Reverse Engineering
Models of Component-Based
Software Systems

CSMR 2008, Athens

Landry Chouambe (Netviewer AG)
Benjamin Klatt (Universität Karlsruhe TH)
Klaus Krogmann (Universität Karlsruhe TH)

Chair Software Design and Quality
Institute for Program Structures and Data Organization (IPD)
Faculty of Informatics, Universität Karlsruhe (TH)
Motivation

- Recognize component structures from source code
  - Comprehension of legacy software systems
  - Planning the extension of existing systems

- Target Model: Source for model transformations
  - Generate code stubs for extended software
  - Prerequisite for model-based reasoning

- Static source code analysis
- Use of source code metrics
- Clustering approach

Motivation / Introduction ➔ Approach ➔ Evaluation ➔ Conclusion
Component

- A component...
  - is a unit defined by its interfaces
  - allows composition (hierarchy)
- And is NOT equal to...
  - A single class
  - A package
  - A module
  - A method
  - An EJB
A Strong Component Definition

- Components are solely communicating via their interfaces
  - Provided Interfaces
  - Explicit Required Interfaces
- Components can be part of composite structures (composite components)
- Interfaces define a number of services
  - Serving as a contract between component services
  - Use data structure arguments only

cf. [Szyperski2003]
What is Recognized?

- Components
  - Boundaries
  - Their related classes
  - Composite Component

- Interfaces
  - Connectors

- Target Model: Instance of an ECORE-Meta-Model
  (Palladio Component Model)
Related Work

- Pattern-based approaches
  - Regular expressions (Pinzger et al.)
  - Patterns as queries (Kazman et al.)
  - Recognizer (Chase et al.)
- Clustering (e.g. Mancoridis et al.)
  - Reconstruction as an optimisation problem
  - Search for good clusters ("subsystem")
- Voting approach (Koschke)
  - Uses total sums of metrics
- Tools: Bauhaus / Sotograph / Lattix LDM / SonarJ
  - Follow a weaker component definition (class, module, ...)

Motivation / Introduction ➔ Approach ➔ Evaluation ➔ Conclusion
Approach

Process, Metrics, Weights, Implementation
Process

Motivation / Introduction ➤ Approach ➤ Evaluation ➤ Conclusion

Krogmann: Reverse Engineering Component-Based Systems 17/09/2008 8
1. Recognition of initial Components

Implemented Interfaces

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>CB</td>
<td>CC</td>
<td>CC</td>
</tr>
</tbody>
</table>

Required Interfaces

<table>
<thead>
<tr>
<th>CA</th>
<th>CA</th>
<th>CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2</td>
<td>I3</td>
<td>I1</td>
</tr>
</tbody>
</table>

Intermediate Components

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>CB</td>
<td>CC</td>
<td>Cc</td>
</tr>
</tbody>
</table>

Initial Components

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>CB</td>
<td>CC</td>
<td>Cc</td>
</tr>
</tbody>
</table>
2. Evaluation of Component Candidates

- Name Resemblance of $\{C_i, C_j\}$
- Interface Violations of $(C_i, C_i)$
- Interface Violations of $(C_j, C_i)$
- Coupling of $(C_i, C_j)$
- Coupling of $(C_j, C_i)$
- Package Mapping of $\{C_i, C_j\}$
- Subsystem Component of $\{C_i, C_j\}$
- Distance from the Main Sequence of $\{C_i, C_j\}$

$Q = \sum \left( x_1 F_1(x_1, w_1) + x_2 F_2(x_2, w_2) + x_3 F_3(x_3, w_3) + x_4 F_4(x_4, w_4) + x_5 F_5(x_5, w_5) \right)$
3. Composition

- Component candidates
  - To reduce complexity, weights are combined for pairs of component candidates
  - Complexity: $O(n^2)$
Approach

Initialisation of the reverse engineering process

- Initial Components

Evaluation of components
- Uses Metrics

Combining existing components
- Sets of more than two components

- Add non-assigned classes to existing components

Judge recognised components
- Save results
- Creation of a PCM-instance

Extraction of sourcecode structures using Sotograph

Deployment-Descriptors
Evaluation

Case Studies
Comparison of manual and automated Reverse Engineering Results
Evaluation: Case Studies

<table>
<thead>
<tr>
<th>System</th>
<th>LOC</th>
<th>Classes / Interfaces</th>
<th>Component-Based</th>
<th>EJB</th>
<th>Detected Components</th>
<th>Iterations / Time</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUBiS</td>
<td>8,202</td>
<td>41 / 34</td>
<td>✔</td>
<td>✔</td>
<td>17</td>
<td>2 / 1.5 sec</td>
<td>Good</td>
</tr>
<tr>
<td>Open Source</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ohioedge</td>
<td>78,516</td>
<td>249 / 132</td>
<td>✔</td>
<td>✔</td>
<td>13</td>
<td>5 / 31 sec</td>
<td>Good</td>
</tr>
<tr>
<td>Open Source</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoCoME</td>
<td>9,521</td>
<td>126 / 21</td>
<td>✔</td>
<td>✗</td>
<td>18</td>
<td>3 / 8 sec</td>
<td>Good</td>
</tr>
<tr>
<td>Open Source</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Additional projects for testing:

- “Negative Test” → openArchitectureWare (Open Source)
  - NOT component-based ✗
  - → Bad reverse engineering results ✗
- Scaling → Industrial Project (Commercial)
  - 646,527 LOC / 2,613 classes / 452 interfaces
  - Component-based, EJB
  - 3 iterations: 12 hours

Motivation / Introduction  ▸ Approach  ▸ Evaluation  ▸ Conclusion
Conclusion

Assumptions, Limitations, Findings
Limitations / Assumptions

- Java systems required
- Approach currently limited to component-based systems
  - System needs to be designed with components or with components in mind
  - Only little deviations tolerated (intentionally)
- Weights / weighting currently hard coded
Conclusion

- Approach capable to reverse engineer components and composite components
  - Strong component definition by [Szyperski2003]
  - Target model is suitable for use in model driven development

- Proposed metrics for evaluation of components wrt. their composition
  - Empirical evaluation of metrics and weights using real software systems
  - Respecting interdependencies of metrics

- Process for iterative and interactive reverse engineering of component-based software systems