Exploring Role-Based Adaptation

Sebastian Götz, Defense “Großer Beleg” 20.11.08
1. Motivation
2. Elements of Role-based Adapters
3. Realization in ObjectTeams/Java
4. Demo
5. Conclusion
6. Future Work
1. Motivation

- **Student**
  - printGrades() : void
  - getMatrikel() : int

- **Report**
  - printReport(data: DataRows) : void

- **DBComponent**
  - retrieve(q: DBQuery) : ResultSet

- **already existed**
- **is to be improved**
1. Motivation

(Adapter Design Pattern [1] applied twice)

(intertwined code)

Student

- printGrades() : void
- getMatrikel() : int

Report

- printReport(data: DataRows) : void

DBComponent

- retrieve(q: DBQuery) : ResultSet

Adaptee 1

Target 1&2

Adaptee 2

Roles annotated as in [2]

//Create Query from Student's Data
DBQuery q = new DBQuery(...student.getMatrikel()...)
ResultSet rs = db.retrieve(q);

//Transform ResultSet to DataRows
DataRows data = new DataRows();
Subject sbj = new Subject(rs.getString(1), rs.String(2));
data.addString(sbj.toString());

//...
report.printReport(data);
1. Motivation

Goal: separation of adaptation concerns

Student

- printGrades() : void
- getMatrikel() : int

Report

- printReport(data: DataRows) : void

Database Domain

- DBCOMPONENT
- retrieve(q: DBQuery) : ResultSet

University Domain

- Student

Reporting Domain

- Report
2. Elements of Role-based Adapters

➤ implement adapter roles as first-order programming constructs

- **Student**
  - `printGrades() : void`
  - `getMatrikel() : int`

- **Report**
  - `printReport(data: DataRows) : void`

- **DBComponent**
  - `retrieve(q: DBQuery) : ResultSet`

- **Target 1&2**

- **Adapter 1&2**

- **Adaptee 1**

- **Adaptee 2**
2. Elements of Role-based Adapters

⁻ split adapter

<table>
<thead>
<tr>
<th>Target role</th>
<th>Adapter role</th>
<th>Adaptee role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>Report</td>
<td>DBComponent</td>
</tr>
<tr>
<td><code>printGrades()</code> : void</td>
<td><code>printReport(data: DataRow) : void</code></td>
<td><code>retrieve(q: DBQuery) : ResultSet</code></td>
</tr>
<tr>
<td><code>getMatrikel()</code> : int</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Client role omitted (as it is left open)*
2. Elements of Role-based Adapters

- type conversion implemented twice (for Report and DB component)
- code replication

- type conversion should not be the task of In-/Out-Roles
2. Elements of Role-based Adapters

- UniAmbassador
- ReportingAmbassador
- Student
  - printGrades() : void
  - getMatrikel() : int
- Report
  - printReport(data: DataRows) : void
- DatabaseAmbassador
- DBComponent
  - retrieve(q: DBQuery) : ResultSet

- Ambassadors for type conversion
- no code replication
- reusable
3. Realization in ObjectTeams/Java

- use ObjectTeams [3]
- roles are played by actors in a context [4]
3. Realization in ObjectTeams/Java

Student
- `printGrades() : void`
- `getMatrikel() : int`

Report
- `printReport(data: DataRows) : void`

`StudentAdapter`
- `+distribute(matrikel : int) : void`

`StudentIn`
- `printGrades « printGrades`

`ReportOut`
- `printReport » printReport`

`DBOut`
- `retrieve » retrieve`

`DBComponent`
- `retrieve(q: DBQuery) : ResultSet`

Ambassadors are omitted for clarity.
Demonstration
5. Conclusion

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Less effort for maintenance</td>
<td>• More initial coding effort</td>
</tr>
<tr>
<td>• More flexible allocation of developers</td>
<td>• More initial training effort</td>
</tr>
<tr>
<td>• Easier development of adapters</td>
<td></td>
</tr>
<tr>
<td>• (Reusable Ambassadors)</td>
<td></td>
</tr>
</tbody>
</table>

Published as ECOOP-workshop paper [5] at RAM-SE 2008
6. Future Work

- Collection of Empirical Data Using an Experiment
- Parameterizable RBAs
- Using RBAs for Change Encapsulation
Thank you very much for your attention.

Any Questions?
5. Future Work

- **Collection of Empirical Data Using an Experiment**
  
  - 2 Teams of 5 Students each
    
    1st Phase:
    - 1st Team develops the UMS using class-based adapters
    - 2nd Team uses RBAs
    
    2nd Phase:
    - Both Teams incorporate a set of changes
    
    Time measurement, Software Metrics and Surveys after 1st and 2nd Phase.
5. Future Work

• **Parameterizable Role-based Adapters**

  - RBAs take an *ordered* set of components as parameter
5. Future Work

- **Sequence RBA**

- **Pick RBA**
5. Future Work

- Using Role-based Adapters for Change Encapsulation

- Easy location of positions in code to change
- But multiple RBAs need to be adjusted
5. Future Work

- Using Role-based Adapters for Change Encapsulation

- Put an RBA in front of the new version
- Adapt new version to old version

⇒ Role-based realization of **ComeBack**-adapters [6]


