Recovering Traceability Links between a Simple Natural Language Sentence and Source Code Using Domain Ontologies

Takashi Yoshikawa, Shinpei Hayashi, and Motoshi Saeki
Department of Computer Science
Tokyo Institute of Technology, Japan

1. Motivation


2. Problem and Our Solution

- Problems of two basic criteria for feature-location
  - Usage of word similarity: leads to false positives (A) / negatives (B)
  - Usage of method invocations: leads to false positives (C)
  - Combination of both approaches could not be a solution

Another criterion is needed
- representing the characteristics of the product’s problem domain

3. Methodology

The system overview

4. Case Study

- Evaluation target: JDraw 1.1.5
  - Having 1,450 methods
  - Picked up 7 sentences from JDraw’s manual on the Web
  - Prepared an ontology for painting tools (38 concepts, 45 relationships)

- Improved accuracy
  - (1st and 2nd cases) recalls increase (although precision a little decrease)
  - (3rd case) newly detected

- Our solution: using domain ontology
  - Ontology = binary relationships between concepts
  - Representing the domain knowledge of the target product family
  - e.g., the draw function concerns the color concept
  - Checking co-occurrence of relationships

5. Conclusion

Domain ontologies give us valuable guides for traceability-recovering and feature-location.

Troubleshooting
- Need to recover doc-to-code traceability links
  - for reducing maintenance costs
  - for easily reusing existing code

- Simple NL sentences as documents
  - some software products do not contain rich information
  - e.g., just feature names, agile software development
  - feature-location approach


Domain ontologies give us valuable guides

• Prepared an ontology for painting tools (38 concepts, 45 relationships)
• Picked up 7 sentences from JDraw’s manual on the Web
• Having 1,450 methods

7. “grayscaling” - - - - - -
6. “colour reduction” 0.74 0.95 0.83 0.74 0.95 0.83
5. “save JPEGs of configurable quality” 0.40 1.00
4. “image rotation” 1.00 0.35
3. “image rotation” 0.82 0.98
2. “plain, filled and gradient filled ovals” 0.82 0.98
1. “plain, filled and gradient filled rectangles” 0.83 0.94

5. Normalizing the whole score of invocations by the inclusion rate
- How many input words the candidate includes?

6. Emphasizing method invocations
- Choose the methods having the important roles

7. Extracting results
- The traversed set of methods are the candidates of feature-related code

Experimental Results

<table>
<thead>
<tr>
<th>Input sentences</th>
<th>Prec (w/ ontology)</th>
<th>Rec (w/ ontology)</th>
<th>F (w/ ontology)</th>
<th>Prec (w/o ontology)</th>
<th>Rec (w/o ontology)</th>
<th>F (w/o ontology)</th>
</tr>
</thead>
<tbody>
<tr>
<td>“plain, filled and gradient filled rectangles”</td>
<td>0.83</td>
<td>0.94</td>
<td>0.88</td>
<td>1.00</td>
<td>0.19</td>
<td>0.32</td>
</tr>
<tr>
<td>“plain, filled and gradient filled ovals”</td>
<td>0.82</td>
<td>0.98</td>
<td>0.89</td>
<td>1.00</td>
<td>0.21</td>
<td>0.35</td>
</tr>
<tr>
<td>“image rotation”</td>
<td>1.00</td>
<td>0.35</td>
<td>0.52</td>
<td>0.00</td>
<td>0.00</td>
<td>-</td>
</tr>
<tr>
<td>“image scaling”</td>
<td>0.22</td>
<td>0.68</td>
<td>0.33</td>
<td>1.00</td>
<td>0.58</td>
<td>0.73</td>
</tr>
<tr>
<td>“save JPEGs of configurable quality”</td>
<td>0.40</td>
<td>1.00</td>
<td>0.57</td>
<td>0.67</td>
<td>1.00</td>
<td>0.80</td>
</tr>
<tr>
<td>“colour reduction”</td>
<td>0.74</td>
<td>0.95</td>
<td>0.83</td>
<td>0.74</td>
<td>0.95</td>
<td>0.83</td>
</tr>
</tbody>
</table>

Picked up the results having the highest F-measure from the results ranked in the top 5