Object Relational Database Design – impedance mismatch or expectation mismatch?

David Bowers
Agenda

• Object Relational Design
• The challenges
  – Making the right choices
  – Understanding “impedance mismatch”
  – Deciding what to include
• Why is it so difficult?
• The real problem?
People

- Part-time research students
  - Pat Roberts (University of Brighton)
  - Chris Ireland
  - Chris Anderton
- Co-supervisors
  - Mike Newton
  - Kevin Waugh
  - Leonor Barroca
Motivation

• Object-Relational features specified in SLQ-99
  – User defined types, typed tables
  – Inheritance, identity
  – Collection types, nested tables
• Implementations emerged over next few years
  – Of varying completeness
Is OR the “next great wave”?

• Not widely adopted

• Why not?

• Should it be?

• Does something make it difficult?
Received “wisdom”

- “impedance mismatch” doesn’t exist?
- Prescriptive approaches
  - Ambler
  - Soutou
  - Marcos et al
  - Mok & Paper
Is the wisdom really that wise?

• If it’s that easy, why aren’t all databases OR?
• The “Vietnam” of computer science (Neward)
  – Initial quick wins based on received wisdom rapidly overtaken by quagmire of issues
• 25 - 50% of Object – Relational application code addressing impedance mismatch (Keller et al)
• Missing the point?
Increasing the design options (Roberts)
Making choices

• Several options available for any archetype
  – Simple class – *simple table, typed table, type as column*
  – Structure attribute – *table, structured type, row type*
  – Two classes, 1:many –
    • *two tables with FKs*;
    • *three tables with FKs*;
    • *two typed tables with REF*;
    • *three typed tables with REFs*;
    • *A collection of REFs*;
    • *A collection of FKs*;
    • *A collection of REFs with return FK*;
    • *A collection of FKs with return FK*;
    • *Nested tables*
More choices

• Seven other archetypes
  – Multi-valued attribute (3)
  – Two classes with many:many association (6)
  – Two classes with one:one association (5)
  – Two classes with directional association (2)
  – Two classes, association and association class (5)
  – Aggregation (9)
  – Generalization (5)
Consequences of choices

- May impede usability
- Or ability to accommodate change
  - For example, if use typed tables, difficult to change structure of underlying SUDT
- Or even of CRUD operations
  - With some choices, particular operations become … challenging
    - e.g., finding elements in ARRAYS
- Not considering efficiency
  - Likely to be dominated by implementation platform
A “quality” approach

• No notion of “absolute” quality

• Rather, “fitness for purpose”

• Are the choices the most appropriate
  – For the application
  – And its environment
Quality criteria

- 1. Functionality (Does it represent a correct transformation from the conceptual model?)
- 2. Usability (Can it be used effectively?)
- 3. Maintainability (Can it cope with change?)

Quality characteristics:
- Integrity
- Seamlessness
- Simplicity
- Flexibility

From ISO 9126 Software Design Quality
A decision support framework

<table>
<thead>
<tr>
<th>Main priority:</th>
<th>Integrity</th>
<th>Simplicity</th>
<th>Seamlessness</th>
<th>Flexibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Realizations for a many-to-many association</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.1 Two foreign keys in the breakdown table</td>
<td>R R R</td>
<td>A A G</td>
<td>R A A</td>
<td>G A A</td>
</tr>
<tr>
<td>5.2 Two pointers in the break-down table</td>
<td>A A A</td>
<td>A R R</td>
<td>A R A</td>
<td>A A A</td>
</tr>
<tr>
<td>5.3 Collection of pointers and return collection</td>
<td>A A A</td>
<td>A A A</td>
<td>A A G</td>
<td>A A A</td>
</tr>
<tr>
<td>5.4 A collection of keys</td>
<td>A A A</td>
<td>A G G</td>
<td>A G G</td>
<td>A G G</td>
</tr>
<tr>
<td>5.5 One pointer and a collection of pointers</td>
<td>A A A</td>
<td>A R R</td>
<td>R R R</td>
<td>A A A</td>
</tr>
<tr>
<td>5.6 Nested normal form</td>
<td>G G G</td>
<td>G A A</td>
<td>A A A</td>
<td>A A A</td>
</tr>
</tbody>
</table>
Has impedance mismatch just “gone away”? (Ireland)

• Impedance mismatch first “appeared” with attempts to combine Objects and Relational Databases
• Different conceptual frameworks
• Different processing models

• Much effort to work around
  – Note comments by Neward and Keller
What makes a mismatch?

• No two “languages” have identical expressive power
  – Which means that may be some concepts that can be expressed in only one of them.
  – Translation between two languages is non-loss in at least one direction
  – May also be contrived

• When the two “languages” are “OO” and “database”, this problem is recognised
  – “Impedance mismatch”
Another framework!

- Two “silos” –
  - Relational, Object
- Four levels of abstraction
  - NOT modelling/design stages
  - Constructs at each level expressed in “vocabulary” of level above
  - Some (vague) similarity to MDA frameworks
Conceptual framework
The essence of a mismatch

- “round trip” transformation is non-loss
- Either expressive powers incompatible
- More likely – mixing up concepts from different levels of abstraction
- Mismatches can occur at each level
  - If not resolved, can impact on “implementation” at lower levels
  - Confusion of “semantics”
So, what about OR?
How much should you include? (Anderton)

• If classes transform to SUDTs…
  – Should you include every Class from the static view of a UML model?
  – If not, which ones?
  – How do you know?
• Which classes are required only during execution, which need to be persistent?
  – Why?
Automatic schema generation

• If you can identify which classes to include, then schema generation can be automated

• Marcos et al – tag class model
  – So (OO) modeller needs to understand DB design

• Can you deduce from whole model
  – Including active models
    • Actions such as create, retrieve, store…
A more fundamental problem?

• OOA developers use database technology to introduce persistence
  – Is persistence model well-defined in UML?
• Databases have undergone a long evolution for data processing
• Fundamentally different processing models
  – Navigation versus set-at-a-time
• Is there really a clash of cultures?
If you have a database...

• Then:
  – You NEED
    • transactions
    • shared access
    • real-world identity
    • ad-hoc access
    • integrity constraints
    • security
    • etc.
But, from the OOA perspective…

- Do you just want somewhere to “store” your objects?
- They’re YOUR objects – no-one else should even look at them
- Some notion of OID
- The ability to get them back as complete objects
  – One at a time!
- And you’re quite capable of looking after constraints, transactions
  – And everything else…?
An “Expectation Mismatch”?

- How much of the “impedance mismatch” is a consequence of differing expectations?
- Do persistent objects need “proper” DBMS services?
  - Or be stored as Objects?
- Does it really matter if products such as Hibernate mix up several different levels of abstraction
  - From the perspective of the OOA designer?
- If you have a database full of Objects…
  - Could you ever stop other people fiddling with them?
Summary

- Despite the pressure to develop OR extensions to SQL, they are not widely used.
- Perhaps the additional design choices seems to make them difficult to use?
- Is there really a good understanding of impedance mismatch?
- Is there clarity on how much of an application to transform into an OR schema?
- Are the two communities just looking for completely different things?
References

• Structural Transformations in Object-Relational Design: A Framework for Improving Quality, Patricia Roberts, PhD thesis, Open University 2010
• A Classification of Object-Relational Impedance Mismatch, Chris Ireland, David Bowers, Kevin Waugh, Mike Newton, DBKDA 2009
• Exploring the use of Mixed Abstractions in SQL:1999, Ireland, Bowers, Newton, Waugh, DBKDA 2010
• Exploring the Essence of an Object-Relational Impedance Mismatch, Ireland, Bowers, Newton, Waugh, submitted to DBKDA 2011
References (2)