Rule-Based Maintenance of Post-Requirements Traceability Relations

Patrick Mäder, Orlena Gotel and Ilka Philippow

16th IEEE International Requirements Engineering Conference
September 10, 2008
- Requirements traceability
- Our approach
- Traceability maintenance rules
- Prototype and preliminary validation
- Conclusion and future work
Traceability of development processes

- Allows to follow the development process and relates the intermediate products

requirements ← analysis ← design ← implementation

Each traceability relation represents an activity

Requirements traceability supports

- Analyzing the impact of changing requirements
- Verifying the implementation of requirements
- Program comprehension
- Supporting regression test
- Reuse …
Problems

- Necessity to manually create and update relations (Large numbers of relations even for small systems)
- Accurate set required for comprehensive results
- Insufficient method and tool support

Focus

- Maintenance of traceability relations during evolution and refinement of structural UML models
Requirements traceability

Our approach

Traceability maintenance rules

Prototype and preliminary validation

Conclusion and future work
Approach to tackle the problem of traceability decay due to evolution of related model elements

- **Stage 1**: Capturing changes to model elements and generating elementary change events
- **Stage 2**: Recognizing the wider development activity applied to the model element, as comprised several elementary change events
- **Stage 3**: Updating the traceability relations associated with the changed model element
Example: Convert attribute to a class

**Step 1:** Change of a traced use case
(Numbers on elements depict OUT-IN trace relations; relations backward from dependent to independent element)

**Step 2:** ADD a new class

**Step 3:** MOD - rename new class and add additional properties

**Step 4:** ADD association between class Order and AudioSystem

**Step 5:** DEL original attribute audioSystem

**Step 6:** Traceability links have been updated automatically
(2 incoming links on use case, 1 outgoing link on each class)
Main challenges

- Relate several elementary change events to one activity
- Recognize different orders of the same events as one activity
- Recognize different sequences of events as the same activity
- Requirements traceability
- Our approach
- Traceability maintenance rules
- Prototype and preliminary validation
- Conclusion and future work
Traceability maintenance rules

- Patterns of change activities that require traceability updates and directions to perform it
- Stored as XML rule-set

Status

- 21 rules capturing 37 development activities
- Supported elements: associations, inheritance, attributes, methods, classes, components, and packages
Rule application process

- on new event
  - try to complete OpenRules
    - browse RuleCatalog for rules with matching TriggerEvent
  - put event in EventCache
    - [on new OpenRule] search EventCache for additional matching events
      - [if rule is completed]
        - [on match] create new OpenRule
        - browse RuleCatalog for rules with matching TriggerEvent
        - [on new OpenRule] search EventCache for additional matching events
          - delete oldest event from EventCache and all OpenRules
          - do link update; delete events from all OpenRules and inactivate events in EventCache
Limitations

- Only predefined activities can be recognized
- Current rule-set is reusable and stable, but unlikely to reflect all possible activities – need for customization likely

Support for rule definition?

- Rule editor and validator available
- Requirements traceability
- Our approach
- Traceability maintenance rules
- **Prototype and preliminary validation**
- Conclusion and future work
traceMaintainer

- Prototype implementation to be able to validate the approach
- Tool-independent rule engine and link maintainer
- Special adapter per CASE tool necessary, available for Sparx Enterprise Architect, ARTiSAN Studio, and ToolNET
Two experiments using traceMaintainer to explore the following research questions

1. Is it capable of maintaining links at a level of accuracy comparable to manual maintenance?

2. How much manual effort can be saved by using automated maintenance in relation to the kind of modeling undertaken?
Experiment

- Two developers spent three hours on two sample projects respectively
- Analysis of correctness and completeness after refining an analysis model into the design model

Results

Result: **94–97% precision, 93–98% recall** between manual and automatically maintained set of traceability relations

<table>
<thead>
<tr>
<th></th>
<th>Changes</th>
<th>Activities</th>
<th>Modeling</th>
<th>Manual maintenance</th>
<th>trace Maintainer</th>
<th>Saved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimistic execution</td>
<td>127</td>
<td>35</td>
<td>64,5 min</td>
<td>62,0 min</td>
<td>18,2 min</td>
<td>71%</td>
</tr>
<tr>
<td>Pessimistic execution</td>
<td>176</td>
<td>49</td>
<td>82,2 min</td>
<td>115,8 min</td>
<td>18,2 min</td>
<td>84%</td>
</tr>
</tbody>
</table>
- Requirements traceability
- Our approach
- Traceability maintenance rules
- Prototype and preliminary validation
- Conclusion and future work
Conclusion

- Approach supports the automatic maintenance of traceability relations
- Preliminary results show the approach capable of reducing the effort in maintaining traceability quite dramatically and at a high level of precision

Future work

- Semi-automatically creation of new rules (rule recorder)
- Further integration into the development process
- Additional actions after activity recognition
Thank you. Patrick Mäder