The Future of Persistence

Craig Russell
Sun Microsystems, Inc.
JAX Conference May 2005

Sun microsystems
jax 2005

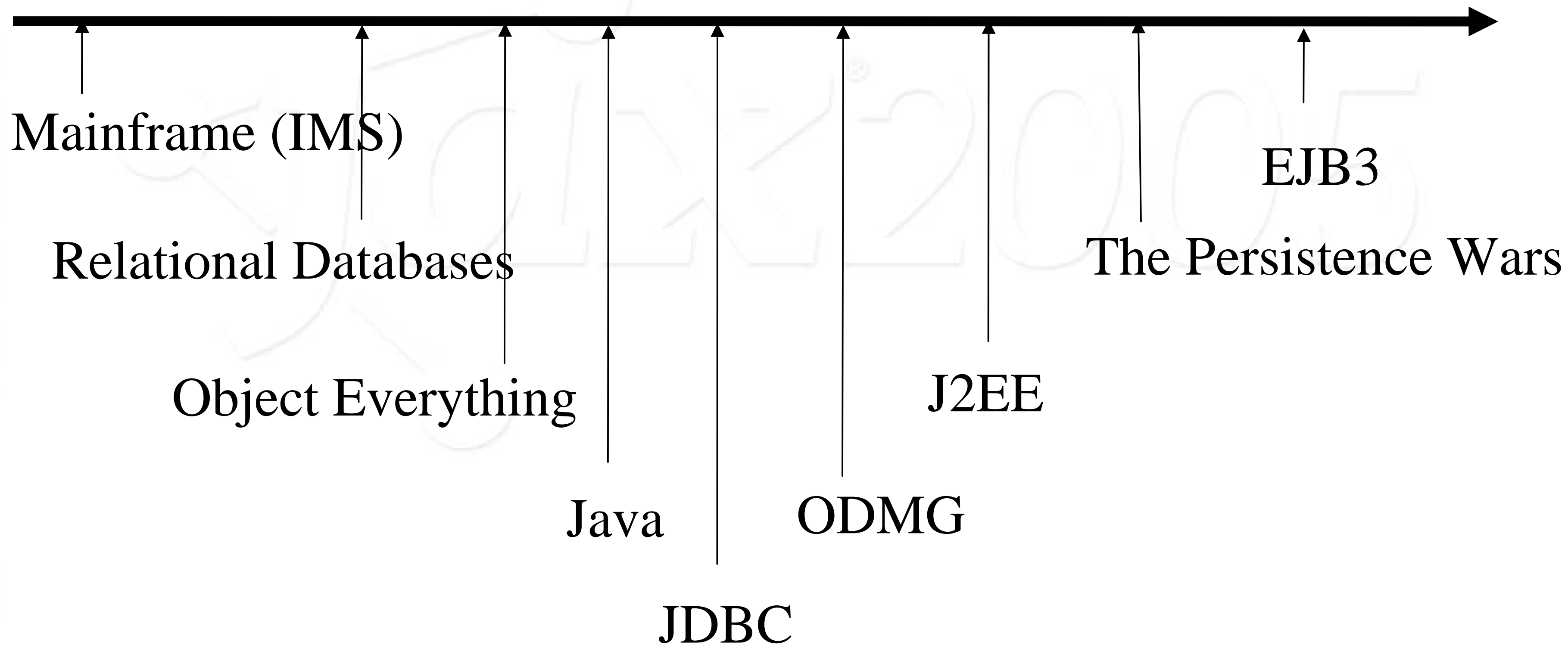
Objectives

- A Brief History
- Models of Persistence
- Current Issues in Persistence

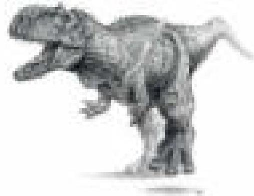
Presenter's Baggage

- Craig Russell, Architect
  - Sun Java Web Services
    - Sun Java System Application Server
    - Java Blend, CMP, EJB3
  - Specification Lead
    - JSR 12 Java Data Objects
    - JSR 243 Java Data Objects 2.0
  - Author of "Java Data Objects" published 2003, O'Reilly

How Did We Get Here?



How Did We Get Here?



Mainframe (IMS)

Relational Databases

Object Everything

Java

JDBC

ODMG

J2EE

The Persistence Wars

EJB3

Sun microsystems
jax2005

How Did We Get Here?

Mainframe (IMS)

Relational Databases

Object Everything

Java

JDBC

ODMG

J2EE

The Persistence Wars

EJB3

Sun microsystems
jax2005

How Did We Get Here?

Mainframe (IMS)

Relational Databases

Object Everything

Java

JDBC

ODMG

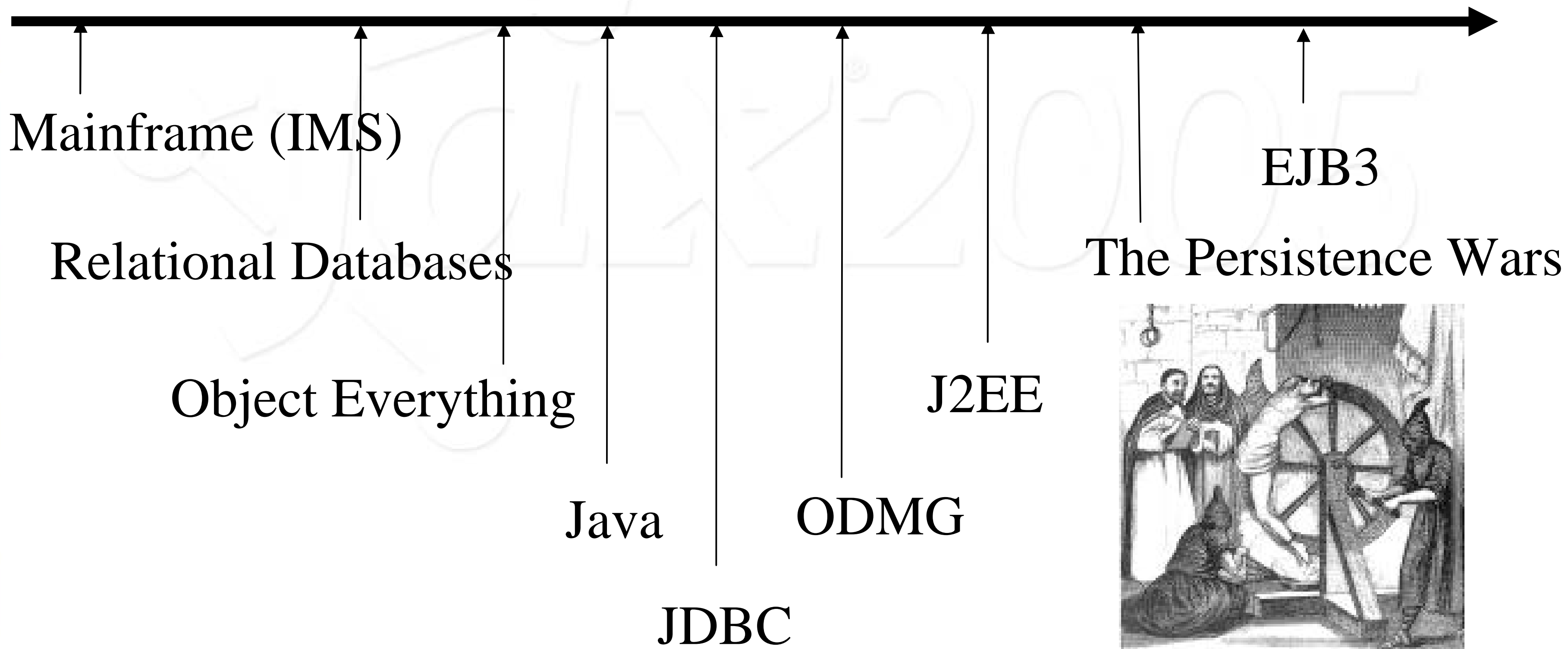
J2EE

The Persistence Wars

EJB3

Sun microsystems jax2005

How Did We Get Here?



Sun microsystems
jax2005

How Did We Get Here?

- Mainframe (IMS)
- Relational Databases
- Object Everything
- Java
- JDBC
- ODMG
- J2EE
- The Persistence Wars
- EJB3

Sun microsystems

jax2005

Alternatives for Persistence

- JDBC
- Component Model (CMP)
- Data Access Object
- Domain Object Model

JDBC

- Simple in Concept
  - Write SQL, execute it, voila!
- Tricky in Practice
  - Repetitive, Repetitive, Boring, Error-prone
  - SQL is not portable
  - Tools can help...
- JDBC 4.0 might provide some benefits
  - Map ResultSet to JavaBean

Component Model

- Persistent Entities = Components (CMP)
- Well-defined life cycle, query
- Combine Domain Model with Component
- Local, Remote Access
- Work only inside Application Servers

Data Access Object

- Abstract all Persistence code into DAO
  - Removes details of implementation from application – GOOD!
  - Allows substituting different implementations:
    - EJB CMP, JDO, Hibernate, TopLink,...
    - Different back end systems: LDAP, XML,...
- Moves problem one step lower in stack
  - Persistence code in domain logic – BAD!

Domain Object Model

- Separate Persistence-Aware Classes from Domain Classes
- Persistent Schema (Entities) are represented by Java Objects (Classes)
- Instances in memory represent corresponding data in database
- Modifications to instances are reflected in the database

Domain Object Model

- POJO: Plain Old Java Object
- POPO: Plain Old Persistent Object
- Restrictions on Domain Objects
  - No dependencies on environment:
    - Thread, System, Socket, Stream, etc.
  - Relationships expressed as Java Collections
  - Final, static fields cannot be persistent
  - Requirements on equals, hashCode

Domain Object Model

Life Cycle and Query

- Persistence-Aware Classes need Persistence Manager for:
  - Storing instances the first time
  - Deleting instances from the database
  - Querying the database
  - Managing the cache of persistent instances
  - Demarcating transaction boundaries

Domain Object Model Mapping

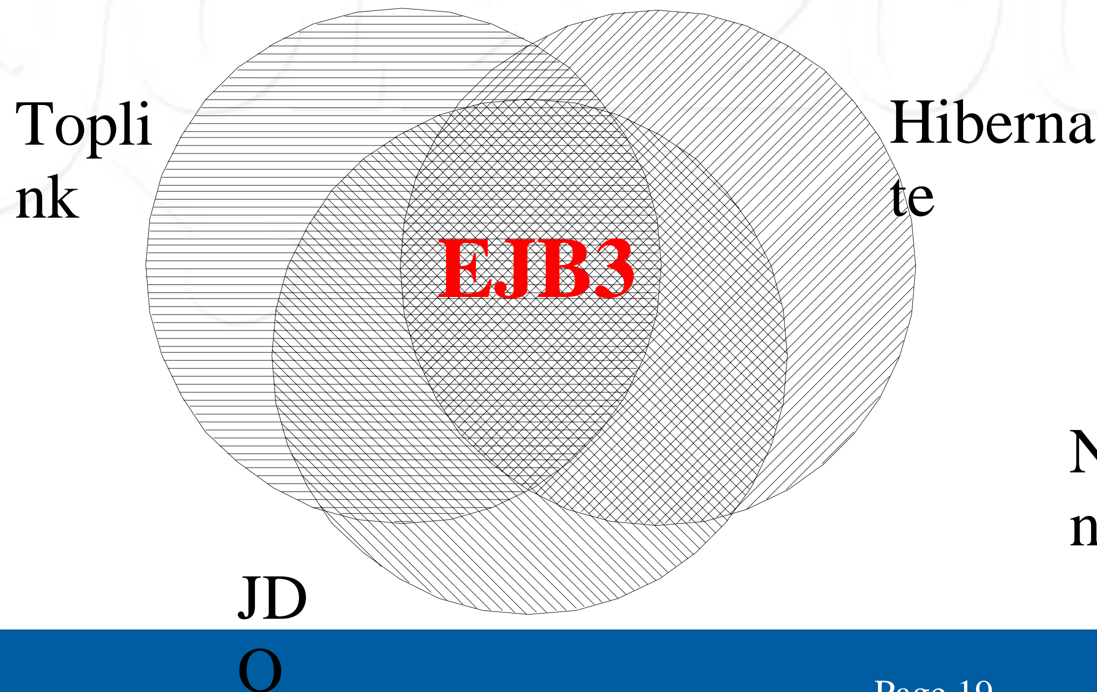
- Persistent Schema are mapped to Java Objects
  - Tables are mapped to Classes
  - Columns are mapped to fields
    - Simple column types to simple or wrapper types
  - Relationships (foreign key constraints) are mapped to Java fields
    - Join tables are mapped to Java Set (both sides)
    - Multi-column join tables are mapped to Java Map

The Debate has Changed

- Before:
  - Components vs. Domain Object Model
  - DAO vs. Everything Else
  - CMP vs. JDO
- After JDO/EJB3 Announcement:
  - What should we call the new interface?
  - How best to implement EJB3
  - How best to extend EJB3

Implement and Extend EJB3

- EJB3 doesn't solve every problem
  - (There, I've said it)



NOTE: Figure not to scale

If EJB3 Isn't Enough

- Wait for the Next Release, or
- Extend EJB3
  - 1. “Write part of your app in Hibernate”
  - 2. “Write part of your app in TopLink”
  - 3. “Write part of your app in JDO”
    - And if that doesn't work,
    - Write part of your app in
    - {Kodo, LiDO, OpenAccess JDO, JPOX,...}

Current Issues

- Annotations vs. XML for Metadata
- Field vs. Property Persistence
- Transaction Control
- Data Transfer Objects (DTO Pattern)
- Byte-code Modification, Subclassing, or Reflection

Annotations vs. XML

- Object Model Metadata
  - Dependent objects (e.g. LineItem)
  - Object Query
- Mapping Metadata
  - Class-to-Table, Field-to-Column
  - Special field types: Map, List, Array mapping
  - Inheritance mapping
  - Database Query

Annotations vs. XML

Use Annotations for Object Metadata

Use XML for Mapping Metadata

Sun microsystems

jax 2005

Property vs. Field Persistence

```
class Employee {
    private String spouse;
    public String getSpouse() {
        return spouse;
    }
    public void setSpouse(String Spouse) {
        if (!isMarried()) error("Not Married!");
        this.spouse = spouse;
    }
    private int maritalStatus;
    public boolean isMarried() {
        return maritalStatus == MARRIED;
    }
    public int getMaritalStatus() {
        return maritalStatus;
    }
    public void setMaritalStatus(int status) {
        maritalStatus = status;
    }
}
```

Property vs. Field Persistence

- Property
  - You implement get and set methods needed by impl
  - Mixes persistence and business logic (validation rules)
- Field
  - The implementation does this for you under the covers
  - Clean separation of business rules vs. impl

Separating the business interface (client view) from the persistence interface is best practice.

Property vs. Field Persistence

Use Field Persistence

Implement Accessors with Business Logic

Use Property Persistence for Lazy Loading, e.g. Large Objects

Transactions

- Inside the J2EE Container
  - Container-Managed Transactions
  - Bean-Managed Transactions
- Outside the J2EE Container
  - No Container-Managed Transactions
  - IOC Frameworks are your friends

Transaction Control

```
class TransactionFilter(...) {
    void doFilter {
        try {
            userTx.begin();
            pmanager = pmanagerfactory.get();
            context.add("PMANAGER", pmanager);
            next.doFilter(...);
            userTx.commit();
        } catch (Exception ex) {
            userTx.rollback();
        } finally {
            context.remove("PMANAGER");
            pmanager.close();
            pmanager = null;
        }
```

Transactions

Inside Server

- Use Container-Managed Transactions

Outside Server

- Put Transaction Logic in one Component
- Better: Put it into one class
- Best: Use IOC to isolate it

Sun microsystems
jax2005

Data Transfer Objects

- Multi-tier Distribution Pattern
  - Value objects
  - Limited relationships with others
- Consider Simple Persistent Objects
- Use Attachment and Detachment APIs
  - Objects are detached upon Serialization
  - Objects are reattached (merged) explicitly

Data Transfer Objects

- Use Detached Objects
  - instead of DTO
- Keep Detached Objects Simple
  - Behavior
  - Field Types

Sun microsystems
jax 2005

Byte-code Modification, Subclassing, or Reflection

- Implementation Choice for Domain Object Classes
- Their Choice Affects Your Code

Sun microsystems
jax 2005

Byte-code Modification, Subclassing, or Reflection

Byte-code Modification is the Best Strategy for Field Persistence

Reflection Currently Does Not Perform

Take Away Thoughts

- Domain Object Model is King
  - Become an Expert
- EJB3 is becoming
  - It will evolve
- Choose Now for the Future

The Future of Persistence

Craig Russell