figura, la persistencia de la parte que gestiona el metamodelo relacional está encomendada a JPA, mientras que la parte que gestiona las instancias de los esquemas relacionados codificados en dicho metamodelo está directamente gestionada por DAOs.

El siguiente esquema ejemplifica la arquitectura software presente en nuestro proyecto OdaJ2EE.

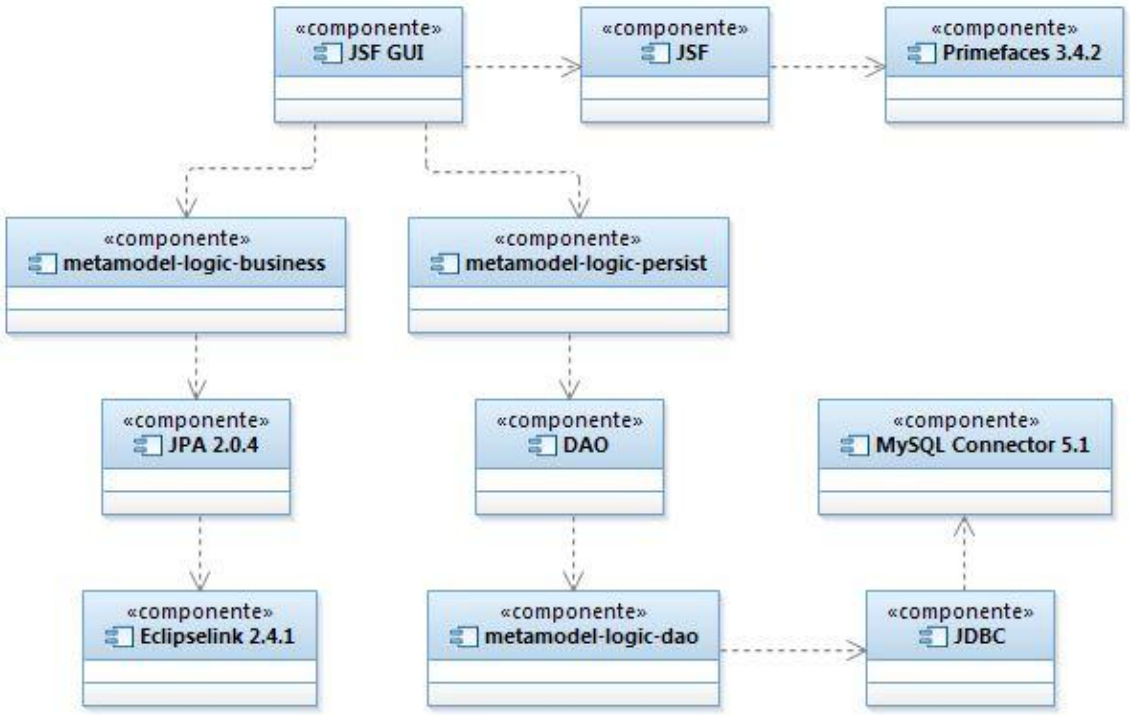


Figura 3.2.1. Diagrama de componentes de la arquitectura software.

Como se observa se hace uso de Arquitectura Multicapa.

La arquitectura multicapa considera una capa de presentación, otra de negocio, y otra de integración.

La *capa de presentación* encapsula toda la lógica de presentación necesaria para dar servicio a los clientes que acceden al sistema.

Para ella usamos el nuevo framework de Java: JSF (Java Server Faces) que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE.

Y como extensión de JSF usamos PrimeFaces Es una librería muy liviana, todas las decisiones hechas son basadas en mantener a PrimeFaces lo más ligero posible. No necesita dependencias ni configuraciones.

La *capa de negocio* proporciona los servicios del sistema.

Como hemos mencionado antes, hay una parte de negocio responsable de la gestión del metamodelo relacional que utiliza objetos de negocio implementadas como entidades JPA (Java Persistence API), el API de persistencia desarrollada para la plataforma Java EE. Usamos la implementación de EclipseLink 2.4.1.

El objetivo que persigue el diseño de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos (siguiendo el patrón de mapeo objeto-relacional), y permitir usar objetos regulares (conocidos como POJOs).

También hay una parte que gestiona la creación explícita del esquema de datos relacional acorde al metamodelo relacional que, al definir dinámicamente esquemas de datos no puede basarse en JPA, e interactúa por tanto con la base de datos a través de DAOs.

La *capa de integración* es responsable de la comunicación con recursos y sistemas externos.

Además de JPA, para esta capa usamos DAO's (*Data Access Object*) que es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una base de datos o un archivo.

Y por último, hacemos uso de MySQL 5.1 que es un sistema de gestión de bases de datos relacional, multihilo y multiusuario.

3.3 Metamodelo

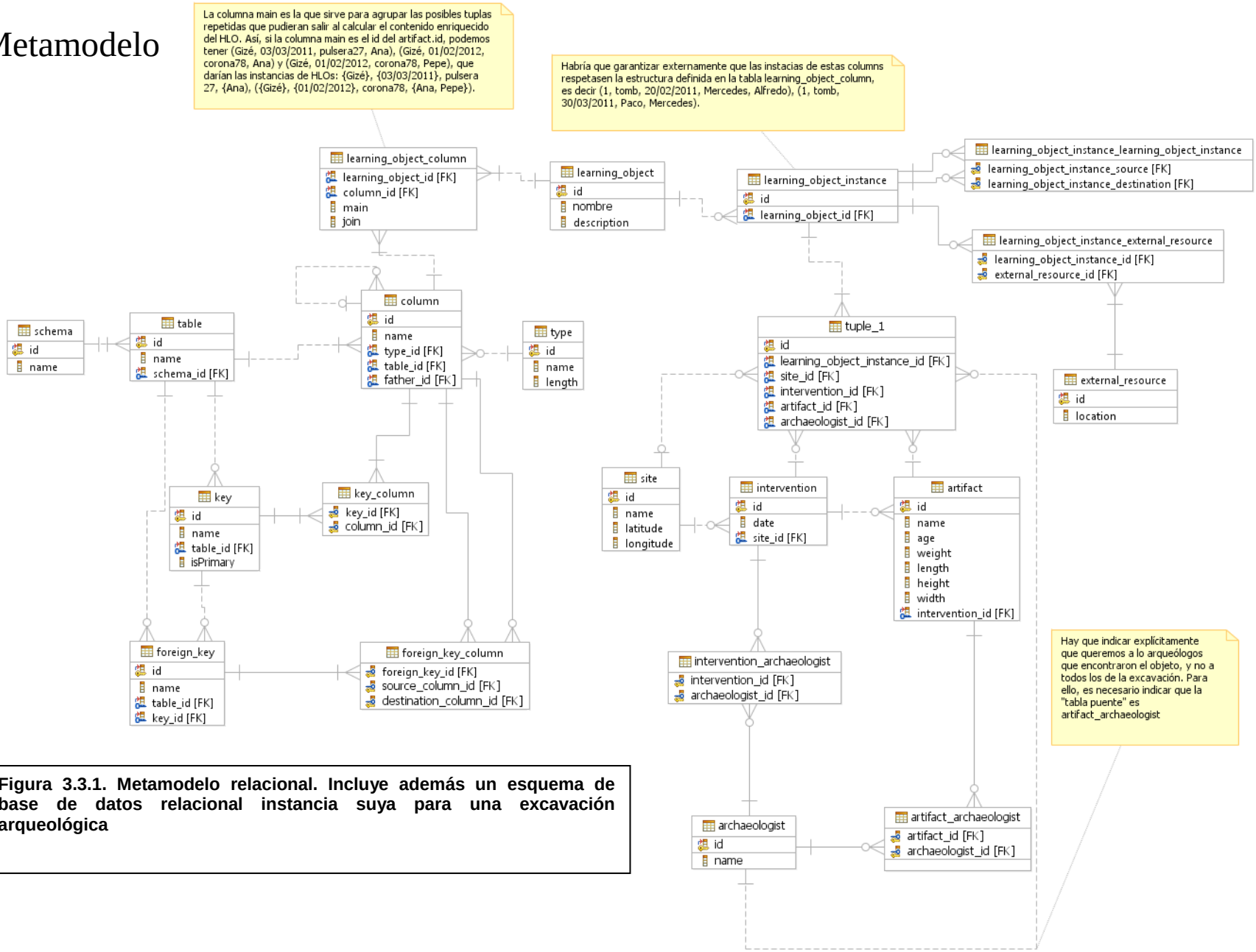


Figura 3.3.1. Metamodelo relacional. Incluye además un esquema de base de datos relacional instancia suya para una excavación arqueológica

OdA J2EE implementa un metamodelo relacional para la definición dinámica de metadatos dependientes del dominio para objetos de aprendizaje híbridos. El objetivo es poder codificar esquemas relacionales como el descrito en la Figura 3.3.1.

El meta-modelo definido en la Figura 3.3.1 tiene los siguientes elementos:

- `schema`: representa el esquema de base de datos. Por ejemplo, un esquema de base de datos para una sitio arqueológico.

- `table`: representa las tablas que pueblan este esquema. Por ejemplo, la tabla "Artifact". Las tablas se generan explícitamente como se muestra en el modelo "Site", "Artifact, "Intervention"…

- `column`: representa las columnas de las tablas definidas en el esquema. Por ejemplo, la columna "Nombre" de la tabla "Artifact".

- `type`: representa el tipo de las columnas de la tabla. Por ejemplo, el nombre de la columna puede ser varchar [50].

- `key`: representan las claves principales y únicas de las tablas. Por ejemplo, la columna "id" de la tabla "Artifact" es una key. Por lo tanto, no pueden existir valores repetidos en esta columna.

- `key_column`: representa las columnas que forman parte de una clave. Por ejemplo, la clave de la tabla "Artifact" está formado por su columna "id".

- `foreign_key`: representa las claves foráneas de una tabla. Por ejemplo, "intervention_id" de la tabla "Artifact" se ha definido en la columna "id" de la tabla "Intervention".

- `foreign_key_column`: representa las columnas de la clave extranjera. Por ejemplo, la columna "intervention_id" de la tabla "Artifact" referencia a la columna "id" de la tabla "Intervention".

- `learning_object`: define la estructura de información de dominio dependiente del LO en términos de columnas del esquema de base de datos relacional codificados. Por ejemplo, podemos definir el LO "artefacto arqueológico".

- `learning_object_column`: define las columnas específicas que componen unLO. Esta es sólo información estructural acerca del LO. Por ejemplo, el LO "artefactoarqueológico" puede estar formado por las columnas "Site.name", "Intervention.date", "Artifact.name", Artifact.high(cm) y Artifact.diameter(cm).

- `learning_object_instance`: define las instancias de información de dominio dependiente del esquema del LO, agrupadas para su visualización por la columna directora del LO.

- `tuple_n`: define las instancias de información de dominio dependiente del esquema del LO, sin hacer ningún tipo de agrupamiento.

- `learning_object_instance_column_instance`: define los valores del dominio dependiente de columnas que componen el LO.

- `external_resource`: define los recursos externos de un LO (es decir, sus archivos de contenido), tales como: fotos, vídeos y archivos de audio. Por ejemplo, la photo145.jpg es un recurso externo que puede ser ubicado en%PAHT%/oda/external_resources/.

- `learning_object_external_resource`: define los recursos externos específicos del LO. Por ejemplo, una instancia del objeto arqueológico LO puede estar vinculada al recurso externophoto145.jpg.

- `learning_object_instance_learning_object_instance`: define las referencias que el LO puede tener con otros LOs.

En la figura también pueden verse algunas tablas que se corresponderían con las tablas construidas explícitamente para soportar el modelo de la Figura y. Estas tablas son:

- `site`: define los lugares de las excavaciones.

- `intervention`: indica la fecha en la que tuvieron lugar las excavaciones.

- `artifact`: indica los objetos que se descubrieron.

- `archaeologist`: define los arqueólogos.

- `intervention_archaeologist`: tabla de join que asocia los arqueólogos con las intervenciones en que participaron.

- `artifact_archaeologist`: tabla de join que asocia los artefactos con los arqueólogos que los descubrieron.

3.4 Diseño Detallado

Está sección analiza y detalla en mayor profundidad el diseño de los elementos principales de la aplicación, en términos UML. A pesar de haber realizado un modelo UML (en términos de diagramas de clases y secuencia) utilizando la herramienta IBM RSA, el tamaño de dicho modelo hace inviable su descripción completa en la memoria.

No obstante, en esta sección se describe la estructura de paquetes del mismo, así como la estructura básica en términos de patrones multicapa (Alur et al., 2003) que componen el modelo.

En lo referente a los patrones multicapa utilizados, los mismos determinan el diseño que se detalla a continuación.

3.4.1 Diseño de Arquitectura Detallada

La arquitectura multicapa marca el diseño de la aplicación.

Para la capa de presentación:

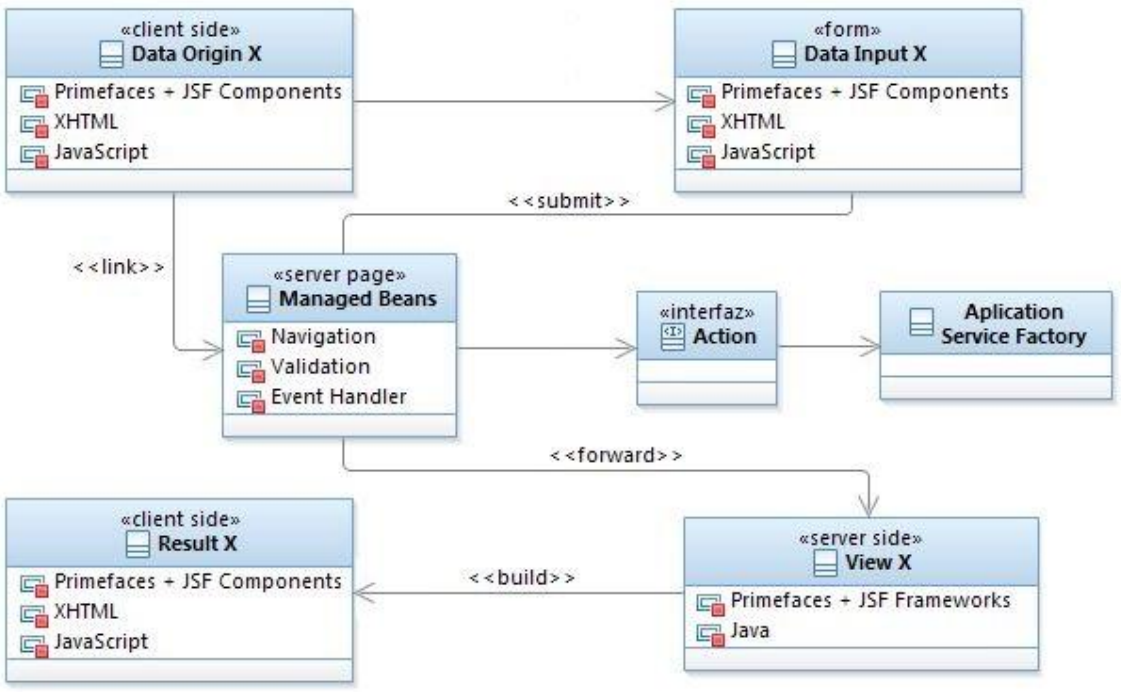


Figura 3.4.1.1 Los beans controlan la navegación y validación de datos, conectando las peticiones con los servicios de aplicación

El ciclo de vida de JSF comienza cuando un usuario hace una petición HTTP a través de su navegador y termina cuando el servidor le responde con la página correspondiente.

Para la capa de negocio:

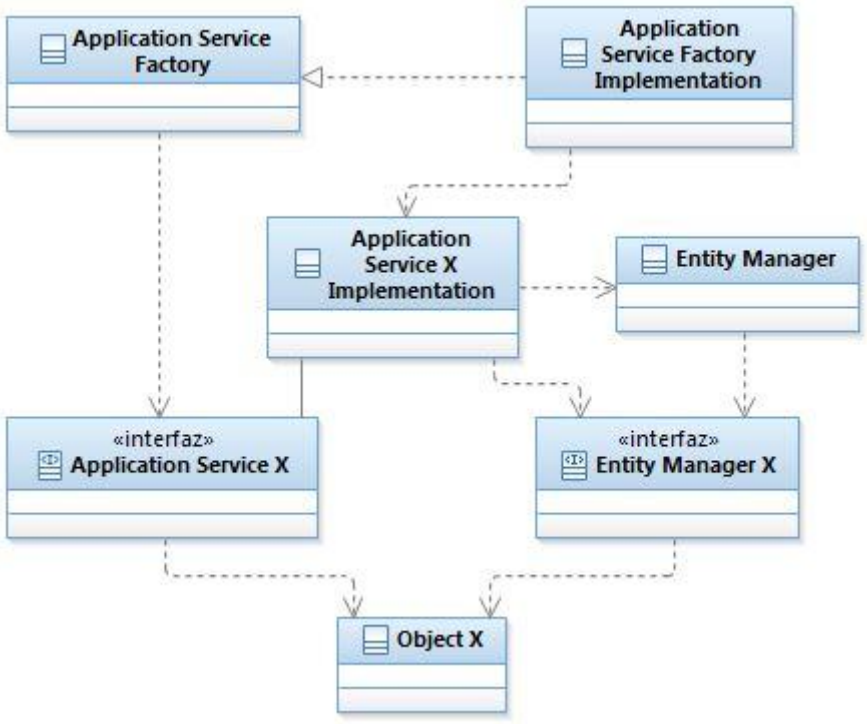


Figura 3.4.1.2 La capa de negocio hace uso de factorías abstractas para crear los servicios de aplicaciones

Los ApplicationServices nos proporcionan la funcionalidad para la gestión de los objetos del negocio y su interacción. Estos ApplicationServices se encargan de relacionarse con los EntityManager específicos.

Los EntityManager se configuran para ser capaces de persistir o administrar tipos específicos de objetos, leer y escribir en una base de datos, y ser implementados por un proveedor de persistencia en particular.

Todos los EntityManagers provienen de fábricas tipo EntityManagerFactory. La configuración de un EntityManager es formateado por la EntityManagerFactory que lo creó, sin embargo se define por separado como una unidad de persistencia.

Una unidad de persistencia dicta, ya sea implícita o explícitamente, los valores y clases de entidad que utilizan todos los EntityManagers obtenidos a partir de la instancia única del EntityManagerFactory asociada a esa unidad de persistencia. Existe, por tanto, una correspondencia uno a uno entre una unidad de persistencia y su EntityManagerFactory concreto.

No todos los servicios de aplicación interactuúan con entidades JPA. Algunos interactúan con la capa de integración a través de DAOs y transfers. Estos son los servicios de aplicación responsables del manejo de los esquemas relacionales explícitamente definidos en la aplicación. Precisamente, la capa de integración y transacciones con las que interactúan estos servicios de aplicación se describen a continuación.

Para la capa de integración:

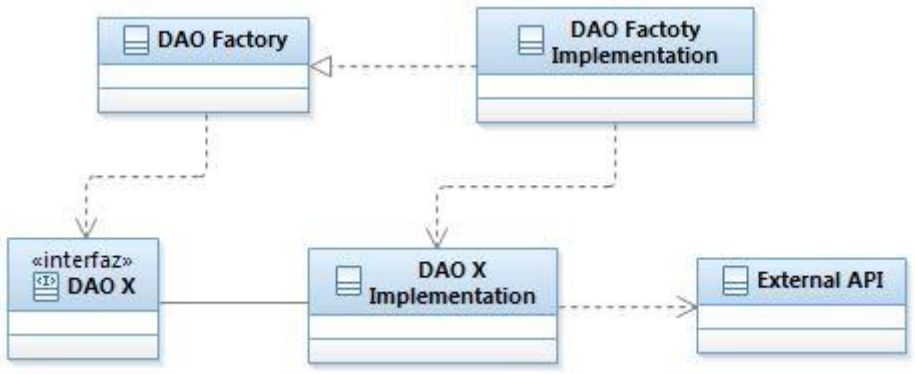


Figura 3.4.1.3 Los DAO se encargan de la transferencia

Está capa se encargará de devolver una interfaz implementada según se haya decidido usar. La aplicación misma no cambia, solo cambiará la implementación del acceso a la base de datos.

Esa es la idea del DAO + Factory: solo cambiar el acceso a la base de datos, el resto de la aplicación no debe sufrir modificación alguna. De tal manera que si se necesita cambiar la manera de acceder a la base de datos, solo se necesita cambiar en el la implementación que necesitemos.

Para las transacciones de la parte DAO:

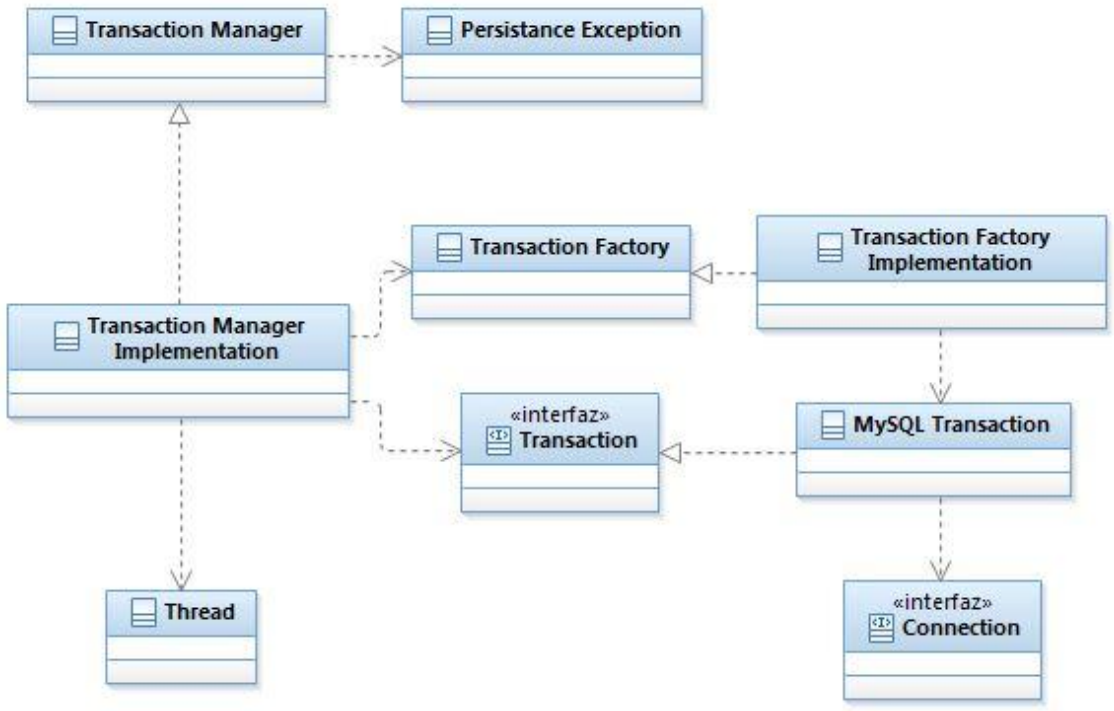


Figura 3.4.1.4 Las transacciones sobre objetos del negocio se delegan en JPA

La transacción es la unidad de trabajo de JPA. Cuando se cierra una transacción, JPA ve cual es el estado de las entidades y realiza las operaciones necesarias sobre base de datos para mantener la coherencia entre las entidades y la base de datos. En nuestro caso las transacciones se generan por medio del DAO, tal como se muestra en la figura 3.4.1.3.

Aunque se pueden crear, actualizar y eliminar en cualquier contexto, estas operaciones se realizan normalmente en el contexto de una transacción debido a que una transacción es necesaria para que los cambios que sean guardados en la base de datos. Los cambios son realizados en la base de datos ya sean éxito o fracaso, por lo que la visión persistente de una entidad de hecho debe ser transaccional.

En la memoria de ejecución del ordnador, es una historia ligeramente diferente en el sentido de que las entidades pueden ser modificadas sin que los cambios sean persistidos. Incluso cuando se listaron en una transacción, se puede dejar indefinido o en un estado inconsistente en el caso de un retroceso o el fracaso de la transacción.

Las principales propiedades de estas transacciones son atomicidad, consistencia y aislamiento.

3.4.2 Modelo de Objetos del Negocio

El siguiente modelo muestra los principales objetos de negocio del proyecto, especificando sus atributos, métodos y relaciones. Cada Entidad de nuestro modelo será utilizada por un servicio de aplicaciones como medio de comunicación y acceso externo.

Schema

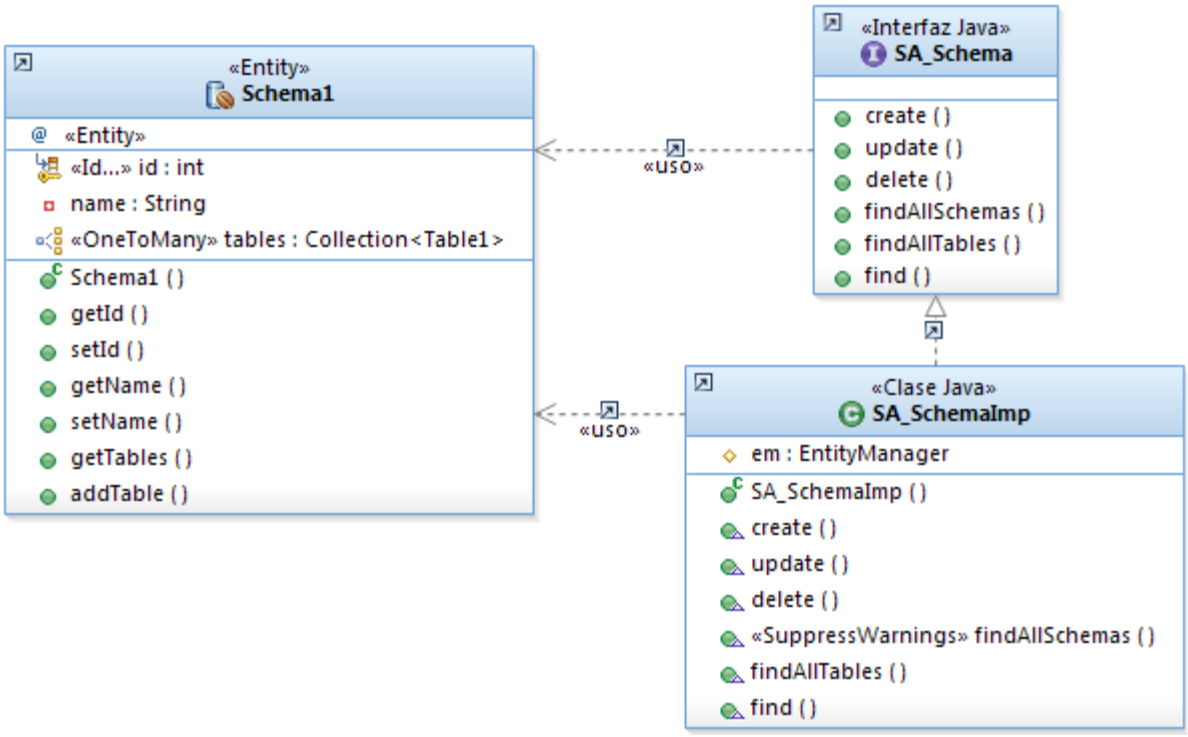


Figura 3.4.2.1 Objeto Schema

Schema1 representa el esquema de una base de datos. Este esquema puede contener de una a varias tablas.

Table

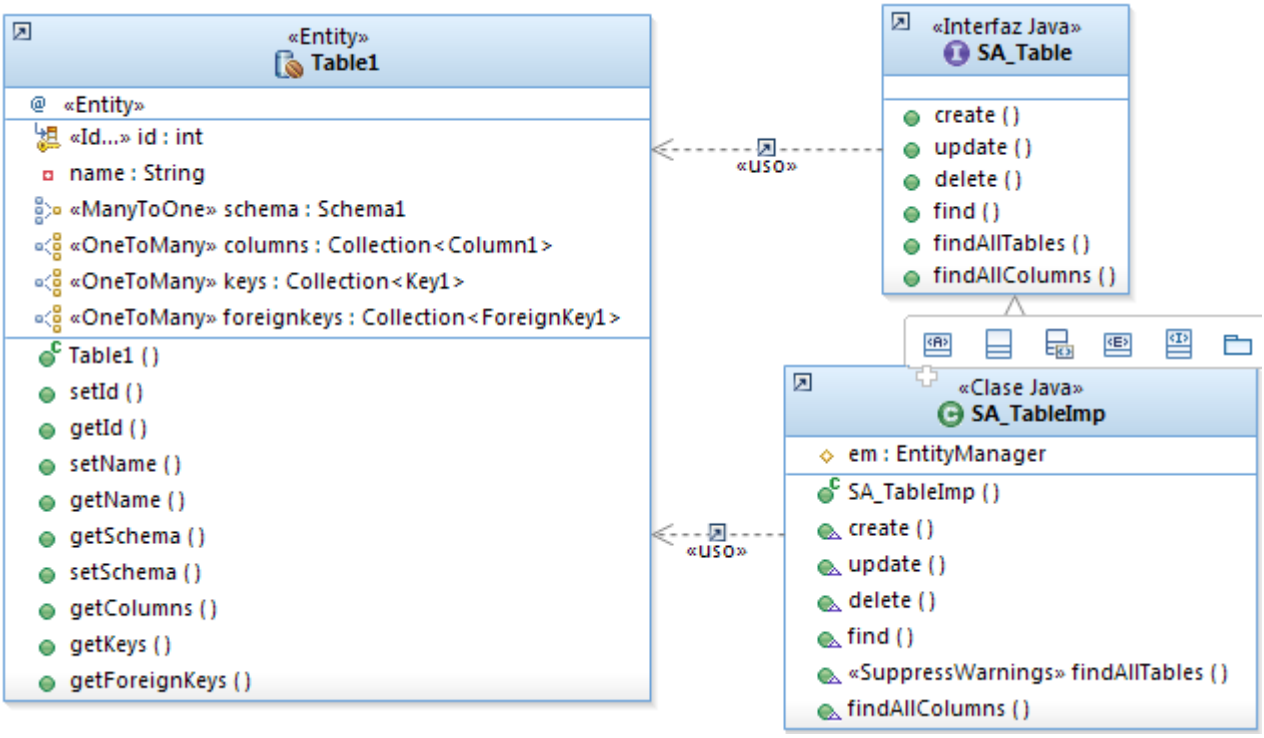


Figura 3.4.2.2 Objeto Table

Table1 representa a una tabla de un esquema. La tabla pertenece sólo a un esquema, y puede contener una o más columnas, claves principales y claves extranjeras.

Column

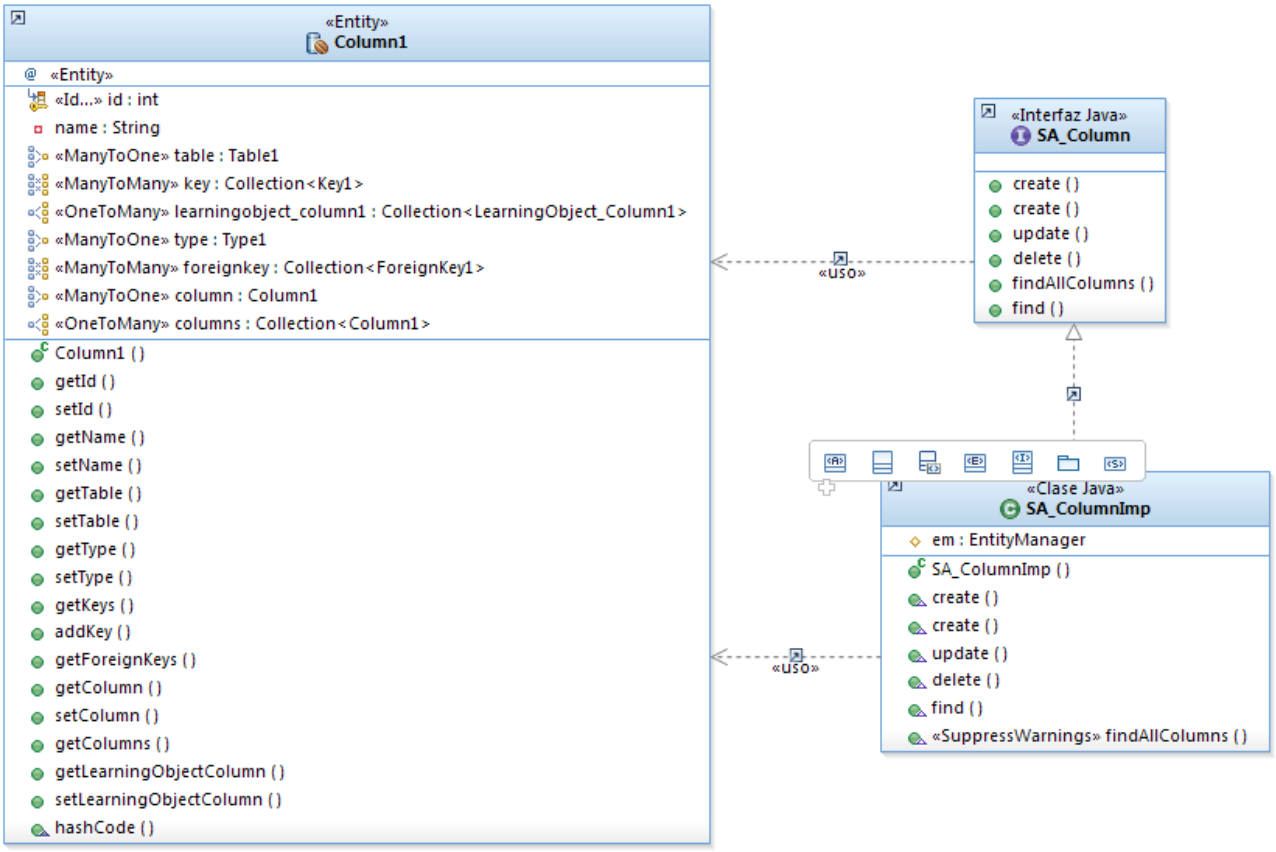


Figura 3.4.2.3 Objeto Column

Column1 representa una columna perteneciente a una tabla. A su vez, esta columna puede pertenecer a una o varias claves principales, una o varias claves extranjeras y uno o más columnas pertenecientes a objetos de aprendizaje. Además contiene un tipo específico de la base de datos.

Key

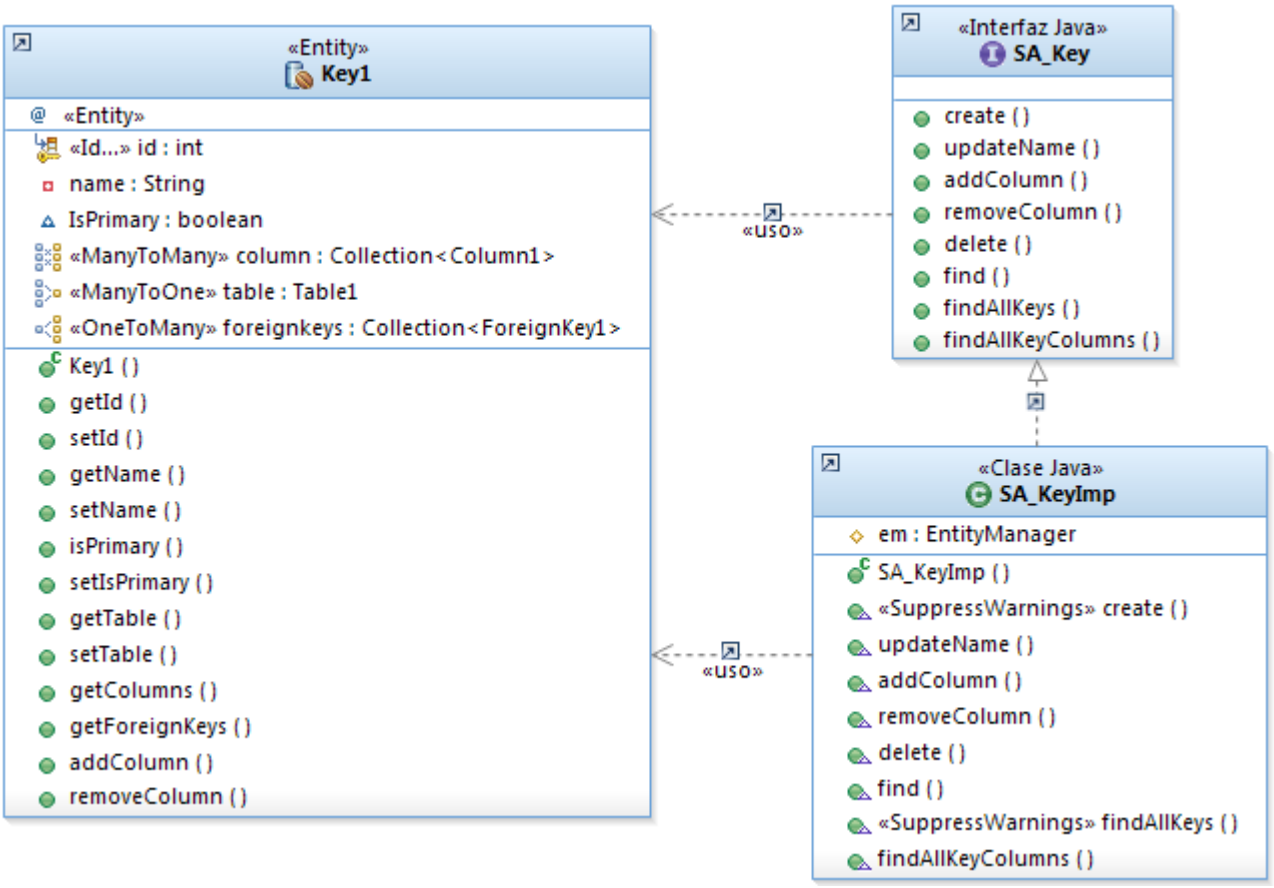


Figura 3.4.2.4 Objeto Key

Key1 representa la clave primaria de una tabla de la base de datos. Está clave está compuesta por varias columnas de esa tabla, además de poder estar relacionada con una clave extranjera.

Foreign Key

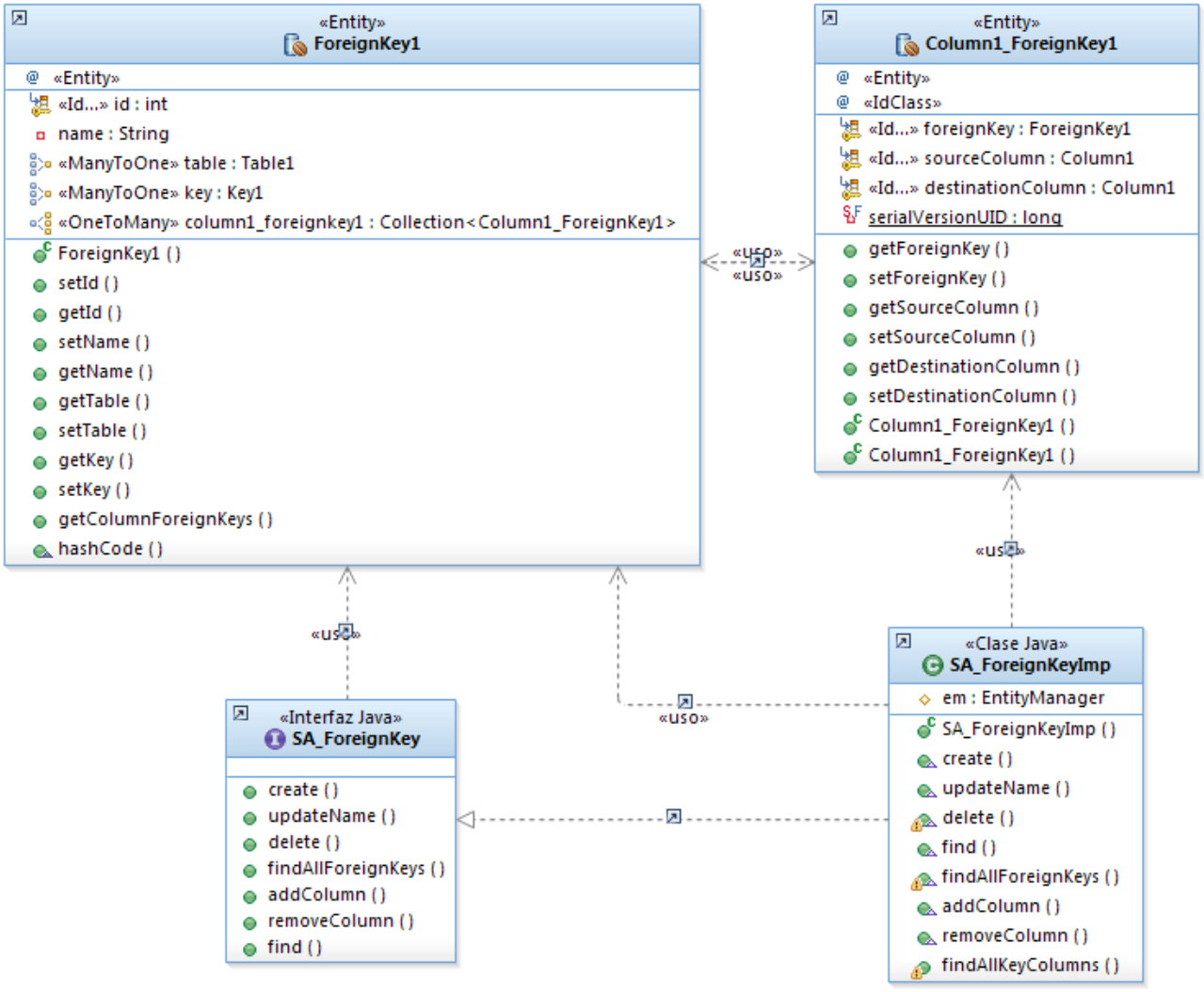


Figura 3.4.2.5 Objeto Foreign Key

ForeignKey1 representa una clave extranjera de una tabla de la base de datos. Está clave está compuesta por varias columnas de esa tabla, además de poder estar relacionada con una clave primaria. Para su correcta implementación, utiliza la entidad Column1_ForeignKey1 para asociar las columnas de la clave a la que apunta, especificando el origen y destino.

Learning Object

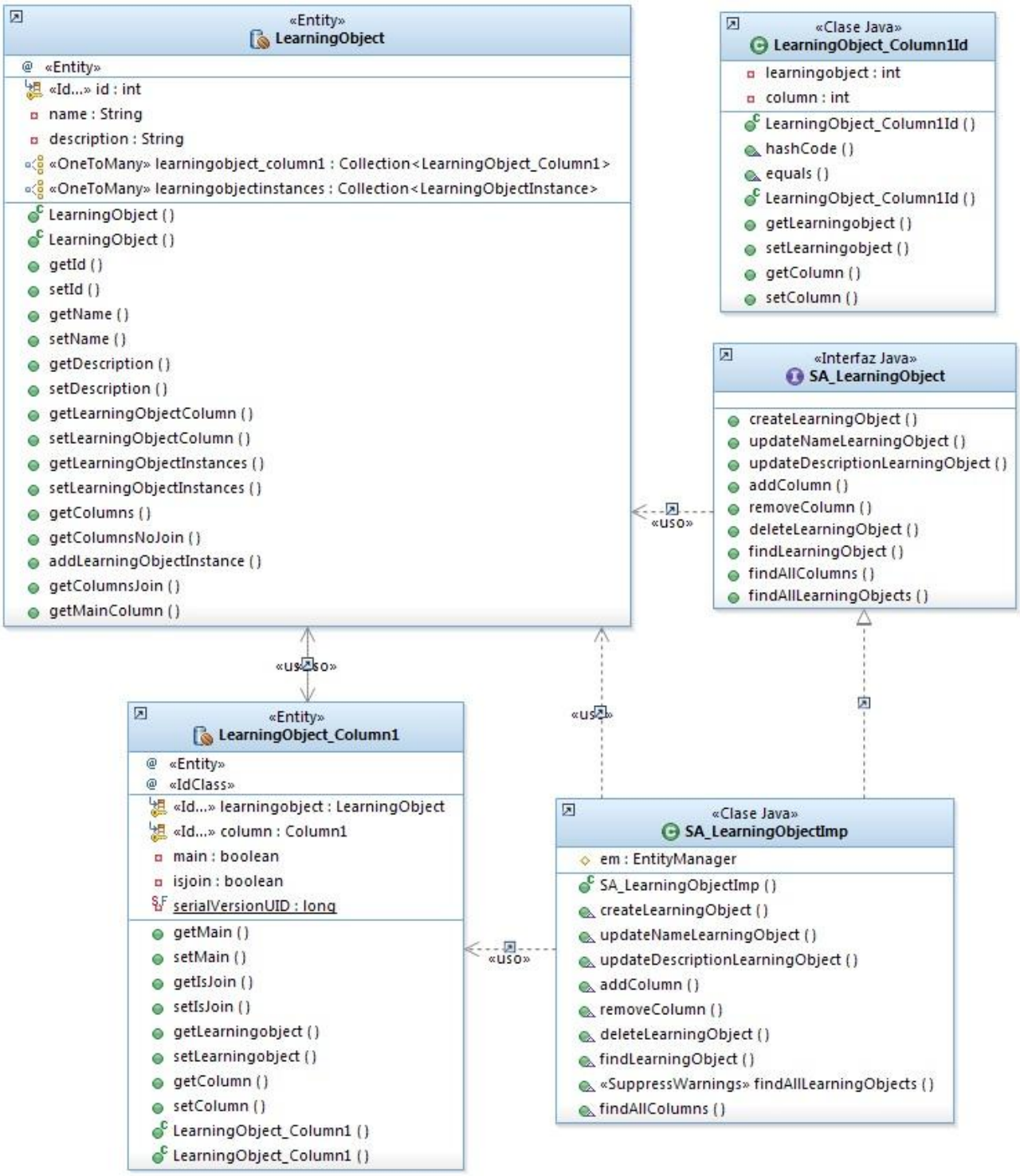


Figura 3.4.2.6 Objeto Learning Object

LearningObject1 representa un objeto de aprendizaje. El objeto está compuesto por columnas (de la misma tabla o diferentes tablas), conteniendo las distintas instancias que pueda generar. Además, nos apoyamos en la entidad LearningObject_Column1 para controlar si las columnas pertenecen al objeto son columnas main o de join.

Learning Object Instance

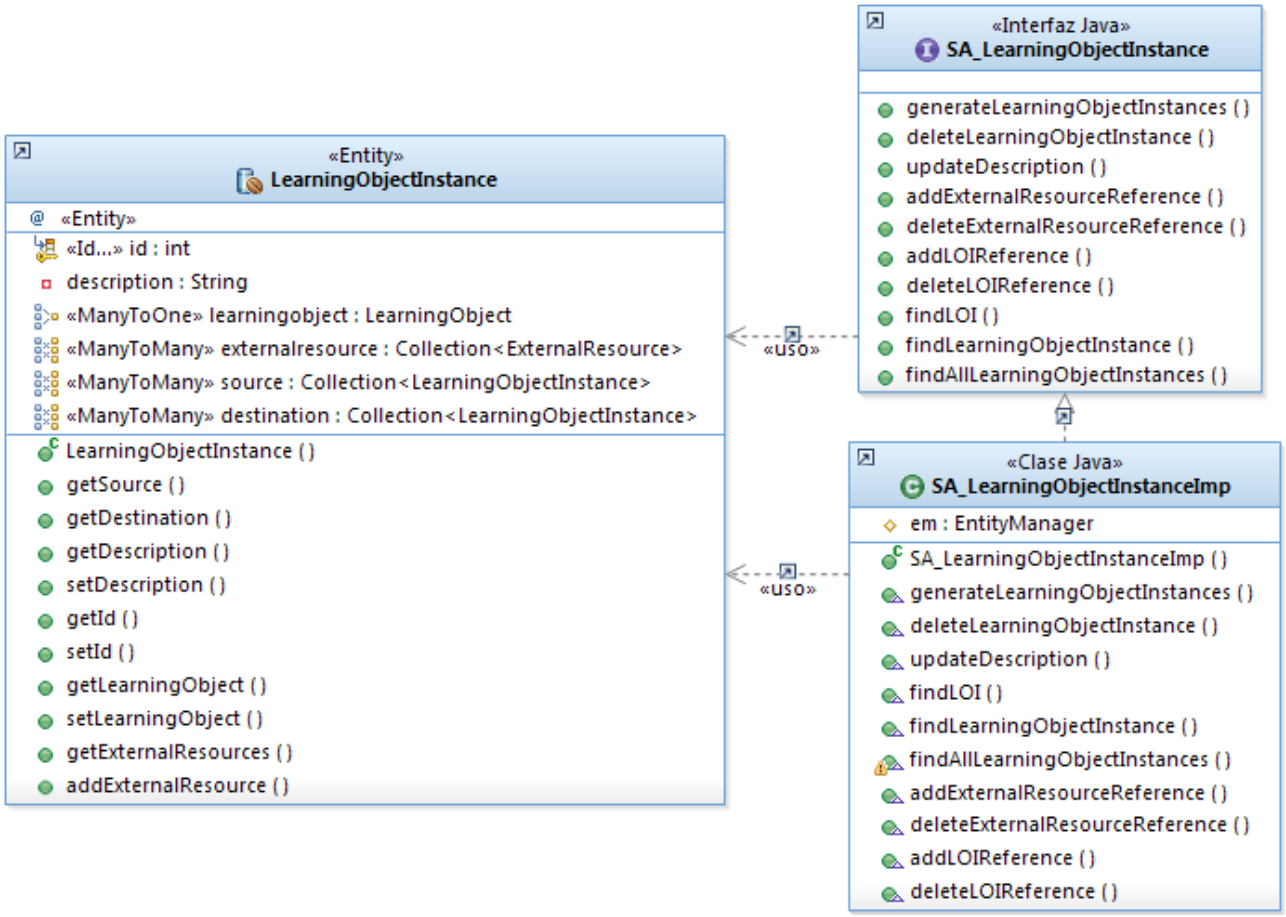


Figura 3.4.2.7 Objeto Learning Object Instance

LearningObjectInstance representa una instancia de un objeto de aprendizaje. Está instancia además de señalar su objeto de aprendizaje al que pertenece, contiene los recursos externos asociados y las relaciones con otras instancias de objetos de aprendizaje, indicando su origen y destino.

External Resource

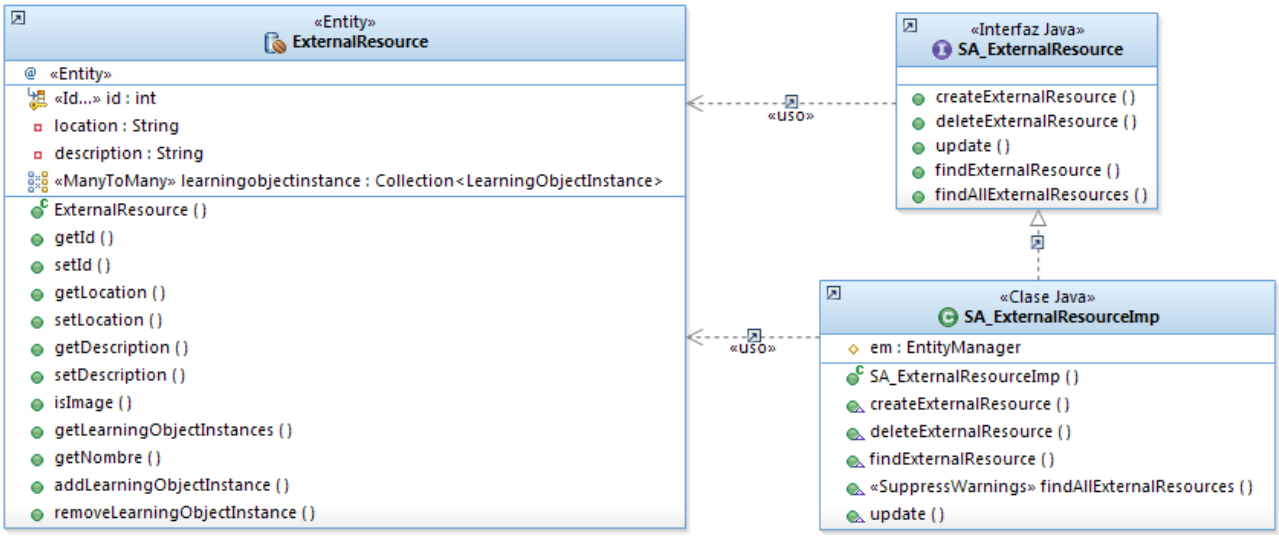


Figura 3.4.2.8 Objeto External Resource

ExternalResource representa un recurso externo al que se asocia una o varias instancias del objeto de aprendizaje. Puede ser desde una imagen a cualquier otro tipo de documento.

Instance

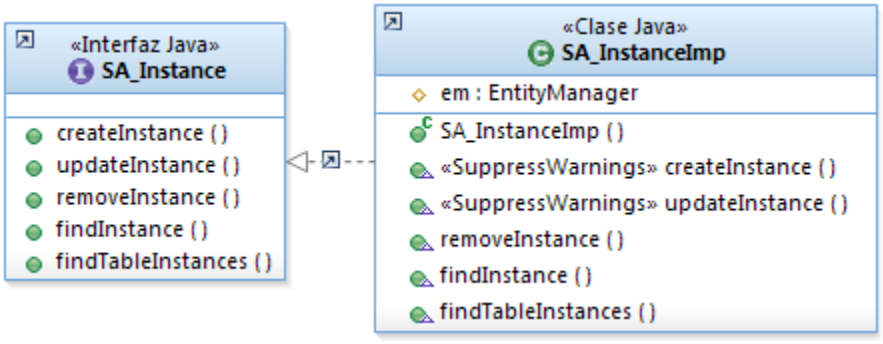


Figura 3.4.2.9 Objeto Instance

Aunque las instancias de nuestras tablas de un esquema de la base de datos no quedan representadas en un objeto propio, utilizamos un servicio de aplicaciones similar a los vistos anteriormente para su manejo. Estas instancias manejadas son los datos con los que se rellenan los distintos campos de una tabla.

User

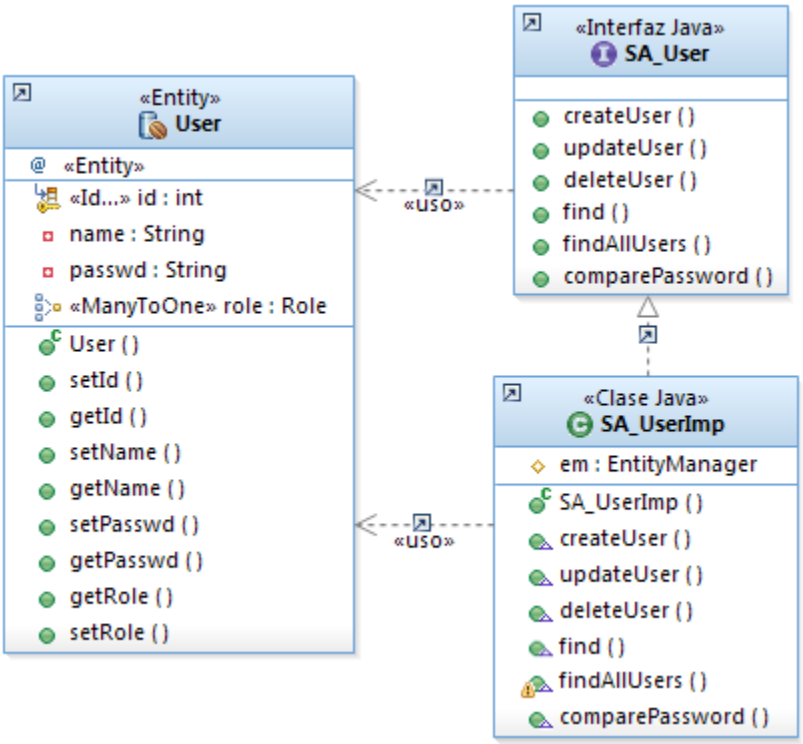


Figura 3.4.2.10 Objeto User

User representa al usuario de nuestra aplicación. Cada usuario está compuesto por un rol, nombre y contraseña.

Role

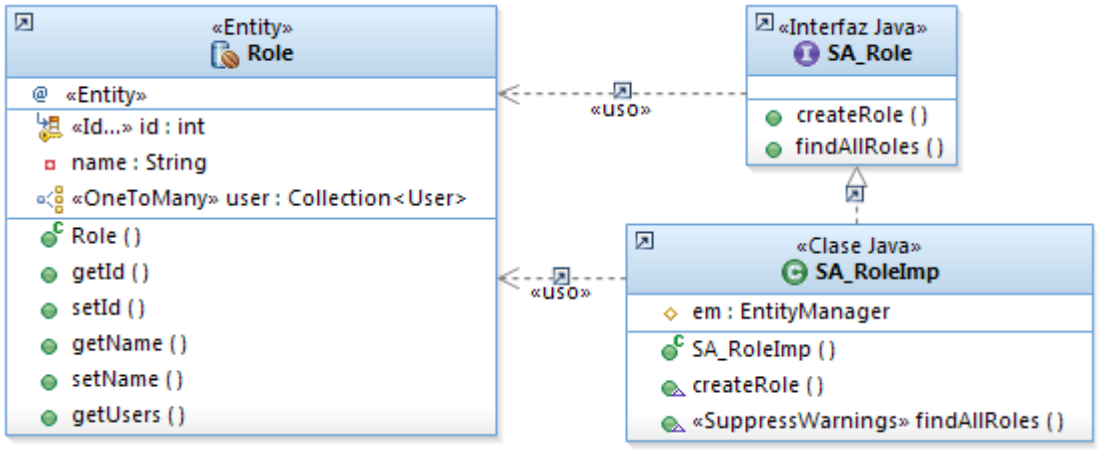


Figura 3.4.2.11 Objeto Role

Role representa los roles de usuario de nuestra aplicación. Cada rol está relacionado con varios usuarios diferentes.

Type

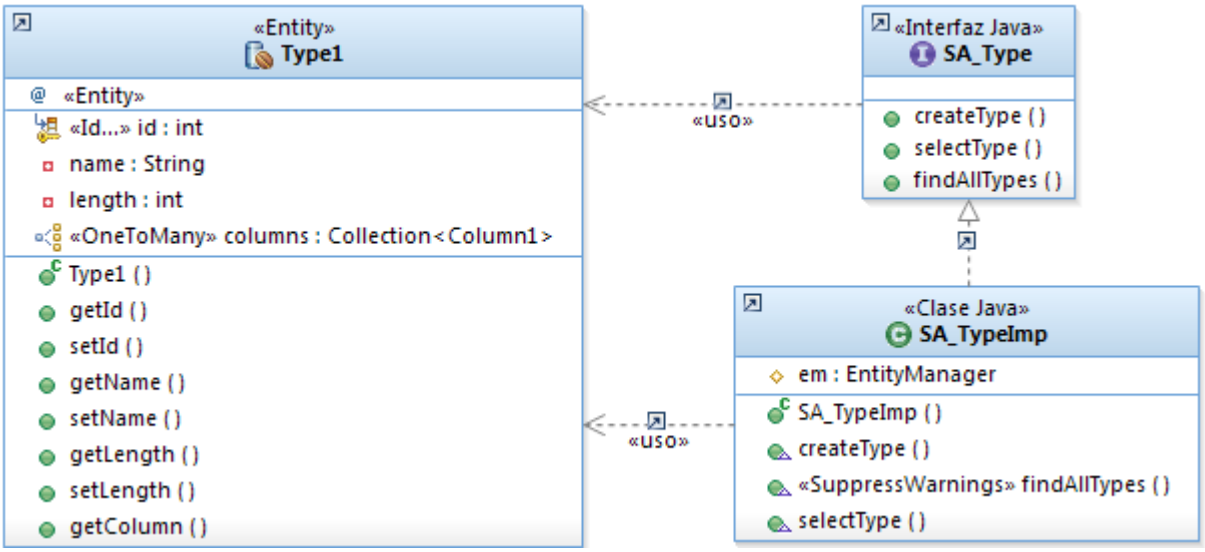


Figura 3.4.2.12 Objeto Type

Type1 representa los distintos tipos de datos existentes en la base de datos, así como su longitud.

3.4.3 Modelo Relacional de la BBDD

El siguiente modelo relacional muestra el diseño de nuestra base de datos necesario para implementar la solución y las relaciones entre las diferentes tablas.

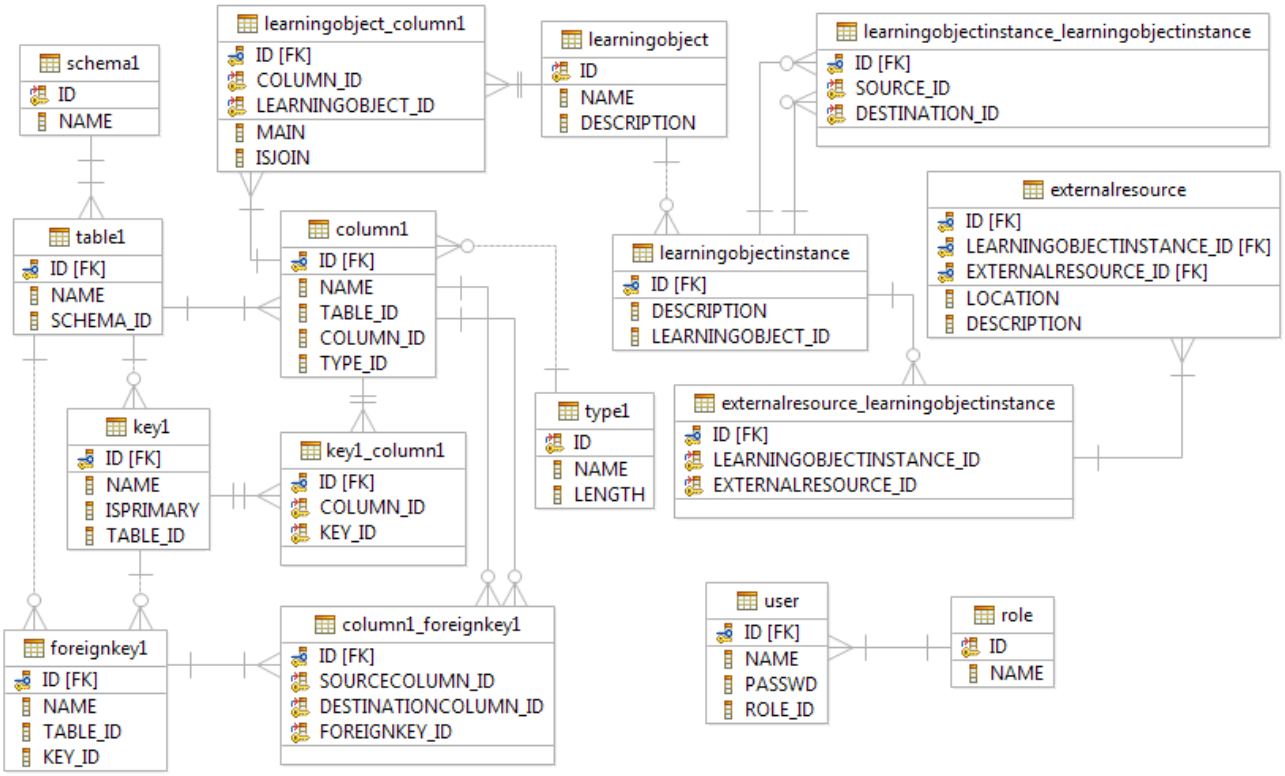


Figura 3.4.3.1 Detalle del modelo relacional

En el diagrama se observa las tablas propias de los objetos de negocio y las tablas externas necesarias para realizar algunas de las relaciones, como key1_column, column1_foreignkey1 o externalresource_learningobjectinstance.

3.5 Despliegue

Este diagrama de despliegue modela la arquitectura en tiempo de ejecución del sistema. Esto muestra la configuración de los elementos de hardware (nodos) y muestra cómo los elementos y artefactos del software se trazan en esos nodos.

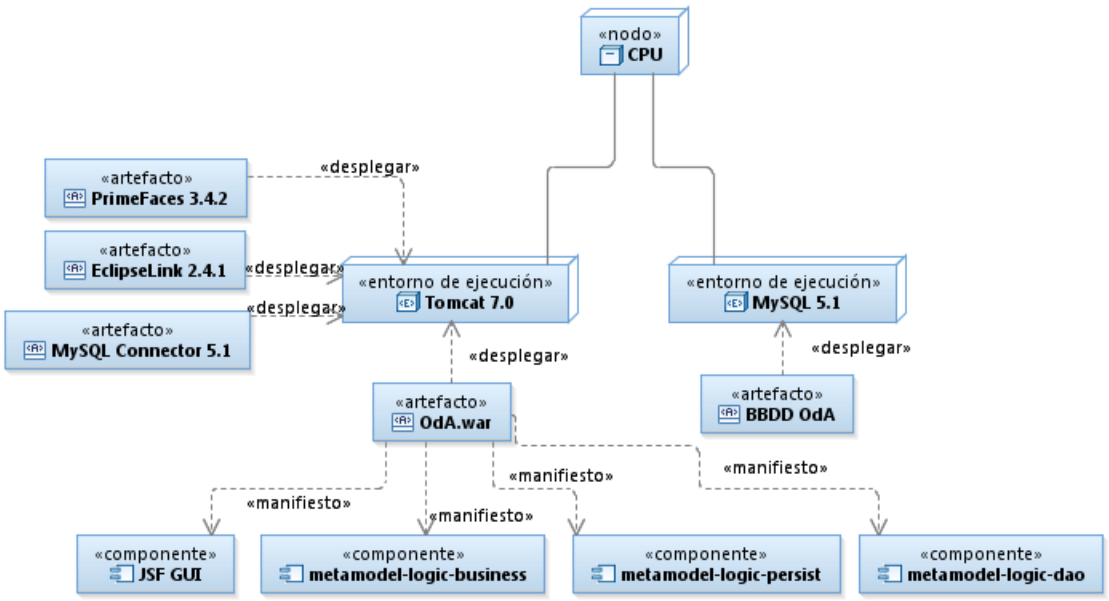


Figura 3.5.1 Detalle del diagrama de despliegue

Los entornos de ejecución necesarios para el despliegue son:

- Tomcat 7.0: En este entorno de ejecución desplegaremos la funcionalidad de la aplicación en sí, mediante un WAR que recoge la lógica de los componentes de negocio, dao, persistencia y vista. Estos componentes se ven realizados por los artefactos PrimeFaces 3.4.2, EclipseLink 2.4.1 y MySQL Connector 5.1.

- MySQL 5.1: En este entorno de ejecución desplegaremos la base de datos para la aplicación.

3.6 Algoritmo de generación dinámica de instancias de HLOs

El algoritmo es necesario para el manejo dinámico de los esquemas de metadatos dependientes del dominio.

Para el desarrollo del proyecto ha sido necesaria su implementación basada en que:

- A partir de un Objeto de Aprendizaje, de las columnas que lo componen y las instancias de datos que rellenan esas columnas. Con esta información y a través de las Foreign Keys que relacionan las tablas y las Columnas de Join, vamos asociando/vinculando los datos y generando las instancias totales.

- Una primera parte del algoritmo se dedica a la generación de estas instancias.

- La segunda parte del algoritmo se dedica a asociar a cada instancia un identificador, basándose en la Columna Directora del Objeto de Aprendizaje. Todas las instancias que tengan el mismo valor para la Columna Directora, tendrán el mismo identificador de instancia.

- Asociamos este identificador para poder, posteriormente, realizar una correcta visualización de los datos y agruparlos bajo la Columna Directora o Columna Main. Una Columna Directora es una columna elegida por el usuario cuando se crea un LO y a partir de la cual realizamos la fusión de los valores de las instancias para su visualización y almacenamiento. Cuando el valor de la columna directora es idéntico en cierto número de instancias, todas ellas llevarán el mismo identificador de instancia.

- Para la visualización de las instancias de un LO en Oda, en el momento que las instancias tengan el mismo valor para el Identificador de Instancia, se deja el valor de la columna directora (común a todas) y se agrupan el resto de valores de las columnas.

- La Columna Directora es la que sirve para agrupar las posibles tuplas repetidas que pudieran salir al calcular el contenido enriquecido del HLO. Así, si nos fijamos en la figura 3.3.1; y tenemos un LO compuesto por: site.id, intervention.date, artifact.id y archaeologist.id y la columna main es el id del artifact.id, podemos tener (Gizé, 03/03/2011, pulsera27, Ana), (Gizé, 01/02/2012, corona78,Ana) y (Gizé, 01/02/2012, corona78, Pepe), que darían las instancias de HLOs: {Gizé},{03/03/2011},pulsera27,{Ana} y ({Gizé},{01/02/2012},corona78,{Ana,Pepe}). Como se ve en la figura 3.6.

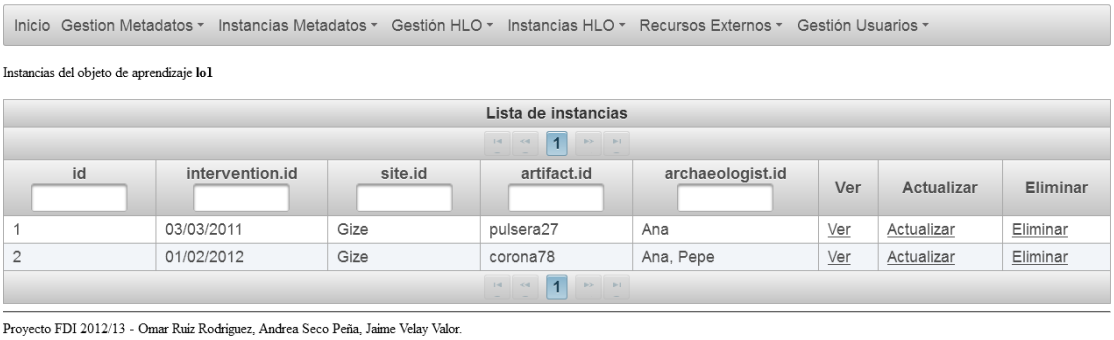


Figura 3.6. Visualización de las instancias de un LO.

Este algoritmo permite recalcular dinámicamente los metadatos de un HLO, ya que OdA permite tanto el cambio en la estructura de dichos metadatos, como en los valores de los mismos.

Así, por ejemplo, en el caso que presentamos, de excavaciones arqueológicas, los usuarios pueden definir información estructural y de instancia (lugares, objetos encontrados, participantes de la expedición…) y a partir de esta información, generar los contenidos de las instancias de los HLOs definidos según la estructura de HLOs. Si ya existieran instancias de HLO y se introducen datos relativos a él, en la siguiente generación de instancias, se anexaría dicha información, sin incluir ninguna instancia repetida perteneciente a datos antiguos que ya fueron procesados en su momento.

Además, se incluye la posibilidad de poder variar la estructura del LO, pudiendo incluir o descartar datos. Por ejemplo, fijándonos en la figura 3.3.1, podríamos tener un LO compuesto por el lugar de la excavación, los arqueólogos y los objetos descubiertos, y posteriormente decidir incluir la nacionalidad del arqueólogo. Al generar de nuevo instancias de ese LO ya modificado, el algoritmo incluiría la nueva información en nuevas instancias, y las anteriores no se eliminarían ya que pueden tener recursos externos asociados, como fotos, videos, sonidos, textos… y/o vínculos con otras instancias de otros Objetos de Aprendizaje.

Facilitamos tres versiones del algoritmo en pseudocódigo.

1) Simple.
2) Astuto.
3) Astuto v2.

1) Simple

- No permite modificar la estructura del Objeto de Aprendizaje.
- Calcular las instancias del Objeto de Aprendizaje.
    - Genera join.
    - Comprueba tuplas nuevas con tuplas antiguas.
        - Si ya hay ese valor para la columna Main: se tira la tupla.
        - Si no: se crea un nuevo Identificador de Instancia y se guarda.

2) Astuto

- No permite modificar la estructura del Objeto de Aprendizaje.
- Calcular las instancias del Objeto de Aprendizaje.
    - Genera join.
    - Comprueba tuplas nuevas con tuplas antiguas.
        - Si ya hay ese valor para la columna Main.
            - Si las tuplas son iguales, se tira la nueva.
            - Si no, se inserta luego esa tupla, respetando el Identificador de Instancia que se tenga para ese Main.
        - Si no: se crea un nuevo Identificador de Instancia y se guarda.

3) Astuto v2

- Se permiten modificaciones en la estructura del Objeto de Aprendizaje.
- Calcular las instancias del Objeto de Aprendizaje.
    - Genera join.
    - Comprueba tuplas nuevas con tuplas antiguas.
        - Si ya hay ese valor para la columna Main.
            - Se tira la tupla antigua y se guarda la nueva, respetando el Identificador de Instancia que se tenga para ese Main.
        - Si no: se crea un nuevo Identificador de Instancia y se guarda.

Para el proyecto hemos optado por la última versión, la más completa, que permite cambios tanto en la estructura del objeto de aprendizaje, como en los valores de sus instancias.

Capítulo 4. Caso de prueba

A continuación desarrollaremos un caso de prueba para describir el funcionamiento y la interfaz de usuario de la herramienta. Supongamos el esquema de bases de datos relacional descrito en la Figura 4.1, utilizado para caracterizar los metadatos dependientes del dominio de unos HLO relacionados con una excavación arqueológica (Navarro et al., 2003).

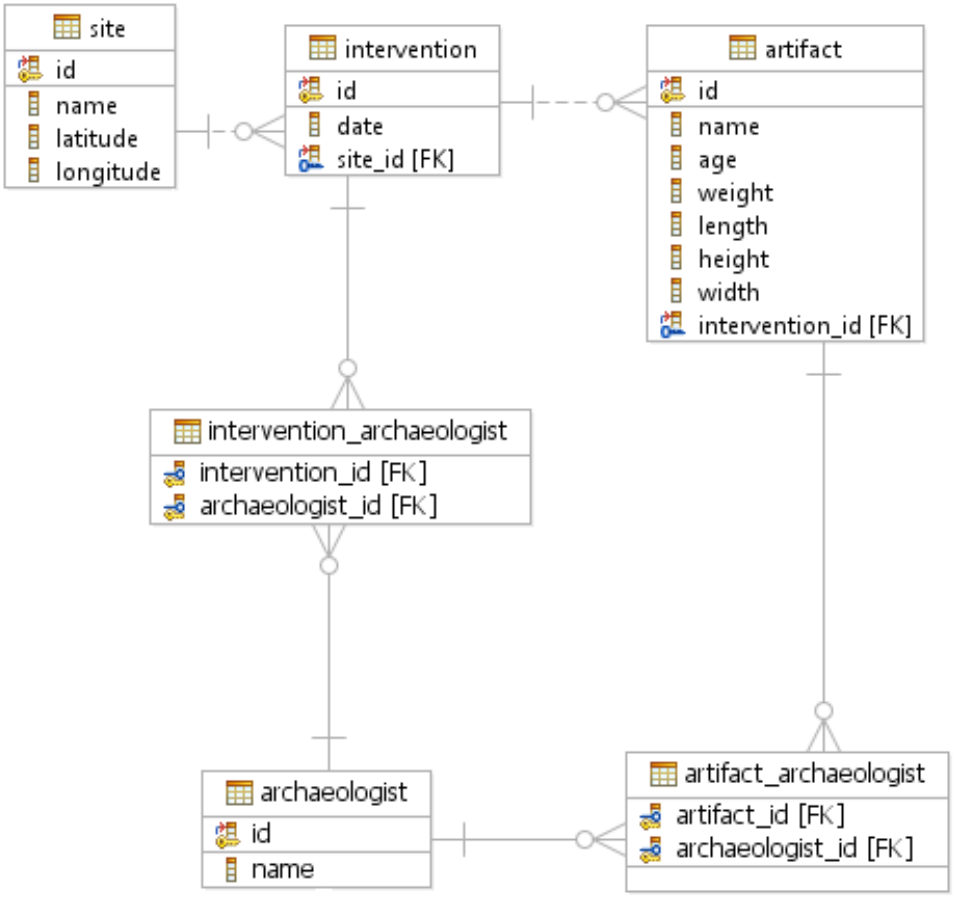


Figura 4.1. Esquema de base de datos relacional para el ejemplo

Se definiría con la aplicación OdaJ2EE del siguiente modo.

Creamos el esquema que contendrá toda la información, en nuestro caso se llamará “Arqueología”.

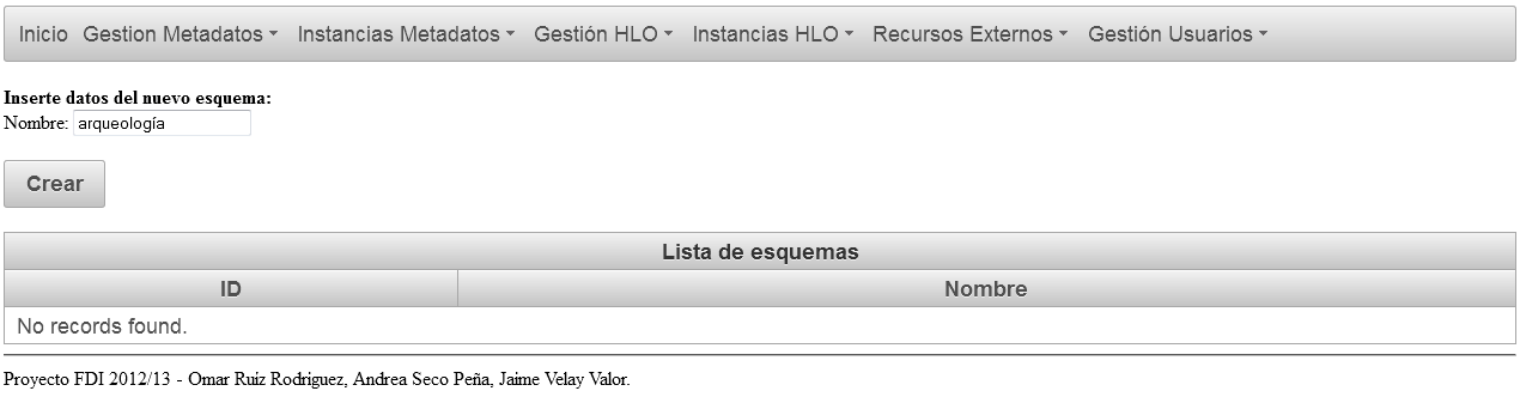


Figura 4.2. Creamos el esquema “Arqueología”

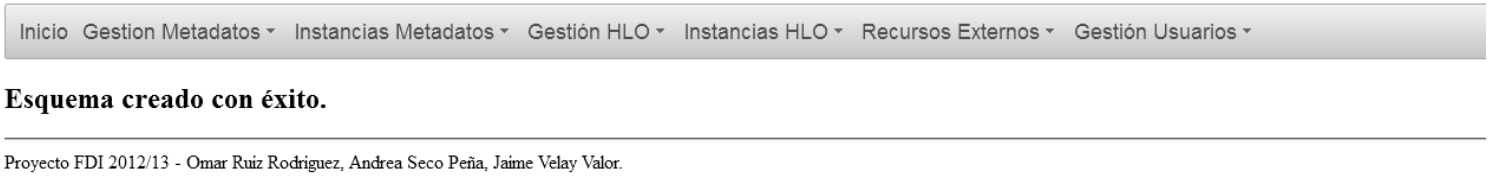


Figura 4.3. Pantalla de finalización al crear el esquema

Consultamos en la lista de esquemas el creado anteriormente.

Figura 4.4. La consulta de la lista de esquemas

La Figura 4.5 describe el aspecto de la BBDD después de crear el esquema.

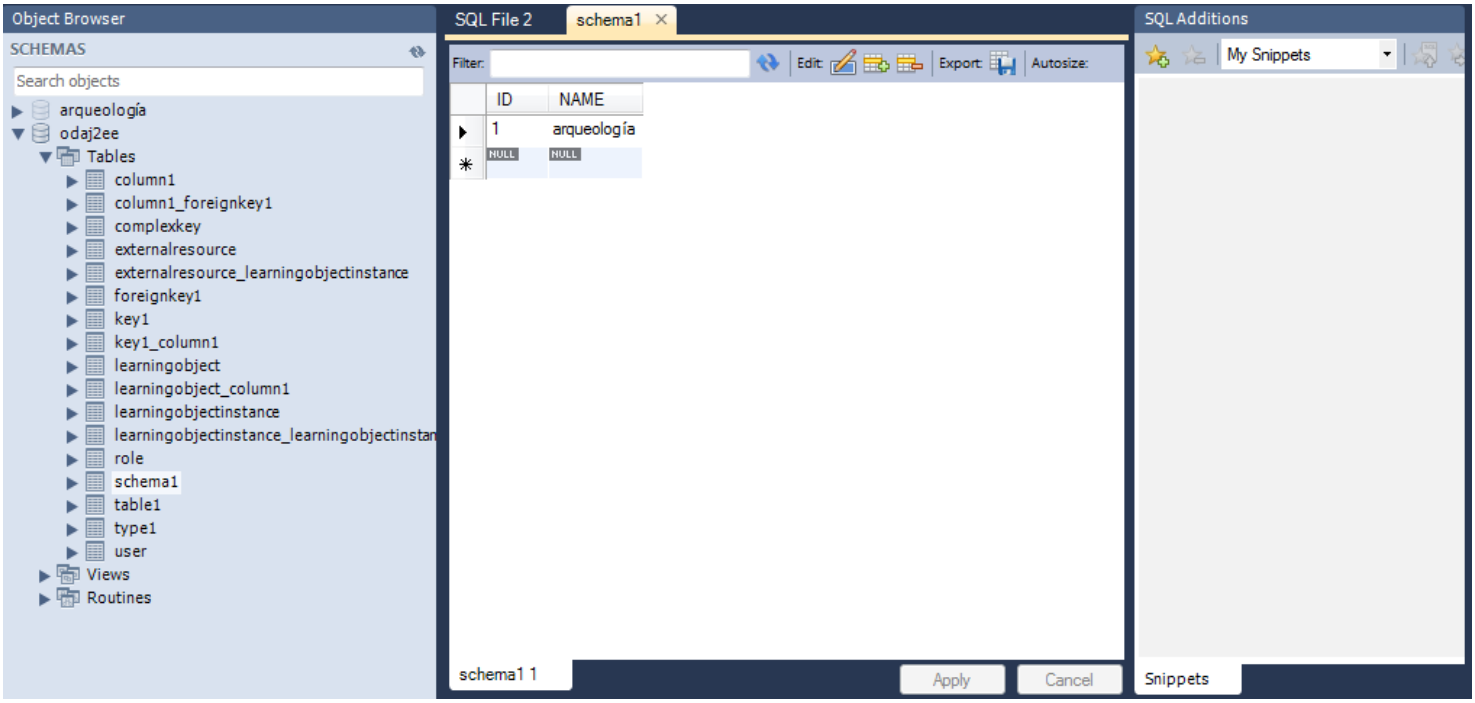


Figura 4.5. El esquema visto en MySQL

Creamos la tabla “Site” perteneciente al esquema “Arqueología”.

Figura 4.6. Creación de la tabla “Site” asociada al esquema anterior

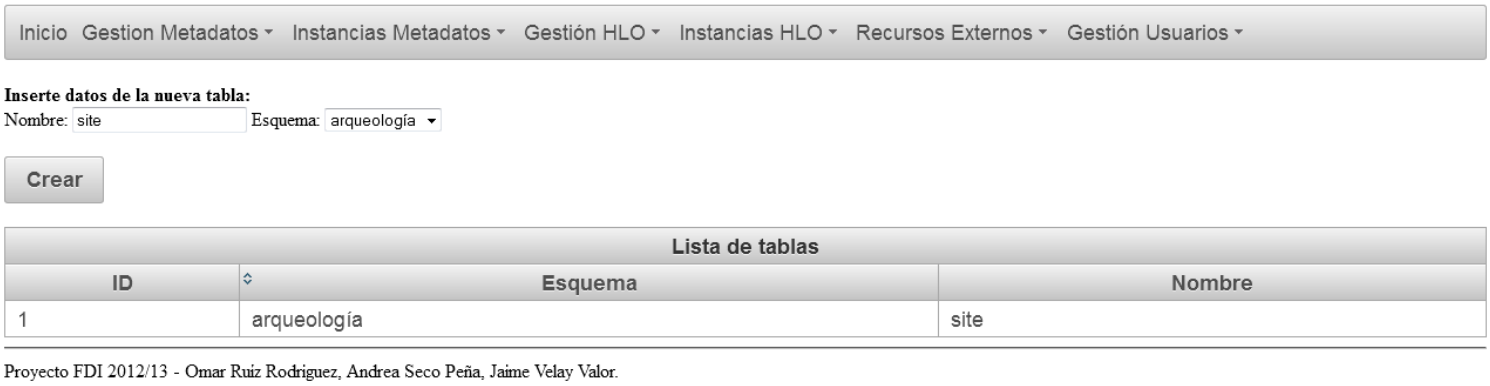


Figura 4.7. Continuación de la creación de la tabl

Figura 4.8. Pantalla de finalización al crear el tabla

La Figura 4.9 describe el aspecto de la BBDD después de crear la tabla “Site”.

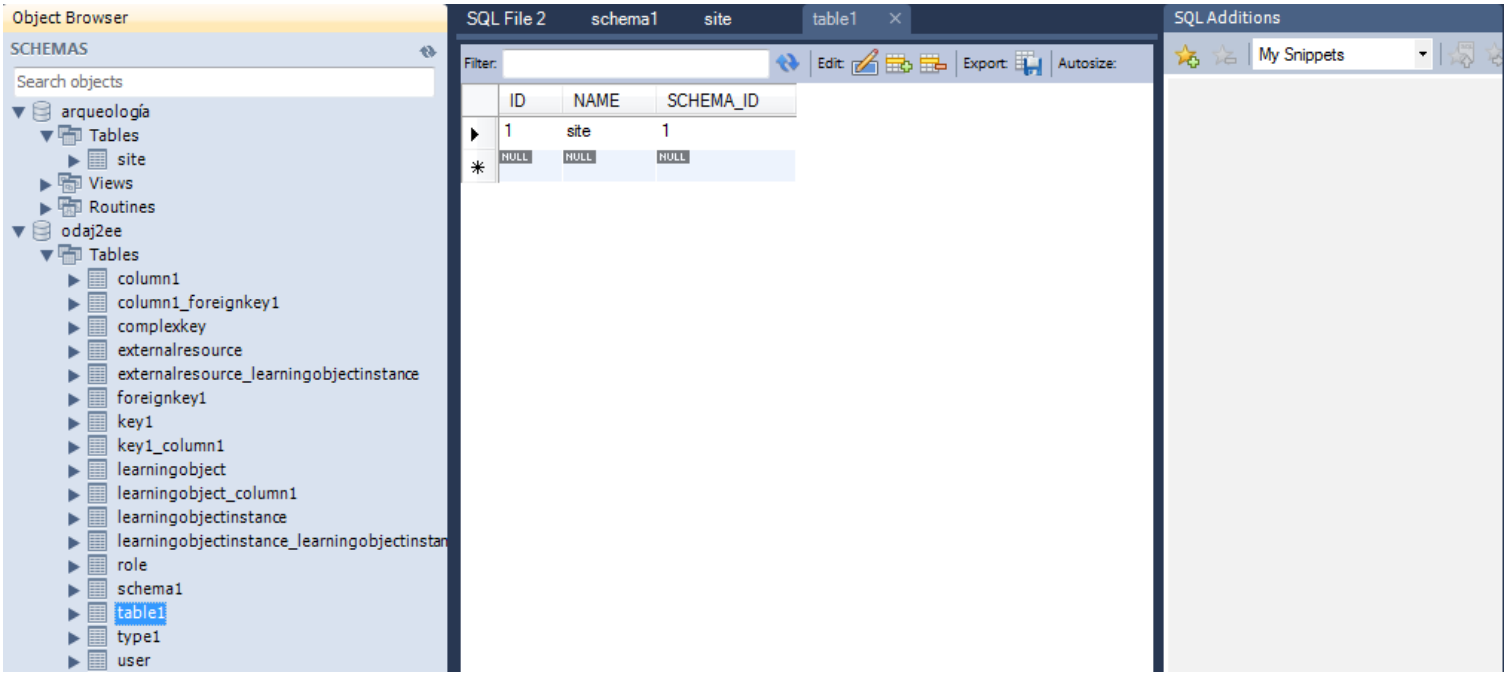


Figura 4.9. La tabla vista en MySQL

Añadimos la columna “Name” a la tabla “Site”.

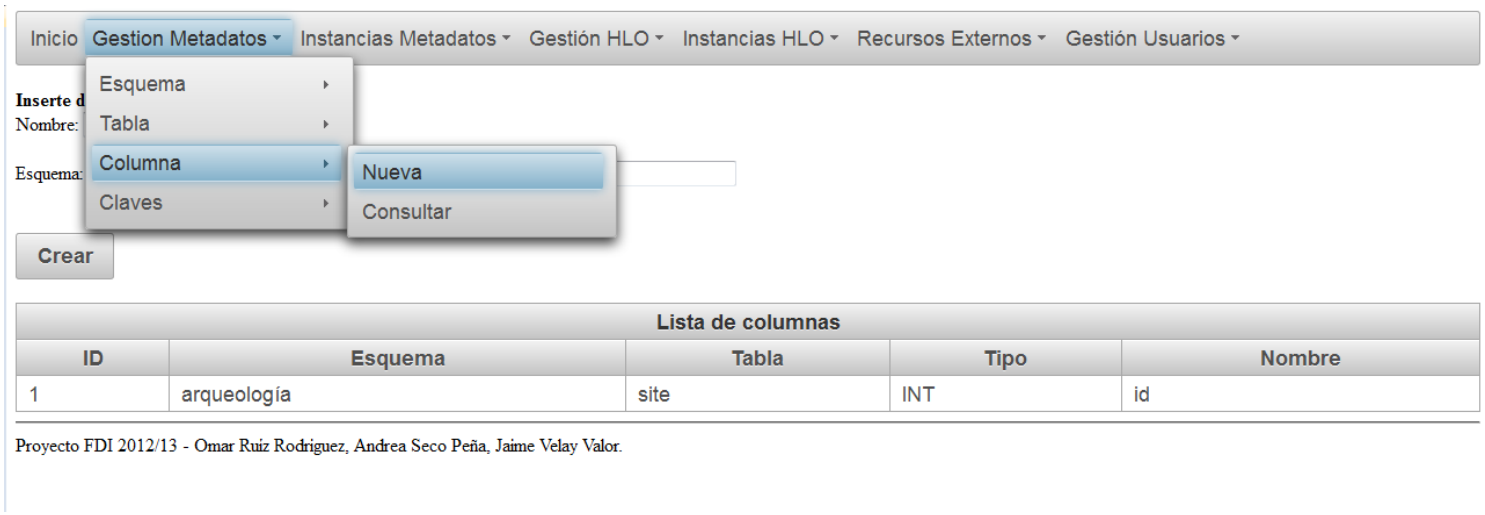


Figura 4.10. Creación de la columna “Name” para la tabla “Site” (I)

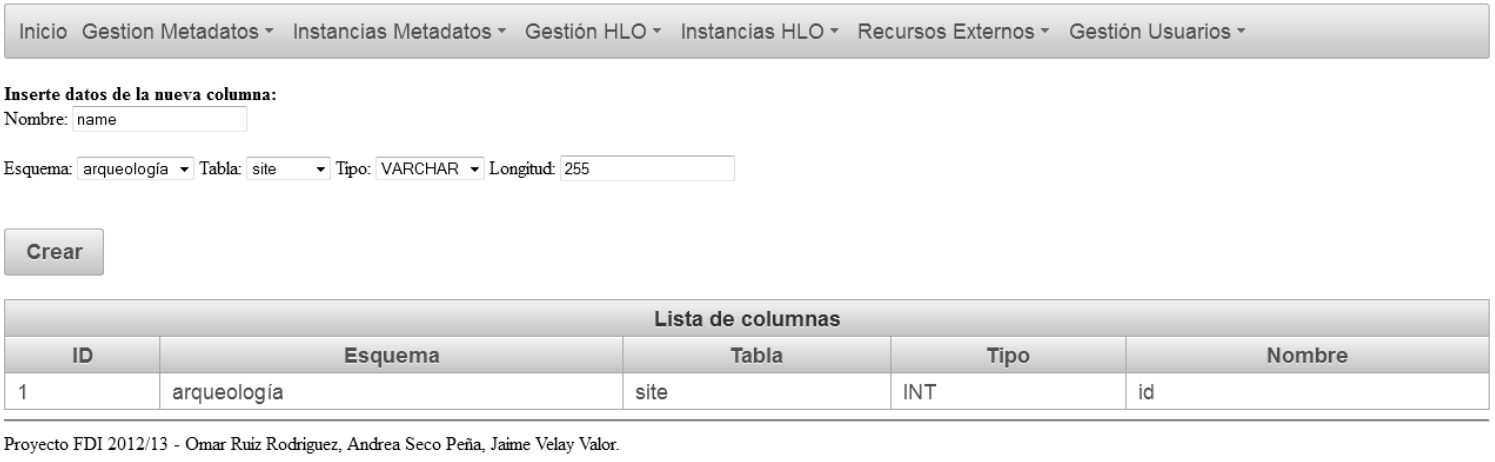


Figura 4.11. Creación de la columna “Name” para la tabla “Site” (II)

Figura 4.12. Pantalla de finalización al crear la columna

La Figura 4.13 describe el aspecto de la BBDD después de crear la columna “Name”.

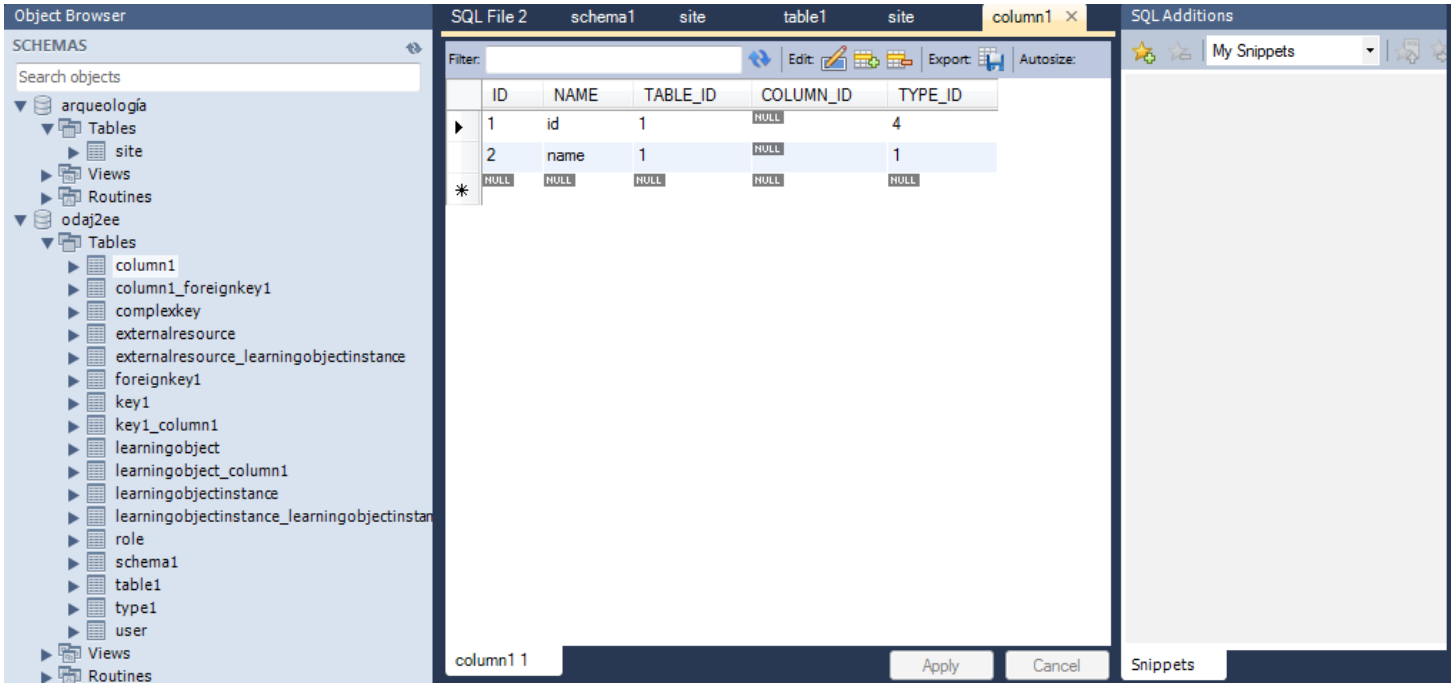


Figura 4.13. Las columnas en BBDD

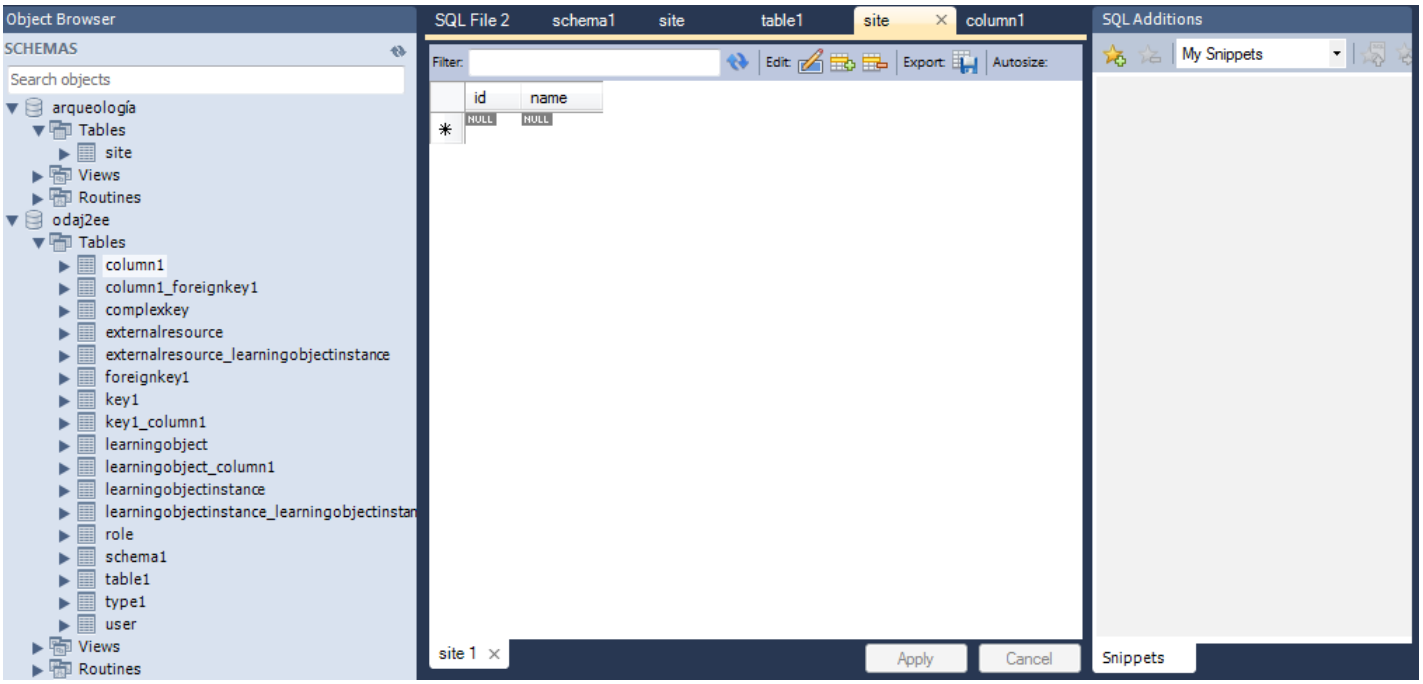


Figura 4.14. Las columnas de la tabla “Site” en BBDD

Actuamos igual con el resto de tablas que componen el modelo: “Intervention”, “Artifact”,“Archaeologist”,“Intervention_Archaeologist” y “Artifact_Archaeologist”.

La BBDD nos quedaría tal y como describe la Figura 4.15.

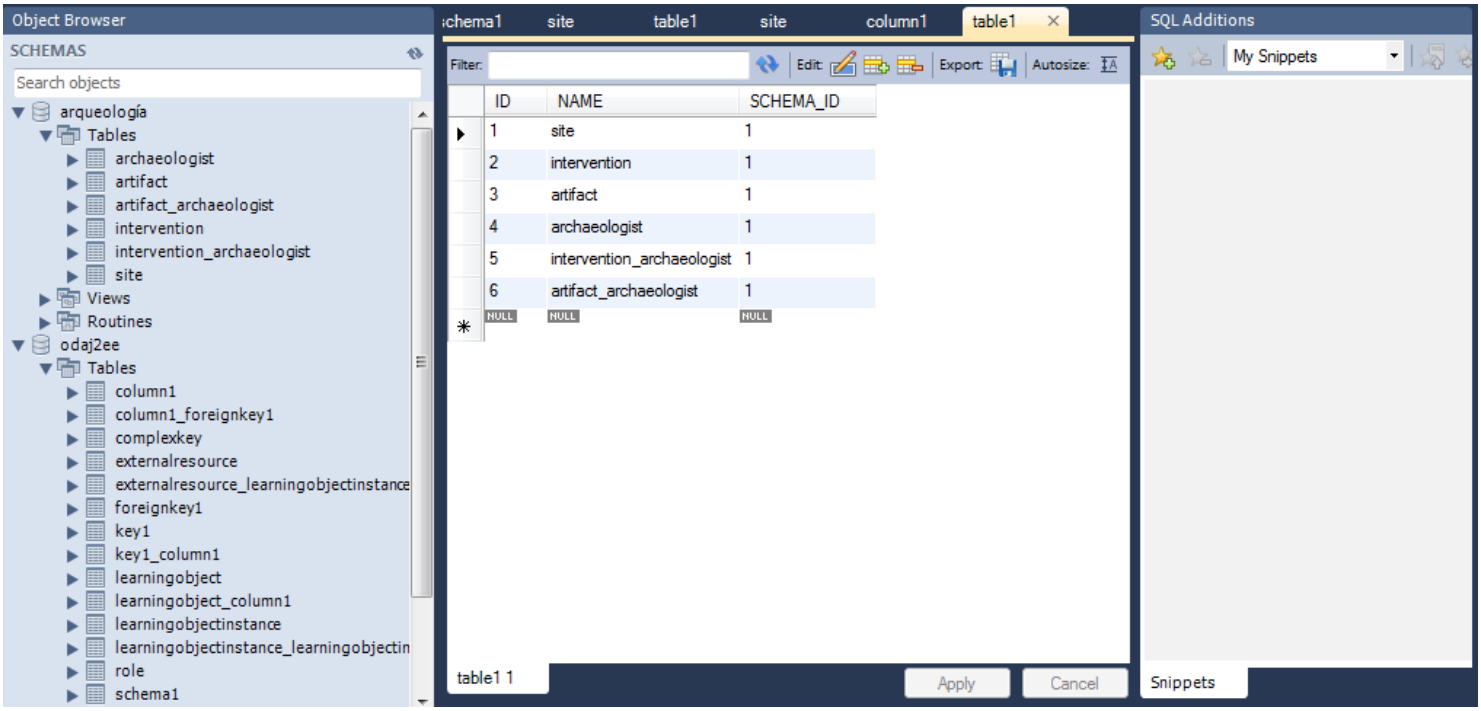


Figura 4.15. Las tablas totales de nuestro ejemplo en BBDD

Ahora establecemos las relaciones entre tablas mediante ForeignKeys:

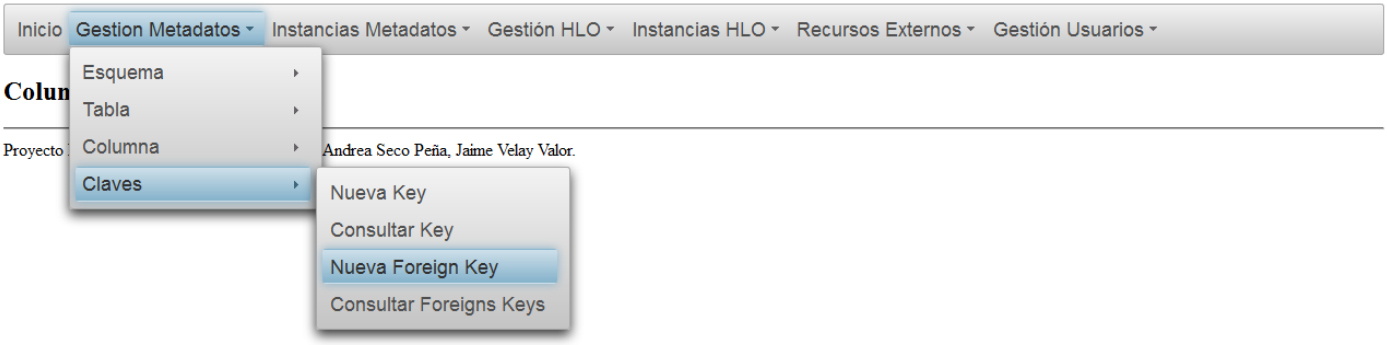


Figura 4.16. Creación de una Foreign Key

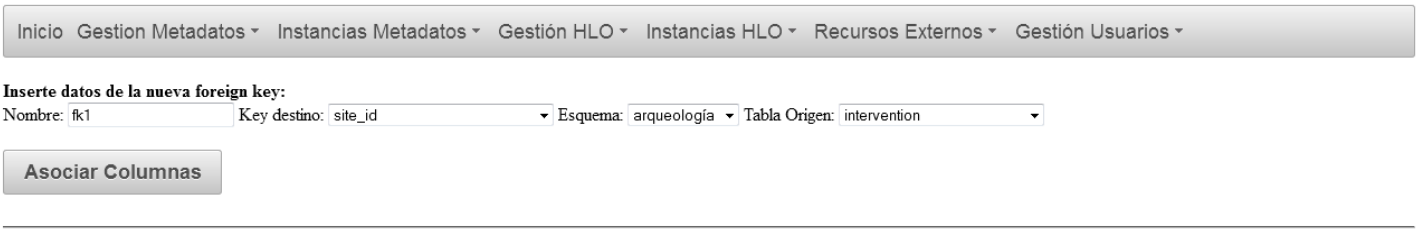


Figura 4.17. Asignación de propiedades a la Foreign Key

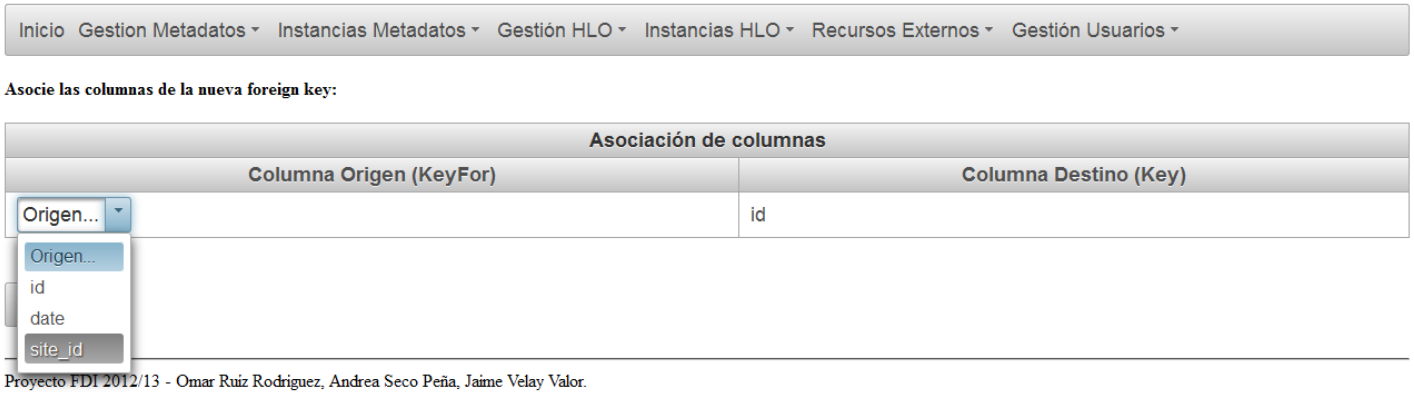


Figura 4.18. Asignación de columnas origen y destino a la Foreign Key

Figura 4.19. Pantalla de finalización de la creación de la Foreign Key

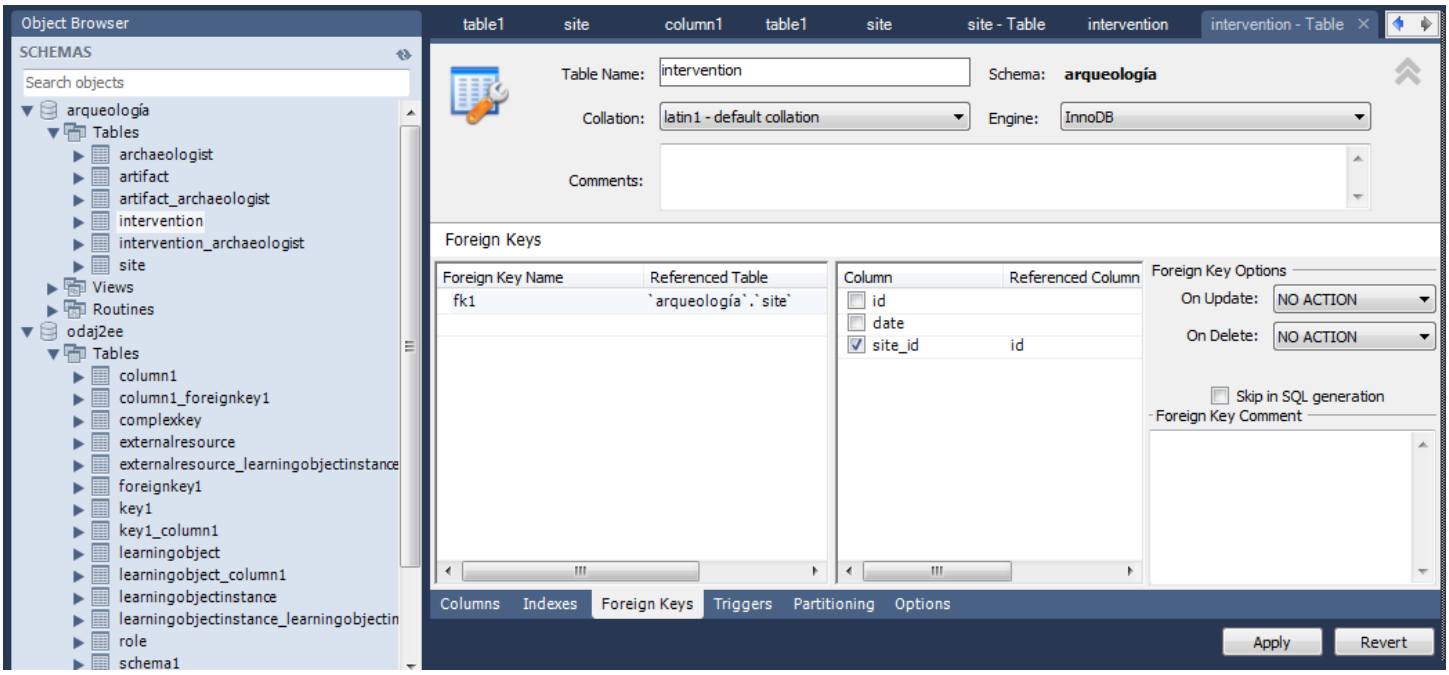


Figura 4.20. Visualización de la Foreign Key en BBDD

Actuamos de manera similar con el resto de foreign keys del modelo.

A continuación, generamos instancias de las tablas.

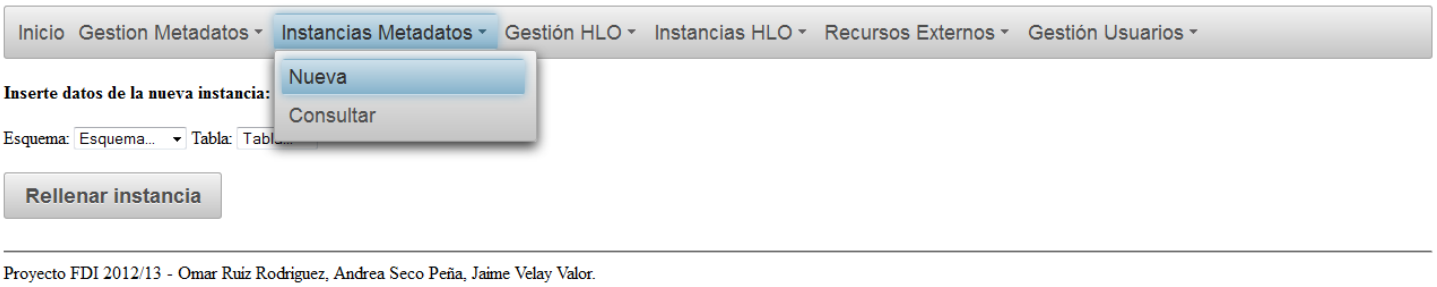


Figura 4.21. Creación de Instancias para las tablas

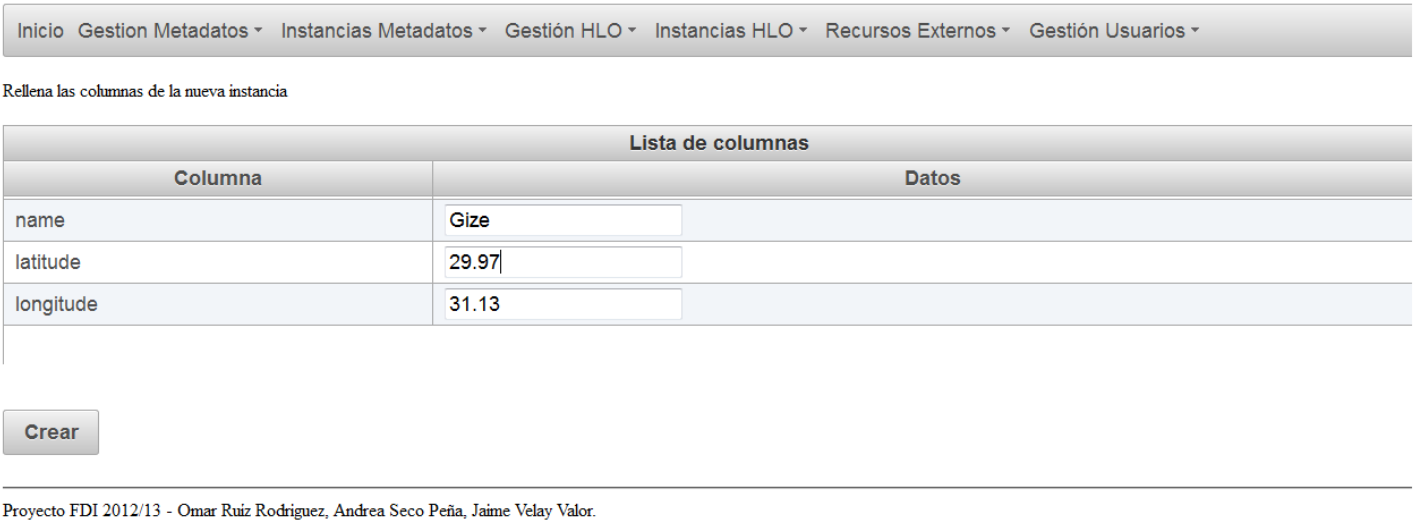


Figura 4.22. Introducción de datos para la tabla "Site"

Figura 4.23. Pantalla de finalización de la creación de la instancia

- Consultamos las instancias.

Figura 4.24. Visualización de las instancias

La BBDD quedaría según describe la Figura 4.25.

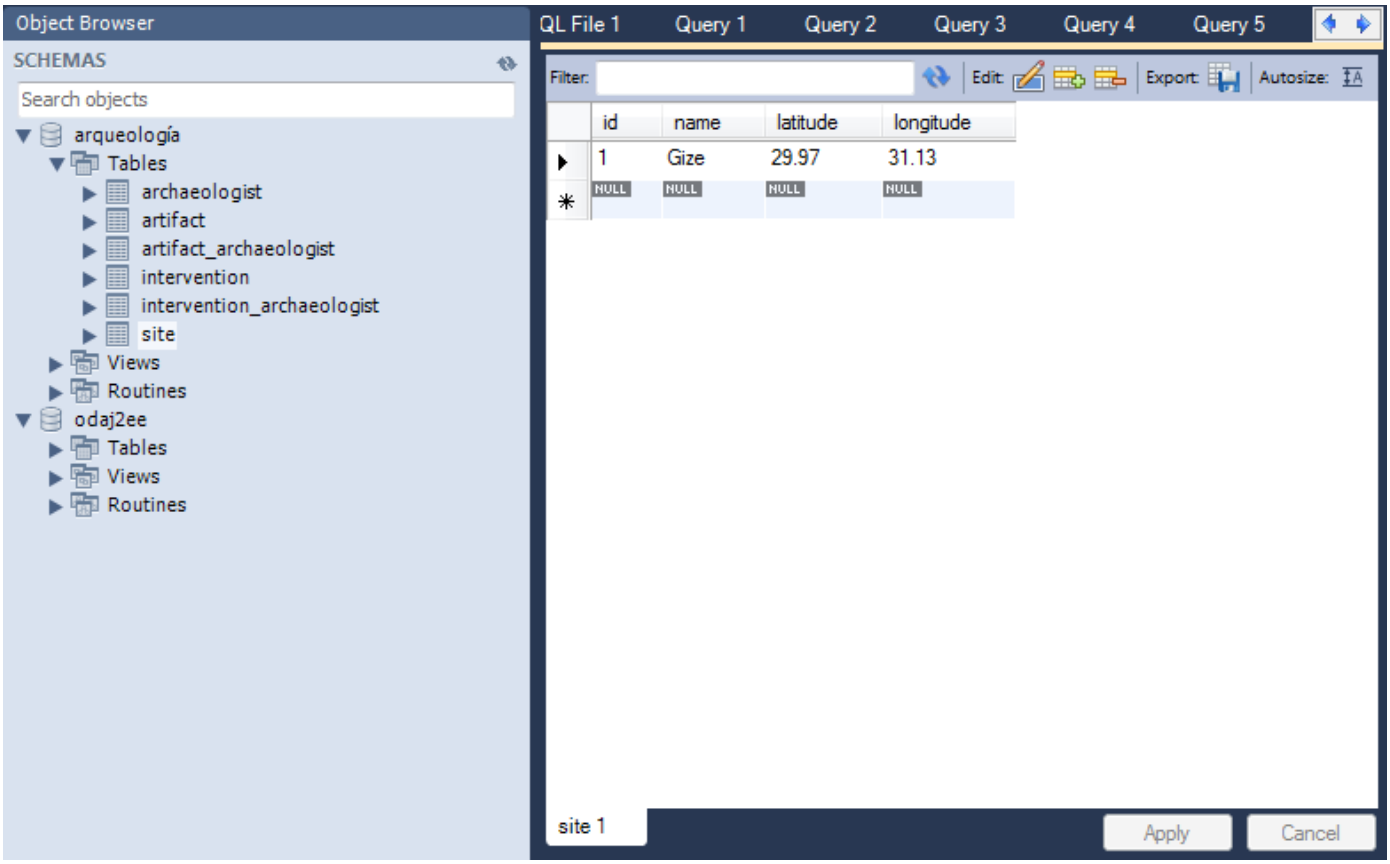


Figura 4.25. Las instancias en BBDD

De manera similar, lo llevamos a cabo para todas las instancias de las tablas que deseemos incluir.

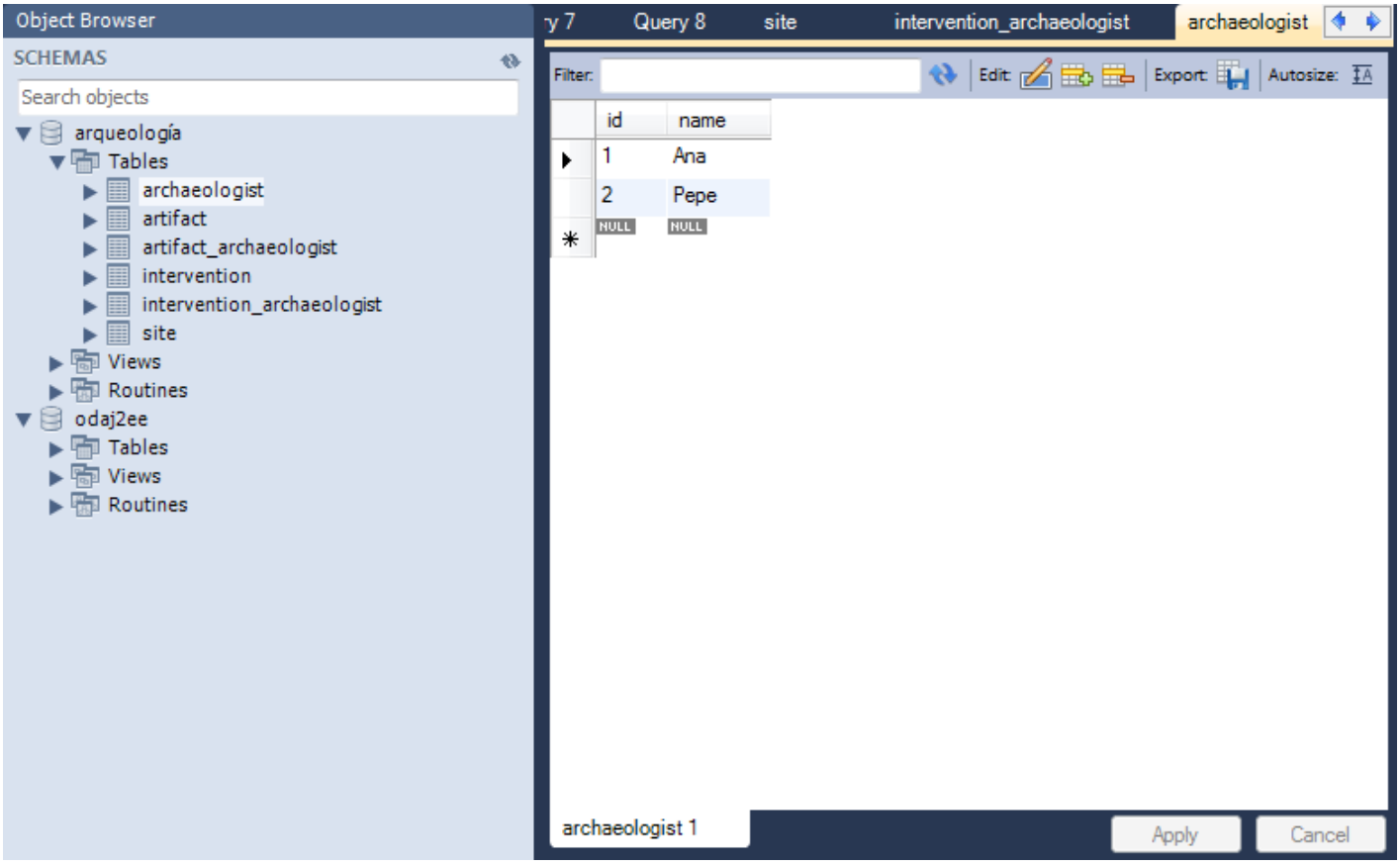
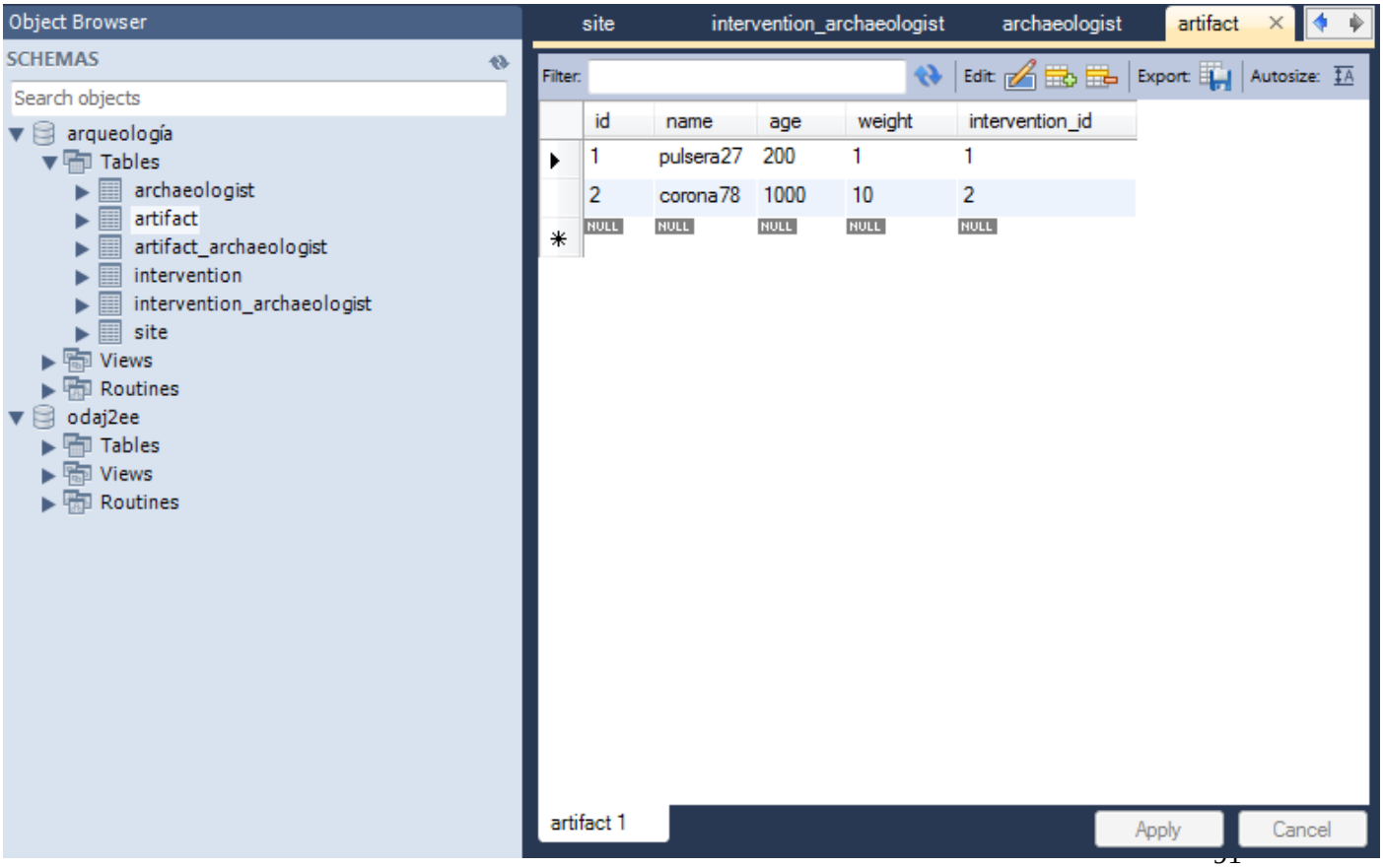


Figura 4.26. Instancias de la tabla “Archaeologist” en BBDD



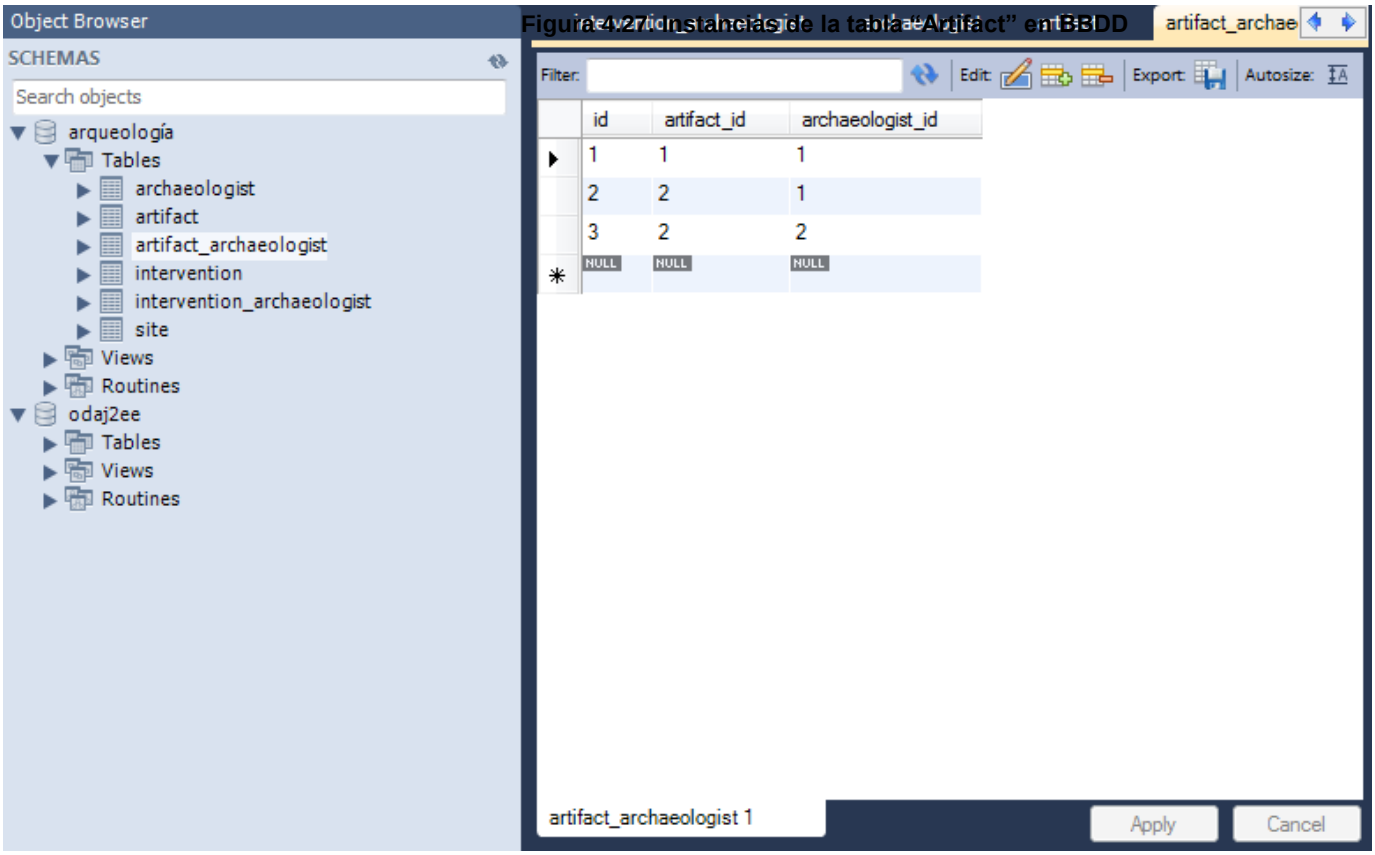
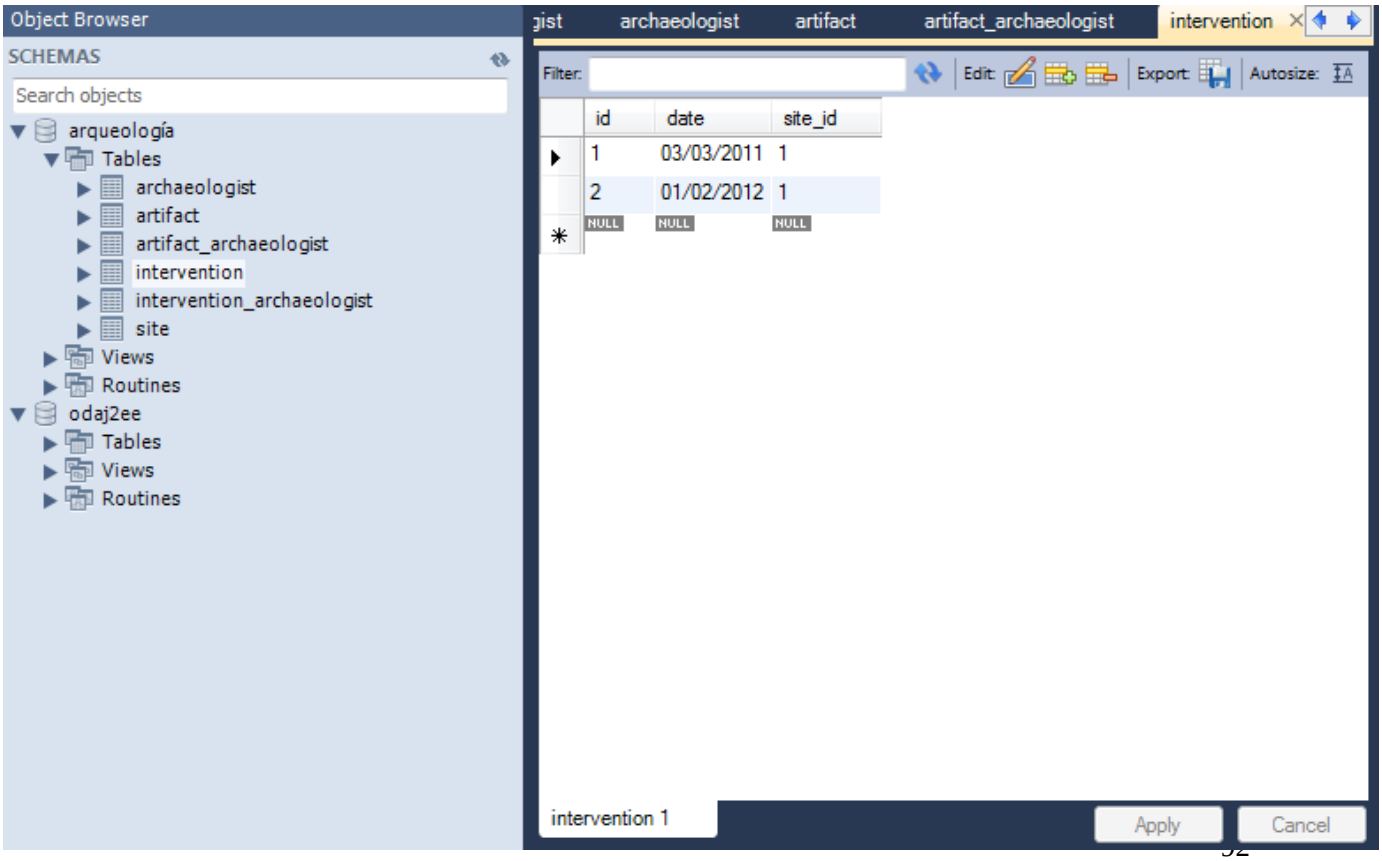


Figura 4.28. Las instancias de la tabla "Artifact_Archaeologist"



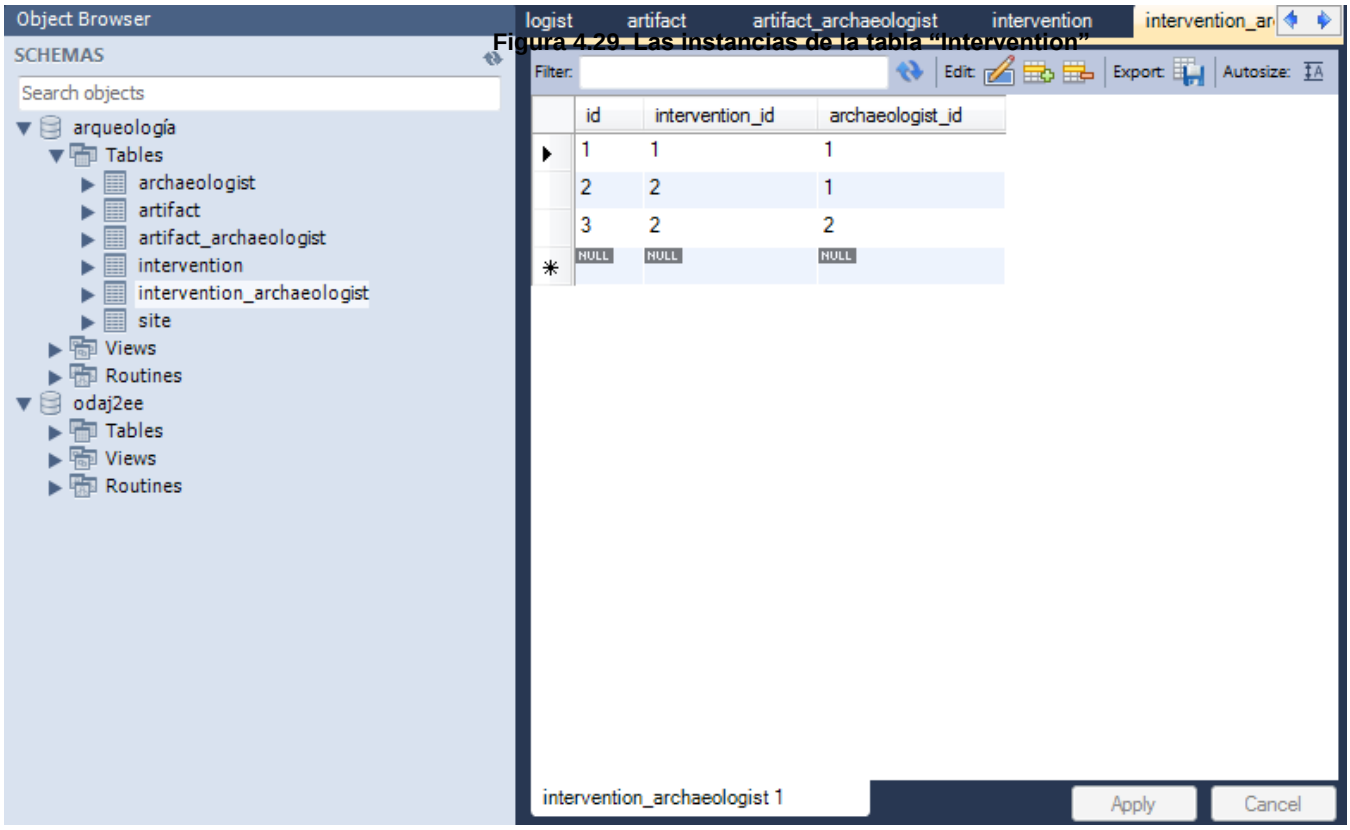


Figura 4.29. Las instancias de la tabla "Intervention"

Figura 4.30. Las instancias de la tabla "Intervention_Archaeogist"

Creamos un Objeto de Aprendizaje nuevo:

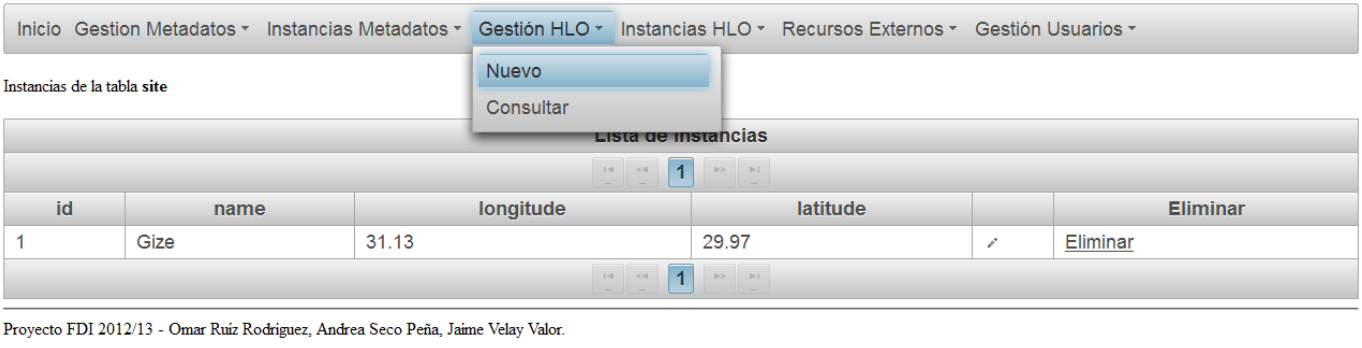


Figura 4.31. Creación de un objeto de aprendizaje

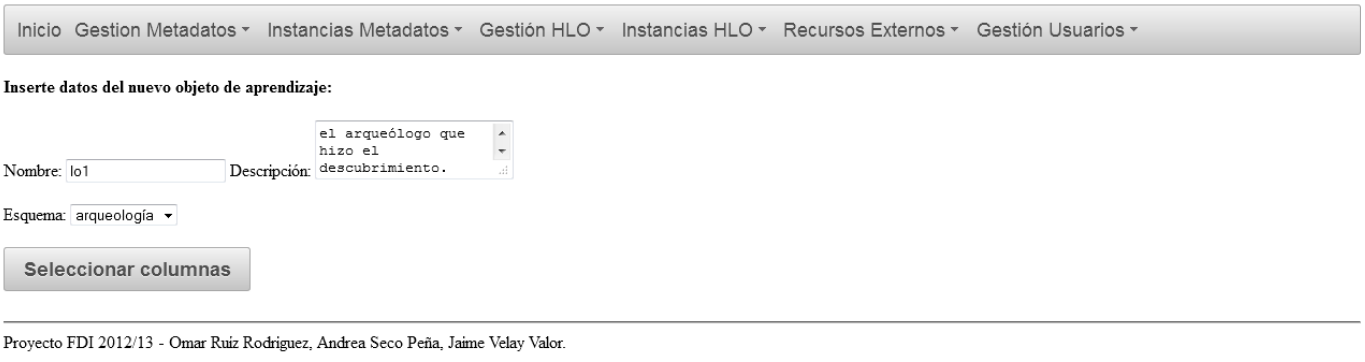


Figura 4.32. Incorporación de los datos del Objeto de Aprendizaje

Elegimos las columnas que forman el LO.

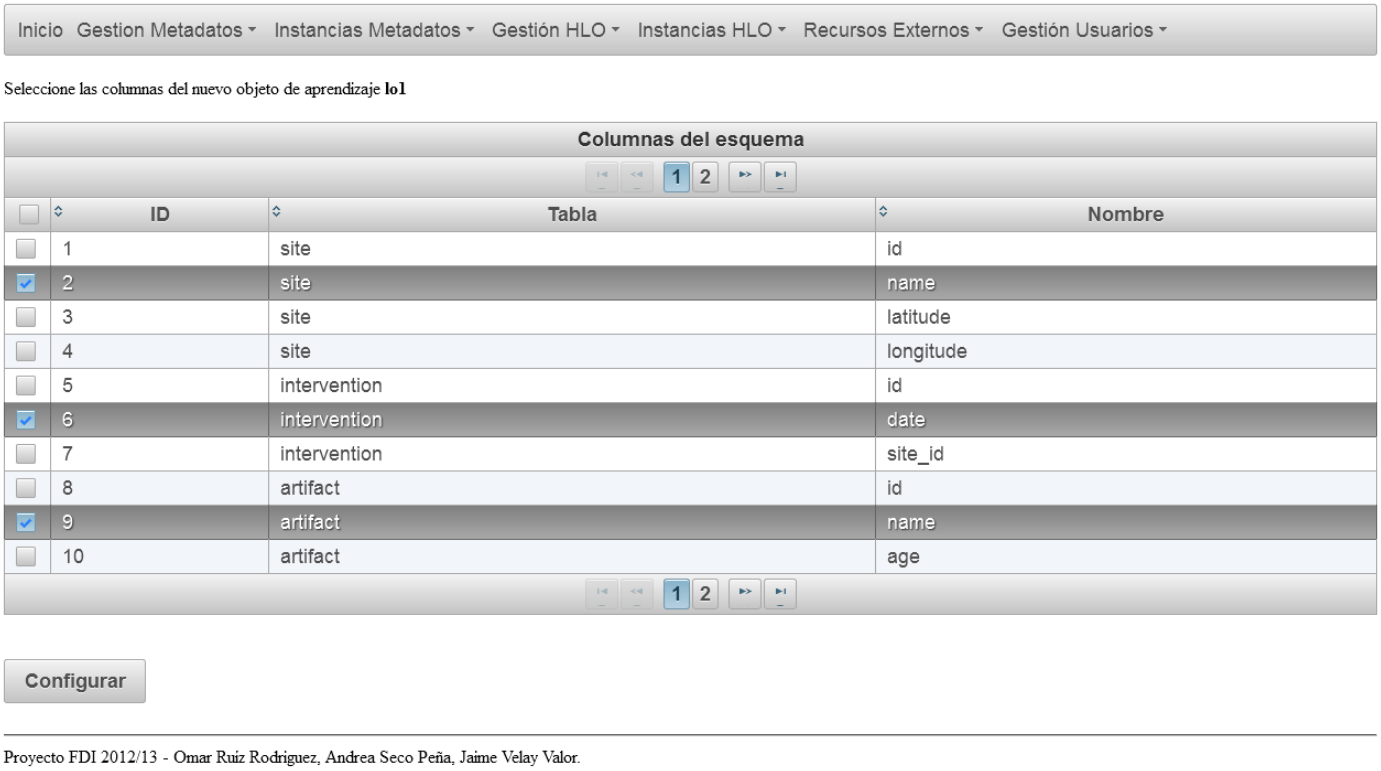


Figura 4.33. Selección las columnas que componen el LO (I)

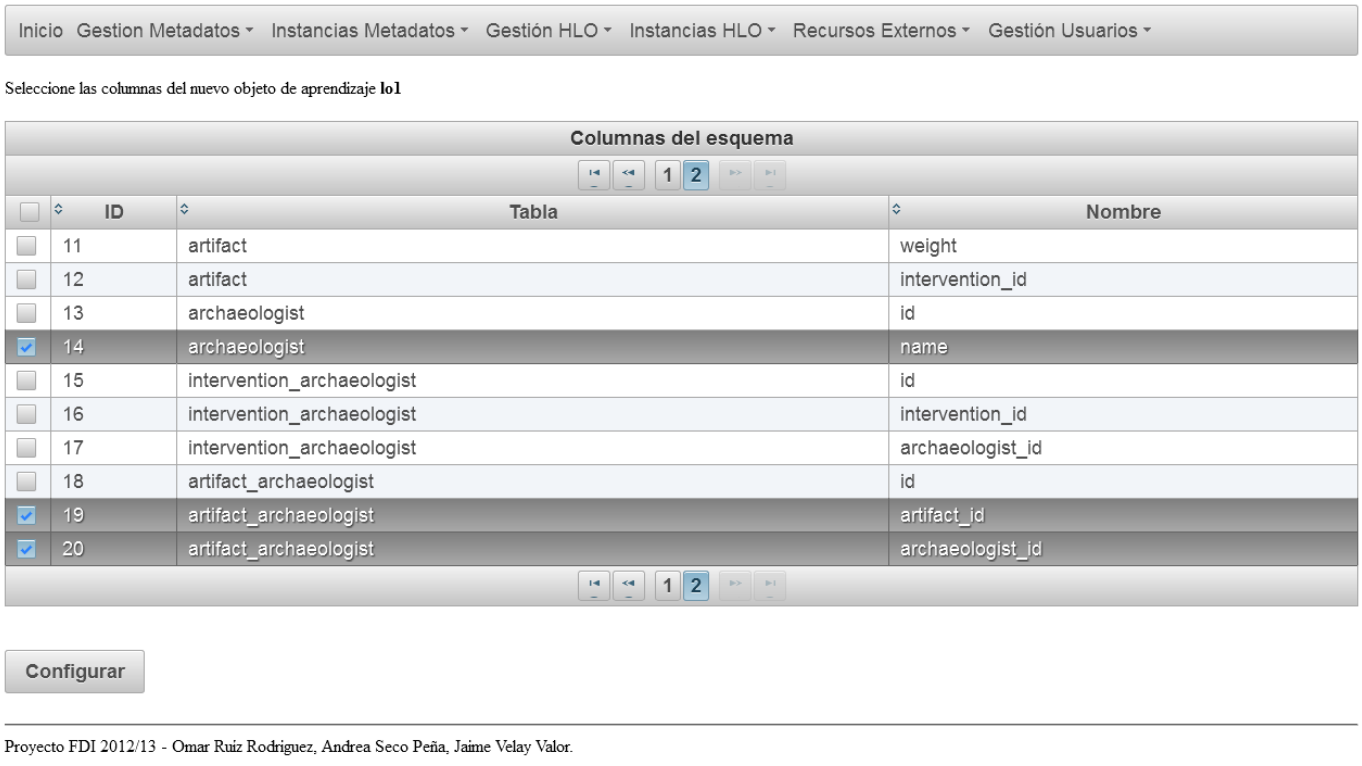


Figura 4.34. Selección de las columnas que componen el LO (II)

Elegimos las columnas de JOIN y la columna MAIN (columna directora).Las columnas de JOIN son a través de las cuales hacemos la unión de las tuplas. En nuestro caso unimos a través de Intervention y Archaeologist

La columna MAIN será a partir de la cual se designarán las instancias, cada dato diferente en la columna main conllevará la creación de una nueva instancia.

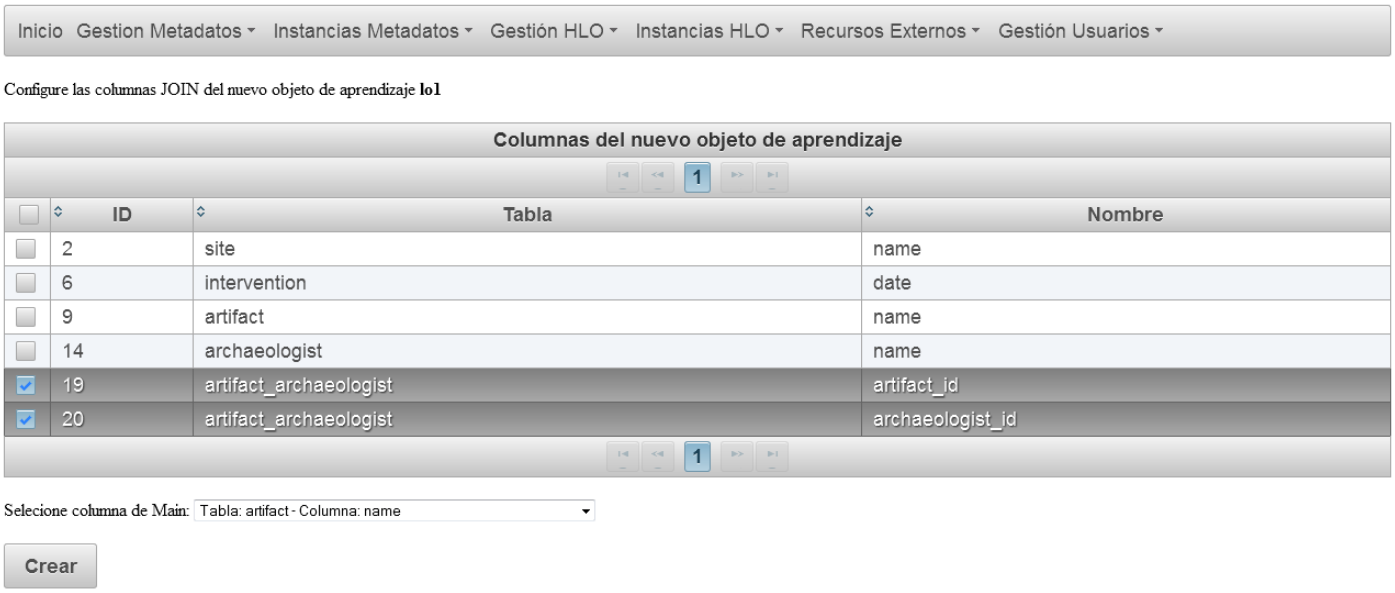


Figura 4.35. Selección de las columnas que de JOIN componen el LO

Figura 4.36. Pantalla de finalización de la creación del LO

Consultamos el Objeto de Aprendizaje creado.

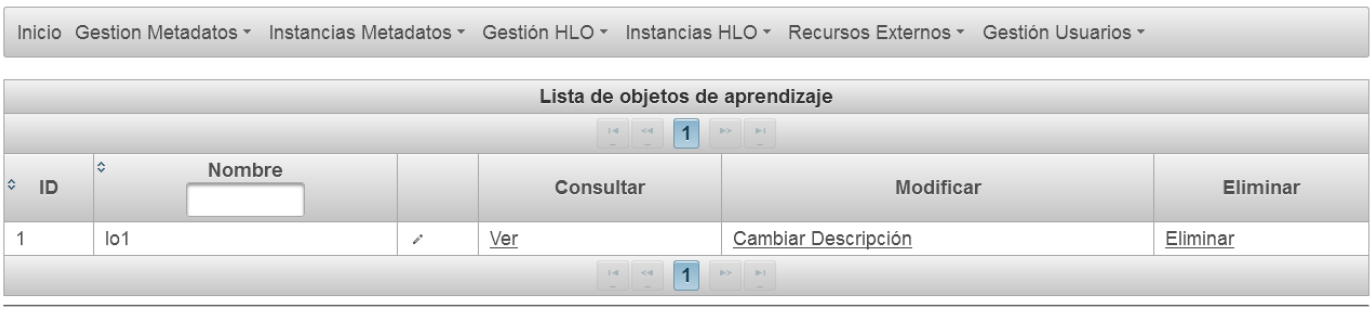


Figura 4.37. Consulta del LO

Generación de instancias del Objeto de Aprendizaje.

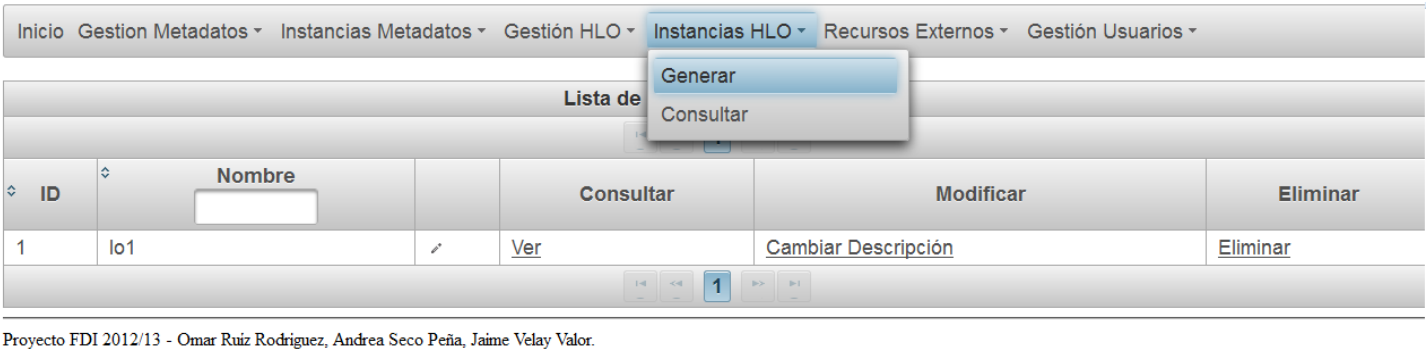


Figura 4.38. Generación de instancias del LO

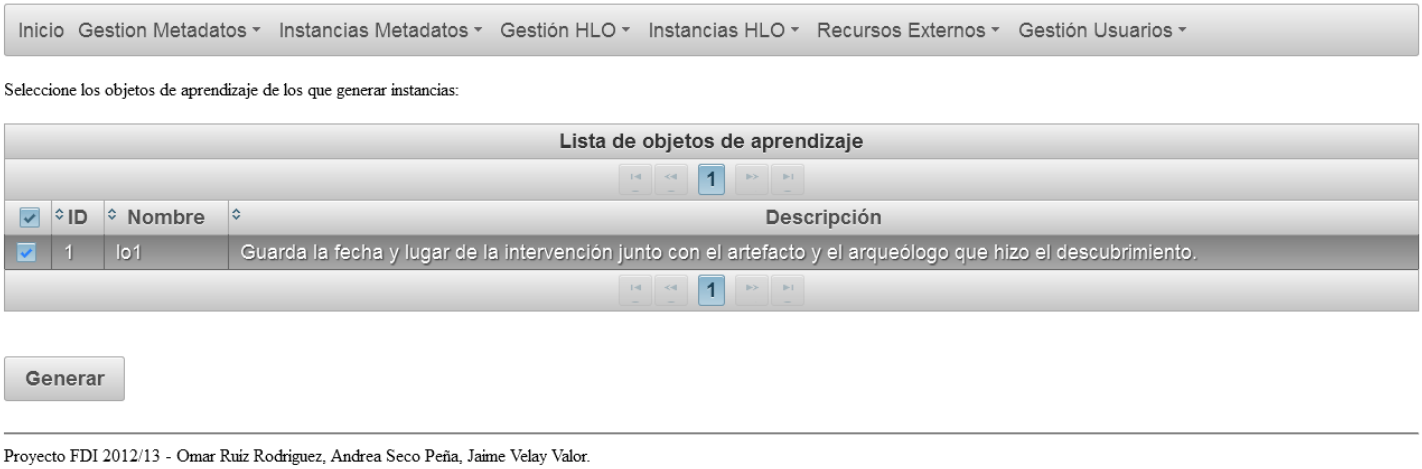


Figura 4.39. Continuaciónde la generación de instancias del LO

Figura 4.40. Finalización de la generación de instancias del LO

Consultamos las instancias que acabamos de generar.

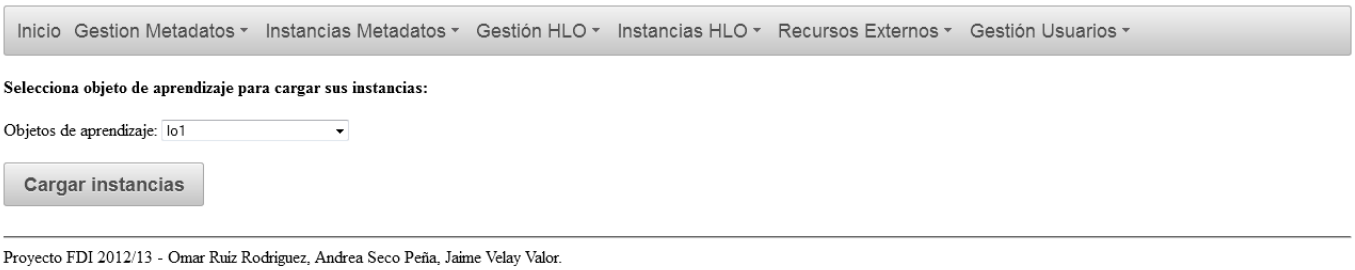


Figura 4.41. Consultamos las instancias recién generadas del LO

En la Figura 4.4.2 se observan las instancias generadas.

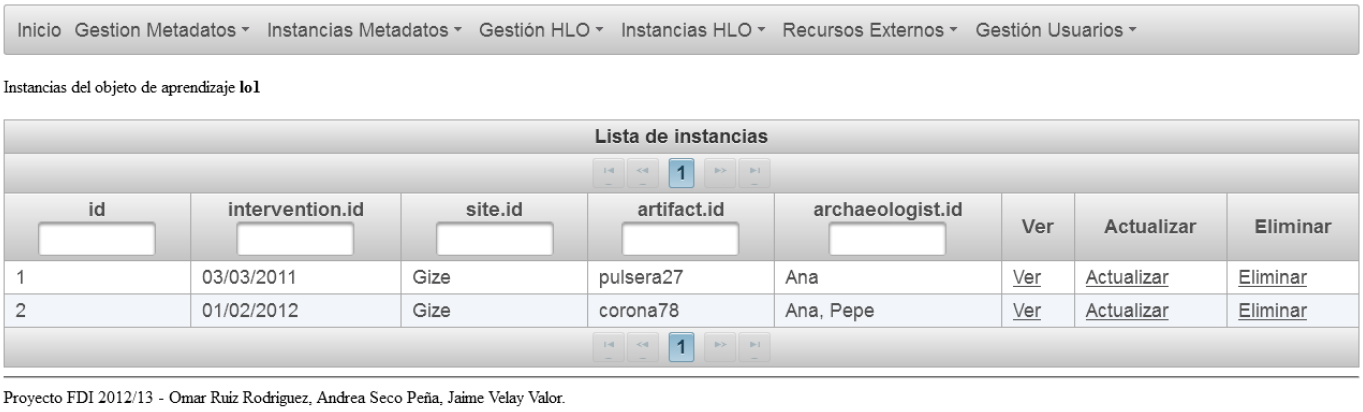


Figura 4.42. Visualización de las instancias recién generadas del LO

Podemos filtrar tuplas por cualquiera de las columnas o por una combinación de ellas, permitiéndose la navegación por HLOs a través de sus metadamos.

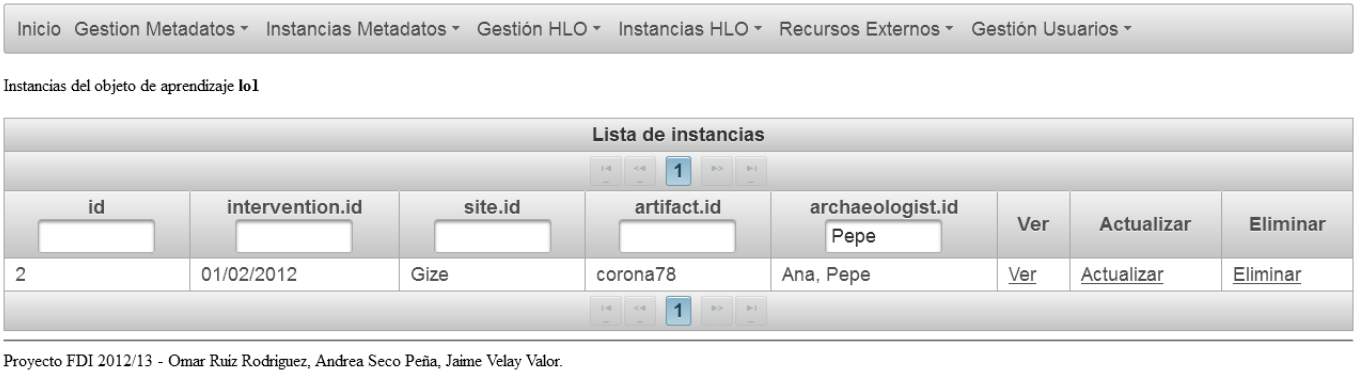


Figura 4.43. Filtrado de las tuplas por la columna Archaeologist.

La BBDD presenta un aspecto como el descrito en la Figura 4.44. Se ha creado la tabla Tuple_1 (Correspondiente al Objeto de Aprendizaje con número de identificación 1).

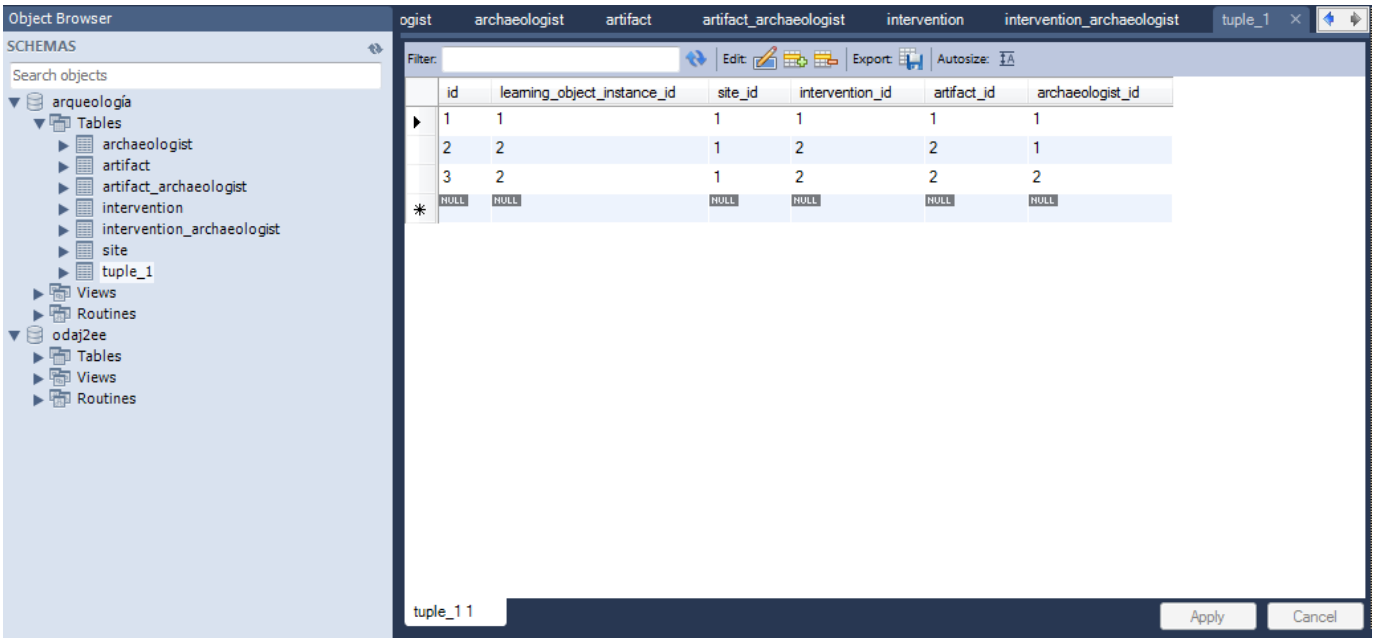


Figura 4.44. Visualización de las instancias recién generadas del LO en BBDD

Subimos un recurso externo.

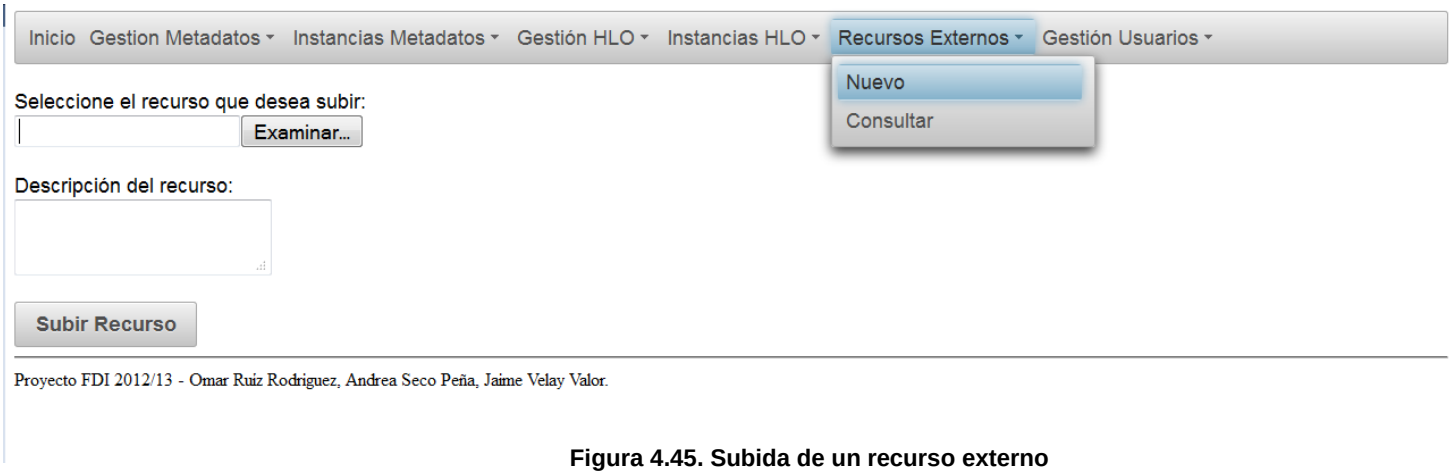


Figura 4.45. Subida de un recurso externo

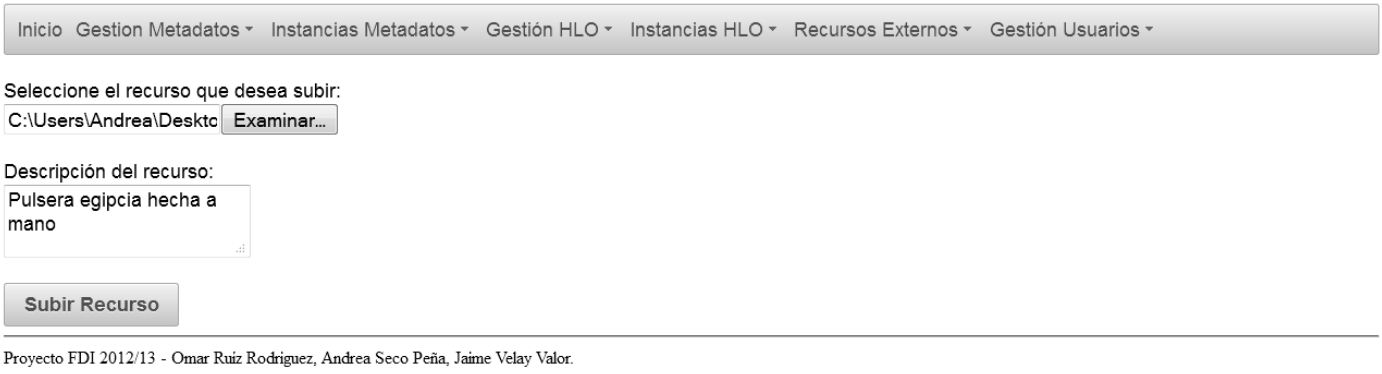


Figura 4.46. Continuación de la subida de un recurso externo

La consulta de una instancia de un Objeto de Aprendizaje

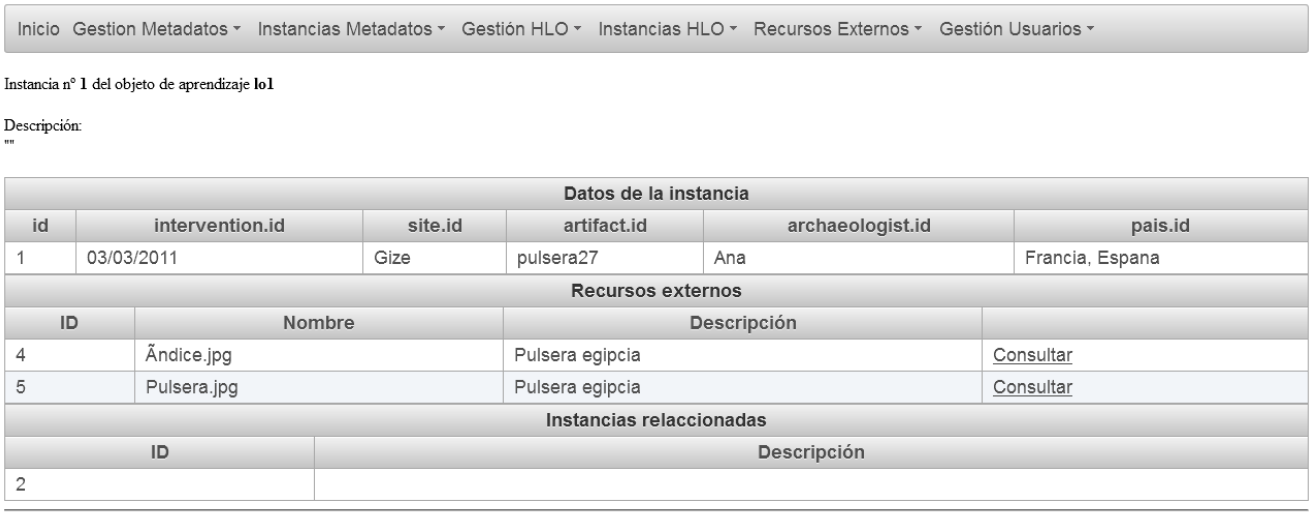


Figura 4.47. Consulta de una instancia generada del LO

La consulta de uno de sus recursos externos vinculados.

Figura 4.48. Visualización del recurso externo asociado a la instancia anterior

Capítulo 5. Conclusiones y trabajo futuro

5.1 Conocimiento Adquirido

La realización del trabajo de fin de grado demuestra la consecución de los objetivos perseguido por la titulación de graduado en Ingeniería del Software. Nos ha permitido aplicar los conocimientos teóricos y prácticos adquiridos en diversas asignaturas a lo largo de toda la carrera de manera satisfactoria.

Los diferentes marcos J2EE, utilizadas (JSF, JPA, JAAS), así como los patrones de arquitectura multicapa que hemos utilizado aplicado en el trabajo de fin de grado nos van a permitir destacar nuestro perfil profesional en el sector de las Tecnologías de la Información y nos han proporcionado una sólida base para que podamos seguir formándonos y también poder aplicar nuestros conocimientos en el ámbito empresarial.

La realización del proyecto nos ha servido como una gran experiencia en cuanto a gestión de proyectos. Ha sido clave encontrar un buen modelo de trabajo donde compatibilizar nuestros esfuerzos adaptándolo a las distintas situaciones laborales y superar con éxito las complicaciones que un proyecto de esta envergadura conlleva. La gestión del equipo ha presentado retos donde superarlos nos ha aportado tanto madurez profesional como personal.

El desarrollo del trabajo y las características del mismo nos han permitido formarnos profesionalmente de manera que podamos desempeñar eficazmente los conocimientos técnicos adquiridos con un amplio abanico de habilidades instrumentales y de relación interpersonal.

5.2 Trabajo Futuro

El trabajo futuro incluye la integración de OdA J2EE con diferentes LMSs. Esta característica permitirá a los usuarios de LMSs buscar LOs directamente en el LOR. Los LOs entonces se podrán publicar automáticamente en el LMS.

Para permitir la importación y exportación de los LOs, OdAJ2EE incluirá *IMS Content Packaging* (referencia), que define un formato digital estándar para recolectar y empaquetar contenidos educativos en formato digital.

Como mejora visual, la interfaz básica actual puede ser refinada. Tanto estéticamente cómo un mayor control y aclaración de los mensajes al usuario.

También la exportación de funcionalidad a través de servicios web, donde la arquitectura de este proyecto encajaría de forma clara.

OdA J2EE es una herramienta que permite construir objetos de aprendizaje que hace necesaria la selección de las tablas de join en las relaciones M a N a la hora de definir la estructura de HLOs. Para evitar esto, sería conveniente implementar un sistema de

definición de columnas de HLOs que permitiese seleccionar las relaciones entre las tablas ya creadas en el caso de las relaciones M a N.

Por último, debido al auge de las tecnologías móviles, la integración de OdA J2EE en aplicaciones móviles es otro campo interesante a explorar.

Bibliografía

[1] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2002.

[2] [En línea]. Available: http://www.ariadne-eu.org/. [Último acceso: 2013 09 13].

[3] [En línea]. Available: http://www.merlot.org/merlot/index.htm. [Último acceso: 2013 09 13].

[4] [En línea]. Available: http://door.sourceforge.net/index.html. [Último acceso: 2013 09 13].

[5] [En línea]. Available: http://www.fedora-commons.org/. [Último acceso: 2013 09 13].

[6] «The Java EE 5 Tutorial,» [En línea]. Available: http://docs.oracle.com/javaee/5/tutorial/doc/docinfo.html. [Último acceso: 2013 09 13].

[7] [En línea]. Available: http://www.oracle.com/technetwork/java/javase/jaas/index.html. [Último acceso: 2013 09 13].

[8] «830-1998 - IEEE Recommended Practice for Software Requirements Specifications,» [En línea]. Available: http://standards.ieee.org/findstds/standard/830-1998.html. [Último acceso: 2013 09 13].

[9] E. Gamma, R. Helm, R. Johnson y J. Vlissides, Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 2005.

[10] M. Keith y M. Schincariol, Pro JPA 2: Mastering the Java™ Persistence API, Apress, 2009.

[11] D. Alur, D. Malks y J. Crupi, Core J2EE Patterns: Best Practices and Design Strategies (2nd Edition), Prentice Hall, 2003.

[12] M. Juric, R. Nagappan, R. Leander y S. Jeelani Basha, Professional J2EE EAI, Wrox Press, 2001.