MITOO: un framework para la migración de datos hacia bases de datos orientadas a objetos

MITOO: A Framework for the Migration of Data to Databases Object-Oriented

Paul Mendoza del Carpio*

http://dx.doi.org/10.21503/CienciayDesarrollo.2010.v12.05

RESUMEN

Este artículo presenta a MITOO, un framework para la migración de datos desde cualquier base de datos relacional hacia bases de datos de objetos. El framework sigue un enfoque basado en configuración XML. Las características de diseño del framework son presentados mediante patrones de diseño en notación UML. MITOO está enfocado en aplicaciones Java y permite la migración de datos hacia bases de datos que cumplan el estándar JDO y DB4O. Los resultados de la investigación han sido evaluados empleando métricas de diseño orientadas a objetos, las cuales demuestran las cualidades de reusabilidad del framework. Una distribución de MITOO ha sido desarrollada en el lenguaje Java y está disponible en el sitio Web de la comunidad de desarrollo de DB4O.

Palabras clave: *framework, migración de datos, base de datos, JDO.*

ABSTRACT

This article presents MITOO, a framework for migrating data from any relational database to an object database. The framework follows an approach based on SQL queries and XML configuration. The design features of the framework are presented through design patterns and design artifacts in UML notation. MITOO is focused in Java Applications and allows migrating data to JDO-compliant and DB4O object databases. The results of the research has been evaluated using metrics of object-oriented design which demonstrate the reusability qualities of the framework. A distribution of MITOO has been developed in Java language and is available in the site of the developer community of DB4O.

Key words: *framework, data migration, database, JDO.*

* Profesor de la Escuela Académico-Profesional de Ingeniería de Sistemas e Informática de la Universidad Alas Peruanas. Arequipa, Perú. e-mail: pmendozadelcarpio@yahoo.es

DOI: http://dx.doi.org/10.21503/CienciayDesarrollo.2010.v12.05

INTRODUCCIÓN

El creciente uso de los lenguajes orientados a objetos para desarrollar aplicaciones ha dado lugar en la actualidad a la creación de bases de datos orientadas a objetos (BDOO). Puesto que las aplicaciones trabajan con objetos, resulta ideal almacenar los objetos directamente en una base de datos sin tener que mapear o convertir los datos para ajustarse a algún modelo objetivo diferente al objetual. Los BDOO se han diseñado para que se puedan integrar directamente con aplicaciones desarrolladas con lenguajes orientados a objetos.[1, 10, 11]

A partir del año 2004, ha surgido un conjunto original de bases de datos BDOO que solucionan el problema de la diferencia objeto-relacional, y que son de fácil uso con el lenguaje Java (e.g. DB4O, ObjectDB, Perst, entre otros), muchas de ellas de código abierto. Entre las referidas bases de datos, DB4O merece una mención especial debido al número de casos de éxito que presenta y al tiempo que lleva en el mercado.

Los BDOO basados en programación persistente cuentan con un nicho en áreas de aplicación como la ingeniería y bases de datos espaciales, telecomunicaciones, áreas científicas (e.g. biología molecular), así como áreas de servicios financieros.[16]

Este trabajo presenta a MITOO (*MIgrating To Object-Oriented databases*), un *framework* orientado a objetos para la migración de datos desde las tradicionales bases de datos relacionales hacia bases de datos orientadas a objetos, objeto de la tesis de maestría del autor. Este es un *framework* de caja negra que cuenta con clases e interfaces para aquellos elementos necesarios en la consulta de datos en bases de datos relacionales y en la persistencia de objetos en bases de datos orientadas a objetos.[13]

Diferencia objeto-relacional

En el ámbito de aplicaciones desarrolladas empleando programación orientada a objetos (OO) y bases de datos relacionales, es posible distinguir dos paradigmas diferentes. El modelo relacional de la base de datos trata con relaciones y conjuntos, mientras que el paradigma orientado a objetos trata con objetos, atributos y asociaciones. En cuanto se requiere persistir los objetos utilizando una base de datos relacional, se observa una desavenencia entre estos dos paradigmas, la cual es denominada diferencia objeto-relacional.[12]

La solución a la incompatibilidad entre los modelos objetual y relacional puede requerir un gasto significativo de tiempo y esfuerzo. Según Bauer,[5] más del 30% del código de una aplicación en Java es para manejar la diferencia objeto-relacional, y a pesar del esfuerzo empleado, aún el resultado no es el más óptimo. Incluso algunos proyectos fracasan debido a la complejidad y falta de flexibilidad de sus capas de acceso a datos.

Frente al problema de la diferencia objeto-relacional se pueden considerar dos estrategias para lidiar con el problema en mención. La primera estrategia consiste en coexistir con el problema empleando un mapeador objeto-relacional (ORM). El mapeo entre tablas relacionales y clases de dominio permitirá ocultar la diferencia objeto-relacional al momento de implementar el acceso a datos persistentes. Las implementaciones de acceso a datos realizarán operaciones de almacenamiento y recuperación de objetos de dominio. Luego, el ORM se encargará de aplicar las operaciones sobre el modelo objetivo real, que es el modelo relacional de la base de datos, el cual está formado por tablas y registros de datos.

La segunda estrategia consiste en evitar el problema de la diferencia objeto-relacional.

Esto se lograría empleando un BDOO. Aquí no se requiere de ningún mapeo, debido a que la aplicación trabaja con objetos y la base de datos almacena y recupera también objetos. Uno de los principales objetivos de MITOO es proporcionar soporte para aquellos que deseen atender el problema de la diferencia objeto-relacional empleando un BDOO.

Estándares para bases de datos orientadas a objetos

Los BDOO han estado disponibles comercialmente alrededor del inicio de los años noventa. Un estándar para estos sistemas fue definido por ODMG (Object Data Management Group), pero no alcanzó una aceptación universal.[9]

En el año 2001 se liberó el estándar ODMG 3.0. Después de ello, en poco tiempo ODMG cedió ODMG *Java Binding* a JCP (*Java Community Process)* como base para la especificación JDO (*Java Data Objects)*. Luego, ODMG fue disuelto.

JDO es una especificación que forma parte de JPC, la cual presenta una API (Application Programming Interface) para persistencia transparente. JDO proporciona una interfaz estándar para el acceso, almacenamiento y procesamiento de objetos persistentes.

A partir del año 2001 en adelante, apareció un grupo singular de BDOO para Java. La principal característica de estas bases de datos es su facilidad de uso y persistencia transparente. MITOO trabaja sobre este mencionado grupo de bases de datos, entre los cuales se encuentran DB4O y otros que cumplen con la especificación JDO, como JDOInstruments.

Proceso de migración de datos

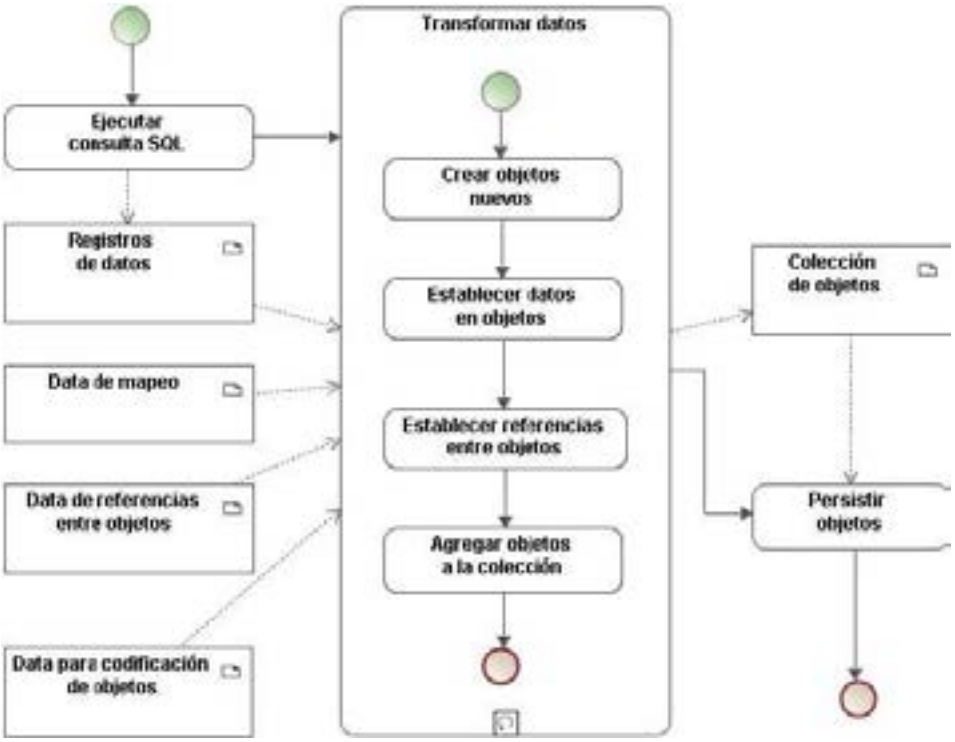
MITOO emplea un enfoque basado en consultas SQL, donde los registros de datos a migrar son obtenidos a partir de la ejecución de una consulta SQL.[13] Este enfoque presenta algunas características importantes enumeradas a continuación:

Cuadro 1: Agrupaciones y paquetes de MITOO

Agrupación	Paquetes
Soporte	pe.pmmc.mitoo.core.vo (*value objects*) pe.pmmc.mitoo.core.exception pe.pmmc.mitoo.core.exception.jdo
Manejo de clases y objetos	pe.pmmc.mitoo.core.oo
Interacción BDR	pe.pmmc.mitoo.core.jdbc pe.pmmc.mitoo.core. jdbc.connection pe.pmmc.mitoo.core.jdbc.query
Interacción BDOO	pe.pmmc.mitoo.core.oodb pe.pmmc.mitoo.core. oodb.jdo
Transformación	pe.pmmc.mitoo.core.transform pe.pmmc.mitoo. core.transform.imp
Migración	pe.pmmc.mitoo.core.migration

Fuente: elaboración propia.

Figura 1. Proceso de migración de datos



Fuente: P.M. Mendoza [13].

- Compatible con una estrategia incremental de migración de datos.
- Esquema objetivo independiente del esquema de datos de la base de datos de origen.
- Alta especificación de los registros de datos a migrar.

La figura 1 presenta el proceso de migración de datos soportado por MITOO. Luego, la interacción entre clases de MITOO para la creación de objetos desde registros de datos y la persistencia de objetos en una BDOO son descritos mediante diagramas de secuencia en las figuras 2 y 3 respectivamente.

Elementos del diseño

MITOO está conformado por un conjunto de clases cooperantes que han sido organizadas en seis agrupaciones para una mejor identificación de su propósito:[13]

- Soporte: clases que representan excepciones y objetos de valor que son pasados como argumentos o retornos en firmas de operaciones del *framework*.
- Manejo de clases y objetos: interfaces y clases para el manejo genérico de objetos.
- Interacción BDR: clases que permiten la interacción con bases de datos relacionales, por lo general consulta de datos.
- Interacción BDOO: interfaces y clases que permiten la interacción con bases de datos orientadas a objetos, en forma general, operaciones de persistencia de objetos.
- Transformación: interfaces y clases para la creación de objetos a partir de registros de datos.
- Migración: interfaces y clases relacionadas a la migración de datos.

Los elementos del *framework* están organizados por paquetes. En la figura 4 se presenta el diagrama de paquetes del núcleo de MITOO. Cada

paquete contiene elementos que dan soporte a las agrupaciones mencionadas anteriormente. El cuadro 1 muestra esta correspondencia.

MITOO presenta una estructura general para aplicaciones cliente, donde la base de datos relacional, origen de datos, puede encontrarse en un servidor accesible desde el computador donde se ejecuta la aplicación de migración de datos.[13] La figura 5 presenta el diagrama de despliegue para aplicaciones de migración de datos con MITOO.

Patrones de diseño

El uso de patrones de diseño[7] dentro del *framework* ha sido de relevancia para posibilitar la implementación de aplicaciones con diversos escenarios de uso. A continuación se presentan los patrones de diseño presentes en el *framework*.

Creación de objetos. El *framework* presenta una interfaz para la creación de objetos, a implementar por los desarrolladores siguiendo el patrón de diseño *Factory Method*.

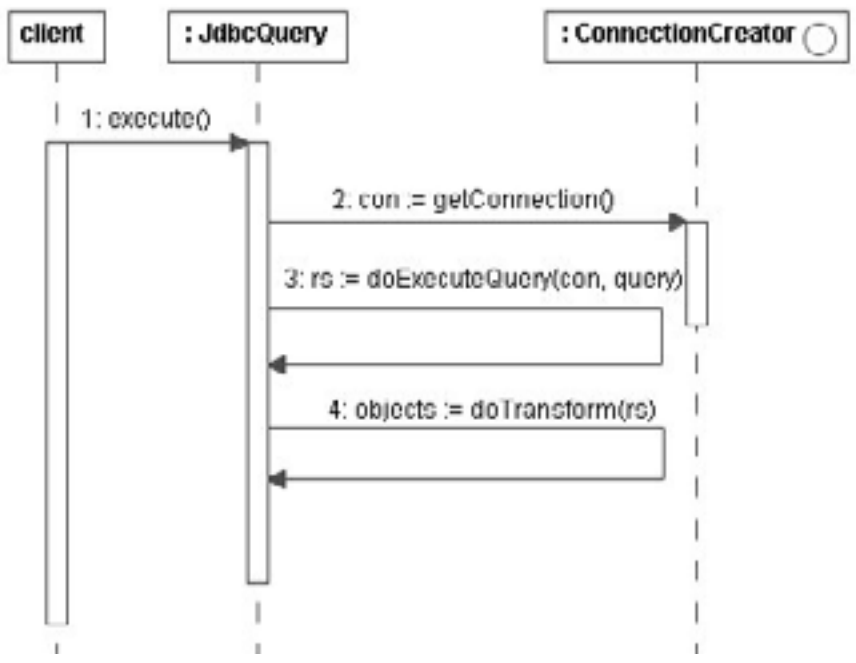
Conexión a bases de datos relacionales. MITOO emplea el patrón *Strategy* para la obtención de conexiones a bases de datos relacionales vía JDBC, y emplea cualquiera de ellos en forma indiferente.

Consultas SQL. El *framework* incluye soporte para la ejecución de consultas normales y consultas parametrizadas. Se emplea el patrón *Template Method*, postergando la implementación de la ejecución de la consulta SQL hacia las clases concretas

Interacción BDOO. MITOO comprende clases que posibilitan el almacenamiento y recuperación de objetos persistentes desde BDOO. El patrón *Strategy* es empleado para poder interactuar con diversos proveedores de BDOO en forma indiferente.

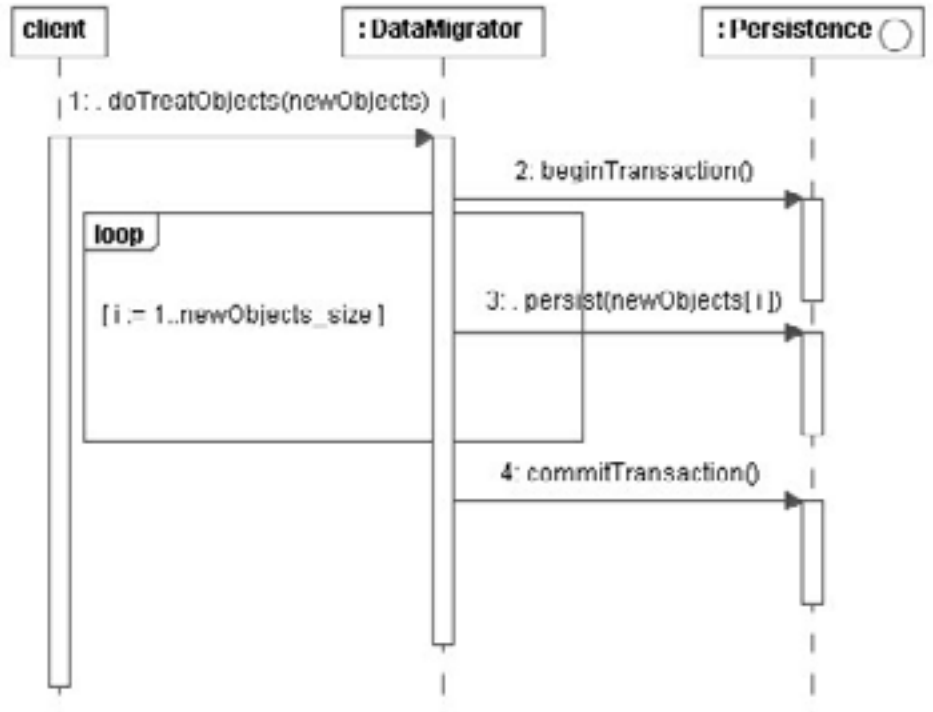
Transformación de registros de datos. Una vez que se obtienen los registros producto de

Figura 2. Creación de nuevos objetos en MITOO



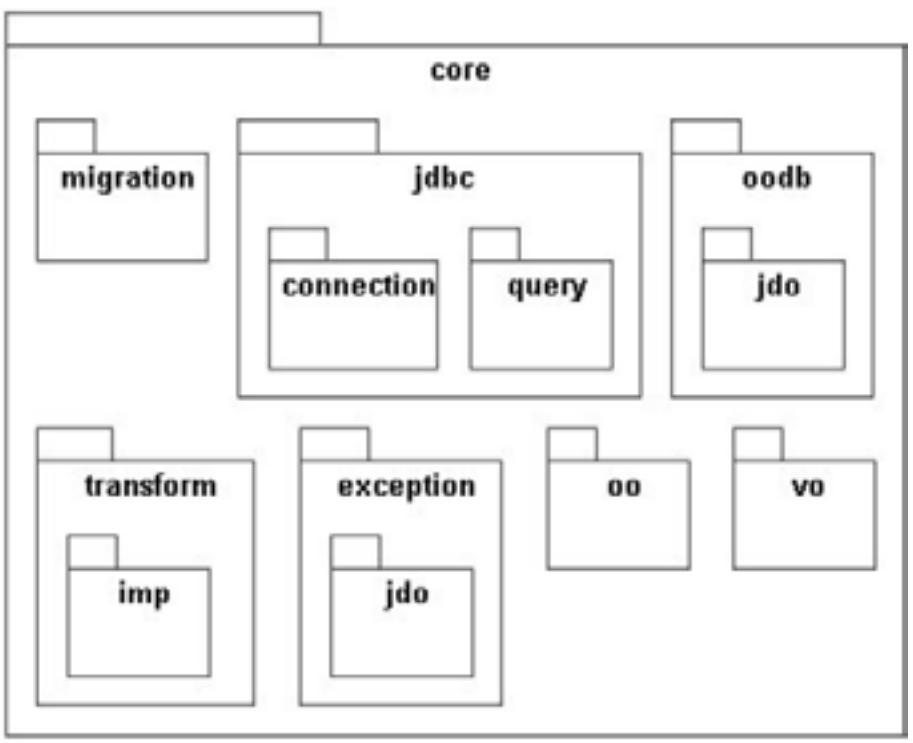
Fuente: P.M. Mendoza.[13]

Figura 3. Persistencia de objetos en MITOO



Fuente: elaboración propia.

Figura 4: Diagrama de paquetes de MITOO



Fuente: P.M. Mendoza.[13]

consultas SQL sobre una base de datos relacional, se recorre el conjunto de registros obtenidos, realizando una operación sobre el conjunto. Esta operación es considerada como una transformación dentro del *framework*, y consiste en trabajar con los datos obtenidos y producir algún objeto de valor para la aplicación. Mediante el uso del patrón *Strategy*, distintos transformadores concretos pueden ser utilizados indiferentemente dentro del *framework*.

En caso que se requiera realizar alguna operación adicional sobre cada registro que se recorre, MITOO proporciona una clase a ser extendida como una implementación del patrón *Decorator*.

Migración de datos. MITOO encapsula las operaciones de migración en comandos a ser invocados desde un ejecutor de comandos. Aquí se sigue el patrón *Command*.

Escenarios de uso

MITOO atiende seis escenarios de migración de datos donde los nuevos objetos son hechos persistentes. Los escenarios son descritos a continuación:

1) Registros de datos hacia objetos de una clase singular: un nuevo objeto es creado por cada registro de datos. Los objetos nuevos son instancias de una clase determinada.
2) Registros de datos hacia objetos de clases múltiples: varios objetos de diferentes clases son creados por cada registro de datos.
3) Registros de datos hacia objetos referenciados: varios objetos de diferentes clases son creados por cada registro de datos. Adicionalmente, se establecen referencias entre los nuevos objetos.
4) Registros de datos hacia objetos de una clase singular, sin repeticiones: un nuevo objeto es creado por cada registro de datos. Estos nuevos objetos son instancias de una misma clase. Se evita la repetición de objetos con los mismos datos a través del establecimiento de campos clave.
5) Registros de datos hacia objetos de clases múltiples, sin repeticiones: varios objetos de diferentes clases son creados por cada registro de datos. Se evita la repetición de objetos con los mismos datos a través del establecimiento de campos clave.
6) Registros de datos hacia objetos referenciados, sin repeticiones: varios objetos de diferentes clases son creados por cada registro de datos. Se evita la repetición de objetos con los mismos datos a través del establecimiento de campos clave. Adicionalmente, se establecen referencias entre los objetos nuevos.

Aplicación de ejemplo

Esta sección presenta una aplicación de ejemplo que migra datos de libros. Los objetos a crear son instancias de las clases presentadas en el diagrama de la figura 7. Para este ejemplo, la BDOO a emplear es DB4O 5.2.

Se tienen dos clases para la realización de la migración de datos: *SimpleMigration*, *CompleteMigration*. *SimpleMigration* trabaja sobre el escenario 1) y migra solo datos de autores de libros. *CompleteMigration* trabaja sobre el escenario 6) y migra datos de libros, autores y editores. Ambas clases emplean un comando para migrar datos.

El archivo *mitoo-database-access.xml* especifica las bases de datos involucradas:

```
<bean id="ConnectionCreator"
 class="pe.pmmc.mitoo.core.jdbc.
 connection.DriverManagerCreator">
  <property name="driverClassName"
   value="org.postgresql.Driver"/>
  <property name="password"
   value="postgres"/>
  <property name="url"
   value="jdbc:postgresql://
```

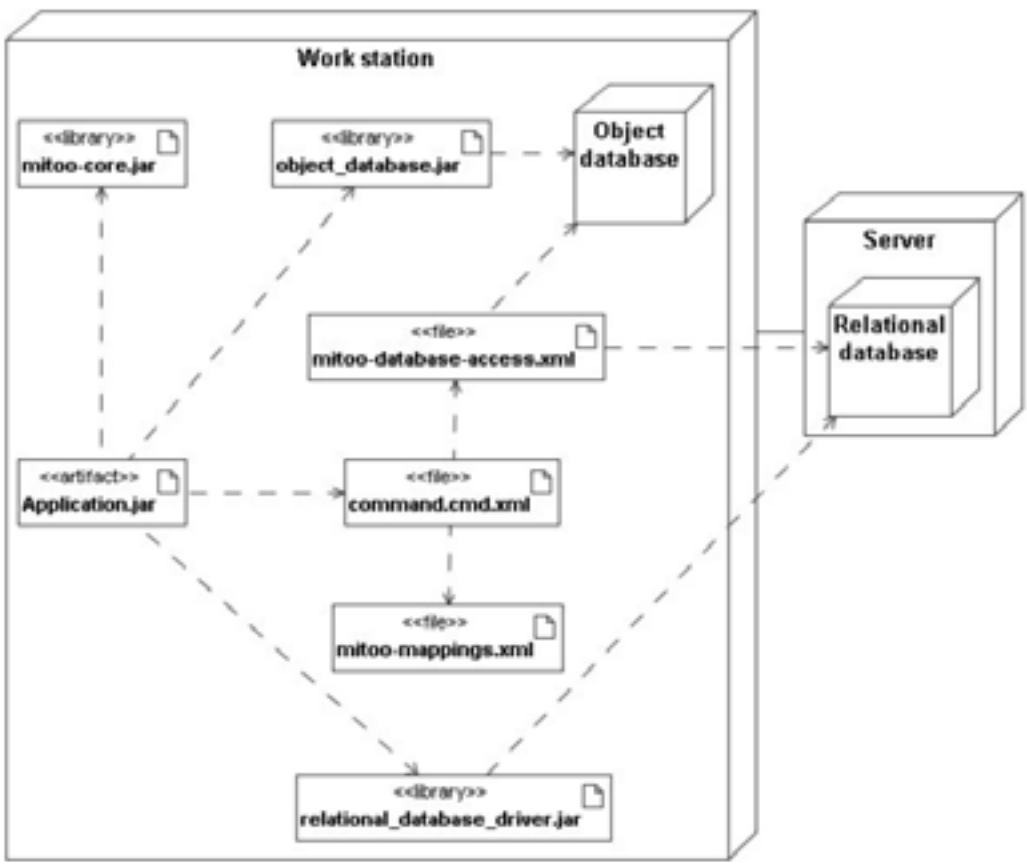
```
   localhost/booksdb"/>
  <property name="userName"
   value="postgres"/>
</bean>
<bean id="Persistence"
 class="pe.pmmc.mitoo.core.oodb.
 Db4oPersistence">
  <constructor-arg>
    <value>booksobjects.yap</value>
  </constructor-arg>
</bean>
```

Connection Creator es la fábrica de conexiones para acceder al origen de datos: la base de datos booksdb. Persistence es la clase para acceder a la base de objetos, una base de datos DB4O denominada *booksobjects*.

En la presente aplicación de ejemplo se emplea una base de datos DB4O como base de datos objetivo. Si se requiere emplear una BDOO que siga la especificación JDO, se puede emplear la clase *JdoPersistence* de MITOO, la cual requiere como argumento el nombre de la clase que cumplirá la función de crear objetos *PersistenceManager* de JDO. MITOO proporciona la clase *ProxyPmCreator* para la creación de objetos *PersistenceManager* en aplicaciones que empleen la BDOO JDOInstruments. A continuación se presenta un ejemplo de la configuración para el uso de persistencia con JDOInstruments en MITOO:

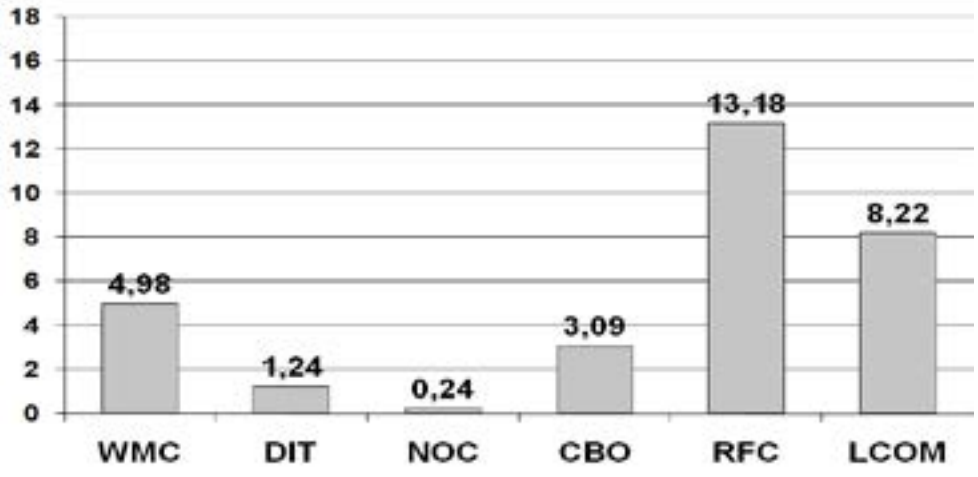
```
<bean id="PMCreator"
 class="pe.pmmc.mitoo.core.oodb.jdo.
 ProxyPmCreator">
  <constructor-arg
   type="java.util.Properties">
    <props>
      <prop key=
       "ClassBaseDirectory">
        d:/test/classes/
```

Figura 5. Diagrama de despliegue para aplicaciones con MITOO



Fuente: P.M. Mendoza.[13]

Figura 6. Métricas CK para MITOO



Fuente: elaboración propia.

```
      </prop>
      <prop key=
      "javax.jdo.
    PersistenceManagerFactoryClass">
        org.jdoinstruments.
      PersistenceManagerFactoryProxy
      </prop>
    </props>
  </constructor-arg>
</bean>

<bean id="Persistence"
 class=
  "pe.pmmc.mitoo.core.oodb.jdo.
   JdoPersistence">
   <constructor-arg>
     <ref bean="PMCreator" />
   </constructor-arg>
</bean>
```

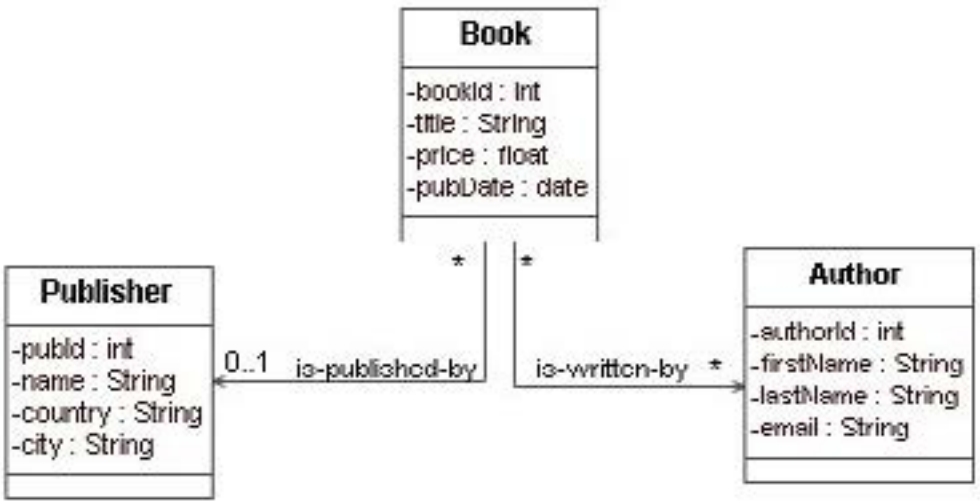
El archivo *mitoo-mappings.xml* especifica los mapeos entre campos de registros y los atributos de los objetos a crear:

```
<util:map id="MapBook" >
  <entry key="BOOK_ID" value="id"/>
  <entry key="TITLE" value="title"/>
  <entry key="PRICE" value="price"/>
  <entry key="PUB_DATE"
    value="pubDate"/>
</util:map>

<util:map id="MapAuthor">
  <entry key="AUTHOR_ID"
   value="id"/>
  <entry key="FIRST_NAME"
   value="firstName"/>
  <entry key="LAST_NAME"
   value="lastName"/>
  <entry key="EMAIL" value="email"/>
</util:map>
    <util:map id="MapPublisher">
  <entry key="PUB_ID" value="id"/>
  <entry key="NAME" value="name"/>
  <entry key="COUNTRY"
   value="country"/>
  <entry key="CITY" value="city"/>
</util:map>
```

El comando usado en *SimpleMigration* es configurado en authors.cmd.xml:

```
<bean id="authors"
 class="pe.pmmc.mitoo.core.
 migration.command.SingleMigrator">
  <property name="query">

    <value>SELECT * FROM
     AUTHORS</value>
  </property>
  <property
   name="connectionCreator">
    <ref bean="ConnectionCreator"/>
  </property>
  <property name="persistence">
    <ref bean="Persistence"/>
  </property>
  <property name="namedClassData">
    <ref bean="NamDatAuthor"/>
  </property>
</bean>
```

Figura 7: Diagrama de clases de la aplicación de ejemplo



Fuente: elaboración propia.

El archivo XML del comando configura la consulta SQL para obtener los datos a migrar. Luego, el cuerpo principal de la aplicación invoca al comando como se muestra a continuación:

```
public static void main(
      String[] args) {
   final CommandExecutor executor
     = new CommandExecutor();
   executor.addCommand(
     "commands/reset");
   executor.addCommand(
     "commands/authors");
   executor.executeCommands();}
```

RESULTADOS

El escaso código procedimental presente en la aplicación de ejemplo da muestra del reducido esfuerzo requerido por el desarrollador cuando utiliza MITOO. Por otro lado, el desarrollador requiere conocimiento de programación declarativa para la elaboración de XML.

Las métricas CK de Chidamber y Kemerer[6] fueron empleadas para proporcionar indicadores cuantitativos de la calidad del diseño de MITOO. El gráfico de la figura 6 muestra los valores de métricas calculados para las clases de MITOO.

Si bien MITOO se presenta como un *framework* para la migración de datos a bases de datos de objetos, su diseño soporta la utilización de los objetos en otros contextos, puesto que parte del proceso incluye la creación de colecciones de objetos. Luego de esta creación, los objetos pueden ser manipulados para un objetivo específico, que en el caso normal MITOO considera almacenarlos en una base de datos de objetos.

CONCLUSIONES

MITOO apoya el empleo de persistencia transparente en aplicaciones sobre la plataforma Java brindando soporte para la migración de datos hacia BDOO. El diseño del *framework* permite al desarrollador una más fácil y no exhaustiva implementación de aplicaciones para migración de datos hacia BDOO. Adicionalmente, MITOO cuenta con características de configuración que permiten cambiar el modo de operar de la aplicación en el escenario requerido mediante el uso de archivos XML.

El espacio de proyecto de MITOO se encuentra ubicado en la Web en la comunidad de

desarrolladores de DB4O, desde donde se puede obtener una distribución de MITOO así como documentación acerca del mismo:

http://developer.db4o.com/ProjectSpaces/view.aspx/MITOO

REFERENCIAS BIBLIOGRÁFICAS

1. A. Behm, A. G. y. K. D. "On the migration of relational schemas and data to object oriented database systems". In: *5th International Conference on Re-Technologies in Information Systems* (diciembre de 1997).
2. A. Singh, K. S. Khalon, J. S. R. S. S. S. y. D. K. "Mapping relational database schema to object-oriented database schema". *Transactions on engineeering, computing and technology VI* (diciembre de 2004).
3. B. Wu, D. Lawless, J. B. R. R. J. G. B. W. D. O. . The butter and methodology: "A gateway-free approach for migrating legacy information systems". In *3rd IEEE Conference on Engineering of Complex Computer Systems* (ICECCS97) (Septiembre de 1997).
4. Behm, A. *Migrating Relational Databases to Object Technology*. Ph.D. thesis, Wirtschaftswissenschaftlichen FakultÄat, UniversitÄat ZÄurich, 2001.
5. C. Bauer y G. King. *Hibernate in Action*. Manning Greenwich, 2005.
6. Chidamber, S. R. y Kemerer, C. F. *A Metrics Suite for Object Oriented Design*. IEEE Transactions on Software Engineering, 1994.
7. E. Gamma, R. Johnson, R. Helm y Vlissides, J. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
8. Fayad, M. E., and Schmidt, D. C. "Lessons learned: Building reusable OO frameworks for distributed software". *Communications of the ACM* 40, 10 (octubre de 1997).
9. J. Paterson, H. Häorning, S. Edlich y Häorning, R. *The Definitive Guide to DB4O*. Apress, 2006.
10. Horstmann, J. "Migration to open source databases". *Tech. Rep*., Technical University Berlin, Septiembre 2005. Diploma thesis.
11. Lin, C. *Object-oriented database systems: A survey*. Abril, 2003.
12. M. Glögs. *Persistencia de objetos Java: el camino hacia Hibernate*. Trad. J.A. Palos. 2005.
13. P. M. Mendoza. *Marco de trabajo para la migración de datos de bases de datos relacionales a bases de datos orientadas a objetos*. Tesis de Maestría. Universidad Nacional de San Agustín, Arequipa, 2008.
14. S. Tyagi, K. McCammon, M. V. y H. B. *Core Java Data Objects*. Prentice Hall, 2003.
15. Taligent Incorporated. *Building Object-Oriented Frameworks*, 1994. A Taligent White Paper.
16. Wikipedia. *Object database*. 2006. Actualizado el 28 de marzo de 2006.