Improving Automatic Model Creation using Ontologies

Sven J. Körner, Tom Gelhausen
Institute for Program Structures and Data Organization
Chair of Prof. Dr. Walter F. Tichy
University of Karlsruhe, Germany
Ontologies Improve Automatic Model Creation

- Automatic model creation is possible through adding semantics
- Automatic model creation has potential benefit from textual specifications (10000+ pages)
- Considers (implicit) semantics; embeds common sense
- Moves manual step to automatic/mechanical steps
  - Less-error prone model creation then humans
  - Repeatable
  - Deterministic
Software Development Today

- How the Customer explained it
- How the Project Leader understood it
- How the Analyst designed it
- How the Programmer wrote it
- How the Business Consultant described it
- How the Project was documented
- What Operations installed
- How the Customer was billed
- How it was supported
- What the Customer really needed
Requirements Engineering – Tool Support

• Supported
  • Customer specification
  • Requirements/functional specification
  • Code stub-generation from models

• Problem
  • Model creation from specification
  • Manual task to create models
Automatic Model Creation – an Overview

1. **Spec**
2. Add semantics
3. Transform
4. Code (Specs + Annotations)
5. Process semantics
6. Internal Model
“The game of chess is played between two opponents. The player with the white pieces commences the game”
[ The game of chess is played \textit{ACT} between two opponents \textit{AG}.

[ The player \textit{AG} with the white pieces commences \textit{ACT} the game ].
Annotated Text in the SENSE/SAL$_E$ System

[ #The game_of_chess|PAT #is played|ACT #between *two opponents|AG ].

[ [ The ^player|POSS #with #the $white pieces|HAB ]|AG commences|ACT #the game|PAT ].
Limitations of Current Solution

- Manual annotations
- Model creation needs common sense
  - Conceptual relatedness

```
GameOfChess -isPlayed
\   \           \                 \ 2
1                2
```

- Choosing the right kind of model

```
Opponents
\   \\
\     \\
\      \\
\       \\
\        \\
\         \\
\          \\
\           \\
\            \\
\             \\
\              \\
\               \\
\                \\
\                 \\
\                  \\
\                   \\
\                    \\
\                     \\
\                      \\
```

```
Player
\   \\
\     \\
\      \\
\       \\
\        \\
\          \\
\           \\
\            \\
```

+commence(in GameOfChess)
Related Work

• Gelhausen:
  • Automatic model creation
  • SENSE/SAL$_E$

• Meystre/Haug
  • Automatic annotation and natural language processing
  • Specialized ontologies work in small domains

• Bethard et al.
  • Timeline extraction
  • Timely dependencies of textual artifacts

• Liu/Lieberman
  • High coherence between natural language and code
  • NLP with students
  • ConceptNet (special trained ontology)
Improve Model Creation with Ontology

Add semantics

Ontology

Build Graph

Transform

Graph

"Code"

UNIFIED MODELING LANGUAGE

Text
Architecture Including the Ontology

1. Text
2. Annotated SAL\textsubscript{E}-Text
3. Graph
4. Rules
5. Ontology
   - List of AGs (player, opponent)
   - List of ACTs (plays, ...)
   - ...
6. UML

Sven J. Körner & Tom Gelhausen, SEKE 2008
• Are „Opponents“ and „Player“ related?

```
predicate opponents:
isa: IrreflexiveBinaryPredicate
     SymmetricBinaryPredicate
     CoexistingObjectsPredicate
     (argIsa opponents 1 Agent-Generic)
     (argIsa opponents 2 Agent-Generic)
     (opponents AGENT1 AGENT2)

Collection Player:
isa: Agent-Generic
```
Decreasing Complexity of UML models with Ontologies

- Earlier UML model

<table>
<thead>
<tr>
<th>GameOfChess</th>
<th>-isPlayed</th>
<th>Opponents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

- New, improved UML model using ontologies

<table>
<thead>
<tr>
<th>GameOfChess</th>
<th>-isPlayed</th>
<th>Player</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>-isOpponent</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>+commence(in GameOfChess)</td>
</tr>
</tbody>
</table>

+commence(in GameOfChess)
# Thematic Relations Matched to UML

<table>
<thead>
<tr>
<th>Thematic Relation</th>
<th>Explanation</th>
<th>UML model element</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG</td>
<td>The acting person or thing executing the action</td>
<td>Class- Role, or Instance</td>
</tr>
<tr>
<td>ACT</td>
<td>The action, executed by a person or thing</td>
<td>Method, State- (Transition), or Relation</td>
</tr>
<tr>
<td>PAT</td>
<td>Person or thing affected by the action or on which the action is being performed</td>
<td>Class, Role, or Instance</td>
</tr>
<tr>
<td>HAB</td>
<td>Possession or belonging; person or thing being received or passed on by person or thing</td>
<td>Class, Role, or Instance</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
UML Representations for ACT

- **instantaneous** modeled as state-(transition)
- **eternal** modeled as relation
- **duration** modeled as method

Transition `ACT` modeled as state transition.

Class 1

Class 2

<table>
<thead>
<tr>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>method <code>ACT1()</code></td>
</tr>
<tr>
<td>method <code>ACT2()</code></td>
</tr>
</tbody>
</table>
ACT as State (-Transition)

- Instantanious
- Modeled as state-(transition)

Game On-Going → Win → Game Won

Collection: Winning
isa: ConflictEventStatus
AtemporalNecessarilyEssential
CollectionType
AtemporalThing
genls: AtemporalThing
ACT as Relation

eternal

modeled as relation

Help Desk

Operator

uses

Predicate: `usesIn`
isa: `TernaryPredicate`
arg1Isa: `Agent-PartiallyTangible`
arg2Isa: `PartiallyTangible`
arg3Isa: `Action`
(argIsa usesIn 1 Agent-PartiallyTangible)
(argIsa usesIn 2 PartiallyTangible)
(argIsa usesIn 3 Action)
ACT as Method

duration
modeled as method

Chesspiece

<table>
<thead>
<tr>
<th>move()</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack()</td>
</tr>
</tbody>
</table>

Collection: CausingAnotherObjects
TranslationalMotion
isa: EventOrRoleConcept
FirstOrderCollection
genls: ActionOnObject
Movement-TranslationEvent
TemporalThing
TemporallyExistingThing
TemporallyExtendedThing

Collection: Movement-TranslationEvent
isa: EventOrRoleConcept
FirstOrderCollection
genls: TemporalThing
TemporallyExistingThing
TemporallyExtendedThing
Conclusion

• Automatic modeling improved through more precise (more compact) modeling
• Our approach moves software process development control to earlier stages
• Lessons learned
  • Ontology-coverage has blind spots
  • Ontology has not been improved
  • Could not distinguish: „I eat fish“ vs „I am eating fish“: Determination vs Action
Future Work

• Annotation
  • Working on tool for annotation support (Eclipse PlugIn)

• Evaluation
  • Using other specifications then FIDE
  • Comparing RCyc ontology to other ontologies

• Pre-Processing
  • Skip non-functional aspects from specifications
  • Find and omit „Why?“-clauses
Questions?