Accediendo a Base de Datos desde aplicaciones Web desarrolladas con J2EE: patrones de diseño.

Carlos Presedo Varela, Nieves R. Brisaboa, Antonio Fariña

Laboratorio de Bases de Datos. Departamento de Computación.
Universidad de A Coruña. Campus de Elviña, s/n. 15071.
A Coruña. Spain

c_presedov@yahoo.es, {brisaboa, fari}@udc.es

Resumen

En este artículo se presenta un conjunto de patrones de diseño que facilitan el acceso a Bases de Datos utilizando JDBC desde la capa modelo de aplicaciones Web desarrolladas según el patrón arquitectónico *Model - View - Controller*. También se presenta una aplicación práctica, el portal Web de la Real Academia Gallega, en el que se podrá ver la forma de utilizar estos patrones. Este portal ha sido desarrollado con el ánimo de suplir la carencia de información existente en la Web sobre la academia gallega, al mismo tiempo que pretende difundir y promover el uso del gallego en Internet. Este portal ha sido llevado a cabo completamente en el Laboratorio de Base de Datos de la Universidad de A Coruña en colaboración con la Real Academia Gallega y en su desarrollo se han aplicado variados patrones de diseño que han facilitado su comprensión y mantenimiento.

Palabras clave: patrones de diseño, J2EE, aplicaciones Web, acceso a Bases de Datos, patrón Data Access Object.

1. Introducción: Arquitectura general de las aplicaciones empresariales.

El rápido crecimiento de Internet en los últimos tiempos ha hecho que cada vez se demande más el desarrollo de aplicaciones distribuidas que trabajen de forma transaccional conservando niveles de rapidez, seguridad y escalabilidad aceptables. Para soportar la demanda de rendimiento de las nuevas aplicaciones Internet, la arquitectura cliente/servidor en dos capas ha evolucionado a estructuras más complejas formadas por más capas y en las cuales existe una separación clara de responsabilidades de cada una de las capas. En este contexto se suele aplicar el patrón *Model - View – Controller (MVC)* [5], que permite separar la lógica de la aplicación de la vista y el controlador, forzando así a desarrollar un diseño modular, mantenible y fácilmente escalable.

El patrón arquitectónico *MVC* hace una separación clara entre el modelo (lógica de negocio) y la vista (interfaz gráfica), gracias a un controlador que los mantiene desacoplados y al mismo tiempo se encarga de comunicarlos gestionando las peticiones del usuario (figura 1). De este modo se posibilita la reutilización de un mismo modelo con distintas vistas (por ejemplo, una vista Web y una basada en ventanas) al mismo tiempo que se pueden crear roles de trabajadores que faciliten la separación del trabajo entre equipos especializados.

La misión de cada una de las capas del patrón *MVC* es la siguiente:

- *Capa vista*. Formada por el conjunto de interfaces de usuario, se encarga de interactuar con el usuario. Usando *J2EE* [2, 3 y 8] esta capa suele implementarse mediante páginas JSP [1 y 9].

- *Capa controlador*. Encargada de realizar la comunicación vista – modelo para que éste último atienda las peticiones realizadas por los usuarios. Esta capa suele implementarse como uno o varios Servlets [10] al trabajar con *J2EE*.

- *Capa modelo*. En esta capa reside la lógica de la aplicación, independiente de la vista y del controlador. Esta capa suele hacer uso de una Base de Datos para llevar a cabo las operaciones solicitadas por el

controlador y se suele implementar utilizando *Enterprise Java Beans (EJB)* [2 y 12] o usando clases que acceden a Base de Datos a través *de JDBC (Java DataBase Connectivity)* [1 y 11].

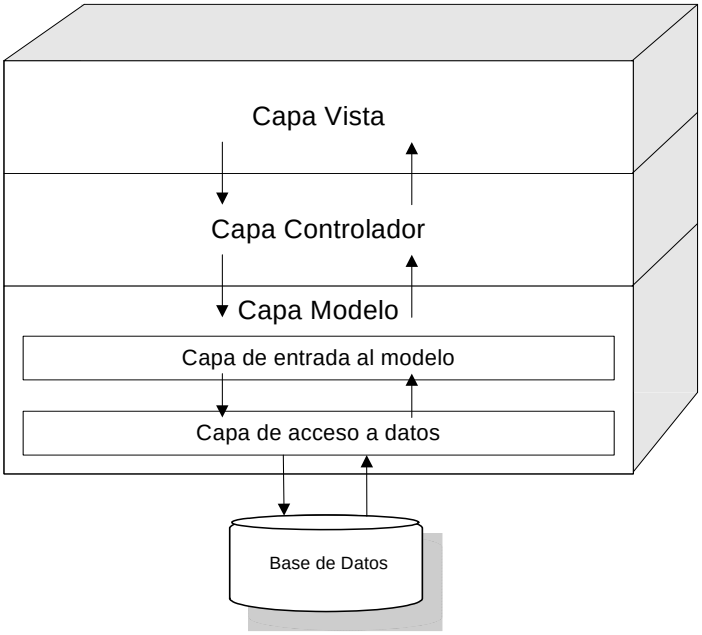

Figura 1. Capas del patrón arquitectónico Model – View – Controller.

Para modelar la lógica de acceso a Base de Datos desde la capa modelo de aplicaciones Web desarrolladas usando J2EE existen una serie de patrones de diseño que facilitan la labor de diseñadores y desarrolladores. Estos patrones son el *patrón Value Object* [2, 3, 5, 6 y 7], que permite agrupar atributos procedentes de uno o varios objetos del dominio, el *patrón Data Access* Object [2, 3, 5, 6 y 7], que permite desacoplar la lógica de negocio del acceso a datos y el *patrón Version Number* [2 y 5], que permite gestionar los cambios en los datos cuando existen vistas de actualización.

2. Descripción de patrones.

A continuación se describen los patrones de diseño utilizados para acceder a Base de Datos desde la capa modelo en una arquitectura en tres capas usando tecnología JDBC.

2.1 Patrón Value Object.

- **Intención**. Agrupar un conjunto de atributos procedentes de uno o varios objetos del dominio y facilitar el intercambio de datos entre la capa modelo y la capa vista.

- **Estructura**.

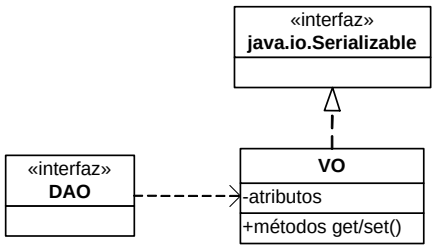

Figura 2. Estructura del patrón Value Object.

- **Participantes**.
  - *VO*. El objeto valor.
  - *DAO*. Son los encargados de manejar la persistencia de los VO.

- **Colaboraciones**. Un DAO devuelve VOs en sus métodos de búsqueda (*find*) y los recibe en sus métodos creación (*create*) y actualización (*update*).

- **Aplicabilidad**.
  - Cuando necesitamos representar un conjunto de atributos procedentes de uno o varios objetos del dominio.

- **Consecuencias**.
  - *Beneficios*.
    - Permite representar un conjunto de atributos procedentes de uno o varios objetos del dominio.
  - *Riesgos*.
    - Información obsoleta.

- **Variantes**.
  - *Domain Value Object*. Un Value Object es un Domain Value Object cuando sus atributos corresponden a los de un objeto del dominio.
  - *Custom Value Object*. Se denominan así aquellos Value Object que son específicos de un caso de uso, es decir, que sólo tienen los atributos necesarios para ese caso de uso.
  - *Data Transfer HashMap*. En una aplicación de tamaño medio puede existir un gran número de Value Objects, lo que genera un problema de mantenimiento. La solución es usar mapas para almacenar pares <*nombre-atributo, valor*> en vez de las clases usadas hasta ahora, aunque tiene el inconveniente de que es necesario establecer convenciones de nombrado.

2.2 Patrón Data Access Object.

- **Intención**. Desacoplar la lógica de negocio de la lógica de acceso a datos, de modo que se pueda cambiar la fuente de datos fácilmente.

- **Estructura**.

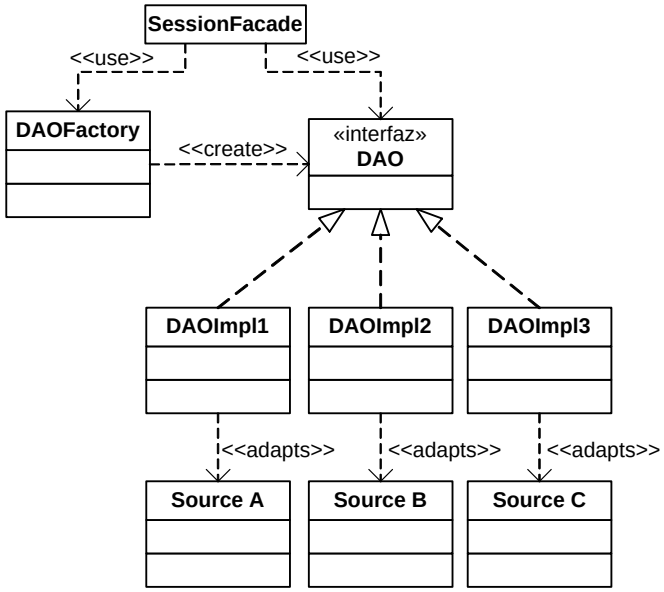

Figura 3. Estructura del patrón Data Access Object.

- **Participantes**.
  - *Session Facade*. Abstrae las operaciones de negocio y para llevarlas a cabo utiliza los DAOs para obtener los datos. Gracias a la interfaz DAO no depende de la fuente de datos.
  - *DAOFactory*. Clase factoría encargada de crear una instancia del DAO del tipo adecuado según la fuente de datos utilizada.
  - *DAO*. Abstrae las operaciones sobre la fuente de datos, proporcionando una API para acceder y manipular datos.
  - *DAOImpl*. Adapta el interfaz anterior a una fuente de datos concreta.
  - *Source (Oracle, Informix, Sybase, PostgreSQL, SQL Server, BD Orientada a Objetos, fichero plano, servidor LDAP, etc.).* Proporciona acceso y manipulación de datos mediante una API que necesita ser adaptada.

- **Colaboraciones**. Un SessionFacade accede a los datos a través de un DAO, el cual adapta el API que ofrece la fuente de datos.

- **Aplicabilidad**.
  - Para separar la lógica de negocio de la lógica de acceso a datos.
  - Poder seleccionar el tipo de fuente de datos durante la instalación de una aplicación.

- **Consecuencias**.
  - *Beneficios*.
    - Independencia del vendedor de la fuente de datos.
    - Extensibilidad, ya que se facilita el cambio de fuente de datos. Los métodos del DAO reciben y devuelven Value Objects, por lo que aunque se cambie la fuente de datos, el resto de objetos que utilizan los DAOs no sufren modificaciones.
  - *Riesgos*.
    - Más complejidad.

2.3 Patrón Version Number.

- **Intención**. Evitar la actualización de un Value Object construido a partir de información obsoleta.

- **Problema**. Supongamos que disponemos de una aplicación empresarial que permite tanto la consulta como la actualización de datos. Dos administradores obtienen la misma información a partir de un Value Object y uno de ellos la modifica. Sin actualizar la información que está visualizando, el otro administrador modifica los datos y solicita grabarlos. En este caso, la segunda actualización sobrescribe a la primera.

- **Solución**. El Value Object debería tener un atributo privado *versionNumer* de tipo entero[1] y un método *get()* para leer su valor. En consecuencia, la tabla subyacente que da soporte a la persistencia de este objeto también debería presentar una columna *versionNumber*. Cada vez que se actualiza la información asociada a este objeto en Base de Datos se debe realizar el siguiente proceso (ejecutado en la misma transacción):
  - Leer de la Base de Datos el *versionNumber* del objeto.
  - Si es igual al del Value Object, se incrementa este *versionNumber* y se actualiza el objeto.
  - En otro caso se lanza una excepción.

- **Estructura**.

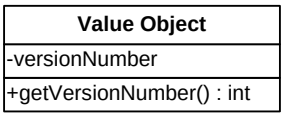

Figura 4. Estructura del patrón Version Number.

[1] Existen descripciones de este patrón que usan un *timestamp* en vez del número de versión descrito.

- **Aplicabilidad**.
  - Cuando se desea consultar y actualizar la versión adecuada de una información.

- **Consecuencias**.
  - *Beneficios*.
    - Se reduce el riesgo de información obsoleta.
  - *Riesgos*.
    - Más complejidad.

2.4 Funcionamiento global

Mediante la aplicación de los patrones anteriores puede realizarse el modelado e implementación del acceso a datos desde la capa cliente de un modo sencillo y fácilmente mantenible. Para ello, las clases presentadas en esos patrones colaboran como se muestra en el ejemplo de la figura 5, en el cual se realiza una búsqueda.

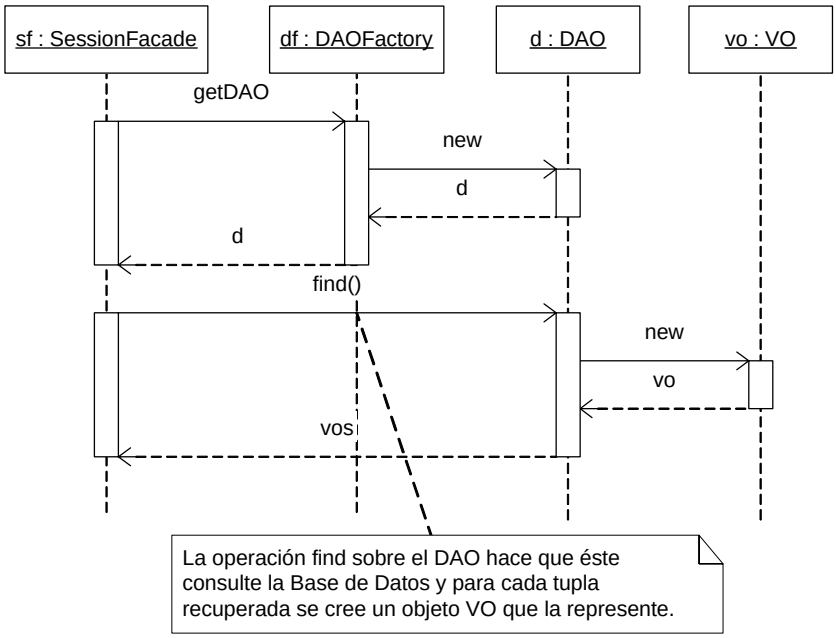

Figura 5. Diagrama de secuencia de los patrones presentados.

La clase factoría (*DAOFactory*) recibe una petición y devuelve una instancia de la clase DAO adecuada para la fuente de datos utilizada, como por ejemplo una implementación para Oracle. Sobre esta clase DAO, los clientes invocan operaciones (como por ejemplo una búsqueda), trabajando siempre con Value Objects. De este modo, si se cambia de fuente de datos basta con que la clase factoría devuelva una instancia adecuada para esa fuente de datos, de modo que los clientes del DAO no tienen que cambiar su implementación, pues trabajan en términos de interfaces. Esto favorece claramente el mantenimiento, pues se independiza la aplicación de posibles cambios en el almacén de datos.

3. Caso práctico: Portal Web de la Real Academia Gallega

La solución de diseño basada en los patrones anteriores ha sido aplicada para el desarrollo del portal Web de la Real Academia Gallega[2] (RAG), proyecto llevado a cabo dentro del Laboratorio de Bases de Datos de la Universidad de A Coruña. Dicho portal trata de cubrir dos objetivos:

- Cubrir el vacío existente en Internet acerca de información sobre la Real Academia Gallega, sus académicos y actividades. Desde su fundación en 1906, la Real Academia Gallega ha sido testigo de numerosos acontecimientos relacionados con Galicia y ha recopilado gran cantidad de información y obras de incalculable valor literario, por lo que se hacía indispensable difundir este importante patrimonio de la cultura gallega.

- Promover el uso del gallego en Internet, tratando así de evitar su desaparición. Hoy en día, la ausencia de una lengua en el dominio de las nuevas tecnologías en general, y en Internet en particular, incrementan su posibilidad de desaparecer. Además, el gallego es una de las lenguas representadas en el Comité Europeo titulado *European Lesser Used Languages* (Lenguas Europeas poco usadas) y recientemente ha sido incluido en la lista de 3.000 lenguas en peligro de la UNESCO [4].

Desarrollado con tecnología J2EE y usando JDBC para acceder a una Base de Datos mySQL, el portal se ha dotado de los siguientes servicios o secciones con el fin de cumplir los objetivos expuestos anteriormente:

- *Sección Información Institucional*. Da a conocer los orígenes de la academia, su historia, los proyectos actuales de la institución, etc.

- *Sección Académicos*. Esta sección ofrece contenidos acerca de los académicos actuales y pasados de la institución gallega, permitiendo la consulta de la composición de la academia en una fecha cualquiera desde su creación, y pudiendo acceder a partir de la pantalla de resultados a la ficha personal de cada uno de los académicos. También ofrece la posibilidad de realizar un seguimiento histórico de cada uno de los sillones que forman la academia.

- *Sección Actualidad*. Ofrece noticias relacionadas tanto con la Real Academia Gallega como con la cultura gallega en general.

- *Sección Biblioteca*. En esta sección los visitantes pueden consultar el rico catálogo bibliográfico con el que cuenta la Real Academia Gallega, y que constituye una de las mejores bibliotecas existente sobre literatura gallega. Se permiten realizar búsquedas sobre el catálogo por diversos criterios, como autor, título o fecha de publicación.

- *Sección Enlaces*. Esta sección constituye un directorio de enlaces a las principales páginas de la Web gallega (páginas institucionales, editoriales, imprentas, etc.) incrementando así el atractivo del portal y convirtiéndolo en un referente cultural de primera magnitud desde el cual los navegantes puedan acceder a toda la información relacionada con la cultura y la literatura gallega.

La información manejada por cada uno de estos servicios se almacena en Base de Datos, y para su gestión se utilizan los patrones descritos anteriormente. A continuación se presenta, a modo de ejemplo práctico, el diseño de la sección de Académicos.

3.1 Diseño de la sección Académicos.

Como se ha comentado, la *sección de Académicos* permite consultar la composición de la Real Academia Gallega en una fecha cualquiera desde su creación en 1906, así como acceder a información sobre cada uno de los académicos que han formado parte de ella. Sobre cada uno de los académicos se ofrece variada información, como

[2] El portal de la Real Academia Gallega puede consultarse en http://www.realacademiagalega.es. En la actualidad se está realizando la carga de datos, por lo que el portal todavía no es accesible. De todos modos, en breve se podrá acceder a él, pudiendo estar disponible en el momento de presentación de este artículo.

biografía y datos personales, datos relacionados con la academia (fecha de entrada y de salida, cargos, etc.) y discurso de entrada en la misma.

El modelo de clases diseñado para acceder a Base de Datos en esta sección se presenta en la figura 6. Como puede observarse, se han utilizando los patrones descritos en las secciones anteriores.

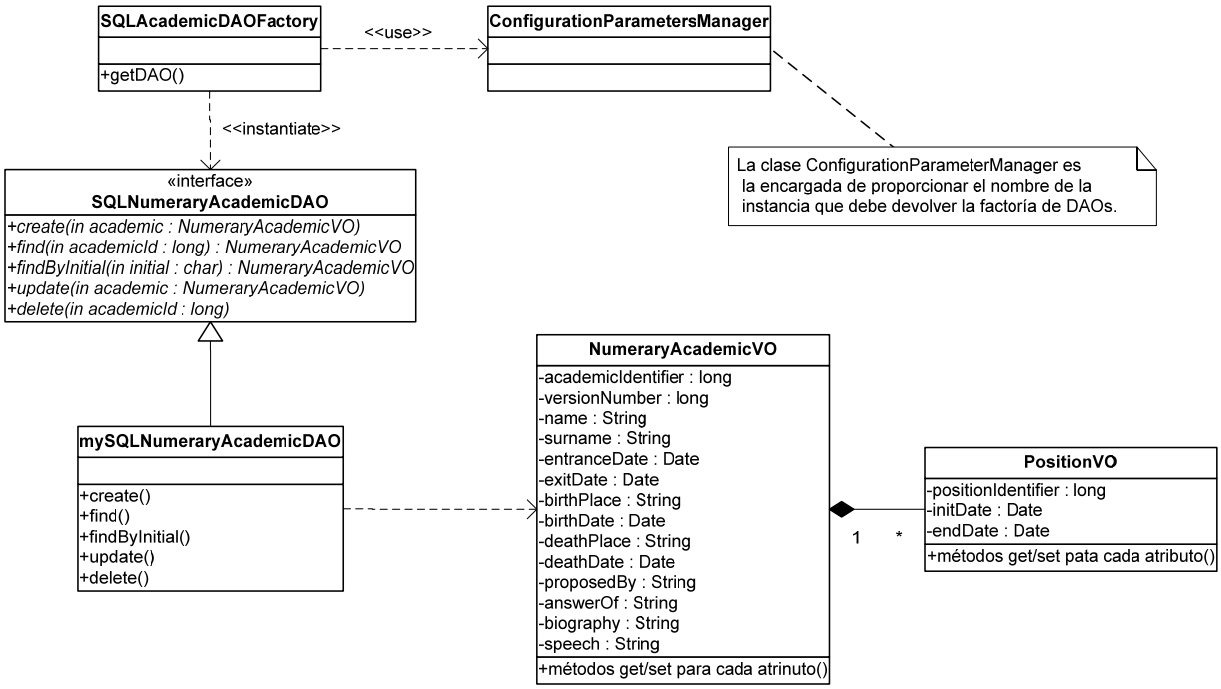

Figura 6. Diseño de la capa de acceso a datos de la sección de académicos usando los patrones *VO*, *DAO* y *Version Number*.

Para diseñar el concepto de académico numerario se aplica el patrón *Value Object*, de modo que se crean las clases necesarias para modelar su estado (*NumeraryAcademicVO* y *PositionVO*). Estas clases presentan todos los atributos necesarios y métodos *get* y *set* para cada uno de ellos, permitiendo así tanto el acceso como la modificación del estado de un académico en cualquier momento. No se incluye ninguna lógica de negocio en estas clases, independizándolas así de posibles cambios en los procesos a realizar.

La clase *SQLNumeraryAcademicDAO* representa la API para manejar la persistencia de los académicos numerarios. Publica operaciones de creación (*create*), de búsqueda (*find* y *findByInitial*), de actualización (*update*) y de borrado (*delete*). Sobre esta API se proporcionan distintas implementaciones para distintas fuentes de datos, como la clase *mySQLNumeraryAcademicDAO*, la cual implementa todos los métodos publicados usando las peculiaridades del lenguaje SQL del gestor relacional mySQL. En el caso de que se desease cambiar el gestor de Bases de Datos (por ejemplo, migrando la Base de Datos a Oracle) o incluso que se cambiase la tecnología de almacenamiento (por ejemplo, usando ficheros XML o Bases de Datos Orientadas a Objetos), bastaría con proporcionar una nueva implementación de la interfaz *SQLNumeraryAcademicDAO* para la fuente de datos específica y configurar la factoría[3] para que devuelva instancias de la nueva clase. De este modo, los clientes que requieran acceso a datos no se ven afectados ante cualquier cambio ya que trabajan en términos de interfaces (trabajan contra la interfaz *SQLNumeraryAcademicDAO*).

[3] Al tratarse de una aplicación Web desarrollada con Java (J2EE) la especificación de la instancia a devolver se hace a través de la declaración de una variable con el nombre de la instancia en el fichero de configuración de la aplicación Web (web.xml).

4. Beneficios obtenidos

Tradicionalmente, las aplicaciones empresariales desarrolladas utilizando tecnología J2EE suelen organizarse como un conjunto de páginas JSP que engloban todo el código de la aplicación, tanto la vista del sistema como la lógica de negocio y la lógica de acceso a Base de Datos. Esta organización complica el mantenimiento de los sistemas, requiriéndose la presencia de personal técnico formado en dicha tecnología para acometer cualquier tipo de cambio, incluso cambios en la vista de la aplicación.

La arquitectura y patrones presentados en apartados anteriores permiten resolver este problema. El diseño de aplicaciones Web desarrolladas a partir de los patrones descritos proporcionan dos claros beneficios: facilidad de mantenimiento y separación de roles.

Ambas ventajas suponen un ahorro de tiempo, lo cual repercute directamente en los costes de desarrollo y mantenimiento del proyecto. Por una parte, la utilización de patrones conocidos y probados permite establecer la base de comunicación entre los miembros del equipo encargado de desarrollar el proyecto, al mismo tiempo que facilita la comprensión de las aplicaciones en las labores de mantenimiento, con la consecuente reducción de tiempo y coste. Por otro lado, la división de los sistemas en tres capas bien diferenciadas permite una separación de roles en los equipos de trabajo, asignando personal especializado a la construcción del controlador y modelo (requiere conocimientos de tecnología J2EE) y diseñadores gráficos (sin conocimientos J2EE) al desarrollo de la vista. Esto favorece un ahorro considerable ya que los costes de especialistas J2EE son más elevados y no tiene que aplicarse a todo el proyecto.

5. Conclusiones

En este trabajo se ha presentado una solución para acceder a Bases de Datos desde aplicaciones Web desarrolladas con J2EE. La solución propuesta se basa en patrones de diseño bien conocidos y probados, lo cual permite dotar a nuestras aplicaciones de una mayor robustez y fiabilidad, al mismo tiempo que facilita su mantenimiento. Entre los patrones presentados destaca el patrón Data Access Object, que permite independizar la lógica de la aplicación de la fuente de datos utilizada. Se presenta también un caso práctico, el portal de la Real Academia Gallega, que permite ilustrar el uso de los patrones presentados.

6. Referencias

[1] Hans Bersgsten. *JavaServer Pages*. O'Reilly & Asssociates, Inc. ISBN: 1-56592-746-X. USA, 2001.

[2] Floyd Marinescu. *EJB Design Patterns. Advanced Patterns, Processes, and Idioms*. John Wiley & Sons, Inc. ISBN: 0-471-20831-0. USA, 2002.

[3] Deepak Alur, John Crupi, Dan Malks.Core J2EE Patterns. Best Practices and Design Strategies. Prentice Hall. ISBN: 0-13-064884-1. USA, 2001.

[4] Nieves R. Brisaboa, Carlos Callón, Juán Ramón López, Ángeles S. Places and Goretti Sanmartín. Stemming Galician Texts. *Proceedings of the 9th International Symposium on String Processing and Information Retrieval, SPIRE, LNCS*, Vol. 9. Lisboa. Septiembre 2002.

[5] Fernando Bellas Permuy. Integración de Sistemas - Curso 2001-2002. http://www.tic.udc.es/~fbellas/teaching/is-2001-2002/index.html

[6] Core J2EE Pattern Catalog. http://java.sun.com/blueprints/corej2eepatterns/Patterns/

[7] TheServerSide.com: J2EE Patterns Repository. http://theserverside.com/patterns/index.jsp

[8] Java 2 Platform, Enterprise Edition (J2EE). http://java.sun.com/j2ee/

[9] JavaServer Pages. http://java.sun.com/products/jsp/

[10] Java Servlets. http://java.sun.com/products/servlet/index.html

[11] Java DataBase Connectivity. http://java.sun.com/products/jdbc/

[12] Enterprise Java Beans. http://java.sun.com/products/ejb/