Potential users evaluated the application and the data gathered are being used in ongoing works. It is worth noticing that the three versions of *No Risk Planning* agenda have been developed during the last year. Additionally, they will be important in the next framework cycle of evolutionary model: usability and navigability characteristics are the main focus for improving the application. Another result of *No Risk Planning* project was its well-succeeded integration with the *CoWeb* – a Collaborative Web implementation [1]. The integration has been very useful as a support to the learning approach that exists in *CoWeb*, since it allows people to manage their tasks schedule. In this case, tasks are related to the collaborative editing of lecture web pages. For instance, by using the *No Risk Planning*, a teacher can arrange and announce the date the students should turn in homework, and he can also state the related topics and important notes in a CoWeb page. Consequently, students can easily access both the established date and notes related to the work.

Nowadays, *No Risk Planning* has 44 common users, 46 lecturers, 81 disciplines and 29 groups registered. The perspective of the increased amount of all users has given us motivation to keep on the research of how the process can attend to their demand, and to see alternative technical solutions (for instance, by means of frameworks or design patterns) to improve its evolution.

References

[1] Arruda Jr., C.R.E.; Pimentel, M.G.C. Projeto e Implementação de um Sistema Colaborativo de Edição. *Revista Eletrônica de Iniciação Científica da Sociedade Brasileira de Computação (REIC-SBC)*, Ano 1, Novembro 2001. (in Portuguese)

[2] Athula, G.; Murugesan, S. Web Engineering: An Introduction. *IEEE Multimedia*, v.18, Issue 1, p.14-18, 2001.

[3] Costagliola, G.; Ferrucci, F.; Francese, R. Web Engineering: Models and Methodologies for the Design Hypermedia Applications. *Handbook of Software Engineering & Knowledge Engineering, Emerging Technologies*, v.2, p. 181-199, 2002.

[4] eXtropia home page. [Online]. http://www.extropia.com/scripts/calendar.html [21/06/2002]

[5] ISO/IEC 14598-5, International Standard. Information Technology - Software product evaluation - Part 5: Process for evaluators (1997).

[6] Lee, H.. Your Time and my time: a temporal approach to groupware calendar systems. *Information & Management, Elsevier*, 40, 159-164, 2003.

[7] Olsina, L., et al. Specifying Quality Characteristics and Attributes for Web Sites. *First ICSE Workshop on Web Engineering, ACM*, Los Angeles, 1999.

[8] Pimentel, M. G. C., Abowd, G. D., Brotherton, J. Requirements for Capture-based Support to Learning (Accepted in May 2002). *Communications of the ACM*, 2002.

[9] Paiva, D. M. B.; Sanches, R; Fortes, R. P. M. Melhoria de processo de software no contexto do desenvolvimento acadêmico de uma agenda na web. In: 14°. Congresso Internacional de Tecnologia de Software (CITS 2003). *Proceedings*. Curitiba-PR. June, 2003. p. 56-68. (in Portuguese)

[10] Pressman, R. S. *Software Engineering*. 5th edition, McGraw-Hill, 2001.

[11] Project Place home page [Online]. http://projectplace.com/ [21/06/2002]

[12] Ribeiro,T.M.; Fortes, R.P.M.; Freire, A.P. - "Documentação do Software Agenda "*No Risk Planning*" São Carlos-SP, ICMC-USP, Brasil, 2003. 69p. (Relatórios Técnicos do ICMC, 182) (in Portuguese)

[13] Web Organizer home page. [Online]. http://weborganizer.sourceforge.net/ [21/06/2002]

[14] Chronoss. [Online]. http://chronoss.sourceforge.net [18/12/2002]

[15] Calendar Publisher II. [Online]. http://www.upoint.net/calendar2/index.shtml [18/12/2003]

[16] Web Absence. [Online]. http://www.uni-wissen.de/absence/ [18/12/2003]

[17] Calcium. [Online]. http://www.brownbearsw.com/calcium/. [18/12/2003]

[18] Alltasks. [Online]. http://freshmeat.net/projects/alltasks/. [18/12/2003]

[19] CollabOffice. [Online]. http://collaboffice.sourceforge.net/. [18/12/2003]

[20] WebCalendar. [Online]. http://webcalendar.sourceforge.net/. [18/12/2003]

[21] Zope Group Calendar [Online] .http://sourceforge.net/projects/zgc/. [18/12/2003].

Generación Automática de Aplicaciones basadas en Componentes a partir de Bases de Datos

Ignacio García, Macario Polo, Mario Piattini

Grupo Alarcos
Escuela Superior de Informática
Universidad de Castilla la Mancha
Paseo de la Universidad, 4
13071-Ciudad Real
igarcia@proyectos.inf-cr.uclm.es
Macario.Polo@uclm.es
Mario.Piattini@uclm.es

Resumen

Este artículo presenta una propuesta para generar aplicaciones basadas en componentes distribuidos a partir de bases de datos. En particular, se describe el método de reingeniería que aplicamos a la base de datos para obtener, finalmente, la capa de componentes de la aplicación. Inicialmente se obtiene, mediante Ingeniería Inversa, un modelo de clases que representa el esquema conceptual de la base de datos, que puede ser modificado por el ingeniero de software mediante la adición de clases, operaciones, etc.; entonces, se aplica un proceso de generación automática de código, obteniéndose un componente por cada clase seleccionada.

1. Introducción

Una de las herramientas más potentes que ofrece la ingeniería del software para aprovechar o aumentar la funcionalidad de los sistemas heredados es la *reingeniería* (Figura 1). De acuerdo con [3], la reingeniería está compuesta a su vez por otros dos procesos, la ingeniería inversa y la ingeniería directa.

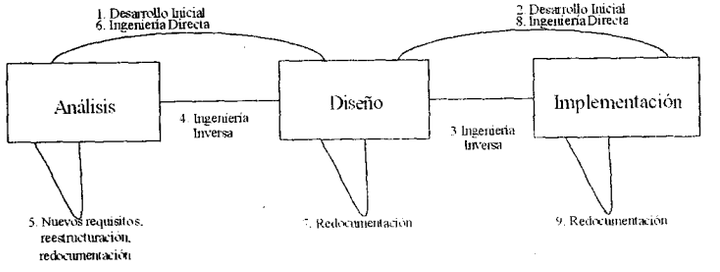


Figura 1. Modelo de reingeniería simplificado

Según [3], la ingeniería inversa es "el proceso de construir especificaciones formales abstractas del código fuente de un sistema heredado (*legacy*), de manera que estas especificaciones puedan ser utilizadas para construir una nueva implementación del sistema usando ingeniería directa".

La ingeniería inversa cobra gran importancia dentro del proceso de mantenimiento, ya que de ella se obtienen representaciones del sistema a un nivel de abstracción más alto que el original [23][1], lo que aumenta la facilidad de comprensión del sistema original, su mantenibilidad, etc. [3]. La ingeniería inversa suele aplicarse a sistemas antiguos, de los que queremos obtener modelos conceptuales y arquitectónicos de más alto nivel. Estos modelos se modifican y mejoran (en la Figura 1 correspondería a la etapa etiquetada *reestructuración*) y se utilizan como entrada para la siguiente fase de ingeniería directa.

Al aplicar ingeniería inversa a una base de datos para recuperar su correspondiente esquema conceptual, lo más habitual es obtener un esquema entidad-interrelación [24] [25] [26] [27], si bien de otras propuestas se obtiene una representación orientada a objetos, normalmente en forma de clases [14]. En particular, la utilización de diagramas de clases para representar el esquema de la BD supone, desde el punto de vista de la reingeniería, la posibilidad de aprovechar las características del paradigma orientado a objetos no sólo para la posterior reconstrucción de la base de datos, sino también para la construcción de aplicaciones capaces de manipular dicha base de datos [3].

En nuestro caso, procesaremos los diagramas de clases obtenidos de la ingeniería inversa para generar aplicaciones multicapa cuya capa de dominio estará compuesta de componentes, en principio tipo EJB[7] [15], que son componentes desarrollados mediante el lenguaje de programación Java [6]. En este artículo presentamos la parte del sistema encargada de realizar la ingeniería inversa y el método seguido para la generación de la capa de componentes.

El método dispone de un fuerte sustrato matemático con el que describimos de manera formal: (1) el metamodelo utilizado para representar los elementos de una base de datos que resultan de interés para nuestro propósito; (2) el metamodelo mediante el cual describiremos la estructura genérica de un EJB [9] y la capa de EJBs de la aplicación; y (3) una función para realizar la transformación entre metamodelos.

Este artículo esta organizado de la siguiente manera: en la sección 2, mostramos algunas tecnologías relacionadas con el software distribuido; en la sección 3 hacemos una breve introducción acerca de los *Enterprise Java Beans*, donde mostramos algunas de sus características; en la sección 4, pretendemos mostrar una visión formalizada del problema y de los conjuntos de elementos que intervienen en la resolución del mismo; en la sección 5 hacemos unos breves comentarios sobre la parte no automática del proceso; en las secciones 6 y 7 ofrecemos algunas conclusiones y algunas líneas de trabajo que podrían seguirse a partir de este trabajo.

2. Tecnologías de componentes

Las aplicaciones que utilizan arquitecturas de componentes distribuidos son relativamente recientes. No obstante, existen distintas propuestas interesantes, que de alguna forma intentan solucionar el mismo problema, que es ofrecer soporte a las aplicaciones que requieran ejecutar alguna funcionalidad a través de la red.

Microsoft, hace algún tiempo, tomo parte en esta joven tendencia. Su particular propuesta o contribución consistió en la "*extensión*" de una arquitectura de componentes propietaria. Esta arquitectura de componentes de Microsoft era COM [10] (*Component Object Model*), que fue extendida a DCOM [5] (*Microsoft Distributed COM*). Esta infraestructura permite a una aplicación invocar componentes del Modelo de objetos componentes instalados en otro servidor. A pesar de haberse portado a algunas plataformas, DCOM nunca ha conseguido amplia aceptación en dichas plataformas, por lo que rara vez se utiliza para facilitar la comunicación entre computadores con Windows y sin Windows [2].

Por otro lado, y también basado en el lenguaje de programación Java, tenemos la tecnología Java RMI [11] (*Remote Method Invocation*). Java RMI tiene una filosofía similar a la ya clásica comunicación estilo RPC [12]. En RMI, vamos a disponer de una serie de objetos localizados en un servidor conectado a la red.

Mediante una previa localización del objeto, podremos ejecutar los métodos públicos del mismo esperando una respuesta de esta acción. De manera muy simplificada, de los objetos que existan en el servidor, conoceremos una interfaz, es decir, conoceremos sus operaciones públicas o servicios, y a través de esta interfaz nos comunicaremos con el objeto.

No hace mucho surgió una nueva arquitectura de componentes distribuidos. Su nombre, muy descriptivo, nos da una idea de que son y para que sirven. Esta tecnología recibe el nombre de *Servicios Web* [13]. Un Servicio Web es un componente que esta ejecutándose en un servidor, exponiendo una interfaz, mediante la cual podemos invocar sus servicios para realizar una determinada actividad. Los Servicios Web intentan solucionar algunos de los puntos flacos de las tecnologías, como por ejemplo la interoperabilidad entre plataformas, interfaces fuertemente tipadas, uso de los estándares de Internet existentes, soporte para cualquier lenguaje, etc [2]. Los Servicios Web se apoyan sobra un conjunto de protocolos, que son: UDDI y DISCO para su búsqueda y descubrimiento, WSDL para la descripción, SOAP para d formato de los mensajes, y sobre todos estos XML para permitir la codificación de toda la información. Además, como novedad respecto a otras arquitecturas, los Servicios Web no están ligados a ningún protocolo de transporte específico, tan sólo requiere de este que sea capaz de transmitir caracteres, que es lo que son al fin y al cabo los documentos en XML.

Por último, comentaremos algunos aspectos introductorios sobre otra arquitectura de componentes, CORBA [4]. CORBA es un estándar creado por OMG a principios de los 90. Consiste en un sistema de objetos distribuidos multiproceso, multilenguaje y que ofrece un acceso a la red totalmente transparente. CORBA ofrece independencia multilenguaje, ofreciendo la posibilidad de realizar un desarrollo modular en el que cada parte puede ser programada con el lenguaje que más interese [22]. Mediante un acceso totalmente transparente a la red, podemos hacer uso de objetos remotos, es decir, que se están ejecutando en servidores conectados a la red.

3. Características de los componentes Enterprise Java Beans

Por término general, los componentes tienen dos partes básicas [14]:

- *Interfaz*. Define los servicios que proporciona el componente (Conjunto de métodos y parámetros).
- *Implementación*. Incluye la funcionalidad que aporta el servicio.

Los *Enterprise Java Beans* [9] proporcionan un marco de trabajo independiente de la plataforma para el desarrollo de componentes.

Los EJB son componentes que se ejecutan del lado del servidor. Esta característica les confiere la propiedad de dar soporte a operaciones concurrentes de forma fiable y eficiente. Dentro de la arquitectura de componentes distribuidos EJB, existen varios tipos de Beans, estos son los *"Session Bean"*, *"Entity Bean"* y *"Message Driven Bean"*. Los *Session Bean* simplemente realizan una tarea para un cliente sin utilizar persistencia, los *Message Driven Bean* procesan mensajes de forma síncrona, y los *Entity Bean*, que son los que más interesantes para nuestro propósito, representan una entidad de negocio almacenada que existe de forma persistente.

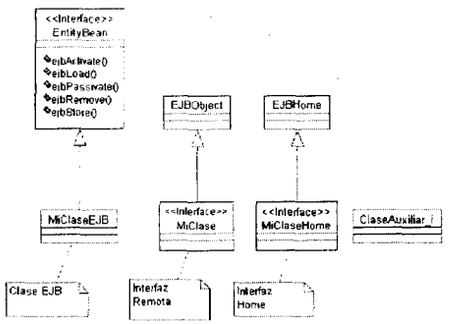


Figura 2. Clases de un Entity Bean.

Teniendo en cuenta que los EJB que vamos a utilizar básicamente son los *Entity Bean*, vamos a describir brevemente su estructura para comprender como funcionan. Según la **Figura 2**, un EJB, se compone de los siguientes elementos:

- *Interfaz Home*: Define los métodos que permiten a un cliente encontrar y crear un *Entity Bean*. Cuenta con los métodos *create* y *findByPrimaryKey*, que permiten al cliente crear una nueva instancia del EJB o encontrar una ya existente mediante la clave primaria. En esta interfaz también se definen los llamados métodos *Finder*, que nos permiten recuperar una interfaz remota del EJB o una colección de ellas, mediante las cuales ya podemos invocar las llamadas que necesitemos realizar.

- *Interfaz Remota*: Esta interfaz define los métodos mediante los cuales el cliente interactúa con el EJB. Los métodos aquí incluidos, generalmente, permiten manipular la información de la base de datos que representa el EJB, bien modificándola, realizando alguna operación de computo, etc. Una referencia a esta interfaz es lo que se devuelve al cliente cuando este ejecuta una operación *create*, *findByPrimaryKey* o alguno de los métodos *Finder*.

- *Enterprise Bean*: Es una clase que implementa la interfaz *EntityBean*. Esta clase tiene como propiedades a los atributos de la tabla de la base de datos a la que representa. En esta clase, se implementan los métodos que permiten gestionar el ciclo de vida del EJB, los métodos *create*, los métodos *findByPrimaryKey*, los métodos *Finder*, etc.

- *Clases Auxiliares*: Estas son otras clases que el EJB necesite utilizar para desempeñar sus operaciones, como podría ser, por ejemplo una clase que implementara algunas funciones matemáticas. Un ejemplo de esto, puede ser el que resulta de usar una clase que nos de soporte para realizar los accesos a la base de datos (Figura 3).

Figura 3. Clase Broker que soporta el acceso a la base de datos.

Además, al utilizar EJB, podemos seguir desarrollando nuestras aplicaciones siguiendo eficientes arquitecturas multicapa [16], mediante las cuales, encapsulamos el software en distintas capas verticales en

función del rol que juegue dentro de nuestro sistema. Como se puede ver en la Figura 4, los componentes distribuidos se encapsularían o situarían dentro de la capa de dominio:

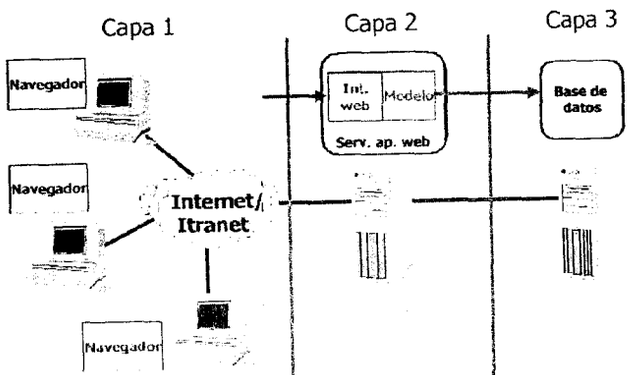


Figura 4. Integración de los EJB dentro de la arquitectura multicapa

Alguna de las tecnologías asociadas a los EJB son *JDBC* [17], que consiste en una API para el acceso a bases de datos relacionales, y *J2EE* [18], que son un conjunto de APIs que nos permiten desarrollar, por ejemplo, *Servlets*, páginas *JSP* (mediante *J2EE*, podemos realizar de manera sencilla interfaces gráficas para aplicaciones WEB), así como aplicaciones para manipular documentos *XML* [19] (que es un medio excelente para el intercambio de información entre aplicaciones heterogéneas), configurar aplicaciones, generar interfaces de páginas web (esto junto con hojas de estilo *XSL* [20]), mediante *JAXP* o *DOM* [21].

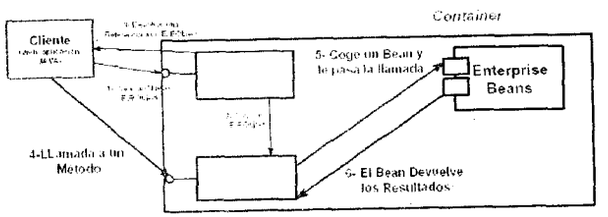


Figura 5. Interacción entre el cliente y el EJB

Por último y para terminar esta sección, en la Figura 5 podemos ver la interacción del cliente con el EJB. El cliente, mediante la *Interfaz Home*, recupera o crea una instancia del objeto, mediante la cual realiza las operaciones que necesita realizar el EJB.

4. Formalización del problema

En esta sección se describen los metamodelos correspondientes a la base de datos y a la capa de la aplicación formada por los componentes distribuidos.

El conjunto de componentes que generemos dependerá de la estructura de la base de datos a partir de la cual comencemos el proceso de ingeniería inversa. De alguna forma necesitamos formalizar la información que extraigamos acerca de la estructura interna de la base de datos, por lo que almacenaremos un metamodelo que representa el esquema relacional de la base de datos.

De la misma forma, diseñaremos también un metamodelo para representar de manera formal la estructura de un *Entity Bean* (el tipo de EJB que nos interesa) y de un conjunto de componentes de este tipo. En este caso, el metamodelo nos permitirá no sólo formalizar los componentes que generemos, sino también las relaciones entre ellos. El metamodelo nos permitirá modelar también las posibles relaciones entre los componentes que generemos, ya que nuestra meta es generar *una capa* de componentes distribuidos que permita acceder a la base de datos original de forma remota, y para que esto se lleve a cabo de manera correcta, cada EJB, que representará una ocurrencia de la tabla, debe saber qué otras ocurrencias de otras tablas podrían verse afectadas por modificaciones, consultas, inserciones o eliminaciones. Es decir, que otros EJB deben ser notificados por posibles cambios. Todas las claves ajenas y restricciones de integridad que puedan existir en la base de datos se implementarán a base de relaciones entre los componentes EJB.

Esto se refleja en que, al final, lo que queremos es obtener un sistema compuesto por módulos entre los que se establecen una serie de relaciones, formalmente:

$$S = (M, R)\ ,\ R \subseteq M \times M$$

4.1. Metamodelo de una base de datos relacional

A continuación, se describe brevemente el metamodelo que hemos desarrollado con anterioridad y que puede encontrarse en [14]. Nótese que sólo describimos los elementos de la base de datos que, de momento, son relevantes para nuestro propósito.

Este metamodelo describe la información acerca de la estructura de la base de datos que es relevante y necesaria para nuestro propósito. Para nuestro caso, necesitamos almacenar información sobre las tablas que existen en la base de datos, las relaciones que existen entre ellas en forma de claves ajenas o *foreignkeys*, las columnas de que consta cada tabla, el tipo de dato que almacena cada columna, si es clave primaria, si es clave ajena, si admite valores nulos, etc.

Consideraremos que una base de datos DB, estará compuesta por tablas y por relaciones entre las tablas, las claves ajenas. Estas claves ajenas se definen entre dos tablas, la tabla padre y la hija. Algebraicamente expresado:

$$DB = (Tablas, FKS)$$

donde $FKS \subseteq Tablas \times Tablas$.

Dado $t \in Tablas = (Nombre, Columnas, Indices)$, que son, respectivamente, el nombre de la tabla, el conjunto de las columnas que la componen, y los índices, que es también un conjunto de columnas que constituyen una clave alternativa.

Dado $c \in Columnas = (Nombre, tipo, permitidoNotNull, esPK)$, que son, respectivamente, el nombre de la columna, el tipo de dato, si acepta valores nulos y si es clave primaria.

Por último, dado un $f \in FKS = (tPpal.Nombre, tSec.Nombre, cPpal, cSec)$, que son, respectivamente, el nombre de la tabla principal, el nombre de la tabla secundaria, la clave primaria de la tabla principal, y las columnas que forman la clave ajena en la tabla secundaria, y cuyos valores son restringidos por la clave primaria de la tabla principal.

4.2. Metamodelo del diagrama de clases

Dentro de nuestro proceso de reingeniería, el modelo de clases es el punto intermedio al cual queremos llegar al aplicar ingeniería inversa a la base de datos relacional, y del cual deseamos partir para realizar ingeniería directa para obtener la capa de componentes EJB.

Al igual que ocurre con la base de datos, necesitamos una medio formal y consistente para representar nuestros diagramas de clases, de forma que la estructura relacional de las bases de datos quede reflejada sin ambigüedad en nuestro modelo conceptual abstracto.

Los ítem a representar en este tipo de diagramas son las clases y las relaciones que pueda existir. Dentro de las clases, habría que distinguir entre clases normales, clases abstractas e interfaces. Las clases normales, incluyen métodos y atributos, las clases abstractas incluyen también métodos y atributos, con la salvedad de que algunos de estos métodos pueden no estar implementados, y las interfaces sólo contienen métodos sin implementar. Las interfaces pueden considerarse como una clase totalmente abstracta, donde la única relación que puede establecer es mediante implementación.

Formalizaremos la representación de estos diagramas, al igual que hacíamos en 3.2, mediante el álgebra relacional. Así pues, adoptaremos la formalización de un sistema basado en clases que se realizaba en [14].

Un sistema M, sería una tupla $M = (C, R)$, donde C es el conjunto de clases e interfaces existentes en el sistema, y R el conjunto de relaciones existentes entre los elementos de C, de forma que $R \subseteq C \times C$.

El conjunto $R = (A, D)$, donde A es el conjunto de asociaciones y agregaciones, y D es el conjunto de todas las dependencias.

A continuación, describiremos con más detalle los conjuntos C y A, que como veremos más adelante, nos darán parte de la descripción formal de nuestros componentes.

4.2.1. Formalización del conjunto de clases. Teniendo $c \in C = (Nombre, Campos, Métodos, Padres, esInterfaz)$, donde *Nombre* es el nombre de la clase, *Campos* es el conjunto de campos de la clase, *Métodos* es el conjunto de métodos de la clase, *Padres* es el conjunto de clases de las que hereda c y las interfaces que implementa, y *esInterfaz* es un valor lógico que dice si c es una interfaz o no.

Siguiendo una definición recursiva a partir de la definición de c, continuaríamos de la siguiente manera.

Dado $f \in Campos = (Nombre, Tipo, Visibilidad, esPK)$, donde *Nombre* es el nombre del campo, *Tipo* es el tipo del campo, *Visibilidad* es la visibilidad del mismo y *esPK* dice si forma parte de la clave primaria. Puede extrañar que a un campo de una clase se le asocie un atributo *esPK*, pero teniendo en cuenta que este diagrama de clases va a representar el esquema relacional de la base de datos, y que cada clase se corresponderá con una tabla, necesitamos saber, a fin de establecer las correspondientes relaciones, cuales de los campos de las clases constituyen las claves primarias en las tablas de la base de datos.

Dado $m \in Métodos = (Nombre, Tipo, Visibilidad, Argumentos, esEstática)$, donde *Nombre* es el nombre de la clase, *Tipo* es el tipo del valor que devuelve, *Visibilidad* es la visibilidad del propio método, *Argumentos* es un conjunto de elementos del tipo $(Nombre, Tipo)$ que pasamos a m y *esEstatica* que toma valores entre $\{verdadero, falso\}$.

$c.Padres$ vendría dado por la expresión $\{p \in C \mid p \text{ es una clase padre o interfaz implementada por } c\}$.

El atributo *esInterfaz* tomaría los valores $\{true, false\}$ en función de que la clase c sea una interfaz o no.

4.2.2. Formalización del conjunto de relaciones R. Consideraremos miembros de R a las asociaciones, agregaciones y dependencias. Teniendo en cuenta, que en esta parte, como hemos mencionado ya, hemos tomado como referencia el trabajo incluido en [14], comentaremos algunos aspectos necesarios para entender mejor la aproximación a la resolución del problema.

Las agregaciones y las asociaciones, se tratan de la misma manera, por lo que caen dentro del mismo conjunto, A, diferenciándolas más tarde en ciertos aspectos que no mencionaremos aquí. Las dependencias, que son relaciones temporales entre clases, caen en el conjunto D, de forma que $R=(A,D)$.

Dado $a \in A=(cL,cR,isFinalL,isFinalR,cardL,cardR,Name,Fields)$, donde cL y cR, son las clases que forman parte de la relación, *isFinalL* y *isFinalR*, dicen si estas clases son finales, en el sentido de la navegabilidad, *cardL* y *cardR*, representan la cardinalidad de cL y cR en la relación. *Name* se refiere al posible nombre que podamos darle, y *Fields*, representa un posible conjunto de campos de la asociación o de la agregación.

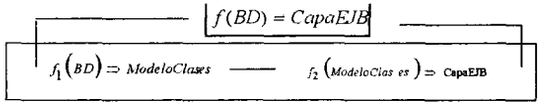
Dado $d \in A=(cL,cR)$, donde $cL \in C$ y $cR \in C$, y representan las clases que intervienen en la dependencia, siendo cL la que depende de cR.

4.3. Función de transformación

A groso modo, el concepto o idea que intentamos plasmar en este artículo, es el de el diseño y la implementación de una función que mediante técnicas y patrones de la ingeniería del software, nos permita realizar una compleja transformación que genere una capa de componentes EJB a partir de una base de datos relacional.

$$f(BD)=CapaEJB$$

Esta función de transformación, estaría definida como una función por partes, es decir, podría descomponerse en otro par de funciones, que actuarían sobre distintas partes del dominio del problema:



En los apartados anteriores, hemos descrito de una manera formal como se realiza la primera parte de la transformación a partir de la base de datos, es decir, cómo operaría la función f_1. Para la segunda fase de la transformación, que se correspondería con el proceso de ingeniería directa, tomaríamos como entrada para la función f_2 la salida de la primera función, es decir, el diagrama de clases que representa el esquema relacional de la base de datos.

Antes de analizar la función f_2, debemos representar formalmente lo que queremos obtener, es decir, debemos modelar de manera genérica esa capa de componentes EJB que interpondremos entre la base de datos y las aplicaciones cliente que intentes acceder a ella.

4.4. Formalización de la capa EJB

Llamaremos "*Capa EJB*" al conjunto de *Entity Beans* que permitirán al cliente realizar cierto tipo de operaciones sobre las tablas de la base de datos. El número de los componentes de los que esté conformada esta capa, podría dependerá de distintos factores, entre los que se encuentra la propia estructura de la base de datos, y el patrón de diseño [16] que apliquemos a la hora de realizar la "traducción". Más detalladamente,

cuando pasamos de un diagrama de clases a una base de datos, existen distintos patrones de diseño que nos ofrecen soluciones genéricas a problemas bien conocidos.

En nuestro estudio, eliminamos esta ambigüedad, generando siempre un número predecible de componentes. Cuando creemos la capa de componentes a partir de la base de datos, tendremos siempre tantos componentes como tablas existan en la base de datos. Así y de esta forma nos evitamos el tener que realizar cálculos complejos para determinar determinadas estructuras en la base de datos.

Así pues, llamaremos E a la capa EJB, y diremos que $E=(S,R)$, donde S es el conjunto de *Entity Bean* y R es el conjunto de relaciones existentes entre los componentes de la capa: $R \subseteq SxS$

4.4.1. Metamodelo de un Entity Bean. Como hemos visto hasta ahora, nuestra motivación y nuestro fin es el generar esta capa de componentes que nos permitirán realizar diversas operaciones sobre la base de datos. A lo largo de este apartado se describe de manera formal la estructura genérica de un EJB *Entity Bean*, es decir, vamos a intentar plantear un metamodelo que nos permita representar los EJB y las posibles relaciones que existan entre ellos (véase sección 3).

Diremos que $s \in S=(InterfaceHome, InterfaceRemota, ClaseEJB, Auxiliares)$, donde *InterfaceHome* contiene las cabeceras de los métodos que permiten al cliente encontrar y crear instancias de un *Entity Bean*, *InterfaceRemota* contiene los métodos o servicios que el cliente puede realizar sobre la base de datos a través del componente, y *Auxiliares*, que representa al conjunto de clases que pueden ser necesarias para que el componente realice su función.

ClaseEJB = (*CamposT*, *CampoC*, *OtrosAtributos*, *ImpMHome*, *ImpMRemota*, *metodosAcceso*, *OtrosMetodos*), donde *CamposT* son los atributos del EJB que corresponden con los campos de la tabla, *CampoC*, es un atributo que almacena el contexto del componente, *OtrosAtributos* es el conjunto de atributos adicionales que podrían ser necesarios, *ImpMHome* sería el conjunto de métodos definidos en *InterfaceHome*, pero ya implementados, *ImpMRemota* sería el conjunto de métodos definidos en *InterfaceRemota*, pero ya implementados, *metodosAcceso* son métodos *get_* y *set_* que permiten acceder y establecer los valores de todos los atributos, y *OtrosMetodos* sería el conjunto de métodos que el desarrollador necesite agregar para realizar determinadas operaciones, como puede ser realizar una inserción en la base de datos.

4.5. Descripción formal proceso de ingeniería directa, la función f_2

El algoritmo principal, que genera la capa de *Entity Bean*, toma como entrada el diagrama de clases que se ha obtenido previamente aplicando ingeniería inversa a la base de datos. A partir de este diagrama, el algoritmo calcula cuáles son los componentes EJB necesarios y los crea también, basándose en el metamodelo que hemos comentado brevemente en los puntos anteriores.

A continuación se detalla un fragmento de este algoritmo:

```
1   crearCapaEJB(M:DiagramaClases):E{
2     E = (∅,∅);
3     S = ∅;
4     R = ∅;
5     ∀c ∈ M.C{
6       vCamposTabla = ∅;
7       vCampoContext = inicializarContext();
8       vOtrosAtributos = ∅;
9       vMetodosAcceso = ∅;
10      ∀f ∈ c.Fields{
11        if(f.isPK)
12          vCamposTabla = vCamposTabla ∪ f;
13        else
14          vOtrosAtributos = vOtrosAtributos ∪ f;
15        vMetodosAcceso = vMetodosAcceso ∪ ("get" + f.Name,f.Type,"#",∅,False);
16        vMetodosAcceso = vMetodosAcceso ∪ ("set" + f.Name,void,"#",f.Type,False);
17      }
18      vClaseEJB = (vCamposTabla,vCampoContext,vOtrosAtributos,∅,∅,∅,vMetodosAcceso);
19      S = S ∪ (vClaseEJB,crearInterfaceHome(),crearInterfaceRemota());
20    }
      ⋮
      [resto de función]
nn  }
```

donde se puede observar, entre otras cosas, que la entrada es un diagrama de clases, y que devuelve un elemento E que, como vimos en 3.4, representa la *capa EJB*. Para tener E completo, debemos calcular S y R, que son el conjunto de Entity Bean y las relaciones entre los mismos respectivamente.

En el algoritmo, se ve lo que podría equivaler al cálculo de S. Puede observarse, como tomamos el conjunto de clases generado en el proceso de ingeniería directa, y lo transformamos a un conjunto de *Entity Bean*. Por cada clase hemos creado un componente EJB que se completa con la información extraída de la clase.

Cada uno de los EJB generados contendrá los métodos estándar *findBy* que nos permitirán la materialización de instancias a partir de la información de la base de datos. Entre estos métodos cabe destacar los *findByPrimaryKey*, cuya misión es realizar búsquedas por la clave primaria de la tabla que le pasaremos; si la tabla a la que representa el EJB dispone además de la clave primaria de alguna clave alternativa, también se generará un método tipo *findBy[ClaveAlternativa]* al que pasaremos como argumento el valor de dicha clave para la búsqueda. En el caso de que la clave primaria de la tabla o la alternativa estén formadas por más de una columna, generaremos, además, un método del tipo *findBy[NombreDel.aColumna]* para cada una de las columnas que formen parte de las claves, a fin de obtener una colección de registros de la base de datos a partir del dato que le pasemos como argumento.

5. Refinamiento Manual

Cuando generamos toda la capa de Entity Bean es preciso instrumentar los componentes EJB para indicar qué operaciones va a haber a disposición del cliente.

Junto con la interfaz del componente, hay otros métodos, que no se generan automáticamente. Algunos de estos métodos son los que puedan ser necesarios para permitir que el Bean realice su tarea de cómputo, es decir, métodos privados.

Como es lógico, el objetivo de nuestro estudio es conseguir la mayor automatización posible de este proceso, permitiendo la generación de la capa de EJB de forma sencilla según distintas alternativas. Para mermar la dificultad o incomodidad que pudiera derivarse de la realización de este código que es inevitable realizar, hemos pensado en la realización de una serie de asistentes que en cierto modo van a permitir añadir estas funcionalidades a nuestros componentes en desarrollo.

La primera propuesta es la de implementar un pequeño entorno que permita realizar de forma gráfica y mediante diagramas de actividad, la especificación de las operaciones que tengamos que agregar de forma manual [8]. El uso de estos diagramas requeriría ciertos conocimientos del lenguaje de modelado UML para poder realizar la descripción de las operaciones.

Como podría caber la posibilidad de que algunos de los posibles usuarios no tuvieran los conocimientos o practica necesaria para utilizar dicho tipo de diagramas, también se implementaría, dentro de este entorno, un asistente para que de forma sencilla permita incluir manualmente el código de los métodos. Este asistente permitiría que la tarea de editar los archivos de texto se convierta en un sencillo proceso, en el que los pasos triviales serían tan sencillos que solo tendríamos que pensar qué código es el que tenemos que insertar.

6. Conclusiones y líneas de trabajo futuras

En este artículo se han presentado los primeros resultados de un trabajo con el que pretendemos generar de manera semiautomática aplicaciones basadas en componentes distribuidos, constituyendo una línea de continuación del trabajo presentado en [14]. Se han descrito los diferentes metamodelos involucrados y el método seguido para generar la capa de componentes, estando aún pendiente la descripción de los metamodelos y funciones de transformación correspondientes al resto de capas de la aplicación.

A lo largo del artículo, sólo hemos hablado de un tipo de componente EJB denominado *Entity Bean*. Es muy posible que podamos de alguna forma utilizar las otras clases de componentes EJB para ofrecer soluciones con un nivel de granularidad más fino, que sea capaz de ofrecer soluciones más personalizadas y eficientes a problemas más concretos, así como adaptar el trabajo para generar aplicaciones para otras plataformas, como Microsoft .NET.

Como idea para una futura mejora del sistema, se plantearán otras alternativas a la hora de generar el middleware de componentes EJB a partir de la base de datos. Esto se refiere a que en lugar de realizar la traducción *una tabla, un EJB*, se planteará la posibilidad de utilizar alguna heurística para agrupar conjuntos de tablas (que puedan ser agrupadas por representar herencias u otro tipo de relaciones) para ser asociadas a un solo componente.

7. Agradecimientos

Este trabajo está parcialmente financiado por el proyecto MÁS (Ministerio de Ciencia y Tecnología/FEDER, TIC2003-02737-C02-02).

8. Referencias

[1]Piattini, M.G., Calvo-Manzano, J.A., Cervera, J. y Fernández, L. *Aplicaciones informáticas de gestión*, Editorial RA-MA, Madrid, 1996.

[2]Short, S. *Creación de Servicios Web XML para la plataforma Microsoft ® .NET*, Editorial Mc Graw Hill, Madrid, 2002.

[3]Arnold, R.S., *Software Reeingineering*, IEEE Press, 1992

[4]*http://www.omg.org*, Disponible en 15-12-2003.

[5]*http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp*, Disponible en 15-12-2003.

[6]http://java.sun.com, Disponible en 15-12-2003.

[7]http://java.sun.com/products/ejb/, Disponible en 15-12-2003.

[8]Booch, G., Rumbaugh, J., Jacobson, I., *El Lenguaje Unificado de Modelado*. Editorial Addison Wesley, Madrid, 1999.

[9]http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts4.html, Disponible el 15-12-2003.

[10]http://www.cs.umd.edu/~pugh/com/, Disponible en 15-12-2003.

[11]http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html, Disponible el 15-12-2003.

[12]http://www.ietf.org/html.charters/oncrpc-charter.html, Disponible el 15-12-2003.

[13]http://www.w3.org/TR/ws-arch/, Disponible el 15-12-2003.

[14]M. Polo, J.A. Gómez, M. Piattini, F. Ruiz, *Generating three-tier applications from relational databases: a formal and practical approach*, Information and Software Technology 44 (2002) 923-941.

[15]http://java.sun.com/products/javabeans/, Disponible el 15-12-2003.

[16]Gamma, E. et al, *Design patterns: elements of reusable object-oriented software*, Editorial Addison-Wessley, 1995.

[17]http://java.sun.com/products/jdbc/, Disponible el 15-12-2003.

[18]http://java.sun.com/j2ee/, Disponible el 15-12-2003.

[19]http://www.W3.org/TR/2000/REC-xml-20001006, Disponible el 15-12-2003.

[20]http://www.w3.org/Style/XSL/, Disponible el 15-12-2003.

[21]http://java.sun.com/xml/jaxp/index.jsp, Disponible el 15-12-2003.

[22]http://rodrigo.gnome-db.org/documentation/talk-corba/, Disponible el 15-12-2003.

[23]T.J. Biggerstaff, B.G. Mitbander, D.E. Webster, *Program understanding and the concept assignment problem*, Communications of the ACM 37 (5) (1994) 72-83.

[24]M. Andersson, en: Loucopolous (Ed.), *Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering*, Proceedings of the 13th International Conference on Entity-Relationalship Approach, Springer-Verlag, Berlin, LNCS, vol. 881, 1994, pp. 403-419.

[25]L. Pedro de Jesús, P. Sousa, en: Nesi, Verhoef (Eds.), *Selection of Reverse Engineering Methods for Relational Databases*, Proceedings of the Third European Conference on Software Maintenance, IEEE Computer Society, Los Alamitos, CA, 1998, pp. 194-197.

[26]R. Chiang, T. Barron, V.C. Storey, *Reverse engineering of relational databases: extracting of an EER model from a relational database*, Journal of Data and Knowledge Engineering 12 (2) (1994) 107-142.

[27]J.L. Hainaut, J. Henrard, J.M. Hick, D. Roland, V. Englebert, *Database Design Recovery*, Proceedings of Eighth Conferences on Advance Information Systems Engineering, CAiSE'96, Springer, Berlin, 1996, pp. 463-480.

Elicitação de Requisitos para o Desenvolvimento de uma Ferramenta de Apoio à Metodologia de Elicitação de Requisitos de Software Baseada na Teoria da Atividade (META[1])

Simone Franceto
UNIMEP - Universidade Metodista de Piracicaba
simonefranceto@ig.com.br

Luiz Eduardo Galvão Martins
UNIMEP - Universidade Metodista de Piracicaba
lgmartin@unimep.br

Abstract

This work presents the requirements elicitation for the development of an automated tool for a Requirements Elicitation Methodology Based on Activity Theory. The objective of such tool is to support the use of the methodology (called META), not only stimulating and improving the productivity of the development team but also improving the quality of the requirements engineering process. The tool should also collect and store the data in a repository, adding quality to the process of requirements elicitation, besides helping the understanding of the system context to the subsequent phases of the requirements engineering and the implementation of the software system. Documentation, requirements analysis and logical project of the tool also are presented in this article.

Keywords: META, Automated Tool, Activity Theory, Requirements Elicitation, Requirements Engineering.

1. Introdução

Na fase inicial da Engenharia de Software, a utilização de métodos, técnicas e ferramentas dão suporte à fase da Engenharia de Requisitos, auxiliando na obtenção de requisitos claros e consistentes para o desenvolvimento do software. O levantamento de requisitos compreensíveis por todas as partes envolvidas no desenvolvimento do sistema (clientes, analistas, desenvolvedores etc.) é um fator essencial que previne falhas no entendimento do problema a ser solucionado e, conseqüentemente, evita que tais falhas sejam propagadas ao sistema que está sendo desenvolvido.

Este trabalho apresenta a elicitação de requisitos feita para o desenvolvimento de uma ferramenta automatizada que dê suporte à META (a metodologia de elicitação de requisitos empregada foi a própria META). O objetivo da ferramenta proposta é agilizar e otimizar o emprego da META, impulsionando sua utilização e melhorando a produtividade da equipe de desenvolvimento que adotar esta metodologia, bem como melhorar a qualidade da elicitação de requisitos realizada. A ferramenta auxiliará o emprego da metodologia, garantindo a organização e o armazenamento das informações levantadas ao longo da elicitação de requisitos feita com a META, agregando qualidade ao processo de elicitação de requisitos, além de auxiliar na compreensão do ambiente do sistema.

Para elicitar os requisitos da ferramenta proposta, foram seguidos os princípios e conceitos da Teoria da Atividade, conforme discutida por [8] [4]. Assim, a ferramenta está fundamentada na Metodologia de Elicitação de Requisitos de Software Baseada na Teoria da Atividade proposta por Martins [5].

2. Revisão Bibliográfica

2.1. Papel da Engenharia de Requisitos

---

[1] META: Metodologia de Elicitação de Requisitos Baseada na Teoria da Atividade