UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

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

Autores: Javier Corvi
Directores: Javier Díaz - Claudia Queiruga
Carrera: Licenciatura en Informática

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 qué 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 dejo 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.

Palabras Claves

Auditoría, monitoreo, logs, normas de calidad, norma ISO/IEC 27001, Java, programación orientada a aspectos, AspectJ, Anotaciones, JPA, Hibernate, Spring, Struts 2.

Trabajos Realizados

Estudio y análisis de la tecnología orientada a aspectos y anotaciones de código, para modelar y diseñar soluciones que se centren en requerimientos no funcionales, en particular lo relacionado a la auditoría de los objetos del modelo de dominio.
Estudio de la norma ISO/IEC 27001 que se centra en el Sistema de gestión de seguridad de la información.
Diseño y desarrollo de la librería de Aspectos Java AuditTrail.
Diseño y desarrollo del proyecto WEB StaffingProject para testear la librería AuditTrail.

Conclusiones

La tecnología orientada a aspectos es la más adecuada para desarrollar los requerimientos no funcionales de los sistemas. Las aplicaciones Web implementadas con Frameworks Java pueden ser integradas y extendidas fácilmente con esta tecnología.
En cuanto al marco teórico, el desarrollo y testeo de la librería AuditTrail cumple con los objetivos de control dispuestos en la norma ISO 27001.
Profesionalmente, el desarrollo de mi tesina me sirvió para implementar soluciones de software que todavía no son muy conocidas en la industria.

Trabajos Futuros

Extensión de los requerimientos de la librería. En lo inmediato monitorear las modificaciones en atributos binarios, en especial documentos de texto; monitorear el acceso de usuarios al sistema; y monitorear accesos de los usuarios a entidades particulares.
Así mismo, existe un gran campo a explotar en cuanto a la auditoría de la información en los sistemas.

Fecha de la presentación: Noviembre / 2011

Índice

Capítulo I: La librería AuditTrail y su Marco Teórico.

Introducción

En las aplicaciones de software que manejan información crítica, 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 qué 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 utilizar *triggers* definidos en la base de datos. Dado a mí entender no son soluciones óptimas en cuanto a la modularidad, escalabilidad y facilidad de desarrollo.

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 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.

Personalmente, estas ideas de calidad, mejora continua del negocio, normas ISO, procedimientos, auditorías y seguridad de la información tienen su origen en el ambiente laboral en el cual me estoy desempeñando. Durante los últimos 6 años he trabajado en una empresa de software denominada “Sisdam” cuyo principal mercado está orientado a realizar software para gestionar la calidad del negocio. La aplicación “LOYAL”, desarrollada en Java, se encuentra fundamentada en las normas ISO, brindando un ambiente propicio para cargar documentos, procedimientos, auditorías y varios formularios centrados en la calidad.

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**.

Además se utilizara la aplicación WEB StaffingProject para probar la funcionalidad de la librería. La misma fue desarrollada durante la materia Taller de Multimedia y adaptada, con algunos cambios, para mostrar la funcionalidad de la librería AuditTrail.

Durante el desarrollo de este documento se hará hincapié tanto en el diseño como en el desarrollo de la librería AuditTrail y del Proyecto WEB StaffingProject. Si el lector desea indagar más sobre estos aspectos, se proporcionará un documento interactivo denominado ***Documentacion.html*** para ver en forma completa los casos de uso, diagramas de clases, diagramas de interacción, diagrama de Base de datos y el Javadoc de la implementación, tanto de la librería AuditTrail como de la aplicación WEB StaffingProject. El mismo podrá accederse en forma online a través del proyecto Web entregado.

Marco Teórico

En muchas empresas, los activos más valiosos que poseen residen en la información. Por lo tanto, las empresas exitosas reconocen los beneficios de la tecnología de la información (TI) y la utilizan para una mejora continua en el negocio.

La necesidad del aseguramiento de valor de TI, la administración de los riesgos asociados a TI, así como el incremento de requerimientos para ***controlar la información***, constituye la esencia del gobierno de TI.

Las organizaciones buscan continuamente la calidad en el negocio, para ello muchas implementan las normas ISO 9000 para mejorar distintos aspectos dentro de la

organización. Los cuales abarcan desde la estandarización de actividades por medio de documentación, incrementar la satisfacción de los clientes, medición, monitoreo y mejora de los procesos, mejora continua de los productos, eficacia, etc. En sí, el objetivo de la implantación de esta norma es mejorar la calidad general de la organización.

Además, a nivel de mercado la empresa busca certificarse en la norma para mejorar su imagen, con el fin de atraer clientes. En algunos casos, las empresas les exigen a sus proveedores que estén certificados en las normas.

Existen entidades encargadas de realizar el proceso de certificación que se encargan de auditar a la organización para definir si se están llevando a cabo todos los requisitos para poder certificarse en la norma. De no ser así, si el auditor encuentra áreas de incumplimiento de la norma, la organización tiene un plazo para poder adoptar medidas correctivas.

Una de las ramas de la norma, la que está relacionada con mi tesina, es la norma ISO 27001 que se centra en la seguridad de la información.

La Norma ISO/IEC 27001

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). La adopción de un SGSI debe ser una decisión estratégica de la empresa. El diseño y la implementación de un SGSI están influenciados por las necesidades, objetivos, requerimientos de seguridad, los procesos empleados y el tamaño de la organización. Es esperable que un SGSI vaya escalando de acuerdo a las necesidades del negocio.

Una organización necesita identificar y gestionar numerosas actividades con el fin de funcionar en forma efectiva. Cualquier actividad que utilice recursos y gestión con el fin de permitir la transformación de entradas en salidas puede ser considerada un proceso. A menudo el resultado de un proceso es la entrada del siguiente proceso.

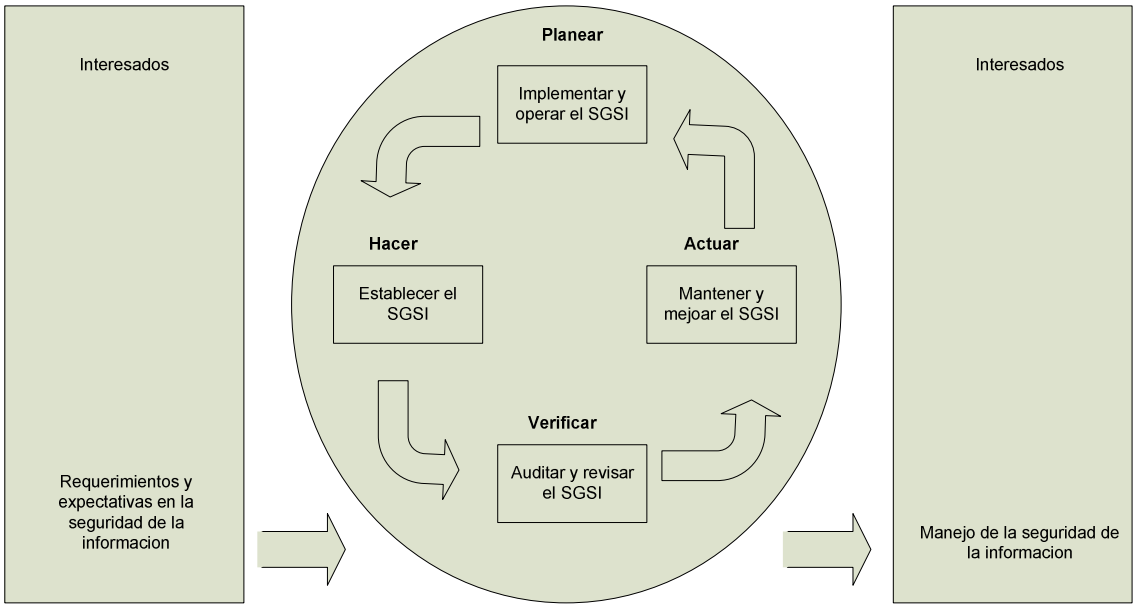
La aplicación de un sistema de procesos dentro de una organización, junto con la identificación e interacciones de estos procesos, y su gestión, puede ser referida como un enfoque basado en procesos.

El enfoque de procesos para la gestión de seguridad de la información anima a sus usuarios hacer hincapié en la importancia de:

- Entender los requerimientos de seguridad de la información en la organización y la necesidad de establecer políticas y objetivos de seguridad de la información;
- Implementar y operar los controles para manejar los riesgos de seguridad de la información que existen en la organización;
- Monitorear y evaluar el rendimiento y la eficacia del SGSI; y
- Mejoras continuas basadas en mediciones objetivas.

El estándar adopta el conocido “Ciclo de Deming”: PDCA - acrónimo de Plan, Do, Check, Act (Planificar, Hacer, Verificar, Actuar); y es aplicado a todos los procesos de SGSI.

La siguiente figura muestra cómo un SGSI toma como entrada los requerimientos de seguridad y las expectativas del negocio; y a través de las acciones y procesos necesarios se producen los resultados esperados.



Planear: Establecer la política del SGSI, objetivos, procesos y procedimientos relacionados con gestión del riesgo y la mejora de seguridad de la información para obtener resultados en conformidad con las políticas generales y objetivos de la organización.

Hacer: Implementar y operar la política de SGSI, controles, procesos y procedimientos.

Verificar: Evaluar y, cuando fuese posible, medir la eficiencia del proceso en contra de la política del SGSI, los objetivos, la experiencia práctica e informar los resultados para su revisión.

Actuar: Tomar acciones correctivas y preventivas, basadas en los resultados de la auditoría interna del SGSI para llevar a cabo una mejora continua del mismo.

Ejemplo 1

Un requerimiento puede llegar a ser que ante una falla de seguridad en la información no cause un daño financiero severo a la organización.

Ejemplo 2

Una expectativa del negocio podría ser que si se produce un incidente grave, como el hackeo de un sitio Web de la organización, debería haber personas con el entrenamiento suficiente para minimizar el impacto.

Sistema de Gestión de la seguridad de la información (SGSI)

Requerimientos generales

La organización debe establecer, implementar, operar, monitorear, revisar, mantener y mejorar un SGSI.

Establecimiento y gestión

La organización debe:

a) Definir el alcance y los límites del SGSI en términos de las características de la empresa, la organización, su ubicación, los activos, la tecnología y detalles; así como la justificación de cualquier exclusión del ámbito de aplicación.

b) Definir una política de SGSI en términos de las características del negocio, la organización, su ubicación, los activos y la tecnología que:

- Incluya un marco para establecer objetivos y un sentido general de dirección y principios en materia de seguridad de la información;
- Tenga en cuenta requerimientos regulatorios, legales y de negocio; y obligaciones de seguridad contractuales;
- Se alinee con el contexto estratégico de riesgo de la organización, en donde se llevará a cabo el establecimiento y mantenimiento del SGSI;
- Establezca los criterios con los que se evaluara el riesgo; y
- Haya sido aprobada por la dirección.

c) Definir una metodología de estimación y evaluación de los riesgos de la organización.

- Identificar una metodología de evaluación de los riesgos que se adaptan al SGSI, a la seguridad de la información, a los requisitos legales y reglamentarios.
- Desarrollar criterios para la aceptación de riesgos, junto con los niveles aceptables.

d) Identificar los riesgos.

- Identificar los activos dentro del alcance del SGSI, junto con sus responsables.
- Identificar las amenazas a esos activos.
- Identificar las vulnerabilidades que podrían ser explotadas por las amenazas.
- Identificar los impactos que las pérdidas de la confidencialidad, integridad y disponibilidad pueden tener sobre los activos.

e) Analizar y evaluar los riesgos.

- Evaluar el impacto que podría resultar de un fallo de seguridad, teniendo en cuenta las consecuencias de una pérdida de confidencialidad, integridad o disponibilidad de los activos.
- Evaluar la probabilidad real de que ocurra un fallo de seguridad, teniendo en cuenta las amenazas actuales, las vulnerabilidades, los impactos asociados a los activos, y los controles actualmente implementados.
- Estimar los niveles de los riesgos.
- Determinar si el riesgo es aceptable o requiere un tratamiento especial.

f) Identificar y evaluar opciones para el tratamiento de los riesgos.

Las posibles acciones incluyen:

- Aplicar controles apropiados;
- Aceptar los riesgos consciente y objetivamente; siempre y cuando cumplan con claridad las políticas de la organización y los criterios de aceptación del riesgo.
- Evitar los riesgos.
- Transferencia de los riesgos a otras partes, por ejemplo, aseguradores, proveedores, etc.

g) Seleccionar los objetivos de control y controles para el tratamiento de los riesgos.

Se deben seleccionar los objetivos de control y controles e implementarlos para contrarrestar los riesgos identificados. Los objetivos de control y controles del anexo A de la norma serán seleccionados como parte de este proceso; pero el negocio puede definir otros objetivos de control que se crear necesarios.

El Anexo A contiene una lista exhaustiva de objetivos de control y controles que se encuentran comúnmente en las organizaciones. Los usuarios de esta norma se dirigen al anexo A como punto de partida para garantizar que no se pasen por alto controles importantes.

h) Obtener autorización de la dirección sobre los supuestos riesgos secundarios.

i) Obtener autorización de la dirección para implementar y operar el SGSI.

j) Preparar una declaración de aplicabilidad.

La declaración debe contener:

- Los objetivos de control y controles, seleccionados y las razones de su selección.
- Los objetivos de control y los controles que se aplican actualmente.
- La exclusión de cualquier objetivo de control y control del anexo A, junto con la justificación de esta decisión.

Implementar y operar

La organización debe:

a) Formular un plan de tratamiento de riesgos que identifique las acciones apropiadas, recursos, responsabilidades y prioridades para la gestión de los riesgos.

b) Aplicar el plan con el fin de alcanzar los detalles en cuanto a la financiación, asignación de roles y responsabilidades.

c) Implementar los controles seleccionados para cumplir con los objetivos.

d) Definir cómo medir la eficacia de los controles seleccionados y especificar cómo estas medidas se van a utilizar para evaluar la eficacia de control.

e) Implementar los programas de entrenamiento.

f) Gestionar las operaciones del SGSI.

g) Gestionar los recursos para el SGSI.

h) Implementar los procedimientos y otros controles que hagan posible la pronta detección y respuesta a incidentes de seguridad.

Supervisar y revisar

La organización debe:

a) Realizar el monitoreo y revisión de los procedimientos y otros controles para:

- Detectar rápidamente errores en los resultados de los procesos.
- Identificar rápidamente intentos y éxitos en la violación de la seguridad.
- Permitir a la administración determinar si las actividades delegadas a responsables o implementadas a nivel informático se están realizando como se esperaba.
- Ayudar a detectar los eventos de seguridad y así evitar incidentes de seguridad mediante el uso de indicadores.
- Determinar si las medidas adoptadas para resolver una violación de la seguridad fueron efectivas.

b) Llevar a cabo revisiones periódicas de la eficacia del SGSI (incluida la reunión de política y objetivos del SGSI y la revisión de los controles de seguridad), teniendo en cuenta los resultados de las auditorías de seguridad, los incidentes, mediciones de eficacia, sugerencias y comentarios de todas las partes interesadas.

c) Medir la eficacia de los controles para verificar que los requisitos de seguridad se han cumplido.

d) Revisar las evaluaciones de riesgo en intervalos planificados y revisar el nivel de los riesgos secundarios y los riesgos aceptables, teniendo en cuenta los cambios a:

- La organización;
- las tecnologías;
- objetivos y procesos de negocio;

- amenazas detectadas;
- eficacia de los controles aplicados; y
- acontecimientos externos, como cambios en el entorno legal o regulatorios, cambios en las obligaciones contractuales y los cambios en el clima social.

e) Realizar, en intervalos planificados, auditorías internas al SGSI.

f) Llevar a cabo un examen de la gestión del SGSI de forma regular para garantizar que el alcance sigue siendo adecuado y las mejoras en el proceso de SGSI están siendo identificadas.

g) Actualizar los planes de seguridad para que tengan en cuenta los resultados encontrados en el monitoreo y en la revisión.

h) Registrar acciones y eventos que podrían tener un impacto en la eficacia o el rendimiento del SGSI.

Mantener y mejorar

La organización debe:

a) Implementar las mejoras identificadas en el SGSI.

b) Adoptar las medidas correctivas y preventivas. Aplicar las lecciones aprendidas de las experiencias de seguridad de otras organizaciones y las de la propia organización.

c) Comunicar las acciones y mejoras a todas las partes interesadas con un nivel de detalle adecuado a las circunstancias y, en su caso, ponerse de acuerdo sobre cómo proceder.

d) Asegurarse de que las mejoras alcancen los objetivos previstos.

Objetivos de control y controles de la norma

Los controles descriptos en el Anexo A de la norma se encuentran todos enmarcados en la seguridad de la información, pero varían en cuanto a sus objetivos. A continuación se especificarán cada uno de estos objetivos de control. En algunos casos, cuando sean objetivos de control que no se encuentran alineados con el marco teórico de mi tesina, solo dando información básica a la que se refieren. En cambio, se desarrollaran extensamente los objetivos de control y controles en donde aprecié una relación con los objetivos de la librería AuditTrail; o incluso objetivos de control importantes para la norma.

A.5 Política de seguridad	
A.5.1 Política de seguridad de la información Objetivo: Proporcionar gestión y apoyo para la seguridad de la información, en conformidad con los requisitos del negocio y con las leyes y reglamentos pertinentes.	
A.5.1.1 Documento de políticas sobre la seguridad de la información.	Control El documento de política de seguridad de la información deberá ser aprobado por la administración, publicado y comunicado a todos los empleados y las partes externas relevantes.
A.5.1.2 Revisión de la política de seguridad de la información.	Control El documento de políticas de seguridad de la información se revisará en intervalos planificados, o si se produjeron cambios importantes, para asegurar su continua adecuación y eficacia.

A.6 Organización de la seguridad de la información.

Los controles de este punto se refieren a gestionar el manejo de la seguridad de la información dentro y fuera de la empresa. Cuando nos referimos a controles fuera de la empresa, estamos hablando de proveedores externos, clientes, etc.

A.7 Gestión de activos	
A.7.1 Responsabilidad en el manejo de activos Objetivo: Conseguir y mantener la protección adecuada de los activos de la organización.	
A.7.1.1 Inventario de activos	Control Todos los activos deberán estar claramente identificados
A.7.1.2 Propietarios de los activos	Control Toda la información y activos asociados con el procesamiento de la información, será propiedad de una parte designada de la organización.

A.7.1.3 Uso aceptable de los activos	Control Reglas para el uso aceptable de la información deberán ser identificadas, documentadas y aplicadas.
A.7.2 Clasificación de la información Objetivo: Garantizar que la información reciba un nivel adecuado de protección.	
A.7.2.1 Guías de clasificación	Control La información deberá ser clasificada en términos de su valor, necesidades legales, sensibilidad y criticidad para la organización.
A.7.1.2 Clasificación y manejo de la información	Control Un conjunto adecuado de procedimientos de etiquetado y manejo de la información se elaborará y aplicará en conformidad con el sistema de clasificación adoptado por la organización.

A.8 Seguridad sobre los recursos humanos.

Controles a tener en cuanta sobre los recursos humanos de la organización. Abarcan controles que debe realizar la organización previos a la contratación, durante el lapso de trabajo del recurso y luego de terminada la relación del recurso.

A.9 Seguridad física y ambiental.

Controles a realizar para impedir el acceso físico no autorizado, daños e interferencias a las instalaciones de la organización y de la información. Estos controles hacen foco en la seguridad a nivel física.

A.10 Gestión de comunicaciones y operaciones.

Este punto se divide en varios ítems en donde algunos se relacionan con la librería AuditTrail.

A.10.1 Los procedimientos operacionales y las responsabilidades.

Los controles abarcan la documentación de los procedimientos operacionales, gestión de los cambios, separación de las funciones, etc.

A.10.2 Gestión de servicios que incluyen terceras partes.

Controles que revisan los servicios que brindan las terceras partes, gestionar servicios provistos a clientes, etc.

A.10.3 Planificación de los sistemas y aceptación

Controles para minimizar el riesgo de fallas en los sistemas.

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.10.6 Gestión de seguridad en la red.

Controles para garantizar la protección de la información en las redes y la protección de la infraestructura de apoyo.

A.10.7 Intercambio de información.

Controles para mantener la seguridad de la información y el software de intercambio dentro de una organización y con cualquier entidad externa.

A.10.8 Servicios electrónicos de comercio.

Controles para garantizar la seguridad de los servicios de comercio electrónico.

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.11 Control de accesos.

En este punto los controles abarcan: Políticas de acceso a la información, prevenir accesos sin autorización, acceso sin autorización a la red, acceso sin autorización a los sistemas operativos, etc.

A.12 Adquisición, desarrollo y mantenimiento de los sistemas de información.

Estos controles abarcan: evitar errores, pérdida, modificación no autorizada o mal uso de la información en las aplicaciones; Controles criptográficos; Seguridad en

el file system; seguridad en el desarrollo y soporte de los procesos; gestión de las vulnerabilidades técnicas; etc.

A.13 Gestión en el manejo de incidentes de seguridad de la información.

A.13.1 Notificación de incidentes y debilidades de seguridad

Objetivo: Garantizar que los eventos y debilidades de la seguridad en la información, asociados con los sistemas de información, sean comunicados de forma que se lleven a cabo las medidas correctoras correspondientes.

A.13.1.1 Notificación de incidentes de seguridad	Control Los eventos de seguridad en la información se comunicarán por canales apropiados lo más rápido posible.
A.13.1.2 Notificación de debilidades en la seguridad	Control Todos los empleados, contratistas y terceros de los sistemas de información y servicios estarán obligados a señalar y denunciar, cualquier sospecha de debilidad de seguridad en los sistemas o servicios.

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 Continuidad de la gestión de negocio.

Controles para contrarrestar las interrupciones a las actividades económicas y proteger los procesos críticos de negocio de los efectos de los fallos de los sistemas de información, asegurando su reanudación oportuna. Los siguientes controles de esta sección están relacionados con los objetivos del AuditTrail:

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 Conformidad

A.15.1 Cumplimiento de los requisitos legales.

Controles para evitar quebrar cualquier ley, estatuto, obligaciones regulatorias o contractuales y cualquier requisito de seguridad.

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.15.2 El cumplimiento de las políticas y normas de seguridad

Controles que abarcan el cumplimiento de las políticas y normas de seguridad; además de cumplir con los estándares de seguridad en los sistemas de información.

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	Control Actividades y requerimientos de auditoría, que impliquen chequeos en los sistemas, deben ser

información	cuidadosamente planificados para minimizar el riesgo de las interrupciones de los procesos de negocio.
A.15.3.1 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.

Caso de estudio teórico: Laboratorio farmacéutico.

Supongamos el caso de un laboratorio farmacéutico. La función de elaborar medicamentos implica documentar procedimientos de elaboración, en donde se detallan los componentes del medicamento junto con las dosis de cada uno y otros detalles importantísimos para la lograr la elaboración correcta.

Además de los procedimientos de elaboración, el negocio, puede definir procedimientos de higiene dentro del establecimiento, así como también cursos donde se describe el uso de los elementos del laboratorio.

Como puede apreciarse **la información que maneja un laboratorio es crítica para llevar a cabo la elaboración de los medicamentos**, por lo tanto es muy probable y conveniente que la misma se encuentre almacenada sobre un soporte informático donde, a través de un sistema, la misma pueda ser accedida y modificada.

Una manipulación errónea en la información puede ocasionar resultados catastróficos para el negocio. Es de esperar que se definan objetivos de control bien marcados centrados en el acceso, en la autorización para modificar y en el monitoreo de la información. Este último objetivo de control es cubierto por la librería AuditTrail, la cual se encarga de registrar todas las modificaciones que ocurran sobre la información.

Veamos un ejemplo práctico: supongamos un medicamento denominado "Ibupirac 400", la información de elaboración de este medicamento es:

Nombre	**Acción Terapéutica**	**Composición**	
Ibupirac 400	Analgésico, antiinflamatorio y Antifebril	Ibuprofeno	400 mg
		Aerosol	2 mg
		Lactosa	20 mg
		Avicel	2 mg
		Ac-Di-Sol	4 mg
		Est de magnesio	10 mg

El laboratorio cuanta con un sistema que permite acceder a esta información y también, para los usuarios autorizados, la modificación de la misma.

El sistema utiliza la librería AuditTrail que deja constancia de todas las modificaciones realizadas sobre los procesos de elaboración de medicamentos, registrando el usuario responsable, la acción realizada, la fecha, los valores viejos y los valores nuevos.

Analicemos diferentes situaciones que pueden darse en el negocio:

1) El negocio solicita crear el procedimiento de elaboración del medicamento "Ibupirac 400". El responsable del sistema crea el procedimiento de elaboración dentro del sistema.
La información registrada por el AuditTrail es la siguiente:

Entidad	Identificador	Usuario	Fecha	Acción
Procedimiento de elaboración	3	jcorvi	05/06/2010	Creación

Valores Auditados: creación, todos los atributos.			
Nombre	**Acción Terapéutica**	**Composición**	
Ibupirac 400	Analgésico, antiinflamatorio y Antifebril	Ibuprofeno	400 mg
		Aerosol	2 mg
		Lactosa	20 mg
		Avicel	2 mg
		Ac-Di-Sol	4 mg
		Est de magnesio	10 mg

2) Luego de una investigación, el laboratorio decide subir la dosis de **Lactosa** de 20 a 25 mg y la dosis de **Avicel** de 2 a 4 mg, por lo tanto el responsable correspondiente debe actualizar el sistema para que todos los empleados de planta tengan la información actualizada.

Supongamos que la información del medicamento es actualizada en forma incorrecta, el responsable modifica la dosis de Lactosa de 20 a 30 mg.

La librería registra la modificación de la siguiente forma:

Entidad	Identificador	Usuario	Fecha	Acción
Procedimiento de elaboración	3	Jcorvi	05/09/2010	Actualización

Valores Auditados: actualización, solo los atributos modificados		
Composición		
	Valor viejo	**Valor nuevo**
Lactosa	20 mg	30 mg
Avicel	2 mg	4 mg

3) Supongamos la misma situación. Luego de la investigación el responsable debe modificar la información correspondiente. La actualización de la información se realiza en forma correcta, pero las pruebas realizadas en el laboratorio fueron de baja calidad y

la dosis era muy alta para personas que sufren alguna enfermedad en particular y esta contraindicación no fue advertida por los investigadores.

Registro de la librería, esta vez con la modificación realizada en forma correcta:

Entidad	Identificador	Usuario	Fecha	Acción
Procedimiento de elaboración	3	Jcorvi	05/09/2010	Actualización
Valores Auditados: actualización, solo los atributos modificados				
Composición				
	Valor viejo		**Valor nuevo**	
Lactosa	20 mg		25 mg	
Avicel	2 mg		4 mg	

4) Podrían ocurrir acciones malintencionadas por parte de usuarios internos o externos a la empresa, modificando o eliminando información sensible del sistema. Supongamos que un hacker burla la seguridad del sistema y elimina el procedimiento de elaboración del medicamento "Ibupirac 400".

La librería AuditTrail registra esta acción de la siguiente forma.

Entidad	Identificador	Usuario	Fecha	Acción
Procedimiento de elaboración	3	Hacker	08/10/2010	Eliminación

Cualquiera fuera la acción que dejo en forma incorrecta la información, debe ser posible detectar al responsable y el momento exacto en que ocurrió la inconsistencia para analizar las causas y tomar acciones.

En todos los casos descriptos anteriormente, la información registrada por el AuditTrail sirve para analizar los errores cometidos y en qué momento ocurrieron. Incluso el caso 3), en donde no hay una modificación incorrecta o maliciosa de la información, sino simplemente un error a nivel de investigación, hay que conocer con exactitud la fecha en que ocurrió la modificación para tomar las medidas correspondientes, las cuales incluyen, sustancialmente, restaurar la información correcta.

Estas situaciones pueden tornarse más complejas a nivel de negocio, ya que si se han realizado partidas de medicamentos con información incorrecta en su elaboración las mismas deben ser retiradas del mercado inmediatamente.

Situaciones similares ocurrirán en cualquier empresa, sobre todo las que tengan como fin la elaboración de algún tipo de producto, ya que probablemente manejen procedimientos y demás información crítica. Existen casos reales, muy similares a este marco teórico que se ha planteado, en donde por errores ocurridos en la elaboración de productos, empresas muy importantes se encontraron en situaciones muy serias. A continuación se agrega un apartado en donde se describen dos casos que se hicieron públicos ya que debieron retirarse del mercado los productos deficientes.

Situaciones reales: Multinacional Mattel y laboratorio Pfizer S.R.L

Uno de los casos más conocidos ocurrió en el 2007 cuando la multinacional de juguetes "**Mattel**" lanzó un aviso mundial "para retirar del mercado todos los ejemplares del muñeco *Sarge*, de la película de dibujos animados *Cars*, porque para fabricarlos se ha utilizado **pintura contaminada con plomo**". En ese entonces se comunicó que la pintura era procedente de China. En la noticia publicada por el diario el País de España [9], se informa que en España se habrían distribuido 10.000 unidades de este juguete. La empresa habilito un teléfono para que las familias que ya lo tengan en su casa puedan llamar para que se les informe como debe hacerse la retirada del producto.

Así mismo, esto no fue el único inconveniente, "…la multinacional del juguete ha extendido la alerta a todos los modelos que lleven incorporado un imán y hayan sido producidos entre 2002 y el 1 de enero de 2007. El problema es que el anclaje es deficiente, y la compañía ha descubierto en sus análisis que **los imanes se pueden desprender; y como son piezas pequeñas un niño puede ingerirlas**".

En total, como indica el diario, en España se distribuyeron **500.000** unidades de modelos con imanes defectuosos. En el resto del mundo se estima que se distribuyeron alrededor de **18,2 millones** de unidades afectadas.

El caso que se describió en el marco teórico, es idéntico al ocurrido en el 2010 cuando se retiraron del mercado 33 lotes alterados de Ibupirac, fabricados por la empresa *Pfizer S.R.L*. La nota del diario "Tiempo Sur" de Río Gallegos[10], describe la situación vivida en la ciudad: "…**según un comunicado de la ANMAT** [11] **(Administración Nacional de Medicamentos, Alimentos y Tecnología Médica), se dispuso la quita del mercado de 33 lotes del analgésico y antifebril denominado Ibupirac,** por presentar alteraciones en el olor y sabor**…"**

Hay innumerables casos de productos que debieron ser retirados del mercado. En la Argentina, la Administración Nacional de Medicamentos, Alimentos y Tecnología Médica (ANMAT) **es el organismo que se encarga de aplicar los sistemas de vigilancia a los productos. Así mismo, se encarga de publicar los productos que deben ser retirados del mercado, los mismos pueden verse en su sitio WEB.**

Capitulo 2: Análisis funcional de la librería AuditTrail

Introducción

El negocio, como hemos visto, puede definir diferentes objetivos de control centrados en la información que se encuentra dentro de un sistema. En la etapa de implementación nos referimos a los mismos como requerimientos de control y deben llevarse a cabo en forma automática.

Existen requerimientos de seguridad básicos que todos los sistemas deben implementar:

- *Autenticación*: verificar la identidad de una persona para permitir el ingreso al sistema.
- *Autorización*: verificar si el usuario que ingreso tiene el permiso para realizar alguna acción determinada dentro del sistema.

Los programadores Java que diseñan e implementar el sistema, pueden optar por realizar su propio componente que se encargue de manejar la seguridad o pueden utilizar algún Framework que se encargue de esta tarea. En la actualidad el Framework más utilizado para implementar la seguridad en un sistema Java es Spring Security.

La librería AuditTrail tiene otro tipo de objetivos de control; cubrir los requerimientos de control enfocados en el monitoreo de la información, ósea en el registro de los cambios que sufre la misma durante su ciclo de vida.

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

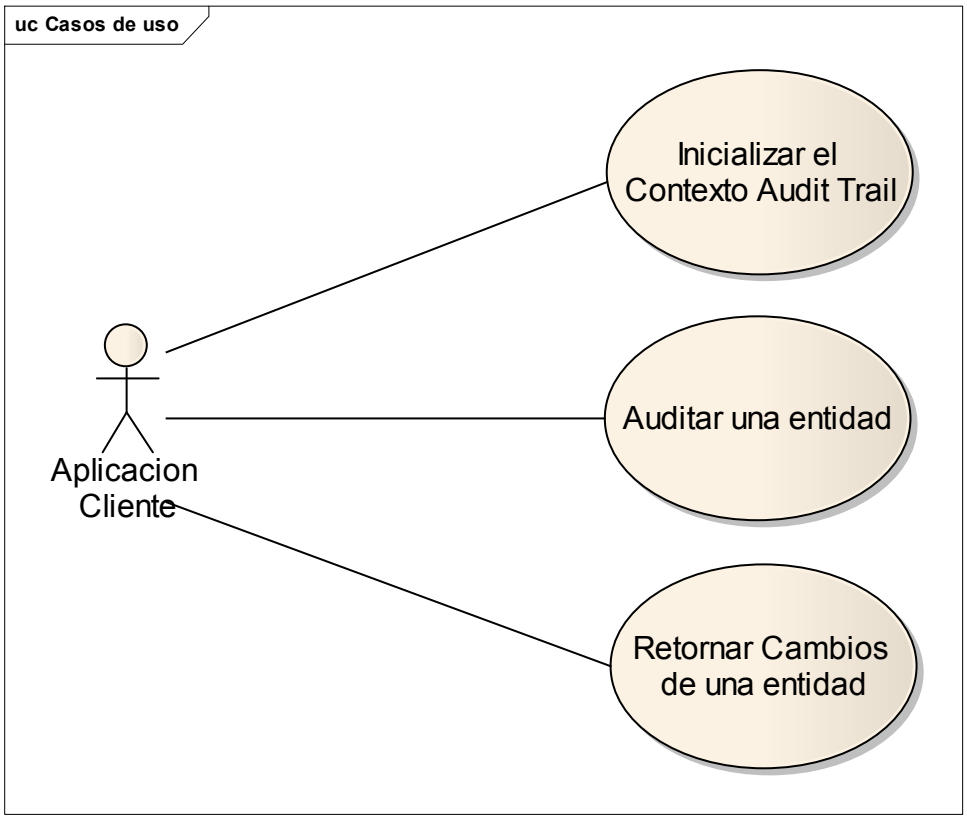
Separar la información generada por el AuditTrail de la información del sistema:

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.

Casos de usos

El sistema que se está auditando funciona como cliente de la librería AuditTrail, a continuación se describen los casos de uso provistos. Para diseñar los casos de uso, diagramas de clases y diagramas de interacción de la tesina se utilizó la herramienta de diseño **Enterprise Architect** [12].



Inicializar el Contexto AuditTrail: El sistema cliente invoca a la inicialización de la librería, la misma queda operativa para comenzar a auditar entidades.

Auditar una entidad: Requerimiento encargado de auditar una entidad. Reconocer los cambios sufridos y persistirlos.

Retornar cambios de una entidad: Requerimiento que retorna todas las modificaciones que ocurrieron sobre una entidad.

Análisis de Auditoría por tipo de operación

Cuando manipulamos un objeto pueden darse tres operaciones que modifican el estado del mismo:

- Creación

  Cuando se crea una entidad no se completan los valores viejos, automáticamente se toman como vacíos. Solamente se indica cuales son los valores del objeto.

- Actualización

  Cuando se actualiza una entidad se completan los valores viejos y nuevos.

- Borrado

  Cuando se borra una entidad solamente se indica el identificador de la entidad que se eliminó.

Análisis de Auditoría por tipo de atributo

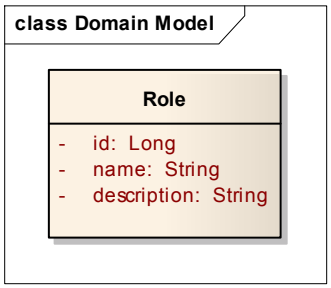
El caso de uso Auditar un objeto del modelo debe tener un análisis a nivel de tipo de atributo que se está modificando. Pueden darse tres casos diferentes:

- Modificación de un atributo simple.
- Modificación de un atributo que referencia a otro objeto.
- Modificación de una colección de una entidad.

A continuación analizaremos cada uno de estos casos, tomando como ejemplo el modelo de dominios de la aplicación WEB StaffingProject.

Modificación de un atributo simple de un objeto:

Es el caso más simple. Nos estamos refiriendo a campos de tipo String, Integer, Boolean, etc. En la aplicación StaffingProject tenemos la entidad Rol con campos simples.



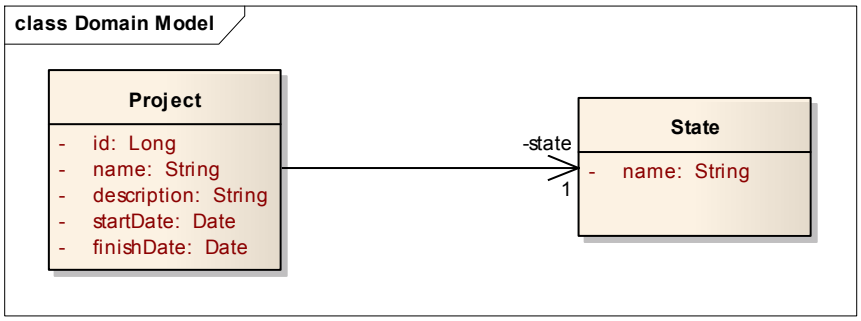
Supongamos que se crea un nuevo rol, la información que queda registrada es:

Entidad	Identificador	Usuario	Fecha	Acción
Role	1	Jcorvi	05/09/2010	Creación

Atributo	Valor viejo	Valor nuevo
name		Programador
description		Desarrollador en php

Modificación de un atributo que referencia a otro objeto:

Este caso se enmarca en la modificación de un atributo que hace referencia a otro objeto, lo que en el modelo de objetos se conoce como **composición de objetos**. En nuestra aplicación WEB StaffingProject un Proyecto contiene una referencia a un objeto Estado, el cual varia durante el ciclo de vida del Proyecto.



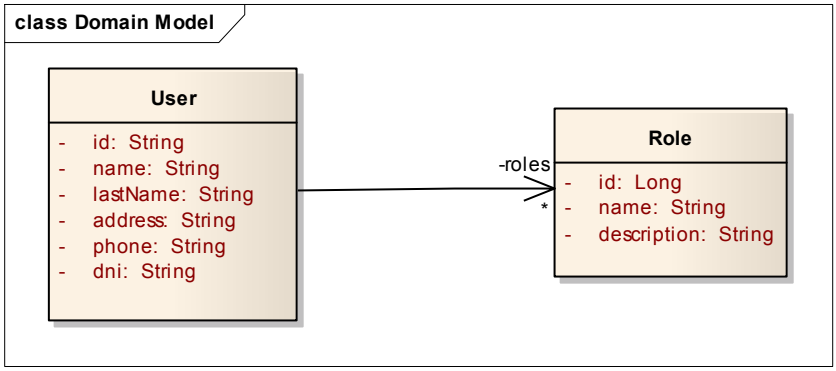
Supongamos que se detiene el proyecto, la información que queda registrada es:

Entidad	Identificador	Usuario	Fecha	Acción
Project	2	Jcorvi	05/09/2010	Actualización

Atributo	Valor viejo	Valor nuevo
state	Ejecución	detenido

Modificación de una colección de un objeto:

En este caso el campo auditado es una colección de objetos. En nuestra aplicación WEB de ejemplo un Usuario contiene un atributo de tipo colección con los roles que el mismo posee. En diferentes ediciones de un usuario se pueden agregar o quitar roles al mismo.



Información que quedara registrada:

Entidad	Identificador	Usuario	Fecha	Acción
User	3	Jcorvi	05/09/2010	Actualización

Atributo	Valores agregados	Valores removidos
roles	Programador, Arquitecto	Tester

Capitulo 3: Aplicabilidad y tecnologías utilizadas

Introducción

Durante el desarrollo de mi tesina se estudiaron diferentes tecnologías, las cuales se describirán en forma precisa en este capítulo.

Algunas de las tecnologías que se utilizaron son base para el desarrollo de la tesina, ósea fueron utilizadas para el desarrollo de la librería AuditTrail o son determinantes para saber si la librería podrá aplicarse en el sistema que se quiera auditar. Por lo tanto a continuación primero nos centramos en la aplicabilidad de la librería a nivel de tecnologías y luego se realizará un repaso de las demás tecnologías utilizadas. Tener en cuenta que algunas tecnologías y técnicas de desarrollo son bastante amplias y abarcan una gran cantidad de funcionalidad, y por motivos obvios se realizará un resumen de las mismas.

Alcance y aplicabilidad de la librería AuditTrail: JPA y Hibernate

El objetivo de esta sección es dejar en claro el alcance de la librería, incluyendo el paradigma de programación y las tecnologías en donde la librería podrá aplicarse para auditar la información.

La librería AuditTrail se encarga de monitorear información que se encuentra representada en el Paradigma Orientado a Objetos (POO). En este paradigma, la información del negocio se define a través del modelo de dominios o modelo de objetos.

El ***modelo de dominio*** (domain model) identifica las entidades[1] de negocio más importantes del sistema y como las mismas se relacionan entre sí para resolver el problema del negocio planteado. Visto desde el paradigma orientado a objetos, el modelo de domino, se representa como una colección de clases, en donde se describen los atributos de cada una, las relaciones entre las mismas y los métodos que implementan la lógica; comúnmente se denomina como diagrama de clases o modelo de objetos.

Durante la ejecución de un programa orientado a objetos se generan instancias de estas clases, denominados objetos, que contienen estado y comportamiento propio.

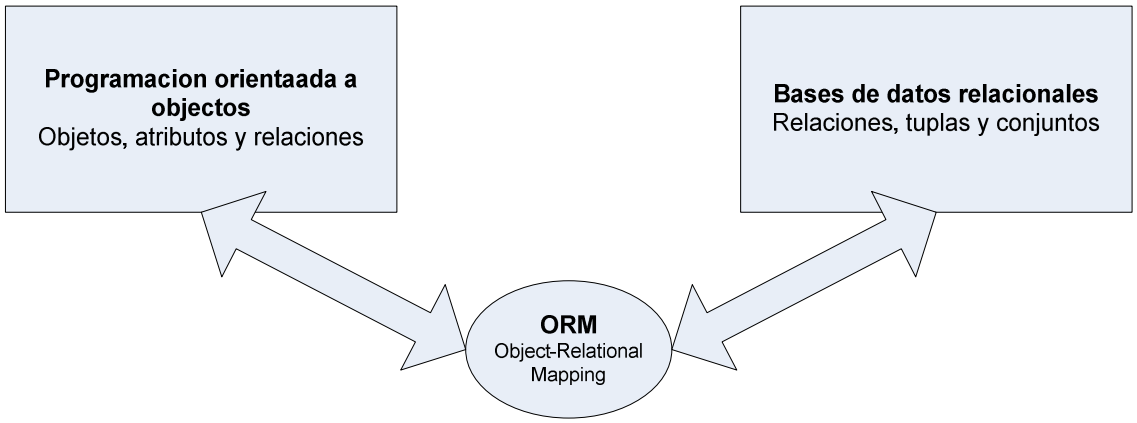
Existen diversos lenguajes orientados a objetos, entre los cuales se destacan **JAVA**, C++, C#, Smalltalk, entre los más conocidos. Mi tesina se encuentra implementada en el lenguaje **JAVA**, que actualmente es el lenguaje más utilizado por la industria para el desarrollo de aplicaciones WEB.

El AuditTrail monitorea las modificaciones que se realizan sobre los objetos del modelo de dominios de una aplicación desarrollada en JAVA; registrando como fue variando su estado, el cual es representado a través de sus atributos.

[1] En la industria, es muy común que se denomine entidad a los objetos del modelo de dominio que pueden persistirse, por lo tanto durante el desarrollo de la tesina se brindará esta denominación cuando el contexto sea el adecuado.

Las bases de datos relacionales siguen siendo las más utilizadas y confiables en la industria. La incompatibilidad manifiesta que se observa entre el mundo orientado a objetos y el relacional ha generado el desarrollo de ORMs. Un ORM o Mapeador Objeto-Relacional es un Framework que propone una nueva forma de modelar los datos, y que permite solucionar la diferencia que existe entre la POO y el modelo relacional.

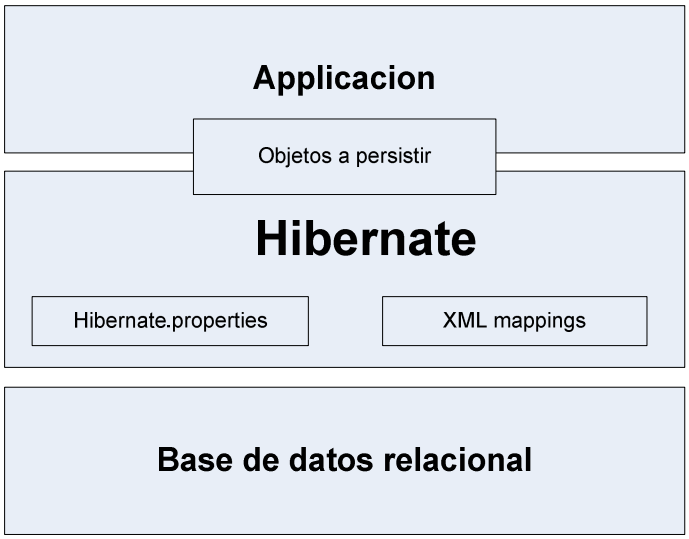
El modelo Relacional se basa en tablas y tuplas por lo cual tiende a ser de carácter matemático, mientras que el modelo de la POO se basa en objetos y las relaciones entre ellos; el problema entre estos dos modelos surge en el momento de querer persistir los objetos de negocio.



Además de realizar la conversión entre los objetos y las tablas, la utilización de un ORM permite abstraerse de la sintaxis de los diferentes motores de base de datos haciendo el código portable a más de un motor, sin necesidad de recompilar el mismo.

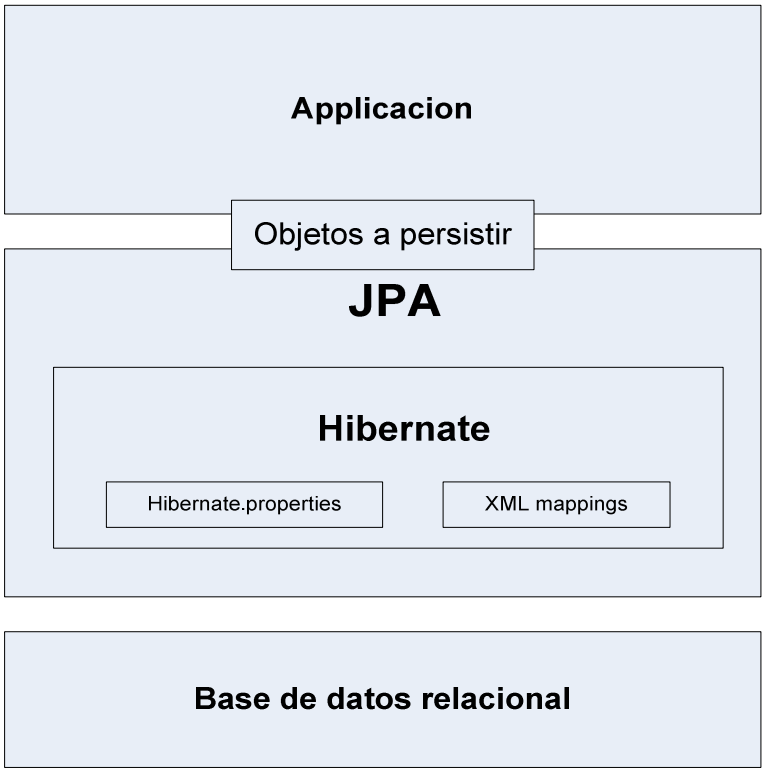
Existen diversas implementaciones de ORMs para Java, entre las más importantes se encuentran **Hibernate**, TopLink (Oracle), IBatis y Eclipse Link, entre otras.

En el desarrollo de mi tesina se utilizó **Hibernate**, aquí vemos como se conforma su arquitectura:



La elección de Hibernate radica en todas las ventajas que posee al ser el Framework de persistencia más utilizado en la industria: eficiencia, facilidad de uso, estabilidad (actualmente se encuentra en la versión 3), documentación, etc.

A partir de JAVA 5 se agrego la **Java Persistente API (JPA)**. Esta API establece una interfaz común para realizar el mapeo objeto-relacional. La misma es implementada por un proveedor de persistencia de nuestra elección, en nuestro caso **Hibernate**, de manera que podemos elegir en cualquier momento el proveedor que más se adecue a nuestras necesidades**.** Así, es el proveedor quien realiza el trabajo, pero siempre funcionando bajo la API de JPA.



La librería AuditTrail se encuentra desarrollada para aplicaciones que utilicen JPA junto con el Framework Hibernate. Gracias a la estandarización de JPA, podríamos cambiar de ORM sin inconvenientes.

El mapeo entre el objeto de una clase y la tabla física en donde será persistido se realiza en un archivos descriptor xml con extensión .hbm (Hibernate mapping). Solo como ejemplificación a continuación se muestra un mapeo realizado a una entidad User del modelo del análisis funcional desarrollado anteriormente:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="facultad.multimedia.staffing.model">
  <class name="User" table="users">
    <id name="id" column="id">
        <generator class="native"></generator>
    </id>
    <property name="lastName" type="java.lang.String" column="lastname" not-null="true"/>
    <property name="name" type="java.lang.String" column="name" not-null="true"/>
    <property name="dni" type="java.lang.Integer" column="dni" not-null="true"/>
    <property name="address" type="java.lang.String" column="address" not-null="false"/>
    <property name="phone" type="java.lang.String" column="phone" not-null="false"/>
    <bag name="roles" table="user_role"  lazy="false">
        <key column="id_user"/>
        <many-to-many column="id_role" class="Role" />
    </bag>
  </class>
</hibernate-mapping>
```

De esta forma se indica en que tabla se mapearan los usuarios y la información correspondientes a los atributos con sus respectivas columnas.

Además, para completar las ventajas de abstracción del motor de base de datos, Hibernate, provee un lenguaje de consultas HQL (Hibernate Query Languaje), el cual tiene una sintaxis orientada a objetos muy fácil de utilizar.

Hibernate es un Framework muy amplio, con mucha utilidad en la industria y que posee muchas ventajas en su uso, si el lector desea indagar mas sobre el mismo en la bibliografía se encontrarán libros muy útiles, además de la documentación oficial de la página.

Tecnologías estudiadas e implementadas

Anotaciones

A partir de JAVA 5 se agregó a la especificación de JAVA las ***Anotaciones*** [13]. Las mismas brindan una forma simple de agregar metadatos al código fuente; ya sea a una clase, un método, un atributo, un parámetro o un paquete. Muchas veces se usa como una alternativa a la tecnología XML

La sintaxis para crear una anotación es similar a la de una interfaz con la diferencia que se utiliza el símbolo @ en forma precedente, por ejemplo:

```
public @interface AuditTrailAnnotation {
        ..............
}
```

Las anotaciones pueden contener elementos que describen a la misma para ser utilizada:

```
public @interface AuditTrailAnnotation {
        String action();
}
```

Para utilizar las anotaciones se agrega @ seguido por el nombre de la anotación y entre paréntesis una lista de pares elemento-valor:

@AuditTrailAnnotation(action="Actualización").

Las anotaciones pueden incluir meta-anotaciones descriptivas sobre la misma anotación creada, por ejemplo:

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface AuditTrailAnnotation {
        .....
}
```

@Retention(RetentionPolicy.*RUNTIME*):
Significa que la anotación se mantiene en tiempo de ejecución; de esta forma los programadores, a través de reflexión, pueden manipular el programa que se está ejecutando. Por ejemplo, podrían leer los parámetros de la anotación para realizar algún tipo de cómputo.

@Target(ElementType.*METHOD*):
Significa que la anotación se aplicara solamente a métodos dentro del programa.

Existen otras meta-anotaciones, que no se utilizan en mi tesina, las mismas pueden verse en la documentación correspondiente.

Durante la descripción del diseño de la librería AuditTrail, veremos más en detalle la anotación desarrollada para la librería.

Programación Orientada a Aspectos (AspectJ)

Ya hemos comentado que el requerimiento de auditar entidades del modelo de objetos se enmarca como un *requerimiento* o *concern* denominado *no funcional*. Una implementación adecuada y práctica para desarrollar estos requerimientos es realizarlo con AOP.

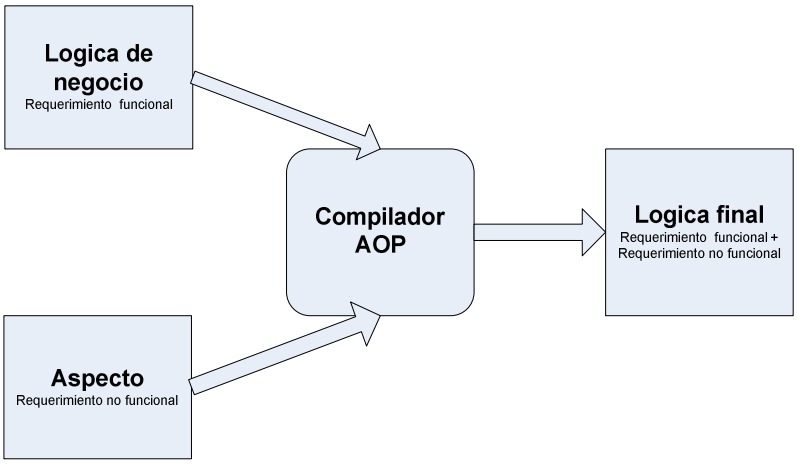
Los beneficios que brinda AOP son básicamente varios de los objetivos de diseño que se busca continuamente en un sistema:

- Mayor grado de modularización
- Reusabilidad de código
- Código menos confuso

AspectJ es una extensión del lenguaje JAVA que agrega los conceptos básicos de AOP. Provee un compilador capaz de interpretar los aspectos, ***extensión .aj***, junto con el código fuente .java; integrando así, la lógica de los aspectos con la lógica de negocio.

Actualmente el desarrollo y el soporte de **ajc** se encuentran en manos del Proyecto Eclipse, el cual además, brinda un entorno de programación de aspectos

denominado AJDT (Aspecto Development Tools) que es utilizado para el desarrollo e integración de aspectos en mi tesina.



No es objetivo de mi tesina dar una descripción detallada del funcionamiento de AspectJ; aquí se describirán los conceptos básicos, para mayor documentación referirse a la bibliografía utilizada.

Aspect

Un aspecto es la ***unidad de modularización en AOP***. Equivale a una clase en el paradigma orientado a objetos. La declaración de un aspecto es muy parecida a la de una clase, pero su extensión es .aj.

```
public aspect AuditTrailAspect{
    ……..
}
```

Join points

Los join points son los ***lugares de interés dentro del programa*** en donde el requerimiento no funcional, que modela el aspecto, debe integrarse Un join point podría ser, métodos que comiencen con "save", métodos que retornan valores boléanos o métodos que contengan una Anotación.

Pointcut

Colección de join points en donde el requerimiento no funcional debe integrarse. Todos los métodos que comiencen con la palabra "save":

```
pointcut auditTrail(Object object) : execution(* com..*.save *());
```

Todos los métodos que contengan la Anotación "AuditTrailAnnotation":

```
pointcut auditTrail(Object object)
```

```
: execution(@AuditTrailAnnotation * *(..)) && args(object);
```

Advice

Es el ***código ejecutado cuando se alcanza un join point*** seleccionado por un pointcut. Los advice ***pueden ejecutarse antes, después o alrededor*** de los join points. El cuerpo de un advice es similar al de un método, ***encapsula la lógica de requerimiento no funcional***.

```
Object around(Object object) : auditTrail(object) {
        ……
}
```

Weaving

Se denomina weaving al momento o instancia en donde la lógica del aspecto se integra con las clases. AspectJ provee tres modelos de weaving:

- **Source weaving**
- **Binary weaving**
- **Load-Time weaving**

Source weaving

La integración es parte del **compilador de aspectos ajc**; la entrada al ***"weaver"*** consiste en clases y aspectos en código fuente. Se procesan los archivos fuentes generando así byte-codes que pueden ser interpretados por cualquier **JVM.** Cuando se utiliza de esta forma **el compilador de aspectos ajc remplaza al javac. Ósea los .class generados ya tienen integrada la lógica de los aspectos.**

Tener en cuenta que si deseamos trabajar con este tipo de weaving **todos los archivos deben estar presentes al momento de la compilación** para poder generar los byte-codes con la lógica integrada. Si el código de los aspectos y de las clases se presentan en forma separada deberá utilizarse Binary o Load-time weaving.

Binary weaving

En el weaving binario, la entrada al "weaver" consiste en archivos ya compilados que se encuentran en byte-code. La idea es compilar las clases y los aspectos en forma separada, sin que se integren, y luego realizar el weaving con los .class generados.

Con el weaving binario podemos **compilar aspectos en forma separada e incluirlos dentro de una librería .jar**.

Load-Time weaving

En load-time weaving las entradas consisten en archivos binarios de clases y aspectos. En load-time **un agente se encarga de realizar el weaving cuando las**

clases son cargadas por la VM (Virtual Machine). Cuando utilizamos LTW **es necesario configurar un archivo aop.xml** que contiene la configuración de weaving.

Posteriormente, durante el análisis de la Arquitectura de la librería AuditTrail, se describirá en detalle el aspecto desarrollado para encapsular la lógica de la auditoría.

Struts 2

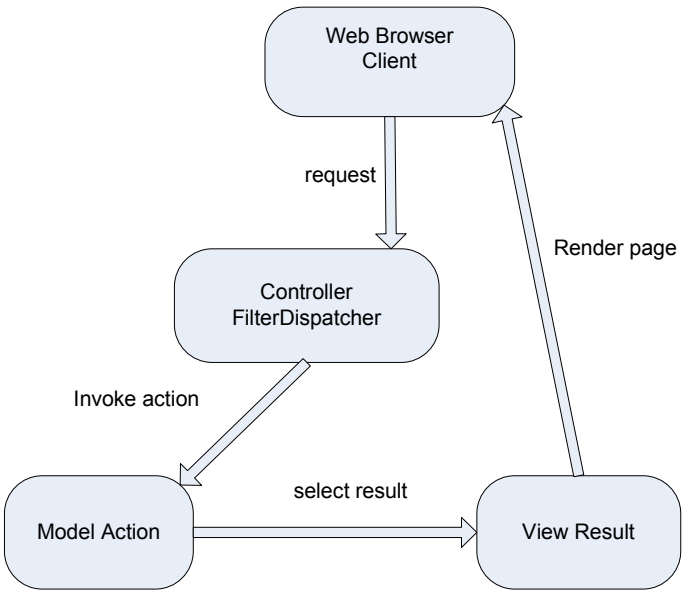
La arquitectura básica de las aplicaciones WEB escritas en JAVA se encuentra modelada con Servlets. Los mismos se encargan de atender los requerimientos que provienen desde la interfaz (paginas html/jsp) y de brindar una respuesta. Si bien es una técnica válida e incluso muy utilizada actualmente; hoy en día existen diversos Frameworks WEB, como es el caso de **Struts 2** [14] [15], que proveen **innumerables ventajas** que simplifican las tareas comunes a la hora de llevar a cabo el desarrollo de una aplicación WEB.

Struts 2 es el Framework Web que se utilizó durante el desarrollo del proyecto StaffingProject; proyecto auditado por la librería AuditTrail en mi tesina.

Entre sus objetivos principales se encuentran el de automatizar la atención de requerimientos que llegan desde las páginas de la interfaz a través de la configuración del archivo descriptor ***struts.xml***. En dicho archivo se indica por cada requerimiento cual va a ser el ***Action*** y el método que va a atender la petición Web.

Además, Struts 2 brinda un conjunto de librerías especiales de tags para utilizar dentro de las paginas .jsp; las mismas facilitan el desarrollo de las paginas.

A nivel de arquitectura se dice que **Struts 2 permite desarrollar una aplicación siguiendo el patrón MVC (Model View Controller).**



Cuando llega un requerimiento el FilterDispacher, utilizando la información que se encuentra en el **struts.xml**, se encarga de delegar al **Action** correspondiente para que el mismo atienda la petición y retorne el resultado.

Ajax

AJAX (Asynchronous JavaScript And XML) [15] se utiliza para crear aplicaciones Web interactivas; el cliente mantiene comunicación asincrónica en segundo plano hacia el servidor, permitiendo realizar cambios en la las páginas sin necesidad de recargarlas completamente.

Struts 2 brinda soporte y una buena integración con la tecnología AJAX a través de los plugins que provee: Dojo Plugin (utilizado en la tesina) y JQuery Plugin.

Actualmente, Struts 2, dada su maduración y documentación disponible, es el Framework más utilizado en la industria para la construcción de la parte Web Vista-Controlador de las aplicaciones.

Personalmente lo he utilizado en innumerables proyectos, habiendo siempre cumplido con mis expectativas.

Spring Framework

Es un Framework para desarrollo de aplicaciones J2EE considerando fuertemente la filosofía: "**J2EE should be easy tu use**".

Surge como una alternativa a los EJBs (v2.1) para desarrollar aplicaciones distribuidas, ya que éstos presentan un ambiente demasiado complejo.

En un principio, Spring era sólo un integrador de las mejores tecnologías disponibles, sin hacer por si mismo ninguna tarea. Con el tiempo esto ha ido cambiando y el mismo Framework **provee alternativas para cada una de las capas de una aplicación**.

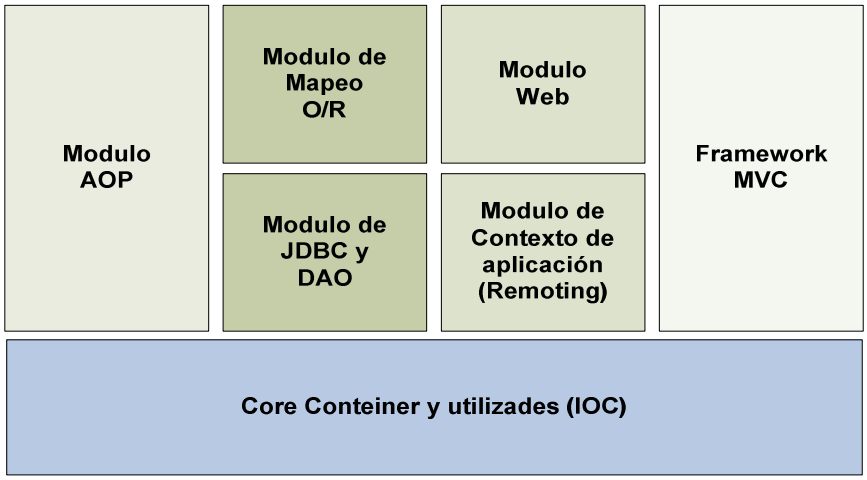
Arquitectura de Spring:

Contenedor de beans.

Cada bean representa un pojo y permite agregarle declarativamente todos los servicios que ofrece Spring a sus beans.

Un bean en Spring es una clase Java definida en un XML.

Módulos de Spring



Como podrá apreciarse Spring provee módulos para todas las capas de la aplicación. Incluso para la capa Web, con **Spring MVC, el cual sería la "competencia" de Struts 2**. En mi experiencia personal es más cómodo y tengo preferencias en utilizar Struts 2 sobre Spring MVC, pero va en el gusto de cada programador y en ver las ventajas y desventajas de cada uno. Vale aclarar que **la integración entre Spring y Struts 2 es muy simple de realizar y no trae complicaciones**.

Así mismo provee un modulo de persistencia el cual **se integra perfectamente al uso de JPA con Hibernate.**

Brinda otra alternativa, a través de su modulo de AOP, para utilizar Aspectos; no se va a ahondar más en este tema, solo se comentará que para mantener una independencia entendible yo prefiero utilizar Aspectos puros implementados en AspectJ, sin depender de ningún Framework.

El corazón de Spring es el Core Container. A través la declaración de beans y la inyección de dependencia, el Core Container permite entrelazar todos los componentes de la aplicación. A continuación se dará una descripción de su funcionamiento.

Core Container:

- Provee la funcionalidad fundamental de Spring.
- Su componente principal es la "BeanFactory".
- BeanFactory es una implementación del patrón de diseño Factory.
- Las dependencias entre los distintos elementos se especifica a través del principio de Inversión de Control (IOC). El mismo se implementa a través de Inyección de Dependencia.

Implementación XMLBeanFactory

Los beans que contiene están especificados en XML:

```
<bean id="daoEmpledo" class="modelo.DaoEmpleado">
```

Inyección de dependencia (Dependency Injection)

Las dependencias de los objetos se externalizan, de modo que el contenedor inyecta todas las "necesidades" en el momento requerido.

Las búsquedas de los recursos, o incluso el conocer el nombre específico de las clases de los colaboradores son removidos completamente del código.

Se puede utilizar:

Setter injection (a través de las propiedades del java bean).
Constructor injection (a través de argumentos en el constructor).

Ejemplo Inyección de dependencia:

```
public class ServicioEmpleado{

    private EmpleadoDao empleadoDao;
```

```
        public void setEmpleadoDao (EmpleadoDao empleadoDao) {
              this.empleadoDao = empleadoDao;
          }

        public EmpleadoDao getEmpleadoDao() {
             return this. empleadoDao;
        }
        .....
}
```

El xml sería:

```
<bean id="servicioEmpledo"  class="modelo.ServicioEmpleado">
        <property name="daoEmpleado">
                <ref bean="daoEmpleado"/>
        </property>
</bean>
```

Otras ventajas de Spring:

Lightweight: Es liviano (solo un jar pequeño). No es necesario un application server para disponer del contenedor.
No intrusivo: Los POJO´s no dependen de clases de Spring.
Documentación completa: Muy usado en la industria.

Desventajas

Cuando la aplicación es pequeña no se justifica su uso.
Si se quieren utilizar todos sus módulos y componentes existe una curva de aprendizaje importante.

Spring Security

Como se describió en el análisis funcional de la librería, existen requerimientos de seguridad básicos que todos los sistemas deben implementar: *Autenticación y Autorización.* En la actualidad el Framework más utilizado para implementar la seguridad en un sistema Java es Spring Security.

Durante mi tesina se utilizó Spring Security en el desarrollo del proyecto StaffingProject, solo para la autenticación al sistema, ósea solo para en el login a la aplicación. No tenía sentido agregarlo para la autorización ya que el sistema StaffingProject no requería otros aspectos de seguridad.

Spring Security provee otras ventajas, las cuales he utilizado laboralmente, como lo son: Autenticación LDAP, tags de seguridad para los jsp, Autenticación Anónima, y muchísimas otras ventajas que escapan al desarrollo de mi tesina y que podrían ahondarse, si el lector está interesado, en la bibliografía correspondiente.

Entonces para resumir, utilizamos Spring para automatizar la interacción entre los componentes y layers del sistema a través de Dependency Injection.

Además de agregar a la aplicación Aspectos (AOP) para manejar requerimientos no funcionales, como lo es por ejemplo el manejo de transacciones, logging, etc.; y además se utiliza Spring Security para manejar la autenticación.

Otra vez se hace hincapié, que tanto Spring, Struts 2 y Hibernate son Frameworks muy utilizados en la industria, que poseen una gran cantidad de funcionalidad y brindan muchas ventajas al implementarlos en los escenarios correctos. En mi experiencia profesional, en los últimos años los he utilizado en innumerables ocasiones junto con una arquitectura de n-capas y los resultados han sido muy satisfactorios. Los sistemas quedan muy ordenados, escalables, fáciles de entender y modificar. El resultado es una arquitectura limpia en donde cada componente queda bien ordenado y cumple un rol bien delimitado en el sistema.

En este último tiempo, desde que comencé a estudiarlos en la tesina, he agregado Aspectos junto con Anotaciones en varios sistemas existentes para realizar tareas de validación, caching de excepciones y loggin. A partir de ahora cada requerimiento no funcional, que los Framework no estén cubriendo actualmente, es implementado con Aspectos.

Ambiente de ejecución: Tomcat y MySql

Para realizar el desarrollo de la librería AuditTrail se usaron las tecnologías open source más utilizadas en la industria. Nos estamos refiriendo al motor de base de datos **MySql [16]** y al contenedor Web **Tomcat [17]**.

MySql es el sistema de gestión de base de datos relacional open source con mayor aceptación en la industria; provee una licencia GNU gratuita y, para empresas existe una licencia especial de uso comercial. Durante el desarrollo de mi tesina utilicé la versión 5.1.

Desde el 2008, MySql es desarrollado por Sun Microsystems, empresa que fue adquirida en el 2009 por Oracle.

Personalmente he utilizado este motor de base de datos durante toda mi carrera, es liviano, simple y creo que es la opción viable a la hora de implementar sistemas a nivel universitario. Pero sin lugar a dudas en la industria los dos motores de base de datos más utilizados son SQLServer y Oracle. La elección de los mismos está dada por su robustez, potencia, seguridad, etc. Están orientados a sistemas que manejan un gran volumen de datos.

Apache Tomcat fue el contenedor Web utilizado para realizar la aplicación StaffingProject en mi tesina. Es una implementación open source de las especificaciones de Servlets y de Java Server Pages (JSP). En este proyecto participan en su desarrollo programadores de todo el mundo, colaborando para mejorar continuamente la calidad del producto.

Tomcat es el contenedor que tengo como preferencia y fue el que utilice desde que comencé a realizar desarrollos Web con Java (hace mas de 8 años). Puede recalcarse que el contenedor Tomcat es el más utilizado en la industria incluso por empresas grandes, esto nos da una idea de la robustez y calidad del mismo. Otros contenedores importantes utilizados, pero en menor medida, son Websphere y Glassfish.

Capitulo 4: Diseño y Arquitectura de la librería AuditTrail

Introducción

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 cuales fueron desarrollados utilizando las tecnologías indicadas en el capitulo anterior.

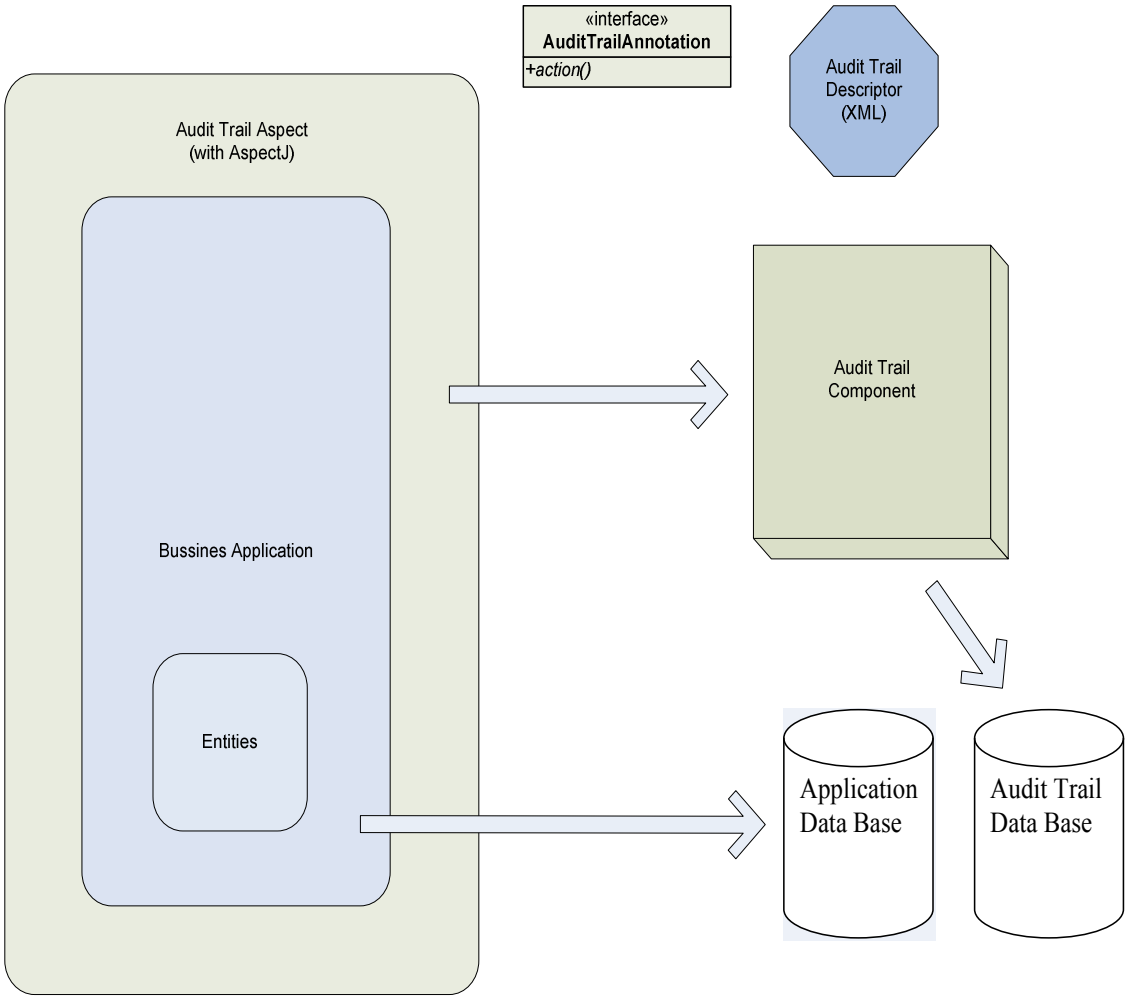
Los elementos que la componen son:

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:



AuditTrail Descriptor

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.

El archivo descriptor del ejemplo funcional es:

```
<entities>
    <!-- declaracion de la entidad a auditar, se agrega el nombre de la clase -->
    <entity name="facultad.multimedia.staffing.model.Role">
        <identifier>id</identifier> <!-- indica el nombre del identificador -->
        <fields> <!-- coleccion de campos a auditar -->
            <field> <!--campo a auditar -->
                <!--nombre del atributo -->
                <name>name</name>
                <!--i18n del atributo (para visualizacion) -->
                <i18nname>comm.i18n.role.name</i18nname>
            </field>
            <field>
                <name>description</name>
                <i18nname>comm.i18n.role.description</i18nname>
            </field>
        </fields>
    </entity>
    <entity name="facultad.multimedia.staffing.model.User">
        <identifier>id</identifier>
        <fields>
            ..............
            <field>
                <name>roles</name>
                <i18nname>comm.i18n.user.roles</i18nname>
                <!--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>
        </fields>
    </entity>
    <entity name="facultad.multimedia.staffing.model.Project">
        <identifier>id</identifier>
        <fields>
            ...........
            <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[2], invocar los ***getters*** del objeto que está

[2] 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.

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 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 roles de la entidad User.

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.

AuditTrail Annotation

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.

```
/**
 * Anotación que se utiliza cuando se quiere auditar un método del sistema
 * @author javi
 *
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface AuditTrailAnnotation {
	/**
	 * Enumeration para tipo de acción
	 * @author javi
	 *
	 */
	public enum ActionType { CREATE, UPDATE, DELETE};
	/**
	 * Tipo de accion que se ejecuta
	 * @return
	 */
```

```
        ActionType actionType();
        /**
         * Nombre del atributo en el método que debe ser auditado
         * @return
         */
        String auditedObject();
        /**
         * Accion descriptiva que se ejecuta
         * @return
         */
        String action();
}
```

Ejemplo de declaración de la anotación en un método:

```
/**
* Creación de un objeto
 * @param object
*/
@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.

AuditTrail Aspect

Anteriormente se realizó una introducción a AOP con AspectJ, 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.

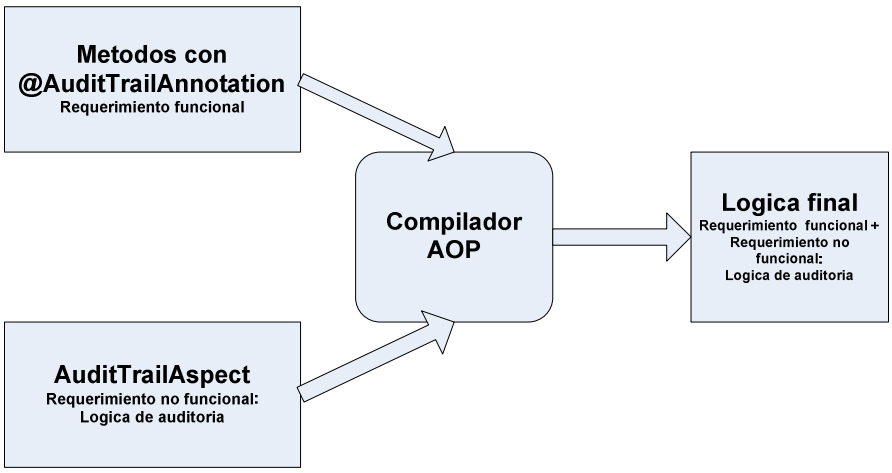
Aspecto desarrollado dentro de la librería:

```
package info.unlp.edu.ar.tesis.audit.aspect;

import .....
/**
 * 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 join point métodos con 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:



Weaving

La idea de mi tesis es generar una librería .jar, a partir de aquí denominada **audittrail.jar**, que contenga solamente los archivos binarios (byte codes). Por lo tanto el weaving debe ser Binario. El programador que utilice la librería no tiene porque conocer el código fuente de la misma; solamente debe agregarla como librería de la aplicación y luego indicar que la misma es una librería que contiene aspectos que afectan al proyecto. El ambiente eclipse provee el contexto adecuado, a través de AJDT, para realizar el deploy en forma simple. La instalación y las herramientas propuestas se verán más en detalle en sus secciones correspondientes.

Análisis del funcionamiento del Aspecto y la Anotación

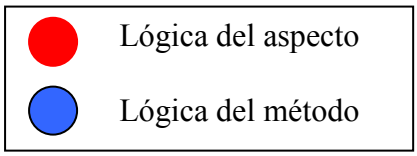
Analizando el funcionamiento del aspecto AuditTrailAspect junto con la anotación AuditTrailAnnotation, notamos que la auditoría se realiza sobre un objeto que es pasado como parámetro en un método.

```
@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.

Por ejemplo:

```
public void createUser(String name, String email,Integer age){
        //lógica de negocio para crear al usuario
        …
        //lógica de persistencia
        entityManager.getTransaction().begin();
        entityManager.persist(user);
        entityManager.getTransaction().commit();
        //más lógica de negocios
}
```

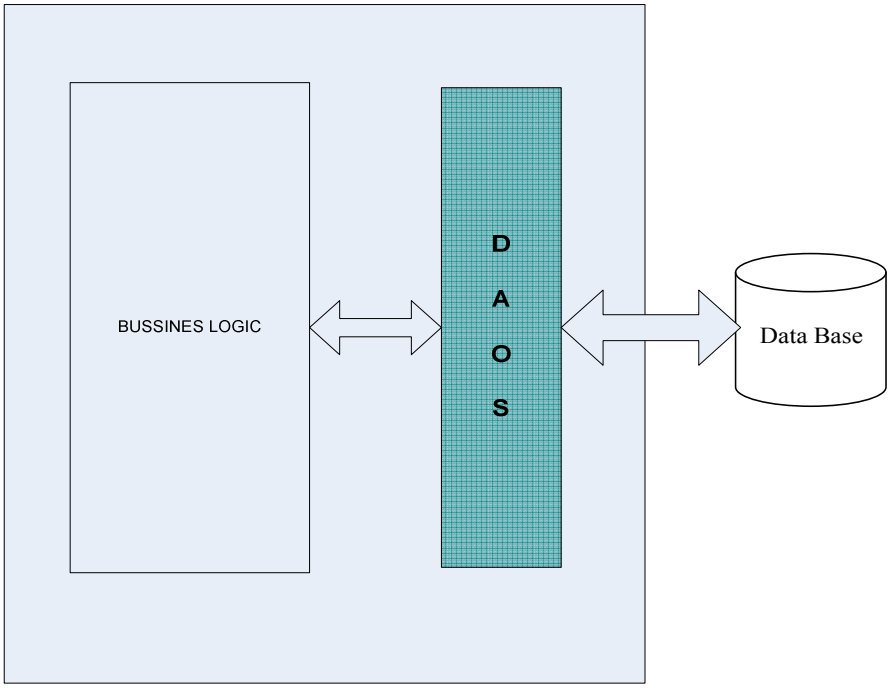
En este caso la lógica de persistencia no se encuentra modularizada-separada en un método propio. Por lo tanto no se puede incluir la anotación, y la librería no puede implementarse. Si analizamos la situación nos daremos cuenta de que no respeta la modularización adecuada para que el aspecto pueda llegar a alcanzar algún join point.

Si refactorizamos de la siguiente forma:

```
public void createUser(String name, String email,Integer age){
        //lógica de negocio para crear al usuario
        …
        //invocación a la lógica de persistencia
        this.persistUser(user);
        //más lógica de negocios
}
@AuditTrailAnnotation(auditedObject="user",
actionType=ActionType.CREATE,action="com.action.create")
private void persistUser (User user){
        entityManager.getTransaction().begin();
        entityManager.persist(user);
        entityManager.getTransaction().commit();
}
```

Ya podemos agregar la anotación al método correspondiente y es posible implementar el AuditTrail.

En la actualidad, es muy común que las aplicaciones deleguen la comunicación con la base de datos en una capa transaccional de DAOs (Data Access Object); patrón Core J2EE DAO (Data Access Object) Layer [18].



Bussines Logic.

```
public void createUser(String name, String email,Integer age){
        //lógica de negocio para crear al usuario
        …
        //invocación a la capa de persistencia
        dao.persistUser(user);
        //más lógica de negocios
}
```

DAO Logic

```
@AuditTrailAnnotation(auditedObject="user",
actionType=ActionType.CREATE,action="com.action.create")
private void persistUser (User user){
        entityManager.getTransaction().begin();
        entityManager.persist(user);
        entityManager.getTransaction().commit();
}
```

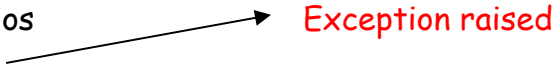
El método que está siendo auditado, el que se encarga de persistir el objeto, ***debe ser transaccional*** para que no ocurran inconsistencias con la auditoría.

```
@AuditTrailAnnotation(auditedObject="user",
actionType=ActionType.CREATE,action="com.action.create")
private void persistUser (User user){
        entityManager.getTransaction().begin();
        entityManager.persist(user);
        entityManager.getTransaction().commit();
}
```

De esta forma nos aseguramos de que cuando se persista la información recopilada por el AuditTrail la transacción a terminado y el objeto fue persistido correctamente.

Si el diseño transaccional se maneja en forma desprolija, el monitoreo realizado por el AuditTrail puede ser inconsistente. Veamos el siguiente ejemplo:

```
public void createUser(String name, String email,Integer age){
        entityManager.getTransaction().begin();
        //lógica de negocio para crear al usuario
        ...
        //invocación a la capa de persistencia
        dao.persistUser(user);
        //más lógica de negocios
        //envío de mail
        ...
        entityManager.getTransaction().commit();
}
```




```
@AuditTrailAnnotation(auditedObject="user",
actionType=ActionType.CREATE,action="com.action.create")
private void persistUser (User user){
        entityManager.persist(user);
}
```

Aquí la transacción se inicia fuera del método auditado y agrupa demasiadas sentencias que no tienen que ver con una operación atómica en la base de datos. No es un diseño transaccional eficiente.

Que pasaría si ocurre una excepción luego de invocar al método persistUser. El AuditTrail no se percataría de este hecho y registraría la información de la auditoría; luego la transacción fallaría por un error de lógica y no se llevaría a cabo la persistencia del objeto, quedando inconsistente la información de la auditoría.

En resumen, para implementar la librería AuditTrail, el sistema cliente debe tener un diseño correcto en cuanto a la modularización y al manejo transaccional. Caso

contrario la librería no podrá ser implementada o podría llegar a registrar información incorrecta.

Observación:

Cuando indicamos que la librería "no va a poder ser implementada" si no existe una modularización correcta en la persistencia, en realidad lo que ocurre, como se ha indicado, es que no podrá utilizarse el aspecto definido dentro de la librería; pero no habría ningún inconveniente en introducir las invocaciones al componente de auditoría dentro de la lógica de negocio:

```
public void createUser(String name, String email,Integer age){
        //lógica de negocio para crear al usuario
        AuditTrailFacade auditTrail = AuditTrailFacade.getInstance();
        entityChange = auditTrail.audit(entity, annotation.actionType()
        ,annotation.action());
        //lógica de persistencia
        entityManager.getTransaction().begin();
        entityManager.persist(user);
        entityManager.getTransaction().commit();
        auditTrail.save(entityChange);
        //más lógica de negocios
}
```

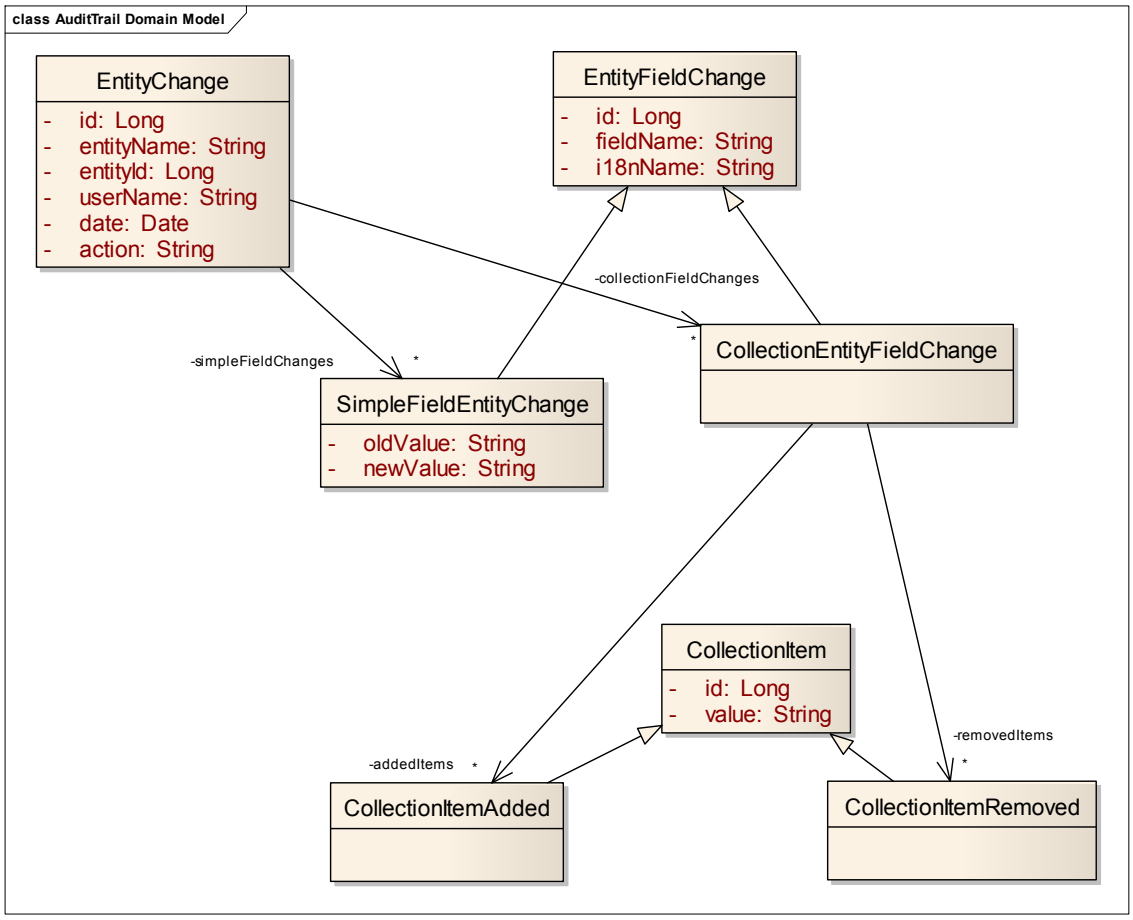
Si bien de esta forma podríamos utilizar la lógica de la librería; no se cumpliría con uno de los objetivos propuestos en la tesis, el de separar la lógica de auditoría de la lógica de negocio. Podríamos verlo como una forma alternativa para sistemas que no se encuentran bien diseñados.

AuditTrail Component

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.

En primera instancia debemos resolver el problema del negocio planteado, ósea diseñar el ***modelo de dominio*** (domain model) que identifica las entidades de negocio más importantes de la librería y como las mismas se relacionan entre sí para resolver el problema de auditar objetos.

Modelo de dominio



EntityChange:

Representa una modificación que se realizo sobre una entidad. Contiene las colecciones con los cambios a los atributos simples y una colección con los cambios a atributos de tipo colección.

EntitySimpleFieldChange:

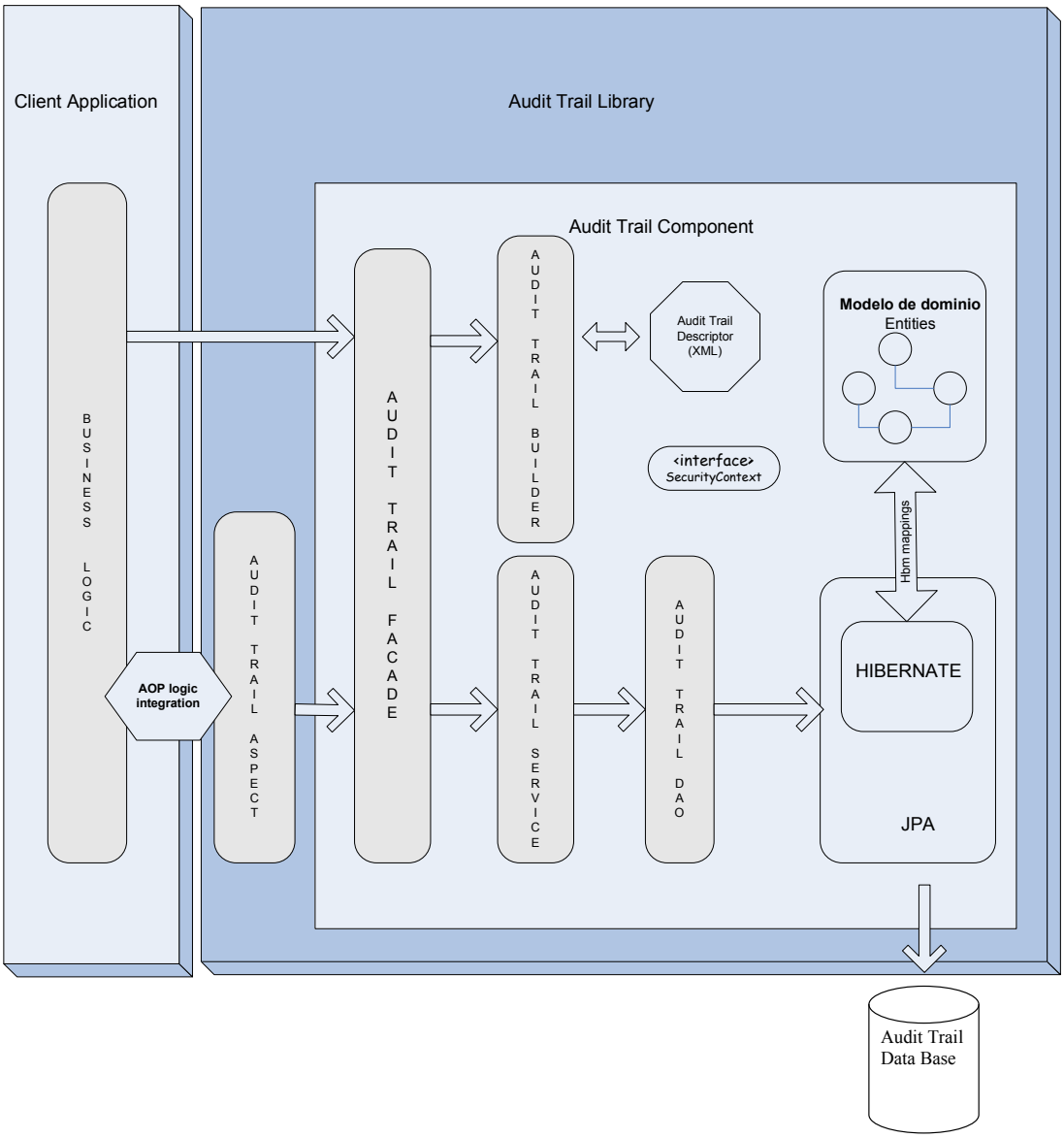
Representa una modificación a un campo simple de una entidad, por ejemplo, un campo de tipo String o Boleano; y también representa una modificación a un campo anidado que contiene otra entidad. Se describe cual es el valor anterior y el valor nuevo.

EntityCollectionFieldChange:

Representa una modificación a un campo de tipo colección de una entidad. La misma tiene una colección con los valores agregados (**CollectionItemAdded**) y otra colección con los valores borrados (**CollectionItemRemoved**).

Arquitectura del componente AuditTrail

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



AuditTrailFacade:

Provee una interfaz unificada y simple para utilizar el componente. Recibe los requerimientos en primera instancia. Es una implementación del patrón estructural Facade [19].

AuditTrailService:

Encargado de realizar la lógica de negocios del AuditTrail. Implementación del patrón empresarial J2EE Service Layer [20].

AuditTrailBuilder:

Encargado de inicializar el contexto del AuditTrail a través del audit-trail.xml. Implementación del patrón Builder [19].

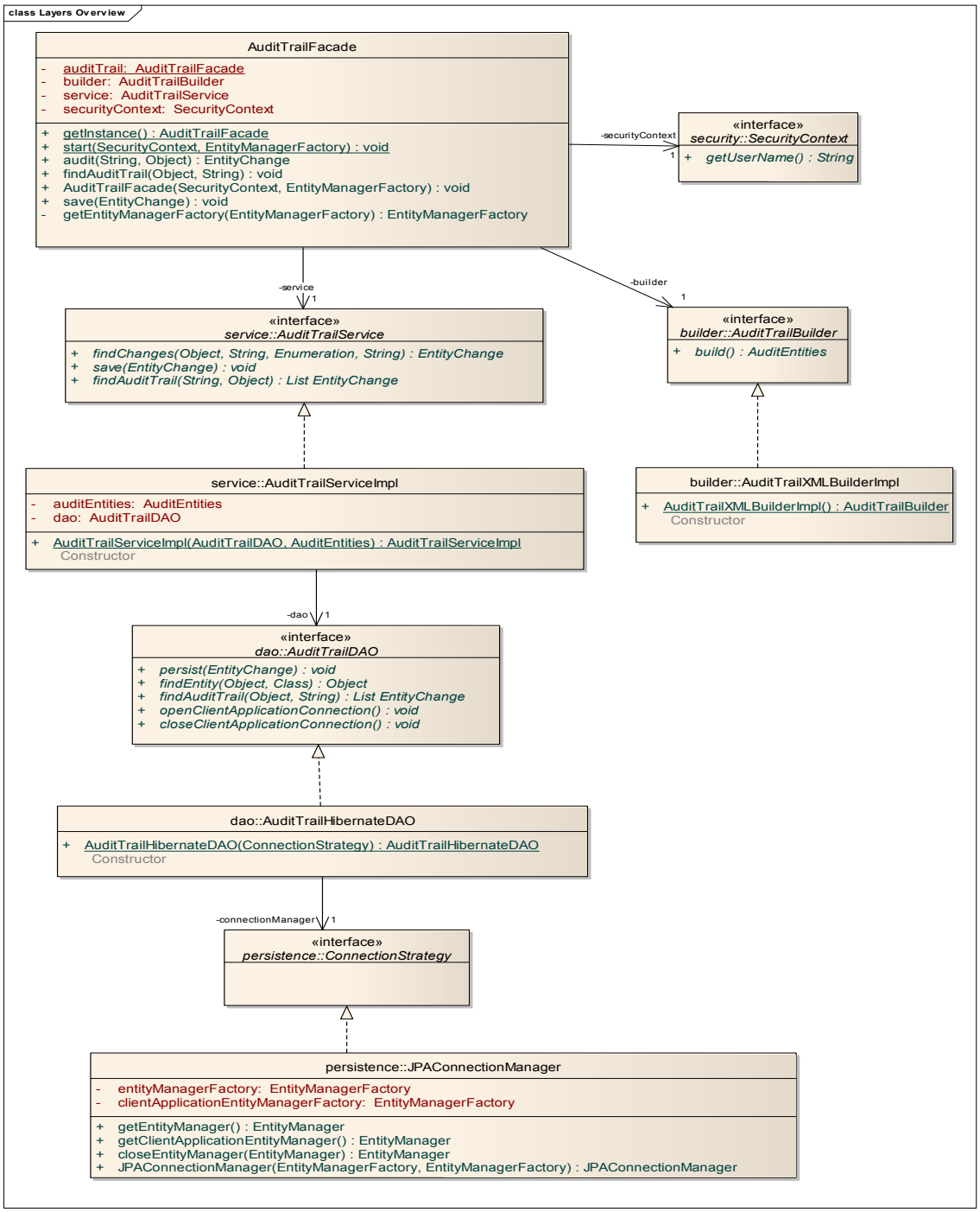
AuditTrailDAO:

Encargado de acceder a la base de datos del AuditTrail. Implementación del patrón Core J2EE DAO (Data Access Object) Layer.

SecurityContext:

Interfaz que define los métodos para acceder a la seguridad de la aplicación cliente. Define el método `getUserName()` utilizado para acceder al usuario logeado de la aplicación. Posteriormente, se describirá más en detalle el rol de esta interfaz desarrollada en la librería.

Diagrama de clases: Overview de Layers



Contexto de seguridad

Durante la auditoría de la información, es necesario que el AuditTrail tenga **acceso al usuario logeado** para poder registrar quien fue el responsable de las modificaciones en los cambios realizados.

En las aplicaciones WEB la seguridad puede estar implementada de distintas formas, por lo tanto, el acceso al usuario logeado varía según la solución adoptada. Por ejemplo, si la aplicación no utiliza ningún Framework de seguridad, normalmente el usuario logeado se guarda en el objeto ***HTTPSession***. En cambio, si utilizamos un Framework como ***Spring Security***, el acceso varía totalmente.

La solución adoptada en mi tesina fue proveer una interfaz denominada ***SecurityContext*** que debe ser implementada por alguna clase de la aplicación WEB cliente para poder acceder al usuario logeado.

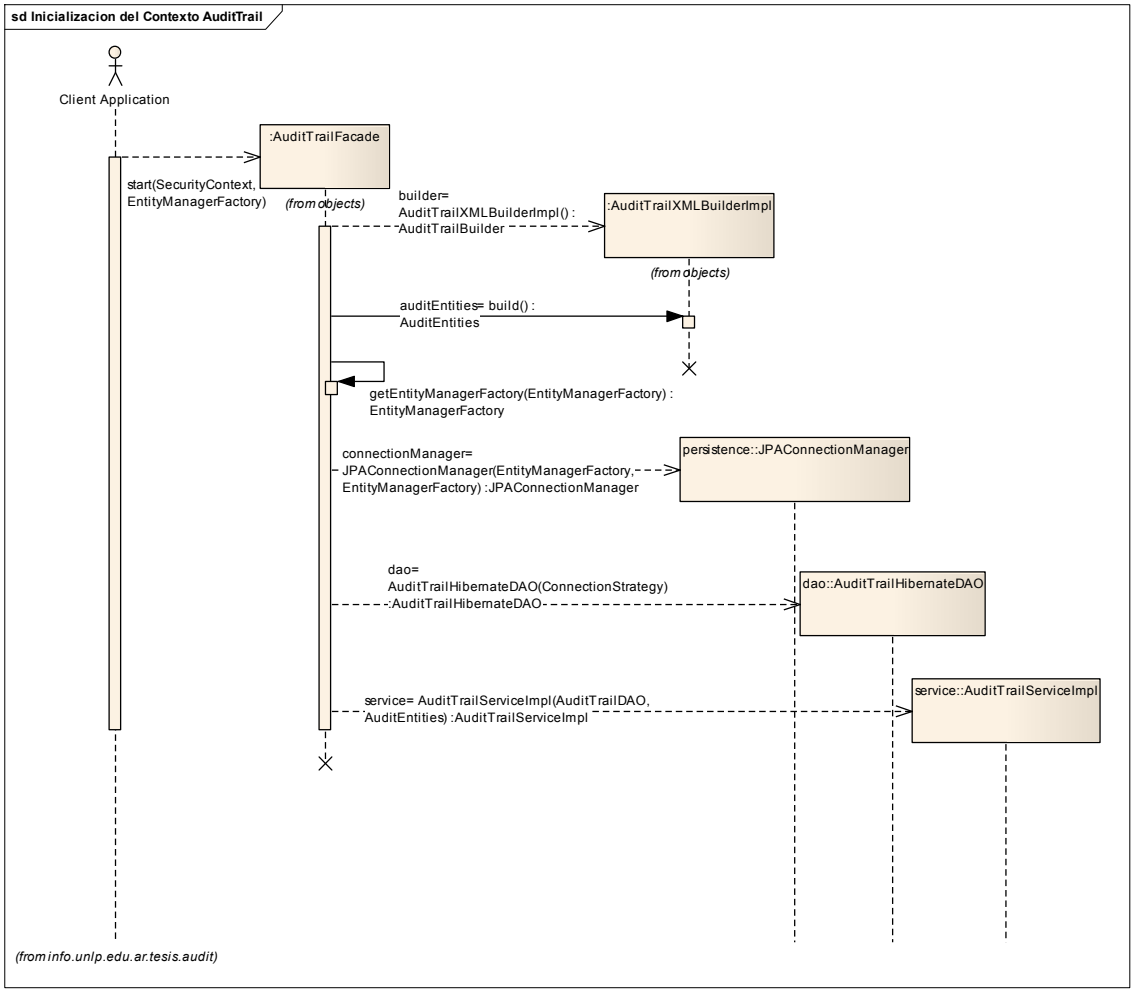
```
/**
 * Interfaz que debe implementar el contexto de  seguridad
 * de la aplicacion cliente que utiliza la libreria AuditTrail.
 * Su proposito es determinar la interfaz del metodo que retorne
 * el usuario logeado actual del sistema.
 *
 * @author jcorvi
 *
 */
public interface SecurityContext {
    /**
     * Retorna el nombre del usuario logeado actualmente en el sistema
     * @return Nombre de usuario
     */
    public String getUserName();
}
```

Diagramas de Interacción

Inicializar el contexto del AuditTrail

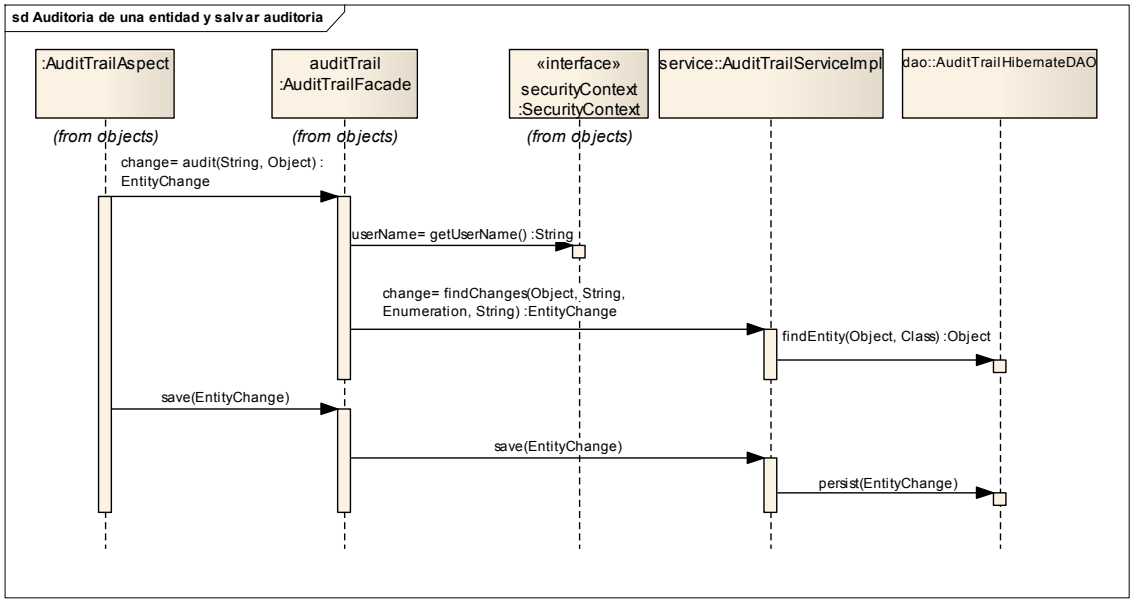
La aplicación cliente debe encargarse de iniciar el componente a través del método estático *start()*. En la inicialización, la aplicación cliente, debe enviarle dos parámetros importantes:

- El contexto de persistencia de **JPA** para acceder a la base de datos de la aplicación cliente (EntityManagerFactory); y
- El contexto de seguridad de la aplicación cliente. Es una clase que debe implementar la interfaz SecurityContext definida dentro de la librería del AuditTrail.



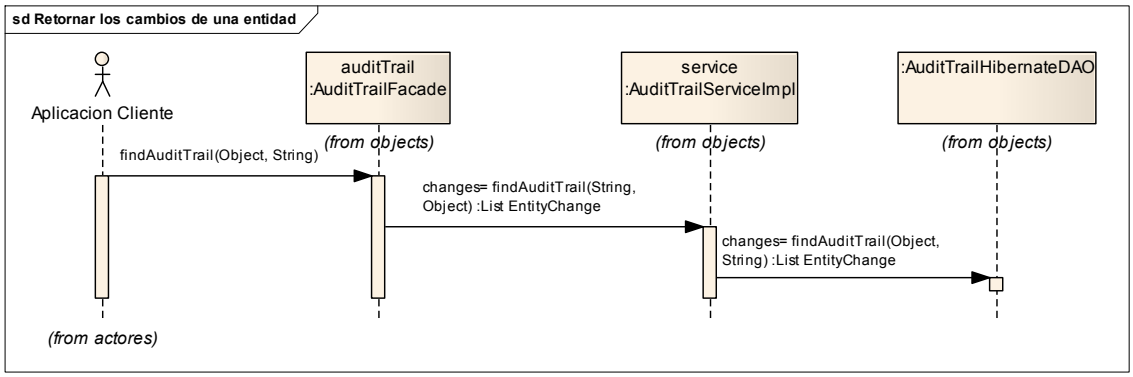
Auditoría de una entidad

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



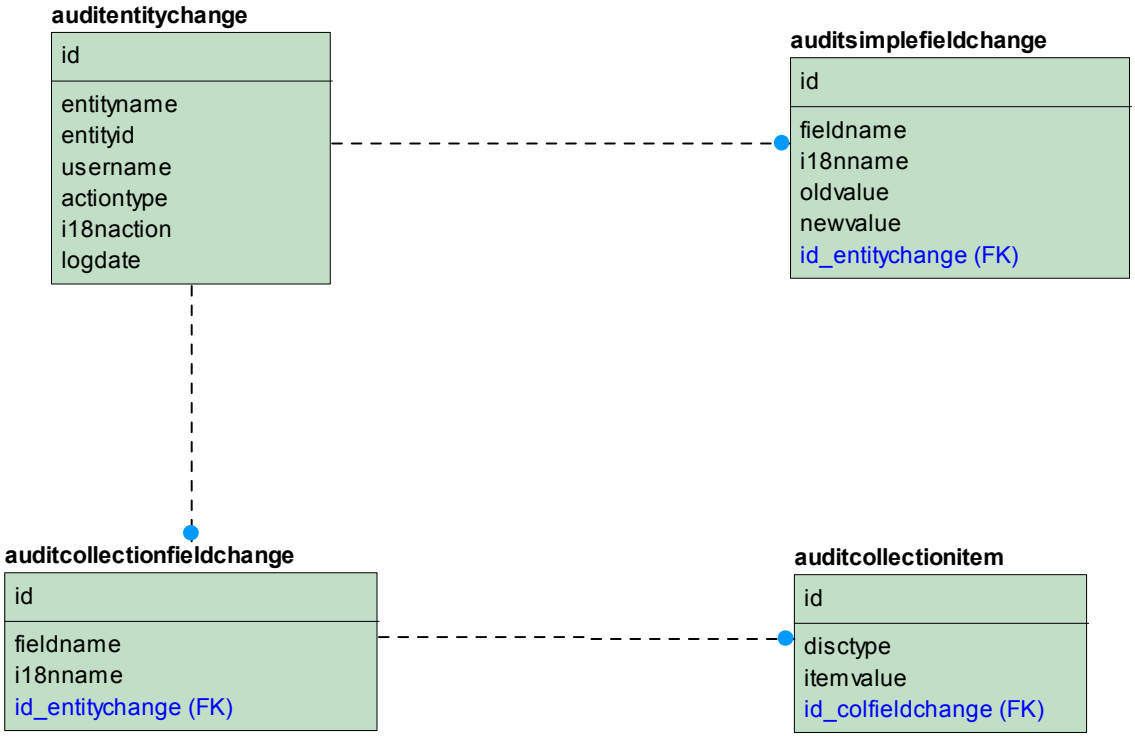
Retornar los cambios de una entidad

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.



Diseño de la base de datos

El diseño de la base de datos se encuentra alineado con el modelo de dominio que se describió en el diagrama de clases. Para diseñar los diagramas de bases de datos de la tesina se utilizo la herramienta de diseño **ER/Studio**.



Auditentitychange:

En esta tabla se registran las entidades que se modificaron, junto con los datos más importantes del monitoreo. Representa el mapeo relacional de la clase ***EntityChange*** del modelo de dominio.

Auditentitychange:

En esta tabla se registran los campos simples modificados, indicando el valor viejo y el valor nuevo. Representa el mapeo relacional de la clase ***EntitySimpleFieldChange*** del modelo de dominio.

Auditcollectionfieldchange:

En esta tabla se registran los campos de tipo colección modificados. Representa el mapeo relacional de la clase ***EntityCollectionFieldChange*** del modelo de dominio.

Auditcollectionitem:

En esta tabla se registran los ítems agregados y borrados de una modificación a un campo de tipo colección. Representa el mapeo relacional de las clases ***CollectionItemAdded*** y ***CollectionItemRemoved*** del modelo de dominio. El campo ***disctype*** indica si se trato de un ítem agregado (added) o un ítem removido (removed)

Capitulo 5: Aplicación de la librería AuditTrail

La librería AuditTrail en sistemas actuales

La librería AuditTrail actualmente se encuentra implementada en dos aplicaciones Web de gran envergadura. La primera es la aplicación orientada a la calidad, LOYAL, que fue nombrada en la introducción de la tesina. La misma posee dos módulos principales; DMS (Document Management System), en donde se almacenan los documentos de las empresas, en su mayoría procedimientos variados para el funcionamiento del negocio; y QMS (Quality Management System), en donde se cargan las auditorías (internas y externas), las No Conformidades y las Acciones Correctivas, entre otros formularios referidos al proceso de auditoría.

Además, la aplicación maneja usuarios y áreas; cada usuario pertenece a una o varias áreas dentro de la organización; a su vez los documentos están orientados a una o varias áreas. De esta relación, se provee la seguridad, ya que un usuario solo podrá visualizar los documentos pertenecientes a sus áreas.

La librería AuditTrail monitorea todas las modificaciones que ocurren en el modelo de objeto de la aplicación. El objetivo más importante que tiene la librería en esta aplicación es llevar el control y registrar los cambios que ocurren en los documentos, ya sean procedimientos, normas, informes de auditoría, no conformidades, acciones correctivas, etc. En el mismo nivel crítico se encuentra el monitoreo a la seguridad para la visualización de los documentos. Cambiar en forma incorrecta los alcances de un usuario podría llevar a que él mismo visualice información que no se encuentre autorizado a acceder.

La librería esta implementada en una versión de LOYAL que todavía no fue lanzada al mercado, su lanzamiento se prevé para el año que viene. La versión actual se encuentra implementada en 120 clientes aproximadamente, entre los más importantes se pueden destacar Arcor, Telecom, Edenor, Monsanto, Bayer y Bristol Mayers, entre otros.

La otra aplicación, en este caso ya productiva, se denomina Experto y tiene como objetivo aprobar o desaprobar préstamos a personas reales o jurídicas, para ello utiliza un conjunto de reglas que es definido por el cliente. Por cuestiones de confidencialidad no se puede divulgar el nombre de la empresa que tiene implementada esta aplicación.

En este caso el rol que cumple la librería es monitorear las modificaciones que ocurren sobre estas reglas, las cuales definen si una persona es o no solvente para recibir un préstamo de determinada cantidad de dinero. Modificaciones incorrectas sobre esta información crítica podría llevar a la emisión de préstamos en forma incorrecta, por lo tanto el control de cambios de estas reglas es un requerimiento crítico para el negocio.

En mi tesina, para mostrar el funcionamiento de la librería, se utilizó una aplicación WEB denominada StaffingProject que fue desarrollada para la materia Taller de Multimedia. A continuación se describe la configuración y el uso de la librería sobre este proyecto.

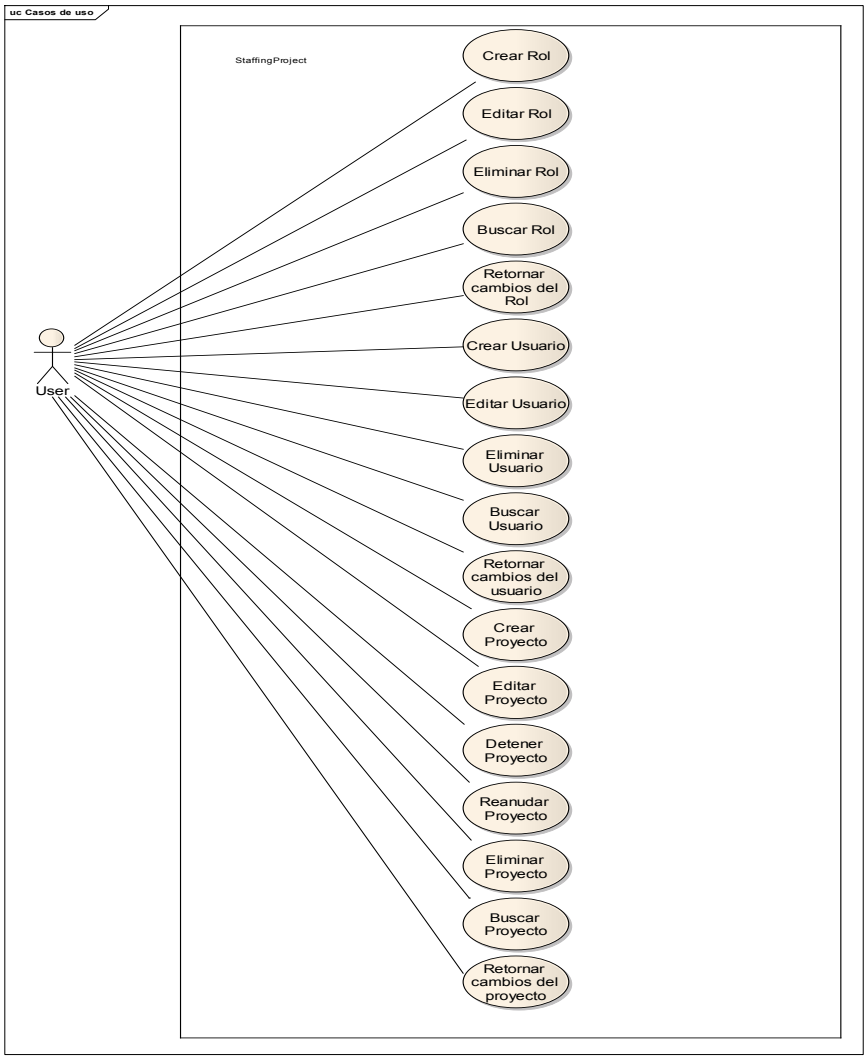
Proyecto StaffingProject

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.

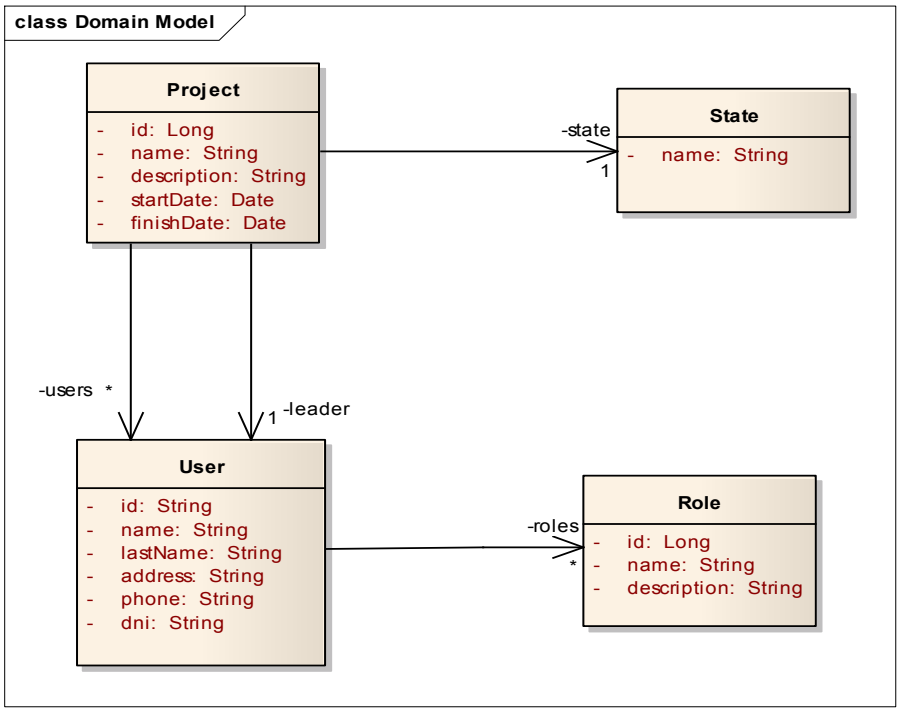
Requerimientos funcionales

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.

Casos de Uso

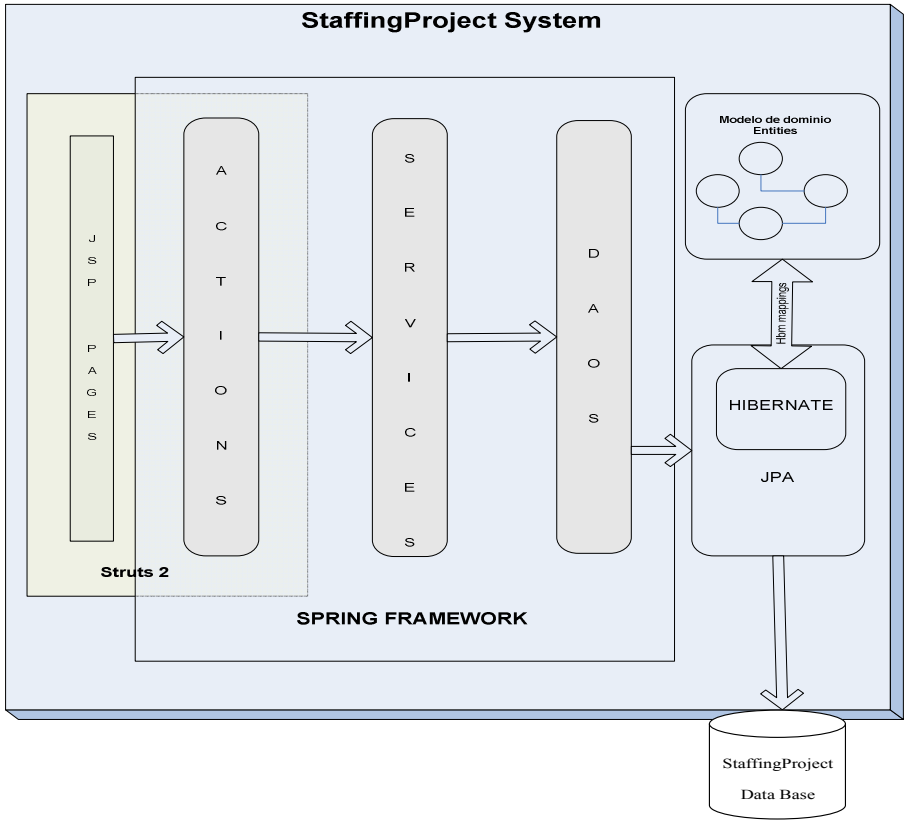


Modelo de dominio



Diseño

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



Frameworks y tecnologías utilizadas:

- Hibernate
- Struts 2
- Spring
- Spring Security
- Ajax

Interacción entre las capas:

Cada capa cumple un rol y tiene diferentes responsabilidades dentro del sistema; la comunicación entre ellas se realiza en forma ordenada y sin saltos.

Los **Actions** se encargan de recibir los requerimientos Web que llegan desde las páginas **jsp** de la aplicación, todo esto monitoreado y controlado por **Struts 2**. Desde el Action se invoca a los **Servicios** para que lleven a cabo la lógica correspondiente y; en caso de ser un requerimiento que requiera acceder a los datos, se invoca a los **DAOS** para realizar esta tarea. Con la capa de DAOS se deja definido desde donde se accede a la base de datos.

Spring se utiliza para terminar de organizar toda la arquitectura. Los Actions, Servicios y DAOS se agregan como beans de Spring y, a través del Contenedor IoC de Spring se realiza el ensamblado de los objetos de las distintas capas.

Diagrama de interacción: creación de Rol

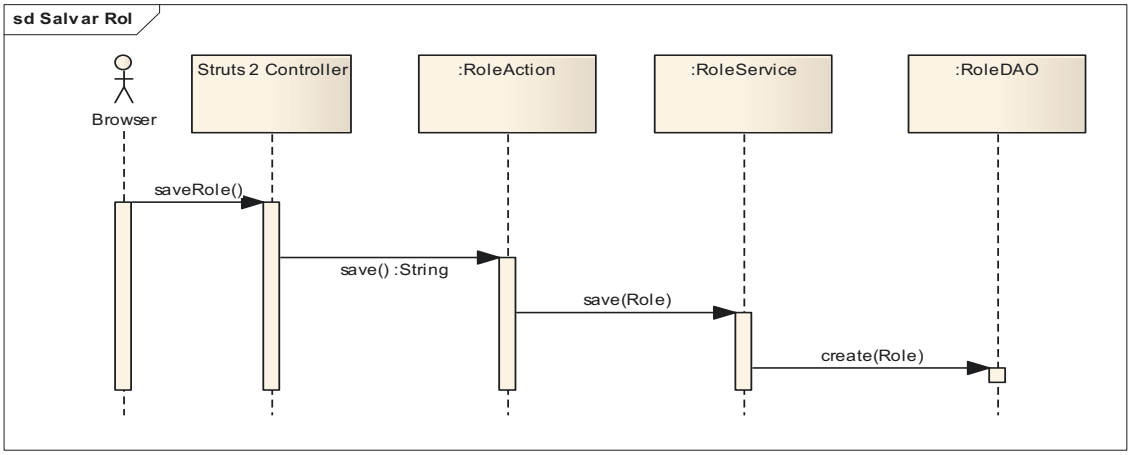
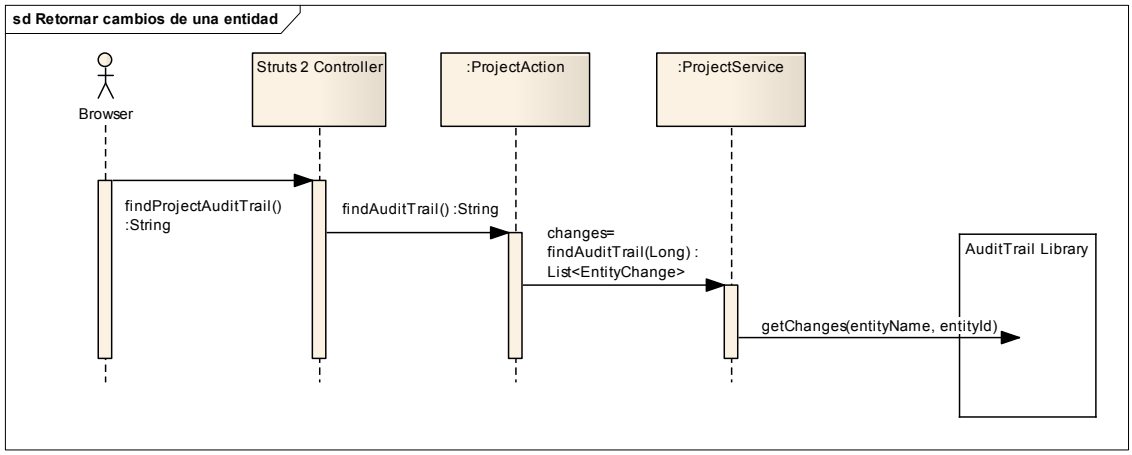


Diagrama de interacción: Retornar todos los cambios realizados a un proyecto

El requerimiento viene delegándose por las diferentes capas hasta que llega al servicio de proyecto, donde se invoca a la librería para retornar los cambios (Ver caso de uso del AuditTrail "*Retornar los cambios de una entidad*")

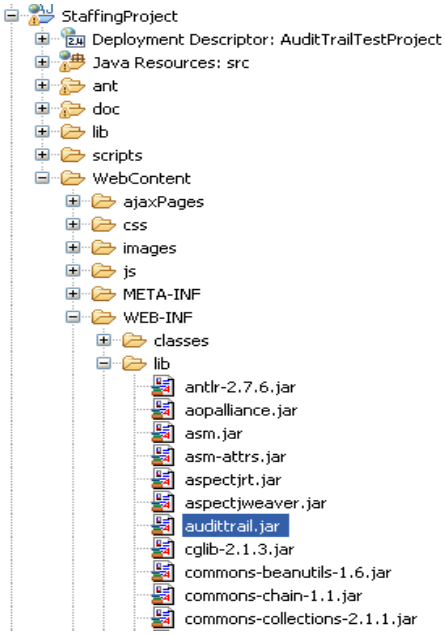

sd Retornar cambios de una entidad
Browser
Struts 2 Controller
:ProjectAction
:ProjectService
AuditTrail Library
findProjectAuditTrail()
:String
findAuditTrail() :String
changes=
findAuditTrail(Long) :
List<EntityChange>
getChanges(entityName, entityId)

Capitulo 6: Instalación y Configuración de la librería AuditTrail en el proyecto StaffingProject

Instalación

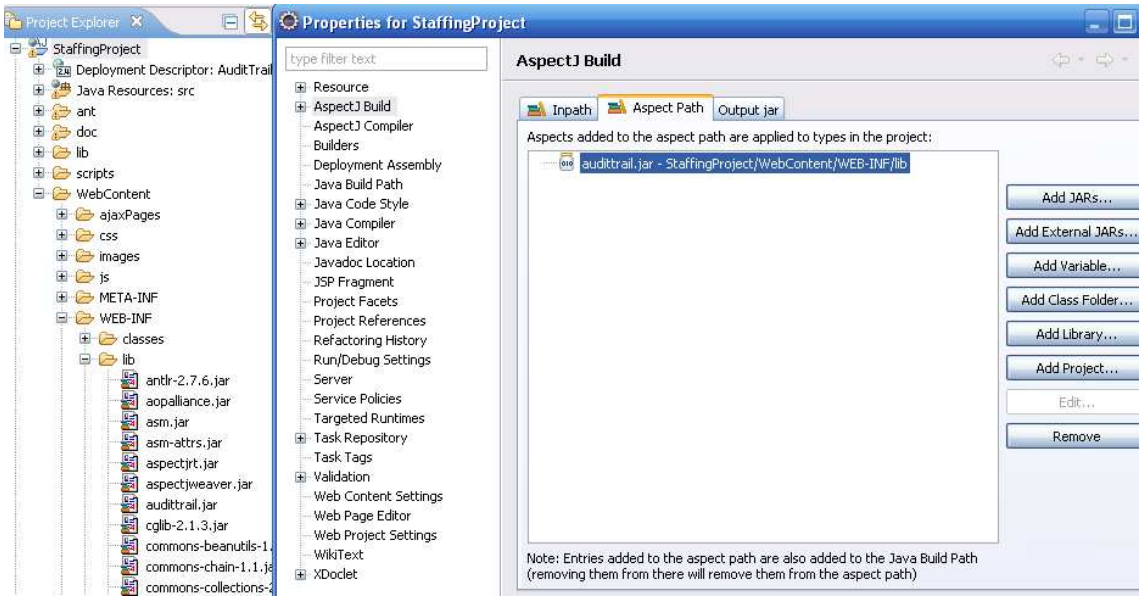
Paso 1

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**.



Paso 3

Creación de la base de datos AuditTrail. Archivo AuditTrail.sql.

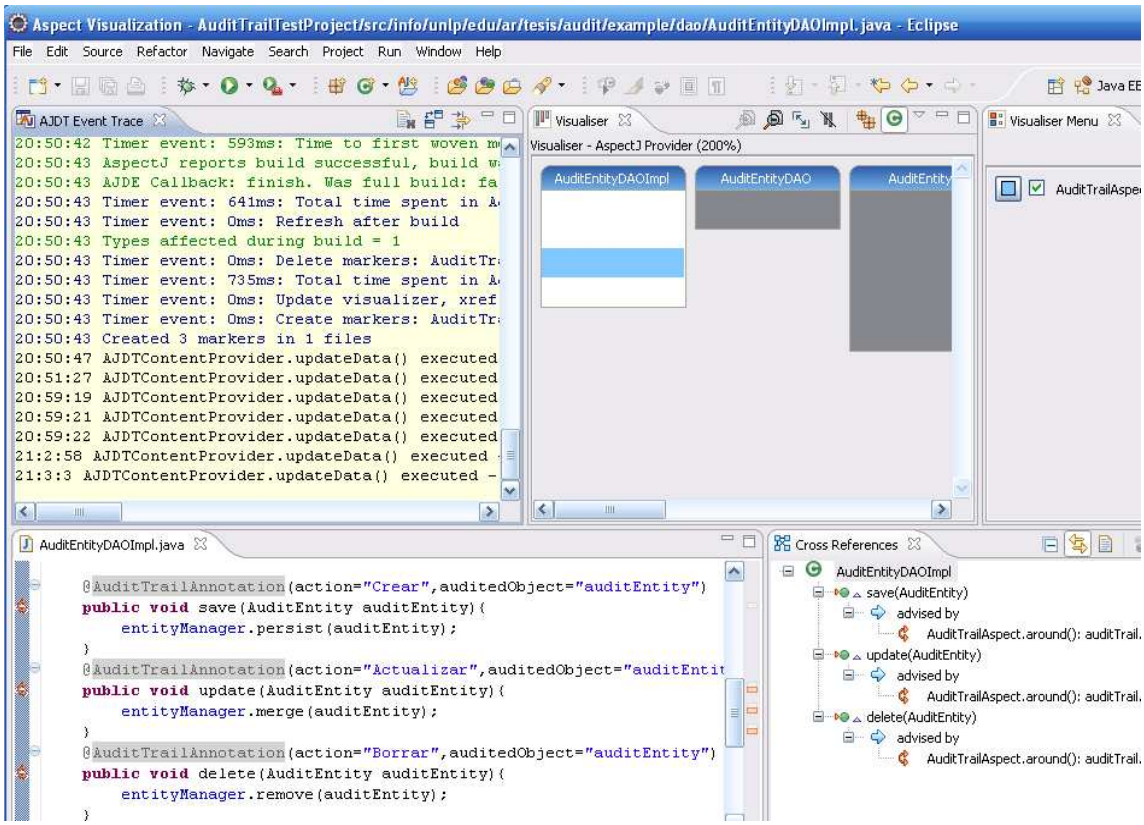
Integración con el ambiente de desarrollo Eclipse: AJDT

AspectJ Development Tools (AJDT) [21] provee al ambiente Eclipse las herramientas básicas necesarias para AOP con AspectJ. El objetivo de estas herramientas es brindarle al usuario un ambiente propicio para realizar desarrollos utilizando AOP.

AJDT fue el ambiente que elegí para desarrollar Aspectos en mi tesina; y pienso que es el más completo y fácil de utilizar. Alternativamente, el programador podría utilizar las herramientas básicas para poder compilar la librería AuditTrail junto con su proyecto, para lo cual tendría que instalarse en forma separada el compilador ajc y realizar la integración a través de líneas de comando. Trabajar de esta forma es similar a realizar la compilación del proyecto por líneas con el comando *javac*, sin utilizar ningún ambiente de desarrollo. Por lo tanto, si bien tenemos esa posibilidad no es muy viable trabajar de esta manera. Los comandos para ajc se encuentran descriptos en la bibliografía que se utilizó sobre Aspectos.

Además si bien la versión original de ajc fue desarrollada por Xerox Parc, ahora se encuentra desarrollada por el Proyecto Eclipse.

Conjunto de herramientas provistas por AJDT:



AJDT Event Trace:

Consola que muestra los eventos del compilador de aspectos.

Visualiser:

Descripción visual de las clases que están siendo alcanzadas por algún aspecto.

Visualiser Menu:

Aspectos que están alcanzando clases del proyecto.

Cross References:

Descripción visual de los join point, junto con los advices.

Configuración

Conexiones a las bases de datos

Cuando utilizamos **JPA** es necesario declarar la unidad de persistencia (***Persistente Unit***) en un archivo denominado **persistente.xml** ubicado en la carpeta **META-INF** de la aplicación. El Persistente Unit declarado contiene la información necesaria para conectarse a la base de datos.

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
    version="1.0">
    <!--  Persistence Unit de la aplicacion WEB cliente-->
    <persistence-unit name="punit">
         <properties>
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
            <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver" />
            <property name="hibernate.connection.url" value="jdbc:mysql://localhost/staffing" />
            <property name="hibernate.connection.username" value="root" />
            <property name="hibernate.connection.password" value="password" />
            <!-- mapeos del componente AuditTrail -->
            <property name="hibernate.ejb.cfgfile" value="/audit-trail-hibernate.cfg.xml"/>
         </properties>
    </persistence-unit>
</persistence>
```

La última propiedad del Persistente Unit agrega los mapeos del modelo del AuditTrail para poder persistir la información de la auditoría.

Alternativamente, si quisiéramos persistir los cambios del AuditTrail en otra base de datos distinta a la de la aplicación cliente, debemos crear un Persistente Unit denominado "auditpersistence" con la información correspondiente de la base para el AuditTrail.

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
    version="1.0">
    <!--  Persistence Unit de la aplicacion WEB cliente-->
    <persistence-unit name="punit">
        .........
    </persistence-unit>
    <!--  Persistence Unit propio del AuditTrail -->
    <persistence-unit name="auditpersistence">
        <properties>
             <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
             <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver" />
            <property name="hibernate.connection.url" value="jdbc:mysql://localhost/audittrail" />
            <property name="hibernate.connection.username" value="root" />
            <property name="hibernate.connection.password" value="password" />
            <property name="hibernate.ejb.cfgfile" value="/audit-trail-hibernate.cfg.xml"/>
        </properties>
    </persistence-unit>
</persistence>
```

Inicialización de la librería

Cuando se levanta el contexto de la aplicación, el AuditTrail debe ser inicializado. Anteriormente se ha descripto que la aplicación cliente debe invocar a la inicialización del AuditTrail enviándole los parámetros correspondientes. En mi tesina se utilizó Spring como Framework de aplicación para el proyecto WEB StaffingProject, por lo tanto, la librería incluye un adaptador para Spring denominado ***AuditTrailSpringAdapterBean***. Este adaptador no es más que una clase declarada como ***bean de Spring*** que se encarga de inicializar el contexto, invocando el método *start*() de la clase AuditTrailFacade.

```
<bean id="audittrail"
     class="info.unlp.edu.ar.tesis.audit.spring.AuditTrailSpringAdapter
     ........
</bean>
```

Por supuesto, si la aplicación cliente de la librería no utiliza Spring puede inicializar el contexto del AuditTrail desde cualquier clase. Podría definirse, por ejemplo, un filtro para realizar esta inicialización.

Contexto de persistencia de la aplicación cliente

El componente AuditTrail necesita conocer el contexto de persistencia de **JPA** para acceder a la base de datos de la aplicación cliente, el mismo comúnmente lleva el nombre de ***EntityManagerFactory***. En nuestro ejemplo WEB, esta configuración se lleva a cabo inyectándole al adaptador ***AuditTrailSpringAdapterBean,*** a través de IOC (Inversión de Control), la referencia del entityManagerFactory.

```
<bean id="audittrail"
    class="info.unlp.edu.ar.tesis.audit.spring.AuditTrailSpringAdapterBean" init-method="initialize">
    <property name="entityManagerFactory" ref="entityManagerFactory" />
    ......
</bean>
```

Contexto de seguridad de la aplicación cliente

La clase del proyecto StaffingProject que implementa la interfaz es:

```
/**
 * Servicio que contiene información básica de seguridad del sistema
 * @author javi
 *
 */
public class SecurityContextImpl implements SecurityContext{
        /**
         * Retorna el nombre de usuario logeado
         */
        public String getUserName(){
                User currentUser = (User)
        SecurityContextHolder.getContext().getAuthentication().getPrincipal();
                return currentUser.getUsername();
        }

}
```

Al igual que con la referencia al entityManagerFactory, se inyecta a través de IOC el bean que implementa la interfaz SecurityContext.

```
<bean id="securityContext" class="info.unlp.edu.ar.tesis.audit.example.security.SecurityContextImpl">
</bean>
.............
<bean id="audittrail"
    class="info.unlp.edu.ar.tesis.audit.spring.AuditTrailSpringAdapterBean" init-method="initialize">
    <property name="entityManagerFactory" ref="entityManagerFactory" />
    <property name="securityContext" ref="securityContext" />
</bean>
```

De igual manera en la que se indicó en la inicialización, si la aplicación cliente no se encuentra implementada con Spring, cuando se inicialice invocando al método start() se le deberán enviar los parámetros correspondientes; el contexto de persistencia y el contexto de seguridad de la aplicación cliente.

AuditTrailFacade.start(entityManager,sercurityContext);

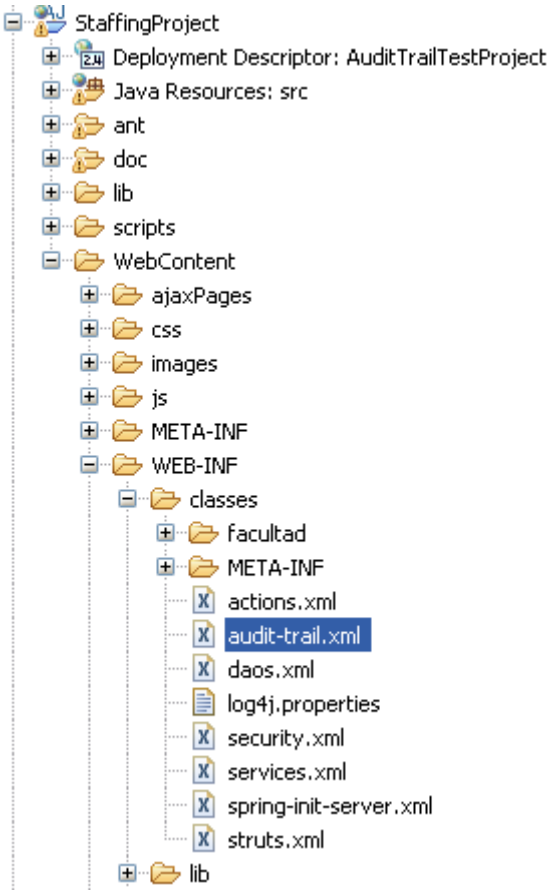
Creación del archivo descriptor audit-trail.xml

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

```
<?xml version="1.0" encoding="UTF-8"?>
<entities>
    <!-- declaracion de la entidad a auditar, se agrega el nombre de la clase -->
    <entity name="facultad.multimedia.staffing.model.Role">
        <identifier>id</identifier> <!-- indica el nombre del identificador -->
        <fields> <!-- coleccion de campos a auditar -->
            <field> <!--campo a auditar -->
                <!--nombre del atributo -->
                <name>name</name>
                <!--i18n del atributo (para visualizacion) -->
                <i18nname>comm.i18n.role.name</i18nname>
            </field>
            <field>
                <name>description</name>
                <i18nname>comm.i18n.role.description</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>
            <field>
                <name>lastName</name>
                <i18nname>comm.i18n.user.lastName</i18nname>
            </field>
            <field>
                <name>address</name>
                <i18nname>comm.i18n.user.address</i18nname>
            </field>
            <field>
                <name>phone</name>
                <i18nname>comm.i18n.user.phone</i18nname>
            </field>
            <field>
                <name>dni</name>
                <i18nname>comm.i18n.user.dni</i18nname>
            </field>
          <field>
              <name>roles</name>
              <i18nname>comm.i18n.user.roles</i18nname>
              <!--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>
      </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>description</name>
            <i18nname>comm.i18n.project.description</i18nname>
        </field>
        <field>
            <name>startDate</name>
            <i18nname>comm.i18n.project.startDate</i18nname>
        </field>
        <field>
            <name>finishDate</name>
            <i18nname>comm.i18n.project.finishDate</i18nname>
        </field>
        <field>
            <name>leader</name>
            <i18nname>comm.i18n.project.leader</i18nname>
            <!--si el campo es una entidad anidada -->
            <isEntity identifier="id" fieldValue="lastName"/>
            <!--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>
        <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>
        <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);
}
/**
 * Borrado de un objeto
 * @param object
 * @throws DeleteConstrainException
 */
@Transactional(propagation=Propagation.REQUIRES_NEW,readOnly=false)
@AuditTrailAnnotation(auditedObject="object",actionType=ActionType.DELETE,
action="com.action.delete")
public void delete(ModelEntity object)  {
        object = (ModelEntity)this.find(object.getId());
        entityManager.remove(object);
}
```

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

Capitulo 7: Testing de la librería AuditTrail y el Proyecto StaffingProject

Instalación del proyecto StaffingProject

Se provee el archivo StaffingProject.war para realizar el deploy del proyecto; y los scripts de base de datos: Staffing.sql para crear la base de datos. Además, se realizó un backup de la base con datos ya creados para poder importarla.

Toda la información, paso a paso, para poder instalar el proyecto StaffingProject y comenzar a probar la librería, se describe en el archivo Instalacion.txt

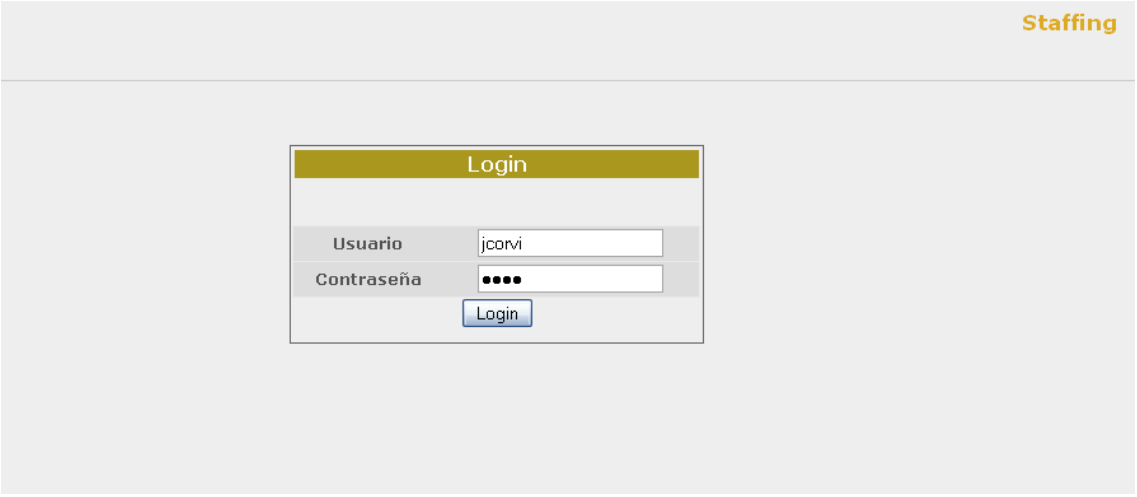
Análisis del AuditTrail en el proyecto StaffingProject

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

En primer lugar cuando ingresamos al sistema StaffingProject nos lleva al login de la aplicación. No se realizó la funcionalidad de login contra la base de datos, ya que no es un requerimiento primordial para probar la librería. En cambio, los usuarios que pueden logearse están definidos en el archivo security.xml utilizado por Spring Security.

Usuarios disponibles

```
<security:user-service>
  <security:user name="jcorvi" password="pass" authorities="ROLE_USER, ROLE_ADMIN" />
  <security:user name="jdiaz" password="pass" authorities="ROLE_USER, ROLE_ADMIN" />
  <security:user name="cqueiruga" password="pass" authorities="ROLE_USER, ROLE_ADMIN" />
</security:user-service>
```

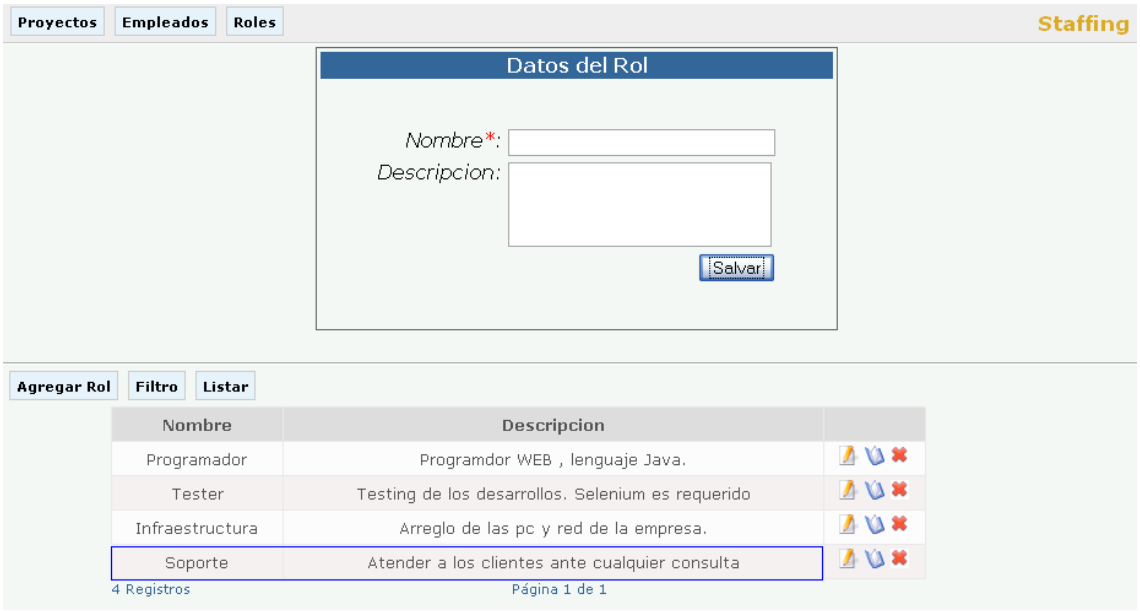


Una vez logeado el usuario, el sistema lo redirección al home del sitio donde visualiza los proyectos que se encuentran cargados.

Creación de una entidad

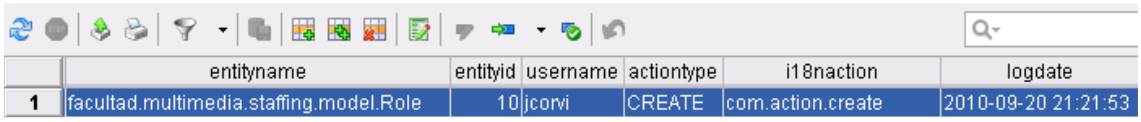
Comencemos analizando la creación del rol “Soporte”.

Luego de crear el nuevo rol, se recarga la lista



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

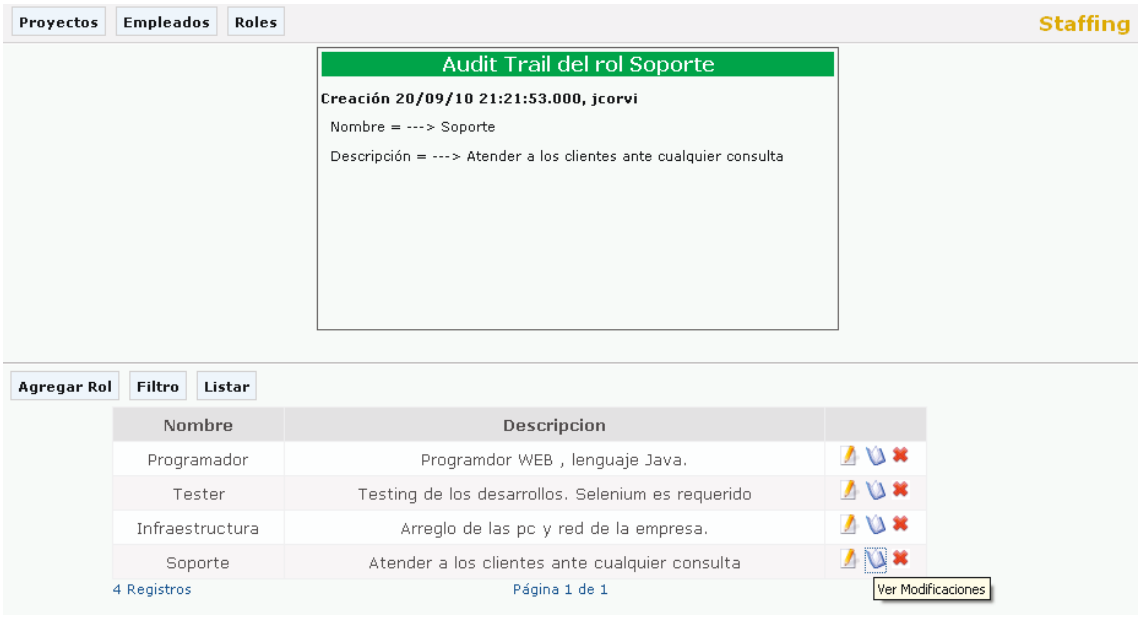
El campo que indica la clase auditada, *entityname*, junto con el campo que indica el identificador, *entityid*, describen a la entidad auditada; en este caso la entidad es un Rol con el id=10.

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.

Uno de los requerimientos del sistema StaffingProject es ver los cambios realizados a la entidad, por lo tanto esta información podemos verla a través de la interfaz del aplicación.



Edición de una entidad

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

Actualmente el proyecto tiene valores que deben ser modificados.

Datos que se van a modificar:

- Nombre: “Instalación del Core”.
- Fecha de Fin: “10/10/2010”.
- Líder: Martín.
- Participantes: se agregan al proyecto Javier y Martín.

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***

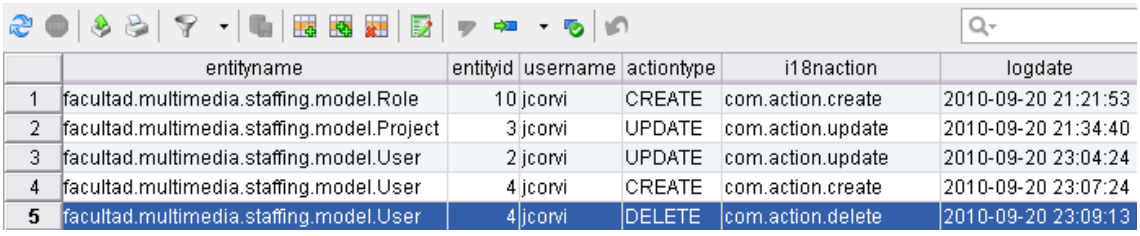
	id	disctype	itemvalue	id_colfieldchange
1	1	added	Corvi	1
2	2	added	Ferra	1

Borrado de una entidad

Borrado del usuario Marcos Sierra.

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

Capitulo 8: Continuidad y Conclusión de la tesina

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 seria 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.

Al haber implementado la librería en sistemas productivos de gran envergadura, sobre todo en el sistema de calidad LOYAL, van a surgir requerimientos de auditorías. Personalmente creo que el proyecto AuditTrail va a tener varias etapas evolutivas con diferentes requerimientos.

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 y los sistemas productivos en mi ambiente laboral. 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.

Otros objetivos secundarios, pero no menos importantes, también pudieron ser implementados; es el caso de la separación de la persistencia de la información de auditoría, generada por la librería, con la información del sistema auditado en sí. Para ello, la librería brinda una configuración en donde puede indicarse la base de datos donde quedará registrada la información de auditoría. Este objetivo tenía como fin mantener distintos niveles de seguridad sobre la información de la compañía. Gracias a esta alternativa, se podría, por ejemplo, delegar la administración de las bases de datos a diferentes usuarios de la compañía, descentralizando la seguridad de la información.

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.

Como se ha comentado, la librería se encuentra implementada en proyectos del orden laboral, por lo tanto, y como fue señalado en los próximos trabajos futuros, se agregarán los requerimientos de auditoría que sean solicitados, para lograr los objetivos de control dispuestos.

Profesionalmente el desarrollo de mi tesina me sirvió para implementar soluciones de software que todavía no son muy conocidas y utilizadas en la industria.

Referencias bibliográficas

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

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

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

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

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

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

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

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

[9] Artículo del diario el País de España: http://www.elpais.com/articulo/sociedad/Mattel/retira/mercado/espanol/miles/juguetes/pintura/plomo/imanes/mal/fijados/elpepusoc/20070814elpepusoc_3/Tes

[10] Articula del diario Tiempo Sur de Río Gallegos: http://www.tiemposur.com.ar/nota/10912-retiran-del-mercado-33-lotes-alterados-de-ibupirac-.html

[11] ANMAT. Sitio oficial: http://www.anmat.gov.ar/

[12] Enterprise Architect. Sitio oficial: http://www.sparxsystems.com.au/

[13] Documentación oficial de Anotaciones: http://download.oracle.com/docs/cd/E17476_01/javase/1.5.0/docs/guide/language/annotations.html

[14] Brown, D.; Davis, C.M.; Stanlick, S. (2008). Struts 2 In Action. Greenwich: Manning.

[15] Struts 2. Sitio oficial: http://struts.apache.org/2.x/

[16] MySql. Sitio oficial: http://www.mysql.com/

[17] Apache Tomcat. Sitio oficial: http://tomcat.apache.org/

[18] Alur, D.; Crupi, J.; Malks, D. (2003) Core J2EE Patterns: Best Practices and Design Strategies**.** . Second Edition. Prentice Hall / Sun Microsystems Press.

[19] Gamma, E. (1995) Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley.

[20] Fowler, M. (2003) Patterns of Enterprise Application Architecture. Addison Wesley.

[21] AJDT. Sitio oficial: http://www.eclipse.org/ajdt/