The Future of Object Persistence

St Louis JUG

13 January 2005

Patrick Linskey
pcl@solarmetric.com
About Me

Patrick Linskey

- CTO at SolarMetric
- Involved in object/relational mapping and EJB since 1999
- Frequent JDO presenter at JUGs, Java conventions and seminars
- Member, JDO Expert Group
- Member, EJB Expert Group
- Luminary, JDOcentral

- Co-author of *Bitter EJB* along with Bruce Tate, Mike Clark, and Bob Lee
Corporate Profile

- International Company Based in Austin, TX
  - Offices in London, California, Massachusetts
  - Through partners, reach is worldwide
- Founded in 2001 by MIT alums.
  - Core team has been together since 1997.
- Frustrated with trying to do Java object persistence with:
  - Proprietary Tools – Vendor Lock-in
  - Entity beans – difficult, slow, impose undesirable constraints on object model and development patterns
- Committed to Technical Quality and Innovation, Customer Support
- Leading JDO implementation
- Client base is diverse, both in terms of industry and size
  - 300+ customers
- Regular JDO Training courses throughout the world
SolarMetric’s Role

JSR 220 (EJB 3)
- Actively contributing member
- Actively developing preview version of the EJB 3 specification

JSR 243 (JDO 2)
- Will continue active role on the development and release of JDO 2
- Will continue active development of Kodo’s JDO bindings, for JDO 2 and for future JDO versions
Object / Relational Mapping
Object-oriented programming languages are vastly different than relational data languages.

Object / relational “impedance mismatch” has plagued enterprise programmers for years.
- Mapping
- Remoteness of data

Proprietary O/R mapping products exist for a variety of languages: Smalltalk, C++, Java, others.

JDO and EJB3
- designed for object/relational mapping and
- designed to work inside and outside a container
Why not just JDBC?

- JDBC misses the “O” part of O/R mapping
  - Interface is not at an object level but rather at SQL (row and column) level
  - Not Java
  - Creates complexity especially when leveraging OO concepts e.g., inheritance, polymorphism

- JDBC is a low level API
  - Used as a building block by most O/R mapping tools

- Sadly, SQL is not portable
  - Many different dialects
Goals of an O/R specification

- Abstract API for object persistence
- No need to write persistence infrastructure
- Standard means of persisting objects. Low risk of vendor lock-in
- Portability between data stores
Requirements of an O/R Specification

- Persist objects whether simple or complex mapping is required
- Query those persisted objects
- Minimize visibility of O/R mapping APIs
- Connection management
- Transaction management
- Allow object model and data model to be optimized independently
Mapping Objects to RDBMS

- Many ways to “map” (describe the relationship) between an object model and a schema
  - Map directly to a column
  - Relationships between objects can be mimicked with foreign key relationships in schema
  - Collections of objects can be:
    - One-to-many
    - Many-to-many with a join table

- Schema and classes are not tied together

Most good O/R Solutions Have Tools To Help with Mappings
Benefits of these specs

There are already a number of persistence solutions out there. What does JDO and EJB3 UPS bring to the table?

- Defer decisions
- Focus on core competencies
- Use Java to its fullest
- Project maintainability
Defer data store decisions

- Choice of persistence technology should not require any other architecture decisions

- Entity beans prior to EJB3: require the use of a container

- JDBC and proprietary object-relational mapping: require a relational database
  - often, results in dependencies to a particular SQL variant

- Changes to schema would cause pain and large amounts of code rewriting

- JDO and EJB3: seamlessly work with J2EE, but do not require it. Can be used with all sorts of data stores (relational, legacy EISs, J2ME devices, ...)
Focus on core competencies

- 20% to 40% of typical JDBC projects is persistence infrastructure
- JDBC code is repetitive and difficult, and therefore prone to errors
Focus on core competencies

- O/R mapper’s infrastructure is typically closer to 5%
- No boilerplate code, so less room for cut-and-paste errors
Use Java to its fullest

O/R mapping is designed to integrate seamlessly with Java

- Polymorphism, both in queries and relations
  - Show me all the Vehicles in the warehouse
- Data encapsulation supported

Developers need not re-learn new rules and limitations. Regular Java concepts behave exactly as expected.
Maintenance over time

- Most interaction happens through object model
- Direct use of APIs is minimal, so is readily understandable
  - query for objects
  - transaction demarcation (when used outside a JTA environment)
  - adding objects to database
Why O/R mapping?

To sum up, an O/R mapping framework enables you to:

- Make architecture and data store decisions as needed
- Do more “real work” in a given amount of time
- Do the same amount of “real work” in less time
- Write regular Java, without learning all sorts of restrictions
- More easily maintain your code base through its lifecycle
JDO 2 Status / Schedule
JDO 2 is an Evolution from JDO 1, Not a Revolution

- Maintain Java Data Objects 1.0 Compatibility
- Standardize Mapping to Relational DB
  - SQL as a supported query language
- Multi-tier Development Improvements
  - Disconnected Object Graphs
- Usability Improvements
- Better Object Modelling
- Richer Queries
  - Single-line format
  - Projections
  - Aggregates
  - Paging Query Results
- More Vendor Support
  - Remove PersistenceCapable requirement
JDO 2.0 schedule

- Public Draft currently available
- TCK, RI being developed in the Apache project
- JDO2 jars should be available in the next few weeks
- Final JDO2 specification (including TCK and RI) in four to eight months
Unnamed Persistence Specification (UPS) Status

Enterprise Java Beans 3
The Announcement


Historical background
- Core of EJB and JDO specs have included persistence
- Data persistence models have differed
- Sun is leading a community effort to have a single POJO persistence model
The Announcement


Summary of announcement

- Added 6 JDO members to JSR-220 (EJB3 Spec Team)
- All current JDO experts are listeners on JSR-220 list
- Work will start under JSR-220 but independent of EJB3
- Goal: Single O/R Mapping Framework for J2EE, J2SE
- Timeframe: J2EE 5.0 (January 2006 currently)
Unnamed Persistence Specification (UPS) is part of EJB3

Work in progress on a second Early Release Draft

Heavy utilization of annotations

Lots of mis-information out there
  - No existing product is UPS
Criteria to Consider When Choosing a Persistence Solution
Looking for a Persistence Solution? (1 of 3)

Criteria to Consider

- Standards based vs. Proprietary
  - Portability – avoid vendor lock-in
  - Specification created by experienced community members vs. 1 individual or 1 company with other goals

- Which standard?
  - JDO (27+ implementations)
  - EJB3
  - Both

- Total Cost of Ownership
  - Upfront Costs vs. Runtime Costs
  - Support Costs
  - Maintenance Costs
  - Training Costs
Looking for a Persistence Solution? (2 of 3)

Criteria to Consider

- **Ease of Use**
  - Tooling
- **Ability to Optimize Scalability & Performance Trade-Offs**
- **Supported Mappings**
  - Custom Mappings Available?
- **Extensibility**
- **Datastore Flexibility**
  - Relational
  - Non-relational (legacy, hierarchical, object, etc.)
- **Architecture flexibility**
  - In container – session beans, CMP, BMP
  - Outside container – servlets, Spring, JSP, client-server
Looking for a Persistence Solution? (3 of 3)

Criteria to Consider

- Performance and Scalability Issues
  - Lazy Loading
  - Dirty Field Tracking
  - Caching (pluggable caching with variety of invalidation strategies)
  - Minimize Round-Tripping

- Legacy Database Support
  - Stored Procedure Support
  - SQL Support

- Supporting Organization
  - Tied to another product?
  - Organization’s thought leadership
  - Support response times

- Legal Issues
  - Indemnification
Kodo Product Suite
Kodo Roadmap

- Continue to build an underlying engine that promotes:
  - Performance
  - Scalability
  - Flexibility
  - Ease of Use

- Will offer multiple bindings to the underlying Kodo engine
  - Kodo JDO – JDO bindings
  - Kodo “UPS” – “UPS” (Unnamed Persistence Spec) bindings

- Interoperability between Kodo JDO and Kodo “UPS”
  - Will be possible to simultaneously use both APIs with same domain model
Current Situation – Kodo JDO

JDO

Kodo Persistence Kernel
Future Situation – Kodo JDO and Kodo “UPS”

Kodo Persistence Kernel
Kodo JDO 3.2

- Significant support for JDO 2 including:
  - JDO 2 Queries (single string JDOQL queries, implicit parameters / variables, named query support, subqueries)
  - More flexibility with detachment APIs, including automatic detachment on PM close
- Improvements to Kodo Development Workbench, Management Console, and Profiler
- Reverse Mapping Tool is accessible via a guided wizard
- Improved eager fetching
- Support for managed inverses
- Intersystems Caché database support
- Tools – Workbench and Management Console
- High-performance database cache
- Prepared statements, statement caching, statement batching
- Intelligent handling of large result sets
- SQL as query language
- flat, vertical, horizontal inheritance mappings
- Extensible architecture
- Reverse and forward engineering of database schema
- Schema evolution
Management Console (JMX)
Supported Databases

Relational:
- Oracle 8i/9i/10
- Microsoft SQL Server
- IBM DB2
- Sybase Adaptive Server Enterprise
- Informix IDS
- MySQL
- PostgreSQL
- Firebird
- Interbase
- Hypersonic SQL
- JDataStore
- FoxPro
- MS Access
- Pointbase
- Cloudscape
- InstantDB
- Empress
- Extensible database support

Non-Relational:
- Legacy EIS (CICS, Unisys, others) (ask for details)
- LDAP (coming soon)
- BerkeleyDB (coming soon)
- InterSystems Caché
Kodo JDO IDE Support

- All IDEs with Ant Support
- Tight integration with:
  - Borland JBuilder 7 or newer
  - Sun One Studio / NetBeans
  - Eclipse / WSAD / WSED
- Standalone Development Workbench

Kodo JDO Supported App Servers

- WebLogic 6.2 – 8.1
- WebSphere 5
- JBoss 3.0 – 3.2
- SunONE
- Trifork
- JRun 4
- Borland Enterprise Server
More Info

- Web site: http://www.solarmetric.com
- On-site and off-site Kodo JDO training available
- Professional Services available
- Email contact: info@solarmetric.com
- Speaker
  - Patrick Linskey: pcl@solarmetric.com

- Advanced Webinar – December 9 – 10:00 AM Pacific
  - Covers Performance and Scalability
  - Using JDO with J2EE
  - Other Advanced Topics
  - Register at http://www.solarmetric.com
Backup Slides
What’s New In JDO 2?
Goals for Java Data Objects 2.0

- Maintain JDO 1.0 Compatibility
- Standardize Mapping to Relational DB
  - SQL as a supported query language
- Multi-tier Development Improvements
  - Disconnected Object Graphs
- Usability Improvements
- Better Object Modelling
- Richer Queries
  - Single-line format
  - Projections
  - Aggregates
  - Paging Query Results
- More Vendor Support
  - Remove PersistenceCapable requirement
Standardize Mapping to RDBMS

- **Simple Mapping**
  - Class ↔ Table
  - Field ↔ Column
  - 1-1 relationship ↔ unique foreign key
  - 1-many relationship ↔ foreign key
  - many-many relationship ↔ join table

- **Complex Mapping**
  - Inheritance strategies
  - Multiple tables per class
  - List, Map, Embedded
Multi-tier Development

-Disconnected Object Graphs
  - Disconnect objects from PersistenceManager
  - Close PersistenceManager
  - Modify disconnected objects
    - send to different tier, or
    - apply changes directly to objects
  - Begin new transaction
  - Reconnect objects to PersistenceManager
  - Commit transaction (applies changes)

-Disconnected Objects keep original identity and version
How do you find a PersistenceManagerFactory?

Java Data Objects 1.0 Properties:

```java
javax.jdo.PersistenceManagerFactoryClass:
    com.sun.jdori.fostore.FOStorePMF
javax.jdo.option.ConnectionURL:
    fostore:/shared/databases/jdo/dbdir
javax.jdo.option.ConnectionUserName:craig
javax.jdo.option.ConnectionPassword:faster
javax.jdo.option.Optimistic:true
javax.jdo.option.RetainValues:true
javax.jdo.option.NontransactionalRead:true
```
Bootstrap

- JDO 1.0 Properties instance:
  - `getPersistenceManagerFactory (Properties props)`

- JDO 2.0 Properties loaded from:
  - stream
    - `getPersistenceManagerFactory (InputStream props)`
  - File (use to get a stream)
    - `getPersistenceManagerFactory (File propsFile)`
  - resource name (use to get a stream)
    - `getPersistenceManagerFactory (String propsResource)`

- JDO 2.0 JNDI lookup
  - `getPersistenceManagerFactory (String name, Context c)`
Usability

- JDO 2.0 Transaction adds:
  - boolean getRollbackOnly();
  - void setRollbackOnly();
- JDO 2.0 Query can be defined in metadata and accessed by name
Usability

- ThreadLocal JDOHelper methods
  - getThreadLocalPersistenceManager
  - setThreadLocalPersistenceManager
  - getThreadLocalPersistenceManagerFactory
  - SetThreadLocalPersistenceManagerFactory

- Allows business delegates to be independent of their environment
  - Web server
  - App server
  - Two tier
Better Object Modeling

- **Persistent Abstract Class Support**
  - map abstract classes to tables
  - map properties, fields to columns
  - create new implementation instances

- **Persistent Interface Support**
  - map interfaces to tables
  - map properties to columns
  - create new implementation instances
  - use interfaces in queries
Richer Queries

- Projections
- Aggregates
- More String expressions
- More numeric, Map functions
- Paging query results
- Native SQL
- User-defined Result Class

Preserve Expressability in SQL
Projections

- JDO 1.0: query results are a subset of instances in the candidate collection.
- JDO 2.0: query results can be projections of:
  - candidate collection instances;
  - variable instances;
  - fields;
  - parameters;
  - combinations of the above.
Projections: Example

Query q = pm.newQuery (Employee.class,
           "dept.name.startsWith(deptName)");
q.declareParameters ("String deptName");
q.setResult("name, salary, boss");
Collection names = (Collection) q.execute("R&D");

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bill Jones</td>
<td>12546</td>
<td>Employee@18d8788</td>
</tr>
<tr>
<td>Sam Adams</td>
<td>15948</td>
<td>Employee@18d8788</td>
</tr>
<tr>
<td>Will Clinton</td>
<td>50453</td>
<td>Employee@1867480</td>
</tr>
<tr>
<td>Westlake Clerk</td>
<td>18737</td>
<td>Employee@18720</td>
</tr>
<tr>
<td>Harvey Dean</td>
<td>14657</td>
<td>Employee@18720</td>
</tr>
<tr>
<td>Ed Muskrat</td>
<td>13009</td>
<td>Employee@18720</td>
</tr>
</tbody>
</table>
Aggregates

- Query Results can be aggregates
- Aggregates have standard semantics
  - min
  - max
  - avg
  - sum
  - count
- Group by projected fields
- “Having” allows conditional filtering
Aggregates: Example

Query q = pm.newQuery (Employee.class,
  "dept.name.startsWith(deptName)");
q.declareParameters ("String deptName");
q.setResult("dept.name, min(salary), max(salary)");
q.setGrouping("dept.name having count(dept.name) > 1");
Collection names = (Collection) q.execute("R&D");
for (Object[] i : (Collection<Object[]>) results) {
    println(i[0], i[1], i[2]);
}

R&D Santa Clara 12546 15948
R&D Burlington 13009 18737
String Expressions

- toLowerCase(), toUpperCase()
- indexOf(String), indexOf(String, int)
- matches(String pattern)
  - pattern is a subset of regular expressions:
    - (?i) global case-insensitive
    - . match any one character
    - .* match any number (0 to n) of characters
    - pattern is literal or parameter only
- substring(int), substring(int, int)
Other Query Filter Methods

- Math.abs(numeric expression)
- Math.sqrt(numeric expression)
- Map.containsKey(Object)
- Map.containsValue(Object)
Paging Query Results

- Improves performance for some applications
  - Skips already-returned results
  - Limits number of results
- Query.setRange (int fromIncl, int toExcl);
- default
  - fromIncl = 0
  - toExcl = Integer.MAX_VALUE
User-Defined Result Class

- JDO 1.0 query results are of type Collection
- JDO 2.0 allows user to specify result class:
  - Primitive wrapper (unique results)
  - Collection<Primitive wrapper>
  - Object[] (projected or aggregate unique results)
  - Collection<Object[]>
  - User-defined Class (unique results)
  - Collection<User-defined Class>
class Info {
    public String name;
    public Float salary;
    public Employee reportsTo;
}

class Employee {
    private String name;
    private float salary;
    private Department dept;
    private Employee boss;
}

Query q = pm.newQuery (Employee.class,
    "dept.name == deptName");
q.declareParameters ("String deptName");
q.setResult("name, salary, boss as reportsTo");
q.setResultClassName(Info.class);
Collection results = (Collection)
q.execute("R&D");
for (Info i : (Collection<Info>) results) {
    println(i.name, i.salary, i.reportsTo.name);
}
JDO Code Examples
package kodo.example;

import java.util.*;

public class Employee extends Person
{
    private float salary;
    private Company company;
    private Set projects = new HashSet();

    public Employee (String firstName,
            String lastName) {
        super (firstName, lastName);
    }

    public void giveRaise (float percent) {
        salary *= 1 + percent;
    }

    public Collection getProjects () {
        return projects;
    }
}
import javax.jdo.*;

public class MyPersistenceCode
{
    public static void main (String[] args)
    {
        // configure system
        PersistenceManagerFactory pmf =
            JDOHelper.getPersistenceManagerFactory (System.getProperties());
        PersistenceManager pm = pmf.getPersistenceManager();

        // business code
        Employee emp = new Employee("Marc", "Prud'hommeaux");
        emp.setCompany (new Company("SolarMetric, Inc."));
        emp.getProjects ().add (new Project("Kodo");
        emp.giveRaise (.10F);

        // persistence code
        pm.currentTransaction().begin();
        pm.makePersistent (emp);
        pm.currentTransaction ().commit ();
        pm.close ();
        pmf.close ();
    }
}
JDOQL Examples

- Basic Query:
  ```java
  String filter = "salary > 30000";
  Query q = pm.newQuery (Employee.class, filter);
  Collection emps = (Collection) q.execute ();
  ```

- Basic Query with Ordering:
  ```java
  String filter = "salary > 30000";
  Query q = pm.newQuery (Employee.class, filter);
  q.setOrdering ("salary ascending");
  Collection emps = (Collection) q.execute ();
  ```
JDOQL Examples

- Query with Relation Navigation and Parameters:

```java
String params = "float min, float max";
String filter = "company.revenue > min"
    + " && company.revenue <= max";
Query q = pm.newQuery(Employee.class, filter);
q.declareParameters(params);
Collection emps = (Collection) q.execute
    (new Float(500000F), new Float(1000000F));
```
JDOQL Examples

- Query with Multi-value Navigation:

```java
String vars = "Project p";
String filter = "projects.contains (p)"
    + " && p.name == "Kodo"";
Query q = pm.newQuery(Employee.class, filter);
q.declareVariables (vars);
Collection emps = (Collection) q.execute ();
```
Native SQL

- JDO 1.0 does not recognize SQL
- JDO 2.0 supports:
  - obtaining java.sql.Connection from PersistenceManager
  - defining SQL queries
    - may improve performance (good)
    - makes query non-portable (bad)
    - may support SQL-specific constructs (?)
    - may allow SQL DBA more control (!)
More Vendor Support

- Remove PersistenceCapable requirement
  - Remove pre-processor, post-processor requirement
  - Allow non-enhancer versions of JDO implementation
  - All other compliance requirements remain