Desarrollo de Sistemas de Información

Tema 5. Arquitectura de las aplicaciones con acceso a BD

Marta Elena Zorrilla Pantaleón

Dpto. de Matemáticas, Estadística y Computación

Referencias

- **Libros**
  - Scott W. Ambler. The Design of a Robust Persistence Layer For Relational Databases. 2005 http://www.ambysoft.com/downloads/persistenceLayer.pdf
  - Clinton Begin, Brandon Goodin, Larry Meadors. iBatis in action. New York: Manning, cop. 2007.
  - César de la Torre et al. Guía de Arquitectura N-Capas DDD .NET 4.0. Krassis Press, 2011

Referencias

- **Recursos web**
  - iBatis http://ibatis.apache.org/
  - Hibernate http://www.hibernate.org/
  - Comparativa Frameworks: http://crowdranking.com/crowdrankings/t420g0--best-java-persistence-frameworks
  - Data Skills for Agile Software Developer http://www.agiledata.org/essays/developerSkills.html
  - Encapsulating Database Access: An Agile "Best" Practice http://www.agiledata.org/essays/implementationStrategies.html
  - Scott Ambler's Articles and Other Writings http://www.ambysoft.com/onlineWritings.html
  - Comparison and Benchmarks on ORMBattle.NET http://ormeter.net/

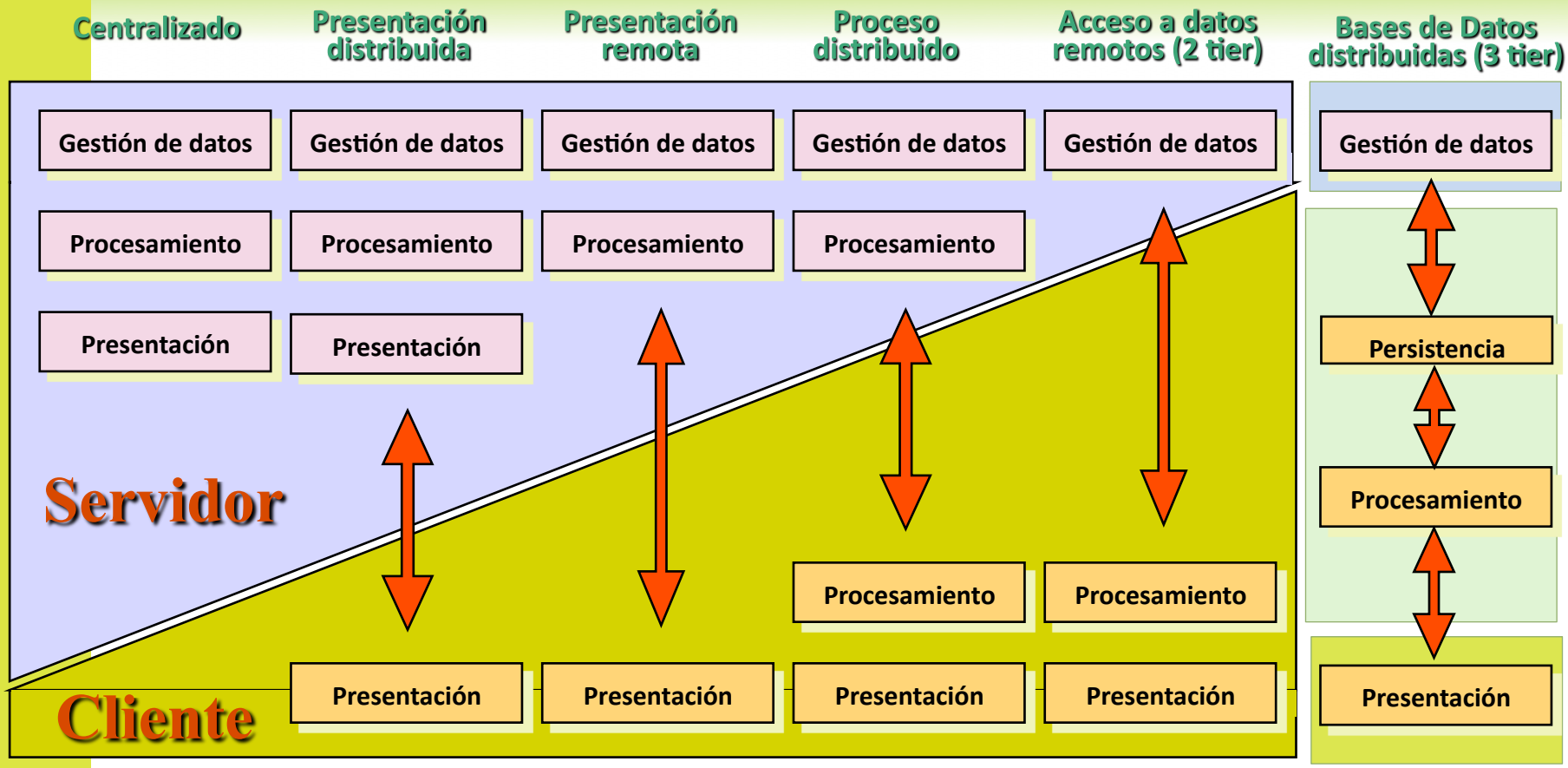
Introducción

- Mayoritariamente, los Sistemas de Información hacen uso de *bases de datos relacionales* para la persistencia de sus datos. En menor medida se usan *BD orientadas a objeto* y/u *Objeto-relacionales*. Y está tomando relevancia las *BD XML*, como consecuencia del intercambio de datos y estándares de la industria.
- *SQL*, estándar desde 1987, es el lenguaje de interrogación y definición ampliamente adoptado. Ha sido extendido para abordar el modelado de los datos objetual (OR) y semi-estruturado (XML). En vigor, estándar SQL:2008.
  - Limitaciones: sólo una parte de SQL es realmente estándar cuando trabajas con distintos gestores de BD
  - A favor: declarativo, potente, ampliamente utilizado

Introducción

- Por otra parte, los sistemas de información se ven muy afectados por los cambios normativos y organizativos de las corporaciones, lo que conlleva a cambios en los requisitos y continuas adaptaciones en el software.
- Paralelamente las tecnologías no paran de avanzar y de proponer gran cantidad de alternativas que permiten construir aplicaciones, que siguiendo una arquitectura modular, se adapten muy bien a diferentes escenarios.
- ¿Cómo abordarlo?
  - Analicemos la historia

REVISIÓN HISTÓRICA



Desde los años 80 hasta la actualidad la arquitectura de las aplicaciones se ha ido modificando pasando desde una arquitectura centralizada (todo en el host) a una distribuida, en el que se reparte el aplicativo en varias máquinas (cliente y servidores) aprovechando así las capacidades gráficas y de cómputo de cada tipo de equipo

Layers and Tiers

- Es importante distinguir los conceptos de "**Capas**" (*Layers*) y "**Niveles**" (*Tiers*)
- Las Capas (*Layers*) se ocupan de la división lógica de los componentes que constituyen el software, y no tienen en cuenta dónde estará ubicados físicamente.
- Los Niveles (*Tiers*) se refieren a los distintos equipos en donde los componentes software se desplegarán.
- Ambos usan nombres similares (presentación, proceso/ servicio, negocio y datos), pero es importante no confundirlos.

Características de las aplicaciones

- Aplicaciones centralizadas (1 tier) (desde 1970 a ..):
  - Interfaces orientadas a texto
  - Desarrollado principalmente con Cobol, y lenguajes 4GL que embebían el lenguaje SQL del gestor de BD (Informix-4GL, Oracle, etc.)
  - Aplicaciones eficientes, ágiles, pero poco usables
- 2 tier (1990-95):
  - Interfaces gráficas en cliente con capacidad de procesamiento.
  - Se basaba en la invocación de instrucciones SQL desde cliente. A veces estas instrucciones eran procedimientos almacenados que, incluían no sólo el SQL sino también lógica de negocio. Se llevaba todo el cómputo al servidor.
  - Recordemos que los procedimientos almacenados siguen lenguaje procedimental y no declarativo. Impide portabilidad.
  - Las aplicaciones presentaban problemas de escalabilidad y rendimiento y su despliegue era una locura “el infierno de las dlls”
  - Herramientas: VB, PowerBuilder, Oracle Forms,… con API ODBC. Prob. Con el pool de conexiones

Características de las aplicaciones

- 3-tier o N-tier (1994-…):
  - Interfaces “thin” y “thick”
  - Surgen con la llegada de Internet y el protocolo http.
  - Se divide la aplicación en capa de presentación, de negocio y de datos.
  - Los **procedimientos almacenados** se llevan a la capa de negocios (como llamadas a procedimientos remotos) donde se mejora el rendimiento con una gestión más optimizada del pool de conexiones y los recursos de gestor.
  - Estos son muy eficientes en cuanto a que manipulan la información que está en el propio gestor pero son más difíciles de escribir y probar pues no están ligados a las metodologías software actuales (orientadas a objetos vistos desde la perspectiva de aplicación).
  - Algunos gestores ya permiten la programación de procedimientos en java o C# pero es un parche que solo resuelve la portabilidad de la BD.

Características de las aplicaciones

- 3-tier o N-tier (1994-…):
  - **Inline SQL**: a continuación aparecieron lenguajes como SQLJ (estandarizado en SQL2003) que embebía instrucciones SQL en java pero no ha sido muy utilizado.
    - SQL no estándar y, por tanto, no se puede hacer un parser general
    - Necesidad de un precompilador
    - Dificultad para integrarlo en los IDE de desarrollo
  - **Dynamic SQL**: resuelve el problema del precompilador, escribiendo el SQL en un string. Esto requiere una API para configurar los parámetros y recuperar los datos (JDBC, ADO Net)
    - Ventaja: flexibilidad para construir dinámicamente las SQLs
    - Inconveniente: mucho código repetitivo cada vez que quieres ejecutar una consulta. Además el SQL puede venir concatenado lo que resulta difícil de leer y mantener

EJ. SQLJ Y JDBC

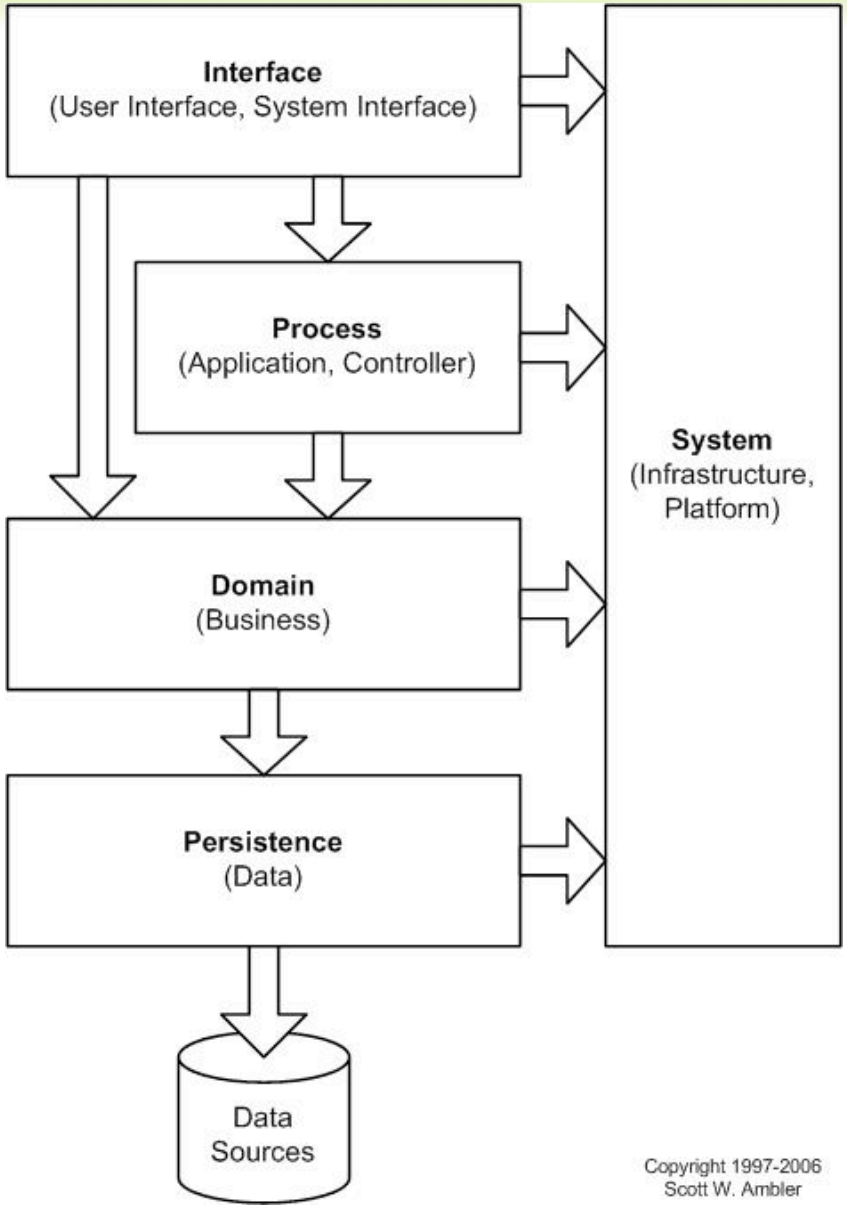
SQLJ
 	
estándar
estático
	#sql [ctx] {
	SELECT MAX(SALARY), AVG(SALARY)
	INTO :maxSalary, :avgSalary
	FROM DSN8710.EMP
	};

JDBC
 	
no estándar
dinámico
	PreparedStatement stmt = conn.prepareStatement(
	"SELECT MAX(SALARY), AVG(SALARY)" +
	" FROM DSN8710.EMP");
	rs = stmt.executeQuery();
	if (!rs.next()) { // Error -- no rows found }
	maxSalary = rs.getBigDecimal(1);
	avgSalary = rs.getBigDecimal(2);
	if (rs.next()) { // Error -- more than one row found }
	rs.close();
	stmt.close();

Características de las aplicaciones

- 3-tier o N-tier (1994-…):
  - Actualmente, la solución adoptada es diseñar una capa que realice el mapeo objeto-relacional.
  - Sus siglas **ORM** se refieren al mapeo de objetos del software de aplicación a relaciones o tupas del modelo relacional para su persistencia y recuperación posterior.
  - Su uso se extiende también a otras tecnologías no relacionales (BD XML, ficheros, etc.)
  - Ventajas:
    - Muchas herramientas ORM generan código SQL directamente, realizan convenientemente la gestión de transacciones, ofrecen posibilidades de caching
  - Algunas, están diseñadas con ciertas reglas que obligan a que la BD está normalizada perfectamente, lo que influye negativamente en su rendimiento. Muchas empresas construyen su propio ORM.

Arquitectura Lógica



- **User-interface classes should not directly access your persistence mechanisms.**
- **Domain classes should not directly access your persistence mechanisms.**
- **The class-type architecture is orthogonal to your hardware/network architecture**

Arquitectura Física

Class type	Stand Alone	Fat-client	Thin –client	N-tier
User Interface	Client	Client	Client	Client
Controller/ process	Client	Client	Server	Application server
Domain/ business	Client	Client	Server	Application server
Persistence	Client	Server	Server	Database Server
System	Client	All machines	All machines	All machines
	1 tier- c/s	2 tier- c/s		3 tier- c/s

Necesidad de la Capa de Persistencia

- **Persistencia** es la capacidad de un lenguaje de programación o entorno de desarrollo de programación para, almacenar y recuperar el estado de los objetos de forma que sobrevivan a los procesos que los manipulan.
- **¿Es necesaria?** Si, debido al desajuste de impedancia entre el modelo de datos de los paradigmas de programación orientados a objetos y el de los gestores de bases de datos (relacionales, OR, xml, etc). Para el caso relacional,
  - Modelo de objetos (objetos con atributos, métodos y relaciones con otros objetos) debe descomponerse en varias tablas (de acuerdo a las relaciones 1:N y N:M) para almacenarse y su recuperación requiere varios JOINs
  - En los gestores relacionales el acceso es asociativo, esto es, basado en el contenido (valor de clave) y no navegacional, basado en el movimiento entre registros individuales.
  - Los gestores provee tipos de datos que no tienen los lenguajes de programación OO (p. ej. Interval, Date).

Requisitos a Cumplir

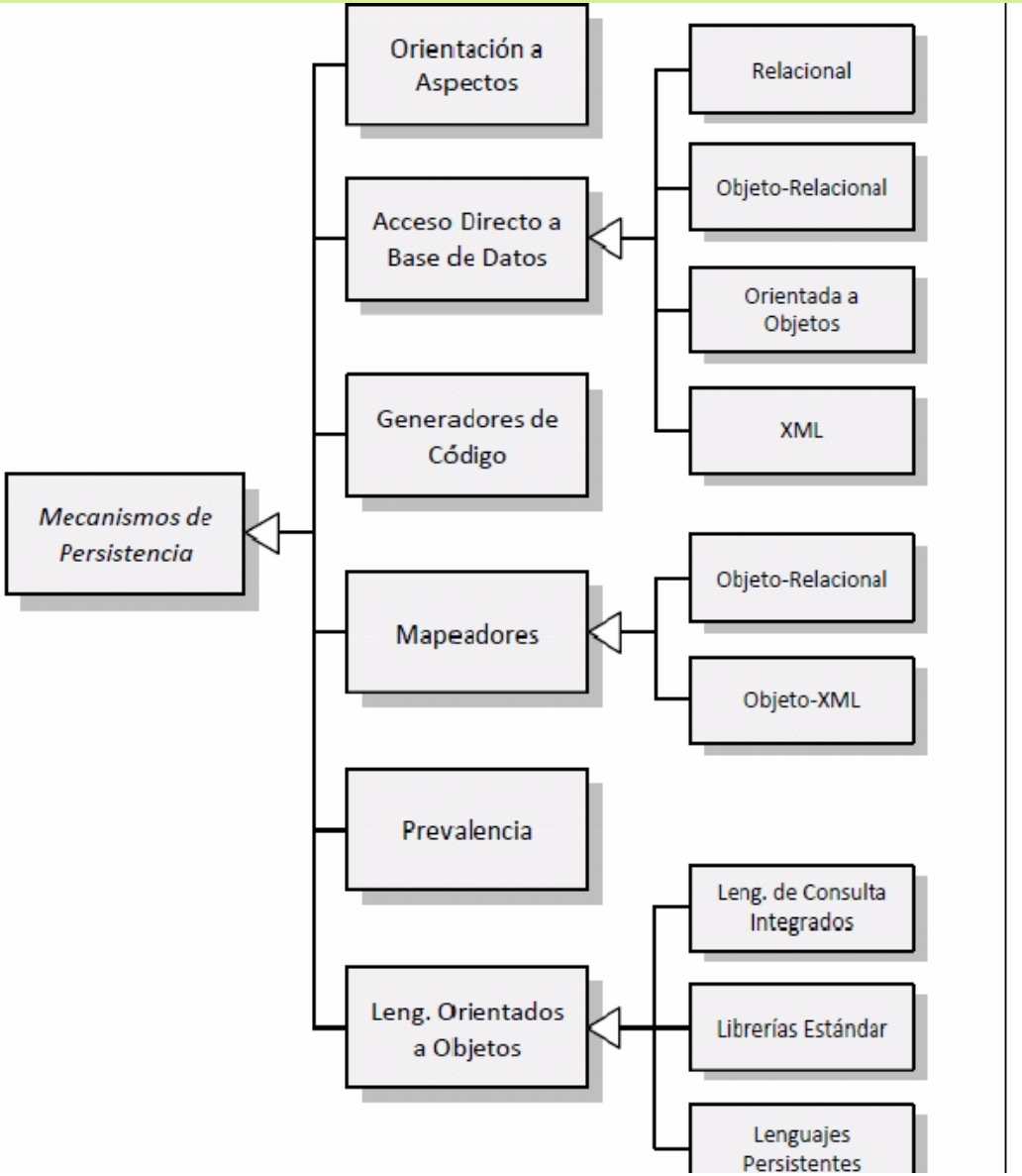
- La capa de persistencia encapsula el comportamiento necesario para escribir, recuperar, actualizar y borrar objetos (de la aplicación software) a/desde el gestor de almacenamiento persistente (relacional, XML, ficheros, etc.).

- Esta debe soportar, de acuerdo a Scott Ambler (2005):
  - Diferentes mecanismos de persistencia (relacional, XML, ficheros, etc.).
  - Encapsulación completa: no debe ser necesario escribir ninguna línea de código que acceda al gestor de persistencia, deben ser suficiente los métodos definidos en la capa de persistencia
  - Acciones sobre varios objetos al tiempo (borrar varios objetos o consultarlos, etc.)
  - Transacciones simples y anidadas (todo o nada o con commit parcial)
  - Extensibilidad: fácil para sustituir los mecanismos de persistencia existentes y poder incorporar nuevas características
  - Identidad de objetos persistentes: la capa de persistencia debe asociar un identificador a cada objeto persistente que sirva para localizar de forma unívoca el objeto guardado en el gestor de persistencia

Requisitos a Cumplir

- Esta debe soportar (cont.):
  - Cursores: debe poder recuperar el número de objetos que se le pidan y evitar el manejo de volúmenes grandes de datos
  - Proxies: complementa al cursor, un objeto que representa a otro pero sin traerlo a memoria hasta que se solicite expresamente
  - Múltiples arquitecturas: debe poder adaptarse a cualquier tipo de arquitectura (centralizada, c/s, n-capas)
  - Diferentes gestores: si se produce un cambio de gestor las aplicaciones no se ven afectadas
  - Múltiples conexiones: debe permitir abrir conexiones a diferentes bases de datos dentro de una misma aplicación
  - Suporte a drivers nativos y no nativos (JDBC, ODBC,…)
  - Consultas SQL: aunque se debe evitar el SQL en el código OO, en ocasiones es necesario por rendimiento y debe permitirse.

Mecanismos de Persistencia



Scott Ambler, 2005

Mecanismos de Persistencia

- *Acceso directo a BD*. Implica el uso directo de la BD haciendo uso de una interfaz estandarizada y un lenguaje de consulta soportado por el gestor. No existe capa de persistencia entre el gestor y la aplicación.
- *Mapeadores*: mecanismos que se basan en la traducción bidireccional entre los datos encapsulados en los objetos de la lógica de un Sistema OO y el modelo de datos de la fuente de datos. El mapeador es el encargado de realizar la conversión entre paradigmas, bien de forma manual o automática.
- *Generadores de código*: usar herramientas generadores de código basadas en patrones para resolver la persistencia, centrándose el desarrollador en el código que resuelve la lógica de negocio.

Mecanismos de Persistencia

- *Orientación a aspectos*: paradigma orientado a modularizar las aplicaciones en determinados conceptos, uno de ellos es la persistencia
- *Lenguajes orientados a objetos*: utilizar funcionalidades de persistencia provistas por el propio lenguaje de programación evitando la inclusión de frameworks.
  - Librerías estándar usando técnicas como la serialización de objetos (no gestiona transacciones ni incorpora lenguajes de consulta); lenguajes que provean persistencia o usando lenguajes que integren un lenguaje de consulta sobre el sistema persistente como OQL, HQL.
- *Prevalencia*, técnica para hacer persistentes los datos que están en memoria ppal. usando técnicas de serialización. Manejan transacciones y pueden recuperar un estado estable si cae el sistema.

Frameworks de Persistencia

- Framework es un conjunto de librerías o clases que proporciona una infraestuctura que actúa de soporte para desarrollar aplicaciones. Esto es, es un esquema o patrón para el desarrollo de una aplicación completa o de una parte específica de la misma.
- Los frameworks, a diferencia de un conjunto de librerías, ofrecen una serie de servicios para implementar el artefacto software ocultando su complejidad al programador y ahorrando tiempo de programación. Por otra parte, está más limitado un cambio en su funcionalidad.
- Framework de persistencia es un marco de trabajo que se sitúa entre la lógica de negocio y la capa de base de datos, abstrayendo uno del otro.

Frameworks para JAVA

- Ibatis:
  - **MyBatis** is a persistence framework available for Java and .NET that couples objects with stored procedures or SQL statements using an XML descriptor or annotations.
- Hibernate:
  - **Hibernate** is an object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database.
- OpenJPA**:**
  - **OpenJPA** is an open source implementation of the Java Persistence API specification. It is an object-relational mapping (ORM) solution for the Java language, which simplifies storing objects in databases
- EclipseLink
  - **EclipseLink** is the open source Eclipse Persistence Services Project from the Eclipse Foundation. The software provides an extensible framework that allows Java developers to interact with various data services, including databases, web services, Object XML mapping (OXM), and Enterprise Information Systems (EIS). EclipseLink supports a number of persistence standards

Otros frameworks para Java

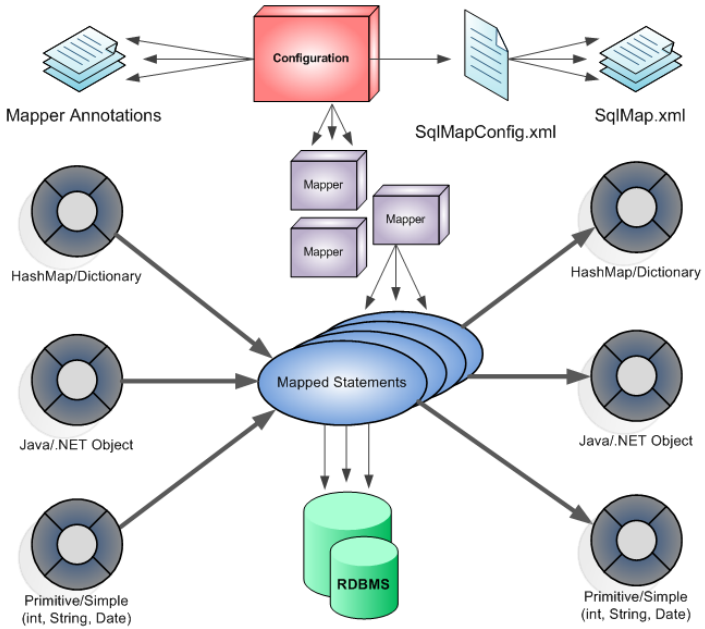
- Torque
- Castor
- OJB
- Cayenne
- JDBM
- PBeans
- Simple ORM
- JULP (Java Ultra-Lite Persistence)
- Smyle
- Speedo
- XORM
- JDBCPersistence
- JDO Genie
- LiDO
- JDO Toolkit
- FrontierSuite
- Jrelay
- IntelliBO
- kodoJDO
- ….

Otros frameworks para Net

- Sql client
- Entity Framework
- LINQ to SQL
- BTLToolkit
- DataObjects.Net
- LinqConnect
- Nhibernate
- OpenAccesss
- Etc….

iBatis

- iBatis es un framework que facilita el diseño de la capa de persistencia utilizada en las aplicaciones Java para acceder a nuestro repositorio de datos.
- iBatis une los *Objetos* con las *sentencias SQL* mediante un *descriptor XML*.
- iBatis necesita uno o más ficheros con las sentencias SQL que usará el programa
- Ibatis necesita un fichero de configuración en XML donde se indiquen los parámetros de conexión a la base de datos y los ficheros de mapeo XML



Hibernate

- Hibernate es una herramienta de Mapeo objeto-relacional para la plataforma Java (y disponible también para .Net con el nombre de NHibernate) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.

- Hibernate no requiere conocer el modelo de datos al cual se accede. Se tiene una aplicación Orientada a Objetos y él mapea el diseño OO en una capa de persistencia.

- La mayor diferencia entre Hibernate e iBATIS proviene del hecho de que el último basa su funcionamiento en el mapeo de sentencias SQL que se incluyen en ficheros XML. Eso significa que, al contrario que Hibernate, requiere conocimiento de SQL por parte del programador. Por otra parte, permite la optimización de las consultas, ya sea con lenguaje estándar o con SQL propietario del motor de base de datos utilizado. Con iBATIS, siempre se sabe lo que se está ejecutando en la base de datos y permite generar consultas dinámicas muy potentes.

Hibernate vs ibatis

- Hibernate facilita el desarrollo al no tener por que tener conocimiento alguno acerca de SQL, pero se desaconseja para desarrollos en los que la definición del modelo de datos está ya definido o si las relaciones en el modelo de datos van a ser complejas.
- Por otra parte, Hibernate al utilizar su propio lenguaje de consulta de datos, lo convierte en multimotor de base de datos.
- Ibatis soporta la posibilidad de almacenar consultas en la caché (usa OpenSymphony Cache) y pool de conexiones (usa Yacarta).
- En definitiva, Hibernate separa más la capa de negocio y persistencia que Ibatis en la que el desarrollador “define” el enlace (menos nivel de abstracción).
- Usaremos Ibatis en las prácticas.