Change-Driven Model Transformations

Derivation and Processing of Change Histories

István Ráth, Gergely Varró, Dániel Varró
rath@mit.bme.hu
Outline of the talk

- Introduction
- Motivation
- Overview of the concept
- Change-driven transformations in detail
- Summary
- Future work
Motivating scenario: forward model synchronization

Source

$M_A$

$M_A'$

Target

$M_B$

$M_B'$

change
Incremental model synchronization and *time*

- On-demand: batch transformations
  - The “traditional” way
Incremental model synchronization

- Re-transform
- Target incrementality
Model synchronization and time

- On-demand: batch transformations
  - The “traditional” way

- Instantly: live transformations
  - React instantly to context (model) changes
    - “event-driven” transformations
    - Transaction-oriented approach
    - Reactions possible to arbitrarily complex changes
Live model synchronization

1. Watch for changes
2. React to changes
3. Merge
Model synchronization and *time*

- **On-demand:** batch transformations
  - The “traditional” way
- **Instantly:** live transformations
  - React instantly to context (model) changes
    - “event-driven” transformations
    - Transaction-oriented approach

**Common assumptions:**
1. All models are available in memory
2. Changes are propagated “synchronously”
Asynchronous synchronization

- What if...
  - Some models cannot (should not) be materialized in memory?
    - Models are too large
    - Models have to be manipulated “inside” their native environment (tool)
  - Changes are to be applied/reproduced “later”?
    - Changes have to recorded for e.g. traceability

→ Asynchronous (off-line) synchronization
Motivating scenario

High level (domain-specific) process model

Deployed process template (jPDL)
Case study and challenges

- Tool integration in a heterogeneous environment
  - Developed for the SENSORIA and MOGENTES EU research projects

- High level process models describe (complex) development process segments
  - E.g. automated test generation, deployment configuration generation

- Processes are executed in
  - A distributed environment (workstations, tool servers)
  - Orchestrated by the jBPM process execution engine.
Challenges

- Challenge #1: high level models are edited → changes have to be propagated to the deployed process template
- Challenge #2: changes are mapped asynchronously in time
  - Not (necessarily) by the process engineer
Conceptual overview

1. **Record** changes into traceability models (=CHMs)

2. **Map** source *changes* to target *changes* (=CHMs to CHMs) instead of source models to target models

3. **Apply** changes to external models through an interface (IF)
Change history models

- Traceability models
  - Operational difference models
  - Record historical operation sequences
    - WHEN (timestamps in a linked list structure)
    - WHAT (CUDM)
    - Context (referenced model elements)
  - “weak” references
    - IDs or FQNs
    - Allows to reference external (non- or partially materialized) models
Change history metamodel

Weak references

Operation categories

Historical record
Generation of CHMs

- Live transformations
  - Editor-independent!
- Generate trace model snippets as the user is editing the model
  - Timestamps
  - Contextual references
Generation of CHMs: Generic example

Sample execution sequence:

**Step 1**
- **w0**: Workflow

**Step 3**
- **w0**: Workflow
- **IO**: Invocation
  - targetFQN: model.i0
  - parentFQN: model.w0
  - typeFQN: meta.Invocation

Entity and Relation are basic VIATRA concepts for graph node and edge.
Generating domain-specific CHMs

1. Use a compound pattern as precondition, corresponding to a (complex) model structure

2a. Create a compound CHM sequence as postcondition

2b. Use a “compressed” CHM element corresponding to a complex domain-specific operation
Change-driven transformations

- **Input:**
  - Changes of the source model

- **Output**
  - Corresponding changes of the target model

- **May be formulated as:**
  - Live transformation
  - Batch transformation

- **Granularity?**
  - “one-to-one”
  - “n-to-m”
For each newly created Invocation, create a corresponding JPDL node together with an “function” attribute (domain-specific mapping logic).
Applying CHMs to external models

- Applying CHMs = model “interpretation”
- External models are manipulated through a (service) interface
  - VIATRA: “native functions”
Manipulating non-materialized models with VIATRA

VIATRA native functions allow for DOM-style manipulation of the deployed jPDL process template.
Summary

- Change-driven transformations =
  - An innovative synthesis of known techniques:
    - Trace models
    - Live transformations
    - Non-materialized model manipulation
  - A solution for an engineering problem
  - Lots of open questions and new ideas...
Future work

- A beginning, rather than an end...
- Lots of open questions
  - How to write CDTs?
  - How to generate CDTs from “traditional” transformations?
- Are they useful?
  - Efficient, intuitive model synchronization
  - Change representation, processing ($\rightarrow$ (re)verification, change impact analysis)
  - Model merging (~operational merging)
- Thank you for your attention.