DISEÑO E IMPLANTACION DE APLICACIONES DISTRIBUIDAS SOBRE BASES DE DATOS RELACIONALES: UN ENFOQUE ORIENTADO A OBJETOS

José Abásolo Prieto, Martha Inés Ortiz

Universidad de los Andes
Depto. de Sistemas y Computación
AA 4976, Santafé de Bogotá, COLOMBIA
Fax. (571) 2841570
e-mail: jabasolo@uniandes.edu.co (internet)

RESUMEN

Este documento contiene una propuesta para implantar aplicaciones distribuidas sobre redes de computadores cuyos sitios servidores de datos trabajen con bases de datos relacionales, y más específicamente con Oracle7. La propuesta incluye una metodología para diseñar la distribución de datos y procesos a partir de los resultados del análisis estructurado, incorporando conceptos de la orientación a objetos. Como complemento, se presentan desarrollos de programación que ofrecen una interfaz de alto nivel sobre la base de datos a desarrolladores de aplicaciones y a cierto tipo de usuarios finales.

AREAS: *bases de datos, sistemas distribuidos.*

1. INTRODUCCION.

En libros clásicos sobre bases de datos distribuidas [8][16] se presentan metodologías de diseño de distribución de bases de datos relacionales, basadas en la fragmentación y localización de tablas identificadas en una etapa previa de análisis. Aplicar esas metodologías a los cientos de tablas que manejan normalmente las organizaciones sería una tarea muy dispendiosa. Es por ello que, en el marco de un proyecto de investigación conjunto entre la Universidad de los Andes y Oracle Colombia, se armó una metodología de distribución de datos y aplicaciones que, conservando las ideas básicas de las metodologías clásicas, disminuye notablemente la complejidad del problema, ya que introduce el concepto de objetos de distribución, a los cuales se aplica la fragmentación, y cuyos fragmentos son luego localizados. Esta metodología se presenta en el capítulo 2.

La distribución de los datos añade un nivel adicional de complejidad a las aplicaciones. Aunque algunos autores hablan de una transparencia total a la distribución como algo ideal, pensamos que ésta no siempre es posible: en algunas áreas de negocios que pueden funcionar en modo distribuido, tales como la banca, los problemas de comunicaciones pueden afectar la semántica de ciertas operaciones. Un ejemplo de lo anterior sería la transferencia de dinero de una cuenta a otra: si la cuenta origen está disponible y la destino no, la operación debe realizarse parcialmente, dejando los registros y previsiones para completarla o anularla cuando se restablezcan las comunicaciones. Entonces, la lógica puede cambiar dependiendo de si los datos estan centralizados o distribuidos. Es por lo tanto importante ofrecer facilidades para el manejo de la distribución, sin ocultarla.

Considerando lo anterior, en el marco del proyecto Uniandes-Oracle se desarrolló un api que maneja un catálogo de distribución y que ofrece a los programadores de aplicaciones una visión de objetos fragmentados dispersos entre los sitios de una red. El api permite ubicar los objetos, utilizando la información del catálogo. Este último es una meta-base de datos, también distribuida. Api y catálogo son descritos en el capítulo 3, al igual que una herramienta de creación y mantenimiento de la base de datos distribuida.

El tratamiento de consultas generales distribuidas es tema de investigación. En el proyecto Uniandes-Oracle se está trabajando en el tratamiento asincrónico de consultas que se dirigen a muchos sitios y cuya respuesta es la union de los resultados obtenidos al ejecutarla en cada uno de ellos. En el capítulo 4 se describe este trabajo.

Tanto la metodología como los programas mencionados en este artículo están siendo sometidos a prueba en un proyecto de distribución de datos y aplicaciones de un importante Banco comercial colombiano. De ello se habla en las conclusiones.

1.1. Relación con trabajos anteriores.

Una versión preliminar de esta metodología, sin el enfoque de objetos, fué presentada en [4]. En dicha versión se hacía énfasis en el concepto de replicación, y para soportarla se desarrolló un api que permite insertar, consultar, modificar y suprimir tuplas, dada su llave primaria. El api da transparecia a la fragmentación, maneja diferentes esquemas de replicación y fue escrito en Pro*C. Al intentar aplicar dicha metodología al problema real de un Banco comercial, con más de 1500 tablas, se sintió la necesidad de simplificar la tarea, incorporándose entonces el concepto de objeto. Nacieron así la metodología y el api presentados en este artículo.

2. METODOLOGIA DE DISTRIBUCION.

La metodología planteada combina aspectos del análisis y diseño estructurado con algunas características de la orientación a objetos, produciendo como resultado unas aplicaciones distribuidas que operan sobre una base de datos relacional tambien distribuida.

La presentación de la metodología se hará en el siguiente orden:
(i) Arquitectura del sistema distribuido al que se desea llegar siguiendo la metodología propuesta.
(ii) Pasos de la metodología.

2.1 Arquitectura del sistema distribuido.

El sistema consta de una o mas aplicaciones, distribuidas sobre un conjunto de sitios. Ejemplos de aplicaciones, en un sistema bancario, serían: aplicación de cuentas corrientes y de ahorro, aplicación tarjeta de crédito, aplicación clientes, aplicación manejo de oficinas, etc.

A continuación se presentan los componentes de una aplicación cualquiera y la composición de cada sitio.

2.1.1. Componentes de una aplicación.

Cada aplicación está compuesta por un conjunto de tipos de objeto y un conjunto de operaciones que pueden ser invocadas por los usuarios. Estas operaciones invocan a su vez los procedimientos o servicios de los primeros, los cuales encapsulan sus datos.

Tomando como ejemplo una aplicación de cuentas corrientes y de ahorro, sus tipos de objeto podrían ser el objeto cuenta y el objeto tarjeta débito. El objeto cuenta podría ofrecer servicios tales como devolver saldo, hacer crédito y hacer débito. Algunas operaciones de la aplicación podrían ser consultar saldo, hacer transferencia, hacer consignación y cobrar cheque. Nótese que, por ejemplo,la operación de transferencia invoca los servicios débito y crédito de las cuentas involucradas. Cobrar cheque invoca un débito. Etc.

Tipos de objeto y operaciones de una aplicación son descritos a continuación.

2.1.1.1 Tipos de objeto.

Un tipo de objeto es un conjunto de información relacionada que puede tratarse como una unidad dentro del sistema distribuido para efectos de localización. Cada tipo de objeto tiene una estructura que describe el estado de sus objetos y un conjunto de procedimientos que constituyen su comportamiento.

2.1.1.1.1 Estructura de un tipo de objeto.

La estructura de un tipo de objeto se modela como una tabla no normalizada (no en 1FN), donde cada tupla describe un objeto. Estas últimas son una agregación de:

- Uno o mas campos atómicos, que constituyen la llave primaria del objeto.
- Cero o mas campos atómicos, que corresponden a los atributos monovalores del objeto.
- Cero o mas campos complejos (subtablas), que corresponden a atributos multivalores del objeto.
- Cero o mas campos, atómicos o complejos, que describen relaciones con otros objetos del mismo o diferente tipo. Las relaciones se representan guardando la llave primaria del objeto relacionado y, en algunos casos, atributos de la relación o del otro objeto que se replican por eficiencia.

Las tablas no normalizadas se particionan en uno o mas fragmentos verticales. Los fragmentos verticales se particionan en subconjuntos horizontales, cada uno de los cuales constituye una unidad de localización y de replicación.

A un fragmento vertical le corresponden una o mas tablas relacionales (normalizadas). Cada una de ellas tendrá la misma partición horizontal del fragmento.

2.1.1.1.2 Comportamiento de un tipo de objeto.

El comportamiento de un tipo de objeto se modela como un conjunto de servicios que se implantan como procedimientos almacenados en la base de datos. Cada servicio trabaja sobre un fragmento vertical de una tupla (objeto), y puede invocar servicios de otros objetos del mismo o diferente tipo.

En la figura 1 se muestra de manera general cómo cada tipo de objeto tiene una estructura y un comportamiento, donde la estructura se visualiza como una tabla no normalizada y el comportamiento como un conjunto de servicios. La tabla no normalizada se fragmenta verticalmente y cada fragmento vertical se particiona a su vez horizontalmente. A cada fragmento vertical corresponden una o mas tablas relacionales, cada una con la misma partición horizontal del fragmento.

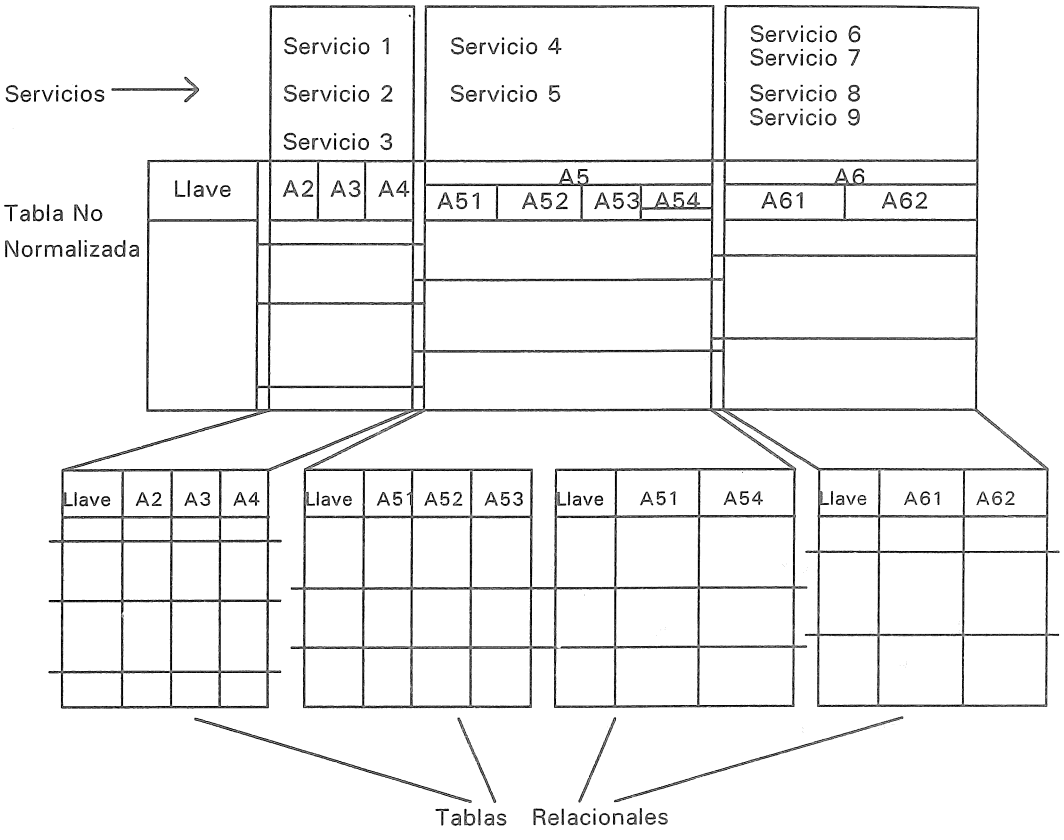

Figura 1. Visualización de un tipo de objeto.

2.1.1.2 Operaciones de una aplicación que pueden ser invocadas por los usuarios.
Típicamente, las operaciones que ofrece una aplicación son de dos tipos:
(i) Transaccionales: combinan interacción con el usuario y proceso en memoria principal, con llamados al api, a servicios de uno o mas objetos y, eventualmente, a operaciones de otras aplicaciones. Los servicios invocados pueden ser de objetos pertenecientes a otras aplicaciones.

(ii) Masivas: cada sitio las aplica a sus objetos locales de un cierto tipo. La operación global se logra por la unión de las operaciones locales.

2.1.2 Composición de un sitio.

El sistema distribuido tendrá sitios servidores de datos allí donde se genere y/o utilice la información. Cada uno de ellos tendrá:

. Las tablas del catálogo de distribución, con cierta información replicada y otra específica al sitio.
. Tablas de referencia de las aplicaciones instaladas localmente.
. Tablas relacionales correspondientes a fragmentos complejos (subconjuntos horizontales-verticales de tablas complejas) asignados al sitio.
. Paquetes de procedimientos correspondientes a servicios ofrecidos por objetos con fragmentos en el sitio.
. Paquetes de procedimientos correspondientes a operaciones ofrecidas por aplicaciones instaladas localmente.
.Paquete de funciones ofrecidas por el api que maneja el catálogo de distribución.

2.2 Pasos de la metodología.

1) Definir aplicaciones: esta definición es uno de los resultados del proceso de planeación estratégica de sistemas de información que cada organización debería realizar con anterioridad al desarrollo de cualquier sistema. Para ello se pueden utilizar metodologías tales como las presentadas en [10] o [11].

Para cada aplicación definida en el paso 1:

2) Realizar la etapa de análisis, cuyos resultados principales son:
. El modelo entidad-relación.
. La descripción de las operaciones que puede realizar cada tipo de usuario, las cuales pueden ser transaccionales o masivas.

3) Definir tipos de objetos de la aplicación:
. En las operaciones transaccionales, identificar argumentos de entrada que correspondan a llaves primarias de entidades que aparecen en el modelo entidad-relación. A cada una de estas entidades corresponderá un tipo de objeto, cuyo núcleo será la entidad en cuestión.

. Para cada tipo de objeto, determinar en el diagrama entidad-relación el subgrafo que describe su composición: además de su entidad núcleo, incorporar al subgrafo aquellas entidades que se puedan alcanzar vía relaciones 1:1 y 1:n desde la entidad núcleo o desde alguna de las entidades ya incorporadas. No se deben incluir entidades identificadas como núcleo de otro tipo de objeto.

Analizar entidades no incluidas en algún tipo de objeto:
- Si no es de referencia, pertenece a un tipo aún no identificado. En dicho caso se debe buscar la entidad núcleo de ese nuevo tipo y definir su composición.
- Si es de referencia, no se incluye en ningún tipo de objeto.

4) Definir tablas relacionales para cada tipo de objeto:
Siguiendo las reglas normales de conversión, pasar a tablas, por separado, cada subgrafo del modelo entidad-relación.
Para aquellas relaciones m:n no resueltas que tienen sus extremos en dos subgrafos, añadir a cada uno de los tipos de objeto correspondientes una tabla que represente la relación.
Analizar las tablas que pertenecen a un tipo de objeto:
- Si existen llaves foráneas que representan relaciones 1:1 en una sola dirección con objetos del mismo tipo, añadir para cada una otra llave foránea que represente la relación inversa.
- Si existen llaves foráneas que representan relaciones n:1 con objetos del mismo tipo, añadir por cada llave una tabla que represente la relación 1:n inversa.
- Tablas que representan relaciones m:n entre objetos del mismo tipo se deben replicar. Si la relación tiene atributos que la describen, éstos solo aparecerán en una de las tablas, aquella que represente la dirección de la relación en que más se utilizan esos atributos.
- Si existen llaves foráneas que representan relaciones 1:1 en una sola dirección hacia otros tipos de objeto, añadir en las tablas núcleo de esos otros tipos de objeto una llave foránea que represente la relación inversa.
- Si existen llaves foráneas que representan relaciones n:1 con otros tipos de objeto, añadir por cada llave, en el otro tipo de objeto, una tabla que represente la relación 1:n inversa.

Analizar tablas que quedaron en más de un tipo de objeto debido a que los subgrafos correspondientes tienen una intersección no vacía:
- Si provienen de una entidad que representa una asociación entre dos o más tipos de objeto, deben permanecer en cada uno de ellos. Si la entidad tiene atributos que la describen, éstos solo aparecerán en una de las tablas, aquella que pertenezca al tipo de objeto con el que más se utilizan esos atributos.
- Si provienen de una entidad que representa algún atributo multivalor de una asociación entre dos o más tipos de objeto, se conserva la tabla que pertenezca al tipo de objeto con el que más se utilizan esos atributos multivalor, y se eliminan las de los otros tipos.

5) Definir una tabla no normalizada para cada tipo de objeto:
Integrar todas las tablas pertenecientes a un tipo de objeto en una sola tabla no normalizada en donde la llave primaria se factoriza y aparece una sola vez. La tabla obtenida representa una estructura jerárquica con, posiblemente, varios niveles de anidamiento. En el primer nivel aparecerán la llave primaria, campos atómicos y, generalmente, campos complejos (subtablas).

6) Analizar los campos complejos del primer nivel para verificar que todas sus instancias sean referenciadas de la misma manera por todas las operaciones. De no ser así, se deben descomponer hasta cumplir con esa condición.

7) Para cada tipo de objeto, fragmentar su tabla no normalizada:
   Fragmentar horizontalmente cada columna de primer nivel que no haga parte de la llave primaria. Para una columna, ésto se traduce en encontrar un conjunto de predicados, cada uno de los cuales define un fragmento homogéneo (cualquier par de filas es referenciado con la misma probabilidad por todas las operaciones). La fragmentación debe ser correcta: disyunta, completa y mínima [Özsu 91].

   A cada fragmento horizontal obtenido en el punto anterior se le asigna:
   - Lista de sitios que lo referencian, con la probabilidad de que ésto suceda.
   - Tipo de replicación: no replicar, replicación sincrónica, replicación asincrónica simétrica, replicación asincrónica por snapshots.

   Agrupar columnas fragmentadas de primer nivel que cumplan las siguientes condiciones:
   - Tienen el mismo número de fragmentos.
   - Sus fragmentaciones están definidas por el mismo conjunto de predicados.
   - Fragmentos correspondientes (con igual predicado) tienen mismo tipo de replicación, lista de sitios y probabilidades de referencia.

   Cada agrupación será un fragmento vertical de la tabla no normalizada.

8) Para cada tipo de objeto, fragmentar sus tablas relacionales:
   Cada tabla debe pertenecer a un fragmento vertical. De no ser así, se debe particionar para que ésto se cumpla.
   La fragmentación horizontal de la tabla relacional será igual a la del fragmento vertical al que pertenece.

9) Localizar fragmentos:
   Cada fragmento horizontal de una tabla no normalizada, al cual corresponden uno o mas fragmentos horizontales de tablas relacionales, se colocará en el sitio que más lo referencie.
   Como regla general, se aconseja no replicar. Sin embargo, para agilizar operaciones masivas, puede ser conveniente - si es posible- tener réplicas asincrónicas de ciertos fragmentos en los sitios donde se realizan esas operaciones.

10) Replicar las tablas de referencia, con mecanismos de actualización asincrónicos, en todos los sitios en los que se instale la aplicación a la cual pertenecen.

11) Para cada tipo de objeto, definir sus servicios (comportamiento):
    En cada operación de tipo transaccional, agrupar en un procedimiento (servicio) pasos que operen como una unidad lógica sobre un fragmento vertical de un objeto. Intercalados con esos pasos pueden haber otros que, o bien corresponden a operaciones de otras aplicaciones, o bien operan sobre fragmentos verticales distintos del mismo o diferente objeto y son a su vez servicios. En esos casos, el procedimiento inicial invocará los procedimientos correspondientes a los pasos intercalados. Si estos últimos son servicios, antes de llamarlos, el procedimiento inicial utilizará el api de distribución para obtener los dblink de los sitios donde están los fragmentos sobre los cuales operan.

12) A cada operación de la aplicación definirle un procedimiento:
Después del paso anterior, una operación queda definida como una combinación de interacciones con el usuario, procesos locales en memoria principal, llamados a operaciones de otras aplicaciones, invocaciones del api y llamados a servicios. Esa combinación se define como un procedimiento.

3. API PARA EL MANEJO DE DISTRIBUCION.

El api maneja un catálogo y ofrece un conjunto de funciones.

3.1 Catálogo.

Para el api de distribución, la base de datos distribuida está conformada por un conjunto de tablas relacionales virtuales. Hay una tabla por cada tipo de objeto. Cada tabla tiene una llave (simple o compuesta) y uno o mas atributos virtuales, tantos como fragmentos verticales tenga el tipo de objeto.

Las tablas virtuales están particionadas con fragmentación mixta (vertical - horizontal). Cada fragmento tiene la llave y uno de los atributos virtuales de la tabla; además, tiene asociado un predicado y se encuentra localizado en un sitio.

El predicado de un fragmento es una conjunción de comparaciones. Cada comparación es una tripleta <atributo, operador de comparación, constante>, donde atributo es uno de los atributos atómicos de primer nivel del tipo de objeto al cual pertenece el fragmento.

Cada tupla de una tabla virtual corresponde a un objeto. Cada atributo virtual del objeto se encuentra en un fragmento.

Si se desea poder buscar un objeto de un cierto tipo dando valores de atributos que no son su llave primaria, se deben crear índices (listas invertidas) para esos atributos. Un índice se localiza en un sitio.

Teniendo en cuenta lo anterior, el api mantiene un catálogo con la siguiente información:
. Por cada tipo de objeto: código, nombre, llave primaria, atributos virtuales, fragmentos e índices de búsqueda por llaves no primarias.
. Por cada fragmento: nombre, predicado y sitio donde está localizado.
. Por cada índice de búsqueda por llave no primaria: llave y sitio donde está ubicado.
. Por cada sitio: su dblink.
. Por cada objeto: llave y nombres de los fragmentos que lo contienen.

La información del catálogo, excepto aquella que es a nivel de objetos, se encuentra replicada asincrónicamente en todos los sitios. La información a nivel de objetos está distribuida de la siguiente manera:
. En el catálogo local de cada sitio donde haya una porción de un objeto, estará la información de los fragmentos que contienen a ese objeto.

Por cada tipo de objeto hay una jerarquía de servidores de catálogo, con información que permite ubicar los objetos. Cada nivel contiene a sus inferiores. Cada servidor puede tener copias asincrónicas. Para cada sitio se establece, por tipo de objeto, el orden de búsqueda en los servidores de catálogo, para el caso en que el objeto buscado no tenga un componente local.

3.2 Funciones.

1. *Buscar atributo virtual de un objeto*:
   Dados el tipo, la llave y el nombre de un atributo virtual de un objeto, devuelve el dblink del sitio donde se encuentra ese atributo virtual de ese objeto.

2. *Buscar todos los atributos virtuales de un objeto:*
   Dados el tipo y la llave de un objeto, devuelve, para cada atributo virtual del objeto, el dblink del sitio donde se encuentra ese atributo virtual de ese objeto.

3. *Buscar sitios de inserción para un nuevo objeto:*
   Dados el tipo del nuevo objeto y los valores que tiene para los atributos de fragmentación, devuelve, para cada atributo virtual, el dblink del sitio inicial donde se debe insertar.

4. *Crear objeto:*
   Dados el tipo, la llave y los valores de los atributos de fragmentación de un nuevo objeto, determina los fragmentos iniciales a los cuales debe ir el objeto, y actualiza el catálogo con esa información de fragmentación.

5. *Borrar objeto:*
   Dados el tipo y la llave de un objeto, elimina del catálogo la información de fragmentación de ese objeto.

6. *Migrar atributo virtual de un objeto:*
   Dados el tipo, la llave y el nombre del atributo virtual de un objeto, así como el nombre del nuevo fragmento para el atributo, actualiza en el catálogo la información de fragmentación del objeto, indicando el nuevo fragmento al cual migra el atributo virtual.

7. *Buscar objeto por llave no primaria:*
   Dados un tipo de objeto, el nombre de una llave no primaria y su valor, devuelve las llaves primarias de los objetos que tienen esa llave no primaria.

3.3. Herramienta de creación y mantenimiento del catálogo.

Para efectos del catálogo existe un sitio denominado maestro, donde está la copia principal de las tablas que se replican, y otra información de control que solo se encuentra allí.

Los pasos a seguir para la creación inicial son:

- Creación del maestro: creación de tablas, compilación del api, creación de dblinks hacia los otros sitios.

- Creación de los restantes sitios: en cada sitio, creación de grupos de "snapshots", tablas locales, dblinks hacia los otros sitios, compilación del api.
- Creación de réplicas de los servidores de catálogo, usando "triggers", paquetes de procedimientos y RPCs asincrónicos.

Para el mantenimiento, existen los siguientes servicios:
- Cambio de dirección (dblink) de un sitio.
- Añadir/suprimir tipos de objeto.
- Cambio de ubicación de un fragmento.
- Añadir/suprimir fragmentos.
- Cambio de ubicación de una copia de un servidor de catálogo.
- Añadir/suprimir copias de un servidor de catálogo.
- Añadir/suprimir índices por llaves no primarias.
- Verificar consistencia del catálogo.

4. CONSULTAS GENERALES DISTRIBUIDAS.

Los sistemas comerciales de manejo de bases de datos no ofrecen, a la fecha, transparencia a la fragmentación de tablas, y no parece esa sea una de sus prioridades, en materia de distribución, para el corto y aún el mediano plazo. Se plantea entonces el problema de cómo se deben expresar y resolver consultas que involucren un gran número de datos dispersos en múltiples sitios de un sistema distribuido. Factor crítico para este tipo de requerimientos es la disponibilidad de los sitios involucrados: si la consulta se resuelve con una serie de llamados sincrónicos a esos sitios, la no disponibilidad de cualquiera de ellos hará fracasar toda la operación.

Considerando lo anterior, en el proyecto Uniandes-Oracle se está trabajando en el tratamiento asincrónico de consultas que se dirigen a muchos sitios y cuya respuesta es la unión de los resultados obtenidos al ejecutarla en cada uno de ellos.

Los pasos que se siguen para una consulta son:
. Utilizando una interfaz basada en formas, el usuario entra su consulta, define el esquema de la tabla respuesta, y selecciona los sitios donde desea enviarla.
. Internamente, se crea dinámicamente la tabla respuesta y se hacen RPC's asincrónicos para cada sitio involucrado, a un procedimiento que recibe como parámetros el texto de la consulta y el sitio donde se originó.
. En cada sitio el procedimiento ejecuta dinámicamente la consulta e inserta el resultado en un snapshot actualizable[14] local en el que cada campo de cada fila de la tabla respuesta será una fila del snapshot.
. Los snapshots se crean y refrescan a partir de una tabla maestra que tiene el mismo esquema. Cada snapshot selecciona de la tabla maestra las filas que tienen como origen de la consulta al sitio donde ellos se encuentran.
. Usando el mecanismo de refresco completo de los snapshots actualizables, que pasan por la tabla maestra, el sitio donde se originó la consulta va recibiendo las respuestas.

Una opción de la interfaz permite al usuario averiguar el estado de su consulta. Cada vez que lo haga, se pasará, del snapshot local a la tabla respuesta,los datos que hayan llegado. La interfaz le informa al usuario qué sitios han respondido y cuáles están pendientes.

5. CONCLUSIONES.

La distribución de datos y procesos comienza a imponerse en organizaciones descentralizadas. Para ello se requiere un sólido soporte tanto metodológico como de hardware, software y comunicaciones.

En este artículo se presentó una metodología de diseño de distribución que incorpora el concepto de objetos de distribución en el desarrollo de aplicaciones distribuidas sobre bases de datos relacionales. Dicha metodología fué utilizada con éxito en el proyecto de desarrollo de un sistema de información distribuido para un importante Banco comercial colombiano. Desafortunadamente, por el momento, la metodología no tiene un soporte computacional estilo CASE, lo que hace un poco engorrosa la documentación y el mantenimiento del diseño.

Para el manejo de la distribución se desarrollaron tres productos de software: un api que permite ubicar objetos dada su llave primaria, una herramienta de creación y mantenimiento de un catálogo de distribución, y una aplicación basada en formas, sql dinámico, RPC's asincrónicos y snapshots actualizables, que permite hacer consultas distribuidas sobre múltiples sitios. Los tres productos se están utilizando en el sistema distribuido bancario mencionado en el párrafo anterior. Dicho sistema, en etapa de pruebas, está conformado por 140 sitios geográficamente dispersos; en cada sitio corre una base de datos Oracle7, y los sitios se comunican entre si con una red privada X.25.

REFERENCIAS.

[1] Abasolo, J., Blanco, R., Mendieta, F. Metodología de planeación de bases de datos. Documento Cátedra Oracle. Universidad de los Andes, Bogotá, Colombia, junio de 1993.

[2] ABASOLO, J., BLANCO, R., MENDIETA, F. *Taxonomía del RDBMS de Oracle - versiones 6 y 7 - como Sistema de Bases de Datos Distribuido.* Documento Cátedra Oracle. Universidad de los Andes, Bogotá, Colombia, junio de 1993.

[3] ABASOLO, J., BLANCO, R., MENDIETA, F. *Manejo de múltiples copias de un dato.* Documento Cátedra Oracle. Universidad de los Andes, Bogotá, Colombia, julio de 1993.

[4] ABASOLO, J., BLANCO, R., MENDIETA, F. *Metodología de diseño de datos distribuidas y su implantación sobre Oracle7.* CLEI Panel 94, Ciudad de México, septiembre de 1994.

[5] BARKER, R. *Case*Method - Entity Relationship Modelling.* Addison - Wesley, 1990.

[6] BLANCO, R., MENDIETA, F., *Diseño de bases de datos distribuidas: metodología e implantación.* Documento de Tesis I. Magister en Ingeniería de Sistemas. Universidad de los Andes, Bogotá, Colombia, enero de 1994.

[7] CAMPO, A. *Distributed Objects with Oracle.* Semana internacional de usuarios Oracle, Filadelfia, USA, septiembre de 1995.

[8] CERI, S., PELAGATTI, G., *Distributed Databases - Principles and Systems.* McGraw-Hill, New York, 1984.

[9] HOLOMAN, S. *Rules for Allocation in a Distributed Processing System.* Reporte HC-BDO-R14, Banco de Occidente, Cali, Colombia, Enero de 1993.

[10] IBM. *Business System Planning: Information Systems Planning Guide, Application Manual* GE 20-0527-1. White Plains, N.Y.: IBM Corporation, August 1975.

[11] MARTIN, J. *Information Engineering.* Book I, *Introduction.* Book II, *Planning and Analysis.* Book III, *Design and Construction.* Prentice Hall, 1990.

[12] MEGHINI, C., THANOS, C. *The complexity of Operations on a Fragmented Relation.* ACM Transactions on Database Systems 16, 1 (1991).

[13] MONTENEGRO, C., QUIROGA, C. *Herramienta administradora del catálogo de una base de datos distribuida.* Documento de tesis. Ingeniería de Sistemas y Computación. Universidad de los Andes, Bogotá, Colombia, agosto de 1995.

[14] ORACLE CORP. *ORACLE 7 Server Concepts Manual,* Dec 1992.

[15] ORTIZ, M. I. *Metodología de diseño de bases de datos distribuidas.* Documento de Tesis II. Magister en Ingeniería de Sistemas. Universidad de los Andes, Bogotá, Colombia, agosto de 1995.

[16] ÖZSU, M., VALDURIEZ, P. *Principles of Distributed Database Systems.* Prentice-Hall, 1991.

[17] ROZO, C. *Mantenimiento de la consistencia en el catálogo de una base de datos distribuida.* Documento de tesis. Ingeniería de Sistemas y Computación. Universidad de los Andes, Bogotá, Colombia, agosto de 1995.

[18] ULLMAN, J. *Principles of Database and Knowledge-Base Systems.* v. I. Computer Science Press, 1988.