Desarrollando Aplicaciones Empresariales Altamente Interactivas sobre J2EE

Emilio Ormeño (*)
(*) eormeno@iinfo.unsj.edu.ar
Instituto de Informática, F.C.E.F. y N.
Universidad Nacional de San Juan, Argentina
Av. José I. de la Roza y Meglioli, edificio Islas Malvinas.
Tel.: (54 264) 4234129

Sergio F. Ochoa (+)
(+) sochoa@dcc.uchile.cl
Departamento de Ciencias de la Computación
Universidad de Chile
Blanco Encalada 2120, Santiago, Chile.
Tel.: (56 2) 678-4362, Fax: (56 2) 689-5531

Resumen

Dentro de los desarrollos de software empresarial, uno de los principales actores continúan siendo las aplicaciones cliente/servidor construidas sobre J2EE. La estabilidad de esta tecnología y la capacidad de usar dichas aplicaciones en el escenario Internet/Intranet, usando tanto clientes java como browsers, les brinda ventajas que han sido ampliamente discutidas por la comunidad científica del área. Sin embargo, la implementación de aplicaciones cliente/servidor altamente interactivas, como por ejemplo un procesador de texto distribuido, continúa siendo todo un desafío cuando se usa J2EE. Esto se debe a la altísima carga de transacciones que se requiere para mantener coordinados al cliente y al servidor, lo cual afecta la performance, la escalabilidad y la capacidad de procesamiento de transacciones de la solución. Para paliar esta problemática, este artículo presenta una plataforma llamada EBCDS (Enterprise Business Component Development System), la cual está compuesta por componentes Java, que permiten construir aplicaciones clientes J2EE que combinan una alta interactividad con el usuario, sin dejar de lado los aspectos de performance, escalabilidad, seguridad y buen soporte para transacciones.

Palabras Clave: Framework de Componentes de Software, J2EE, Desarrollo de Aplicaciones Empresariales.

Evento: II Workshop de Ingeniería de Software y Bases de Datos.

1. Introducción

En la actualidad es indiscutible el rol que juega la plataforma J2EE (Java 2 Enterprise Edition) [12] en el desarrollo de aplicaciones empresariales. Esta plataforma propone patrones de diseño que se adaptan a la gran mayoría de los sistemas de información que una moderna organización requiere, facilitando así el desarrollo de muchas de las aplicaciones empresariales. Estas aplicaciones usualmente involucran componentes distribuidos, seguros, escalables y capaces de soportar una carga de transacciones importante. Aunque este soporte es ampliamente reconocido, la implementación de aplicaciones cliente/servidor altamente interactivas continúa siendo una limitación para J2EE. La implementación de ese tipo de aplicaciones, tales como un entorno de diseño gráfico o un procesador de texto, requiere que cada interacción con el usuario esté asociada a una o más interacciones entre las aplicaciones cliente y servidor. Esto es operativamente poco viable, debido a la sobrecarga de transacciones que esto implica y al ancho de banda que requiere. Aunque las empresas que promueven y apoyan el desarrollo de la plataforma J2EE son conscientes de esta limitación, las implementaciones tanto comerciales como open source continúan si mostrar

avance en este tópico. Tal es el caso de productos como IBM Websphere [21], JBoss [5], Oracle Application Server [6] y su entorno de desarrollo JDeveloper [7], entre otras. Por ese motivo, se puede decir que las aplicaciones empresariales altamente interactivas han quedado fuera de ésta plataforma. Por lo tanto, el equipo de desarrollo es quien debe proveer sus propios mecanismos de soporte.

Por otra parte, distintos tipos de las aplicaciones Web que hacen de front-end de servidores J2EE, también podrían ser utilizadas para implementar aplicaciones cliente altamente interactivas. Si bien el tipo y la calidad de los recursos que hoy pueden ser utilizados con este objetivo se han ido enriqueciendo con cada vez más tecnologías, tales como Applets [4], módulos Flash [10], Javascript [9], etc., éstos aun no son suficientes como para soportar escenarios de interacción de alto rendimiento. Por

Para reducir el impacto de esta problemática, una plataforma llamada EBCDS (Enterprise Business Component Development System), la cual está compuesta por componentes Java, que permiten construir aplicaciones clientes J2EE que combinan una alta interactividad con el usuario, sin dejar de lado los aspectos de performance, escalabilidad, seguridad y buen soporte para transacciones. También provee la capacidad de distribuir sus componentes, al proveer adaptadores en los que es posible conectar mecanismos de persistencia tales como EJB (Enterprise Java Beans) [8], o simplemente un sistema de archivos, en función de las necesidades de la organización.

En la siguiente sección se presenta el concepto de clientes pesados (o altamente interactivos) y su relevancia en determinados tipos de sistemas de información. En la sección 3 se explica la arquitectura de EBCDS y sus componentes principales. En la sección 4 se describen los trabajos relacionados, y finalmente, en la sección 5 se presentan las conclusiones y el trabajo a futuro.

2. J2EE y los Clientes Pesados

En este artículo se le llamará cliente pesado a aquellos módulos cliente, que demandan una alta tasa de interacciones con componentes del servidor. En la actualidad a la hora de desarrollar aplicaciones empresariales, es decir, aquellas que están compuestas por componentes escalables, seguros, capaces de operar usando transacciones, y que permiten el acceso a ellos en forma distribuida, las organizaciones usualmente optan por soportar sus aplicaciones sobre la plataforma J2EE, debido a los beneficios que ésta les aporta:

- Simplifica la compleja tarea de escribir código confiable, distribuido, escalable y con manejo de seguridad, al proveer un conjunto estándar de componentes de software reutilizables y adaptables [14].
- Provee una API estándar, la cual al ser utilizada por diferentes vendedores de software para mantener la compatibilidad entre ellos.
- Provee una clara separación entre la lógica de negocio, la persistencia de datos y despliegue de la aplicación (presentación). Esto promueve la modularización de las aplicaciones [13], la separación de intereses y la distribución de componentes.
- Promueve la separación en capas de la aplicación [15], lo cual mejora la reutilización y el desarrollo de diferentes partes de la aplicación, por parte de diferentes equipos de trabajo

(diseñadores gráficos, arquitectos de software, programadores, administradores de base de datos, etc).

La figura 1 muestra las diferentes capas involucradas en una aplicación J2EE y las tecnologías por ésta provista. En dicha figura se puede observar que el los clientes livianos (thin clients) acceden al servidor de aplicaciones y a los datos sólo a través un servidor Web.

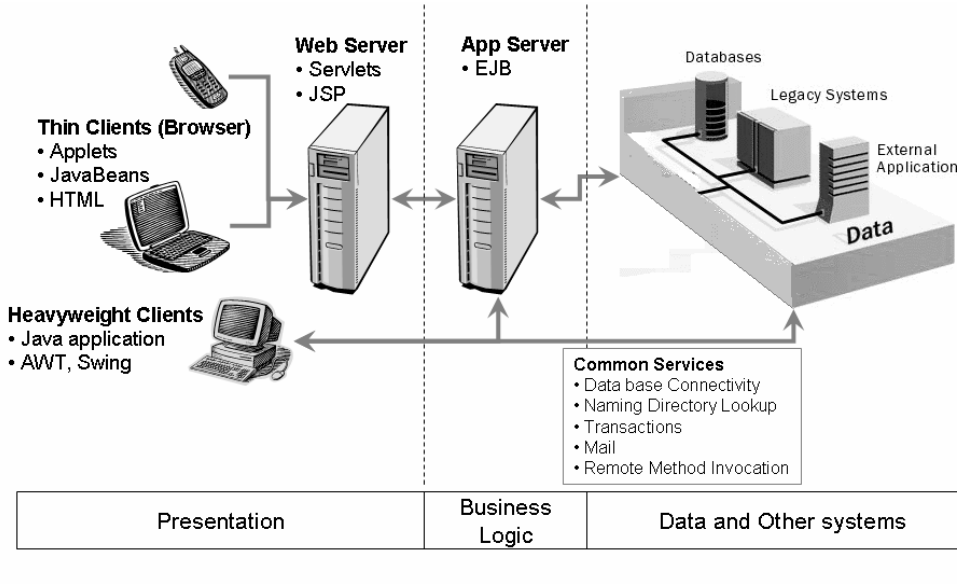

Figura 1. Arquitectura n-capas J2EE

Por otra parte, los clientes pesados (heavyweight clients) o aplicaciones Java comunes pueden acceder directamente al servidor de aplicaciones o a la base de datos, a través de las APIs provistas por la plataforma J2EE. En la medida que se reduzca la granularidad de las operaciones que viajan entre el cliente y el servidor, la sobrecarga de todos los elementos involucrados (incluyendo la red) aumentará en consecuencia.

Con el fin de proveer soluciones reutilizables a este problema de interacción, se han desarrollado una serie de patrones de diseño [1], entre los cuales se destaca el patrón Transfer Object [11, 16]. Este patrón propone la inclusión de un objeto de transferencia (transfer object) entre el cliente y el servidor, como una forma de disminuir la cantidad de transacciones entre éstos. Dicho objeto funciona como un capacitor, que agrupa transacciones del cliente y las replica al servidor en forma de bloque.

Al iniciar la interacción entre un cliente y un servidor de aplicaciones, uno o más objetos de transferencia son creados y alojados en la máquina donde ejecuta la aplicación cliente. Típicamente, esto objetos son copias de uno o más objetos del servidor que viajan hasta el cliente y sobre los cuales se realizan las transacciones, que luego retornan al servidor para actualizar los objetos reales.

Con el fin de ejemplificar este proceso, la figura 2a muestra un esquema típico en donde la aplicación cliente realiza invocaciones directas al servidor. Por otra parte, la figura 2b muestra el mismo cliente realizando transacciones contra un servidor, pero utilizando un objeto de transferencia como intermediario. Debido a que el objeto de transferencia está alojado en el cliente, las interacciones entre éste y la aplicación cliente no utilizan la red, por lo tanto no sobrecargan el

medio de comunicación y ayudan a mantener la performance del sistema. Además, este esquema de trabajo, hace a las aplicaciones más escalables debido a que se puede jugar con la granularidad de los bloques de transacciones que cada objeto de transferencia replica al servidor. De esta manera, el número de invocaciones al servidor se podría reducir significativamente.

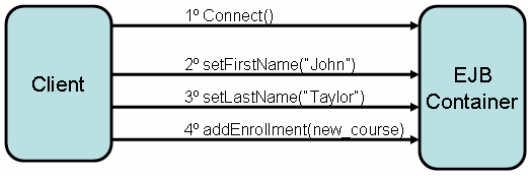

Figura 2a. Invocaciones directas entre el cliente y el servidor

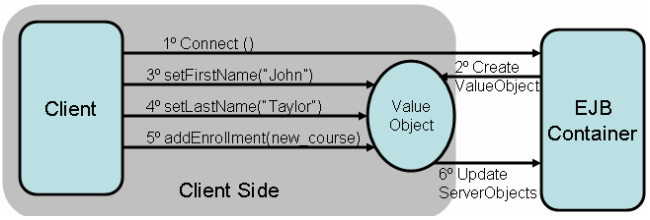

Figura 2b. Introducción de un objeto de transferencia

La plataforma EBCDS implementa el patrón Transfer Object y brinda además un manejo de transacciones transparente para el desarrollador. Esta plataforma ha sido implementada como una extensión de J2EE, como el objetivo de ayudar a superar las limitaciones antes mencionadas. La siguiente sección muestra en mayor detalle los diferentes componentes de EBCDS.

3. Arquitectura de EBCDS

La plataforma EBCDS (Enterprise Business Components Development System) motiva al equipo de desarrollo a mantener una clara separación funcional, a través de clases de *Modelo*, *Vista* y *Servicio* (transacciones). Esto facilita la separación de intereses en productos de software y ayuda a modularizar las aplicaciones.

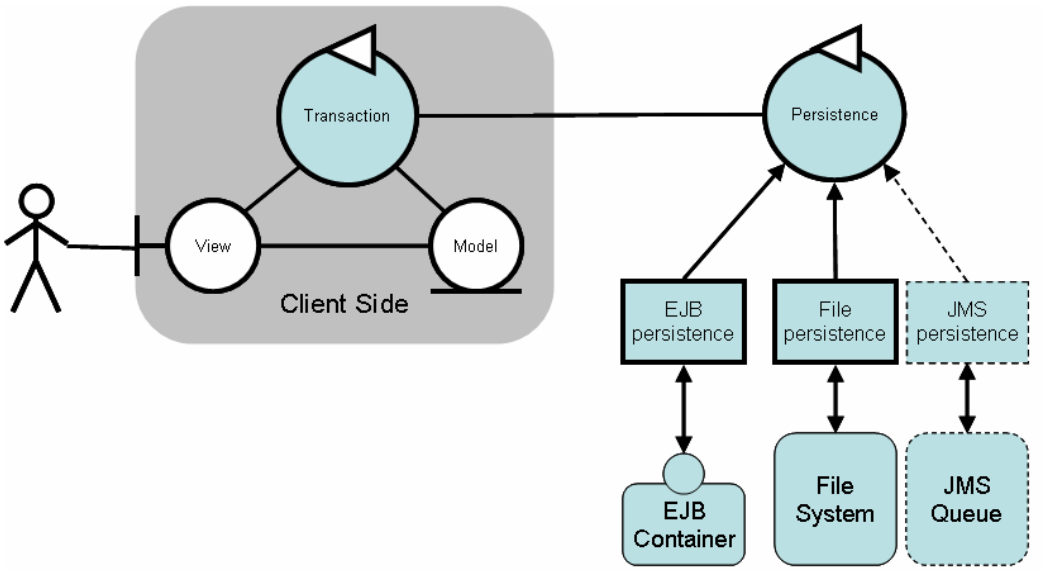

Figura 3. Arquitectura de EBCDS

Cabe hacer notar que los componentes de la plataforma ejecutan (y embeben la lógica de negocio) del lado del cliente, lo cual les permite especializarse para diversos tipos de aplicaciones

empresariales. Especialmente, aquellas con interfaces gráficas altamente interactivas, tales como aplicaciones colaborativas sincrónicas y sistemas distribuidos.

La figura 3 muestra la arquitectura de la versión actual de EBCDS. Allí cada componente que ejecuta del lado del cliente y está soportado por un conjunto de clases y archivos de configuración que viajan desde el servidor. Del otro lado operan los componentes que manejan la persistencia de datos y aquellos que sirven a los componentes EBCDS que residen en los clientes. Estos componentes que ejecutan en el servidor son una especie de interfaz, que independiza a los clientes de las implementaciones particulares de los módulos servidores. A continuación se detallan los elementos más importantes de la plataforma.

3.1. Componentes de Modelo

Los *componentes de modelo* de EBCDS son los encargados de encapsular y proveer la funcionalidad básica para el acceso a los atributos y definición de métodos del negocio, de las clases de modelo específicas de una aplicación. La figura 4 muestra un diagrama simplificado de clases que corresponden al componente de modelo ofrecido por EBCDS. A fin de aclarar cómo se trabaja con estos componentes, se presentará como ejemplo una solución EBCDS a un problema simple. El problema consiste en poder actualizar dos entidades que están directamente vinculadas. Estas entidades son `Employee` y `Task` (Figura 5), donde cada empleado puede tener asociadas cero o más tareas.

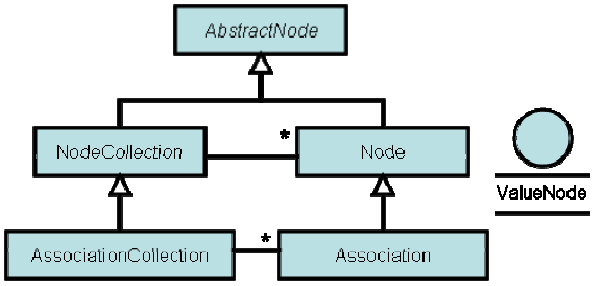

Figura 4. Clases del componente de modelo de EBCDS

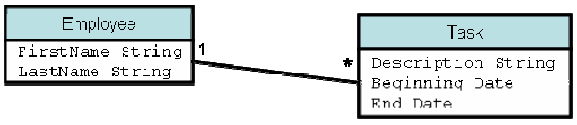

Figura 5. Entidades a implementar con EBCDS

La figura 5, muestra el problema planteado, mientras que la figura 6 muestra un diagrama con una posible solución para implementar la lógica del negocio de estas dos entidades. Esto puede hacerse a través de extensiones a la clase `Node`, que es la clase raíz de toda entidad, incluida la relación entre ellas a través de una clase `NodeCollection`, la cual es la raíz de las colecciones.

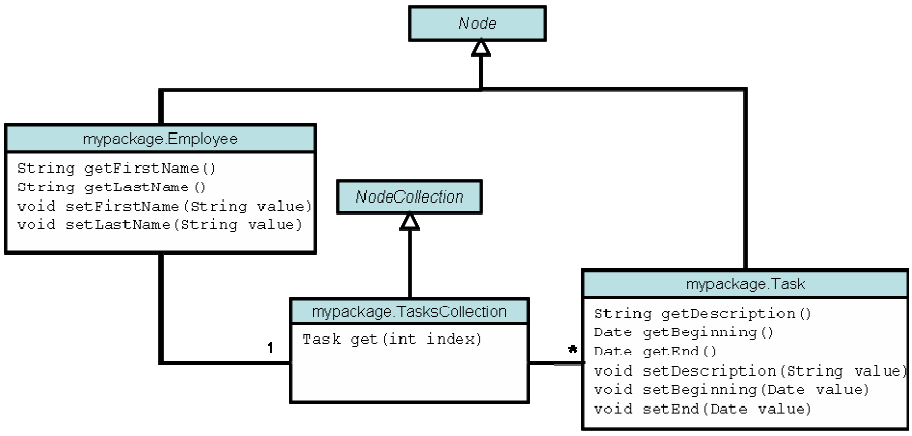

Figura 6. Implementación del problema utilizando las clases de modelo de EBCDS

Las clases de modelo interactúan con las de servicio a fin de lograr dos funciones básicas: acceso a transacciones y acceso a los datos (persistencia). Por esa razón EBCDS provee dos tipos de componentes para apoyar esta funcionalidad, los cuales se presentan en secciones 3.2 y 3.3. Estos componentes de servicio son los únicos componentes que utiliza EBCDS para sincronizar la funcionalidad del cliente con la del servidor.

3.2. Componente de Servicio de Transacciones

Las clases que componen el servicio de transacción reciben notificaciones de las clases de modelo, respecto a las modificaciones en sus atributos, o bien respecto a altas y bajas en las colecciones. La figura 7, muestra el diagrama de clases del *servicio de transacciones*.

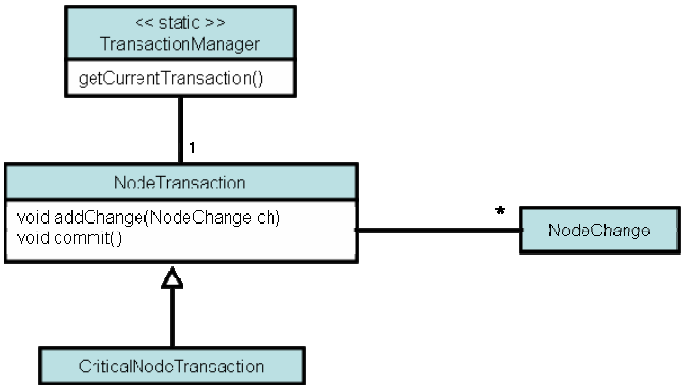

Figura 7. Clases del servicio de transacciones de la plataforma EBCDS

Las clases de servicio modifican las clases de entidad, las cuales residen en el servidor. Cada modificación realizada a una clase de entidad es capturada por el servicio de transacciones y son enviadas al *servicio de persistencia* (que se explica en la sección 3.3) cuando desde el entorno se invoca el método `commit()`.

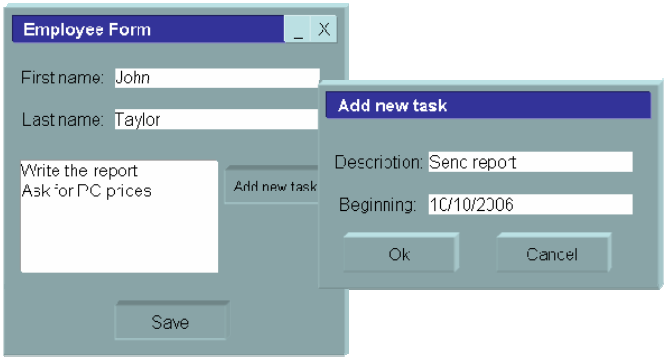

Figura 8. Interfaz de usuario del alta y modificación de las tareas de un empleado

A fin de ejemplificar el funcionamiento del *servicio de transacciones*, se toma como ejemplo la captura de modificaciones realizadas ante un alta, y la modificación de una tarea perteneciente al conjunto de tareas de un empleado. La figura 8 muestra la interfaz de usuario involucrada, mientras que la figura 9 muestra un diagrama de secuencia que sintetiza el funcionamiento del servicio de transacciones para el ejemplo.

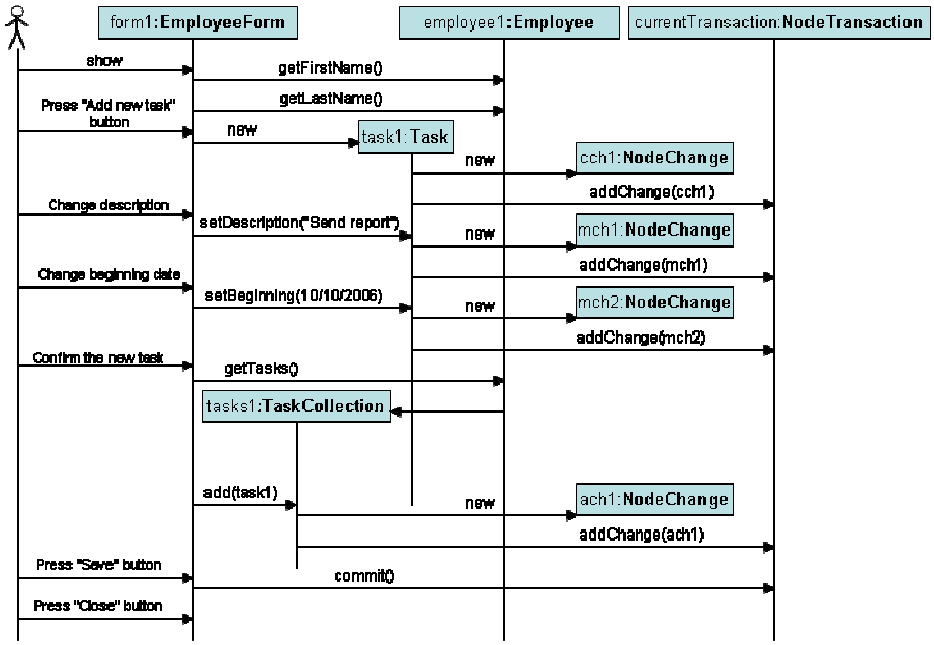

Figura 9. Diagrama de secuencia de un alta y modificación de una tarea

3.3. Componente de Servicio de Persistencia

El componente de *servicio de persistencia* es el encargado de asegurar la persistencia de las entidades y de sus actualizaciones en algún medio físico o de red. Este componente provee además una interfaz común para facilitar la implementación de diferentes mecanismos de persistencia, tales como: EJB, sistema de archivos y colas de mensajes. La figura 10 muestra el diagrama de clases del servicio de persistencia de EBCDS.

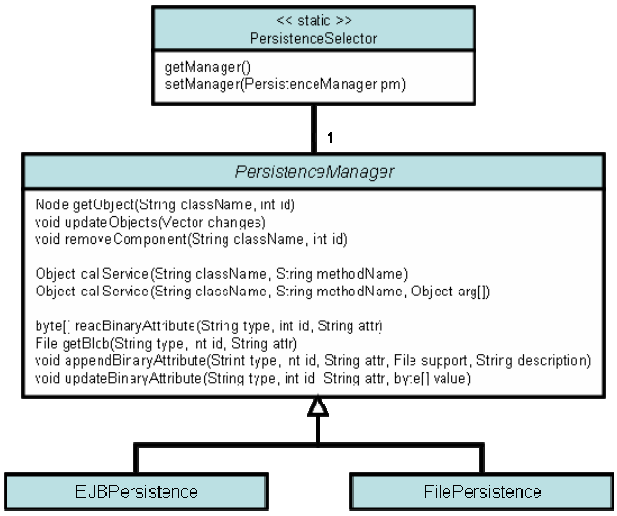

Figura 10. Diagrama de clases del servicio de persistencia de EBCDS

Continuando con el ejemplo de empleados y tareas introducido previamente, la figura 11 muestra cómo la interfaz de usuario accede al conjunto de entidades de empleado, al realizar una solicitud (commit) al servicio de persistencia. El último nivel del diagrama de secuencia corresponde a la interfaz de acceso a datos, la cual implementa el patrón Fachada (*Facade)* [1].

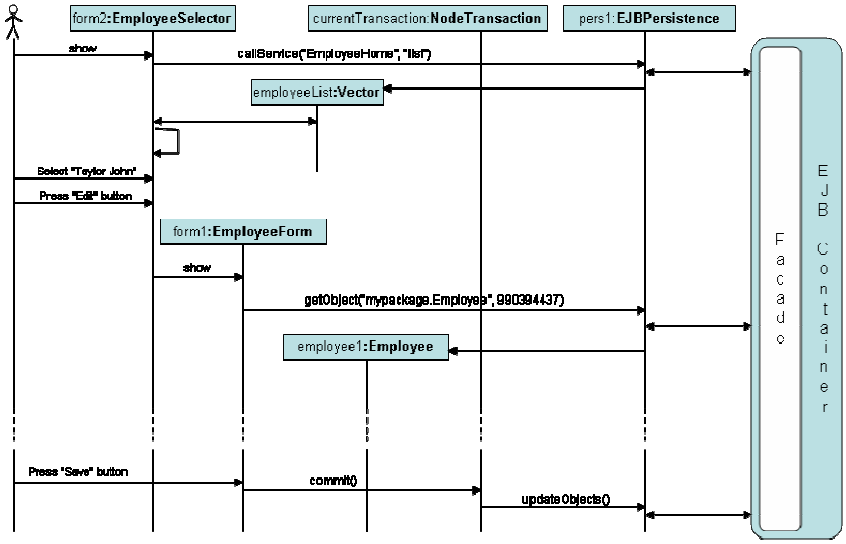

Figura 11. Diagrama de secuencia para la obtención del conjunto de empleados utilizando el servicio de persistencia de EBCDS

3.4. Componentes de Vista

Los componentes de vista de la plataforma EBCDS, están representados a través de un framework llamado Wwing [3], el cual agrupa y relaciona un conjunto de componentes Web visuales [2]. Wwing posee una API que simplifica la construcción de la capa de presentación Web de aplicaciones J2EE. De esa manera motiva al desarrollador a programar sus clases bajo la forma de una típica interfaz orientada a eventos; o sea, a través del ensamblaje de componentes visuales

dentro de un contenedor y programando las acciones a realizar ante los eventos producidos por el usuario. Todo ello, sin tener que lidiar con las complejidades de HTML, JavaScript o con los problemas relacionados al estilo de presentación. Si bien esta API es una parte muy importante de EBCDS, su inclusión excede el alcance de este artículo, debido a que constituye en sí mismo otra plataforma y otro conjunto de especificaciones relacionadas.

Como muestra de la utilidad de EBCDS se desarrolló un ambiente de diseño, construcción y distribución de courseware sobre la base de EBCDS. Este ambiente llamado CDS (Course Development System) [19, 20], sigue un paradigma de construcción en base a componentes didácticos reutilizables (texto, imágenes, video, foros de discusión, chats, y actividades personales y grupales). Los courseware creados utilizando CDS se emplean para apoyar el proceso de enseñanza-aprendizaje en el Departamento de Ciencias de la Computación de la Universidad de Chile. Dichos courseware implementan todos sus componentes didácticos utilizando EBCDS, debido a que existe una elevada interacción entre los usuarios (profesores, ayudantes y alumnos), tanto durante la etapa de construcción como durante su puesta en producción.

4. Trabajos Relacionados

Entre las plataformas estudiadas y que guardan relación con la presente propuesta, quizás la que más se acerca al concepto de soporte para la interactividad de aplicaciones cliente, es Oracle ADF (Application Development Framework) [17]. Este framework de componentes multicapa facilita el desarrollo de aplicaciones J2EE, además de la conectividad con diversas tecnologías de persistencia. La figura 12 muestra la arquitectura general de este producto.

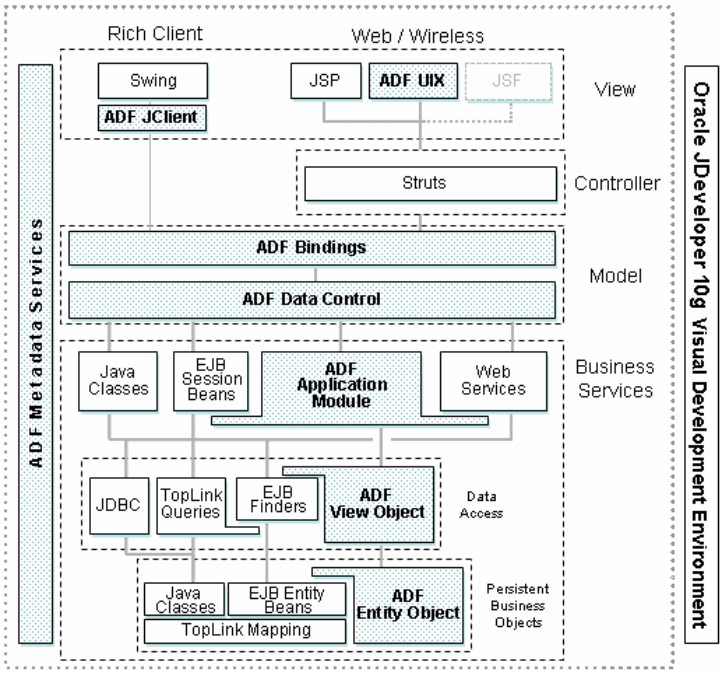

Figura 12. Arquitectura de Oracle ADF

La construcción de aplicaciones con esta plataforma requiere del entorno de desarrollo JDeveloper [7], el cual automatiza la mayoría de las tareas involucradas en una compleja arquitectura, como la de ADF. Como se puede apreciar en la figura 11, la arquitectura del producto incluye clientes pesados (rich clients), los cuales se desarrollan sobre JClient [18]. Este componente de ADF consiste en un conjunto de APIs que facilitan el acceso a las capas inferiores de la arquitectura.

Si bien ADF es una excelente herramienta de desarrollo que facilita en gran medida la implementación del patrón Transfer Object, no se adapta a nuestras necesidades dado que nuestro interés radica en un limitado espectro de clientes pesados, en donde la interacción con el usuario es muy alta. Tal es el caso de un diseñador gráfico, un procesador de texto, un sistema colaborativo o una herramienta de e-learning como CDS (Course Development System) [19, 20]. Por otro lado, todo el entorno JDeveloper está orientado a lograr interfaces de usuario estándares, que si bien están muy bien representadas para la gran mayoría de las situaciones, no se adaptan al tipo de aplicaciones antes mencionadas.

5. Conclusiones y Trabajo Futuro

La plataforma EBCDS es un principio de solución para los problemas que presentan determinados sistemas de información empresariales, en los que la interacción con el usuario es muy elevada. Si bien este tipo de sistemas encajan dentro de una muy estrecha franja de aplicaciones, existe un mercado importante. Muchas organizaciones han optado por la plataforma J2EE debido a las ventajas que esta les provee, razón por la cual EBCDS no representa mas que una extensión, que permite extender dichas ventajas a escenarios que no están considerados en forma nativa en la plataforma J2EE.

El hecho de que EBCDS implemente el patrón Transfer Object, hace que la plataforma herede también la principal problemática que este patrón posee: la concurrencia. Debido a que la interacción entre el cliente y el modelo se realiza con una copia local de uno o más componentes alojados en el servidor, es altamente probable que un acceso concurrente pueda producir versiones incompatibles. Para solucionar este problema un requisito en el que se está trabajando actualmente es en la posibilidad de bloquear componentes que pueden ser accedidos en forma simultánea.

Otra de las tareas en la que se está trabajando en pos de mejorar la plataforma, es en la medición de la satisfacción de los desarrolladores que emplean EBCDS. Para ello se están utilizando cuestionarios confiables [22], con alumnos de la Universidad Nacional de San Juan y del curso de CC51A-Ingeniería de Software de la Universidad de Chile.

Agradecimientos

Este trabajo ha sido parcialmente financiado por las iniciativas Programa Incentivos, Ministerio de Educación de la Nación/Universidad Nacional de San Juan, Nro. 21/E345 (Argentina), Fondecyt No. 1030959 (Chile) y por el Chile-Korea IT Cooperation Center (Chile).

Referencias

[1] Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object Oriented Software. Addison-Wesley. (1994).

[2] Rosenfeld L., Morville P. Information Architecture for the World Wide Web. O'Reilly & Associates, (1998).

[3] Ormeño, E., Ochoa, S., Zapata, S. Swing: Una Plataforma de Componentes Visuales Reutilizables para Aplicaciones Web. IV Workshop Chileno en Ingeniería de Software. Arica, Chile. (2004).

[4] Sun Microsystems. JavaSoft, Applets, http://java.sun.com/applets/. Last visit: June, 2005.

[5] Jboss Inc. http://www.jboss.org. Last visit: June, 2005.

[6] Oracle Corp. Oracle Application Server 10g, http://www.oracle.com/technology/products/ias. Last visit: June, 2005.

[7] Oracle Corp. Oracle JDeveloper 10g, http://www.oracle.com/technology/products/jdev. Last visit: June, 2005.

[8] Sun Microsystem. Enterprise JavaBeans Specification, Version 2.0. Pág. 34.

[09] Netscape. Netscape Deveage: Core JavaScript 1.5, http://devedge.netscape.com/library/manuals/2000/javascript/1.5/guide. Last visit: July, 2005.

[10] Macromedia. Flash MX 2004, http://www.macromedia.com/software/flash/. Last visit: August, 2005.

[11] Oracle Corp. Core J2EE Patterns: Best Practices and Design Strategies, http://www.oracle.com/technology/products/jdev/collateral/papers/10g/adftoystore/readme.html#core-patterns. Last visit: June, 2005.

[12] Sun Microsystems. Java 2 Platform Enterprise Edition http://www.javasoft.com/j2ee. Last visit: July, 2005.

[13] Johnson, R., Roberts, D. Frameworks = (Components + Patterns). Communications of the ACM, Vol. 40, No. 10, October, (1997) 39-42.

[14] Roberts, D., Johnson, R. Evolve Frameworks into Domain-Specific Languages. Proceedings of the 3rd Patterns Languages of Programming Conference, PloP'96. Illinois, EEUU, September, (1996).

[15] Estublier, J., Favre, J. Component Models and Component Technology. Book Chapter in Building Reliable Component-Based Systems, I. Crnkovic, M. Larsson editors. Archtech House Publishers, (2002).

[16] Crawford, W., Kaplan, J. J2EE Design Pattern. O'Reilly & Associates. (2003).

[17] Oracle Corp. Oracle Application Development Framework: ADF, http://www.oracle.com/technology/products/jdev/htdocs/905/adffaq_otn.html. Last visit: June, 2005.

[18] Oracle Corp. ADF JClient Facilities. http://www.oracle.com/technology/products/jdev/collateral/papers/9.0.5.0/jclient10g.pd

f. Last visit: June, 2005.

[19] Ormeño, E., Ochoa, S. CDS (Courseware Development System): Un Ambiente de Desarrollo de Courseware. Memorias del VII Congreso Argentino de Ciencias de la Computación (CACIC 2001). El Calafate, Santa Cruz, Argentina. Oct. 16-20, 2001.

[20] Ochoa, S., Ormeño, E., Pino, J. Reusing Courseware Components. Proceedings of the XIV International Conference on Software Engineering and Knowledge Engineering (SEKE'02). ACM Press. Ischia, Italy. July 15-19, (2002). 549-556.

[21] IBM. Websphere Journal. http://websphere.sys-con.com. Last visit: July, 2005.

[22] Zapata, S., Lund, M., Ochoa, S. Una Herramienta para la Medición Continua de la Satisfacción de Usuarios de Software. II Workshop en Ingeniería de Software, realizado en el marco de la Jornadas Chilenas de Computación (JCC 2002). Copiapó, Chile, 2002.