Jug meeting 30-10-04

Dessì Massimiliano

desmax74@yahoo.it

Quando aiutano i JDO ?

- Permettono la persistenza “ad oggetti”,“mascherando” il gap esistente tra un database relazionale e il linguaggio ad oggetti usato nelle applicazioni java.
- Adatti quando il dominio degli oggetti prevede un flusso load/edit/store
- Adatti quando esiste un sufficiente mappaggio “naturale” tra il dominio degli oggetti e le tabelle
- Quando gli oggetti possono essere acquisiti in grande numero, ma aggiornati o cancellati individualmente

- Con JDO si intende uno standard per la persistenza di oggetti java, utilizzabile su tutti gli ambienti java (da J2ME a J2EE).

- I JDO non fanno parte della J2EE ma sono integrabili con essa, si occupano soltanto del concetto di persistenza.

- Oggetti locali.

- Persistenza ricca di funzionalità.

- Sviluppo semplificato, contenuto informativo dei descrittori di persistenza (metadati) ridotto.
- Trasparenza assoluta dello strato di persistenza.
- Possibilità di utilizzo con tutte le piattaforme JAVA, dalla semplice JVM ai container J2EE (le specifiche parlano di ambienti managed e non).

Quando si parla di trasparenza dello strato di persistenza dei JDO, l’obiettivo è quello di permettere al programmatore di concentrarsi sulla logica di business, senza dover manualmente aggiungere nessuna istruzione di persistenza nelle classi che modellano i business object. Questo codice può essere aggiunto alla classe con differenti modalità, che troviamo nelle specifiche e che possiamo riassumere con: prima della compilazione, dopo la compilazione, direttamente nel codice (in questo caso senza la trasparenza).

Finora quasi tutti i produttori utilizzano la seconda soluzione, con un tool chiamato enhancer, che modifica (anzi, sostituisce) la classe originaria con una capace di persistere (eccetto Xorm).

L’enhancer, ed il codice aggiunto alla classe sono diversi per ciascuna implementazione, ma tutti compatibili, dato che un requisito fissato dalle specifiche è la compatibilità binaria tra le classi “enhanced” dalle diverse implementazioni JDO.

Il programmatore si occupa di scrivere il codice di persistenza, per i JDO le specifiche propongono un enhancer.

Esempio pratico (Xorm): database contenete DVD

db.properties

```
javax.JDO.PersistenceManagerFactoryClass=
                                    org.xorm.InterfaceManagerFactory

javax.JDO.option.ConnectionUserName=postgres
javax.JDO.option.ConnectionPassword=postgres

javax.JDO.option.ConnectionURL=
              jdbc:postgresql://localhost:5432/miodb

javax.JDO.option.ConnectionDriverName=org.postgresql.Driver
javax.JDO.MinPool=2
javax.JDO.MaxPool=5
```

jdo.xml generato da Xorm

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE database SYSTEM "database.dtd">
<database>
    <table name="dvd">
            <column name="titolo" type="varchar" />
            <column name="prezzo" type="integer" />
            <column name="id" type="integer" primary-
                        key="true" auto="true" non-null="true" />
    </table>
</database>
```

db.jdo generato da Xorm

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jdo SYSTEM "jdo.dtd">
<jdo>
    <package name=“org.casamia.catalogo.jdo.db">
        <class name="Dvd">
            <extension vendor-name="XORM" key="table"
                            value="dvd" />
                <field name="titolo">
                   <extension vendor-name="XORM" key="column"
                            value="titolo" />
                </field>
                <field name="prezzo">
                   <extension vendor-name="XORM" key="column"
                            value="prezzo" />
                </field>
         </class>
    </package>
</jdo>
```

Interfaccia generata da Xorm

```
public interface Dvd {

    String getTitolo();
    void setTitolo(String val);

    int getPrezzo();
    void setPrezzo(int prezzo);

}
```

Value Object

```
public class DvdVO implements Dvd, Serializable {

   public DvdVO() {}

   public void setTitolo(String titolo) {
      _titolo = titolo;
   }

   public String getTitolo() {
      return _titolo;
   }

   public void setPrezzo(int prezzo) {
      _prezzo = prezzo;
   }

   public int getPrezzo() {
        return _prezzo;
  }
```

```
package org.casamia.catalogo.jdo;

import org.casamia.catalogo.jdo.db.*;
import java.io.*;
import java.util.*;
import javax.jdo.*;
import org.xorm.*;


public class CatalogoJdo {

    public CatalogoJdo() { }

    public static void main(String[] args) {

        CatalogoJdo catalogo = new CatalogoJdo();
        Collection lista = catalogo.getLista("Dvd", "titolo");
        Iterator iter = lista.iterator();
        while (iter.hasNext()) {
            Dvd item = (Dvd) iter.next();
            System.out.println("prezzo dvd " + item.getPrezzo());
            System.out.println("titolo dvd " + item.getTitolo());
        }
    }
```

```
public Collection getLista(String media, String elementOrder) {

 Collection result = null;
 try {
     InputStream in =  this.getClass().getResourceAsStream
        ("db.properties");
     PersistenceManagerFactory factory =
        XORM.newPersistenceManagerFactory(in);

     PersistenceManager pm =
        factory.getPersistenceManager();
     _extent = pm.getExtent(Dvd.class, true);
     Query query = pm.newQuery(extent);
     query.setOrdering(elementOrder+".ascending");

     result = (Collection) query.execute();

     } catch (Exception ex) {
         …….
```

JDOQL

- **Java programming language syntax**
- JDOQL is data-store independent
- JDO removes SQL dependencies for querying persistent data
- Efficient architecture (no instantiation)
- Query instances created by PersistenceManager
- Application must provide
  - Candidate collection (extent, previous query)
  - Class of result
  - Variable declaration, parameter declarations, filter expression, import declarations, ordering declarations

JDOQL (1)

```
Extent dvds = pm.getExtent(DVD.class, true);

String filter = “prezzo > 30”;

Query q = pm.newQuery(elementOrder, filter);

q.setOrdering(“descending”);

Collection results = (Collection) q.execute();
```

JDOQL (2)

```
String params = “float min, float max”;

 String filter = ”dvd.prezzo > min”
  + “ && dvd.prezzo <= max";

 Query q = pm.newQuery (Employee.class, filter);

 q.declareParameters (params);

 Collection results = (Collection) q.execute
   (new Float (20.5F), new Float (39.30F));
```

salvataggio

```
PersistenceManagerFactory factory =

XORM.newPersistenceManagerFactory(in);

PersistenceManager pm =
factory.getPersistenceManager();

DVD matrix = new DVD(“Matrix revolution”, “27”);

pm.currentTransaction().begin();
pm.makePersistent (emp);
pm.currentTransaction ().commit ();
pm.close ();
pmf.close ();
```

Feature	Serialization	JDBC	JDO
Data Model	Java	Relational table model	Java
Support of Java Classes	Yes	No	Yes
Access granularity	Object Graph	Table cell	Object / Fetch Group
Support of inheritance and polymorphism	Yes	No	Yes
Support of references and collections	Yes	No	Yes
Automatic management of cache	No	No	Yes
Transactions	No	Yes	Yes
Concurrency	No	Yes	Yes
Query Language	None	SQL, each vendor has a different dialect (not portable).	JDOQL, standard language, Java-like syntax
Object model supported in queries	No	No	Yes

Entity Beans

- ✘ Component-oriented
  - ✘ Inheritance
  - ✘ Polymorphism
  - ✘ Encapsulation
- ✘ Couples persistence to tx and security
- ✘ Slow and heavyweight
- ✘ No fine-grained object models
- ✘ EJB QL requires deploy-time definition of queries.

JDO

- ✓ JDO is object-oriented
  - ✓ Inheritance
  - ✓ Polymorphism
  - ✓ Encapsulation
- ✓ Easy to test, develop and deploy
- ✓ Highly flexible
- ✓ Java-centric

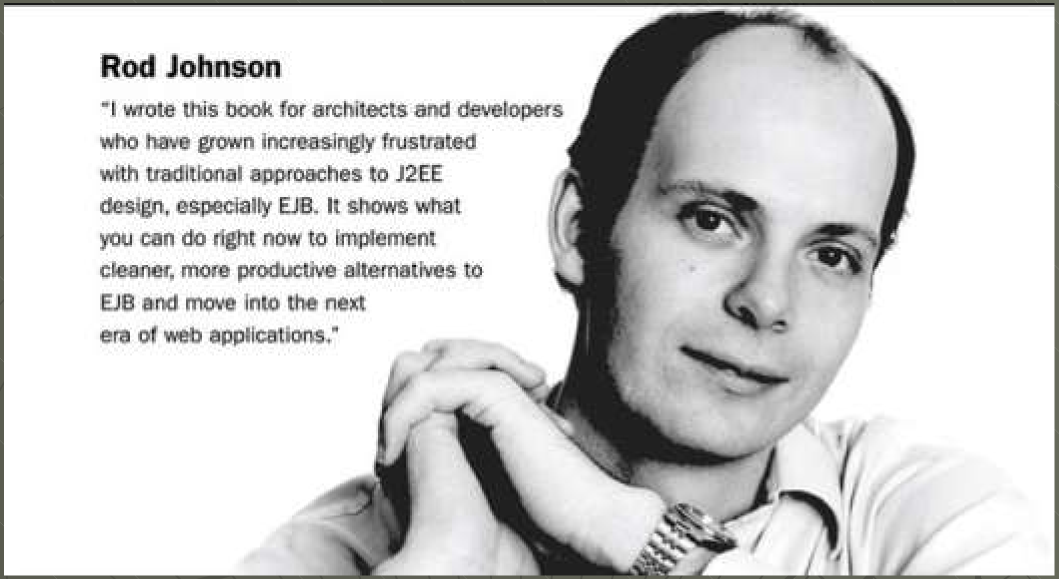
2002-2004 :

- JDO 1.0.1

2004-2005 :

JDO2.0

- Aggregate and Projection Support
- Detach / Attach
- O/R Mapping Meta-Data


Rod Johnson

"I wrote this book for architects and developers who have grown increasingly frustrated with traditional approaches to J2EE design, especially EJB. It shows what you can do right now to implement cleaner, more productive alternatives to EJB and move into the next era of web applications."

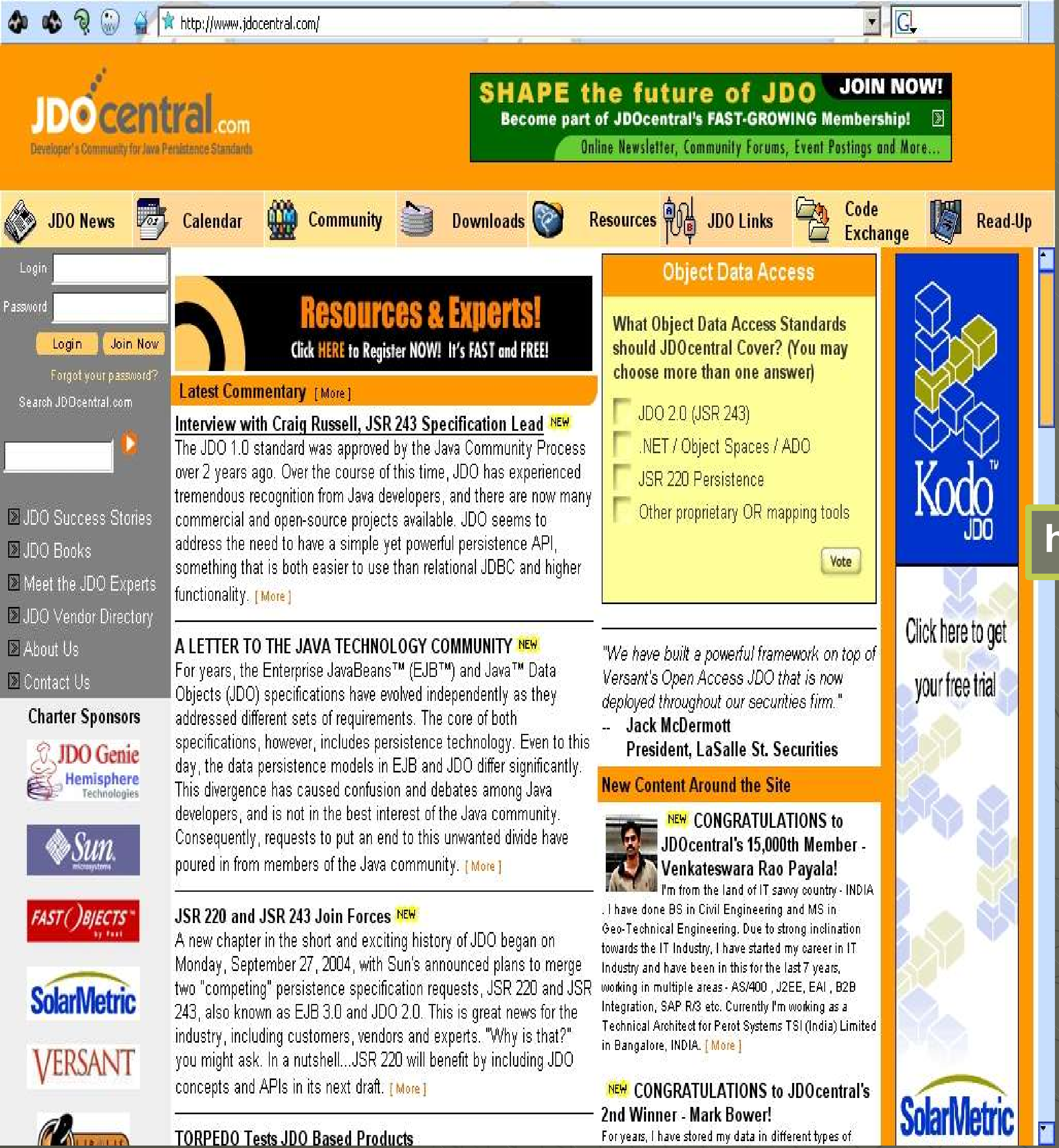
JDO/R 2.0

Fine 2003 JDO 2.0 Expert Group nuovi membri:

Rod Johnson (Spring)

Gavin King (Hibernate)

Hibernate implementerà le specifiche JDO/R 2.0 in alternativa a quelle già presenti.


http://www.jdocentral.com/

JDOcentral.com
Developer's Community for Java Persistence Standards

SHAPE the future of JDO JOIN NOW!
Become part of JDOcentral's FAST-GROWING Membership!
Online Newsletter, Community Forums, Event Postings and More...

JDO News | Calendar | Community | Downloads | Resources | JDO Links | Code Exchange | Read-Up

Login
Password
Login | Join Now
Forgot your password?
Search JDOcentral.com

JDO Success Stories
JDO Books
Meet the JDO Experts
JDO Vendor Directory
About Us
Contact Us

Charter Sponsors
JDO Genie Hemisphere Technologies
Sun microsystems
FAST OBJECTS by Poet
SolarMetric
VERSANT

Resources & Experts!
Click HERE to Register NOW! It's FAST and FREE!

Latest Commentary [More]

Interview with Craig Russell, JSR 243 Specification Lead NEW
The JDO 1.0 standard was approved by the Java Community Process over 2 years ago. Over the course of this time, JDO has experienced tremendous recognition from Java developers, and there are now many commercial and open-source projects available. JDO seems to address the need to have a simple yet powerful persistence API, something that is both easier to use than relational JDBC and higher functionality. [More]

A LETTER TO THE JAVA TECHNOLOGY COMMUNITY NEW
For years, the Enterprise JavaBeans™ (EJB™) and Java™ Data Objects (JDO) specifications have evolved independently as they addressed different sets of requirements. The core of both specifications, however, includes persistence technology. Even to this day, the data persistence models in EJB and JDO differ significantly. This divergence has caused confusion and debates among Java developers, and is not in the best interest of the Java community. Consequently, requests to put an end to this unwanted divide have poured in from members of the Java community. [More]

JSR 220 and JSR 243 Join Forces NEW
A new chapter in the short and exciting history of JDO began on Monday, September 27, 2004, with Sun's announced plans to merge two "competing" persistence specification requests, JSR 220 and JSR 243, also known as EJB 3.0 and JDO 2.0. This is great news for the industry, including customers, vendors and experts. "Why is that?" you might ask. In a nutshell...JSR 220 will benefit by including JDO concepts and APIs in its next draft. [More]

TORPEDO Tests JDO Based Products

Object Data Access
What Object Data Access Standards should JDOcentral Cover? (You may choose more than one answer)
- [ ] JDO 2.0 (JSR 243)
- [ ] .NET / Object Spaces / ADO
- [ ] JSR 220 Persistence
- [ ] Other proprietary OR mapping tools

Vote

"We have built a powerful framework on top of Versant's Open Access JDO that is now deployed throughout our securities firm."
-- Jack McDermott
President, LaSalle St. Securities

New Content Around the Site

NEW CONGRATULATIONS to JDOcentral's 15,000th Member - Venkateswara Rao Payala!
I'm from the land of IT savvy country - INDIA. I have done BS in Civil Engineering and MS in Geo-Technical Engineering. Due to strong inclination towards the IT Industry, I have started my career in IT Industry and have been in this for the last 7 years, working in multiple areas - AS/400, J2EE, EAI, B2B Integration, SAP R/3 etc. Currently I'm working as a Technical Architect for Perot Systems TSI (India) Limited in Bangalore, INDIA. [More]

NEW CONGRATULATIONS to JDOcentral's 2nd Winner - Mark Bower!
For years, I have stored my data in different types of

Kodo™ JDO
Click here to get your free trial
SolarMetric

http://www.jdocentral.com

Remember: developing J2EE applications should be fun.