Persistent and Java – A Balancing Act
Atkinson

Representing Database Programs as Objects
Maier

William Cook
September 9, 2003

Topics

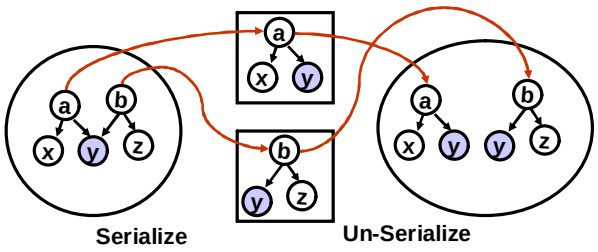
- Review of persistence mechanisms
- Analysis of PJama
- Large-scale experimental CS research

Persistence Mechanisms

- Persistence
  - Data lasts longer programs that use it
- Mechanisms (for Java)
  - Files
  - Serialization
  - JDBC

Serialization

- Store out a graph of objects
  - Used for remote procedure calls
  - Does not preserve identity
  - Doesn't scale well



Other Approaches

- Links to Relational DBs (JDBC)
- Object-Relational Mapping
- Object Database Mapping
  - ODMG and Gemstone/J
- Java Data Objects (JDO)
- Enterprise Java Beans (EJB)

- We will review most of these…

Orthogonal Persistence Hypothesis

- If
  applications developers are provided with
  a well-implemented and well-supported orthogonally persistent programming platform
- then
  a significant increase in developer productivity will ensue
  and
  operational performance will be satisfactory

PJama Project

- Designed to test OPH (Orthogonal Persistence Hypothesis)
  - Orthogonality
  - Persistence Independence
  - Durability
  - Scalability
  - Schema Evolution
  - Platform Migration
  - Endurance
  - Openness
  - Transactional
  - Performance

Missing:
Distribution


```
import java.io.*;
import org.opj.*;
public class Main {
  static { OPRuntime.roots.add( Main.class );}
  private static float exchangeRate;

 public static void main( String args[ ]) {
   while(true) {
      int cmd = in.read( ); String arg = in.readLine( );
      switch (cmd) {
        case 'c': { float amt = Float.parseFloat( arg );
                System.out.println( amt +"converts to" + amt *
                                                  self.exhangeRate); }
        case 'x':self.exchangeRate = Float.parseFloat( arg );
        case 'q': System.exit( 0 ); default: System.exit( 0 );
          }
   OPRuntime.checkpoint( );
   } // while
}// main
}
```

from Sun via Dr Patrick Marais

Achievements – Complete

- Persistence Independence
  - Any object linked from *root* is persistent
  - Just like in normal garbage collection
- Schema Evolution
  - Apply transformations to objects
  - (offline transformation prevents endurance)
- Durability
  - Recovery (ARIES Algorithm)
  - (offline backup prevents endurance)

Achievements – Conditional

- Endurance
  - 100% uptime hard to achieve
    - Durability: Complete backup
    - Schema Evolution: Offline transformation
  - Quality
    - bugs prevent multi-threaded applications from running for more than a few minutes
- Transactions
  - simple mode: long transactions
  - all threads consistent before checkpoint
- Performance
  - 15% slowdown ignoring disk
  - No distribution (multiple servers)

Achievements – Conditional

- Orthogonality
  - Saving window state (AWT and Swing)
  - Saving JDBC and CORBA connections
  - *Thread* caused problems
  - Also: lots of C/C++ code makes this hard
- Platform Migration
  - Moving from Java 1.1 to Java 1.2
  - Conflict with storing code in the database?
- Scalability
  - Only considered data set size
- Openness
  - Only considered restoring external connections

Stabilization and Threads

- Stabilization in multi-threaded programs can be problematic, if the threads are not cooperating.
  - During a stabilization all user threads are stopped, therefore a stabilization is atomic, isolated and durable. However, since a stabilization is global, in that it applies to all persistent roots, there is *no guarantee of semantic consistency* for threads other than the one invoking the stabilization.
- This situation will be corrected in a future release by the provision of persistent threads which will allow a thread to resume and eventually reach a consistent state.

Transient annotation

- "Variables may be marked transient to indicate that they are not part of the persistent state of an object."
  - class Point {
    int x, y;
    transient float rho, theta;
    }
- If an instance of the class Point were saved to storage by a system service, then only the fields x and y would be saved.
  - This specification does not yet specify details of such services; we intend to provide them in a future version of this specification
  - (also conflicts with new Java transient keyword)

Code in the Database

- Considered critical for completeness
  - What code is stored?
    - Just "user code" or base libraries too?
    - Probably all java byte-codes
  - Problem with updating platform
    - how do you tell which code in database to replace?

Orthogonal Persistence Hypothesis

- Need real users
  - typical development teams building and evolving applications
  - users imposing typical workloads
- Observe
  - team's problems, successes, and productivity
  - typical workload
- It's not happening
  - platform quality has not been achieved
  - engineers are right to avoid untested platform

Reaching critical mass

- Scale of experiment
  - Large initial development
  - Support for five years
  - Leverage multiple projects together…
  - Need $25M to test hypothesis
- Problem with hypothesis
  - X improves productivity of application developers under realistic conditions…

Industry Adoption

- Existing practices
- Displaced problems
  - Platform builders don't understand application developers' problems
- Distribution drives applications
- Lack of credibility
- Alternatives look better
- Language lock-in
- Dominance of glue-ware
  - Applications are not just written on Java

Comparison with Relational DBs

- RDBMS
  - provided simple solution to real problems
    - independent from disk formats, concurrency, etc
  - Adopted despite serious technical issues
- Java was probably the same story
- Orthogonal Persistence
  - No simple message
  - "Indexing is not a built-in feature"
    - this requires more work
  - At this point it is still an act of faith

Large-Scale Experimentation

- Is there a case to answer?
  - Experiment must have definite outcome
  - Use theory as a guide
- Design a family of experiments
  - "Several teams attempting tasks from a chosen set, with different technologies"
- Conducting experiments
  - Need to pay the subjects
- Interpretation
  - use case in extrapolating…
- Resources, Teams, Communities
  - long-term goals, like astronomy, biology, etc…

Comment

- "inhibited by negative attitudes toward those who try to measure properties that are not easily *quantified*"
- Will the cost be too great?

Some Thoughts

- If you are going to be radical
  - Pick one problem and solve it well
- Define a hypothesis that can be tested
  - Measuring improved productivity is very hard
- Find ways to evaluate
  - Simulate real load
  - Simulate development process
- Identify a critical problem
  - Do some work to validate it

Evaluating Languages

- A suite of standard problems
  - Solutions in different languages
  - Change the problems occasionally
    - to simulate maintenance
  - Must be sufficiently large to be real
    - but not too large to require massive investment
    - must include appropriate data sets
- Uses
  - Evaluating new tools & approaches
  - Examples for students
  - Best solutions from the field, not just individual

Representing Programs as Objects

- What makes a DB computational environment powerful?
  - Encapsulation of iteration
  - Picking operations out of programs
- Persistent Programming Languages
  - Why they don't solve it all

Encapsulation of Iteration

- Characteristics
  - Small number of iteration constructs
  - Expressed at high level => different orderings of operations are allowed
    - A key characteristic of *functional programming*
- Optimization
  - Iteration constructs examined in detail
  - Optimized based on underlying algebra
  - Use knowledge of physical layout and required access patterns

Picking operations out of programs

- Characteristic
  - Complex data-intensive operations picked out of programs for execution in the storage manager
  - The idea here is that the "query" parts of a unified program can be lifted out and moved to the database engine for execution
    - What kind of a unified programming/query language would support this?
- Proposal
  - Complex operations stored in database

Persistent Programming Languages

- Why they don't solve it all
  - Does not encapsulate iteration
  - Instead, uses traditional "for loop"s
  - "The storage manager ends up with an object-at-a-time interface"
- Alternatives
  - "Logic or functional languages gives a better start"
  - Generally inadequate for expressing update and IO

Encapsulation / Associative Access

- Encapsulation
  - Model behavior and structure (objects)
  - Aids code reuse and modification
- However…
  - Query processing depends on knowledge of structure, rather than just [interface]

Embedded DML

- DML = Data Manipulation Language
  - Refers to use of SQL operators, SELECT, UPDATE, DELETE, from within standard programming languages (PL)
- Coined phrase "Impedance Mismatch"
  - interface between PL and DML
  - No type system spanning PL and DML
  - (No mention of objects)
  - Generating application from data model ensures types are the same initially
    - but does not handle evolution

"Abstract Objects"

- Idea not fully developed
- Avoiding syntax is not significant
- Structures look like XML…