Componente Genérico de Auditoría para Monitorear Cambios en el Modelo de Objetos

Tesina de grado de la carrera Licenciatura en Informática de la Universidad Nacional de La Plata.

Autor: Javier Corvi.

Directores: Javier Díaz – Claudia Queiruga.

Mayo 2012

Componente Genérico de Auditoría para Monitorear Cambios en el Modelo de Objetos

Resumen

En las aplicaciones de software que manejan información crítica, la seguridad y la auditoría de la misma es un requerimiento obligatorio a la hora de realizar el desarrollo de software. Dentro de este contexto se encuentra, entre otros conceptos, la integridad, la confiabilidad y el monitoreo de la información. Es de interés de esta tesina abordar las cuestiones relacionadas al monitoreo sobre los datos del sistema. Este monitoreo o auditoría busca dejar asentadas las modificaciones que se realizaron, el usuario que las realizó y las fechas en que ocurrieron.

Una manipulación errónea en la información puede ocasionar resultados graves para el negocio. Debe ser posible detectar al responsable y el momento exacto en que ocurrió la inconsistencia que dejó a la información en un estado incorrecto para analizar las causas y tomar acciones correctivas.

El objetivo de la tesina es diseñar y desarrollar una librería de aspectos JAVA genérica que audite altas, bajas y actualizaciones de las entidades del modelo de dominio.

El marco teórico, está basado en el estándar ISO/IEC 27001, el cual, a través de sus normas, especifica los requisitos y buenas prácticas necesarios para establecer la seguridad de la información en una organización.

1. Introducción

Las empresas organizadas, automatizan, en aplicaciones de software, la información crítica que manejan; la seguridad de la misma es un requerimiento obligatorio a la hora de realizar el desarrollo de software. Dentro de este contexto se encuentra, entre otros conceptos, la integridad y la confiabilidad de la información, las cuales definen que la misma pueda ser consultada y modificada solo por los usuarios autorizados. El marco teórico de mi tesina, centrado en la seguridad de la información, se abarca a través del estándar ISO/IEC 27001[1] (2005), el cual, a través de sus normas, especifica los requisitos y buenas prácticas necesarios para establecer la seguridad de la información en una organización.

En este marco de seguridad existe otro requerimiento importante que involucra a la información dentro de un sistema de software: **monitorear los cambios realizados sobre los datos del sistema**. Este monitoreo o auditoría sobre los datos busca dejar asentadas las modificaciones que se realizaron, el usuario que las realizó y las fechas en que ocurrieron.

En los requerimientos de seguridad básicos, como lo son la autenticación y la autorización, existen diversos Frameworks y tecnologías que brindan una solución óptima a la hora de realizar el desarrollo de los mismos, por ejemplo, el más utilizado actualmente para desarrollos en JAVA es Spring Security [2] [3] [4]. En contraste,

para realizar el monitoreo sobre los datos debemos incluir la lógica de la auditoría como parte de nuestra lógica de negocios; o bien utilizar algún otro mecanismo como *triggers* o un esquema de Data warehouse.

Agregar la lógica de auditoría dentro de la lógica de negocios es básicamente lo que se intenta evitar, por una cuestión de modularización, reusabilidad y simplicidad en la lógica de negocios.

La solución con triggers está orientada a auditar tablas a nivel de base de datos. Implica definir un trigger para cada tabla que se quiera auditar; cada vez que ocurre una modificación, el trigger inserta en una tabla histórica el cambio realizado. Los inconvenientes son varios: si agregamos una nueva tabla hay que agregar un nuevo trigger junto con la tabla espejo histórica, no hay acceso al usuario que realizó la modificación a nivel de sistema, y además, lo más importante, cada motor de base de datos tiene una definición de triggers particular, si nuestra aplicación es empaquetaba y debe ser implementada en diferentes clientes, debemos tener una implementación de triggers para cada motor de base de datos. En resumen, es complejo, difícil de mantener y no es escalable.

Por otro lado, implementar un Data warehouse es una decisión corporativa importante, ya que tiene un costo elevado. Involucra diseñar e implementar un repositorio histórico complejo, donde cada cierto tiempo (en horarios nocturnos mayormente) el Data warehouse es alimentado a través de un proceso ETL (Extract, Transform and Load) con información del sistema productivo. Es utilizado como una herramienta de análisis de alto nivel. Pero su objetivo principal no está basado en la auditoria de la información, ya que si ocurre más de un cambio en la información por día, estos no se ven reflejados en el Data warehouse.

En vista de las opciones disponibles, el objetivo de mi tesina está centrado en desarrollar un componente genérico que implemente el requerimiento de monitorear la información del sistema de forma simple, modular y escalable.

Las empresas buscan continuamente mejorar la calidad del negocio, para ello, la mayoría opta por aplicar las normas ISO que especifican los requisitos para un buen sistema de gestión de la calidad. Las mismas pueden utilizarse para su aplicación interna, para certificación o con fines contractuales. Durante el desarrollo de mi tesina se apreciará como la librería se aplica a varias de las normas especificadas en las normas ISO 27001, que se enfocan en la seguridad de la información; y por lo tanto ayudan a mejorar la calidad del negocio.

2. Objetivo

Diseñar y desarrollar una librería de aspectos JAVA genérica que audite las altas, bajas y actualizaciones de las entidades del modelo de objetos en una aplicación que utilice la API de persistencia de Java (JPA) [5] junto con el Framework Hibernate [6] [7] encargado de realizar el mapeo objeto-relacional. A este desarrollo se le dará el nombre de **AuditTrail.**

Proveer una herramienta de configuración para que el usuario (programador) que utilice la librería pueda indicar qué entidades y qué campos de la misma se quieren auditar, permitiendo no solo auditar atributos simples de un objeto, sino también auditar colecciones y atributos que referencien a otros objetos.

Alternativamente, el usuario también podrá configurar la herramienta para persistir los datos de la auditoría en otra base de datos distinta a la de la aplicación. Esta configuración brinda otro beneficio extra en cuanto a la seguridad que se quiera adoptar.

Desde el punto de vista de desarrollo de software, separar la lógica de negocios de la lógica de auditoría sobre las entidades, es uno de los pilares más importantes de la modularidad; provee innumerables ventajas que van desde la facilidad de integrar grupos de programadores especializados en auditoría hasta la reusabilidad de código en otros proyectos, adaptación a cambios y modificaciones en los programas. Los lenguajes basados en AOP (Aspect Oriented Programming) [8] permiten programar la lógica de *requerimientos no funcionales* en unidades de software denominadas ***aspectos*** que luego se integrarán a la lógica de negocios**.** En el caso de mi tesina, la lógica de **AuditTrail** será programada usando un lenguaje que extiende a JAVA con ***aspectos*** y que es popularmente utilizado en la comunidad de desarrolladores de software libre, **AspectJ**.

3. Análisis funcional

La norma ISO/IEC 27001 es un estándar internacional que fue preparado con el fin de proveer un modelo para establecer, implementar, operar, monitorear, revisar, mantener y mejorar un Sistema de Gestión de Seguridad de la Información (SGSI).

Además, contiene en el Anexo A una **lista exhaustiva de objetivos de control y controles** que se encuentran comúnmente en las organizaciones. Varios de esos controles están relacionados con los objetivos de la librería AuditTrail. Por cada requerimiento de control que cumpla el AuditTrail se indica que punto de la norma se encuentra relacionado, ver apéndice A.

3.1. Requerimientos de control

Constancia de las modificaciones que ocurrieron sobre la información:

El negocio puede definir un requerimiento que desee conocer todas las modificaciones realizadas sobre la información, dejando constancia de quien fue el responsable de la modificación, la fecha y hora, los valores viejos y nuevos de la información que se modificó.

Puntos de la norma ISO 27001 relacionados: A.10.10.1, A.10.10.2, A.10.10.4, A.13.2.2, A.13.2.3, A.14.1.2, A.14.1.3, A.15.1.3, A.15.3.1.

Recuperar la información en algún momento de su ciclo de vida:

El negocio puede solicitar un requerimiento de recuperación de cierta información crítica que sufrió un cambio incorrecto en algún momento de su ciclo de vida. Esto puede ocurrir por alguna acción malintencionada o por error del responsable de las modificaciones.

Puntos de la norma ISO 27001 relacionados: A.10.5.1, A.14.1.2, A.14.1.3 y A.15.1.3.

Obtener indicadores descriptivos de la información:

El negocio puede solicitar indicadores sobre la información; ya sea para conocer la cantidad de modificaciones que sufre la información o para generar un indicador sobre la información que se está manipulando (como fueron variando sus valores). El AuditTrail genera información histórica que es posible analizar por el negocio para tomar decisiones, a esta técnica se la denomina Datawarehousing.

Puntos de la norma ISO 27001 relacionados: A.13.2.2

Separar la información generada por el AuditTrail de la información del sistema:

El negocio puede solicitar separar en donde reside la información del monitoreo, que contiene los cambios que se realizaron, de la información propia del sistema. Para ello la librería provee una configuración especial en donde se indica cual será la base de datos en donde se registrarán los cambios encontrados.

Puntos de la norma ISO 27001 relacionados: A.15.3.1 y A.10.10.3.

4. Diseño y Arquitectura

El requerimiento de auditar el modelo de objetos es lo que se denomina un *requerimiento no funcional*, como lo son, por ejemplo, el manejo de seguridad, de transacciones, de errores, de logeo, control de concurrencia, etc. Ósea requerimientos que traspasan la lógica de negocios en particular y que todos los sistemas deben poseer, también se los denominan "*Crosscutting concerns*".

La lógica de este tipo requerimientos debe modularizarce de forma que no interfiera con la lógica de los requerimientos funcionales del negocio. AOP (Aspect Oriented Programing) permite abstraer y modularizar los requerimientos no funcionales en unidades de software denominadas ***aspectos*** que luego se integrarán a la lógica de negocios**.** La librería AuditTrail será programada con un lenguaje que extiende a JAVA con Aspectos denominado AspectJ. El desarrollo se realizará dentro del ambiente eclipse, el cual provee las herramientas necesarias para compilar y generar librerías con soporte a aspectos. Además, se utilizaran ***Anotaciones*** para indicar que métodos de la lógica de negocios deberán ser auditados.

Las entidades a ser auditadas serán descriptas en el archivo descriptor ***audit-trail.xml***, en donde se indicarán que entidades participaran de la auditoría junto con los atributos a auditarse de las mismas.

A continuación comenzamos describiendo la arquitectura de la librería junto con los componentes más importantes que la componen.

Los elementos que la componen son:

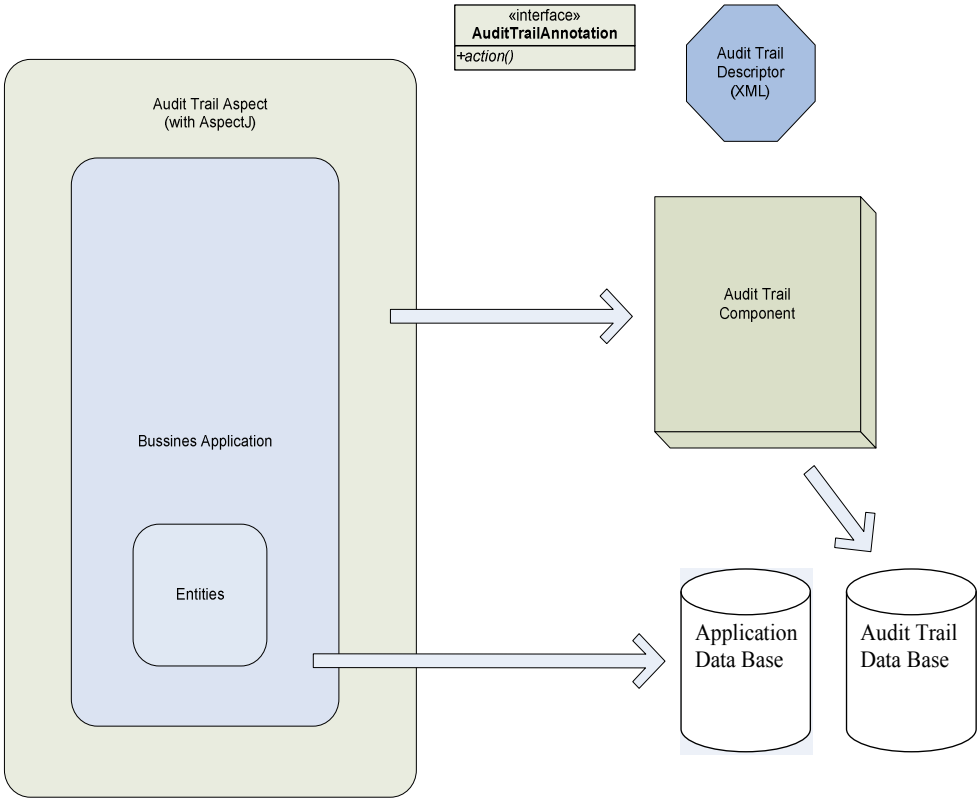
AuditTrailDescriptor: describe que entidades deben ser auditadas.

AuditTrailAnnotation: Anotación que indica que métodos serán auditados.

AuditTrailAspect: Aspecto que separa la funcionalidad de auditoría de la lógica de negocio.

AuditTrailComponent: contiene la lógica en sí de la librería.

En esta sección se analizará cada uno de ellos, comenzando por una descripción a alto nivel de la arquitectura de la librería:



4.1. AuditTrailDescriptor

El archivo **audit-trail.xml** indica cuales son las entidades del modelo que se auditarán, junto con los atributos de las mismas. Debe ser configurado por el usuario programador que utilice la librería.

Analicemos el siguiente audit-trail.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
    <entities>
        <!-- declaracion de la entidad a auditar, se agrega el nombre de la clase -->
        <entity name="facultad.multimedia.staffing.model.Project">
                <fields>
                        <field>
                                <!--nombre del atributo -->
                                <name>name</name>
                                <!--i18n del atributo (para visualizacion) -->
                                <i18nname>comm.i18n.role.name</i18nname>
                        </field>
                        <field>
                        <name>users</name>
                         <!--si el campo es una coleccion -->
                              <isCollection identifier="id" fieldValue="name"/>
                              <!--id es el nombre del identificador de las entidades
                                 de la coleccion -->
                                 <!-- fieldValue es el valor que se tomara de las
                               entidades de la coleccion para verificar el cambio -->
                        </field>
                        <field>
                              <name>state</name>
                             <i18nname>comm.i18n.project.state</i18nname>
                             <!--si el campo es una entidad anidada -->
                             <isEntity identifier="id" fieldValue="name"/>
                             <!--id es el nombre del identificador de esa entidad -->
                             <!-- fieldValue es el valor que se tomara de la entidad
                             anidada para verificar el cambio -->
                        </field>
                </fields>
        </entity>
</entities>
```

Cuando el componente se inicia, parsea el **audit-trail.xml** generando así un grafo de entidades que están siendo auditadas, este grafo será utilizado para obtener información cuando se realice un cambio en alguna entidad. En la declaración de una entidad se agregan los nombres de los fields (atributos) a auditarse, esta información será utilizada para luego, a través de reflexión[1], invocar los ***getters*** del objeto que está siendo auditado. Exactamente lo mismo ocurre cuando el atributo es un objeto anidado o una colección; hay que indicar cuál es el nombre del campo con el que el

[1] Es la habilidad que tiene un programa para realizar algún tipo de computación sobre sí mismo. Permite obtener información, en tiempo de ejecución, acerca de la clase de un objeto: modificadores, constructores, métodos, variables y superclases.

objeto anidado o cada uno de los objetos de la colección serán auditados. Observemos el atributo state de la entidad Project. Se indica que es un atributo que referencia a otro objeto-entidad y se completa la información agregando el nombre del identificador (id) de estado y el nombre del atributo descriptivo del estado (name), este último es el atributo que se utilizará para verificar si han ocurrido cambios. En forma análoga ocurre cuando el atributo es una colección, vemos el caso del atributo users.

El descriptor es validado a través del ***audit-trail.xsd*** que se encuentra dentro de la librería AuditTrail. De esta forma cuando se inicia el componente, si el audit-trail.xml no respeta el audit-trail.xsd falla la inicialización de la librería y el programador es avisado en forma correspondiente.

Es importante destacar el diseño de este xml para configurar las entidades y atributos a auditar, ya que al estar separado del código fuente, la tarea de realizar esta configuración puede llevarse a cabo por una persona encargada en el requerimiento de auditoría, la cual, con el debido entrenamiento, al ver la facilidad de configuración podría llegar a ser un programador junior o una persona con conocimientos en informática.

4.2. AuditTrailAnnotation

La librería AuditTrail incluye una Anotación, desarrollada para mi tesina, denominada ***AuditTrailAnnotation***. La misma debe ser agregada, por parte de los programadores, en los métodos que quieran ser auditados. Para mayor información sobre la tecnología de Anotaciones referirse a la bibliografía o al trabajo completo de la tesina.

```
/**
 * Anotación que se utiliza cuando se quiere auditar un método del sistema
*/
public @interface AuditTrailAnnotation {
        /**
         * Enumeration para tipo de acción
         */
        public enum ActionType { CREATE, UPDATE, DELETE};
        /**
         * Tipo de accion que se ejecuta
         */
        ActionType actionType();
        /**
         * Nombre del atributo en el método que debe ser auditado
         */
        String auditedObject();
        /**
         * Accion descriptiva que se ejecuta
         */
        String action();
}
```

Ejemplo de declaración de la anotación en un método:

```
/**
* Creación de un objeto
*/
@AuditTrailAnnotation(auditedObject="object",
actionType=ActionType.CREATE,action="com.action.create")
public void create(Clase object) {
        entityManager.persist(object);
}
```

En este caso al agregar la anotación estamos indicando que se auditara el parámetro "object" del método "create". Además se indica el tipo de operación que se está realizando (CREATE) y un i18n descriptivo para una futura visualización desde la interfaz de la aplicación de los cambios ocurridos.

4.3. AuditTrailAspect

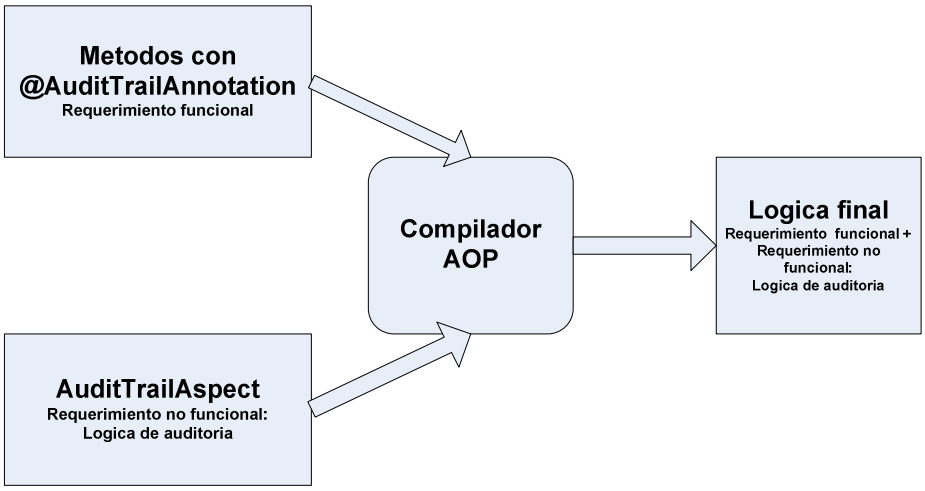
En esta sección se describirá el Aspecto denominado "**AuditTrailAspect**" que he realizado para mi tesina y que tiene como fin de modularizar la lógica de auditoría. La idea es que el aspecto intercepte, a través de la declaración de su pointcut, todos los métodos que contengan la anotación AuditTrailAnnotation; integrando luego la lógica de la auditoría a la lógica de negocios de los métodos.

```
/**
 * Aspecto que se encarga de modelar el requerimiento no funcional de auditoría de
las entidades
 */
public aspect AuditTrailAspect{
        /**
         * Pointcut definido para que intercepte los métodos que contengan
         * el Annotation @AuditTrailAnnotation
         */
        pointcut auditTrail(Object object)
                : execution(@AuditTrailAnnotation * *(..)) && args(object);
        /**
         * Encargado de aplicar la funcionalidad de auditoría.
         * Invoca al Facade AuditTrail, punto de entrada al componente.
         */
        Object around(Object object) : auditTrail(object) {
                //me quedo con la entidad a auditar
                EntityChange entityChange = null;
                try {
                        //invoco al componente para que realice la lógica de la
                        //auditoria, retorna los cambios a la entidad.
                        entityChange = auditTrail.audit(entity,
                                annotation.actionType() ,annotation.action());
```

```
                } catch (.....) {….}
                 //procedo con el flujo normal del método
                Object objectSaved = proceed(object);
                //si la ejecución del método no tuvo inconvenientes, salvo los
                //cambios encontrados a la entidad
                try {
                        auditTrail.save(entityChange);
                } catch (.....) {.....}
                return objectSaved;
        }
}
```

En una primera instancia, antes de que se ejecute la lógica de negocios del método interceptado, el advice **around** se encarga de invocar al componente AuditTrail buscando los cambios realizados al objeto a auditar. Luego, se continúa con la lógica del método en forma normal y al terminar, como parte final del aspecto, si se han encontrado cambios se invoca de nuevo al componente para que persista las modificaciones encontradas y de esta manera dejar asentados los cambios que se le realizaron al objeto.

Luego de analizado el Aspecto y la anotación vemos como queda la integración de lógica de negocio y de la lógica de auditoría:



4.4. Análisis del funcionamiento del Aspecto y la Anotación

Analizando el funcionamiento del aspecto AuditTrailAspect junto con la anotación AuditTrailAnnotation, notamos que la auditoría se realiza sobre un objeto que es pasado como parámetro en un método.

```
@AuditTrailAnnotation(auditedObject="user",
actionType=ActionType.CREATE,action="com.action.create")
```

```
public void createUser(User user){
        entityManager.getTransaction().begin(); //open transaction
        entityManager.persist(user);
        entityManager.getTransaction().commit(); //commit transaction

}
```

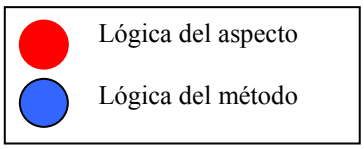
El objeto que debe ser auditado se describe en el elemento **auditedObject**, el cual hace referencia al **nombre del parámetro a auditar**.

Cuando realizamos la integración del aspecto con la lógica de negocios el resultado es el siguiente:

```
public void createUser(User user){
        Object entity=getAuditedObject();//entity = user
        entityChange=auditTrail.audit(entity,annotation.actionType()
        ,annotation.action());

        entityManager.getTransaction().begin(); //open transaction
        entityManager.persist(user);
        entityManager.getTransaction().commit(); //commit transaction

        //sino ocurre ningún error, se persiste el AuditTrail
        auditTrail.save(entityChange);
}
```



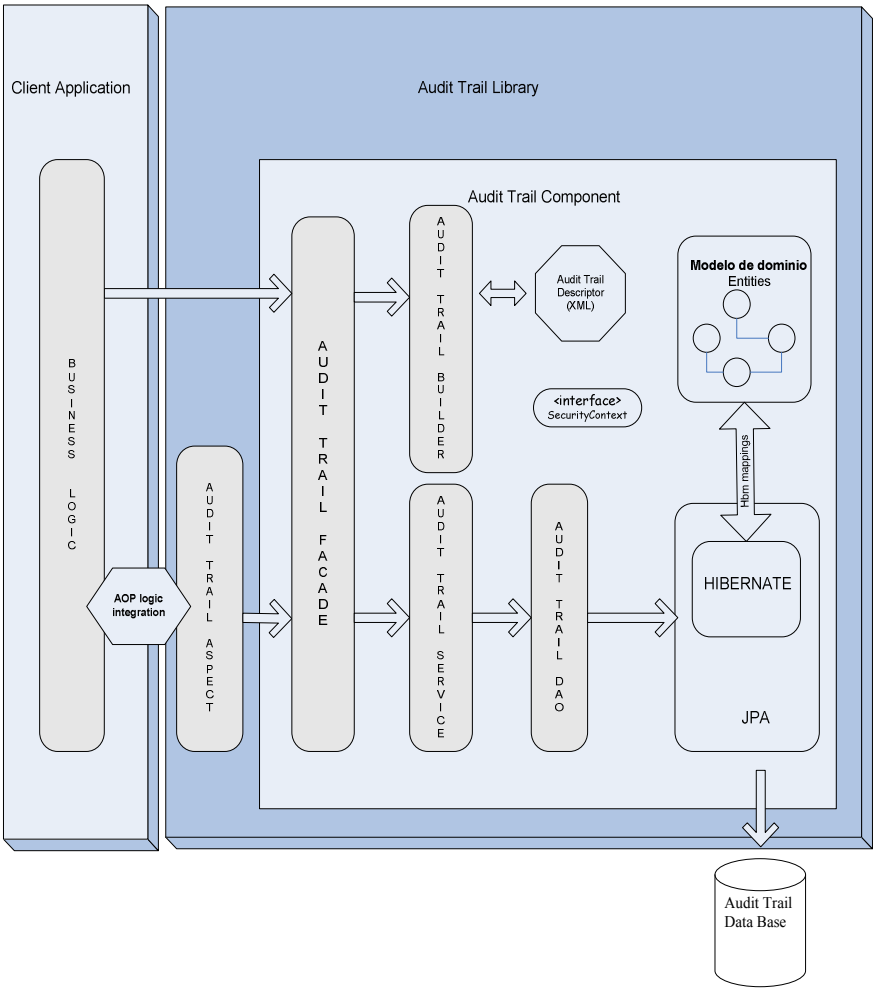
Al observar con detenimiento el funcionamiento de la auditoría, notamos que el sistema cliente debe estar diseñado con una modularización básica, en donde la lógica de acceso a los datos para persistir los objetos debe encontrarse en métodos separados.

Si la aplicación se encuentra diseñada pobremente y la persistencia de los objetos no se encuentra modularizada, la librería AuditTrail no podrá ser implementada correctamente.

4.5. AuditTrailComponent

El componente AuditTrail es el corazón de la librería; incluye la lógica para encontrar los cambios que se han realizado sobre un objeto; así como también la persistencia de la información auditada. Además se encarga de inicializar el ambiente leyendo el descriptor audit-trail.xml.

La arquitectura del componente se encuentra definida en n-capas (layers) en donde cada una tiene un rol determinado:



Si el lector desea indagar más sobre este componente, referirse al apéndice B en donde se describen los diagramas de secuencia de la librería.

5. Trabajos futuros

La librería AuditTrail puede extenderse ampliamente ya que existe un gran campo a explotar en cuanto a la auditoría de la información en los sistemas.

Existen diversos requisitos que no se encuentran implementados, que serian de gran utilidad:

1) Registro de las veces que un usuario ingreso al sistema

A través de este requerimiento puede obtenerse la información necesaria para conocer las veces que el usuario ingreso al sistema: días, horarios, duración de la sesión, etc.

2) Registrar los accesos a entidades

Se podría registrar cada vez que una entidad es consultada. A través de este requerimiento podría conocerse cuales son las entidades más consultadas en el sistema, y realizar indicadores sobre entidades favoritas.

Además podemos optimizar este requerimiento para que registre el usuario que está consultando la entidad, pudiendo así conocer, por usuario, cuáles son sus entidades favoritas, para luego, realizar el análisis correspondiente.

3) Auditoría en atributos File

Monitorear las modificaciones en atributos de tipo File no se encuentra implementado y sería de gran utilidad para la auditoría de la información. En este punto hay que realizar un análisis para vislumbrar como podría llevarse a cabo este requerimiento.

Además, analizar nuevas tecnologías e ideas para mejorar tanto el diseño como el desarrollo de la librería forma parte de los trabajo a futuro.

6. Conclusión

El objetivo de la tesina, desarrollar una librería genérica para auditar el modelo de objetos, pudo implementarse con éxito. Este resultado se manifiesta al analizar el diseño y el desarrollo de la librería; y como la misma fue utilizada para auditar el modelo del proyecto StaffingProject (apéndice C), además de otros sistemas que se encuentran productivos. Los registros que genera la librería en la auditoría del proyecto StaffingProject están alineados con los objetivos planteados en la tesina. Así mismo, durante el análisis funcional de la librería, se apreció que varios requerimientos de auditoría de la librería están se relacionan con objetivos de control propuestos en la norma ISO 27001.

El otro objetivo importante de la librería estaba centrado en el diseño del requerimiento de auditoría; separar la lógica de negocios de la lógica de auditoría, permitiendo así, diversas ventajas en cuanto a la modularización, reusabilidad, adaptación e integración en el desarrollo. A través de la calificación del requerimiento de auditoría como “no funcional”, y diseñando el mismo utilizando los conceptos de AOP, se logró cumplir el objetivo. La lógica de auditoría se encuentra modularizada en un aspecto, el cual se integra, en forma transparente para el usuario programador, con la lógica de negocio.

La facilidad de uso de la librería por parte de los programadores, era también un punto muy importante para analizar el éxito de la misma. Como pudimos apreciar, una vez instalada y configurada, es muy fácil, en mi opinión, utilizar la librería. Basta con configurar el archivo descriptor audit-trail.xml, indicando que entidades y que campos de la misma van a ser auditados.

Durante el desarrollo de mi tesina se utilizaron Frameworks, tecnologías y técnicas de programación que son actuales, estándares y de código abierto; repasando los más importantes podemos nombrar AOP (AspectJ), JPA, Hibernate, Anotaciones, Reflexion, Spring, Struts 2, Ajax, entre otras. Cada una de estas tecnologías aportó

sus mejores características y beneficios para mejorar algún aspecto de diseño y desarrollo en mi tesina.

Referencias bibliográficas

[1] Normas ISO. http://www.iso.org/

[2] Walls, C. (2008). Spring In Action. Second Edition. Greenwick: Manning.

[3] Spring. http://www.springsource.org/

[4] Spring Security. http://static.springsource.org/spring-security/site/

[5] JPA. http://java.sun.com/developer/technicalArticles/J2EE/jpa/

[6] Bauer, C; King, G. (2007). Java Persistence with Hibernate. New York: Manning.

[7] Hibernate. https://www.hibernate.org/

[8] Laddad, R. (2003). AspectJ In Action. Second Edition. Greenwich: Manning.

Apéndice A: Puntos de la norma ISO 27001 relacionados con la librería AuditTrail

A.10.4 Protección contra código malicioso y código móvil Objetivo: Proteger la integridad del software y la información.	
A.10.4.1 Controles contra software malicioso	Control Detección, prevención y controles de recuperación para protegerse del código malicioso.
A.10.4.2 Controles contra códigos móbiles	Control Cuando el uso de código móvil es autorizado, la configuración debe garantizar que el mismo funciona de acuerdo a las políticas de seguridad definidas. Debe impedirse la ejecución de códigos móviles no autorizados.
A.10.5 Back-up Objetivo: Para mantener la integridad y disponibilidad de la información.	
A.10.5.1 Back-up de la información	Control Back-ups de la información y el software deberán ser realizadas y testeadas regularmente de acuerdo a la política de back-up.

A.10.10 Monitoreo Objetivo: Detectar actividades de procesamiento de la información no autorizadas.	
A.10.10.1 Monitoreo de actividades	Control Registrar las actividades del usuario, excepciones, y distintos eventos definidos por la organización. Los registros deberán conservarse durante un período determinado para ayudar en las investigaciones futuras y de supervisión del control de acceso.
A.10.10.2 Monitorear el uso del sistema	Control Procedimientos para monitorear el uso de procesamiento de la información deben ser establecidos y los resultados revisados regularmente.
A.10.10.3 Protección de logs del monitoreo	Control Los logs del monitoreo de la información deben estar protegidos contra manipulación y acceso no autorizado.
A.10.10.4 Logeo de las actividades de los administradores y operadores	Control Las actividades de los administradores y operadores del sistema deben quedar registradas
A.10.10.5 Registro de los errores	Control Los errores deben registrarse, analizarse y tomar medidas apropiadas.

A.13.2 Gestión de incidentes y mejoras de seguridad de la información Objetivo: Garantizar un enfoque coherente y eficaz que se aplique a la gestión de incidentes de seguridad de la información.	
A.13.2.1 Responsabilidades y Procedimientos	Control Las responsabilidades de gestión y procedimientos se establecerán para garantizar una respuesta rápida, eficaz y ordenada a los incidentes de seguridad de la información.
A.13.2.2 Aprender de los incidentes de seguridad de la información	Control Habrán mecanismos para conocer los tipos, volúmenes y costos de los incidentes de seguridad de la información; de esta forma los incidentes pueden ser cuantificados y monitoreados.
A.13.2.3 Recolección de evidencia	Control Cuando una acción de seguimiento contra una persona u organización, después de un incidente de seguridad de la información, implica una acción legal (ya sea civil o penal), las evidencias deben ser recolectadas, retenidas, y presentadas a las autoridades correspondientes.

A.14.1.2 Continuidad de las operaciones de negocio y evaluación de riesgos	Control Los eventos que pueden causar interrupciones en los procesos de negocio deben ser identificados, junto con la probabilidad y el impacto de dichas interrupciones y sus consecuencias para la seguridad de la información.
A.14.1.3 Desarrollar e implementar planes de continuidad incluyendo la seguridad de la información	Control Los planes deberán elaborarse y aplicarse para mantener o restablecer las operaciones y garantizar la disponibilidad de información en el nivel requerido y en las escalas de tiempo necesario tras una interrupción o fracaso de los procesos críticos del negocio.

A.15.1.3 Protección de los registros de la organización	Control Los registros importantes deben estar protegidos contra pérdida, destrucción y falsificación, de acuerdo con los requisitos correspondientes.

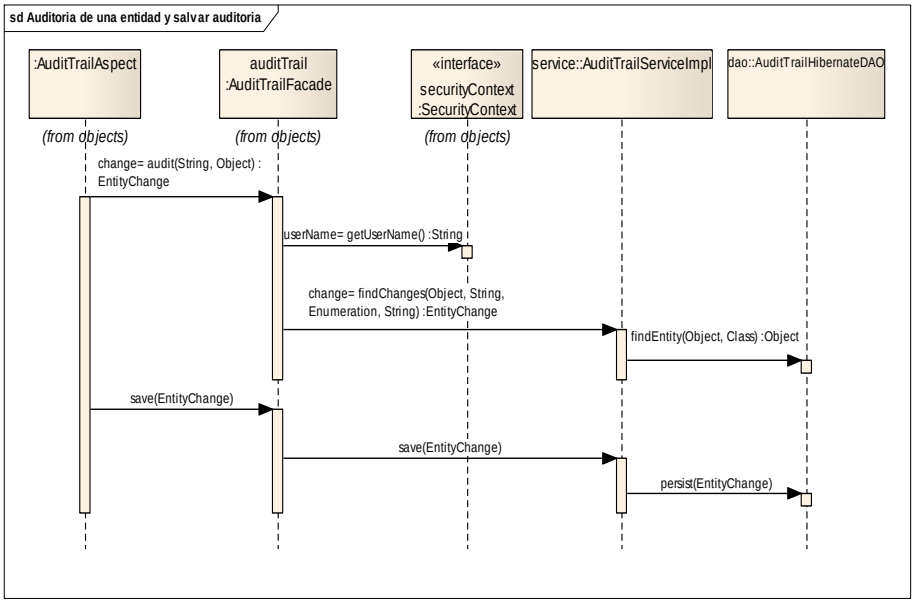
A.15.3 Consideración de los sistemas de auditoría de la información Objetivo: Maximizar la eficacia y minimizar la interferencia desde y hasta el proceso de auditoría de los sistemas de información.

A.15.3.1 Controles sobre la auditoría de los sistemas de información	Control Actividades y requerimientos de auditoría, que impliquen chequeos en los sistemas, deben ser cuidadosamente planificados para minimizar el riesgo de las interrupciones de los procesos de negocio.
A.15.3.2 Protección de las herramientas de autoría de los sistemas de información	Control El acceso a herramientas de auditoría de los sistemas de información estará protegido para evitar cualquier posible abuso o amenaza.

Apéndice B: Diagramas de interacción del la librería AuditTrail

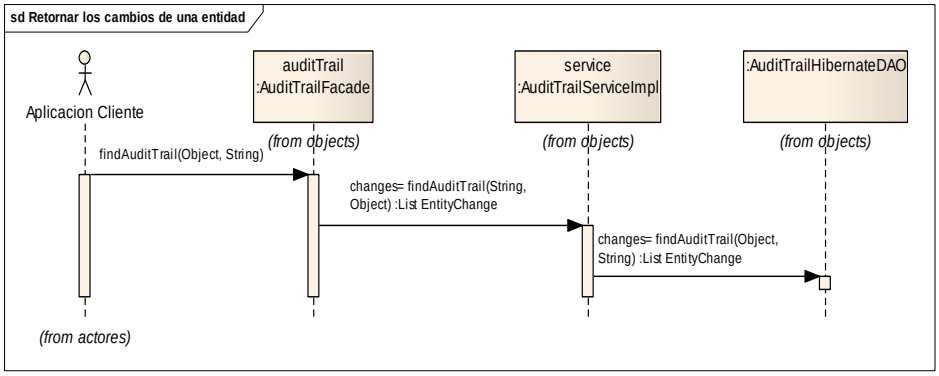
Auditoría de una entidad

El aspecto AuditTrailAspect agrega, a la lógica de negocios del sistema auditado, la invocación al componente para realizar la auditoría del objeto.



Retornar los cambios de una entidad

En este caso, el sistema cliente, sin participación alguna del aspecto, es el encargado de invocar al componente para que retorne los cambios que sufrió una entidad.

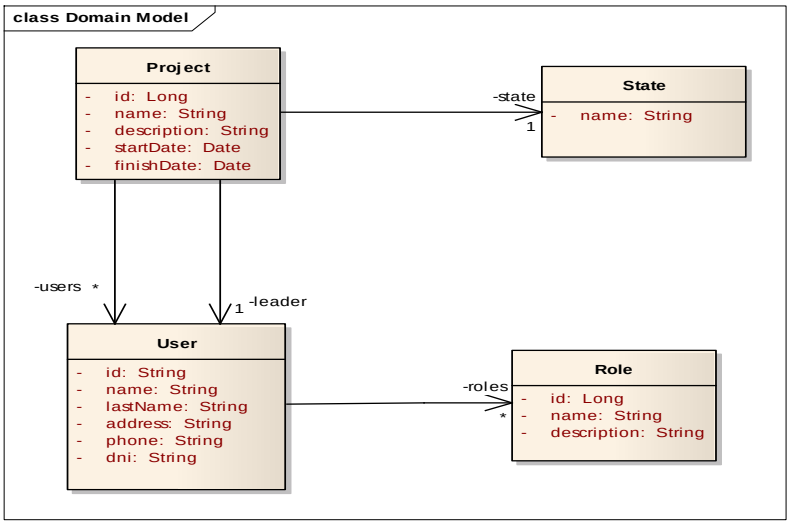


Apéndice C: Ejemplo. Uso de la librería en el proyecto StaffingProject

StaffingProject es un proyecto WEB Java desarrollado durante la cursada de la materia Taller de Multimedia, dentro de mi tesina será el proyecto WEB con el que se analizará y probará el funcionamiento de la librería AuditTrail.

El objetivo de este proyecto es modelar altas, bajas y modificaciones de roles, usuarios y proyectos. La funcionalidad del sistema se basa en automatizar los proyectos que se encuentran en una compañía, asignándole a cada uno fechas límites para su finalización, el líder a cargo, participantes del mismo, etc. Los proyectos durante su ciclo de vida pueden detenerse momentáneamente y luego enviarse a ejecución nuevamente. Por su parte, los usuarios tienen roles definidos y pueden participar en diferentes proyectos. Además, se brinda una búsqueda avanzada de cada una de las entidades.

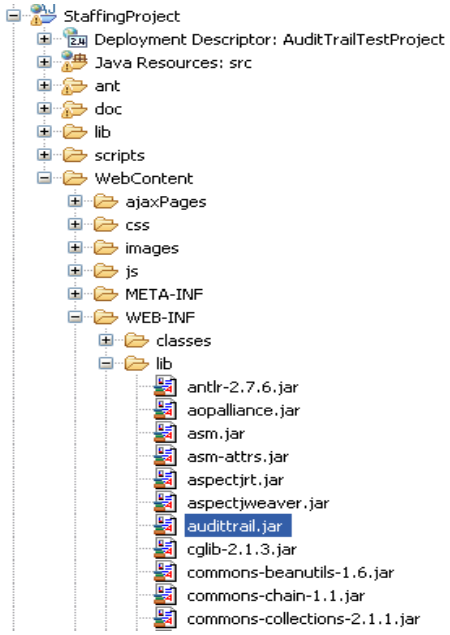
Modelo de dominio



Instalación y Configuración de la librería AuditTrail en el proyecto StaffingProject

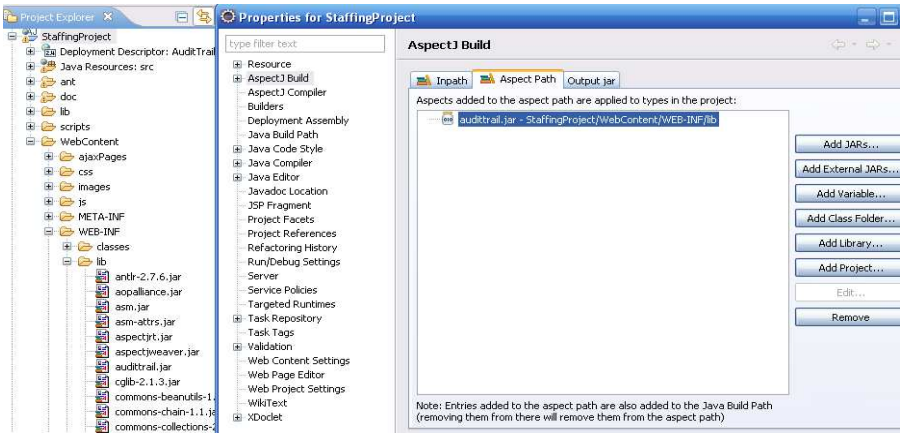
Paso 1

Agregar la librería audittrail.jar dentro de la carpeta WEB-INF/lib del proyecto a ser auditado, en nuestro ejemplo utilizamos el proyecto StaffingProject.



Paso 2

Incluir la librería dentro del **path de aspectos**. Ambiente Eclipse con AJDT.

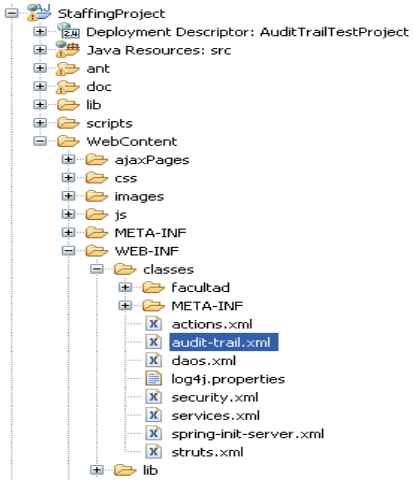


Creación del archivo descriptor audit-trail.xml

El archivo audit-trail.xml para auditar el modelo de dominios del sistema StaffingProject es:

```
<?xml version="1.0" encoding="UTF-8"?>
<entities>
	<entity name="facultad.multimedia.staffing.model.Role">
		<identifier>id</identifier> <!-- indica el nombre del identificador -->
		<fields> <!-- coleccion de campos a auditar -->
			<field>
				<name>name</name>
				<i18nname>comm.i18n.role.name</i18nname>
			</field>
			……..
		</fields>
	</entity>
	<entity name="facultad.multimedia.staffing.model.User">
		<identifier>id</identifier>
		<fields>
			<field>
				<name>name</name>
				<i18nname>comm.i18n.user.name</i18nname>
			</field>
			……
		</fields>
	</entity>
	<entity name="facultad.multimedia.staffing.model.Project">
		<identifier>id</identifier>
		<fields>
			<field>
				<name>name</name>
				<i18nname>comm.i18n.project.name</i18nname>
			</field>
			……
			<field>
				<name>leader</name>
				<i18nname>comm.i18n.project.leader</i18nname>
				<isEntity identifier="id" fieldValue="lastName"/>
			</field>
			<field>
				<name>state</name>
				<i18nname>comm.i18n.project.state</i18nname>
				<isEntity identifier="id" fieldValue="name"/>
			</field>
			<field>
				<name>users</name>
				<i18nname>comm.i18n.project.users</i18nname>
				<isCollection identifier="id" fieldValue="lastName"/>
			</field>
		</fields>
	</entity>
</entities>
```

El archivo debe encontrarse en el root del classpath de la aplicación, ósea **WEB-INF/classes**:



Configurar la anotación AuditTrailAnnotation en los métodos a auditar

En nuestro ejemplo, la aplicación StaffinProject está diseñada con una capa de DAOS; lugar indicado para agregar la anotación AuditTrailAnnotation.

```
/**
 * Creación de un objeto
* @param object
 */
@Transactional(propagation=Propagation.REQUIRES_NEW,readOnly=false)
@AuditTrailAnnotation(auditedObject="object",actionType=ActionType.CREATE,
action="com.action.create")
public void create(Clase object) {
        entityManager.persist(object);
}
/**
* Actualización
* @param object
*/
@Transactional(propagation=Propagation.REQUIRES_NEW,readOnly=false)
@AuditTrailAnnotation(auditedObject="object",actionType=ActionType.UPDATE,
action="com.action.update")
public void update(Clase object){
        entityManager.merge(object);
}
```

La anotación @Transactional es una de la maneras de indicar, cuando utilizamos el Framework de Spring, que el método va a ser transaccional.

Análisis del AuditTrail en el proyecto StaffingProject

En esta sección comenzaremos a interactuar con el sistema pare analizar la información que va guardando el AuditTrail en sus tablas.

Creación de una entidad

Comencemos analizando la creación del rol “Soporte”.

Registros guardado por el AuditTrail

Tabla ***auditentitychange***

	entityname	entityid	username	actiontype	i18naction	logdate
1	facultad.multimedia.staffing.model.Role	10	jcorvi	CREATE	com.action.create	2010-09-20 21:21:53

El campo que indica la clase auditada, *entityname*, junto con el campo que indica el identificador, *entityid*, describen a la entidad auditada; en este caso la entidad es un Rol con el id=10.

Tabla ***auditsimplefieldchange***

	id	fieldname	i18nname	oldvalue	newvalue	id_entitychange
1	1	name	comm.i18n.role.name		Soporte	1
2	2	description	comm.i18n.role.description		Atender a los clientes ante cualquier consulta	1

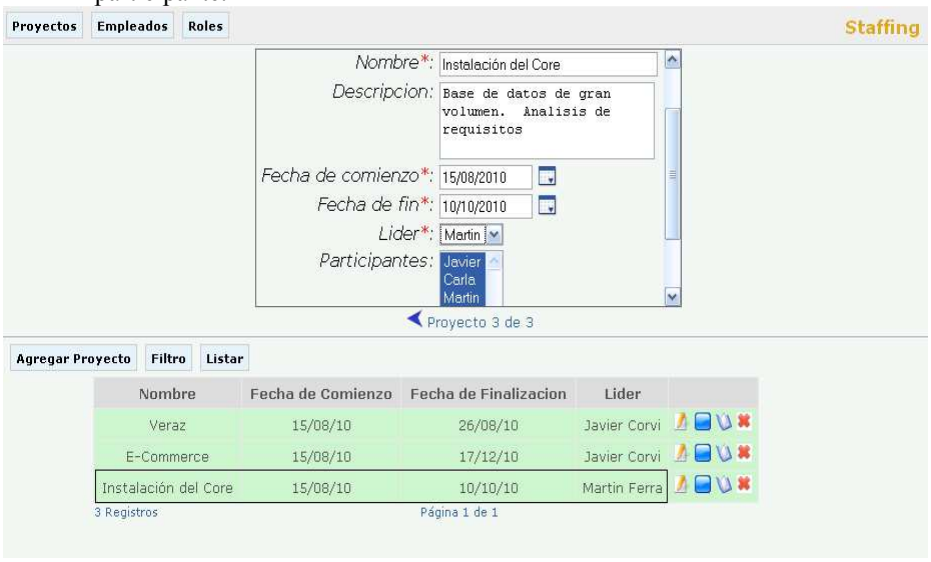
Aquí se describen los campos simples modificados, como se trato de una creación, no se guardaron los valores anteriores.

Edición de una entidad

Pasemos a otro ejemplo, edición del proyecto “Instalación de Core”.

Datos que se van a modificar:

- Nombre: “Instalación de Core” por “Instalación del Core”.
- Fecha de Fin: “10/09/2010” por “10/10/2010”.
- Líder: Carla por Martin.
- Participantes: se agregan al proyecto Javier y Martín, no se elimina ningún participante.



Registros guardado por el AuditTrail

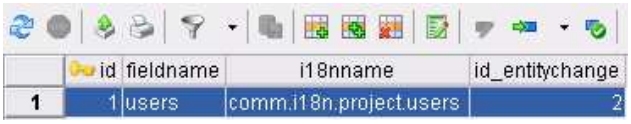
Tabla ***auditentitychange***

	entityname	entityid	username	actiontype	i18naction	logdate
1	facultad.multimedia.staffing.model.Role	10	jcorvi	CREATE	com.action.create	2010-09-20 21:21:53
2	facultad.multimedia.staffing.model.Project	3	jcorvi	UPDATE	com.action.update	2010-09-20 21:34:40

Tabla ***auditsimplefieldchange***

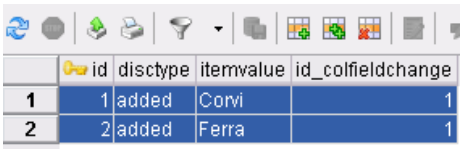
	fieldname	i18nname	oldvalue	newvalue	id_entitychange
1	name	comm.i18n.role.name		Soporte	1
2	description	comm.i18n.role.description		Atender a los clientes ante cualquier consulta	1
3	name	comm.i18n.project.name	Instalacion de Core	Instalación del Core	2
4	finishDate	comm.i18n.project.finishDate	Fri Sep 10 00:00:00 CEST 2010	Sun Oct 10 00:00:00 CEST 2010	2
5	leader	comm.i18n.project.leader	Gomez	Ferra	2

Tabla ***auditcollectionfieldchange***



	id	fieldname	i18nname	id_entitychange
1	1	users	comm.i18n.project.users	2

Tabla ***auditcollectionitem***



	id	disctype	itemvalue	id_colfieldchange
1	1	added	Corvi	1
2	2	added	Ferra	1

Borrado de una entidad

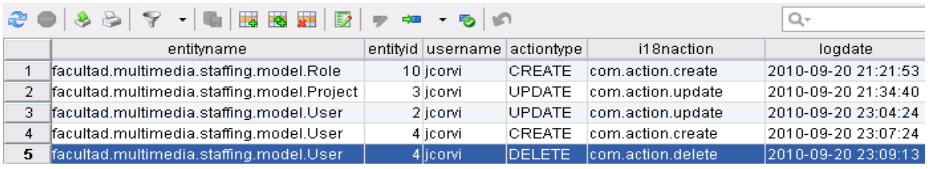
Borrado del usuario Marcos Sierra.

Empleado	DNI	Telefono	
Javier Corvi	322445532	4345533344	
Carla Gomez	1960410619	3555356	
Martin Ferra	2445666	455456654	
Marcos Sierra	24552125	42344553	

4 Registros Página 1 de 1

Registro del AuditTrail

Tabla ***auditentitychange***



	entityname	entityid	username	actiontype	i18naction	logdate
1	facultad.multimedia.staffing.model.Role	10	jcorvi	CREATE	com.action.create	2010-09-20 21:21:53
2	facultad.multimedia.staffing.model.Project	3	jcorvi	UPDATE	com.action.update	2010-09-20 21:34:40
3	facultad.multimedia.staffing.model.User	2	jcorvi	UPDATE	com.action.update	2010-09-20 23:04:24
4	facultad.multimedia.staffing.model.User	4	jcorvi	CREATE	com.action.create	2010-09-20 23:07:24
5	facultad.multimedia.staffing.model.User	4	jcorvi	DELETE	com.action.delete	2010-09-20 23:09:13

Cuando se borra una entidad solamente se guarda registro de eliminación