Exploiting Traceability Uncertainty between Architectural Models and Code

Achraf Ghabi & Alexander Egyed
achraf.ghabi@jku.at & alexander.egyed@jku.at

WICSA/ECSA 2012
Traceability

- **Informal definition:**
  The collection of traces which are documenting the relationship between two artifacts.

- Useful particularly at maintenance time

- In this work:
  - Two Artifacts: Model (solution elements) and Code (code elements)
  - Relationship: „is implemented by“

- Trace example:
  - Model element A is implemented by code element C
Model and Code: VoD Client

Code: classes

<table>
<thead>
<tr>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEGDecoder.Button</td>
</tr>
<tr>
<td>MPEGDecoder.DStore</td>
</tr>
<tr>
<td>MPEGDecoder.LFrame</td>
</tr>
<tr>
<td>MPEGDecoder.Movie</td>
</tr>
<tr>
<td>MPEGDecoder.Header</td>
</tr>
</tbody>
</table>
# Trace Matrix: Certainty

<table>
<thead>
<tr>
<th>Code\mode elements</th>
<th>play</th>
<th>playing</th>
<th>select</th>
<th>stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEGDecoder.Button</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>MPEGDecoder.DStore</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MPEGDecoder.LFrame</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>MPEGDecoder.Movie</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>MPEGDecoder.Header</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Trace: X

No-Trace: }
Traceability is difficult:

- Loosing some key developers
- Understanding of the entire code
- Forgotten details
- Incorrect recollection of facts
- Changes in model/code

→ Engineers might be uncertain about some trace relationships.
Goal

- Allow the engineer to express what she REALLY knows about a system
- Help engineer to
  - Detect incorrectness/inconsistency in her knowledge
  - Derive further traceability information based on her knowledge
Expressing Uncertainty

are implemented at least in

is implemented at most in

are implemented exactly in

Code Elements

Button

DStore

LFrame

Movie

Header

play

select

stop

playing
Footprint Graph

\{\text{stop}\} \text{ implAtMost} \ \{\text{LFrame, Movie}\}

Uncertain Trace

L, M

No-Trace

Code Elem. Group

Model Elem.

Code Elem.
Footprint Graph

\{ select, playing \} \implAtLeast \{ Button, LFrame \}
\{\text{play, playing}\} \implies \text{Exactly} \{\text{DStore, LFrame}\}

\begin{itemize}
  \item \text{D, L}
  \item \text{B, L}
  \item \text{D, L}
  \item \text{L, M}
  \item \text{B, L}
\end{itemize}

\begin{itemize}
  \item \text{play}
  \item \text{playing}
  \item \text{stop}
  \item \text{select}
\end{itemize}

\begin{itemize}
  \item \text{Button}
  \item \text{DStore}
  \item \text{LFrame}
  \item \text{Movie}
  \item \text{Header}
\end{itemize}

\begin{itemize}
  \item \text{select, playing}
  \item \text{play, playing}
  \item \text{select, playing}
  \item \text{play, playing}
\end{itemize}

\begin{itemize}
  \item \text{Uncertain Trace}
  \item \text{No-Trace}
\end{itemize}

\text{Code Elem. Group} \quad \text{Model Elem. Group}
Propagation Rules

- **D, L**: play
- **B, L**: playing
- **D, L**: select
- **L, M**: stop
- **B, L**: select

**Uncertain Trace**: Button (select, playing)
**Trace**: DStore (play, playing)
**No-Trace**: LFrame (select, playing)
**Uncertain Trace**: Movie (play, playing)

**Uncertain Trace**: Header (play, playing)

---

Code Elem. Group | Model Elem. Group

© 2012 Alexander Egyed, Achraf Ghabi
Result in Trace Matrix

- Filled the TM using uncertainties
- But not complete
- Correctness depends on the developer’s knowledge.

<table>
<thead>
<tr>
<th></th>
<th>Code</th>
<th>play</th>
<th>playing</th>
<th>select</th>
<th>stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>DStore</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LFrame</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Movie</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Trace Group

Trace

No-Trace
Further Input

- In big systems usually multiple developers will do the traceability.

→ Different understanding of the system.

- Another Engineer introduces this input in her description:

```
{stop} implAtLeast {DStore}
```
Knowledge Conflict

\{\text{stop}\} \implies \text{AtLeast} \{\text{DStore}\}

\begin{align*}
\text{D, L} & \quad \text{play} \\
\text{D, L} & \quad \text{playing} \\
\text{L, M} & \quad \text{stop} \\
\text{B, L} & \quad \text{select}
\end{align*}

- Button
- DStore
- LFrame
- Movie
- Header

Uncertain Trace

Trace

No-Trace

Code Elem. Group

Model Elem.

Model Elem. Group
Correctness Constraint (1)

- A code element cannot be implementing and not implementing a model element at the same time.

- A code element cannot be implementing and not implementing a model element at the same time.
Correctness Constraints (2)

- Every group must have at least one model element.

```
B, C  | play
B, C  | playing
C, D  | stop
A, C  | select

Button
DStore
LFrame
Movie
Header

Code Elem.        Model Elem. Group
```
Validation

- Correctness, scalability
- Evaluation of all pair wise combinations of the four types of input (implAtLeast, implAtMost, implExactly, implNot)
- 4 case study systems: ArgoUML, Siemens Route Planning, Video on demand client, and USC Inter-Library Loan
Correctness

- In most of the cases incorrect/conflicting input is detected
- The more input the more likely an incorrectness would be detected
- Incorrectness is not detected if an engineer has an incorrect but consistent understanding of the model-to-code mapping → unlikely when multiple engineers are working together
Scalability

- The growth of the footprint graph is polynomial with the size of the model and code
- Size of graph = \( \#C + \#M + \text{input} \times (\#C + \#M) \)
- Largest study case 30,000 nodes (ArgoUML) required less than a minute to convert the input into the footprint graph and propagate the rules for 38 ME
Summary

- Approach proposing how to describe engineers’ knowledge about traceability
- Automatic detection of incorrect/inconsistent knowledge
- Automatically derive further knowledge
- Applicable to all kinds of models that are implemented in the code
Future Work

- Extend the input by adding some kind of weighting to uncertainty constructs
- Use incremental reasoning
- Apply the same technique on model-to-model traceability
- Conduct experiments on industrial projects
Discussion

Achraf Ghabi (achraf.ghabi@jku.at)
Alexander Egyed (alexander.egyed@jku.at)

Institute for Systems Engineering and Automation