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

Outline

1. Background
2. What is Model Completion?
3. Rules and Reasoning
4. Example
5. Implementation
6. Results
7. Unanswered Questions

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

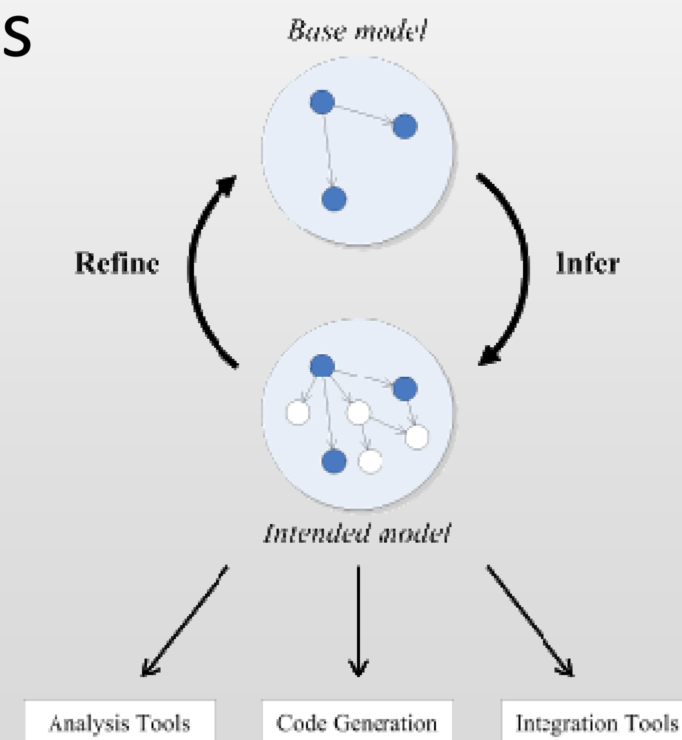
- Key challenge: balancing the level of detail
- Too much abstractness
  - A rigid approach that cannot adapt
- Too much flexibility
  - Large model instances become unmaintainable
  - Lots of scaffolding required

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

IF	*(there exists a boolean property)*
AND	*(there does not exist an editor for it)*
THEN	*(create a checkbox editor for it)*

Model Completion

- What is a model?
  - A simplified abstraction of reality [4]
  - We define it as a set of model artefacts M
  - Can define the universe of all possible models $model \in 2^M$
  - We can restrict these models by defining the *meta-model* S $\mathcal{S} \subseteq 2^M$
- Define model completion as a function C(X) operating on a model X $C : \mathcal{S} \rightarrow \mathcal{S}$ $X \subseteq \mathcal{S}$

Model Completion

- Function requirements
- Extensive $X \subseteq C(X)$
  - Must not retract any information from the base model
- Idempotent $C(X) = C(C(X))$
  - A completed model is complete
- Non-monotonic $X \subseteq Y \nRightarrow C(X) \subseteq C(Y)$
  - A more *refined* base model may change the completed model

From Models to Logic

- Model artefacts → terms
- Artefacts in base model → constant terms
- Properties and relationships → predicates
- Creation of new artefacts → functions
- Type information → unary predicates

Rules

Base model artefact

Non existence of artefact

$$
\begin{aligned}
\Phi_1 &= property(\mathtt{a}) \\
\Phi_2 &= property(x) \wedge \neg\exists y : editor(y) \wedge editorFor(x,y) \\
&\quad \rightarrow checkbox(newCheckbox(x)) \\
&\qquad \wedge editorFor(x, newCheckbox(x)) \\
\Phi_3 &= checkbox(x) \rightarrow editor(x) \\
\Phi_4 &= dropdown(x) \rightarrow editor(x)
\end{aligned}
$$

Additional rules represent type reasoning

Use factory functions to create new artefact

Rules

- Factory functions are injective
  - *newCheckbox(x)* creates a unique new element
- Terms stratification
  - Terms are associated with a rank ≥ 0
  - If *x* is term of rank N, then *newCheckbox(x)* has rank N+1
- Constants (base model elements)
  - Have rank 0

Reasoning

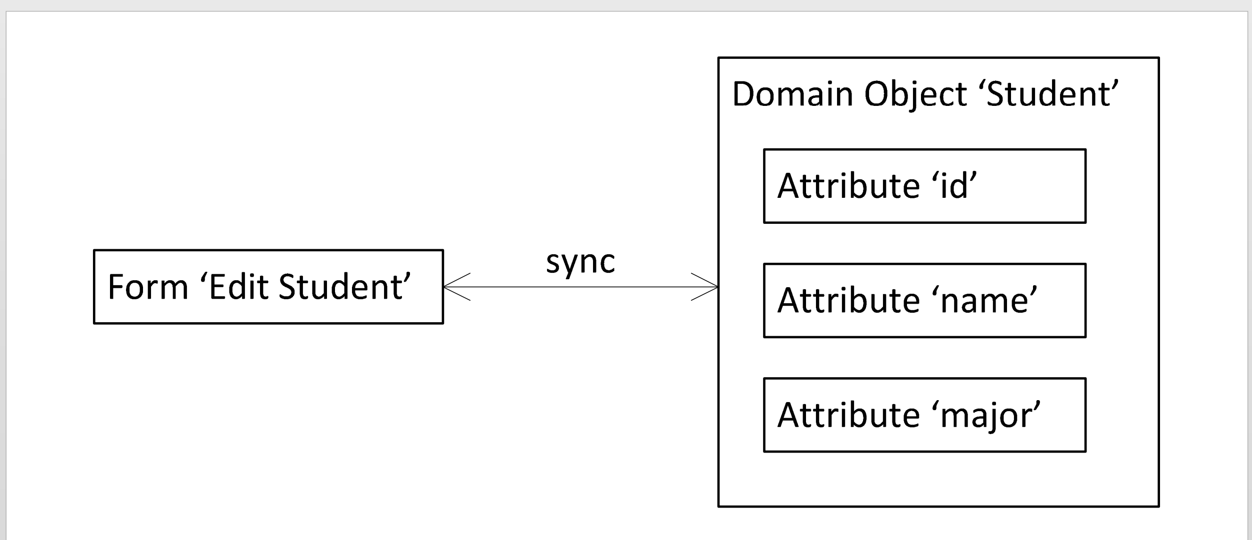
- Apply rules in steps (parameterise rules with rank)
- Each step can only see elements with rank $\leq N$
- New model elements are rank $N+1$
- Existential quantifier only applies to rank $\leq N$
- Only consider (logic) model generated from base model elements and factory functions (Herbrand model)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

Theoretical Aspects

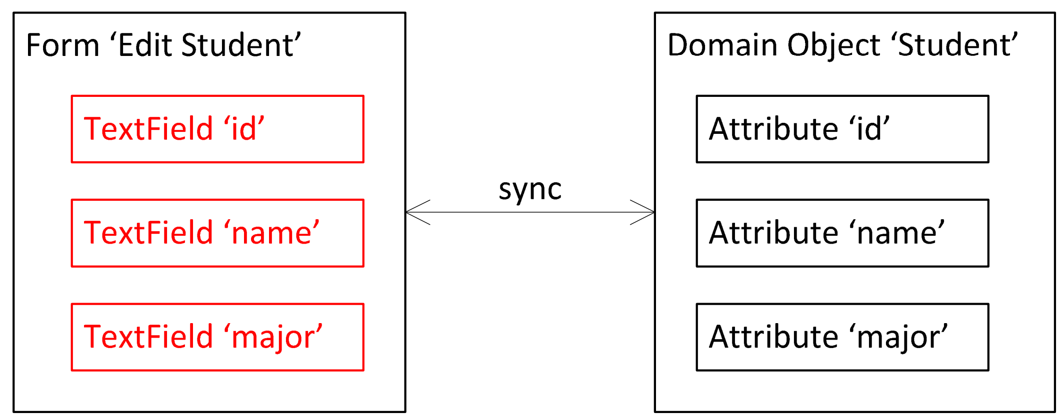
- Classical reasoning (Tarski) is based on all models
  - A is in C(X) if A is valid in all models of X
- Model selection is key idea of NMR
  - A is in C(X) if A is valid in selected models of X
- Reasoning based on distinguished intended models
  - Examples: minimal and stable models
- Captures the 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

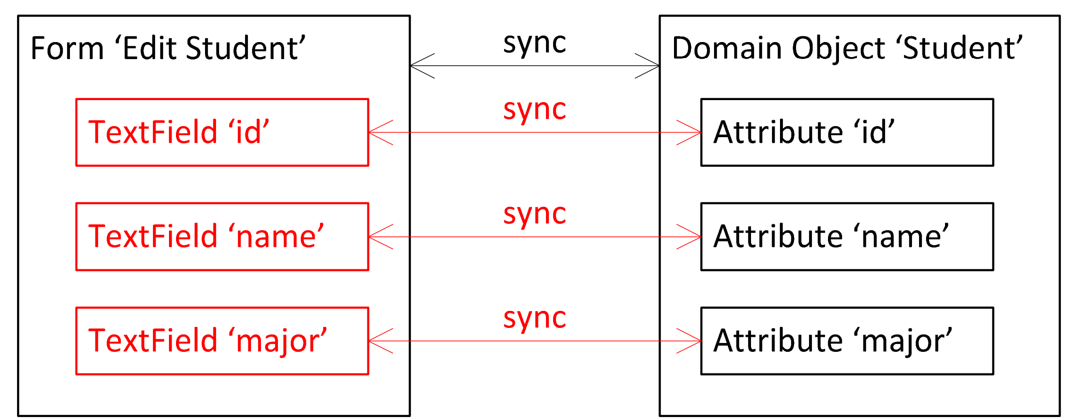


Visualisation: Step 1



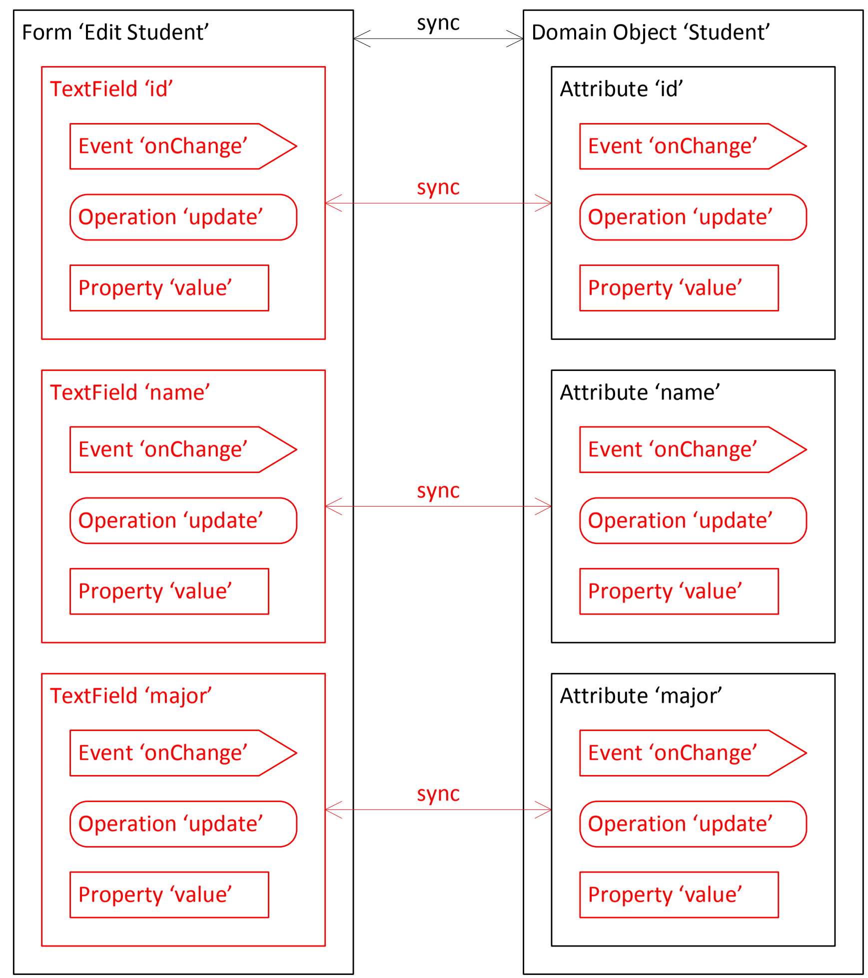
Create text fields: + 3 elements

Visualisation: Step 2



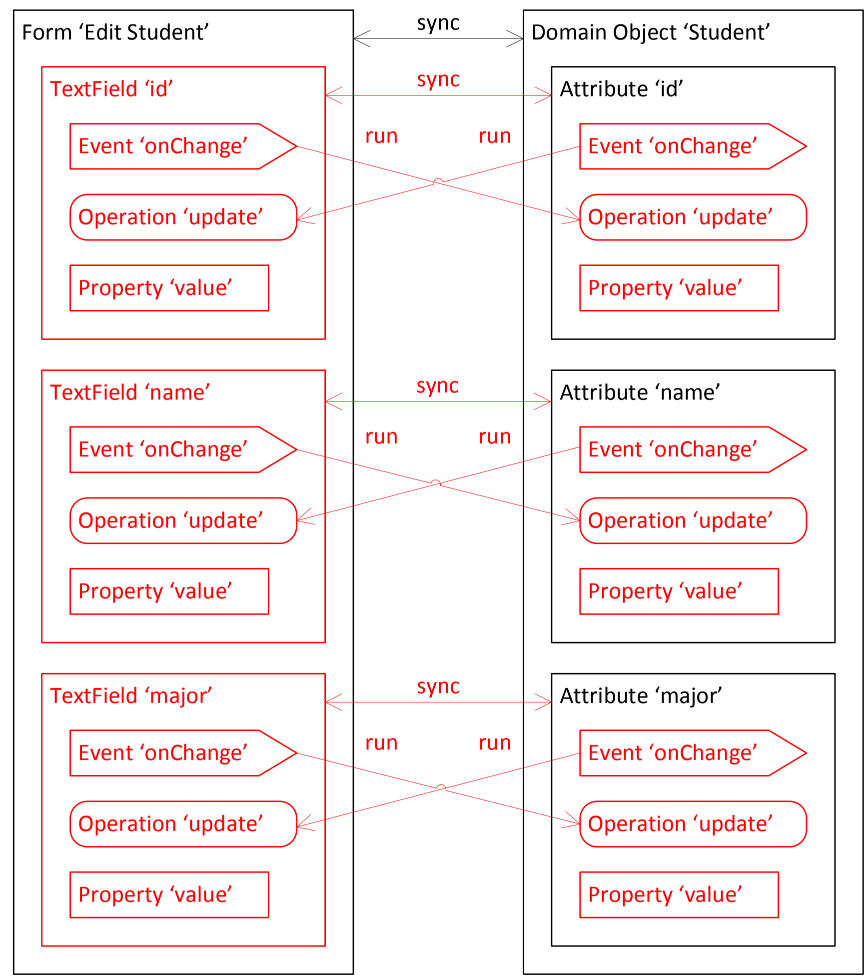
Connect with synchronisation wires: + 3 elements

Visualisation: Step 3



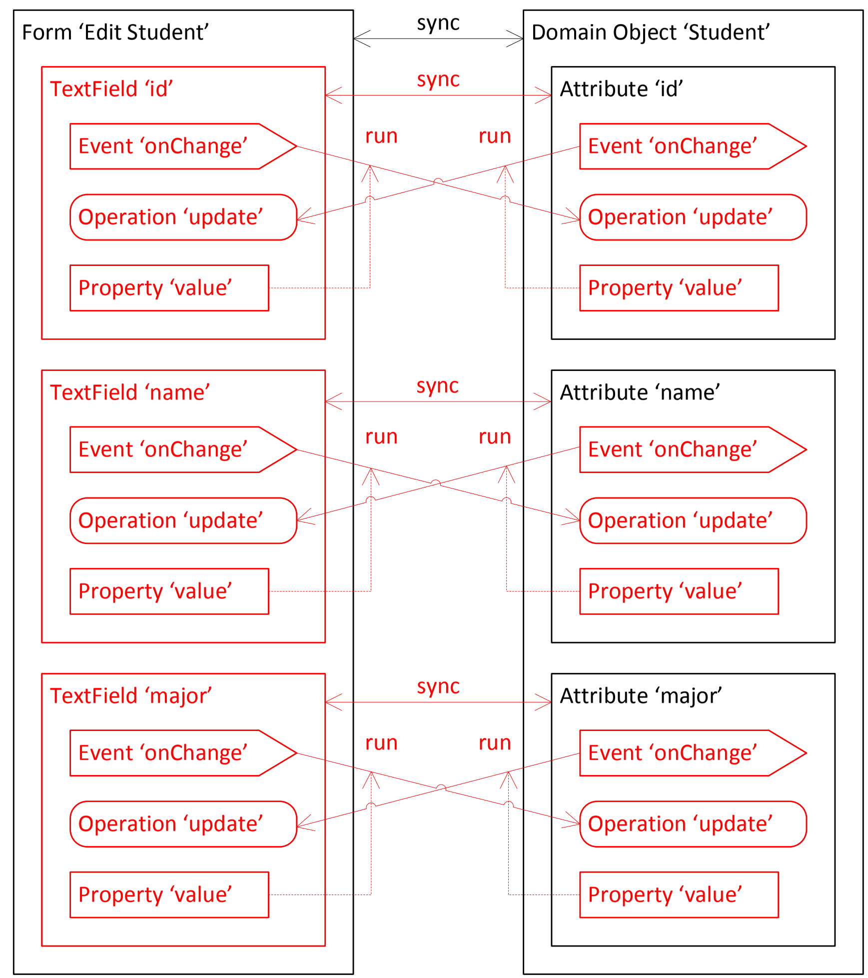
Add events, operations, and properties: + 18 elements

Visualisation: Step 4



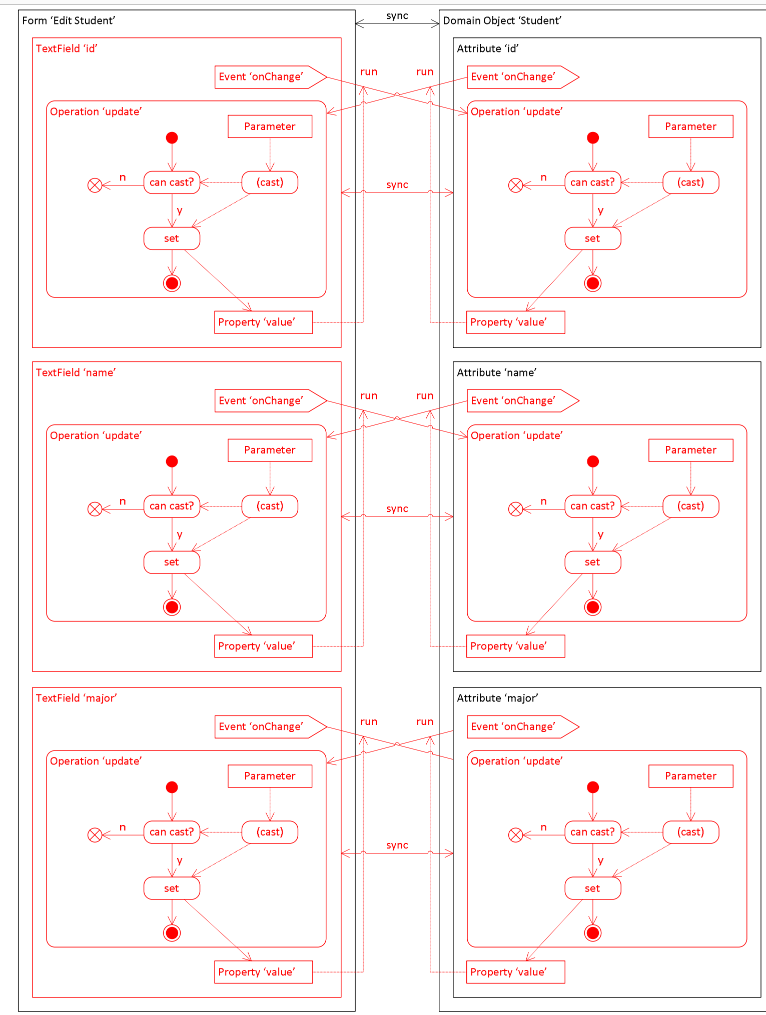
Connect events with operations, using actions: + 6 elements

Visualisation: Step 5



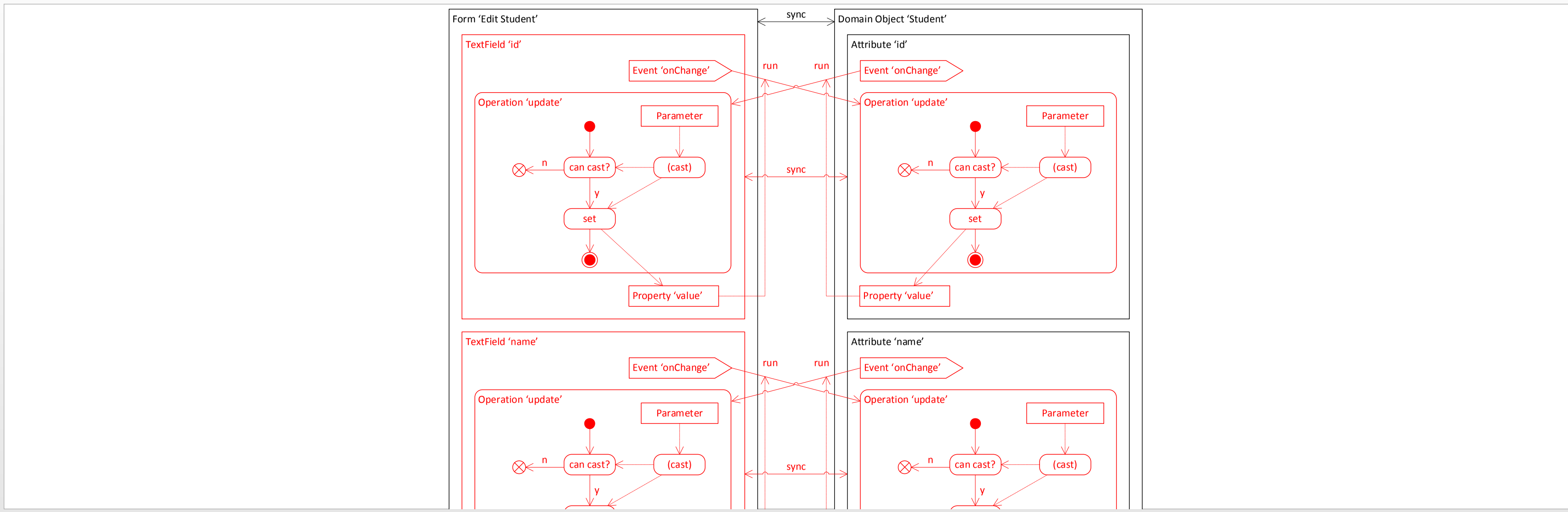
Add parameters to actions: + 6 elements

Visualisation: Step 6



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

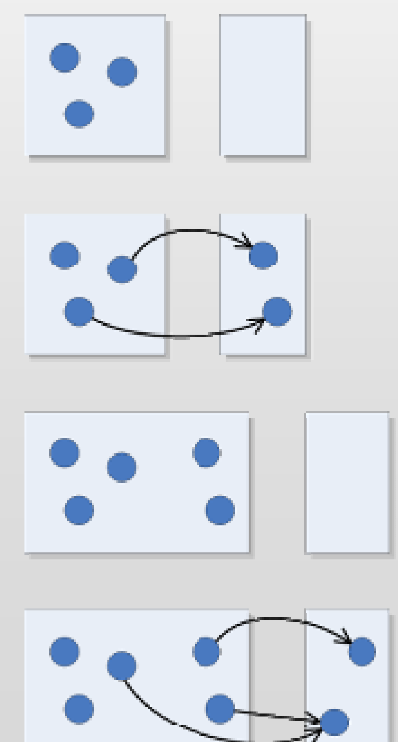
- Implemented with a commercial rule engine
  - JBoss Rules (Drools)
  - BSD/MIT-esque license
  - Integrates well with EMF Models

```
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

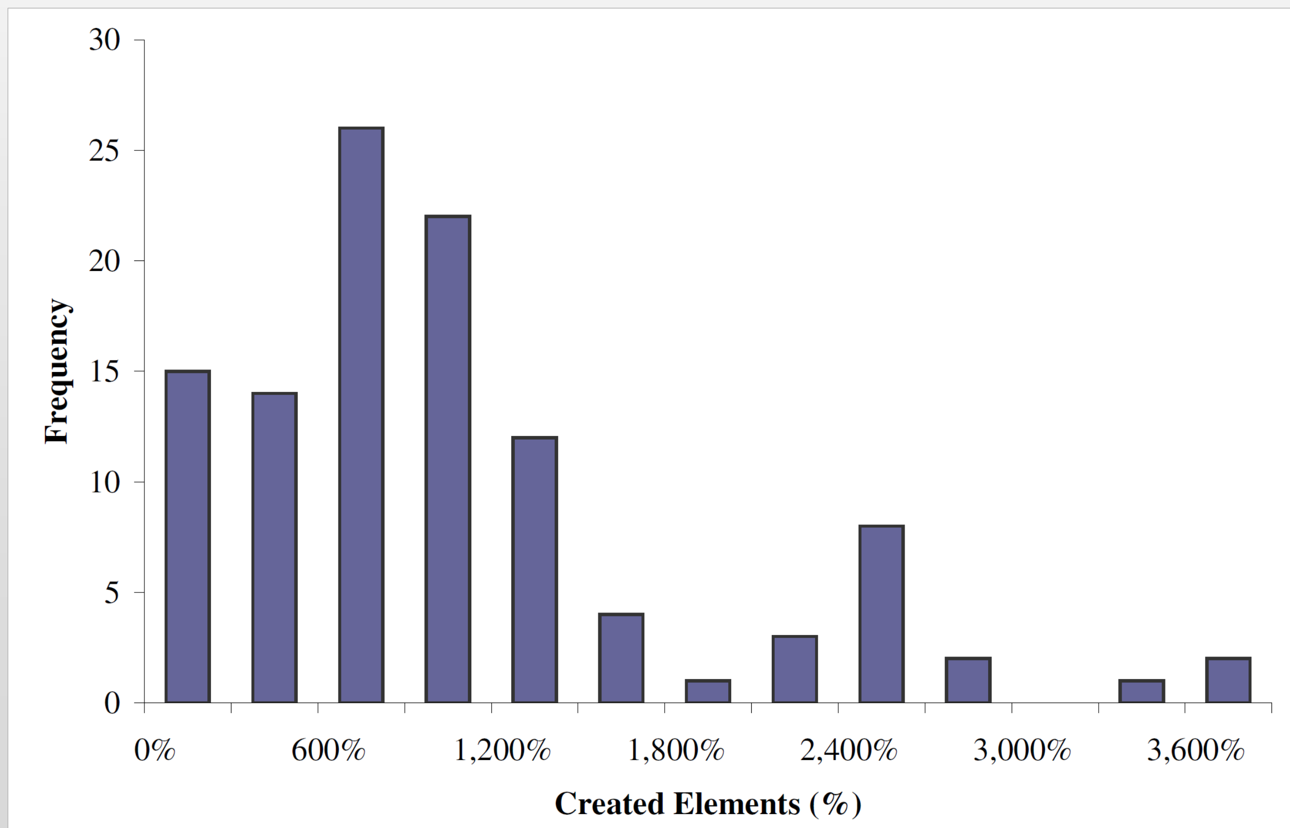
- Some metrics of the test models
  - 110 models used in ASWEC paper

	Minimum		Median		Mean		Maximum		Std Dev	
Metric	Initial	Final	Initial	Final	Initial	Final	Initial	Final	Initial	Final
Elements	2	3	11.5	107	15.1	207.5	76	2,094	13.2	338.9
Attributes	9	9	48	201	58.8	391.7	310	3,934	51.0	635.8
Non-default Attributes	4	6	23	166	28.0	321.8	148	3,219	24.7	520.2
Distinct Attribute Values	6	8	24	135	29.3	240.8	138	2,176	22.7	353.6
References	0	0	12	164	16.5	351.4	112	3,632	21.5	584.7
Children	1	2	10.5	106	14.1	206.5	75	2,093	13.2	338.9
Distinct Types	2	3	9	19	8.9	21.8	19	41	3.7	8.9
Min Degree (References and Children)	0	0	0	0	0.3	0.0	1	1	0.5	0.1
Max Degree (References and Children)	1	1	4	10	5.0	14.0	25	57	3.5	8.6
Children Depth	1	2	3	5	3.2	4.6	4	6	0.7	0.6
Cycles (References and Children)	0	0	2	2	3.0	4.8	20	44	3.6	8.0
Diameter (References and Children)	1	2	5	12	5.6	14.2	16	27	2.9	6.5
Average Completion Time (ms)		0.0		8.6		117.6		5,213.6		696.8

TABLE I
SELECTED MODEL METRICS OF INITIAL AND COMPLETED TEST MODELS ($n = 110$)

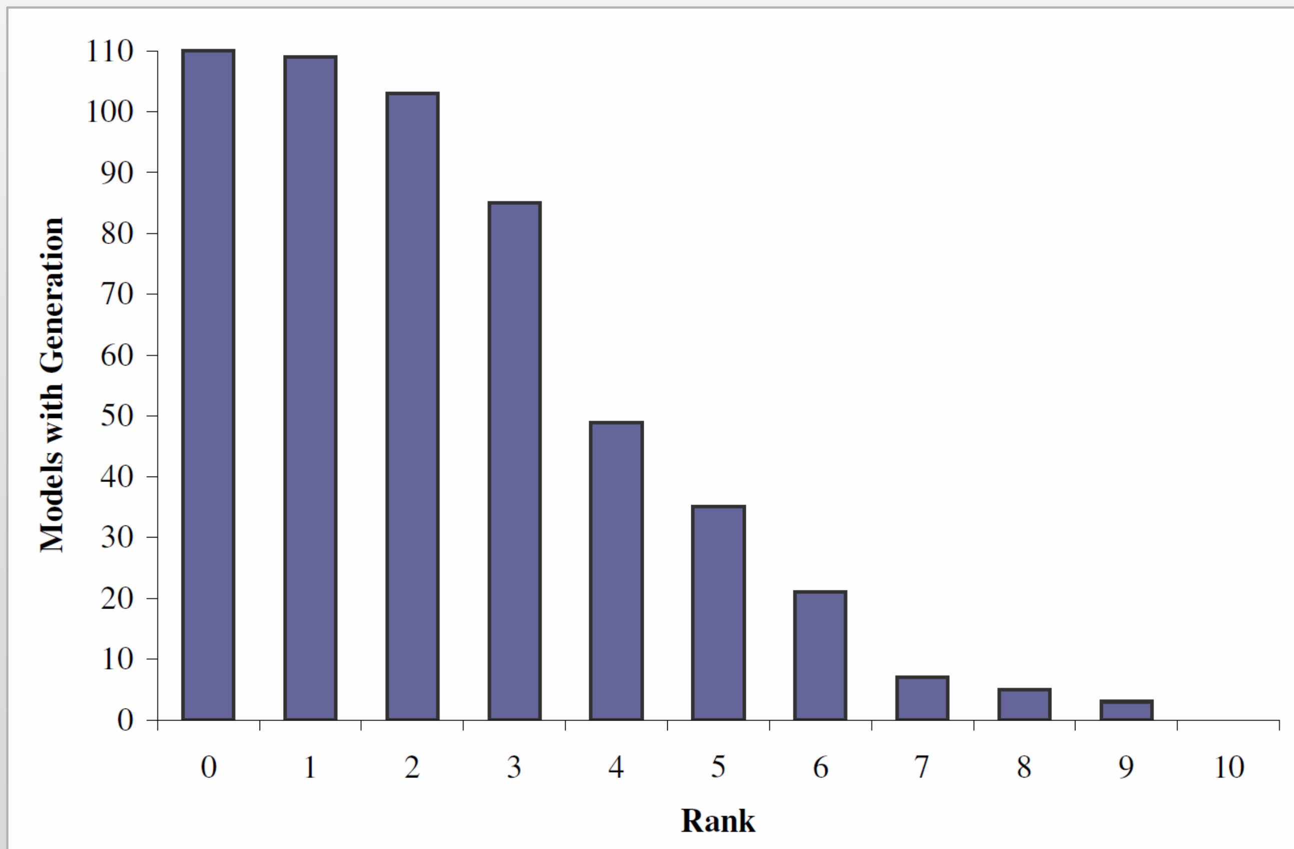
Results

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



Results

- At most, 9 steps required to complete model
- We conservatively limit *k* to 20



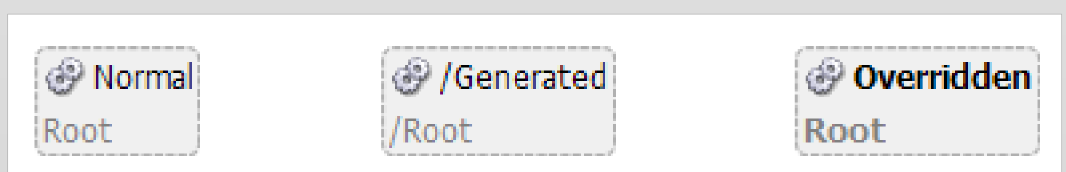
Overriding Model Completion

- 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

```
rule "Example rule"
   when
      ...
      eval ( handler.veto( p ))

   then
      Checkbox c = handler.generatedCheckbox(p);
      ...
```

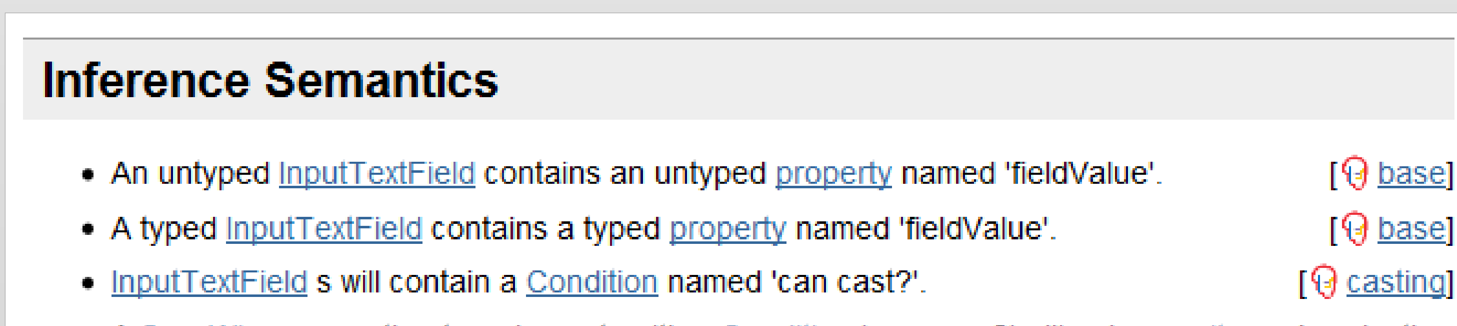
CASE Tool Implementation

- 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



Inference Semantics

- An untyped InputTextField contains an untyped property named 'fieldValue'. [base]
- A typed InputTextField contains a typed property named 'fieldValue'. [base]
- InputTextField s will contain a Condition named 'can cast?'. [casting]

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