Turning Models Inside Out

Alfa Yohannis Fiona Polack Dimitris Kolovos

Department of Computer Science
University of York
York, United Kingdom
{ary506, fiona.polack, dimitris.kolovos}@york.ac.uk

FlexMDE @ MODELS'17

Introduction

- Model-based software engineering in the context of large and complex systems demands ability to process large models.
- One way to tackle this is through the use of incremental model processing techniques.
- Current incremental model processing techniques only deliver limited performance benefit due to:
  - Slow inprecise model change detection capabilities, and
  - Limited to a single-developer environment – not realistic for software development projects.

Our Aim

- We aims at enabling flexible and high-performance incremental model processing through change-based model persistence.
- Instead of persisting snapshots of the state of models, we propose turning models inside out and persisting their change history.
- It has the potential to deliver step-change performance benefits in incremental model processing, as well as a wide range of other benefits and novel capabilities.

An Example of Change-based Techniques (1)

Report Generation Optimisation of an Organisational Chart Model

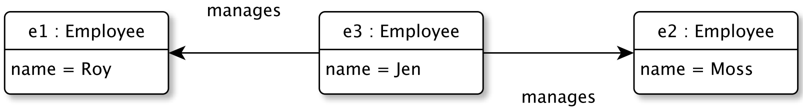

Figure 1: Initial version of the organisational chart model.

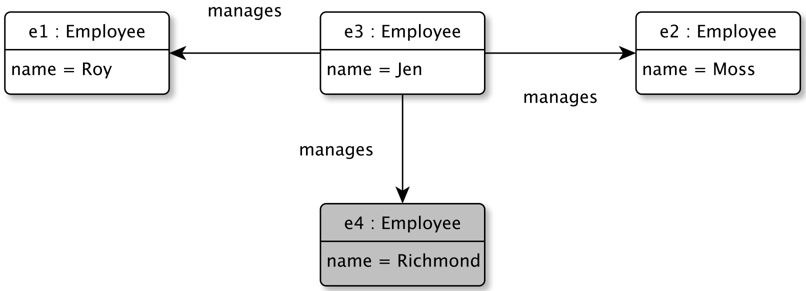

Figure 2: Modified version of the organisational chart model of Fig. 1.

An Example of Change-based Techniques (2)

Report Generation Optimisation of an Organisational Chart Model

- We have two consecutive organisational chart models (Fig. 1 and 2), which after every modification, they need to be:
  - Validated against a domain-specific constraint (that no employee directly manages more than 7 other employees).
  - Transformed into a number of employee reports through a model-to-text transformation. Each report contains the name of the employee, and the names of her direct subordinates.
- At the end of Fig. 1's modification:
  - All employees satisfy the constrain (Roy, Jen, Moss).
  - Three reports are generated (Roy, Jen, Moss).
- At the end of Fig. 2's modification:
  - All employees satisfy the constrain (Roy, Jen, Moss, Richmond).
  - Two reports are (re)generated (Jen, Richmond).

An Example of Change-based Techniques (3)

Report Generation Optimisation of an Organisational Chart Model

Non-incremental vs incremental approaches on optimising model validation and transformation.

- Non-incremental approach:
  - (re)evaluates the constrain against all the employees
  - (re)generates reports for all the employees
- Incremental approach:
  - only (re)evaluates the constrain against Jen and Richmond
  - only (re)generates reports for Jen and Richmond

Non-incremental execution can become a significant bottleneck for large evolving models, particularly in the latter phases of the development cycle when many small changes made to fine-tune the system [1].

Requirements for Incremental Engine

As demonstrated by Egyed [2], to achieve incremental re-execution of (deterministic) queries on structured models, an execution engine needs to:

1. Record model element property accesses during the initial execution of the queries;
2. Identify new and deleted elements and modified model element properties in the new version of the model;
3. Combine the information collected in the steps above to identify the subset of (potentially) affected rules/queries/templates that need to be re-executed.

Identifying Changes in Models

Two approaches for identifying changes in models to enable incremental model processing.

- **Notifications**. The engine listens to the notification facility of a modelling tool and receives notifications as soon as the model is modified (add a new employee).
  - Applied by: IncQuery incremental pattern matching framework [3], ReactiveATL incremental model-to-model transformation engine [4].
  - Advantages: fine-grained change notification facilities are provided for free by modelling tools.
  - Drawbacks: poor for collaborative development when a model is modified by different developers.
- **Model Differencing**. A Model is compared to its previous version and the engine computes their differences.
  - Applied by: SiDiff [5] or EMFCompare[1]).
  - Advantages: works well in a collaborative development with different roles and responsibility of developers.
  - Drawbacks: computationally expensive, memory-greedy.

[1] https://www.eclipse.org/emf/compare/

A State-Based Representation

A state-based representation of the model of Fig. 2 in XMI.

```
<Employee xmi:id="e2" name="Jen">
  <manages xmi:id="e1" name="Roy"/>
  <manages xmi:id="e3" name="Moss"/>
  <manages xmi:id="e4" name="Richmond"/>
</Employee>
```

A Change-based Persistence Representation

A change-based representation of the model of Fig. 2.

```
<session id="s1"/>
<create eclass="Employee" epackage="employee" id="0"/>
<add-to-resource position="0"><value eobject="0"/></add-to-resource>
<set-eattribute name="name" target="0"><value literal="Roy"/></set-eattribute>
<create eclass="Employee" epackage="employee" id="1"/>
<add-to-resource position="1"><value eobject="1"/></add-to-resource>
<set-eattribute name="name" target="1"><value literal="Jen"/></set-eattribute>
<create eclass="Employee" epackage="employee" id="2"/>
<add-to-resource position="2"><value eobject="2"/></add-to-resource>
<set-eattribute name="name" target="1"><value literal="Moss"/></set-eattribute>
<remove-from-resource><value eobject="0"/></remove-from-resource>
<add-to-ereference name="manages" position="0" target="1"><value eobject="0"/></
      add-to-ereference>
<remove-from-resource><value eobject="2"/></remove-from-resource>
<add-to-ereference name="manages" position="1" target="1"><value eobject="2"/></
      add-to-ereference>
<session id="s2"/>
<create eclass="Employee" epackage="employee" id="3"/>
<add-to-resource position="1"><value eobject="3"/></add-to-resource>
<set-eattribute name="name" target="3"><value literal="Richmond"/></set-eattribute
      >
<remove-from-resource><value eobject="3"/></remove-from-resource>
<add-to-ereference name="manages" position="2" target="2"><value eobject="3"/></
      add-to-ereference>
```

Prototype Implementation

- We have implemented a prototype. The prototype is available under `https://github.com/epsilonlabs/emf-cbp`.
- The change-based model persistence format uses the notification facilities provided by the Eclipse Modelling Framework.
  - *EContentAdapter* class `http://download.eclipse.org/modeling/emf/emf/javadoc/2.11/org/eclipse/emf/ecore/util/EContentAdapter.html`
  - *Notification* class `http://download.eclipse.org/modeling/emf/emf/javadoc/2.11/org/eclipse/emf/common/notify/Notification.html`
- Basically, it works by capturing events from the *Notification*: filtering and putting the events into a change-event list, and persisting them into a change-based representation.

Event Classes to Represent Changes of Models

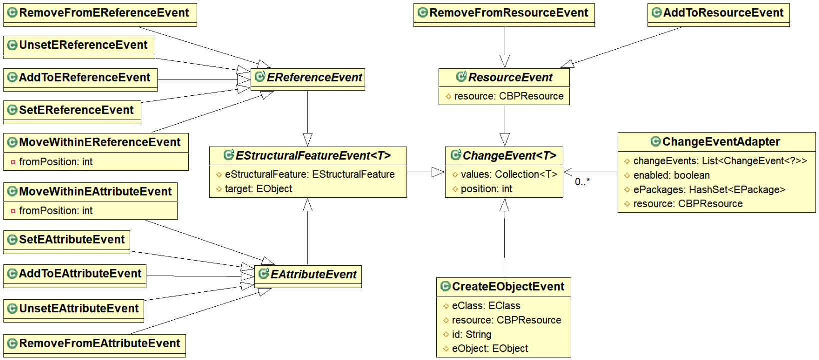

Figure 3: Event classes to represent changes of models.

Java Code to Handle Notifications

Simplified Java code to handle notification events.

```
public class ChangeEventAdapter extends EContentAdapter {
...
@override
public void notifyChanged(Notification n) {
  ...
  switch (n.getEventType()) {
    ... // other events
    case Notification.UNSET: {
    if (n.getNotifier() instanceof EObject) {
      EStructuralFeature feature = (EStructuralFeature) n.getFeature();
        if (feature instanceof EAttribute) {
            event = new UnsetEAttributeEvent();
          } else if (feature instanceof EReference) {
            event = new UnsetEReferenceEvent();
          }
      } break;
    }
    ... // other events
```

Factory, resources, and ChangeEventAdapter classes

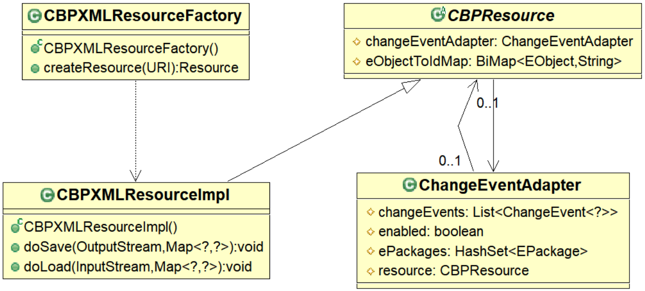
CBPXMLResourceFactory
CBPXMLResourceFactory()
createResource(URI):Resource
CBPResource
changeEventAdapter: ChangeEventAdapter
eObjectToIdMap: BiMap<EObject,String>
0..1
0..1
CBPXMLResourceImpl
CBPXMLResourceImpl()
doSave(OutputStream,Map<?,?>):void
doLoad(InputStream,Map<?,?>):void
ChangeEventAdapter
changeEvents: List<ChangeEvent<?>>
enabled: boolean
ePackages: HashSet<EPackage>
resource: CBPResource

Figure 4: Factory, resources, and ChangeEventAdapter classes.

Benefits and Novel Capabilities

- **Tool Support**. With appropriate tool support, modellers will be able to "replay" the change history of a model (e.g. to understand design decisions made by other developers).
- **Analytics**. By analysing models serialised in the proposed representation, modelling tool vendors will be able to develop insights into how modellers actually use their tools in practice.
- **Tracing**. By attaching additional information to each session, we can trace back the developer that made changes, or the requirements/bug reports that triggered them.
- **Performance**. Persisting changes to large models after an editing session will be significantly faster, as well as the performance and precision of model comparison and merging can be substantially improved.

Challenges and Future Work

- **Loading Overhead**. Loading models into memory by naively replaying the entire change history has a significant overhead. To address it, we will develop dedicated algorithms and data structures that will reduce the cost of the loading (e.g. ignoring events that are later overridden or cancelled out by other events).
- **Fast-Growing Model Files**. A file for persisting models in a change-based format will grow in size faster than their state-based counterparts. To address this challenge, we will (1) propose sound change-compression operations (e.g. remove older/unused information) to reduce size; (2) develop a compact textual format to minimise space required (a textual line-separated format is desirable for compatibility with file-based version control systems); (3) propose a hybrid model persistence format to incorporate change-based and state-based information.

Evaluations

- **Vs Existing Algorithms/Tools**. Evaluation where there are existing approaches that the algorithms and tools developed in this research seek to outperform (e.g. change-based incremental validation vs. state-based incremental validation), comparative evaluation will be conducted to assess the benefits and limitations of our approaches.
- **Vs Seek-to-be-improved Baselines**. For algorithms and tools that have no direct competitors in the literature, their contributions will be assessed in comparison to the baseline they seek to improve (e.g. vs persisting full change histories).

Conclusions

- This research aims at enabling high-performance incremental model processing in collaborative development settings through turning models inside out and persisting their change history.
- A prototype implementation of a change-based persistence format has been presented, the main envisioned challenges have been listed and an evaluation strategy has been outlined.
- The proposed approach also has the potential to enable model analytics, more fine-grained tracing and to improve the precision and performance of model comparison and merging.

References

[1] B. Selic, "The pragmatics of model-driven development," *IEEE software*, vol. 20, no. 5, pp. 19–25, 2003.

[2] A. Egyed, "Automatically detecting and tracking inconsistencies in software design models," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 188–204, 2011.

[3] I. Ráth, Á. Hegedüs, and D. Varró, "Derived features for emf by integrating advanced model queries," *Modelling Foundations and Applications*, pp. 102–117, 2012.

[4] B. Ogunyomi, L. M. Rose, and D. S. Kolovos, "Property access traces for source incremental model-to-text transformation," in *European Conference on Modelling Foundations and Applications*. Springer, 2015, pp. 187–202.

[5] U. Kelter, J. Wehren, and J. Niere, "A generic difference algorithm for uml models." *Software Engineering*, vol. 64, no. 105-116, pp. 4–9, 2005.