Non-Monotonic Model Completion in Web Application Engineering

Australian Software Engineering Conference (ASWEC) 2010
8 April 2010

Jevon Wright
j.m.wright@massey.ac.nz

Jens Dietrich
j.b.dietrich@massey.ac.nz

Massey University
Palmerston North
New Zealand
Background

- Developing a modelling language for RIAs
  - RIAs are very complex
  - No existing language succeeds
  - Previous work (WISE 2008) found 59 core requirements for modelling RIAs

- Internet Application Modelling Language
  - Started research in Feb 2007
  - Follows MDE approach
Background

- Key challenge: balancing the level of detail
- Too much abstractness
  - A rigid approach that cannot adapt
- Too much flexibility
  - Large model instances become unmaintainable
  - Lots of scaffolding required
Background

- Web Frameworks
  - Ruby on Rails, Symfony, ...
- Add abstractness while keeping flexibility
  - Adds common scaffolding
  - According to documented conventions
  - Can be overridden if necessary
- We apply this to model-driven development
Model Completion

- Developer designs an initial model
- We complete the model based on sensible defaults
  - "Intended" model
- Developer then refines their initial model
  - Or modifies the completed model
Model Completion

- Example: A boolean property
  1. We want to edit it with a form
  2. Normally edited by a checkbox
  3. Model completion adds a checkbox and related scaffolding

- Can achieve with normal inference rules
  - Define model completion as a rule program
  - Many rule engines for implementation
Non-Monotonicity

- Usually, we complete the model using incomplete knowledge
- Example: A boolean property
  1. Normally edited by a checkbox
  2. Developer wants it to be a drop-down (yes/no)
  3. Developer adds a drop-down editor
  4. Rule does not fire; checkbox not created

\[
\begin{align*}
\text{IF} & \quad (\text{there exists a boolean property}) \\
\text{AND} & \quad (\text{there does not exist an editor for it}) \\
\text{THEN} & \quad (\text{create a checkbox editor for it})
\end{align*}
\]
Model Completion

- What is a model?
  - A simplified abstraction of reality [4]
  - We define it as a set of model artefacts
  - Can define the universe of all possible models
  - We can restrict these models by defining the *meta-model* $S$
- Define model completion as a function $C(X)$ operating on a model $X$
Model Completion

- Function requirements
- Extensive
  - Must not retract any information from the base model
- Idempotent
  - A completed model is complete
- Non-monotonic
  - A more *refined* base model may change the completed model

\[ X \subseteq C(X) \]

\[ C(X) = C(C(X)) \]

\[ X \subseteq Y \not\implies C(X) \subseteq C(Y) \]
From Models to Logic

- Model artefacts $\rightarrow$ terms
- Artefacts in base model $\rightarrow$ constant terms
- Properties and relationships $\rightarrow$ predicates
- Creation of new artefacts $\rightarrow$ functions
- Type information $\rightarrow$ unary predicates
**Rules**

- **Base model artefact**

\[
\begin{align*}
\Phi_1 &= \text{property}(a) \\
\Phi_2 &= \text{property}(x) \land \neg \exists y : \text{editor}(y) \land \text{editorFor}(x, y) \\
&\quad \rightarrow \text{checkbox}(\text{newCheckbox}(x)) \\
&\quad \land \text{editorFor}(x, \text{newCheckbox}(x)) \\
\Phi_3 &= \text{checkbox}(x) \rightarrow \text{editor}(x) \\
\Phi_4 &= \text{dropdown}(x) \rightarrow \text{editor}(x)
\end{align*}
\]

- **Non existence of artefact**

- **Use factory functions to create new artefact**

- **Additional rules represent type reasoning**
Rules

- Factory functions are injective
  - `newCheckbox(x)` creates a unique new element

- Terms stratification
  - Terms are associated with a rank $\geq 0$
  - If $x$ is term of rank $N$, then `newCheckbox(x)` has rank $N+1$

- Constants (base model elements)
  - Have rank 0
Reasoning

- Apply rules in steps (parameterise rules with rank)
- Each step can only see elements with rank \( \leq N \)
- New model elements are rank \( N+1 \)
- Existential quantifier only applies to rank \( \leq N \)
- Only consider (logic) model generated from base model elements and factory functions (Herbrand model)
- Safeguards application: rules applied later cannot undermine rules applied earlier
Theoretical Aspects

- Classical reasoning (Tarski) is based on all models
  - $A$ is in $C(X)$ if $A$ is valid in all models of $X$
- Model selection is key idea of NMR
  - $A$ is in $C(X)$ if $A$ is valid in selected models of $X$
- Reasoning based on distinguished intended models
  - Examples: minimal and stable models
- Captures the intention of model completion
  - Try to formalise the notion of the model intended by the designer
Visualisation: Step 0

- What might model completion look like?
  - Example: Synchronising a database object 'Student' with an editable form
  - 6 elements
Visualisation: Step 1

Create text fields: + 3 elements
Visualisation: Step 2

Connect with synchronisation wires: + 3 elements
Add events, operations, and properties: + 18 elements
Connect events with operations, using actions: + 6 elements
Add parameters to actions: +6 elements
Create contents of operations: + 84 elements
Visualisation: Step 6

- **No new elements are being created**
  - Model completion may stop

- **Final model is 126 elements**
  - +2000% elements
Implementation

- Implemented with a commercial rule engine
  - JBoss Rules (Drools)
  - BSD/MIT-esque license
  - Integrates well with EMF Models

```java
rule "Example rule"
  when
    p : BooleanProperty ()
      not ( Editor ( for == p ) )
  then
    Checkbox c = handler.generatedCheckbox(p);
    handler.setFor(c, p);
    cache.add(c, drools);
end
```
Implementation

- **Problem:** By default, rule engines execute all rules and facts at once.
- **Solution:** An "insertion cache"

1. Model elements are inserted as facts into the working memory.
2. Rules create new elements, which are added to the *insertion cache*.
3. Once complete, cache elements are inserted as new facts.
4. The insertion cache is cleared, and the rules re-evaluated.
Implementation

- Normally, injective NMR is fragile
  - Results depend on rule execution order
  - Due to injective nature
  - e.g. If A creates B but B prevents C creating D, then running A or C first results in different outcomes

- Our approach also prevents this
Implementation

- **Problem:** Model completion could loop forever
  - A → B → C → A → B → ...
  - Depends on the rules
  - No general way to detect: halting problem
- **Solution:** An iteration limit $k$
  - Apply the given rules against a large suite of sample models to find limit
  - Warn developer if limit is hit at runtime
Test Suite

- This investigation only makes sense if we have a wide range of non-trivial models
- Iterative, test-driven development of IAML
  - Each new feature has test models
  - Wide range of uses
    - Check for conflicts with model completion
    - Develop code generation templates
    - Example models as documentation
    - ...
  - We use these test models as our input
Test Suite

- Some metrics of the test models
- 110 models used in ASWEC paper

<table>
<thead>
<tr>
<th>Metric</th>
<th>Minimum</th>
<th>Median</th>
<th>Mean</th>
<th>Maximum</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initial</td>
<td>Final</td>
<td>Initial</td>
<td>Final</td>
<td>Initial</td>
</tr>
<tr>
<td>Elements</td>
<td>2</td>
<td>3</td>
<td>11.5</td>
<td>107</td>
<td>15.1</td>
</tr>
<tr>
<td>Attributes</td>
<td>9</td>
<td>9</td>
<td>48</td>
<td>201</td>
<td>58.8</td>
</tr>
<tr>
<td>Non-default Attributes</td>
<td>4</td>
<td>6</td>
<td>23</td>
<td>166</td>
<td>28.0</td>
</tr>
<tr>
<td>Distinct Attribute Values</td>
<td>6</td>
<td>8</td>
<td>24</td>
<td>135</td>
<td>29.3</td>
</tr>
<tr>
<td>References</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>164</td>
<td>16.5</td>
</tr>
<tr>
<td>Children</td>
<td>1</td>
<td>2</td>
<td>10.5</td>
<td>106</td>
<td>14.1</td>
</tr>
<tr>
<td>Distinct Types</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>19</td>
<td>8.9</td>
</tr>
<tr>
<td>Min Degree (References and Children)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.3</td>
</tr>
<tr>
<td>Max Degree (References and Children)</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>5.0</td>
</tr>
<tr>
<td>Children Depth</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>3.2</td>
</tr>
<tr>
<td>Cycles (References and Children)</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3.0</td>
</tr>
<tr>
<td>Diameter (References and Children)</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>12</td>
<td>5.6</td>
</tr>
<tr>
<td>Average Completion Time (ms)</td>
<td>0.0</td>
<td>8.6</td>
<td>117.6</td>
<td>5,213.6</td>
<td>696.8</td>
</tr>
</tbody>
</table>

TABLE I
Selected model metrics of initial and completed test models (n = 110)
Results

- +1082% elements on average
  (+3825%: 8 → 314 elements)
Results

- At most, 9 steps required to complete model
- We conservatively limit $k$ to 20
Overriding Model Completion

- Important part of model completion
- We need to allow the developer to modify generated elements
- Current approach
  - Elements are 'generated by' others
  - Add 'overridden' flag

```java
rule "Example rule"
    when
        ...
        eval ( handler.veto( p ))
    then
        Checkbox c = handler.generatedCheckbox(p);
        ...
```
CASE Tool Implementation

- Tools
  - Infer [contained] generated elements
  - Remove [contained] generated elements
  - Infer and record the source rule

- Graphical editor appearance
  - `/` signifies generated (c.f. UML 'derived')
  - Bold signifies overridden (c.f. about:config)
Unanswered Questions

- **How to document completion?**
  - We can't force developers to read source

- **Current work**
  - "Modeldoc": documentation for MDA
  - Adds Javadoc-style annotations to model completion rules
Unanswered Questions

- How to make sure rules don't hit limit $k$?
  - In worst case, solving the halting problem
- Usability of model completion
  - Probably dependent on tool support
  - How often do developers override?
  - Extending or overriding existing rules
Conclusion

- IAML 0.4.4 available online
  - Eclipse-based CASE tool
  - Free and open source (EPL)

Questions?

j.m.wright@massey.ac.nz
http://openiaml.org