tecnim@p Sevilla 2006
30 mayo a 2 junio
060.es
24x7x365

Comunicación

202

EL FRAMEWORK DE DESARROLLO DEL CONSEJO SUPERIOR DE INVESTIGACIONES CIENTÍFICAS

Ángel L. Rodríguez Alcalde
Director de la OPCSIC
Centro Técnico de Informática
(CSIC)

Clara Cala Rivero
Directora Centro Técnico de Informática
(CSIC)

Ángel L. Rodríguez Alcalde
Clara Cala Rivero

Palabras clave

Framework, J2EE, Java, Arquitectura Orientada a Servicios, Spring, arquitectura tecnológica.

Resumen de su Comunicación

En el último año, el CSIC ha emprendido la tarea de llevar a cabo un Plan de sistemas con el objetivo de adecuar sus sistemas, sus redes de comunicaciones y sus entornos y criterios de desarrollo a las tendencias actuales. La presente comunicación es una breve introducción al ámbito del desarrollo de aplicaciones J2EE del CSIC.

Ángel L. Rodríguez Alcalde
Clara Cala Rivero

El Framework de desarrollo del Consejo Superior de Investigaciones Científicas

1. Modelo de arquitectura tecnológica. La conceptualización de un framework corporativo. Definición.

En el desarrollo de software, un **framework** es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Normalmente, un **framework** incluye soporte de programas, librerías y, entre otros, un lenguaje de **scripting** que sirve de apoyo en el desarrollo y la integración de los diferentes componentes de una aplicación.

Un **framework** representa una **Arquitectura de Software** que modela las relaciones generales de las entidades del dominio. Al tiempo, proporciona una infraestructura y una "forma de trabajar" que extiende y/o utilizan las aplicaciones del dominio.

Características generales del framework

Las tendencia actual propone que cualquier servicio ofertado deberá combinar información existente en los distintos sistemas EIS (Enterprise Information Systems) de la organización. Las condiciones bajo las que ha de estar se centran en la alta disponibilidad, la seguridad, la fiabilidad y la escalabilidad.

La arquitectura para ofrecer estos servicios será multicapa, en la que en un extremo están los clientes, en el otro los sistemas EIS, ambos se articulan a partir de una capa intermedia que implementa las funciones de acceso a los EIS, formateo de la información de presentación y control. Esta capa intermedia para clientes web se ha desarrollado utilizando servidores de aplicaciones.

En este mercado arquitectónico existen dos grandes tecnologías J2EE y .NET. El CSIC se ha decantado por la propuesta J2EE. Alguna de las razones las veremos más adelante.

Las aplicaciones orientadas a la gestión de la información generan una problemática específica del mundo del desarrollo y de la arquitectura de aplicaciones (objetos distribuidos, transacciones, acceso a BBDD, generación dinámica de presentaciones, mensajes, contenedores, seguridad, etc.) que dificultan el desarrollo.

Parece evidente que la utilización de una plataforma que enmascare estas complejidades es un punto a tomar en consideración. Si se puede encontrar una solución que proporcione otras capacidades que permita emprender desarrollos de envergadura como es el caso de algunos de los proyectos definidos en el Plan de Sistemas del CSIC, consideramos que el tiempo necesario para definir un framework de desarrollo y ejecución es parte del tiempo necesario para llevar adelante el Plan con garantías de éxito.

El framework del CSIC cuenta con una serie de ventajas operativas:

- **Orientación a Servicios (SOA)**, La mayor parte de la funcionalidad que proporciona el framework se expone como servicios. Esta visión logra simplificar la integración de las operativas desarrolladas y lo hace de forma independiente a la tecnología empleada para resolverla.
- **Integración de Sistemas**, el framework proporciona las herramientas y componentes necesarios para integrar sistemas, permitiendo una reutilización sencilla de elementos desarrollados por terceros.
- **Extensibilidad** para nuevos servicios y tecnologías. Permite la adaptación de la plataforma a las necesidades de la organización y proporciona un amplio margen para su crecimiento futuro.

- **Adopción de estándares** que aseguren el soporte de mercado presente y futuro.
- **Desarrollo homogéneo y productivo**, gracias a una serie de modelos de aplicación definidos, una metodología de desarrollo y una arquitectura común.
- **Unificación en el proceso de desarrollo** gracias a una serie de modelos de patrones y productos documentados, herramientas de desarrollo y una arquitectura común.
- **Unificación de plataformas**, permitiendo reducir la heterogeneidad tecnológica, que conlleva aumento de costes y mayor complejidad.
- **Elevada modularidad**
- **Flexibilidad ante cambios y/o nuevos requerimientos**
- **Minimización del desarrollo**, debido a la existencia de servicios comunes que convierten, en muchos casos, problemas de desarrollo en tareas de parametrización y configuración de servicios ya existentes (acceso a base de datos, acceso a servicios, gestión XML, correos...).
- **Ocultación de la complejidad tecnológica** por la plataforma, aislando al desarrollador de aplicaciones de la complejidad asociada a la tecnología base; aspecto, este último, que proporciona una mayor productividad.

2. Base tecnológica

Como hemos visto, se puede decir que los marcos de infraestructura de Aplicaciones (frameworks) se diseñan y optimizan para resolver la mayor parte de la complejidad de los desarrollos, aprovechando las mejores prácticas y facilitando la construcción de nuevas aplicaciones.

En la actualidad asistimos a una enorme complejidad tecnológica, así tenemos J2EE, EJB, XML, etc. Aunque es cierto que la sencillez y la potencia de una herramienta suelen ser aspectos inversamente proporcionales, algunos arquitectos importantes de la comunidad Java trabajan para cambiar la tendencia desde los frameworks monolíticos hacia otros más sencillos y limpios. Así, varios proyectos se oponen a la tendencia a una mayor complejidad, optando por construir frameworks específicos más ligeros que ayuden a simplificar las aplicaciones.

El framework diseñado por el CSIC utiliza la tecnología J2EE. Esta decisión se aplica también a todas las herramientas que se utilizan en el desarrollo de aplicaciones.

Java es una plataforma virtual de software desarrollada por Sun Microsystems, de modo que los programas creados puedan ejecutarse sin cambios en diferentes tipos de arquitecturas y dispositivos informáticos.

Java se ha hecho muy popular desde que Sun Microsystems introdujo la especificación J2EE (Java 2 Enterprise Edition). Este modelo permite, entre otras cosas, lograr una separación entre la presentación de los datos al usuario (JSP o Applets), el modelo de datos (EJB), y el control (Servlets). Enterprise Java Beans (EJB) es una tecnología de objetos distribuidos que pudo lograr el sueño de muchas empresas de crear una plataforma de objetos distribuidos con un monitor de transacciones.

J2EE es la edición empresarial del paquete Java creado y distribuido por Sun Microsystems. Comprenden un conjunto de especificaciones y funcionalidades orientadas al desarrollo de aplicaciones empresariales. Debido a que J2EE no deja de ser un estándar, existen otros productos desarrollados a partir de ella aunque no exclusivamente.

La modelización que se plantea en los siguientes apartados para el framework del CSIC se basa en los siguientes principios: **Objeto Persistente, Lightweight Container, Abstraction with loss, IoC, AOP**. Veamos brevemente sus principales características.

Persistencia

Las aplicaciones que se desarrollan en el CSIC estructuran la información en bases de datos. Debido a que las bases de datos utilizan un modelo teórico llamado relacional y los programas de ordenador suelen utilizar el paradigma de la programación orientada a objetos, que difiere mucho del modelo relacional, se hace necesario un componente de software que permita la comunicación de estos dos instrumentos. Al componente software encargado de realizar esta tarea se le conoce como capa de persistencia.

La capa de persistencia **traduce** entre los dos modelos de datos: de registros a objetos y de objetos a registros. Cuando el programa quiere grabar un objeto llama al motor de persistencia, que traduce el objeto a registros y llama a la base de datos para que guarde estos registros. De la misma manera, cuando el programa quiere recuperar un objeto, la base de datos recupera los registros correspondientes, los cuales son traducidos en formato de objeto por el motor de persistencia. De esta manera el programa sólo ve que puede guardar objetos y recuperar objetos, como si estuviera programado para una base de datos orientada a objetos. La base de datos sólo ve que guarda registros y recupera registros, como si el programa estuviera dirigiéndose a ella de forma relacional.

Inversión del Control

La **inversión del control (IoC)** es parte fundamental de lo que hace un framework distinto a una librería. Una librería es esencialmente un conjunto de funciones, generalmente organizadas dentro de clases, a las cuales se pueden realizar llamadas. Cada llamada realiza un determinado trabajo y devuelve el control al cliente. Un framework abarca un diseño abstracto, con más comportamiento construido en su interior.

Para utilizarlo es necesario insertar el comportamiento dentro de varios sitios en el interior del framework en lugar de crear subclases o introducirlas dentro de las clases ya existentes. La forma de trabajar del framework está más orientada al manejo de eventos.

La inversión del control mueve la responsabilidad de hacer las cosas ocurridas en el interior del framework y fuera del código de la aplicación. Mientras que el código llama a una clase tradicional, **un framework IoC llama al código.**

La inyección de dependencias es una forma de IoC que elimina la dependencia explícita en los contenedores de API´s; Normalmente se emplean métodos de Java para inyectar dependencias tales como objetos colaboradores o valores de configuración dentro de instancias de objetos de la aplicación. El contenedor hace estas suposiciones basándose en archivos de configuración en XML.

La inyección de dependencias tiene varios importantes beneficios. Por ejemplo:

- Dado que los componentes no necesitan buscar las colaboraciones en tiempo de ejecución, son mucho más simples de escribir y mantener.
- Por la misma razón, el código de la aplicación es mucho más sencillo de probar.
- Una buena implementación de IoC ahorra escribir mucho texto. Con IoC las dependencias se expresan en el código y el framework es el responsable de escribir los moldes. Esto significa que no hay que considerar excepciones del molde en el código.
- Las dependencias son explícitas.

Contenedores Ligeros (Lightweight Containers)

Muchos API de contenedores, tales como los API EJB, obligan a codificar alguna interfaz o componente de modelo. Los contenedores servlet, como Apache Tomcat, implementan el API servlet permitiendo incorporar contenido dinámico en las páginas del servidor, las cuales pueden ser enviadas a un navegador web.

Los contenedores tradicionales obligan a utilizar un modelo de programación dado. **Los contenedores ligeros no:** permiten insertar **POJOs (Plain old Java Object)** que el contenedor enlaza los entre si y les proporciona servicios.

Las características más comunes de los contenedores ligeros incluyen:

- Programación basada en POJOs: Los contenedores ligeros no son evasivos, no refuerzan ninguna API.
- Gestión completa del ciclo de vida: Los contenedores ligeros gestionan el ciclo de vida de los objetos que se sitúan dentro. Como mínimo instancian y destruyen objetos.
- Resolución de dependencias: Proporcionan una estrategia de resolución de dependencias común. La mayoría implementan inyección de dependencias.
- Configuraciones consistentes: Los contenedores ligeros son un lugar adecuado para proporcionar servicios de configuraciones consistentes.
- Conectar servicios: Los contenedores ligeros proveen una manera de conectar servicios a los objetos en el contenedor.

Los contenedores ligeros poseen muchas ventajas sobre otras arquitecturas de contenedores. Por ejemplo permiten usar un modelo sencillo de programación basado en POJOs. Esto implica que las aplicaciones son más sencillas de probar. Los objetos además pueden correr fuera del contenedor (por ejemplo en un caso de prueba). **A través de la inyección de dependencias los contenedores ligeros reducen las dependencias entre los componentes**. Además, protegen la inversión en código al permitir mover más de la aplicación entre contenedores.

Abstracciones con fugas (abstraction with loss)

Todas las abstracciones tienen fugas. Hay situaciones en que no se puede o no se debería abstraer la infraestructura subyacente. Por ejemplo, nunca será posible capturar toda la complejidad de la API nativa de una base de datos en un framework simplificado, y un contenedor ligero nunca será capaz de hacer todo lo que hace un contenedor pesado. Por esta razón, es importante exponer la capa inmediatamente inferior a la nueva abstracción. Por ejemplo, en **Hibernate** (el motor de persistencia) no existe el concepto de procedimientos almacenados. En vez de esto, se permite el acceso completo a la conexión JDBC subyacente, para que se pueda llamar a los procedimientos almacenados desde fuera de **Hibernate.**

Además, una arquitectura adaptable y conectable, con bajo acoplamiento en las áreas apropiadas, permite desconectar los servicios que son inadecuados y conectar otros más robustos. Por ejemplo, Spring permite conectar y desconectar la persistencia y las transacciones.

Por tanto nunca se llega a producir un proceso real de abstracción, dado que siempre es necesario que la abstracción sea parcial y no es posible abstraerse completamente de la complejidad dado que es necesario conocer el nivel de complejidad subyacente.

Programación Orientada a Aspectos (Aspect Oriented Programming)

La **programación orientada a aspectos (AOP)** complementa la programación orientada a objetos (OOP) permitiendo a los desarrolladores modificar dinámicamente el modelo estático orientado a objetos para crear un sistema que pueda crecer para ajustarse a nuevos requerimientos. Del mismo modo que los objetos del mundo real pueden modificar su estado a lo largo de sus ciclos de vida, una aplicación puede adoptar nuevas características a lo largo de su desarrollo.

La AOP permite a los desarrolladores modificar dinámicamente el modelo estático para incluir el código necesario para cumplimentar los requisitos secundarios sin necesidad de modificar el modelo estático ori-

ginal. Mejor aún, será posible a menudo mantener este código adicional en una única localización en lugar de tenerlo repartido a través del modelo existente.

La AOP permite una forma distinta de pensar sobre la estructura del programa. Mientras que la OOP descompone la aplicación en una jerarquía de objetos, AOP descompone los programas en aspectos o conceptos. Esto permite la modularización de conceptos tales como la gestión de transacciones que de otra forma podría trascender múltiples objetos.

Fundamentalmente la manera en la que los aspectos interactúan con el programa base es definido mediante un modelo de **join points** en el cual se especifica el aspecto. El modelo define tres cosas.

- Dónde se pueden aplicar los aspectos. Normalmente conocido como join points.
- Una manera de especificar múltiples join points. Llamado normalmente pointcuts (los pointcuts son realmente una consulta sobre todos los join points encargada de seleccionar un subconjunto de los mismos).
- Un medio para afectar el comportamiento de los joint points. Llamado normalmente advice.

Las características del framework base seleccionado

Con la llegada de Spring y la revisión de EJB (3.0) disponemos de la posibilidad de elegir el framework que mejor se adapte a sus necesidades. Frente a EJB, Spring surge como una solución independiente, supone un enfoque alternativo de software libre, para afrontar gran parte de los problemas presentes en un desarrollo real.

El actual framework EJB 2.1 en Java 1.4 está considerado un framework pobremente diseñado y demasiado complicado. La insatisfacción producida por este framework ha hecho moverse a los programadores en Java hacia los dos frameworks que actualmente despiertan más interés:

- -El framework Spring: Es un framework de código abierto bastante popular. La arquitectura de Spring está basada en el patrón de inyección de dependencias (DI). Spring puede trabajar de manera independiente o con servidores de aplicaciones existentes y hacer uso de sus potentes archivos de configuración en XML.
- El framework EJB 3.0: Es un framework estándar definido por la JCP de Java y soportado por varios vendedores de J2EE. Ya hay disponibles implementaciones de la especificación de la pre-release EJB 3.0 por parte de JBoss y Oracle. EJB hace un gran uso de Java annotations.

El CSIC se ha decantado por Spring. Algunas de sus características que han propiciado esta decisión se basan en la enorme flexibilidad, frente a otras plataformas. Mientras que la integración del código de servicio en Spring se muestra como parte del interfaz de programación, los desarrolladores de la aplicación tienen la flexibilidad de ensamblar los servicios que necesiten. Esta característica permite crear servidores de aplicaciones personalizados de poco peso. Un uso muy común de Spring es combinarlo con Hibernate para proveer servicios de transacción y proporcionar servicios de persistencia.

Por su parte los servidores de aplicaciones EJB 3.0 normalmente no permiten esa flexibilidad de elegir y montar los servicios que se necesiten. La mayoría de las veces se dispone de un conjunto de características pre-empaquetadas, muchas de las cuales puede que no se necesiten. Sin embargo en algunos servidores de aplicaciones, como JBoss (el servidor de aplicaciones corporativo del CSIC), podría ser posible separar las características que no sean necesarias y eliminarlas.

Ángel L. Rodríguez Alcalde
Clara Cala Rivero

3. Visión general de la arquitectura del framework

Hoy, el diseño de sistemas informáticos suele utilizar arquitecturas multinivel o multi-capa. El objetivo es la simplificación y la escalabilidad, a cada nivel se le confía una misión simple que permite la de arquitecturas profundamente escalables que pueden ampliarse con facilidad en caso de que las necesidades aumenten.

El diseño más popular es el construido en tres niveles, comúnmente llamados capas.

- **Capa de Presentación:** presenta el sistema al usuario, le comunica la información y captura la información del usuario realizando un procesamiento mínimo (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio.
- **Capa de Negocio:** en ella residen los programas que se ejecutan, recibiendo las peticiones del usuario y enviando las respuestas tras el proceso. Se denomina capa de negocio (o lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y entregar los resultados, y con la capa de datos, para solicitar que se almacenen o recuperen datos.
- **Capa de Datos:** encargada de hacer persistente toda la información, suministra y almacena información para el nivel de negocio.

Figura 1 Diseño en tres capas

Todas estas capas pueden residir en una única plataforma de hardware, si bien lo más usual es que sea en una estructura múltiple de ordenadores donde reside la capa de presentación. Las capas de negocio y de datos residen en distinto hardware. Así, si el tamaño o complejidad de la base de datos aumenta, se puede separar en varios ordenadores que recibirán las peticiones de la máquina en que resida la capa de negocio. Dada la complicación de las aplicaciones diseñadas en el CSIC se emplean ordenadores dedicados y por tanto, una arquitectura en tres capas y tres niveles.

Los distintos componentes del framework se unen formando parte de esta arquitectura. Cada componente no tiene por qué estar ligado únicamente a una sola de las capas, sino que puede participar en más de una de ellas.

La distribución de los componentes del framework en las distintas capas del modelo sería la siguiente:

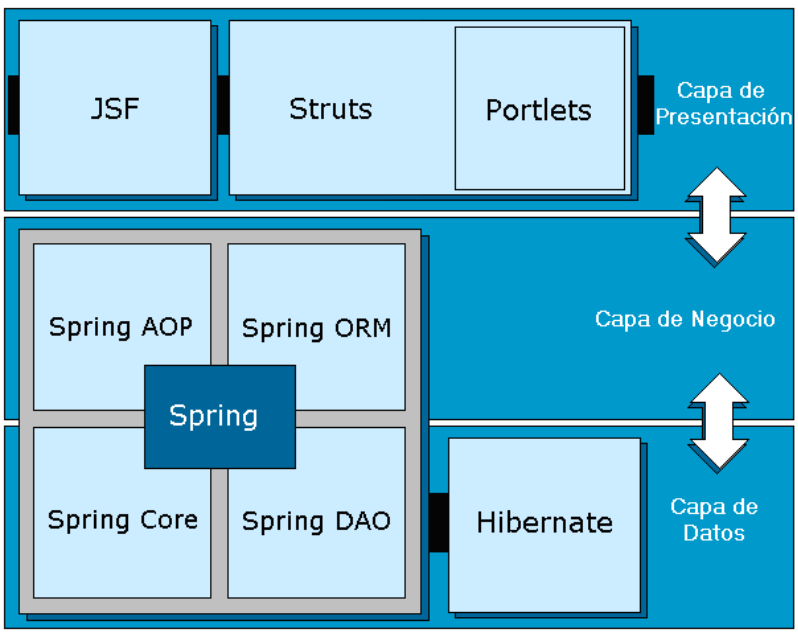


Figura 2 Framework dividido en capas

Los componentes de la capa de presentación colaboran entre ellos basándose en el Modelo Vista Controlador (MVC).

El centro de la arquitectura se encuentra dominado por Spring, el cual a su vez se divide internamente en capas.

Para comprender mejor el trabajo de las capas podemos condensar la arquitectura en tres partes fundamentales:

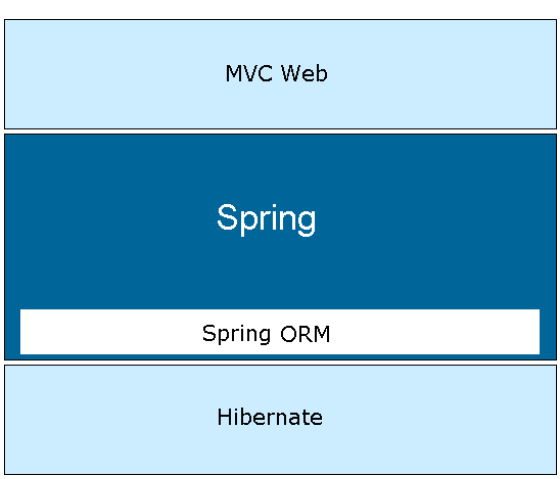
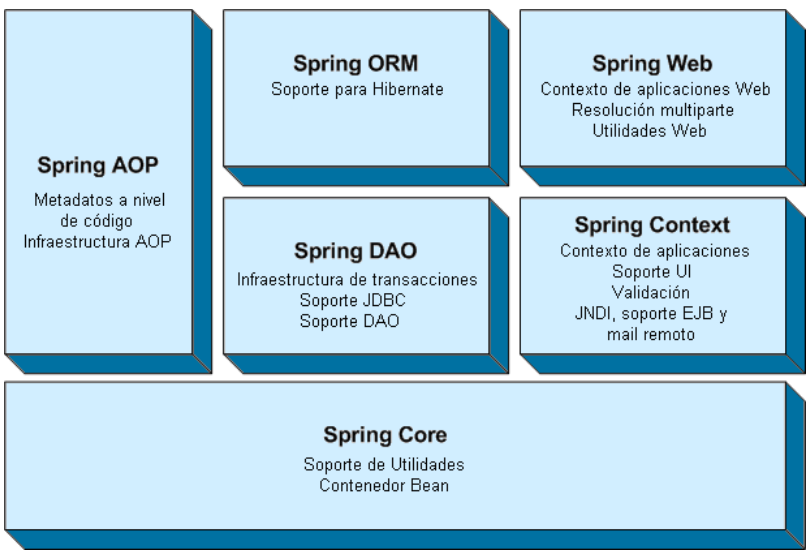


Figura 3 Arquitectura en bloques

Para la integración con Hibernate, Spring incorpora el potente paquete Spring ORM, mientras que para la colaboración con las tecnologías pertenecientes al MVC Spring posee su propia capa MVC. El esquema arquitectónico interno de Spring es el siguiente:



4. Conclusión

El framework definido por el CSIC demuestra una gran solidez y cohesión entre sus componentes. Se trata de una arquitectura que soluciona de manera eficiente la integración entre sus capas y demuestra solvencia a la hora de resolver los supuestos comunes de cada capa. Se basa en un modelo que puede proporcionar una mejora en la sencillez de creación de aplicaciones y facilidad de mantenimiento, tanto para aplicaciones departamentales como para aplicaciones web corporativas.