FACENA, Vol. 19, pp. 33-45, 2003

BASES DE DATOS OBJETO-RELACIONALES Y LA TECNOLOGIA J2EE. CASO PRACTICO: GESTION DEL MACROSISTEMA DEL IBERA. EXPERIENCIAS DURANTE EL DISEÑO Y LA IMPLEMENTACION

María F. GOLOBISKY[1]; Fernando E. FISZMAN[2]; Federico HERNÁNDEZ[2]; Vanina RIZZONI[2]; Maximiliano VALIN[2]; Aldo R. VECCHIETTI[2] y Blanca B. ALVAREZ[3]

ABSTRACT: This work describes a web application design using an Object-Relational Database as data repository. The goal is to generate a computer system to manage the information of reptiles and amphibious (herpetofauna) of the Ibera macrosystem located in Corrientes, Argentina. The ORDBMS selected is Oracle9i, which is one of the most advanced Object-Relational system existing today because it fulfils in a high percentage the SQL:1999 standards. The Object-Relational technology is selected due to the data complexity and the relationships between them including the handling of complex data types, collections, inheritance, composition and association. The application will be executed by standard web browsers. The modules of the application are developed in Java on a J2EE platform. Several tools are used for this system development: Rational Rose (analysis in UML), Object Database Designer, JDeveloper 9i, Oracle Containers for Java, Struts, etc. The article also describes the most relevant parts of the data modeling, the experiences gathered in the development, the solutions adopted in the conceptual design of the database, the application architecture design and the links between this structure and the database.

RESUMEN: En este trabajo se presenta el diseño de una aplicación web, teniendo como repositorio de datos una Base de Datos Objeto-Relacional. El objetivo es generar un sistema de información para gestionar la información de los reptiles y anfibios (herpetofauna) del macrosistema del Iberá en la provincia de Corrientes. El ORDBMS que se emplea es Oracle9i, debido a que en la actualidad es uno de los sistemas Objeto Relacional mas avanzado, que cumple en gran medida con los estándares de SQL:1999. Se emplea una Base de Datos Objeto Relacional por la complejidad de los datos y de las relaciones entre lo mismos, que incluyen el manejo de tipos de datos complejos, colecciones, relaciones de herencia, composición y asociación. La aplicación se ejecutará en browsers estándar en un entorno web. Los módulos de la aplicación se desarrollan en Java sobre una plataforma J2EE. Se emplearon varias herramientas: Rational Rose (análisis en UML), Object Database Designer, JDeveloper 9i, Oracle Containers for Java, Struts, etc. En el trabajo se describen las partes más relevantes del modelo de datos, las experiencias recogidas en el desarrollo del mismo, las soluciones adoptadas en el diseño conceptual de la Base de Datos, el diseño de la arquitectura de la aplicación y cómo se relaciona la arquitectura con la base de datos.

Palabras claves: SQL:1999, Base de Datos Objeto-Relacional, Diseño conceptual, Arquitectura de N capas, J2EE y ORDBMS, Aplicación Web.

Key words: SQL:1999, Object-Relational Database, Conceptual Design, N-layer Architecture, J2EE and ORDBMS, Web Application.

---

(1) Departamento de Informática. Facultad de Ciencias Exactas y Naturales y Agrimensura (UNNE). 9 de Julio 1449. 3400 Corrientes, Argentina. E-mail: mfgolo@exa.unne.edu.ar
(2) Departamento de Sistemas. Facultad Regional Santa Fe (UTN). Lavaise 610. 3000 Santa Fe, Argentina.
E-mail: {ffiszman, fhernand, vrizzoni, mvalin}@frsf.utn.edu.ar; aldovec@ceride.gov.ar
(3) Departamento de Biología. Facultad de Ciencias Exactas y Naturales y Agrimensura (UNNE). Av. Libertad 5470. 3400 Corrientes, Argentina. E-mail: balvarez@exa.unne.edu.ar

INTRODUCCIÓN

Durante muchos años las Bases de Datos Relacionales (RDBMS) dominaron el mercado de las bases de datos y de los sistemas de información de las organizaciones, permitiéndoles a éstas últimas automatizar sus procesos más importantes. La metodología más empleada para modelar los datos de estas aplicaciones es la Entidad/Relación y sus extensiones. Los tipos de datos que se modelan en las Bases de Datos relacionales son del tipo atómico, a los que se refieren frecuentemente como "simplemente estructurados". La tecnología logró gran aceptación desde mediados de la década del '80, pero las ventajas competitivas logradas con los RDBMS han disminuido con el transcurso del tiempo, fundamentalmente en los últimos cinco años. Para lograr ventajas competitivas en nuestros días las organizaciones necesitan aplicaciones del tipo Internet/Intranet y un conjunto más extenso y complejo de tipos de datos que permitan el manejo de datos estructurados, relaciones de composición y herencia así como la administración de imágenes, videos, sonidos, series de tiempo, datos geoespaciales (McClure, 1997). Las tecnologías que surgieron para manejar estos nuevos tipos de datos y las relaciones más complejas que existen entre los mismos son las Bases de Datos Orientadas a Objetos (ODBMS) y las Bases de Datos Objeto Relacionales (ORDBMS). Los ODBMS surgieron para dar un soporte persistente al modelo orientado a objetos que surgió, en primer lugar, como una tecnología de programación con estándares muy definidos orientada a modelar datos complejos y a la generación de módulos reusables y de fácil mantenimiento. Los ODBMS facilitaron el desarrollo de sofisticadas aplicaciones como las CAD/CAM, CASE y GIS

(Vela *et al.*, 2001). Los ODBMS combinan los elementos de orientación a objetos en conjunto con los lenguajes de programación orientado a objetos con capacidades de base de datos, que van mas allá de la simple persistencia de los objetos. Estas bases de datos cuentan con la metodología UML para la generación de los modelos de datos. Como resultado existe una mayor congruencia entre el modelo de datos de la aplicación y el de la base de datos. Estas bases de datos son más apropiadas para el manejo de datos complejos que no sean muy voluminosos.

Los ORDBMS surgen como una respuesta a la incorporación de la tecnología de objetos en las Bases de Datos Relacionales y con ello permitir el tratamiento de datos y relaciones complejas. Por ser una extensión de la tecnología relacional presenta dos ventajas frente a los ODBMS: que es compatible con la tecnología relacional y tiene un mejor soporte para aplicaciones voluminosas (Vela *et al.*, 2001). Ambas tecnologías son nuevas y no han alcanzado aún un alto grado de aceptación y madurez, tal vez porque no existen productos en el mercado que cumplan con los últimos estándares: ODMG3.0 para los ODBMS (Cattel y Barry, 2000) y SQL:1999 (Eisenberg y Melton, 1999) para los ORDBMS. Se espera que en los años por venir y cuando la tecnología adquiera una mayor madurez los ORDBMS ocupen un importante lugar en el mercado (Dorsey, 1999) como lo hizo la tecnología relacional desde mediados de los ’80 hasta mediados de los ’90. Dado que se encuentran en una etapa de desarrollo estas bases de datos no cuentan con una metodología de modelado propia.

Si bien el diseño e implementación de una base de datos objeto-relacional ha sido tratado en la literatura, a la hora de abordar un proyecto concreto surgen muchas dudas y quedan cuestiones sin resolver (Kovacs *et al.*, 1998) (Vela *et al.*, 2001). Es por eso que en este trabajo se presentan las experiencias realizadas en el análisis, diseño e implementación de un sistema, para la administración de la información relacionada con las investigaciones taxonómicas y ecológicas básicas sobre la herpetofauna del macrosistema Iberá. Para el sistema que estamos desarrollando una Base de Datos Objeto Relacional es el medio mas natural para implementarlo, debido a la naturaleza de la información y a la complejidad de las relaciones existentes entre los datos.

Este trabajo se realiza en conjunto con investigadores del área de biología de la Universidad Nacional del Nordeste, que se encuentran trabajando en la región desde

hace ya varios años. El sistema reflejará la composición faunística de una región con características únicas, la que se encuentra ubicada en el centro-norte de la provincia de Corrientes. Este reservorio hídrico y de vida silvestre, con una de las más altas biodiversidades faunística y florística del país puede alterarse dramáticamente, debido a actividades antrópicas que se encuentran afectando las zonas perimetrales del mismo. Cualquier plan de conservación del sistema es imposible llevarlo a cabo si se desconocen aspectos básicos tales como diversidad de especies, distribuciones, biogeografía y ecología. A partir del estudio de los mismos se puede establecer el funcionamiento del sistema sentando las bases para la predicción, comparación y entendimiento del efecto de las actividades humanas sobre el mismo, de allí la importancia del sistema que se modela y presenta en este trabajo.

Análisis del dominio y metodología utilizada

Debido a la complejidad del dominio del problema y a que no existe una metodología disponible que guíe el proceso de análisis y diseño de una Base de Datos Objeto Relacional (Grimes, 1998), se convino utilizar la metodología Orientada a Objetos (OO) para la generación del modelo conceptual esperando lograr una abstracción más ajustada de la aplicación en desarrollo, donde existen relaciones complejas que incluyen herencia, composición y asociación. Con el modelo OO, mucho más expresivo y completo que los sistemas de modelado convencionales, esperamos contar con una definición apropiada de los tipos de datos y lograr que la construcción del sistema se haga de manera más rápida y a menor costo.

En esta etapa se definió el dominio a partir de las abstracciones que forman el vocabulario del dominio del problema, se identificaron las necesidades de información a partir de dicha descripción y se especificaron las características del sistema propuesto en términos de objetos.

Para el análisis del sistema se comenzó con la utilización de herramientas CASE de análisis y diseño Orientado a Objetos, buscando con ello acelerar el proceso de definición de la estructura del sistema, su documentación y la automatización de las diferentes etapas de la construcción del software. Para modelar los objetos se emplearon dos herramientas disponibles en el mercado: Rational Rose (Rational Rose, 2000) y Object Database Designer. Ambas pro-

veen la notación UML (Unified Modelling Lenguage) que es el lenguaje de modelado orientado a objetos más difundido en la actualidad. Se seleccionaron estas herramientas porque además de emplear la notación UML, y permitir modelar complejas relaciones entre los objetos, ambas poseen la capacidad de generar los scripts de creación de los objetos para un conjunto amplio de DBMS.

En la Fig. 1 se presenta el diagrama de clases del subconjunto más relevante de la aplicación generado con UML en el que se detallan las principales características del modelo. En el mismo se visualiza la clasificación taxonómica de las distintas especies (familia, orden, suborden, etc.), los ambientes del macrosistema del Iberá donde se efectuaron las recolecciones de las mismas, la ubicación geográfica de esos ambientes, y el personal que realizó la recolección.

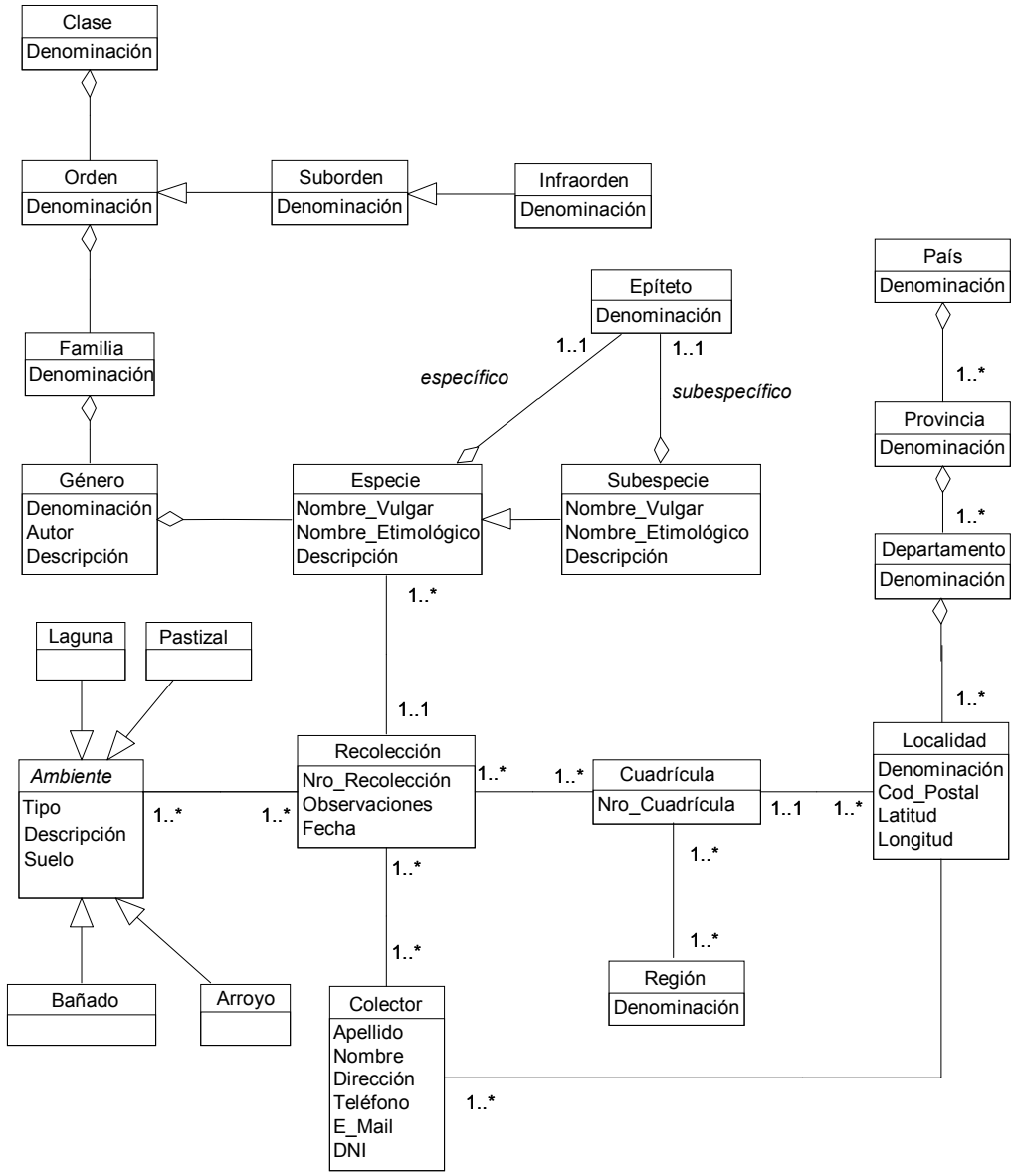

Fig. 1: Diagrama de clases de un subconjunto de la aplicación

Como se puede observar en la figura en este subconjunto se presentan relaciones de herencia, composición y asociación:

- Existe herencia al definir a la clase *Ambiente*, como clase abstracta, de la cual derivan los diferentes am-

bientes que se encuentran en el macrosistema del Iberá: Bañado, Arroyo, Laguna, Pastizal.

- También utilizamos herencia al definir a una especie y a sus correspondientes subespecies: una subespecie además de los atributos que hereda de especie, posee los propios.
- La clasificación taxonómica de la herpetofauna se realizó a través de relaciones de composición que van desde la clase Clase hasta Especie.
- Se utilizó herencia al definir los diferentes niveles de generalización de la clase Orden.
- Las ubicaciones geográficas del macrosistema se modelaron con composición, desde la clase País hasta la clase Localidad.
- Existen Asociaciones entre los objetos Colector, Recolección, Cuadrícula, Región y Localidad.

Diseño Lógico de la Base de Datos

Para el desarrollo de la aplicación se empleó Oracle9i por ser un ORDBMS que implementa las especificaciones para nuevos tipos de datos del estándar "SQL:1999": tipos para objetos grandes, CLOB y BLOB, que posibilitan la implementación de textos grandes, de gráficos, imágenes, sonidos y videos; datos estructurados como colecciones y arreglos, y distintos tipos definidos por el usuario que permiten establecer relaciones complejas como herencia, composición, referencias de objetos (REF), navegación directa, tablas de objetos polimórficas (Gietz, 2001).

La herramienta Rational Rose, posee un "add-in" para generar los scripts de la base de datos Oracle 8i, pero no posee una forma automática para pasar de los diagramas de análisis al modelo conceptual de la Base de Datos. Para realizar esto se deben rediseñar los diagramas definiendo en forma separada los objetos de la base y sus correspondientes tablas. No existe una forma directa de pasar de los diagramas lógicos a los scripts para generar los objetos de la Base. Por consiguiente los diagramas se volvían engorrosos y difíciles de comprender. Existían también limitaciones en cuanto al manejo de herencia y de tablas anidadas, estas últimas, estructuras claves para la implementación de las composiciones de nuestro problema.

Por otro lado Object Database Designer (ODD) (Oracle Designer, 1999) permite pasar directamente de los diagra-

mas generados a los scripts de la Base, pero para la versión 8i de Oracle, versión que no soporta herencia y no permite la utilización óptima de los collections types como tablas anidadas, arrays, ni el multidimensionamiento de los mismos. En conclusión, ninguna de las dos herramientas empleadas genera los scripts adecuados al diseño de los componentes de la Base de Datos seleccionados. De las dos opciones mencionadas se optó por generar el modelo lógico de la base de datos empleando ODD, ya que se adaptaba mejor al ORDBMS seleccionado para este trabajo, y exigía menos esfuerzo que Rational Rose. A partir del modelo lógico obtenido se generaron los scripts de la base, que luego se modificaron manualmente para adaptar las sentencias SQL a Oracle 9i (Oracle Corporation, 2001).

A continuación y con el objeto de presentar un ejemplo ilustrativo, se detalla el código SQL para implementar las relaciones de herencia que se generan a partir de la clase Ambiente y sus tablas asociadas:

```
CREATE OR REPLACE TYPE AMBIENTE_OT AS OBJECT
(DESCRIPCION CLOB,
TIPO VARCHAR2(30),
TIPO_SUELO VARCHAR2(20),
MEMBER FUNCTION
Valor_Correcto(tipoA varchar(30))
RETURN NUMBER)
NOT FINAL;

/*NOT FINAL indica que la clase es abstracta y no permite su instanciación.*/

/*El método Valor_Correcto determina si el atributo tipo a insertar está dentro de los tipos posibles para cada ambiente, cada subclase de ambiente reescribe el método con sus propios valores posibles*/

CREATE OR REPLACE TYPE ARROYO_OT UNDER AMBIENTE_OT
(OVERRIDING MEMBER FUNCTION
Valor_Correcto(tipoA
varchar(30)) RETURN NUMBER)

CREATE TABLE ARROYO_T OF ARROYO_OT
(PRIMARY KEY (TIPO))
```

Implementación del diseño de la Base Objeto Relacional

Los scripts necesitaron ser modificados para poder utilizar estructuras de tablas anidadas para las relaciones de composición. Si bien esta estructura no es una de las especificadas por el estándar "SQL: 1999" se la seleccionó porque es la adecuada para representar una colección de elementos donde no es necesario asignar un máximo (como

en los arreglos variables VARRAYS), el orden de los elementos no es importante y en donde la consulta de un elemento particular de la colección debe hacerse de manera eficiente. La taxonomía de las especies presenta estas características: una Clase está compuesta por un conjunto de Ordenes, cada Orden a su vez tiene un conjunto de Familias, cada Familia tiene Géneros, y así sucesivamente hasta llegar a la Especie. No hay un tamaño definido, tampoco es importante el orden entre los elementos de un conjunto, y sí interesa la búsqueda eficiente de un elemento particular del conjunto, especialmente cuando se navega en la base de datos para inserción y/o búsqueda de una especie. El siguiente código SQL muestra cómo fue implementada esta estructura:

```
/*Creación de Tipos*/

CREATE TYPE SUBESPECIE_OT

CREATE TYPE SUBESPECIE_TT AS
TABLE OF REF SUBESPECIE_OT

CREATE OR REPLACE TYPE
ESPECIE_OT AS OBJECT
(NOMBRE_VULGAR VARCHAR2(60),
NOMBRE_ETIMOLOGICO VAR-
CHAR2(60),
EPITETO VARCHAR2(30),
DESCRIPCION CLOB,
REGIMEN_ALIMENTARIO VAR-
CHAR2(30),
REPRODUCCION VARCHAR2(30),
MORFOLOGIA REF MORFOLOGIA_OT,
SUBESPECIE SUBESPECIE_TT,
MEMBER PROCEDURE vacio) NOT
FINAL

CREATE OR REPLACE TYPE SUB-
ESPECIE_OT UNDER ESPECIE_OT
(OVERRIDING MEMBER PROCEDURE
vacio)

CREATE TYPE ESPECIE_TT AS
TABLE OF REF ESPECIE_OT


/*Creación de Tablas Anida-
das*/

CREATE OR REPLACE TYPE GENE-
RO_OT AS OBJECT
(AUTOR VARCHAR2(30),
DENOMGENER VARCHAR2(30),
DESCRIPCION CLOB,
ESPECIE ESPECIE_TT)

CREATE TYPE GENERO_TT AS TABLE
OF GENERO_OT

CREATE OR REPLACE TYPE
FAMILIA_OT AS OBJECT
(DENOMFAMIL VARCHAR2(30),
GENERO GENERO_TT )

CREATE TYPE FAMILIA_TT AS
TABLE OF FAMILIA_OT

CREATE OR REPLACE TYPE
ORDEN_OT AS OBJECT
(DENOMORDEN VARCHAR2(30),
DENOMSUBOR VARCHAR2(30),
DENOMINFRAO VARCHAR2(30),
FAMILIA FAMILIA_TT )

CREATE TYPE ORDEN_TT AS TABLE
OF ORDEN_OT
CREATE OR REPLACE TYPE
CLASE_OT AS OBJECT
(DENOMCLASE VARCHAR2(30),
ORDEN ORDEN_TT )
```

```
CREATE TABLE SUBESPECIE_T OF
SUBESPECIE_OT
(PRIMARY KEY
(NOMBRE_ETIMOLOGICO))
NESTED TABLE SUBESPECIE STORE
AS SUBESPECIE_NULL_NT

CREATE TABLE ESPECIE_T OF
ESPECIE_OT
(PRIMARY KEY
(NOMBRE_ETIMOLOGICO))
NESTED TABLE SUBESPECIE STORE
AS SUBESPECIE_REF_NT

CREATE TABLE CLASE_T OF
CLASE_OT
(PRIMARY KEY (DENOMCLASE))
NESTED TABLE ORDEN STORE AS
ORDEN_NT
((PRIMARY KEY (DENOMORDEN,
DENOMSUBOR, DENOMINFRAO))
  NESTED TABLE FAMILIA STORE
  AS FAMILIA_NT
  ((PRIMARY KEY (DENOMFAMIL))
    NESTED TABLE GENERO STORE
    AS GENERO_NT
    ((PRIMARY KEY
    (DENOMGENER))
      NESTED TABLE ESPECIE
      STORE AS
      ESPECIE_REF_NT
                )
    )
)
```

En el código SQL anterior se debe notar la implementación de la jerarquía ESPECIE-SUBESPECIE. En la definición de la clase Especie se ha utilizado una función ficticia, el método vacío(), que no realiza ninguna función en especial, sólo permite definir una relación de herencia, que no podría hacerse si no se modifica en algo la subclase (atributo, método, o especialización de método) respecto de la superclase. Implementar herencia en este punto era importante porque en otras partes del sistema, otras clases pueden tener referencias a ESPECIE pero no a SUBESPECIE. Por ejemplo, se puede recolectar una subespecie específica de una especie y por lo tanto se debe almacenar una referencia a subespecie y no a especie, y en otros casos a la inversa, recolectar una especie que no posee subespecies, almacenando una referencia a la especie. Entonces en vez de utilizar dos referencias, una para cada clase y usar una u otra dependiendo del caso, se utilizó herencia porque permite la sustitución de tipos entre la superclase y la subclase.

También se presenta en el código la estructura taxonómica a través de tablas anidadas. Se observa que la tabla anidada ESPECIE_TT es por referencia, sus filas son referencias a especies. Esto es así porque, como se mencionó, otras clases poseen referencias a especies y si estas están almacenadas en una tabla anidada no es posible hacer una referencia a ellas externamente, por limitaciones propias del ORDBMS. Por lo que se creó una tabla especial para las especies, que solucione este problema. Lo mismo ocurre con las subespecies.

Otro caso especial es el de la implementación de herencia para Orden, Suborden e Infraorden. En principio se podría pensar en generar una serie de tablas anidadas relacionadas con Especie, pero esto triplicaría la cantidad de tablas anidadas; se debe crear una tabla anidada de Familia para cada una de las tres clases mencionadas en la jerarquía. La transformación que se realizó en este caso es la que se emplea para jerarquía de herencia cuando existe superposición entre las distintas subclases: se crea una sola tabla que contiene los atributos de la superclase y de las subclases, más un atributo que identifica el tipo al que pertenece, obviamente en este caso se generan valores nulos (Elmasri y Navathe, 2000).

A través de la palabra clave TABLE se puede acceder de manera sencilla, por ejemplo, a todos los nombres etimológicos de Especie que derivan de la Clase 'Reptilia'. El código de la consulta se detalla a continuación:

```
SELECT esp.nombre_etimologico
FROM Clase_T c, TABLE (c.Orden) o, TABLE (o.Familia) f, TABLE
(f.genero) g, TABLE (g.especie) esp
WHERE c.denomclase='Reptilia'
```

Diseño de la arquitectura de la aplicación

El proyecto Iberá está dirigido a generar una aplicación web. Como arquitectura de la misma se ha elegido un modelo de 3 capas debido a que se pueden utilizar web browsers (Internet Explorer, Netscape Navigator, etc.) como clientes, lo que facilita su distribución y mantenimiento, permitiendo que tanto en el ambiente de la Universidad Nacional del Nordeste, donde residirá la aplicación, como en cualquier parte del mundo cualquier persona pueda conocer este macrosistema y cualquier investigador pueda trabajar on-line con toda la información que necesite, sólo con disponer de una conexión a Internet. Además posibilita que la aplicación esté centralizada, esto es la capa de aplicación y la capa de presentación, concentrados en un servidor, para facilitar el mantenimiento y asegurar que los clientes tendrán siempre la versión actualizada de la aplicación. La capa de base de datos separada y centralizada en un servidor de datos, permite que la misma información pueda ser accedida por diferentes aplicaciones con diferentes fines sin la necesidad de replicar datos en varios servidores.

Para implementar esta arquitectura se eligió la plataforma J2EE, propuesta por Sun Microsystems, y basada en el lenguaje de programación Java (Sun Microsystem, 2000). Entre los beneficios proporcionados por esta plataforma al sistema que se describe se pueden destacar: la facilidad de mantenimiento al centralizar la aplicación en un solo lugar; no tener limites geográficos para distribuir la aplicación, debido a la utilización de browsers como clientes y a las facilidades de internacionalización de la arquitectura; la independencia del hardware; la gestión de conexiones a la BD mediante pooling de conexiones JDBC (Java Data Base Connectivity) del Container EJB (Entreprise JavaBeans); la flexibilidad para la elección entre distintos servidores de aplicación, tanto de código abierto como comerciales; la posibilidad de emplear numerosas librerías existentes (JNDI, Collections, Struts); la facilidad para la gestión de sesiones de usuarios mediante "Sessions EJB"; la reutilización de la lógica de negocio mediante "Bean Managed Persistence" de EJB y la clara separación entre presentación y lógica de negocio, permitiendo, en caso de ser necesario, separar estas capas geográficamente.

El gráfico que se muestra a continuación es una muestra reducida de los distintos componentes de nuestro sistema.

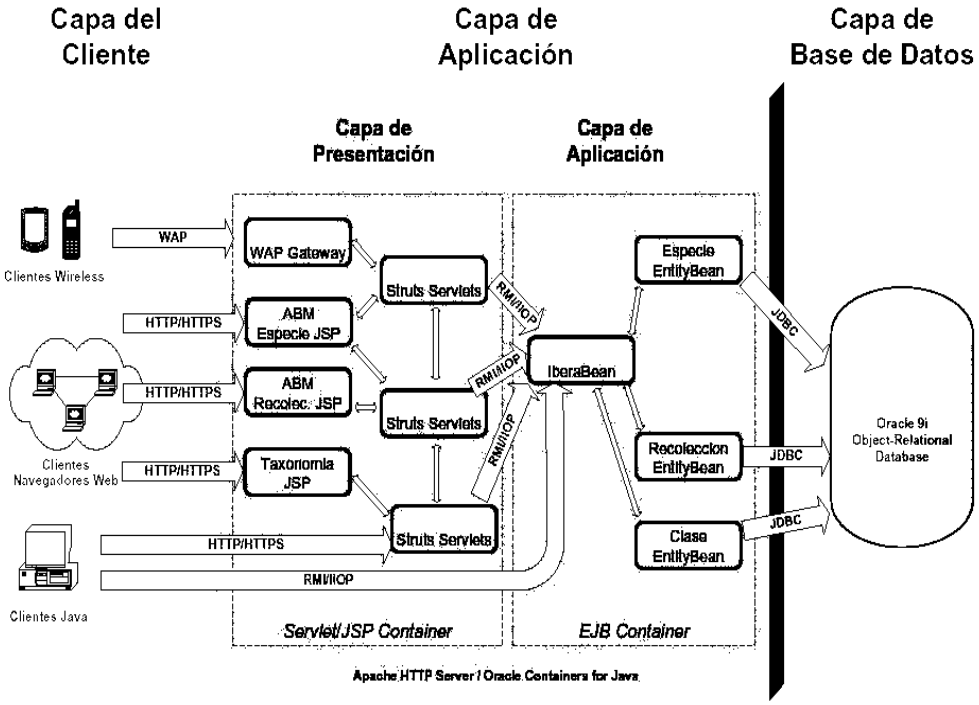

Fig. 2: Arquitectura de la aplicación

En el gráfico se puede distinguir la separación entre las diferentes capas, que interactúan con protocolos estándares y abiertos. El container EJB es el que soporta toda la lógica de la aplicación, manteniendo los objetos de la base de datos "mapeados" a objetos Java, y realizándose todas las operaciones de datos a través de ellos. La capa de presentación se adecua a los diferentes tipos de clientes: Web Browsers, clientes Java o clientes Wireless. El flujo de mensajes se realiza a través de Servlets e IberaBeans, que actúan como concentrador del flujo de mensajes de la aplicación.

La inclusión del comportamiento de los objetos (métodos) en la base de datos es otro punto clave de toma de decisión. Si bien tanto Java como PL/SQL están soportados como lenguajes para escribir métodos, Java requiere que éstos se encuentren ya definidos en clases Java almacenadas en la base de datos. En realidad, esto está diseñado para almacenar en la base de datos objetos que ya se encuentran previamente programados en Java, que no es nues-

tro caso. Por otro lado, con PL/SQL el código se puede incluir como método de los objetos en la base de datos, pero se pierden ventajas con la arquitectura del sistema propuesta como diseño. Por lo tanto, se optó por incluir el comportamiento dentro del código de los EJB y acceder a la base a través de sentencias SQL incluidas dentro de los mismos, vía JDBC, para independizar completamente la aplicación de la base de datos.

Para facilitar el desarrollo de la aplicación Web en Java utilizamos un framework basado en el modelo MVC (Model View Controller). Este framework provee un conjunto de clases y tag-libs que conforman el controlador, la integración con el modelo y facilitan la construcción de vistas. El mismo nos permitió estructurar una arquitectura donde quedó claramente dividida la lógica de negocio (Model), a través de los EJB antes mencionados, la presentación (View) a través del código JSP ("Java Server Pages") y el control de flujo de aplicaciones (Controller), el Servlet que en base a solicitudes del usuario decide qué función de la lógica de negocio se va a realizar y luego delega la presentación del resultado de la solicitud a la JSP correspondiente. Una de las ventajas que se obtiene de este modo es que la presentación y manipulación de datos se puede manejar de manera independiente, de modo tal que, por ejemplo, se pueden presentar los mismos datos en varios lenguajes y diseños dependiendo del usuario situado frente a la pantalla.

Para desarrollar la aplicación se utilizó el IDE Java JDeveloper 9i. Cabe aclarar que existen diversos IDEs en el mercado que permiten desarrollar este tipo de aplicaciones como es el caso del Forte for Java de Sun Microsystems. Pero elegimos JDeveloper con el objeto de mantener una homogeneidad en cuanto a fabricante de productos a nivel de aplicación y de base de datos. En cuanto a la generación de los objetos Java que se conectan con la Base de Datos se analizaron dos posibles estrategias:

- La primera consiste en mapear los objetos de la BD objeto-relacional mediante el framework para manejo de estructuras y colecciones proporcionadas por las clases JDBC de Oracle (JPublisher), que extienden las clases estándares de JDBC brindando ciertas características especiales (Sanko *et al.*, 2001). La ventaja es que este proceso se realiza automáticamente, pero aún cuando el código que se genera permite efectuar muchas operaciones directas, quedan muchos otras a resolver, por ejem-

plo: conexión a la BD, como reutilizar los objetos creados, como hacer "caching" de los mismos, etc.

- La segunda estrategia consiste en generar los EJB directamente en concordancia con los objetos generados en la base de datos. Este proceso es más lento y trabajoso, ya que debe crearse un EJB para cada objeto de la base. De todos modos se obtiene una mejora sustancial en la conexión con la base de datos, en la gestión de los datos del sistema, en la utilización de los beneficios aportados por la arquitectura J2EE y una mayor independencia del código generado de la BD utilizada, ya que cambiar la aplicación para que se conecte con otro motor de BD implica mínimos cambios en el código generado. También se pensó en emplear una estrategia mixta que combine a las dos anteriores, pero la limitación que se tiene es que la representación de los datos en las clases generadas por JPublisher no podían ser serializadas que era una condición requerida por el protocolo de comunicación de datos entre componentes de la arquitectura.

Se optó por utilizar la segunda estrategia porque cuenta con una mejor relación costo-beneficio a la hora de garantizar la persistencia de los objetos.

Consecuentemente, la aplicación final está constituida por:

- El **servidor de base de datos**, en donde se guardarán todos los datos de las especies del macrosistema Iberá.
- El **servidor de aplicaciones**, en donde estarán las aplicaciones internas y externas, y el que realizará los requerimientos correspondientes a la base de datos.
- Los **clientes web**, que será cualquier persona en Internet que ingrese a la aplicación para realizar consultas o cualquier persona de la Intranet, que desee realizar consultas y/o modificaciones en los datos. En ambos casos se utilizará un explorador de internet para acceder a la aplicación.

Conclusiones

La posibilidad de las Bases de Datos Objeto Relacionales de implementar diferentes tipos de datos y relaciones complejas nos permitió modelar e implementar la base de

datos de un modo más natural y directo, al menos para el caso de la aplicación en desarrollo, orientada al empleo de una arquitectura Intranet/Internet.

Como resultados de este trabajo se generaron tablas anidadas para las relaciones de composición, las estructuras provistas por el ORDBMS para herencia, y estructuras que contemplan referencias entre objetos para las asociaciones.

Si bien Oracle, con la versión 9i, ha logrado grandes avances en el desarrollo de las características orientadas a objetos en su motor de base de datos objeto-relacional, no desarrolló aún, una herramienta que integre todas las fases del ciclo de vida del software, principalmente las de modelado lógico y generación de la base, que permitan generar diseños que aprovechen todas las potencialidades de la Base de Datos. Tampoco existe en el mercado una herramienta ni una metodología que guíe el proceso de diseño e implementación de aplicaciones, por lo que estas tareas se deben hacer ad-hoc. Es por ello que, en este trabajo, para resolver el diseño se empleó una estrategia híbrida, basada en la tecnología Orientada a Objetos para la abstracción de clases y las relaciones entre las mismas, los cuales se encuentran plasmados en la Fig.1, y una estrategia basada en los objetivos de la aplicación, el tipo de las relaciones entre clases y las estructuras del ORDBMS disponibles para representar estas relaciones. Para esto último también se contemplaron aspectos tales como la facilidad de acceso a los datos y el mantenimiento de la aplicación.

Como conclusiones de este trabajo se puede decir que la implementación de la aplicación fue lo que presentó mayores dificultades. Se notó la falta de una metodología clara que ayude a discernir la administración y la distribución del comportamiento de los objetos en las diferentes capas. Tampoco esto se puede hacer de manera simple y directa. Esto es así independientemente que los métodos se programen tanto en Java, como en PL/SQL, o algún otro lenguaje. La oferta de herramientas basadas en tecnologías emergentes es amplia, pero éstas no han alcanzado una madurez suficiente y/o el soporte que brindan es limitado. Existen muchos aspectos que se deben abordar a la hora de discernir la generación, distribución y administración de los componentes de la aplicación: claridad en el código, facilidad de mantenimiento, buena performance, buena seguridad, conectividad, etc., que no son fáciles de satisfacer. La opción final para nuestro caso fue la utilización

de EJB para representar nuestra lógica del negocio y el Framework Struts para su presentación al usuario final, ya que con la combinación de estas herramientas pudimos, dada la clara separación de roles entre los distintos componentes, obtener un producto final de mayor calidad, más fácil de mantener y administrar. Cuestiones claves como la administración de conexiones, la escalabilidad de la aplicación, etc., pudieron ser delegadas al servidor de aplicaciones permitiéndole a los desarrolladores enfocarse en los aspectos específicos del sistema.

Como futuros trabajos de investigación se piensa ahondar en metodologías que permitan el diseño e implementación de aplicaciones similares, como experimentar en búsquedas multimedia no convencionales: imágenes por diversos parámetros como color, textura, forma; búsquedas por sonidos, etc.

Bibliografía

Cattel, R. and D. Barry, 2000. *The Object Data Standard*: *ODMG 3.0*. Morgan Kaufmann Publ. 300 p.

Dorsey, P., 1999. *The Promise of the Object-Relational Paradigm* [en línea]. June 1, 1999. Disponible en: <http://www.dulcian.com/papers/>.

Eisenberg, A. and J. Melton, 1999. SQL:1999, formerly known as SQL3. *ACM SIGMOD Record*, *28* (1): 131-138.

Elmasri, R. and S. Navathe, 2000. *Fundamentals of Database Systems*. Addison Wesley. 956 p.

Gietz, B., 2001. *Oracle 9i Application Developer's Guide-Object-Relational Features*, Release 1 (9.0.1). Part No. A88878-01.

Grimes, S., 1998. Modeling Object/Relational Databases. *DBMS*. *11* (4): 51-56.

Kovacs, C. and Van Bommel, P., 1998. Conceptual Modelling-Based Design of Object-Oriented Databases. *Information and Software Technology*, *40* (1):1-14.

McClure, S., 1997. *Object Database vs. Object Relational Database* [en línea]. IDC International Data Corporation. IDC Bulletin #14821E - August 1997. Disponible en: <http://www.cai.com/products/jasmine/analyst/idc/14821E.htm>.

Oracle Corporation, 2001. *Oracle9i Database-Oracle Technology Network* [en línea]. Disponible en: <http://technet.oracle.com/products/oracle9i/content.html>.

Oracle Designer, 1999. *Oracle Designer 6.0 Releases Notes*. Disponible en: <http://technet.oracle.com/products/oracle8/content.html>.

RATIONAL ROSE, 2000. *Rational Rose v2001: Visual Modeling, UML, Object-Oriented, Component-Based Development with Rational Rose* [en línea]. 14 December 2000. Disponible en: <http://www.rational.com/products/rose/>.

SANKO, M.; B. WRIGHT and T. PFAEFFLE, 2001. *Oracle 9i JDBC Developer's Guide and Reference*, Release 1 (9.0.1). Part No. A90211-01.

SUN MICROSYSTEM, 2000. *The JavaTM 2 Enterprise Edition Developer's Guide*. Version 1.2.1 [en línea]. May 2000. Disponible en: <http://java.sun.com/j2ee/>.

VELA, B.; J.M. CAVERO y E. MARCOS, 2001. Diseño de bases de datos objeto-relacionales con UML. *Actas de las Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del conocimiento* (JIISIC). Buenos Aires, Argentina: 59-68.

Recibido/Received/: 14-Abr-03
Aceptado/Accepted/: 07-Ago-03