Role-based Object-Relational Co-Evolution

Sebastian Götz, Sebastian Richly and Uwe Aßmann

27.06.2011

RAM-SE 2011 co-located with TOOLS 2011, Zurich, Switzerland
OO Domain Model:

Relational Schema:

Person
- #
- name
- age
- student_id
- semester

Course
- name
- tutor_id
- course_id

Attendees
- #
- person_id
- course_id

1) ORM = Object-Relational Mapper
Problem

OO Domain Model: evolves:

Relational Schema needs adjustments:

- Time-consuming
- Rare support for changes beyond addition

Splitting Relations

New Relations

Redirection of Foreign Keys

Foreign Key Reification

27.06.2011  Götz et al. - Role-based Object-Relational Co-Evolution  Slide 3 of 12
Goal

Automatic Adaptation of Object-Relational Mappings keeping the relational schema intact

Benefits
- Rapid Development
  - postpone DB adjustments
  - independent parallel work
  - postpone query adjustments

Drawbacks
- Performance Penalties
  - in large scale scenarios (many versions)

Application
<table>
<thead>
<tr>
<th>OO Domain Model</th>
<th>ORM</th>
<th>DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>write</td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>App-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lication</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ORM

Relational Schema

Rapid Development
- postpones DB adjustments
- independent parallel work
- postpones query adjustments

Performance Penalties
- in large scale scenarios (many versions)

OO Domain Model

ORM

Relational Schema

DBMS

Evolves

stays unchanged

27.06.2011
Götz et al. - Role-based Object-Relational Co-Evolution

Slide 4 of 12
Object-Roles

Classes (Players)  Role Types  Contexts

Person

Employee

Company

Student

University

joe:Person

:Employee  IBM:Company

:Employee  SAP:Company

:Student  TUD:University
→ Change Descriptor Roles (CDRs)

**Student**
- student_id
- semester

**Tutor**
- salary

**Lecture**
- lect_no
- name

- old: Course
- old: course_id

**Attendees**
- student_id
- semester

**Student**
- student_id
- semester

**Course**
- name
- tutor_id
- course_id

**Tutor**
- salary

**ORM**

- $M_{oo,new}$
- $M_{oo,old}$
- $M^+_R,old$
1. **Separate Relation per Role**
   - Complex queries / many joins

2. **Single Relation per Player-Role combination**
   - Sparse data
   - Semantic redundancy

3. **Normalized Single Relation**
   - Automatic normalization unfeasible
   - (due to high complexity)

4. **SQL:99 subtable**
   - subtable = Role, supertable = player
   - context requires additional foreign key
   - rarely supported by DBMS
Developer renames class *Lecture* to *Course*

- role decapsulates player (field access)
- interception of player calls
PullUp Attribute CDR

NewOwner
+ attr₁
+ attr₂
+ attrₙ
+ attrₓ

Inheritor
+ attr₁
+ attr₂
+ attrₙ
+ attrₓ

OldOwner
+ attr₁
+ attr₂
+ attrₙ
+ attrₓ

NewOwner'
+ attr₁
+ attr₂
+ attrₙ
- attrₓ

Unaware
+ attr₁
+ attr₂
+ attrₙ
- attrₓ

OldOwner'
+ attr₁
+ attr₂
+ attrₙ
+ attrₓ

ORM

27.06.2011 Götz et al. - Role-based Object-Relational Co-Evolution Slide 9 of 12
Solution

Developer's intent required

Application

Change-descriptor roles

ORM

Database

${M_{\text{OO,new}}}$

${M_{\text{OO,new}} \leftrightarrow M^{+}_{\text{OO,old}}}$

${M^{+}_{\text{OO,old}} \leftrightarrow M^{+}_{R,old}}$

${M^{+}_{R,old}}$
**Related Work**

**Coupled Evolution in Model-driven Engineering** (e.g., [5, 6])

- Evolve model instances according to metamodel changes
- The work presented here does not consider metamodel changes
  - instead evolving source of model transformation (OO to Relational)
  - different goal: shielding the target from complex changes
**ComeBack** [10]

- shielding plugins/clients from framework evolution
- holistic adaptation layer
- Framework looks unchanged to clients vs. Application looks unchanged to ORM
- Focus on control-flow (not on data-flow)
• **MeDEA** [7] allows to apply application schema changes to the DB
  • user needs to provide *migration script*

• **Terwilliger et al.** [8] handle object-relational co-evolution by transforming
  • Changes of the application to
    Changes of the mapping between the application and the DB
  • The goal is not to postpone model migration, but
    ▪ To automatically perform model migration

• Approaches like **PRISM** [9] allow to automatically evolve database queries
**Problem**
- Evolution of object-oriented domain model
- Adjustments to relational schema required
  - Time-consuming
  - Changes beyond additions poorly supported by DBMS

**Goal**
- Evolve OO domain model keeping relational schema intact
  - Fosters development productivity
    - ability to postpone changes
    - parallel development (on the same data)

**Solution**
- Change-Descriptor Roles / Holistic Adaptation Layer
  - Adapting new OO domain model to its old version
  - Hiding changes from the ORM
Thank You

Questions?
[1] Baldoni, M.; Boella, G. & van der Torre, L. 
Roles as a Coordination Construct: Introducing powerJava

The Implementation and Execution Framework of a Role Model Based Language, EpsilonJ

[3] He, C.; Nie, Z.; Li, B.; Cao, L. & He, K.
Rava: designing a Java extension with dynamic object roles

[4] Herrmann, S.
ObjectTeams: Improving Modularity for Crosscutting Collaborations

Towards Synchronizing Models with Evolving Metamodels.


27.06.2011 Götz et al. - Role-based Object-Relational Co-Evolution
• **Why Object-Roles?**
  - Roles (only) perform structural adaptations
  - Two interacting collaborations
    - User interacts with new version objects
    - ORM interacts with old version objects
  - New role layers can be added on the fly

• **Indirection imposes problems:**
  - **Performance**
    - → meant for rapid development
    - → not for productive use
  - **Debugging**
    - → Debugging the OR mapping is hard
    - → Debugging of application is not effected

• **Languages supporting role-based OO**
  - powerJava [1], EpsilonJ [2], Rava [3], **OT [4] (most mature)**
• **Scalability**
  - Tradeoff between usability and scalability in terms of performance
    - Changes between first and current version → 1 layer of indirection
    - Changes between last and current version → N layers of indirection

• **Weaknesses of the Approach**
  - Annotations to be provided by the developer
    (to identify the semantics of changes)

• **Tool Support**
  - The annotations can be generated using, e.g., the IDE refactoring log
  - CDRs are generated by default

• **Changes hard or impossible to model**
  - changed visibility might look like hard to model
    - but is not, due to *decapsulation* [4] by roles
Developer renames class Lecture to Course and introduces new subclasses Lecture and Seminar.