Master Thesis
-Proposal-

Towards Ecore based Meta Model Evolution and Model Co-Evolution

Supervised by:
Prof. Dr. Wilhelm Hasselbring (Uni Oldenburg)
Sven Efftinge (Itemis)

To be Written & Implemented by:
Moritz Eysholdt
Outline

- Problem & Motivation
- EMF Introduction
- Overview: Roles & Processes
- Details: Processes
- Epatch & Metapatch
- Outlook
Problem

<<changes data structure aka meta model>>

<<reads, writes>>

developer

<<no adaptation>>

<<reads, writes>>

data aka model

v1

edit

v2
Problem #2

- Changeability of the meta model: Essential for further advancement of the application
- Model: Often not accessible for the developer
- Changes that can be made to the meta model without breaking the model are limited
Possible Approaches

- Ignore backward compatibility: Schema Revolution!
- Use a “weak” meta model
- Mark elements “deprecated”, leave the migration work to the users

- Wanted: automatic adaptation of the model to be valid for the new meta model.
  - > Meta model evolution & model co-evolution
Eclipse Modeling Framework (EMF)

- Ecore
  - The meta meta model
  - Similar to OMG EMOF
  - Defines itself
    - Classes, DataTypes, Packages, Attributes, (containment/bidirectional) References, Operations, Lists, FeatureMaps, Enums, etc.

- Models Serializable to XMI
- Available Ecore based meta models: XMLSchema, UML2, BPEL, etc.
Proposal

- To derive a model to model transformation from the meta model's editing operations which automatically adapts models to the changes.
- Problem: This can not be done completely automatically in all cases
  - > User assistance necessary
Roles and Processes
(overview, details follow)

Meta Model Engineer
- Changes meta model
- Changes are recorded
- Refines changes
- Changes are classified
- Algo. is generated

Model Engineer
- Opens old model
- Appropriate algo. is chosen
- Algo. is executed
Meta Model Engineer's Process: Step 1

1. **edit**

2. **diff**

3. **observe model**

4. **use ChangeRecorder**

5. **observe CommandStack**

6. **observe LTK operations**

7. **DiffModel**

8. **Notifications**

9. **Commands**

10. **LTK operations**

Derive atomic changes, group and augment them to refined changes.

Refined changes.
Meta Model Engineer's Process: Step 2

- refined changes
- classify
  - not breaking
  - breaking and resolvable
  - breaking and not resolvable

- generate transformations
- generate TODOs
- implement transformations
- model co-adaptation algorithm (transformation)

change classification
generate model transformation algorithm
Model Engineer's Process

- model v1 instance
- detect version
- choose algo
- algo.
- apply algorithm
- model v2 instances

expected target version

algorithms

apply algorithm
My Focus

Focus on the Meta Model Engineer's process:

1. Obtain changes made to the meta model and store them in a model: *Epatch*

2. Group and Enrich these changes and store them in a model: *MetaPatch*

3. Implement an execution mechanism for MetaPatches
The Epatch

- Inspired by the UNIX-world's patch
- A model that describes differences between two models.
- Bidirectional: The same Epatch can upgrade model v1 to v2 as well as downgrade model v2 to v1
- A set of atomic changes that can be reused by the MetaPatch
- Imperative/Declarative format? Declarative preferred
- Must cover element move/copy operations
- Implement a recorder
The MetaPatch

• Extends Epatches that define differences between meta models with instructions how to co-adapt the models

• Group atomic changes from the Epatch to composite changes

• Enrich composite changes with co-adaptation information

• Major part of my work: find reasonable groupings and options for enrichment
Outlook

• Code generation: Generate code for the model co-adaptation algorithm
• Deriving Epatches from diffs: Using EMF Compare? Challenge: Identify copies/moves
• Handling a meta model's extended information
  - XML: XSD Infoset uses EMF Annotations
  - DSL: TMF Xtext has separate grammar files
  - Relational Database: Teneo?
Sources


Questions?
Comments?
Models: M0, M1, M2, M3

M0: instance
M1: model
M2: meta model
M3: meta meta model

EMF Ecore, MOF, XML Schema etc.

EMF Meta Models, Grammars (DSL)
UML Diagram Types etc.

EMF Models, UML Diagrams,
DSL-Script, Sourcecode etc.

Program Instance