Implementación de ODL para proporcionar persistencia en SGBDRs a sistemas replicados orientados a objetos*

J.E. ARMENDARIZ[1], J.J. ASTRAIN[1], A. CORDOBA[1], J. VILLADANGOS[2]
[1]Dpto. Matemática e Informática, [2]Dpto. Automática y Computación,
Universidad Pública de Navarra
Campus Arrosadía, 31006 Pamplona (Spain)

RESUMEN

Actualmente, el desarrollo de sistemas distribuidos está creciendo de manera considerable, principalmente los sistemas de replicación geográficamente distribuidos. Uno de los mayores problemas asociados con la replicación es garantizar la persistencia de los objetos manejados por las aplicaciones remotas. Para resolver este problema, este artículo presenta un traductor que permite almacenar el estado de losobjetos de manera persistente (sus atributos y relaciones univaluadas y multivaluadas) en SGBDRs generando las sentencias SQL correspondientes para almacenar, seleccionar y recuperar objetos. Este traductor nos permite diseñar un sistema de almacenamiento persistente de objetos que soporte a diferente algoritmos de consistencia, replicación y tolerancia a fallos de sistemas de replicación de objetos geográficamente distribuidos.

Palabras Claves: Persistencia, Replicación, Distribuido, RDBMS, ODL, Repositorio de Objetos, OQL.

1. INTRODUCCIÓN

La mayoría de las grandes empresas tienen sedes en diferentes ciudades, incluso en diferentes países. Obviamente, estas delegaciones necesitan acceder y almacenar persistentemente la información que es compartida entre todas ellas. Si no se es cuidadoso en el diseño de la red de la empresa, puede ocurrir que el fallo de una delegación implique la caída de todo el sistema. Para prevenir esta situación, es necesario el replicar y distribuir la información entre las diferentes sedes. En nuestro caso se ha adoptado esta solucióonn, definiendo los datos replicados en una sede como repositorio. Esta solución debe proprocionar una vista única y un acceso uniforme en todo el sistema para aplicaciones que manejan la información almacenada en los repositorios siguiendo el paradigma de la orientación a objetos.

Hoy en día, la programación orientada a objetos consituye uno de los paradigmas más importantes en cuanto a análisis, diseño e implementación. Por otro lado CORBA permite el desarrollo de aplicaciones orientadas a objeto en entornos distribuidos.

A menudo estas aplicaciones necesitan almacenar de manera persistente la información que manejan mediante sistemas gestores de bases de datos orientado a objeto (SGBDOO) [1,2]. Los fabricantes de SGBDOO han realizado un esfuerzo relamente importante a lo largo de los últimos años, para que puedan llegar a ser populares. Los más importantes constituyeron el Object Database Management Group, actualmente Object Data Management Group (ODMG) [1], que han desarrollado un modelo de datos común basado en el modelo OMG, con su lenguaje asociado de consulta llamado Object Query Language (OQL). El estándar ODMG-93 de bases de datos orientadas a objetos [1] tiene como objetivo garantizar la portabilidad entre sistemas, para lo cual se han definido tres interfaces: (i) Object Definition Language (ODL). Define el modelo de datos y es compatible con IDL, permite la definición de objetos complejos, las relaciones entre ellos y sus métodos asociados; (ii) OQL, permite hacer consultas sobre los objetos anteriores, envíod e mensajes a objetos y realizar reuniones y otras operaciones de tipo asociativo. Sus sintaxis es muy similar a SQL [3]; and, (iii) su conexión a través de C++ y SmallTalk, define los interfaces para programar una aplicación en estos lebguajes sobre una base de datos definida en ODL.

Las SGBDOO son especialmente ventajosas en aplicaciones CAD/CAM/CAE, herramientas CASE, etc. Los fabricantes deben enfrentarse al problema de un mercado restringido a las citadas aplicaciones. Este problema ha sido superado por medio de los siguientes tres elementos, no directamente relacionados con las tecnologías de las bases de datos: (i) el lenguaje Java, el ODMG v.2 [4] extiende los interfaces a Java; el Unified Modeling Language (UML), la mayor utilización de UML para el diselo y especificación de los sistemas de información una mayor comodidad para su inserción y gestión en SGBDOO que en SGBDR (un

* Este trabajo ha sido financiado por el Ministerio de Ciencia y Tecnología español como proyecto CICYT número: TIC2003-09420-CO2-O2.

enfoque 100% orientado a objetos, desde el principio hasta el final); y, (iii) el auge del desarrollo de aplicaciones distribuidas empleando CORBA.

Las bases de datos relacionales, y sus SGBDR asociados, han evolucionado para dar soprte a objetos y reglas, además de extender SQL con nuevos tipos y operaciones. Los módulos de almacenamiento persistente son una importante mejora para la productividad y seguridad de las aplicaciones. Éstos mejoran la eficiencia en los entornos cliente-servidor. El nuevo estándar SQL-99 [5] ha tratado de converger al lenguaje ODMG y a OQL. En esta nueva evolución han influido decisvamente por un lado, JDBC y SQLJ [6] y, por otro lado, las bases de datos deductivas no han obtenido el resultado esperado.

La mayoría de las aplicaciones más importantes que gestionan objetos están construidos sobre SGBDRs. A pesar de que éstos no han adquirido el desarrollo deseado para poder incluir los nuevos conceptos asociados, sí que ha quedado patente que son más estables y eficientes que los SGBDOOs. De hecho muchas empresas siguen utilizando SGBDRs, además de que muchas aplicaciones orientadas a objetos emplean como mecanismo de almacenamiento persistente SGBDRs [7].

Las técnicas de propagación de actualización en entornos replicados pueden variar desde técnicas asíncronas, donde las actualizaciones se propoagan cuando sea posible o periódicamente, hasta las totalmente síncronas, en donde el orden de actualización de las réplicas es total [8]. Hay diferentes aproximaciones a las técnicas de replicación de datos. Así, en [9] se propone un sistema de replicación implementado en Java mediante RMI. Este sistema proporciona comunicación entre un grupo de servidores. El sistema Aroma [10], también ofrece replicación de objetos utilizando RMI, y su diferencia principal con el anterior es el empleo de mecanismos de intercepción para ofrecer la replicación de objetos de modo transparente. También existen soluciones más genéricas mediante CORBA por medio de técnicas de agrupamiento de objetos [11]. Estos trabajos adolecen de mecanismos de almacenamiento persistente para las instancias creadas.

El objetivo de este trabajo es proporcionar un soporte de almacenamiento persistente en un entorno de replicación llamado COPLA (*Common Object Programmer Library Access*) [12]. Este sistema proporciona a los desarrolladores de aplicaciones una única capa que engloba y simplifica el acceso a un repositorio de objetos compartido, cuyas caraacterísticas principales es su unicidad consistencia y su total disponibilidad. La novedad de este sistema, en cuanto a nuestro conocimiento, es uqe se proporiona un servicio de almacenamiento persistente de objetos sobre una arquitectura replicada, llamada *Uniform Data Store* (UDS), garantizando un comportamiento transaccional y la posibilidad de acceder al estado de los objetos bien a través de consultas específicas o por la invocación directa de métodos sobre los objetos. Otras características son la posibilidad de recuperar los sitios cuando se producen fallos de parada o la escalabilidad de las aplicaciones desarrolladas en COPLA.

El principal objetivo del sistema de almacenamiento persistente es proporcionar una única capa de software que aisle de los detalles de almacenamiento persistente al resto de componentes. Como medio de almacenamiento se emplea un SGBDR, debido a su madurez tecnológica y a la familiaridad que tienen con ella tanto los administradores de los sistemas como los usuarios finales, además de permitir la migración de la información almacenada en bases de datos a las aplicaciones cliente orientadas a objeto.

Es importante destacar que la arquitectura proporciona un mecanismo de replicación siguiendo el paradigma de la orientación a objetos mientras que el sistema de persistencia emplea el modelo entidad-relación [13,14,15]. Este desajuste debe ser solventado mediante la especificación de un lenguaje de definición de objetos, llamado *Globdata Object Definition Language* (GODL), que establece como deben ser definidas las clases, los atributos y las relaciones (univaluadas y multivaluadas), así como su mapeo relacional sobre el SGBDR. Se debe definir además como se representa el estado del objeto almacenado persistentemente en la aplicación.

Este trabajo presenta un traductor que hace posible el almacenamiento y la recuperación del estado de los objetos en SGBDRs. Comenzando por la definición de un módulo GODL que contendrá la definición de los objetos, el traductor genera las sentencias SQL que permite generar la estructura de tablas adecuada para poder almacenar, actualizar o borrar el estado de los objetos pertenecientes al módulo actual.

El traductor debe resolver dos problemas esenciales para asegurar el almacenamiento persistente en SGBDRs: (i) La gestión del mapeo, es decir, cómo se almacenan los objetos en la base de datos, geestionar lo satributos, las relaciones, la herencia, etc.; (ii) la gestión de los metadatos, como se debe emplear una capa de software para aislar los detalles de la implementación en el SGBDR, hay que definir la meta-información que facilite al UDS para localizar correctamente los objetos. El resto del artículo esta estructurado de la siguiente manera, la Sección 2 específica los trabajos relacionados acerca del almacenamiento persistente de objetos. La arquitectura del sistema se explica brevemente en la Sección 3. La Sección 4 introduce la gramática GODL. La arquitectura del traductor se muestra en la Sección 5. El mapeo de cada uno de los elementos de GODL sobre el SGBDR se describe en la Sección 6. El conjunto de metadatos necesarios para su correcto funcionamiento están contenidos en la Sección 7. Este trabajo finaliza con una sección dedicada a las conclusiones.

2. TRABAJOS RELACIONADOS

El diseño y la implementación de un sistema de almacenamiento persistente se ha tratado en [7]. Tratan el almacenamiento del estado de los objetos en medios no

volátiles como discos duros o bases de datos, relacionales u orientadas a objetos. Nuestra implementación sigue varios aspectos de los trabajos anteriores. Existen implementaciones para entornos concurrentes [16,17]. Otros emplean CORBA para garantizar la persistencia de los objetos [18]. En [19] se trata la persistencia en agentes móviles, mientras que en [20] se describe la persistencia de objetos Java para aplicaciones de *intranet* en empresas o aplicaciones web específicas.

3. ARQUITECTURA COPLA

El traductor que se propone en este trabajo va a ser empleado por un componente software que aisla al resto de componentes COPLA de la persistencia en el SGBDR. La arquitectura COPLA gestiona los fallos del sistema y el rendimiento de la aplicación. Ambos aspectos están estrechamente relacionados, ya que la implementación de técnicas más robustas llevan a un descenso en el rendimiento. Este sistema soporta fallos en el repositorio, los cuales será capaz de recuperar, obviamente el fallo de un nodo afectará a la disponibilidad de los datos. La Figura 1 muestra una representación de la arquitectura del sistema.

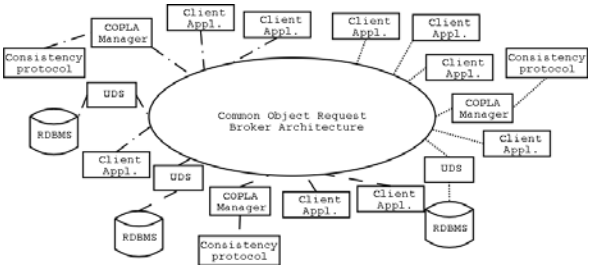

Figura 1: Arquitectura del sistema.

Varios nodos, llamados COPLA *manager*s, dividen el sistema en varias áreas. Éstos sirven a aplicaciones definidas en sus respectivas áreas. Por otro lado, las palicaciones acceden a su *manager* más "cercano", esto es, el *manager* responsable del área. Todas las aplicaciones poseen facilidades para poder cambiar de gestor, si el suyo falla. Un *manager* es responsable de un UDS, es decir, del sistema de almacenamiento persistente. El UDS proporciona una visión lógica de un único repositorio de objetos centralizado para todo COPLA.

Los clientes interactúan con el resto del sistema mediante un conjunto de clases, agrupadas en librerías y llamadas librerías COPLA. Estas librerías son las encargadas de representar el estado del objeto en el lado del cliente. Cada librería interactúa con el *manager* correspondiente que le proporciona acceso al UDS para poder trabajar correctamente.

Cada UDS contiene todos los módulos definidos en el sistema. Lo que no está garantizado, porque depende del protocolo de consistencia empleado [12] es que sea la última versión del objeto en COPLA. El protocolo de consistencia realiza las tareas de coordinación con el resto de *manager*s para garantizar a los clientes un acceso consistente al repositorio. Cada transacción iniciada por una aplicación cliente se realiza sobre el UDS del *manager* local. Si algún *manager* falla, el sistema continúa su actividad hasta que sea factible el continuar trabajando, asegurando siempre la consistencia e integridad de los datos. Las aplicaciones tienen un registro donde tienen almacenado todos los gestores del sistema, por tanto son capaces de continuar trabajando mediante la conexión a un nuevo *manager*.

El objetivo de COPLA es proporcionar acceso a un único repositorio lógico de objetos, el cual estará compuesto por varias copias físicas (veáse Figura 2). Las aplicaciones están definidas mediante GODL. Este lenguaje caracteriza un repositorio de objetos, o módulo. Cada módulo está compuesto por clases, donde cada una de ellas está compuesta por atributos, literales u objetos, y relaciones con otras clases. Estos objetos pueden ser accedidos por medio de consultas hechas en GOQL. Las aplicaciones desarrolladas en COPLA cumplen las siguientes propiedades: (i) alto grado de localidad, significa que la inetersección del conjunto de los objetos escritos en cada área es vacía (los conflictos, si ocurren, se dan en el mismo nodo); y, (ii) algunas áreas sólo necesitan la versión del objeto para mantener la consistencia de los datos, no es necesario transferir el objeto, sólo si la versión es válida.

Como conclusión final diremos que COPLA ofrece un repositorio único virtual, esto es un sistema de replicación de objetos persistente implementado en Java y que se comunica por medio de CORBA. De este modo los desarrolladores de aplicaciones ven que acceden a un único repositorio repositorio de objetos centralizado gracias a los protocolos de consistencia implementados [12]. Los objetos definidos en un módulo son almacenados persistentemente en un SGBDR, lo cual implica el diseño de un medio de traducción del modelo orientado a objetos al modelo entidad-relación.

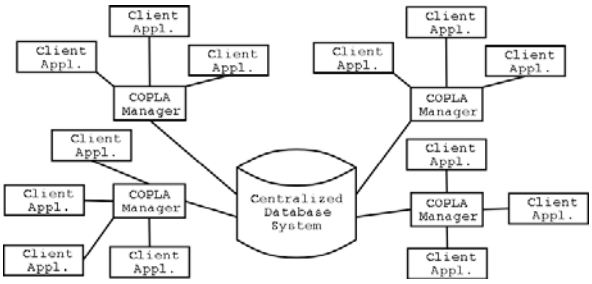

Figura 2: Percepción del sistema por parte del usuario.

4. GRAMÁTICA GODL

GODL es un subconjunto del estándar *Object Definition Language* [1]. La principal restricción es que no se permiten colecciones anidadas, ya que no es uno de los principales objetivos de nuestro trabajo. En esta sección se describirá la

gramática GODL, cuyos principales elementos son los módulos, las clase, los atributos y las relaciones.

Las clases están definidas dentro de un módulo, en ella se especifican un conjunto de atributos y relaciones, así como la posible herencia de otra clase previemente definida. Los atributos, los cuales pueden ser multivaluados (*arrays*, conjuntos, bolsas y listas), son como propiedades que pueden ser de tipos básicos, *strings* o instancias de otras clases. Las relaciones son asociaciones entre instancia de la misma o diferentes clases, univaluadas o multivaluadas. En la Figura 3 se muestra un ejemplo de un esquema GODL.

```
1  module mod_name {
2   class class_name1 {
3     attribute long id_number;
4     attribute string<10> name;
5     attribute class_name3 att_class3;
6     attribute set<class_name2> att_class2;
7     ....
8     relationship type1 class_name2 identifier1
9     inverse class_name2::identifier2;
10    ....
11  };
12  class class_name2 {
13    ....
14    relationship type2 class_name1 identifier2
15    inverse class_name1::identifier1;
16    ....
17  };
18  class class_name3 extends class_name1{
19    ....
20  };
21 }
```

Figura 3: Módulo GODL genérico.

5. TRADUCTOR GODL

El traductor GODL es un compilador cuya entrada es un esquema GODL y su salida es un conjunto de sentencias SQL que permite almacenar el estado de los objetos en un SGBDR. Los bloques que constituyen el traductor son: un analizador léxico y un analizador sintáctico. El analizador sintáctico, toma como entrada los *tokens* enviados por el analizador léxico, para ir construyendo la correspondiente tabla de símbolos. Después genera el mapeo correspondiente al módulo compilado. El traductor es un compilador de dos etapas. En la primera, el fichero que contiene el módulo GODL es analizado sintácticamente, y simultáneamente se va generando la tabla de símbolos. En la segunda etapa, se generan las sentencias SQL correspondientes. JLex y CUP son el analizador léxico y el generador de *parser*s empleados para la realización del compilador.

Inicialmente, el traductor genera un fichero de extensión lex que contiene el istado de las palabras reservadas de GODL, además de otro (con la extensión cup) que contiene la gramática GODL. JLex va obteniendo token del módulo que va enviando al analizador sintáctico, el CUP. Éste debe encontrar la correspondiente regla de producción gramatical para estos *tokens* y ejecutar el código correspondiente a estos segmentos. Si todo funciona correctamente se va construyendo la tabla de símbolos, en caso contrario se lanzará una excepción. Un módulo independiente, denominado *mapping*, extrae la información de la tabla de símbolos y genera el correspondiente mapeo, las sentencias SQL y los *triggers* y funciones para mantener la integridad referencial y la gestión de las colecciones.

6. MAPEO SOBRE SGBDR

El propósito de nuestra propuesta de mapeo es almacenar el estado de los objetos definidos en un módulo, como el descrito en la Figura 3, en un SGBDR. En esta sección se explicará la traducción del módulo a su equivalente en el modelo entidad-relación. En el sistema se emplea un identificador de objeto (GOID) para poder localizar a un objeto dentro de las tablas de la base de datos, que se emplea como clave extranjera de una tabla GOIDS que contiene todos los identifcadores de los objetos del repositorio.

Mapeo de atributos univaluados

El mapeo de atributos univaluados (líneas 3-5 de la Figura 3) se realiza de distinta manera dependiendo de si son literales u objetos como se muestra en la Figura 4. Para cada clase definida en el módulo se genera una tabla que contiene los siguientes campos: `goid`, que identifica al objeto y hace referencia a la tabla; y, tantas columnas como atributos literales o de tipo string (estos son de longitud fija y definidos como VARCHAR(length)). Los tipos válidos de literales son: `char`, `boolean`, `octet`, `short`, `unsigned short`, `long`, `long long`, `float`, `double`, `date`, `time` and `timestamp`. Si existen atributos de tipo `unsigned short` o enumerado se generan las restricciones correspondientes sobre los campos. GODL permite la definición de atributos clave. Los atributos objeto univaluados se mapean en tablas separadas, una por atributo, el nombre de la tabla se construye del siguiente modo: el nombre de la clase donde el atributo ha sido definido, seguido de la cadena "`_O_`" y finalizado por el nombre del atributo. En ella se definen dos campos, el primero contiene el GOID del objeto de la clase y el segundo el GOID del objeto agregado.

```
CREATE TABLE CLASS_NAME(OBJ_ID VARCHAR REFERENCES
AGGREG(OBJ_ID) ON UPDATE CASCADE ON DELETE CASCADE,
LITERAL_TYPE_1 <RDBMS_TYPE>, ... LITERAL_TYPE_n
<RDBMS_TYPE>);

CREATE TABLE CLASS_NAME_O_IDENT(CLASS_NAME VARCHAR
REFERENCES CLASS_NAME(GOID) ON UPDATE CASCADE ON DELETE
CASCADE, IDENT VARCHAR REFERENCES CLASS_NAME_2(GOID) ON
UPDATE CASCADE ON DELETE CASCADE, PRIMARY
KEY(CLASS_NAME), UNIQUE(IDENT));
```

Figura 4: Mapeo de atributos univaluados.

Mapeo de atributos multivaluados

GODL permite la definición de los siguientes tipos de colección: `BAG`, `SET` y `LIST`, (ver la línea 6 de la Figura 3). Todas las colecciones se almacenan en tablas diferentes, como se muestra en la Figura 5. El nombre de la tabla es el nombre de la clase seguido de "`_COLLECTION_`" y el nombre del atributo. Las bolsas no añaden restricciones adicionales. Las primeras dos restricciones son para el caso de la definición de conjuntos, para garantizar que no se introducen elementos repetidos. La última columna aparaece cuando se define una lista pra almacenar la posición en la que está insertado el elemento.

Otro atributo multivaluado considerado en GODL es el ARRAY, que se mapea en una tabla distinta que contiene al menos tres columnas: La primera contiene el propietario del array, la segunda el valor de esa posición del array y el resto serán las correspondientes a la dimensión del array, para identificar la posición del valor dentro del array.

```
CREATE TABLE CLASSNAME_COLLECTION_IDENT(CLASSNAME
VARCHAR REFERENCES CLASSNAME(GOID) ON UPDATE CASCADE ON
DELETE CASCADE, [IDENT RDBMS_TYPE  NOT NULL,| IDENT
VARCHAR NOT NULL REFERENCES CLASS_NAME2 ON UPDATE
CASCADE ON DELETE CASCADE], [UNIQUE(IDENT), PRIMARY
KEY(CLASSNAME, IDENT)]      [CARDINAL_NUMBER INT4,
CHECK(CARDINAL_NUMBER>=0)]);

CREATE TABLE CLASSNAME_A_IDENT(CLASSNAME VARCHAR
REFERENCES CLASSNAME(GOID) ON UPDATE CASCADE ON DELETE
CASCADE, [IDENT RDBMS_TYPE  NOT NULL,| IDENT VARCHAR NOT
NULL REFERENCES CLASS_NAME2 ON UPDATE CASCADE ON DELETE
CASCADE], DIMENSION_1 INT4, CHECK(DIMENSION_1>=0 AND
DIMENSION_1< n), PRIMARY KEY(CLASSNAME, DIMENSION_1));
```

Figura 5: Mapeo de atributos multivaluados.

Mapeo de asociaciones

Este tipo de mapeo es más sencillo que el de los atributos, ya que las relaciones pueden verse como asociaciones dentro del modelo entidad-relación. Asumimos que cada relacion GODL es opcional pudiendo ser ésta univaluada o multivaluada (la declaración de una relación se muestra en la Figura 3 en la líneas 8-9 y 14-15). Los tipos de relaciones válidas se incluyen en la Figura 6. Si en una relación se define en alguno (o en ambos) de los sentidos una lista, el compilador genera los *triggers* y las funciones adecuados para su gestión. El nombre de la tabla está compuesto por la clase en que primero aparezca la relación, los nombres de los campos son los de las relaciones en ambos sentidos, existirán campos adicionales si alguno de los sentidos contiene una lista.

```
CREATE TABLE CLASS_NAME1_IDENT1(CLASS_NAME1_IDENT1
VARCHAR REFERENCES CLASS_NAME1(goid) ON UPDATE CASCADE
ON DELETE CASCADE, CLASS_NAME2_IDENT2 VARCHAR REFERENCES
CLASS_NAME2(goid) ON UPDATE CASCADE ON DELETE CASCADE);
Additional clauses added depending on:
1-1.- To assure that one object is only related with
another:
  PRIMARY KEY(CLASS_NAME1_IDENT1),
UNIQUE(CLASS_NAME2_IDENT2)
1-SET.- One object can be related to several non-
repeated objs.
  PRIMARY KEY(CLASS_NAME1_IDENT1, CLASS_NAME2_IDENT2),
UNIQUE(CLASS_NAME1_IDENT1)
1-LIST.- A new column is added to maintain the order.
  IDENT2_CARDINAL INT4, CHECK(IDENT2_CARDINAL >=0),
UNIQUE(CLASS_NAME1_IDENT1)
SET-SET.- A set of objects can be related to non-
repeated objects.
  PRIMARY KEY(CLASS_NAME1_IDENT1, CLASS_NAME2_IDENT2)
SET-LIST.- The following column is added.
  IDENT2_CARDINAL INT4, CHECK(IDENT2_CARDINAL >=0),
UNIQUE(CLASS_NAME1_IDENT1, CLASS_NAME2_IDENT2)
BAG-BAG.- Nothing is added.
BAG-LIST.- A new column is added.
  IDENT2_CARDINAL INT4, CHECK(IDENT2_CARDINAL >=0)
LIST-LIST.- Two columns are added.
  IDENT1_CARDINAL INT4, CHECK(IDENT1_CARDINAL >=0),
IDENT2_CARDINAL INT4, CHECK(IDENT2_CARDINAL >=0)
```

Figura 6: Mapeo de asociaciones.

7. METADATOS

Los metadatos generados por el compilador sirven para localizar los diferentes elementos que componen un esquema GODL. Contiene información acerca de la herencia, los atributos y las relaciones definidas para un módulo. Toda esta información es necesaria para que la capa que dé soporte a la persistencia de los objetos pueda llevar a cabo correctamente todas las operaciones sobre el estado de un objeto. Los metadatos se desglosan en cuatro tablas: GOIDS; hierarchy; attributes; y, relationships (veáse la Figura 7).

Comenzaremos por la tabla GOIDS. Esta tabla contiene todos los objetos definidos en el repositorio y lleva el control de la agregación de los objetos. Contiene tres campos: el primero para el identificador del objeto, el segundo es para el identificador del objeto al que está agregado y el último contiene la multiplicidad de la citada agregación.

```
CREATE TABLE GOIDS(GOID VARCHAR NOT NULL PRIMARY KEY,
IS_AGGREGATED_TO VARCHAR REFERENCES AGGREG(GOID) ON
UPDATE CASCADE ON DELETE CASCADE, COUNTER_REFERENCES
INT4 DEFAULT 0, CHECK(COUNTER_REFERENCES>=0));

CREATE TABLE HIERARCHY(CLASS VARCHAR(14), SUPERCLASS
VARCHAR(14));

CREATE TABLE ATTRIBUTES(ATTRIBUTE_KEY VARCHAR,
CLASS_NAME VARCHAR, COLLECTION_TYPE VARCHAR(5), TYPE
VARCHAR, ARRAY_SIZE VARCHAR, PRIMARY KEY(ATTRIBUTE_KEY,
CLASS_NAME));

CREATE TABLE RELATIONSHIPS(RELATIONSHIP_KEY VARCHAR,
CLASS_NAME VARCHAR,
INVERSE_NAME VARCHAR, INVERSE_CLASS_NAME VARCHAR,
COLLECTION_TYPE VARCHAR(4), TABLE_NAME VARCHAR, PRIMARY
KEY(RELATIONSHIP_KEY, CLASS_NAME));
```

Figura 7: Mapeo de metadatos.

La tabla hierarchy contiene tantas entradas como clases definidas en el módulo, siempre que no se haya definido ninguna herencia. En caso contrario habrá tantos registros de una clase como profundidad tenga su árbol jerárquico. Todos los atributos definidos están incluidos en la tabla attributes, que contiene: el nombre del atributo, la clase donde se ha definido; su tipo y cardinalidad. Lo mismo puede aplicarse a las relaciones, en la tabla relationships se incluyen todas las relaciones y contiene los siguientes campos: el nombre de la relación; la clase donde ha sido definida; el nombre de la relación inversa; el nombre de la tabla; y el tipo de ambas relaciones.

8. CONCLUSIONES

En este artículo se ha presentado un traductor que permite a una aplicación orientada a objetos almacenar persistentemente el estado de los objetos definidos en ella sobre un SGBDR. El traductor se ha implementado con Java, JLex y CUP. Para ello se ha definido el lenguaje GODL, un subconjunto del estádar propueto por ODMG, que establece cómo deben ser definidas las aplicaciones en el entorno replicado, a las que hemos llamdo módulos. Una de las principales diferencias es que no permitimos la definición de colecciones anidadas.

Un módulo se mapea en una nueva base de datos, las clases con todos sus atributos literales univaluados se mapean en una tabla. El resto de atributos, así como las rlaciones se mapean en tablas diferentes. La gestión de colecciones se realiza mediante mecanismos internos del SGBDR, como son claves primarias *triggers* y funciones. En la literatura no hemos podido (o sabido) encontrar mapeos para poder establecer comparaciones con los tipos multivaluados y las relaciones. El traductor genera meta-información adicional para que una capa de software instalada por encima del SGBDR sea capaz de poder realizar operaciones sobre el estado de los objetos. Además en el proceso de traducción se ha seguido el estándar SQL para hacerlo lo más independiente posible del SGBDR que se vaya a emplear.

9. REFERENCIAS

[1] R.G.G. Cattell y D.K. Barry. **The Object Data Standard: ODMG 3.0**. Morgan Kauffmann Publishers. USA, 2000.

[2] A. de Miguel y M. Piattini. **Fundamentos y modelos de Bases de Datos**. Ediciones Ra-Ma, Segunda edición. Madrid, España, 1999.

[3] C.J. Date y H. Darwen. **A Guide to the SQL Standard: A User's Guide to the Standard Relational Language SQL**. Addison-Wesley Publishers. USA, 1997.

[4] R.G. Cattell y D.K. Barry. **The Object Database Standard ODMG 2.0**. Morgan Kauffmann Publishers. USA, 1997.

[5] P. Gulutzan y T. Pelzer. **SQL-99 Complete, Really**. CMP Books. USA, Marzo 1999.

[6] Welcome to www.SQLj.org! Your Online Resource For Information About SQLj: http://www.sqlj.org

[7] J. Chen, Q.-M. Huang y A.S.M. Sajeev. "Interoperability Between Object-Oriented Programming and Relational Systems". **In Proceedings of the 6th International Hong Kong Database Workshop**, Hong Kong, Marzo 1995.

[8] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme y G. Alonso. "Understanding Replication in Databases and Distributed Systems". **In Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000)**, Taipei, Taiwan, R.O.C., págs. 264-274, April 2000.

[9] A. Baratloo, P.E. Chung, Y. Huang, S. Rangarajan y S. Yajnik. "Filterfresh: Hot Replication of Java RMI Server Objects", **In Proceedings of the 4th USENIX Conference on Object-Oriented Technology and Systems (COOTS)**, New Mexico, USA, Abril, 1998.

[10] N. Narasimhan, L.E. Moser y P.M. Melliar-Smith. "Transparent Consistent Replication of Java RMI Objects", **In Proceedings of International Symposium on Distributed Objects and Applications**, Antwerp, Belgium, Septiembre 2000.

[11] P. Felber, R. Guerraoui y A. Schiper. "Replication of CORBA objects". **In S. Krakowiak and S.K. Shrivastava, editors, Recent Advances in Distributed Systems, volume 1752 of LNCS**. Springer Verlag, 2000.

[12] F.D. Muñoz-Escoi, J.M. Bernabéu-Aubán and L. Irún-Briz. "Globdata: Consistency Protocols for Replicated Architectures", **Youth Forum In Computer Science and Engineering**, Valencia, España, Noviembre 2001.

[13] W.G. Keller. "Mapping objects to tables: A pattern language", **Proc. of the 1997 European Pattern Languages of Programming Conference** (Siemens Technical Report 120/SW1/FB), Irsee, Germany, 1997.

[14] S. Ambler. **Agile Database Techniques - Effective Strategies for the Agile Software Developer**. John Wiley & Sons. USA 2003.

[15] K. Brown y B.G. Whitenack. "Crossing Chasms, A Pattern Language for Object-RDBMS Integration", **Editors: J.M. Vlissides, J.O. Coplien and N.L. Kerth in Pattern Languages of Program Design 2**, Addison-Wesley, USA 1996.

[16] M.P. Herlihy y J.M. Wing. "Linearizablility: A Correctness Condition for Concurrent Objects", **ACM Trans. Programming Languages**, vol. 12, no. 3, págs. 464-492, Julio 1990.

[17] P. Ferreira, M. Shapiro y otros. "PerDiS: design, implementation, and use of a PERsistent DIstributed Store", **Recent Advances in Distributed Systems, Lecture Notes in Computer Science**, Springer-Verlag, vol. 1752, no. 18, págs. 427-452, Febrero 2002.

[18] C. Mayers. "ANSAwise - Persistent data storage with CORBA", Citrix Systems, Cambridge, UK, Abril 1996.

[19] M. Mira da Silva y M. Atkinson. "Combining mobile agents with persistent systems: opportunities and challenges". **2nd (ECOOP) Workshop on Mobile Object Systems**, Linz, Austria, págs. 36-40, 1996.

[20] Bringing a new dimension to Java through easy access enterprise data: http://www.javera.com