Acceso a datos en aplicaciones multihilos orientadas a objetos.

Data access to object oriented multithreads applications.

Yadira Lizama Mué, Lissuan Fadraga Artiles

Universidad de las Ciencias Informáticas

ylizama@uci.cu, lfadraga@uci.cu

Resumen

Las aplicaciones de software, en su mayoría tratan de solucionar al máximo cuatro características fundamentales con el objetivo de satisfacer las necesidades de sus clientes: la persistencia de los datos, el acceso eficiente, gran capacidad de respuesta y alta disponibilidad del servicio de acceso. Las soluciones varían en dependencia de los requerimientos de las aplicaciones que se enfrentan, es por eso que estas actividades requieren gran esfuerzo de los desarrolladores, tienen gran peso en las decisiones de la arquitectura y abarcan gran parte del ciclo de vida del proyecto. En este artículo se exponen los principales elementos a considerar a la hora de enfrentar la persistencia y el acceso a los datos en las aplicaciones que requieren sincronización en la manipulación de sus datos, haciendo especial énfasis en las aplicaciones multihilos desarrolladas en el lenguaje C++.

Palabras clave: Acceso a datos, Concurrencia, Persistencia, Programación Multihilo

Abstract

The applications of data management software try to resolves four key features to satisfy the needs of their costumers: persistence of data, data access, highly responsive and highly available access service. There is a variety of solutions, which depend of application's requirements, that's why these activities require great efforts from developers; have great weight in architecture decisions and cover part of the projects life cycle. This article expose the principal elements to consider the persistence and data access on applications that require synchronization to manipulate their data, specifically multi-threaded applications implemented with C++.

Keywords: *Data Access, Concurrency, Multi-thread Programming, Persistence.*

Introducción

En los programas concurrentes existen ciertas unidades de ejecución internamente secuenciales (procesos o hilos de ejecución) que se ejecutan paralela o simultáneamente. La programación multihilos es muy utilizada para la implementación de sistemas concurrentes. Asocia nuevos conceptos, nuevas formas de análisis y sobre todo diferentes métodos de desarrollo. Las formas básicas de interacción entre los hilos de ejecución son la sincronización, referida al paso de mensajes entre ellos; la comunicación o uso de memoria compartida y la señalización, implementada generalmente con semáforos para controlar el acceso a recursos compartidos. Existe todo un universo relacionado con los sistemas multihilos, pueden aplicarse a sistemas de tiempo real, sistemas paralelos, donde se requiere excelente rendimiento. Estos sistemas están relacionados, en su mayoría con el procesamiento de gran cantidad de datos. Podemos pensar que este aspecto es trivial, pero aplicar soluciones eficaces en entornos monohilos para este tipo de sistemas puede traer graves problemas. Existen dos puntos de gran importancia para las aplicaciones multihilos orientadas a objetos que manipulan gran cantidad de información: la persistencia y el acceso a los datos.

Desarrollo

1. Persistencia de los datos.

La persistencia de la información es una de las partes críticas de las aplicaciones de software. "La persistencia de los datos es la habilidad que tiene un objeto de sobrevivir al ciclo de vida del proceso en el cual reside. Los objetos que mueren al final de un proceso se llaman transitorios." [1]

La persistencia es completamente ortogonal al tipo de datos. En otras palabras, la persistencia es una propiedad de los objetos, no de sus clases. [6] La ortogonalidad se refiere al derecho que tienen todos los objetos de persistir independientemente de su tipo de datos, aunque al persistir un objeto, también lo hace su tipo, no se debe almacenar un valor de un tipo determinado y leerlo como otro.

Grady Booch define a la persistencia de la siguiente manera: "La persistencia es la propiedad de un objeto por la que su existencia trasciende el tiempo (es decir, el objeto continúa existiendo después de que su creador deja de existir) y/o el espacio (es decir, la posición del objeto varía con respecto al espacio de direcciones en el que fue creado) "[7]

Aunque algunos relacionen la persistencia de los objetos directamente al uso de bases de datos, existen muchas formas que resultan útiles en dependencia de la aplicación que se crea. En las aplicaciones orientadas a objetos, tecnología bastante popular en el diseño de software actual, la persistencia se logra principalmente a través de la serialización de los objetos, o el almacenamiento en bases de datos. [8]

Cuando son necesarias ciertas características como la independencia de los datos y los programas que los manipulan, la eliminación de la redundancia en el almacenamiento, la sincronización y la integridad de los datos almacenados, mecanismos de seguridad y recuperación y facilidad de manejo de la información transparente a las complejidades de su almacenamiento, la utilización de las bases de datos para garantizar la persistencia es la solución mas óptima.

Una base de datos es una "colección o depósito de datos integrados, almacenados en soporte secundario (no volátil) y con redundancia controlada. Los datos, que han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse independientes de ellos, y su definición (estructura de la base de datos) única y almacenada junto con los datos, se ha de apoyar en un modelo de datos, el cual ha de permitir captar las interrelaciones y restricciones existentes en el mundo real. Los procedimientos de actualización y recuperación, comunes y bien determinados, facilitarán la seguridad del conjunto de los datos."[3]

Es importante diferenciar el término de bases de datos y el término Sistemas de Gestión de Base de Datos (SGBD). "Un SGBD es la herramienta que permite interactuar los datos con los usuarios de los datos, de forma que se garanticen todas las propiedades definidas en una base de datos."[5] Dichas propiedades se refieren, entre otras, a la integridad, confidencialidad y seguridad. La tecnología de bases de datos se ha consolidado en la actualidad como una solución eficiente para garantizar la persistencia de la información.

2. Acceso a bases de datos

El acceso a los objetos persistidos es un elemento muy relacionado a la persistencia. No solo se trata de la mejor solución para guardar la información, lograr métodos de acceso tan óptimos como sea posible, es una de las prioridades de los desarrolladores.

El acceso a datos es la "parte" del software que media entre el resto de las "partes" y la base de datos. Un ejemplo muy ilustrativo son las aplicaciones con arquitectura en capas donde una capa es un "conjunto de subsistemas que comparten el mismo grado de generalidad y de volatilidad en las interfaces: las capas inferiores son de aplicación general y deben poseer interfaces más estables, mientras que las capas más altas son más dependientes de la aplicación y pueden tener interfaces menos estables."[9]

La capa de acceso a datos es la que ofrece un conjunto de interfaces que permiten la abstracción al programador de las especificidades del proveedor de datos, siendo una de las capas inferiores mencionadas anteriormente. Si ocurren cambios en la estructura de almacenamiento, o sea la base de datos, las modificaciones afectarán solo esta capa de la aplicación y el resto de las capas del programa no sufrirán modificaciones.

Los métodos de implementación de la capa de acceso a datos son tan variables como las aplicaciones que se desarrollan y dependen específicamente de los problemas que surgen a partir de las características de las mismas. En el caso de las aplicaciones multihilos con implementación orientada a objetos se identifican dos problemas fundamentales en este medio:

- La concurrencia en el acceso a los datos almacenados.
- La interacción entre el modelo relacional de la base de datos y el modelo orientado a objetos de los programas.

2.1 Concurrencia en el acceso a datos

La concurrencia es el concurso simultáneo de varias circunstancias [4], o sea, la simultaneidad de hechos. Para aplicaciones multihilos que interactúan con bases de datos multiusuario, la concurrencia se analiza en dos escenarios: el acceso concurrente a través de múltiples conexiones de usuarios al mismo conjunto de datos y la manipulación concurrente de una misma conexión por múltiples hilos de ejecución de la aplicación.

2.1.1 Acceso concurrente de múltiples usuarios

Los SGBD multiusuario garantizan la concurrencia en la ejecución de varias transacciones casi instantáneas sobre el mismo conjunto de datos. Estos SGBD se caracterizan por las propiedades ACID de las transacciones que manipulan:

Atomicidad (Atomicity): Todas las operaciones de una transacción son ejecutadas por completo, o no se ejecuta ninguna de ellas.

Consistencia (Consistency): Una transacción T transforma un estado consistente de la base de datos en otro estado consistente, aunque T no tiene por qué preservar la consistencia en todos los puntos intermedios de su ejecución.

Aislamiento (Isolation): Una transacción está aislada del resto de transacciones. Aunque existan muchas transacciones ejecutándose a la vez, cualquier modificación de datos que realice T está oculta para el resto de transacciones hasta que T sea confirmada.

Durabilidad (Durability): Una vez que se confirma una transacción, sus actualizaciones sobreviven cualquier fallo del sistema. Las modificaciones ya no se pierden, aunque el sistema falle justo después de realizar dicha confirmación.

El Subsistema de Recuperación del SGBD es el encargado de conseguir el cumplimiento de las propiedades de atomicidad y durabilidad de las transacciones. La conservación de la consistencia es una propiedad cuyo cumplimiento han de asegurar, por un lado los programadores de base de datos, y por otro el Subsistema de Integridad del SGBD. El Subsistema de Control de Concurrencia es el encargado de conseguir el aislamiento de las transacciones. Para este caso los programadores tienen que garantizar únicamente que sus transacciones no dejen estados inconsistentes en la base de datos evitando la redundancia de la información y logrando un buen diseño del modelo de datos. El resto es problema del SGBD.

2.1.2 Manipulación concurrente de una conexión por múltiples hilos de ejecución.

Los mecanismos de sincronización son una parte fundamental en el diseño de las aplicaciones multihilos, sobre todo cuando un recurso determinado puede ser utilizado de manera concurrente. Para el acceso a datos en este escenario, se introduce el problema de que una conexión pueda ser manipulada por varios hilos de ejecución y no se apliquen mecanismos de sincronización que impidan la colisión de las operaciones. Existen librerías para acceso a bases de datos desde lenguajes de programación que no son "thread-safety"[1] , otras que lo son pero imponen restricciones en su utilización.

Para garantizar la sincronización en el acceso a la base de datos de aplicaciones multihilos se proponen varias soluciones:

1 Término muy utilizado en la programación multihilo. Una pieza de código es "thread-safety" si funciona correctamente durante la ejecución simultánea de múltiples hilos de ejecución.

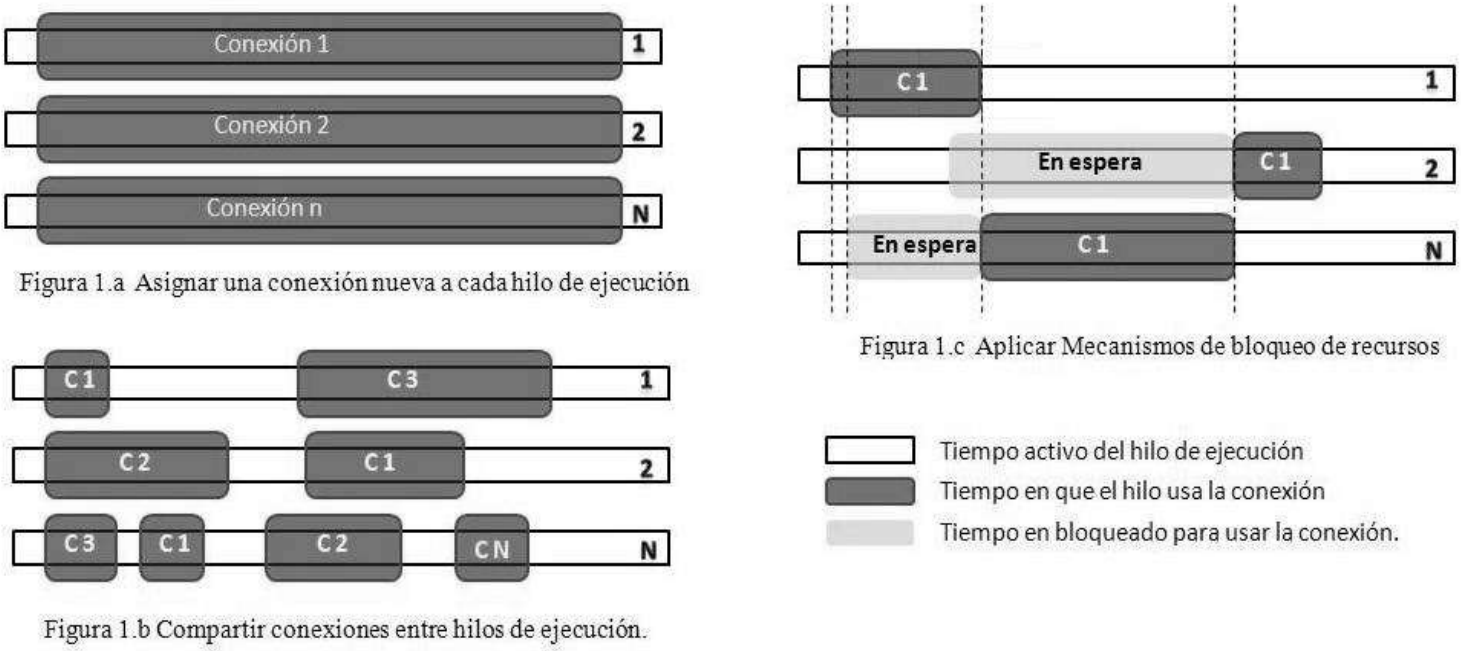

Figura 1.a Asignar una conexión nueva a cada hilo de ejecución

Figura 1.b Compartir conexiones entre hilos de ejecución.

Figura 1.c Aplicar Mecanismos de bloqueo de recursos

Figura 1: Sincronización entre hilos de ejecución para acceso a bases de datos.

a) Asignar una conexión nueva para cada hilo de ejecución. *(Figura 1.a)*

Todas las transacciones ejecutadas por un hilo de ejecución utilizarán una conexión propia. Este método es fácil de elaborar, cada hilo utiliza una conexión diferente por lo que no se producen colisiones en la ejecución de las transacciones. El patrón de diseño Singleton [10] puede ser utilizado para garantizar que la conexión sea única en cada implementación de los hilos. En este caso es medible la cantidad de conexiones a priori que se realizarán en una aplicación si se conoce la cantidad de hilos de ejecución que se levantan y acceden a la base de datos. Pero la existencia de conexiones abiertas que no se estén utilizando en intervalos de tiempo largos no es eficiente por lo que se recomienda solo si la cantidad de hilos que necesitan de una conexión nueva es mínima y se ejecutan operaciones sobre la base de datos constantemente.

b) Compartir conexiones entre hilos de ejecución. *(Figura 1.b)*

Cuando las operaciones de acceso que se realizan en un mismo hilo de ejecución distan unas de otras, puede optimizarse el uso de las conexiones. Para ejecutar una operación, se hace la conexión, se realizan las consultas necesarias y luego se cierra y puede ser aprovechada quizás por otro hilo que la necesite, así solo se utiliza en el momento que es requerida. Cuando se necesite de nuevas operaciones entonces se realiza el mismo procedimiento, siempre tratando de ejecutar el mayor número de operaciones posibles para reducir el costo de la conexión y la desconexión por cada consulta realizada.

Podemos realizar pruebas de estrés al SGBD para conocer el límite de conexiones que puede manejar manteniendo la capacidad de respuesta requerida. Así se crean y se administran dichas conexiones y podremos lograr mejor rendimiento. Es responsabilidad del programador cuando se alcanza un límite de conexiones determinado, acumular las peticiones nuevas que llegan hasta que puedan ser atendidas y no se pierdan los datos. Las librerías para conexión en su mayoría tienen funciones que permiten asignar múltiples conexiones de manera no-bloqueante.[2] Si el SGBD está distribuido y puede soportar grandes cargas de operaciones, no se recomienda limitar el número de conexiones sino explotar las capacidades del SGBD.

c) Aplicar mecanismos de bloqueo de recursos. *(Figura 1.c)*

Puede garantizarse la sincronización de los hilos de ejecución que acceden a la base de datos mediante el uso de semáforos. Se crea una conexión única con el patrón Singleton, cuando va a ser utilizada por un hilo de ejecución se bloquea un semáforo, se realizan las operaciones y se desbloquea el semáforo. Si llegaron nuevas peticiones de uso de la conexión todos los semáforos

2 Ejemplo son las funciones "PQconnectStart" y "PQconnectPoll" de la librería libpq para conexión a PostgreSql desde C++.

bloqueados se ponen en cola de espera y a medida que se van liberando, el próximo en la cola continúa su procesamiento. Este método sólo es aplicable para librerías que impongan restricciones solo sobre la conexión y no es eficiente para sistemas paralelos porque las operaciones sobre la base de datos se realizan secuencialmente, y no se aprovecha al máximo el rendimiento del SGBD.

La concurrencia en el acceso a datos es un factor que influye decisivamente en el diseño e implementación de este tipo de sistemas.

2.2 Interacción entre el modelo relacional de la base de datos y el modelo orientado a objetos de los programas.

2.2.1 Modelo Orientado a Objetos

Los sistemas basados en modelos orientados a objeto fueron inspirados a partir del paradigma de programación orientada a objetos. El primer lenguaje de programación orientado a objetos fue Simula 67 oficialmente presentado por Ole Johan Dahl y Kristen Nygaard en la Conferencia de Lenguajes de Simulación en Lysebu en Mayo de 1967. [11, 12] Sin embargo, el primer lenguaje que popularizó la aproximación a objetos fue Smalltalk (1976). Posteriormente aparecieron lenguajes orientados a objetos más avanzados como C++ en 1980 diseñado por Bjarne Stroustrup [17] y Java desarrollado por la Sun Microsystems a principio de los años 90[13]. Algunos conceptos asociados al Modelo Orientado a Objetos son:

Clase: Define el comportamiento y la estructura de un tipo de objetos determinado. Es una plantilla para crear muchos objetos independientes con las mismas características.

Objeto: Unidad creada a partir de una clase que combina sus propiedades y funciones. Cada objeto recuerda sus propios valores y representa una instancia de una clase.

Atributo: Propiedad de los objetos que pueden adquirir valores de un dominio.

Funciones: Acciones, métodos que puede realizar un objeto.

Identidad del objeto: Permite identificar a un objeto del resto aún cuando pertenecen a la misma clase y tengan igual estado (valores de sus atributos).

Encapsulamiento: Ocultamiento de los atributos y las acciones de los objetos. Oculta los detalles de su implementación.

Abstracción: Denota las características esenciales que distinguen a un objeto de otros tipos de objetos, definiendo precisas fronteras conceptuales, relativas al observador.

2.2.2 Modelo relacional.

La popularidad del modelo relacional se debe primariamente a su simplicidad. Aparece en 1970 presentado por Edgar F. Codd en el artículo "A relational model for large shared data banks" publicado en "Communications of the ACM"[19]. Representó un verdadero hito en el surgimiento de los SGBD. Sobre el modelo relacional se han definido los estándares ANSI e ISO del lenguaje de definición y manipulación de bases de datos relacionales SQL (Structured Query Language, por sus siglas en inglés). [14] A partir de mitad de los años 80 hasta la actualidad el modelo relacional es utilizado por la mayoría de los SGBD y se pueden destacar algunos muy buenos como: PostgreSQL, MySQL (libres) y Oracle, Microsoft SQL Server (no libres). Algunos conceptos asociados al modelo relacional son [14]:

Relación: Representa tanto instancias de una entidad del universo real como interrelaciones entre entidades de distinto tipo. Su representación informal es una tabla. La relación es el elemento fundamental del modelo relacional.

Dominio: Conjunto válido de valores de referencia para definir propiedades o atributos. Un dominio es un conjunto nominado y homogéneo de valores que pueden adquirir los atributos de las relaciones.

Atributo: Representa una propiedad de una relación y tiene dependencia existencial de la misma. Toma valores de un dominio. Su representación informal es una columna.

Tupla: Ocurrencia o instancia dentro de una relación. Permite referenciar una instancia de una entidad en el universo o la

interrelación específica o concreta entre instancias de entidades. Su representación informal es una fila. Una relación tiene un conjunto de tuplas.

Clave primaria: Identificador único para cada tupla que garantiza que nunca existen dos tuplas con los mismos valores de sus atributos.

2.3 Un problema, varias soluciones.

Las diferencias entre ambos modelos, conocida como "desajuste por impedancia" [18], es notable y a medida que el modelo de objetos aumenta, las incongruencias son cada vez mayores. En este caso se referencian un conjunto de soluciones que adoptan los arquitectos en el diseño de aplicaciones orientadas a objetos.

2.3.1 Entorno puramente relacional: No existe un modelo de clases persistentes orientado a objetos. La lógica de la aplicación interactúa directamente con la base de datos o reside generalmente en los procedimientos almacenados en el SGBD. Los programadores están familiarizados con el lenguaje SQL por lo que podría pensarse que no hay problemas, sin embargo las complejidades en el código aumentan considerablemente, las aplicaciones son poco portables y difíciles de mantener.

2.3.2 Entorno puramente Orientado a Objetos: Sustituir el SGBD relacional u objeto relacional por un SGBD orientado a objetos (SGBDOO) nos permitiría un mayor acercamiento del modelo de datos al diseño orientado a objetos de nuestras aplicaciones. Las bases de datos orientadas a objetos (BDOO) son aquellas cuyo modelo de datos está orientado a objetos y almacenan y recuperan objetos de los que se almacena su estado y comportamiento. [15]Las clases utilizadas en la aplicación son las mismas clases que serán utilizadas en una BDOO; de tal manera, que no es necesaria una transformación del modelo de objetos para ser utilizado por un SGBDOO.

Aunque los SGBDOO proporcionan soluciones apropiadas para un gran número de aplicaciones no tienen el nivel de desarrollo de los SGBD relacionales y los SGBD objeto-relacionales que se han consolidado como soluciones factibles en el desarrollo de software en la actualidad.

El modelo relacional tiene una sólida base teórica y los productos relacionales disponen de muchas herramientas de soporte que sirven tanto para desarrolladores como para usuarios finales. El modelo estándar ODMG, con un lenguaje de definición de objetos ODL y un lenguaje de consulta a objetos OQL es bastante eficiente pero aún no tiene fundamento matemático teórico que lo sustente.

Otro aspecto es la optimización de consultas que requiere una compresión de la implementación de los objetos, para mejorar los tiempos de acceso a la base de datos, sin embargo, se compromete el concepto de encapsulamiento.

Respecto a las herramientas existentes, se destaca BD4Objects como una de las soluciones más experimentadas, BDOO libre bajo la licencia GPL, desarrollada para Java y tecnología .NET. [16]

2.3.3 Mapeo Objeto-Relacional: (ORM, acrónimo de Object-Relational Mapping). Los ORM permiten la coexistencia del modelo relacional de la base de datos y el diseño orientado a objetos de las aplicaciones en nuestros entornos de software y la comunicación entre ellos, garantizando cierta abstracción del programador sobre el modelo relacional pudiendo manipular la persistencia de sus objetos con una mínima interacción con el mismo. En su mayoría están compuestos por librerías, clases y formas descriptivas que permiten el mapeo de las clases. Soportan al menos el mapeo a un tipo de bases de datos, son utilizados únicamente por las aplicaciones implementadas con el mismo lenguaje de programación y, por supuesto, mapean bases de datos relacionales u objeto-relacionales únicamente.

Un buen ORM debe permitir [8]:

(a) Mapear las clases del modelo orientado a objetos a las tablas del modelo relacional y las propiedades a columnas.

(b) Persistir objetos a través de un método Orm.Insertar (objeto) encargándose de generar el código SQL correspondiente.

(c) Recuperar objetos persistidos a través de un método objeto = Orm.Cargar (objeto.clase, clave_primaria).

(d) Recuperar una lista de objetos a partir de un lenguaje de consulta especial a través de un método: ListaObjetos = Orm.Buscar (Objeto FROM MiObjeto WHERE Objeto.Propiedad=5), o algo más complejo ListaObjetos = Orm.Buscar (Objeto ORM FROM MiObjeto WHERE Objeto.Relacion1.Relacion2.Propiedad2=5), y el ORM transformará a través de varias consultas entre tablas.

Los ORM pueden clasificarse en dependencia del tipo de mapeo que realizan:

(a) Mapeo de Objetos Ligero: Las entidades se mapean manualmente a las tablas en la base de datos. Las llamadas al RDBMS se separan de la lógica de negocio a través de algún patrón de diseño como el DAO, acrónimo de Data Access Object. Las consultas complejas se elaboran en lenguaje SQL y se ejecutan directamente. Esta estrategia es muy común y tiene éxito en aplicaciones con relaciones sencillas.

(b) Mapeo de Objetos Medio: Es diseñado completamente a través de los objetos que persisten. Cubre las funcionalidades del anterior y además la persistencia de colecciones de objetos. Las sentencias SQL se generan en tiempo de ejecución. Adecuado para aplicaciones de complejidad media y que necesiten de cierta portabilidad a varios SGBD.

(c) Mapeo de Objetos Completo: Este método es una evolución de los anteriores, brinda soporte para relaciones complejas como la herencia y proporciona un lenguaje completo de consultas orientado a objetos. Cubre las relaciones de carga perezosa (lazy loading) para cargar los objetos persistidos, sin el grafo de sus relaciones, solo hasta que sean solicitados y carga activa (eager loading) que extrae un objeto con el grafo de sus relaciones completo. Este nivel de funcionalidad es ciertamente difícil de conseguir y puede demorar bastante tiempo de implementación y prueba.

Una solución ORM característica de ese método es Hibernate [2], catalogado como una de las mejores soluciones de este tipo para Java.

Aunque los ORM completos sean una solución bastante atractiva, porque han aportado muy buenos resultados para muchas aplicaciones tienen el problema de que los lenguajes especiales de consultas entre objetos introducen lentitud en la ejecución de las operaciones en comparación con SQL, tienden a introducir demasiada arquitectura en casos donde la solución es mucho más simple.[20, 21]

Es importante abordar los problemas de la concurrencia y la interacción entre los modelos relacional y orientado a objetos, debido a que son los más influyentes en la programación multihilo orientada a objetos, aunque existen otros que no son tan específicos de esta rama pero que influyen considerablemente como el volumen de recuperación de datos, la seguridad y la portabilidad. Elegir una solución específica para implementar nuestra capa de acceso a datos depende de los requerimientos del software que diseñamos y de la disponibilidad y características de herramientas que podemos utilizar. Muchos desarrolladores prefieren desarrollar sus propios métodos adaptados a las necesidades del software al que se enfrentan, alcanzando mayores índices de eficiencia.

3. Soluciones para acceso a datos desde C++.

Existen varias soluciones para acceso a datos desde C++. Algunas de las que se destacan por el grado de avance en la solución que presentan son:

Debea: Debea es una colección de interfaces que permite mapear objetos a relaciones a aplicaciones desarrolladas en C++. Para crear los objetos es necesario crear el código SQL correspondiente. No elimina completamente el código SQL para la realización de consultas complejas. Tiene soporte para bases de datos: SQLite3, PostgreSQL, todas las bases de datos que pueden ser accedidas usando ODBC o iODBC. Tiene una arquitectura flexible para agregar soporte a nuevos SGBD. No es "thread-safety" y es multiplataforma. [22].

SOCI: SOCI (Simple Oracle Call Interface) es una librería para acceso a bases de datos con C++ que simula la inserción de consultas SQL en la sintaxis del código C++ sin violar las restricciones del Estándar C++. Actualmente en su versión 3.0 ofrece soporte para los servidores de bases de datos PostgreSQL, Oracle y MySQL. Es fácil de utilizar, pero no es robusta en cuanto al

soporte de tipos de datos no propios de C++, por lo que requiere de mucha implementación adicional para lograr un buen diseño del acceso. SOCI no es "thread-safety" y sus instancias no pueden ser accedidas concurrentemente. La solución que ofrece para estos casos son funciones para administrar múltiples conexiones a la vez. [23]

DTL: DTL (Database Template Library) es una librería para el acceso a bases de datos que proporciona los datos en estructuras que pueden ser tratadas igual a las estructuras STL, acrónimo de Standard Template Library de C++. Los cambios que se realicen en dichas estructuras, serán actualizados inmediatamente en la base de datos. Se compila con la librería STL, por lo que se pueden reutilizar los algoritmos para almacenamiento, búsqueda y manipulación de la información. Ofrece soporte para bases de datos con Oracle, Microsoft SQL Server 2000, Access y MySQL. Sus desarrolladores no ofrecen seguridad sobre el soporte para PostgreSQL así como no garantizan que la librería sea "thread-safety". [24] No solo impone restricciones sobre la conexión sino que las estructuras recuperadas no pueden ser manipuladas por varios hilos de ejecución.

LiteSQL: Es un ORM para implementar la persistencia de datos en C++. Útil para servidores de bases de datos LiteSQL, MySQL y PostgreSQL. Garantiza la persistencia de los objetos y sus relaciones. Minimiza, aunque no suprime la necesidad de ejecutar consultas SQL por los programadores. Necesita de la creación de un fichero XML para la definición de objetos y no es "thread-safety". [25]

Cada una de estas soluciones, aunque presentan ciertas semejanzas, han alcanzado un impacto considerable debido a las características particulares que las distinguen. En el caso de Debea y LiteSQL son las soluciones más completas debido a las opciones del mapeo de objetos que engloban, por lo que puede utilizarse en aplicaciones que necesiten un mapeo completo de la base de datos. SOCI, por otra parte, es muy cómoda para aplicaciones sencillas, pero no es recomendable en sistemas que tengan una arquitectura de acceso a datos muy grande. La principal ventaja de DTL es la interacción que facilita con las estructuras STL de C++, lo que puede ser aprovechado en programas que hacen un uso extenso de las mismas. De manera general, los autores no abarcan en ninguna de las soluciones presentadas todos los problemas de la concurrencia en el acceso. Es por eso que plantean que no son “thread-safety”. Sin embargo, de una forma u otra, todas pueden ser utilizadas en aplicaciones multihilos, bajo estricta responsabilidad del programador de resolver los problemas de sincronización. Las cuatro soluciones imponen restricciones sobre la utilización de la conexión por múltiples hilos de ejecución y proponen la creación de una conexión nueva para cada hilo, lo que no es muy recomendable para aquellas aplicaciones que puedan tener muchos hilos de ejecución accediendo a la base de datos en un instante de tiempo determinado. DTL, además, no garantiza que las estructuras STL que devuelve puedan ser accedidas concurrentemente y SOCI ofrece un mecanismo de manipulación de múltiples conexiones que resuelve parcialmente el problema del incremento de las mismas. [26]

Conclusiones

Las aplicaciones multihilos requieren nuevas formas de pensar y de desarrollar. Garantizar la persistencia y el acceso a datos en este tipo de software, por lo tanto, es un proceso de mucha selección y análisis. Muchos arquitectos tratan de forzar las soluciones que en otro tipo de aplicaciones dan resultados pero que en estos entornos reducen dos de los principales factores que se persiguen: la eficiencia y el rendimiento. En este artículo no se pretende destacar una solución específica, sino que se ofrece una gama de soluciones para diversos aspectos característicos de los programas concurrentes en C++ que pueden ser aplicadas, siempre teniendo en cuenta los requerimientos de las aplicaciones que desarrollamos.

Referencias Bibliográficas

[1] A. Alberca, J. Galvez. Modelos Avanzados de Bases de Datos. Universidad de Castilla, La Mancha, 2008. 102pp.

[2] A. Miguel, M. Piattini. Fundamentos y Modelos de Bases de Datos. RA-MA. 1999. 456pp

[3] Booch, G. "Análisis y Diseño Orientado a Objetos con Aplicaciones". 2a Ed. Addison- Wesley/Díaz de Santos. 1996, Vol. 20

[4] B. Stroustrup. Bjarne Stroustrup's homepage. Disponible en: http://www.research.att.com/~bs/.
[Consultada: 15 Abril 2009]

[5] Colectivo de Autores. Diccionario de la Real Academia Española Online. Disponible en: http://www.rae.es/rae.html . [Consultado: 18 Abril 2009]

[6] C. Evrendilek, et.al. A Preprocessor Approach to Persistent C++, Software Research and Development Center Scientic and Technical Research Council of Türkiye. 2005.

[7] D. Dagum. Objetos, Tablas Relacionales y Desajuste por Impedancia (Impedance Mismatch) 2007. Disponible en: http://diegumzone.spaces.live.com/blog/cns!1AD5096D63670065!307.entry [Consultado: 15 Abril 2009]

[8] D. Dagum. Mapeo de Objetos y Tablas Relacionales (O/R-M). Lo Que a Mí me Sirvió. 2008. Disponible en: http://diegumzone.spaces.live.com/Blog/cns!1AD5096D63670065!705.entry. [Consultado: 8 Abril 2009]

[9] E.F. Codd. "A relational model for large shared data banks". Comunications of the ACM. 1970. Vol. 13, Páginas: 377 - 387 ISSN:0001-0782.

[10] I. Jacobson, et al. Proceso Unificado de Desarrollo de Software. Madrid, Addison-Wesley. 2000. 458 pp.

[11] J. Byous. Java technology. The early years. Sun Developer Network, 2003. Disponible en: http://java.sun.com/features/1998/05/birthday.html. [Consultado: 11 Abril 2009]

[12] J. Soulie. History of C++. 2008 Disponible en: http://www.cplusplus.com/info/history . [Consultado: 11 Abril 2009]

[13] J. Sklenar. Introduction to OOP in Simula. 2007. Disponible en: http://staff.um.edu.mt/jskl1/talk.html. [Consultado: 11 Abril 2009]

[14] K. Wolfgang. (2004)Persistence Options for Object-Oriented Programs. Disponible en: http://www.objectarchitects.de/ObjectArchitects/events/OOP2004/PersistenceOptionsOOP2004e.pdf [Consultado: 02/03/2009].

[15] M. Townsend. Exploring the Singleton Design Pattern. 2002. Disponible en: http://msdn.microsoft.com/en-us/library/ms954629.aspx. [Consultado: 18 Abril 2009]

[16] P. Boronat, M. Francisco. Concurrencia y Sistemas Distribuidos. 2003. 348 pp

[17] P. Pizarro. Arquitectura de Software: ORM, Object-Relational Mapping - I Parte. 2005. Disponible en: http://arquitectura-de-software.blogspot.com/2006/05/orm-object-relationalmapping-i-part.html. [Consultado en: 10 Febrero 2009]

[18] Sitio Oficial Debea: Database Acces Library. Disponible en: http://debea.net/trac [Consultado: 11 Abril 2009]

[19] Sitio Oficial de db4o Object Database. Disponible en: http://www.db4o.com/s/objectdb.aspx. [Consultado: 12 Abril 2009]

[20] Sitio Oficial Simple Oracle Call Interface Disponible en: http://soci.sourceforge.net/ [Consultado. 13 Abril 2009]

[21] Sitio Oficial Database Template Library Disponible en: http://dtemplatelib.sourceforge.net/ [Consultado: 15 Abril 2009]

[22] Sitio Oficial de Hibernate. Disponible en : http://www.hibernate.org [Consultado: 16 Abril 2009]

[23] Sitio Oficial LiteSql Disponible en: http://apps.sourceforge.net/trac/litesql/ [Consultado: 17 Abril 2009]

[24] T. Newards. The Vietnam of Computer Science. 2006. Disponible en: http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx [Consultado: 27 Marzo 2009]

[25] Universidad Sevilla. Modelo relacional de Codd. Estructuras y restricciones. 2004 Disponible en: http://www.lsi.us.es/docencia/get.php?id=3183 [Consultad: 4 abril 2009]

[26] Lizama, Y. Concepción, Mario. Accesibilidad y disponibilidad de datos en la plataforma de cómputo paralelo de datos: Génesis. Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2009. 105 pp.