Use of Standardised APIs with db4o

Andy Jefferson

Java Persistent Objects

Persistence Specifications

- JDO (1.0, 2.0, 2.1) present since 2002
- JPA (1.0) present since 2006
- JDO designed for all datastores
- JPA designed for *"mass-market"* (RDBMS)
- All specifications have 3 parts
  - **Persistence Definition** *"what to persist"*
  - **Persistence API** *"how to persist it"*
  - **Query Language(s)** *"how to retrieve it"*

Object Identity

- **Datastore Identity** : surrogate identity added to the object in the datastore
- **Application Identity** : identity formed by field(s)/property(s) of the class.
- JDO supports both
- JPA supports *application-identity* only
- db4o supports *datastore-identity* only

Persistence Definition : JDO

```
public class Person
{
    String firstName;
    String lastName;
    int age;
    Set<Person> friends =
        new HashSet<Person>();

    ...
}
```

Using Annotations

```
@PersistenceCapable(detachable="true")
@Version(strategy=VERSION_NUMBER)
public class Person
{
    String firstName;
    String lastName;
    @Index
    int age;
    @Element(dependent="true")
    Set<Person> friends =
        new HashSet<Person>();

    ...
```

Using XML

```
<class name="Person" detachable="true">
    <field name="age" indexed="true"/>
    <field name="friends">
        <collection element-type="Person"
            dependent-element="true"/>
    </field>
</class>
```

JPOX
Java Persistent Objects

Persistence Definition : JDO

- Classes accessing fields of persistable classes should be made *PersistenceAware*

```
@PersistenceAware
public class PayrollManager
{
    public void calculatePay(Employee emp)
    {
        emp.monthlyIncome = 3000;
    }
    ...
}
```

Persistence Definition : JPA

```
public class Person
{
    String firstName;
    String lastName;
    int age;
    Set<Person> friends =
        new HashSet<Person>();

    ...
}
```

Using Annotations

```
@Entity
public class Person
{
    String firstName;
    String lastName;
    int age;

    @OneToMany(cascade=CascadeType.ALL)
    Set<Person> friends =
        new HashSet<Person>();

    ...
}
```

Using XML

```
<entity class="Person">
    <attributes>
        <id name="firstName"/>
        <id name="lastName"/>
        <one-to-many name=”friends">
            <cascade>
                <cascade-all/>
            </cascade>
        </one-to-many>
    </attributes>
</entity>
```

JPOX
Java Persistent Objects

Transparent Persistence

- All classes **to be persisted** implement *PersistenceCapable*
- All classes **accessing fields** of persistent classes implement *PersistenceAware*
- Byte-code enhancer updates the classes as required (Maven1, Maven2, Ant, Eclipse) using persistence definition.
- db4o recent versions require similar with *Activator/Activatable*

Persistence API : JDO

1. Define your persistence properties

jpox.properties

```
javax.jdo.PersistenceManagerFactoryClass=
    org.jpox.jdo.JDOPersistenceManagerFactory
javax.jdo.option.ConnectionURL=db4o:file:db4o.db

org.jpox.db4o.outputFile=db4o.output
```

2. Create a PersistenceManager

```
PersistenceManagerFactory pmf =
    JDOHelper.getPersistenceManagerFactory("jpox.properties");
PersistenceManager pm = pmf.getPersistenceManager();
```

JPOX
Java Persistent Objects

Persistence API : JDO (2.1)

1. Define your persistence properties

persistence.xml

```
<persistence ...>
    <persistence-unit name="myUnit">
        <class>org.jpox.Person</class>
        <properties>
            <property name="javax.jdo.option.ConnectionURL"
                      value="db4o:file:db4o.db"/>
        </properties>
    </persistence-unit>
</persistence>
```

2. Create a PersistenceManager

```
PersistenceManagerFactory pmf =
    JDOHelper.getPersistenceManagerFactory("myUnit");
PersistenceManager pm = pmf.getPersistenceManager();
```

JPOX
Java Persistent Objects

Persistence API : JPA

1. Define your persistence properties

persistence.xml

```
<persistence ...>
    <persistence-unit name="myUnit">
        <provider>org.jpox.jpa.PersistenceProviderImpl</provider>
        <class>org.jpox.Person</class>
        <properties>
            <property name="javax.jdo.option.ConnectionURL"
                      value="db4o:file:db4o.db"/>
        </properties>
    </persistence-unit>
</persistence>
```

2. Create an EntityManager

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("myUnit", props);
EntityManager em = emf.createEntityManager();
```

JPOX
Java Persistent Objects

Persistence API

	JDO	JPA	db4o
Persist Object	pm.makePersistent	em.persist	oc.set
Attach Object	pm.makePersistent	em.merge	oc.set
Delete Object	pm.deletePersistent	em.remove	oc.delete
RetrieveObject	pm.getObjectById, pm.getExtent	em.find	oc.get
Refresh Object	pm.refresh	em.refresh	
Flush Changes	pm.flush	em.flush	
Transaction	pm.currentTransaction	em.getTransaction	oc
New Query	pm.newQuery	em.createQuery	oc.query

Persistence API : Cascading

- **JDO**
  - **persist/update** operations are cascaded
  - **delete** operations default to **not** cascade
  - **refresh** operations do **not** cascade
  - **fetching** is governed by a FetchPlan
- **JPA**
  - **persist/update/delete/refresh** operations default to **not** cascade
- db4o has own cascading rules

Transactions

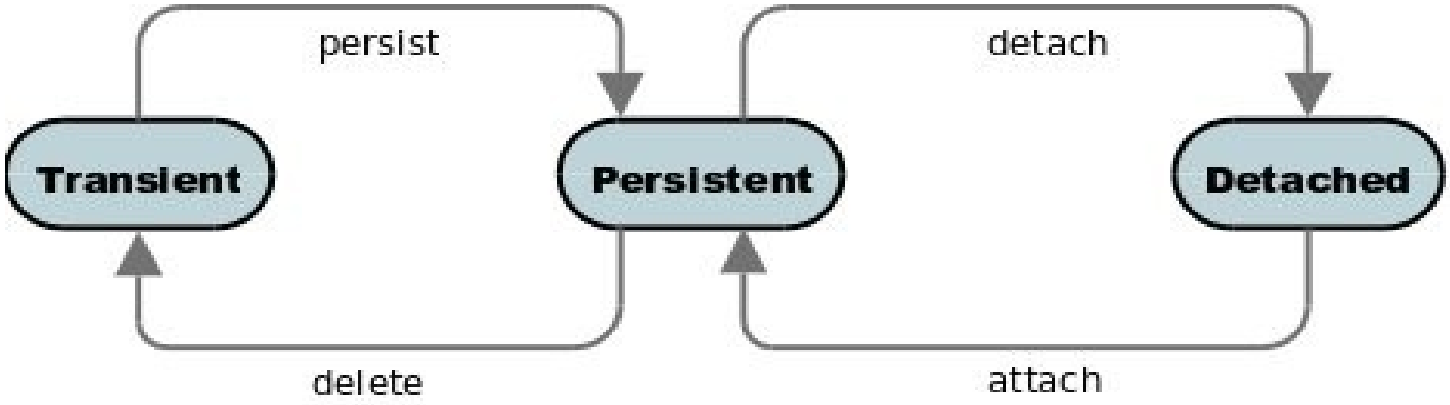
- JDO - Pessimistic/Optimistic/Non-Tx
- JPA – Optimistic/Non-Tx
- db4o – Left to application
- JDO/JPA - flush() for persistence control

```
Transaction tx = pm.currentTransaction();
try
{
    tx.begin();

    [persistence operations]

    tx.commit();
}
finally
{
    if (tx.isActive()) {tx.rollback();}
}
```

Transparent Persistence

- Persist of object *enlists* it in the transaction. All updates persisted with no user calls.
- Retrieve of object *enlists* object in transaction. Updates persisted with no user calls.
- Access of relation field will *activate* the related object
- All managed objects have a *StateManager*

Object Lifecycle



- JDO/JPA have similar lifeycles
- JDO has additional “hollow”, “dirty”, “non-tx” states

Queries : JDOQL

Syntax

```
SELECT [UNIQUE] [<result>] [INTO <result-class>]
       [FROM <candidate-class> [EXCLUDE SUBCLASSES]]
       [WHERE <filter>]
       [VARIABLES <variable declarations>]
       [PARAMETERS <parameter declarations>]
       [<import declarations>]
       [GROUP BY <grouping>]
       [ORDER BY <ordering>]
       [RANGE <start>, <end>]
```

Example of variables, methods

```
SELECT FROM org.jpox.Inventory WHERE products.containsKey(productName) &&
    (productName == "product 1" || productName == "product 2")
    VARIABLES String productName
```

Example of parameters, aggregates

```
SELECT max(price), min(price) FROM org.jpox.Product
WHERE manufacturer == :myParam
```

Example of subqueries

```
SELECT FROM org.jpox.Employee
WHERE this.salary > (SELECT avg(salary) FROM org.jpox.Employee e)
```

Queries : JPQL

Syntax

```
SELECT [<result>]
    [FROM <candidate-class(es)>]
    [WHERE <filter>]
    [GROUP BY <grouping>]
    [HAVING <having>]
    [ORDER BY <ordering>]
```

Example of joins and comparisons

```
SELECT Object(F) FROM org.jpox.Farm F INNER JOIN F.animals A
WHERE F.acreage > 750 AND A.type = "Pig"
```

Example of parameters and keywords

```
SELECT Object(O) FROM org.jpox.Order O
WHERE O.date < CURRENT_DATE AND O.type = :productType
```

Example of subqueries

```
SELECT Object(P) FROM org.jpox.Person P
WHERE P.age > (SELECT avg(Q.age) FROM org.jpox.Person Q)
```

Caching (JPOX)

- Used to minimise datastore lookups of already retrieved objects
- **Level1** : cache of objects per PM/EM.
- **Level2** : cache of objects per PMF/EMF.
- **Level2** can use external products such as Tangosol, EHCache

Example : JDO Persistence

- Persistence

```
PersistenceManagerFactory pmf =
    JDOHelper.getPersistenceManagerFactory("jpox.properties");
PersistenceManager pm = pmf.getPersistenceManager();
pm.setDetachAllOnCommit(true);
pm.setCopyOnAttach(false);
Transaction tx = pm.currentTransaction();

Person p1 = new Person("John", "Smith"); // p1 now "transient"
try
{
    tx.begin();
    pm.makePersistent(p1); // p1 now "persistent"
    tx.commit(); // p1 now "detached"
}
finally
{
    if (tx.isActive()) { tx.rollback(); }
}
```

Example : JDO Persistence

- Attachment

```
p1.setAge(23); // p1 now "detached-dirty"
try
{
    tx.begin();
    pm.makePersistent(p1); // Attaches the change, p1 "persistent"
    tx.commit();
}
finally
{
    if (tx.isActive()) { tx.rollback(); }
}
```

Example : JDO Persistence

- Query + Update

```
try
{
    tx.begin();
    Query q = pm.newQuery("SELECT FROM org.jpox.Person WHERE
lastName == 'Smith'");
    List results = (List)q.execute();
    Person p = (Person)results.get(0); // p now "persistent"

    p.setAge(24); // Changes transparently persisted
    tx.commit();
}
finally
{
    if (tx.isActive()) { tx.rollback(); }
}
```

JPOX Log

```
DB4O SODA : query.constrain(org.jpox.Person)
DB4O SODA : query.descend(lastName).constrain(Smith)
```

JPOX
Java Persistent Objects

Example : JDO Persistence

- Native Query

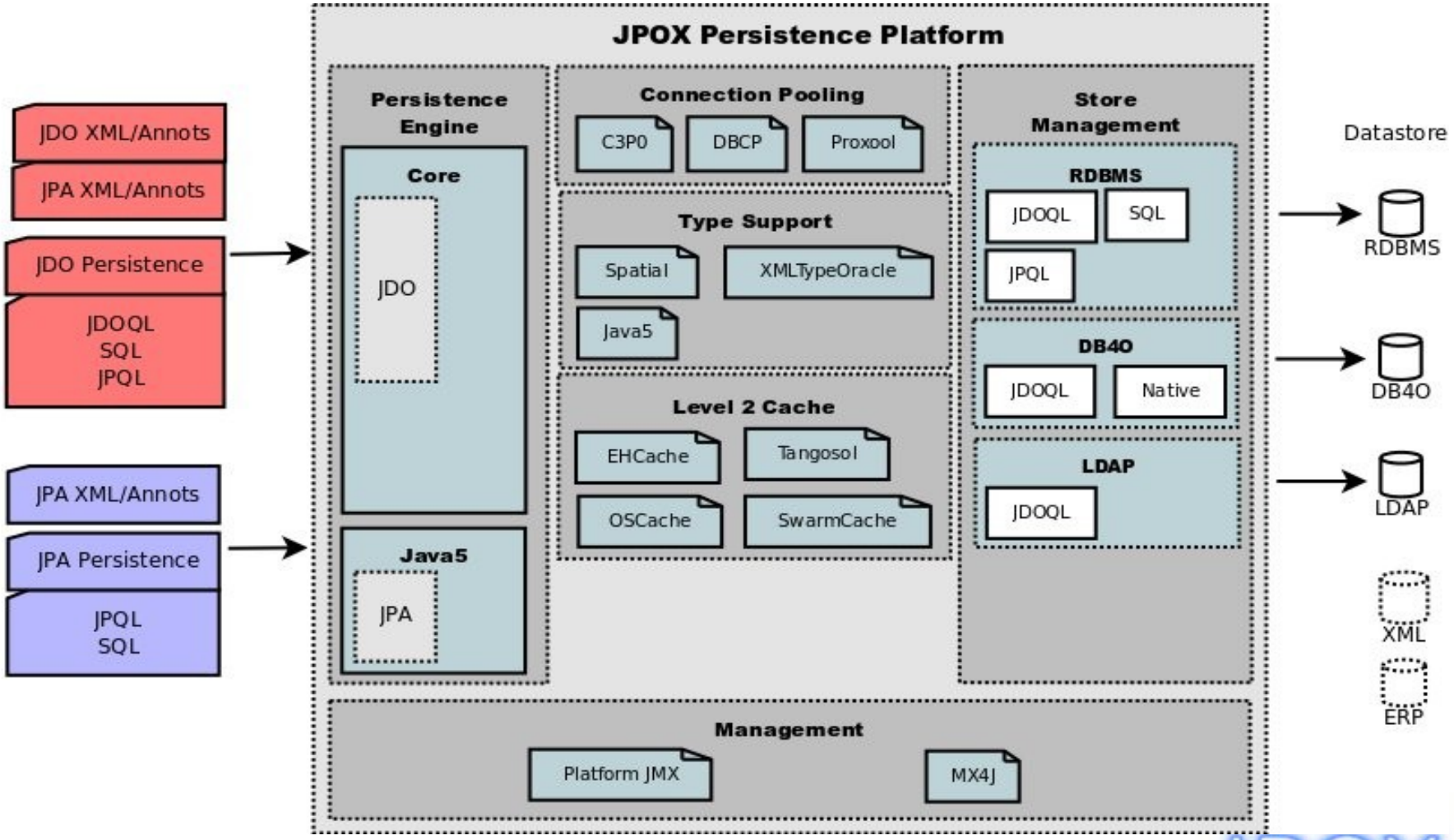
```
try
{
    tx.begin();
    Query q = pm.newQuery("Native", new Predicate()
    {
        public boolean match(Person p)
        {
            return p.getAge() >= 32;
        }
    });

    List results = (List)q.execute();
    tx.commit();
}
finally
{
    if (tx.isActive()) { tx.rollback(); }
}
```

Why use standardised APIs?

- Many JDO/JPA implementations so swapping to other datastores is trivial – allows rapid prototyping using DB4O
- Tooling support e.g Eclipse Dali
- Transparent persistence/activation
- No reliance on vendor classes
- Benefit from other JPOX features
  - *(current)* L2 caching, value generation, app id
  - *(future)* data federation, SDO

JPOX
Java Persistent Objects

JPOX : Architecture



JPOX
Java Persistent Objects

JPOX : Architecture

- Plugin Registry : capabilities discovered at runtime
- JPOX defines *"extension-points"* and plugins provide *"extensions"* (implementations of the extension-points)
- All plugin jars are OSGi bundles (MANIFEST.MF)
- Majority of components of JPOX are *extension-points*

JPOX-db4o : Current

- File-based : ***db4o:file:{my_db4o_file}***
- Server-based : ***db4o:{db4o_host}:{db4o_port}***
- JDO/JPA persistence
- JDOQL : all basic capabilities
- Native db4o queries with JDO

JPOX-db4o : Future

- Complete support for JDOQL
- Direct support for SODA queries
- Support for LINQ
- Support for JPQL
- Support for SQL via *db4o-sql* project

Further Reading

- JPOX : http://www.jpox.org
- Apache JDO : http://db.apache.org/jdo

JPOX
Java Persistent Objects