coremain

Solución	Outsourcing	Fecha:	03/01/2012	Fichero:	Arquitectura para desarrollo ágil de aplicaciones J2EE.odt
Documento:	**Especificación de la arquitectura del esqueleto de aplicaciónes J2EE orientada al desarrollo ágil.**			Código	
Elaborado por:	Marcos Casal Santos			Página:	1 de 18

Especificación de Arquitectura

Especificación de la arquitectura de esqueleto de aplicaciones J2EE orientada al desarrollo ágil

Descripción: ***Especificación de arquitectura y tecnologías utilizadas en el esqueleto para el desarrollo ágil de aplicaciones J2EE***

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

ÍNDICE

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

1. Introducción

1.1. Visión general y objetivos

Este documento tiene como objetivo describir una arquitectura una servirá de esqueleto base para el desarrollo de aplicaciones J2EE robustas y sencillas que permita agilizar el desarrollo y facilitar el mantenimiento. Además se incluyen un conjunto de buenas prácticas así como una propuesta de metodología orientados a los lograr un desarrollo ágil manteniendo la calidad del producto.

Esta arquitectura prima la simplicidad frente a crear un marco de desarrollo muy elaborado en el que la complejidad del mismo dificulta el mantenimiento y la modificación de las tecnologías empleadas. Se trata por tanto de una arquitectura compatible con la metodologías ágiles de desarrollo.

La Arquitectura está organizada en capas basado en el estándar MVC (Modelo-Vista-Controlador)

Fomenta la flexibilidad manteniendo las capas de la arquitectura lo más desacopladas posible, favoreciendo la utilización de otras tecnologías y/o la actualización de las mismas con el mínimo impacto.

Hace uso de patrones de diseño estándar: Bussiness Delegate, Service Locator, DTO, DAO, Filter, Decorator...etc.

Las tecnologías utilizadas son tecnologías estándar de fábrica y de facto, ampliamente extendidos y aceptados por el mercado, con una completa documentación accesible y en la que la mayoría de compañeros de Coremain que desarrollamos en Java poseemos experiencia.

Se trata de una arquitectura orientada a servicios (SOA), buscando la re-utilización de los servicios en otras aplicaciones.

Los servicios se implementarán como proyectos independientes, manteniendo la máxima cohesión funcional posible entre los métodos de un mismo servicio con el fin de facilitar el mantenimiento y re-utilización de los mismos.

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

Aunque la arquitectura en capas se basa en el estándar MVC, una forma más adecuada de ver las capas que componen la aplicación sería la división en las capas: Presentación, Servicios de negocio y Persistencia. (ver Ilustración 1)

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

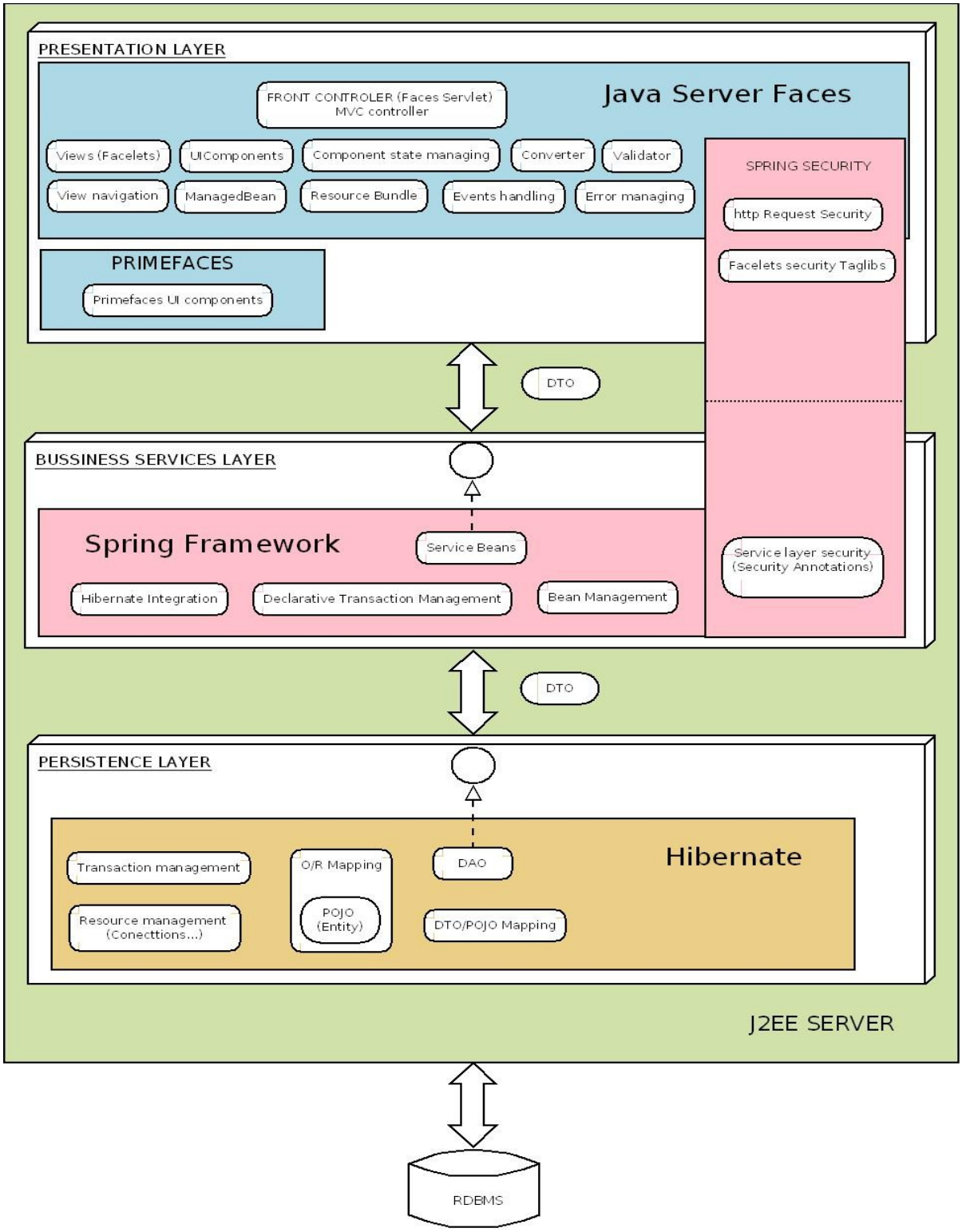

Ilustración 1: Arquitectura y tecnologías empleadas

2. Capa de Presentación

La capa de presentación engloba las capas vista controlador del estándar MVC. Con el fin de proporcionar una arquitectura robusta y sencilla para agilizar el desarrollo y facilitar el mantenimiento se opto por utilizar las siguientes tecnologías:

2.1. JSF + Facelets

Java Server Faces es un estándar de J2EE que permite el desarrollo de la capa de presentación y control para aplicaciones Web.

JSF permite el desarrollo de aplicaciones Web en base a componentes y eventos. Aproximación típica para el desarrollo de aplicaciones interactivas en entornos de escritorio.

Facelets es el sistema de plantillas estándar de JSF 2. Con los componentes de JSF y Facelets es mucho más sencillo reutilizar piezas visuales, ya que estos se pueden "paquetizar" en forma de componentes por composición, que resultarán muy fáciles de usar en distintas partes de la aplicación, cómo si se tratara de un componente estándar, además de mejorar el rendimiento con respecto a las JSPs.

La idea de gestión de la interacción entre el usuario y la máquina en base a eventos, permite realizar interfaces de usuario mucho más ricas desde el punto de vista de la usabilidad, y simplifica el desarrollo de las mismas. Acercándose cada vez más las aplicaciones Web a las funcionalidades visuales típicamente ofrecidas por las aplicaciones de escritorio.

JSF se encuentra actualmente en su versión 2.x. Este nueva versión da soporte directamente a tecnología AJAX lo que mejora la experiencia del usuario usuario haciendo las páginas más dinámicas, evitando los parpadeos al recargar la página completa.

JSF permite el uso de anotaciones para la configuración con lo que se agiliza la configuración de managedBeans y demás componentes; validators, converters... etc

La gestión del flujo de navegación que permite JSF, sin ser tan potente como pueda ser Spring Web Flow, es más que suficiente para la mayoría de aplicaciones y mucho más sencilla, ágil de implementar, por lo que se optó por utilizar la navegación de JSF, aunque

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

debido a las interesantes características de Spring Web Flow sería interesante disponer de una versión que utilizase Spring Web Flow.

Los métodos de los managedBeans únicamente contendrán código que maneje los distintos componentes de la vista, delegando en la capa de servicios de negocio toda la lógica de negocio de la aplicación.

2.2. Primefaces

Primefaces es una librería de componentes para JSF. Estos componentes aportan, frente a los componentes estándar de JSF, una abstracción para el uso de la tecnología AJAX ya soportada en JSF 2. Es decir, el desarrollador puede centrarse en la funcionalidad ofrecida sin tener que preocuparse del JavaScript que se ejecutará en el cliente o de que partes de la pantalla serán necesarias refrescar en respuesta de un evento en la interfaz de usuario.

No siendo Primefaces parte del estándar JEE, ahora es la única librería de componentes visuales que podemos decir que soporta de manera estable la versión 2 de JSF.

El conjunto de componentes es muy amplio y tiene un buen soporte para desarrolladores

Otra ventaja de Primefaces, al apoyarse en el uso extensivo de Jquery es que se puede utilizar el "Jquery ThemeRoller" para crear el conjunto de CSS de la aplicación de modo sencillo, agilizando el diseño y el mantenimiento de las CSS.

2.3. Spring Security

Spring Security es un framework altamente configurable de gestión de accesos, basada en AOP (Aspect Oriented Programing) implementa seguridad transversalmente entre las capas de la aplicación de manera no intrusiva lo que mantiene simplifica el código al no tener que implementar programáticamente la gestión de accesos facilitando el mantenimiento del mismo.

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

Permite configurarse fácilmente para añadir seguridad a los accesos de la aplicación así como para su integración con distintos servicios de seguridad: SSO, LDAP.

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

3. Capa de servicios de negocio

La capa de servicios de negocio contendrá toda la lógica de negocio de la aplicación. Se utiliza una arquitectura basada en servicios (SOA). Se usan interfaces para definir la funcionalidad de los servicios.

Los servicios se implementarán como proyectos independientes, manteniendo la máxima cohesión funcional posible entre los métodos de un mismo servicio y evitando en lo posible el acoplamiento entre los mismos con el fin de facilitar el mantenimiento y re-utilización de los mismos.

Uso de DTOs (Data transfer Object) como unidades de información para comunicarse con el resto de capas de la aplicación (presentación y persistencia). Estos no contienen lógica de negocio por lo que pueden ser utilizados para la transferencia de información con sistemas externos.

Se usa la inversión de control de Spring y las anotaciones (Spring soporta los estándares de inyección de dependencias JSR-250 y JSR-330) para cablear los componentes de forma sencilla y con un bajo acoplamiento

Para la gestión de transacciones también se recomienda utilizar las configuración mediante transacciones de Spring por la agilidad que aporta al desarrollo.

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

4. Capa de persistencia

Esta capa será la encargada de proporcionar acceso a los datos a la capa de servicios de negocio así como de la persistencia de los datos.

Utiliza el patrón DAO (Data Acces Object) con la implementación correspondiente. En caso común de utilizar una Base de Datos utilizaremos la implementación de Hibernate para los DAOs.

Como la capa de servicios de negocio trabajará únicamente con objetos DTOs. El mapeo de los DTOs en POJOs y viceversa será responsabilidad de la capa de persistencia, permitiendo utilizar cualquier tecnología de "mapping" automatizado como Dozer o ModdelMapper.

Para agilizar el desarrollo del mapping entre atributos de objetos y columnas de tablas de Base de datos se usarán anotaciones en los POJOs frente a la configuración en archivos hbm.xml. Se utilizarán las anotaciones de JPA frente a las de Hibernate.

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

5. Maven

Se utilizará la herramienta Maven para la construcción del proyecto. Esta es una de las herramientas clave en la aportación de agilidad a la creación del proyecto por las siguientes características:

Maven es el estándar de facto actual para la construcción de proyectos. Integrable con el IDE eclipse. permite realizar todas las funcionalidades requeridas para la construcción pruebas unitarias, empaquetado y despliegue de un proyecto

Una de las características más importantes de Maven es la gestión de dependencias transitivas entre librerías lo que agiliza la localización, sustitución de estas y evita los errores derivados de los conflictos entre versiones difíciles de detectar de otro modo.

Actualmente hay disponibles gran cantidad de repositorios accesibles por Maven on-line de los que obtener librerías de terceros para el proyecto de forma automática.

Existe la posibilidad de integrar con gestores de repositorios como Nexus de Sonatype para centralizar el acceso a repositorios cacheando las librerías de terceros (artifacts) utilizadas en los proyectos de la organización así como los librerías propias de la misma en una ubicación local de la organización.

Siguiendo la arquitectura indicada en la que cada servicio se implementa como un proyecto independiente y haciendo uso de Maven y un sistema de gestión de Repositorios como Nexus se puede automatizar la tarea de hacer disponibles al resto de compañeros de la organización las versiones *Release* o incluso las *Snapshot* de los servicios implementados agilizando el proceso de versionado y facilitando la reutilización de los servicios.

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

6. Best practices

En este apartado se describen un conjunto de buenas practicas a seguir con el objetivo de agilizar el desarrollo y facilitar el mantenimiento utilizando la arquitectura propuesta.

6.1. Maven

Mantener el conjunto de dependencias definidos en los POMs del proyecto lo más pequeño posible, excluyendo del POM aquellas librerias que se resuelvan automáticamente como dependencias de otras ya definidas.

Usar la vista "Dependency Hierachy" del "Maven POM editor" del plugin m2elcipse del IDE eclipse para analizar estos casos.

6.2. Presentación

Evitar los *managedBeans* demasiados grandes, dividiendo en varios *managedBeans* si fuese necesario cuando crezca demasiado el número de atributos o métodos gestionados en el mismo.

Se usarán las anotaciones propias de JSF para anotar los *managedBean*, *scopes* ...etc De este modo simplificamos la configuración, dejando el archivo faces-config.xml para la configuración de los aspectos que no permiten anotaciones.

El inconveniente de usar las anotaciones de JSF frente a las estándar de Java o Spring es que no se podrá inyectar las clases de los servicios en los managedBeans mediante *IOC* , por lo que usaremos el médodo *getService()* de la clase FacesUtils para esto.

El *scope* de los *managedBeans* será el menos persistente posible con el fin de mantener el tamaño de la sesión lo más bajo posible.

Para la interacción con la mayoría de los componentes de Primefaces que hacen uso de Ajax el scope más adecuado suele ser ViewScope.

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

Los atributos de los *managedBeans* serán los propios DTOs siempre que sean posible. Componiendo los DTOs por otros DTOs si esto sigue la estructura real del objeto que modelan. Con esto se intenta mantener bajo el número de atributos de cada *managedBean*.

Los métodos de los *managedBeans* solo contendrán la lógica necesaria para la interacción con el usuario delegando en los servicios cualquier lógica de negocio.

Para los componentes de la vista que traten con colecciones de objetos se definirá un *converter* que obtendrá el objeto tratado de la colección cargada en el managedBean. (Ver ejemplo RolePickListConverter). Para los *validators* seguiremos una estrategia análoga de modo que utilizamos componentes propios de JSF (converter, validator, phaseListener...) en lugar de manipular los datos en los managedBeans.

6.3. DTOs

No incluir en los DTOs lógica de negocio más allá de funciones que agreguen atributos preparándolos para la vista. Ej getNombreCompleto() que devuelva la concatenación de nombre y apellidos.

Tener en cuenta que este patrón de diseño está pensado para que sean objetos para transferir datos por lo que no deben incluir lógica de negocio.

6.4. Gestión de excepciones

Únicamente se capturarán y lanzarán aquellas excepciones que sean gestionadas por el método llamante, las excepciones debidas e errores inesperados no se capturarán por defecto.

En caso de utilizar un método que defina en su firma el lanzamiento de una excepción específica (*Checked exception*), y esta no se contempla para su gestión se capturará y se convertirá en una RuntimeException.

La gestión de las posibles excepciones inesperadas de la aplicación se realizará a nivel de servlet de JSF evitando la propagación por defecto de excepciones entre las capas de la aplicación

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

De esta forma simplificamos el código al no introducir tryCatch y Throws por defecto mejorando además la eficiencia del mismo, siguiendo la filosofía de primar el desarrollo ágil.

6.5. Nomenclatura

Para agilizar la escritura los objetos DTO, POJO, DAO... etc, los nombraremos poniendo en mayúsculas únicamente la primera Letra del sufijo, ejemplo: userDto, userPojo, userDao.

Para los nombres de métodos y atributos primar el crear un nombre descriptivo del mismo aunque esto implique dar una longitud considerable al nombre. Las opciones de auto completar del IDE Eclipse facilita utilizar estos métodos y el hecho de reconocer la función de un método o atributo por su nombre lo compensa.

Usar una misma notación para las mismas acciones. Para las acciones propias del la programación usar como prefijo preferiblemente los verbos en infinitivo y en inglés:go, get, set add, insert, update, delete, execute. Esto mejora la comprensión del código y permite establecer patrones para los puntos de corte en la programación orientada a aspectos AOP.

En nuestro caso en casi inevitable la programación bilingüe en inglés y castellano, por lo que mejor no aumentar la confusión añadiendo el gallego.

Para las acciones y objetos propios de la programación es preferible usar el inglés y el verbo de uso común en la programación para nombrarlos: comparator, sort, populate, build, list, set, collection,

Para los objetos propios del negocio mejor no traducirlos al inglés en la codificación lo que facilitará entender los algoritmos de programación.

6.6. Documentación

La documentación del proyecto debería seguir los principios de la documentación ágil para no entorpecer el desarrollo ágil del mismo.

La documentación del código es un punto importante para el desarrollo ágil, por lo que deberían tenerse en cuenta las siguientes consideraciones:

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

Importante documentar mediante *javadoc* los métodos de las interfaces, especialmente aquellos que no sea obvio el proceso que realizan o las condiciones el la que devuelve un dato o una excepción específica.

Es muy importante describir en los métodos que devuelven una objeto si este puede ser devuelto como *null*. Por ejemplo, los métodos que devuelven colecciones deberían devolver una colección vacía en lugar de null para evitar *null Exceptions*, pero si se posibilita devolver este tipo de dato debería estar claramente indicado el el javadoc correspondiente al método.

Los criterios de arquitectura, diseño y buenas prácticas que no sean obvios en el esqueleto de la aplicación deberían estar comentados también en el código, en la parte correspondiente ,indicando las ventajas que supone su uso para facilitar el análisis de un cambio en alguno de estos aspectos.

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

7. Metodología propuesta

La arquitectura en si misma facilita el desarrollo ágil por las tecnologías empleadas y el bajo acoplamiento entre las capas que la conforman, pero para lograr agilidad en el proceso de desarrollo es necesario seguir una metodología acorde con el desarrollo ágil de aplicaciones como pueden ser SCRUM o eXtrem Programing

Aunque ambas metodologías tienen sus diferencias, SCRUM se basa más en la administración del proyecto mientras que eXtrem programaming se centra más en la creación del producto, ambas metodologías procuran un desarrollo iterativo con ciclos de entrega de producto "*operativo*" menores al mes, en los que la adaptación a los cambios en los requisitos es uno de sus principios clave.

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

8. Mejoras

1. Versión con Spring Web Flow
2. Versión con JdbcTemplate
3. Sistema de integración continua:

   Para agilizar el desarrollo y mejorar la calidad del código podrían utilizarse las siguientes herramientas:

   Hudson es un servidor de integración continua. Hudson se encarga de, cada vez que hay un cambio en el repositorio de código compilar todo el proyecto, lanzar las pruebas, compilar otros proyectos dependientes y, en general, lanzar cualquier otra tarea que creamos conveniente (por ejemplo lanzar Sonar).

   Con esto lo que conseguimos es detectar los posibles errores de integración en cuestión de minutos. Si un desarrollador sube al repositorio código que en su máquina local funcionaba, pero que al integrarlo con el código del resto de compañeros entra en conflicto, Hudson lo detectará en cuestión de minutos y mandará un correo electrónico para avisar del problema.

   Además Hudson guarda el histórico, de forma que podemos ver si nuestro código evoluciona adecuadamente, cuantas veces hemos roto al compilación (se sube al repositorio código que no compila), o que test fallan (se sube código que compila pero que no funciona), …

   Sonar es un servidor que, en función de una serie de reglas, es capaz de hacer un análisis estático de nuestro código. Realmente Sonar por debajo se basa en PMD, Checkstyle y FindBugs, de forma que con Sonar podemos ver estas herramientas de una forma unificada, como si se tratara de una sola.

   Con Sonar también podemos detectar código duplicado, e incluso medir la cobertura de nuestros test. Es decir, podemos saber que líneas de nuestro código se ejecutan con los test y cuales no.

   Al igual que Hudson, Sonar guarda el histórico del proyecto y nos puede mostrar gráficas donde vemos el avance del proyecto.

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com

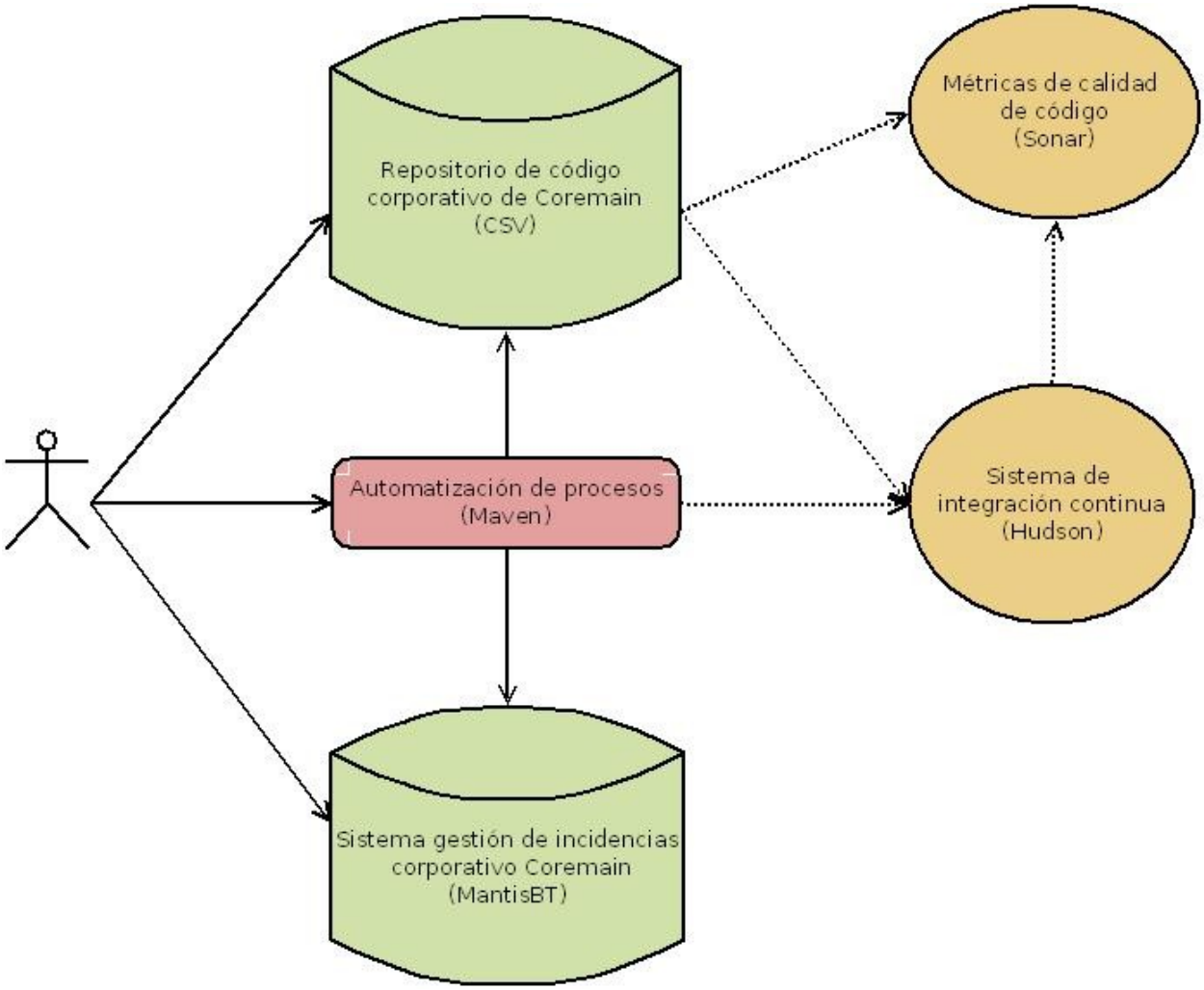

Ilustración 2: Integración continua utilizando Maven y Hudson

OFICINAS CENTRALES
Rua de Amio ,128.
Polígono Empresarial Costa Vella
15705 Santiago de Compostela
T 902 905 590
F 981 571 425
www.**core**main.com