STRIPS Planning as an Approach to Automated Transformations of Object Oriented Models

Oldřich Nouza
Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering
What is a Model Transformation?

• *Model transformation* = an automatic generation of a target model from a source model, according to a transformation definition

• *Transformation definition* = a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language.

• *Transformation rule* = a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language.
Classification of Transformations

• Languages in which the source and targets models are expressed
  – *Endogenous* – source and target models are expressed in the same language (e.g. normalization)
  – *Exogenous* – source and target models are expressed in different languages (e.g. reverse engineering)

• Difference of abstraction level
  – *Horizontal* – source and target models have the same level of abstraction (e.g. refactoring)
  – *Vertical* – source and target models have a different level of abstraction (e.g. refinement)
Refactoring

• An improvement of software system without changing its behavior
• For the same input, the refactored software must return the same output as the original software
• It is possible to prove that
  – each refactoring is composeable from a limited sequence of applications of primitive refactoring operations,
  – the set of these operations is limited, complete, and minimal
  – object normalization (based on Scott Ambler’s approach) and application of some design patterns are a special cases of refactoring
Actual State of Model Transformation Automation Techniques

• There exist satisfiable solutions of automated transformations of models to text

• The same cannot be said about transformations of models to models
  – Many approaches, but little practical experience
  – Still in a phase of research

• Actual CASE tools implementations feature little reusability
  – For each transformation type it is necessary to define a new transformation rule.

• Goal: A new automated transformations engine with a limited complete set of transformation rules.
New Transformation Engine – Requirements

• Support of refactoring, design patterns application, and object normalization

• Input
  – source model
  – condition that must be satisfied by the target model

• Output
  – either the target model satisfying the input condition,
  – or an information that no such model exists

• Source and target model behavioral equivalence

• Reusability, full automation without need of new transformation rules definition
Solution

- Application of STRIPS planning
- Result: SBAT (STRIPS Based Automated Transformation) engine
STRIPS Planning – Introduction (1)

• STRIPS (Standford Research Institute Problem Solver) – one of the first planning systems used in AI
  – uses a first-order logic calculus

• Input
  – initial state $I$ (a limited set of true formulas)
  – goal condition $G$
  – set of operators $O$ – describe possible transitions between states

• Output
  – either final state $s_F$ that satisfies the goal condition $G$,
  – or information that no such state exists
• Operator - an ordered triplet \((P, A, D)\)
  - \(P\) – “pre-condition” – must be satisfied before the operator application
  - \(A\) – “add-effects” – formulas that become true after the operator application
  - \(D\) – “delete-effects” – formulas that become false after the operator application

• Example – movement from position \(a\) to position \(b\)
  - state \(s = [\text{on}(a)]\)
  - operator \(\text{move}(x, y)\):
    • \(P = \text{on}(x) \land \neg \text{on}(y)\)
    • \(A = [\text{on}(y)]\)
    • \(D = [\text{on}(x)]\)
  - application \(\text{move}'(a, b, s)\) changes the state \(s\) to \(s' = [\text{on}(b)]\)
STRIPS Planning – State Transitions

- **Initial State**: $S_0$
- **Actual State**: $S_1$
- **Already Generated State**: $S_2$
- **Not Yet Generated State**: $S_3$
- **Goal State**: $S_F$

States connected by arrows represent transitions in the planning process.
# Refactoring and STRIPS Planning analogy

<table>
<thead>
<tr>
<th>Refactoring</th>
<th>STRIPS planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>source model</td>
<td>initial state $I$</td>
</tr>
<tr>
<td>target model condition</td>
<td>goal state condition $G$</td>
</tr>
<tr>
<td>target model</td>
<td>goal state $s_F$</td>
</tr>
<tr>
<td>set of primitive refactoring operations</td>
<td>operator set</td>
</tr>
<tr>
<td>sequence of applications of primitive refactoring operations</td>
<td>sequence of operator applications</td>
</tr>
</tbody>
</table>
Object Model View

• Model – a state space
  – Model state – based on a simplified UML class model
    • classes
    • attributes and methods assignment to classes
    • generalization
    • data type
    • message sending
  – State transitions – primitive refactoring operations
  – Special classes
    • Object – a parent of all other classes in all model states
    • Client – a client singleton class not contained in any model state, but can send messages to objects of classes in any model state
## Selected Predicates Describing Model States

<table>
<thead>
<tr>
<th>Formal expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inModel(c)</code></td>
<td>The class $c$ is contained in the model</td>
</tr>
<tr>
<td><code>outOfModel(c)</code></td>
<td>The class $c$ is not contained in the model</td>
</tr>
<tr>
<td><code>attrInClass(c, a)</code></td>
<td>The class $c$ has the attribute $a$</td>
</tr>
<tr>
<td><code>superClass(c, d)</code></td>
<td>The class $c$ is a superclass of the class $d$</td>
</tr>
<tr>
<td><code>hasType(a, t)</code></td>
<td>The attribute $a$ is of the type $t$</td>
</tr>
</tbody>
</table>
STRIPS Operator – Example

• Add class $x$ into the model
  – Declaration: $addClass(x)$
  – Pre-condition: $outOfModel(x)$
  – Add-effects: $inModel(x)$
  – Delete-effects: $outOfModel(x)$

• Operator application – transition into $m_s'$ after adding $c$ class into model in $m_s$ state

$$m_s' = addClass'(c, m_s) = m_s \cup [inModel(c)] - [outOfModel(c)]$$

if $outOfModel(c)$ condition is satisfied in $m_s$
• Goal: Transform the model above into the form fulfilling the *Composite* design pattern requirements.
Example – Initial State

- $I = [\text{inModel}(\text{Folder}), \text{inModel}(\text{File}), \text{inModel}(\text{String}), \text{outOfModel}(\text{Element}), \text{attrInClass}(\text{name}, \text{Folder}), \text{attrInClass}(\text{owner}, \text{Folder}), \text{attrInClass}(\text{name}, \text{File}), \text{attrInClass}(\text{owner}, \text{File}), \text{attrOutOfClass}(\text{name}, \text{Element}), \text{attrOutOfClass}(\text{owner}, \text{Element}), \text{superClass}(\text{Object, File}), \text{superClass}(\text{Object, Folder}), \text{hasType}(\text{owner}, \text{Folder}), \text{hasType}(\text{name}, \text{String}), \text{sending}(\text{Client, main, Folder, name}), \text{sending}(\text{Client, main, Folder, owner}), \text{sending}(\text{Client, main, File, name}), \text{sending}(\text{Client, main, File, owner})]$
Example – Goal State Condition

• $G = \text{inModel}(\text{Element}) \land \text{attrInClass}(\text{name}, \text{Element}) \land \text{attrInClass}(\text{owner}, \text{Element}) \land \text{superClass}(\text{Element}, \text{File}) \land \text{superClass}(\text{Element}, \text{Folder})$
Example – STRIPS Operators Application (1)

\[ m_0 = I \]
Example – STRIPS Operators Application (2)

\[ m_1 = addClass'(Element, m_0) \]
Example – STRIPS Operators Application (3)

\[ m_2 = \text{changeSup}'(\text{File}, \text{Object}, \text{Element}, m_1) \]
Example – STRIPS Operators Application (4)

\[ m_3 = changeSup'(Folder, Object, Element, m_2) \]
Example – STRIPS Operators Application (5)

\[ m_4 = \text{attrUp}'(\text{name, Element, (File, Folder), } m_3) \]
Example – STRIPS Operators Application (6)

\[ m_5 = \text{attrUp}'(\text{owner}, \text{Element}, (\text{File}, \text{Folder}), m_4) \]
Example – Goal State

$m_5$ satisfies $G$, so it is a goal state
STRIPS Application – Example – Goal State

- $s_F = [\text{inModel}(\text{Folder}), \text{inModel}(\text{File}), \text{inModel}(\text{String}), \\
\text{inModel}(\text{Element}), \text{attrInClass}(\text{name}, \text{Element}), \\
\text{attrInClass}(\text{owner}, \text{Element}), \text{attrOutOfClass}(\text{name}, \\
\text{Folder}), \text{attrOutOfClass}(\text{owner}, \text{Folder}), \\
\text{attrOutOfClass}(\text{name}, \text{File}), \text{attrOutOfClass}(\text{owner}, \text{File}), \\
\text{superClass}(\text{Element}, \text{File}), \text{superClass}(\text{Element}, \text{Folder}), \\
\text{superClass}(\text{Object}, \text{Element}), \text{hasType}(\text{owner}, \text{Folder}), \\
\text{hasType}(\text{name}, \text{String}), \text{sending}(\text{Client}, \text{main}, \text{Folder}, \\
\text{name}), \text{sending}(\text{Client}, \text{main}, \text{Folder}, \text{owner}), \\
\text{sending}(\text{Client}, \text{main}, \text{File}, \text{name}), \text{sending}(\text{Client}, \text{main}, \\
\text{File}, \text{owner})]$
Conclusion

• SBAT transformation engine
  – The next step towards the full automation of object oriented model transformations
    • More effective schedule of human resources on software projects, consequently improvement of software quality
  – Theoretical background for research activities in the area of model transformations

• Future work
  – Model state space optimization
    • Reduction of generated states count
  – Implementation of SBAT in CASE tools


References (2)


References (3)


Thank You for Your Attention

Oldřich Nouza

Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering
Department of Software Engineering in Economy

nouza@fjfi.cvut.cz