Change-Driven Model Transformations

Derivation and Processing of Change Histories

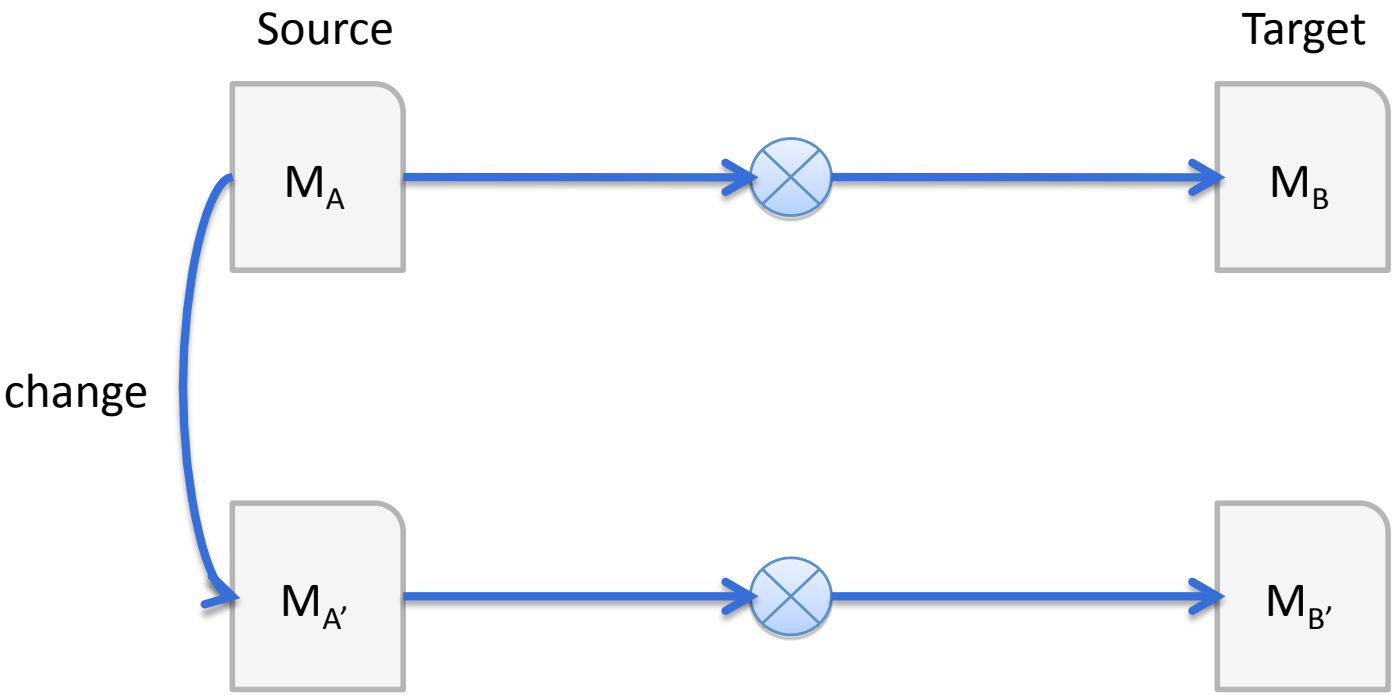
István Ráth, Gergely Varró, Dániel Varró

rath@mit.bme.hu

Outline of the talk

- Introduction
- Motivation
- Overview of the concept
- Change-driven transformations in detail
- Summary
- Future work

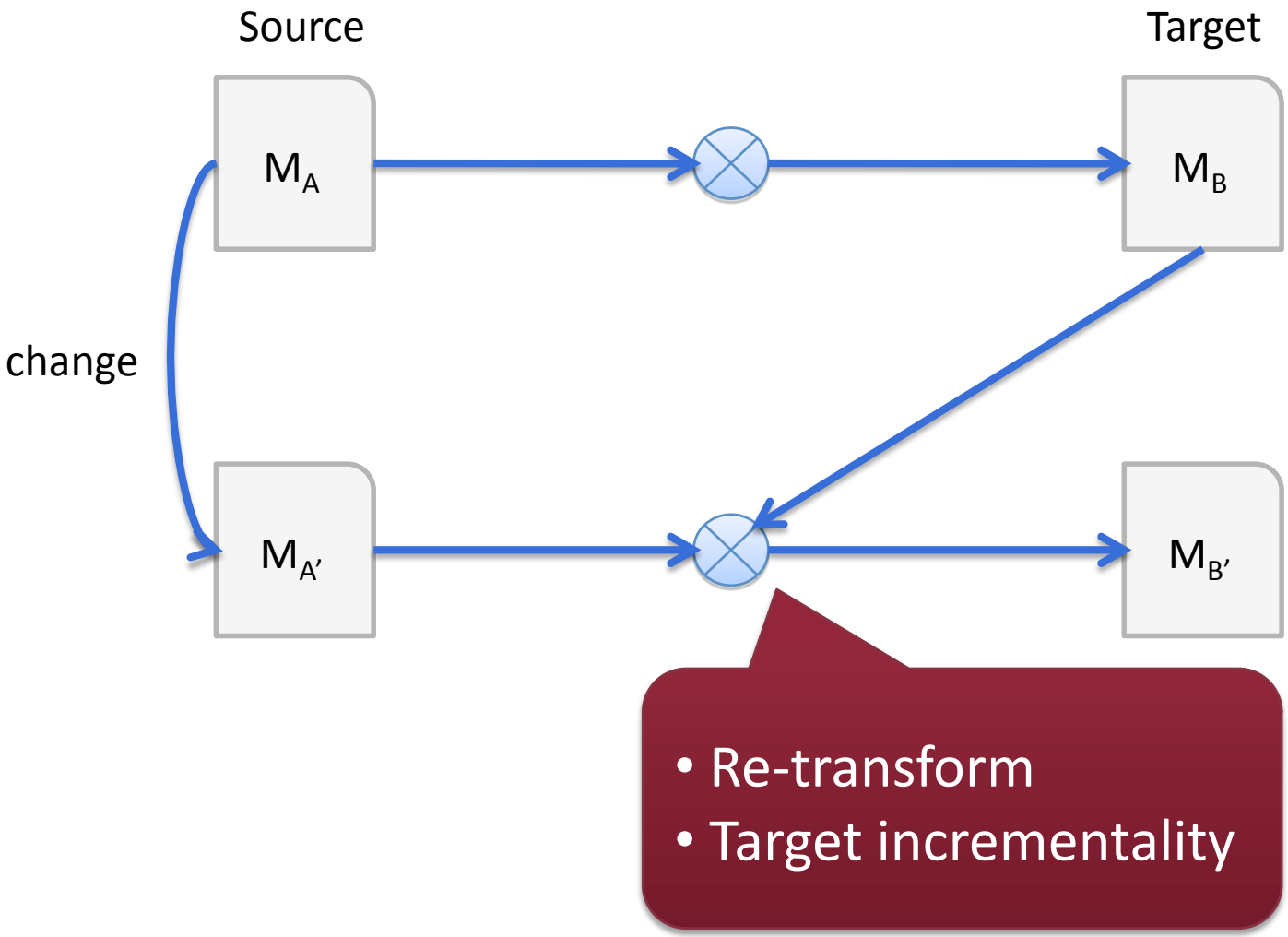
Motivating scenario: forward model synchronization



Incremental model synchronization and *time*

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

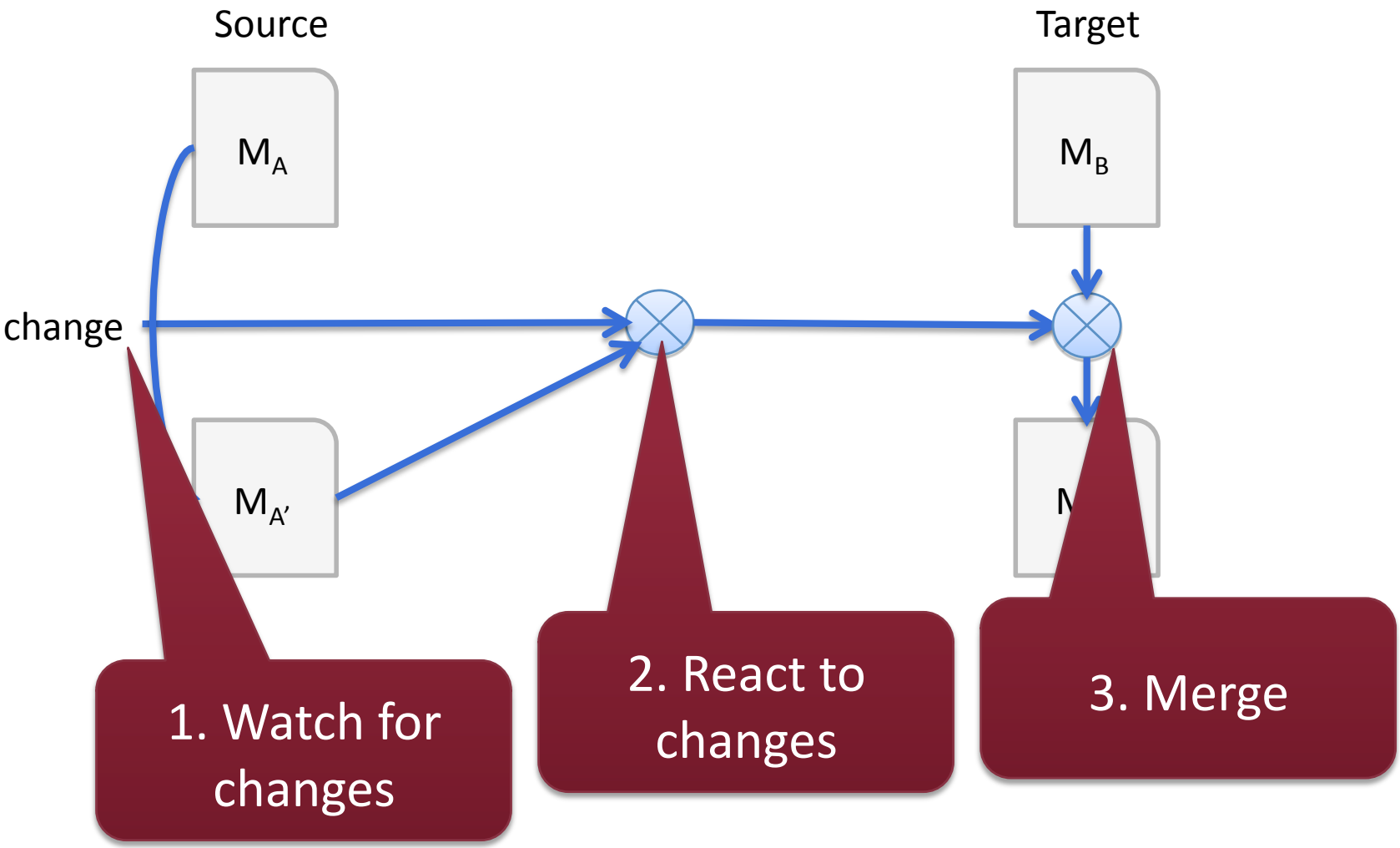
Incremental model synchronization



Model synchronization and *time*

- On-demand: batch transformations
  - The "traditional" way
- Instantly: live transformations
  - React instantly to context (model) changes
    - "event-driven" transformations
  - Hearnden-Lawley-Raymond. *Incremental Model Transformation for the Evolution of Model-driven Systems.* MODELS 2006.
  - Ráth-Ökrös-Bergmann-Varró. *Live model transformations driven by incremental pattern matching*. ICMT 2008.
    - Transaction-oriented approach
    - Reactions possible to arbitrarily complex changes

Live model synchronization



Model synchronization and *time*

- On-demand: batch transformations
  - The "traditional" way
- Instantly: live transformations
  - React instantly to context (mod nges
    - "event-driven" transformations
  - Hearnden-Lawley-Raymon *Transformation for the Evo Systems.* MODELS 2006.
  - Ráth-Ökrös-Bergmann-Var *transformations driven by matching*. ICMT 2008.
    - Transaction-oriented approach

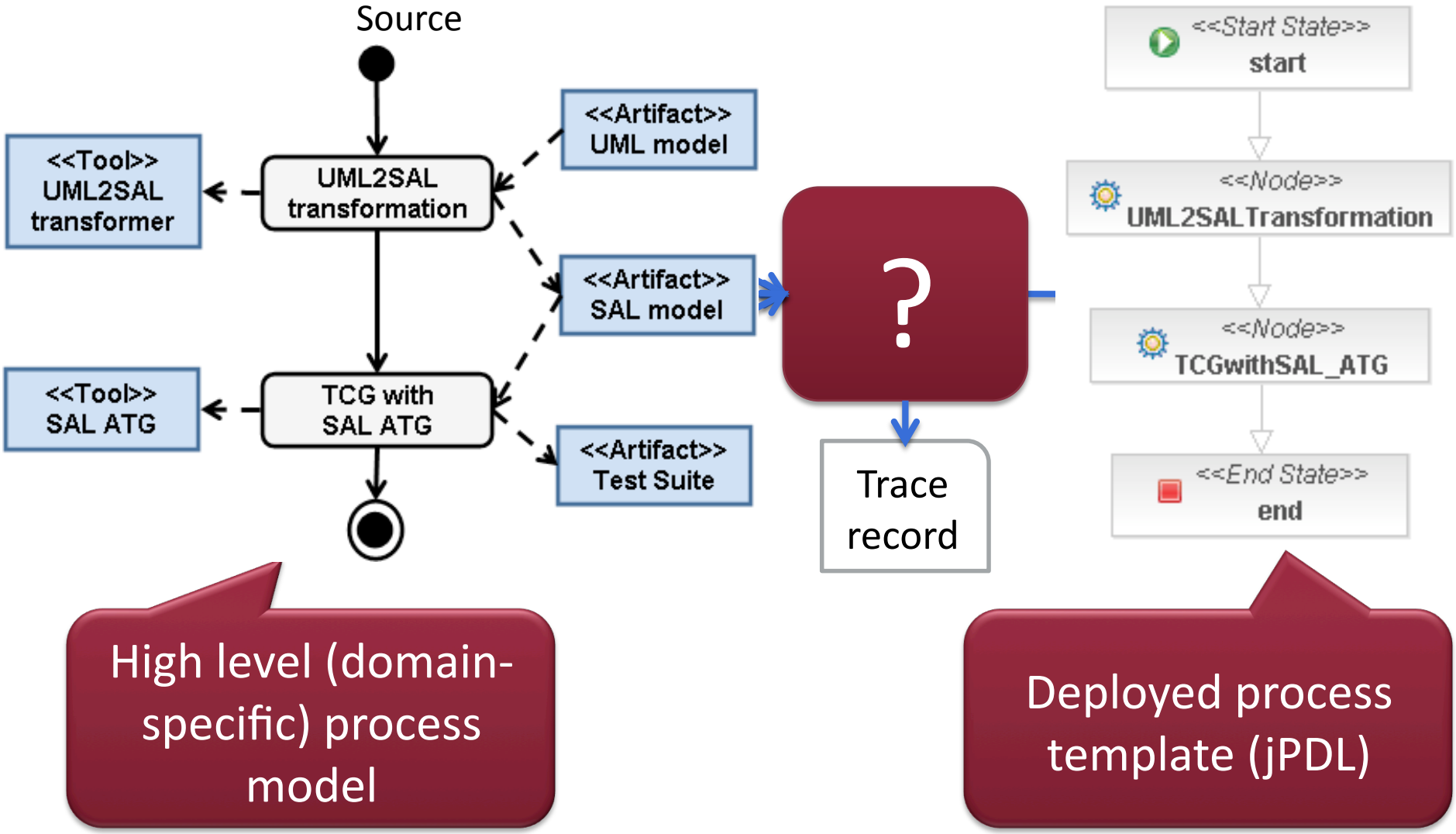
Common assumptions:

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



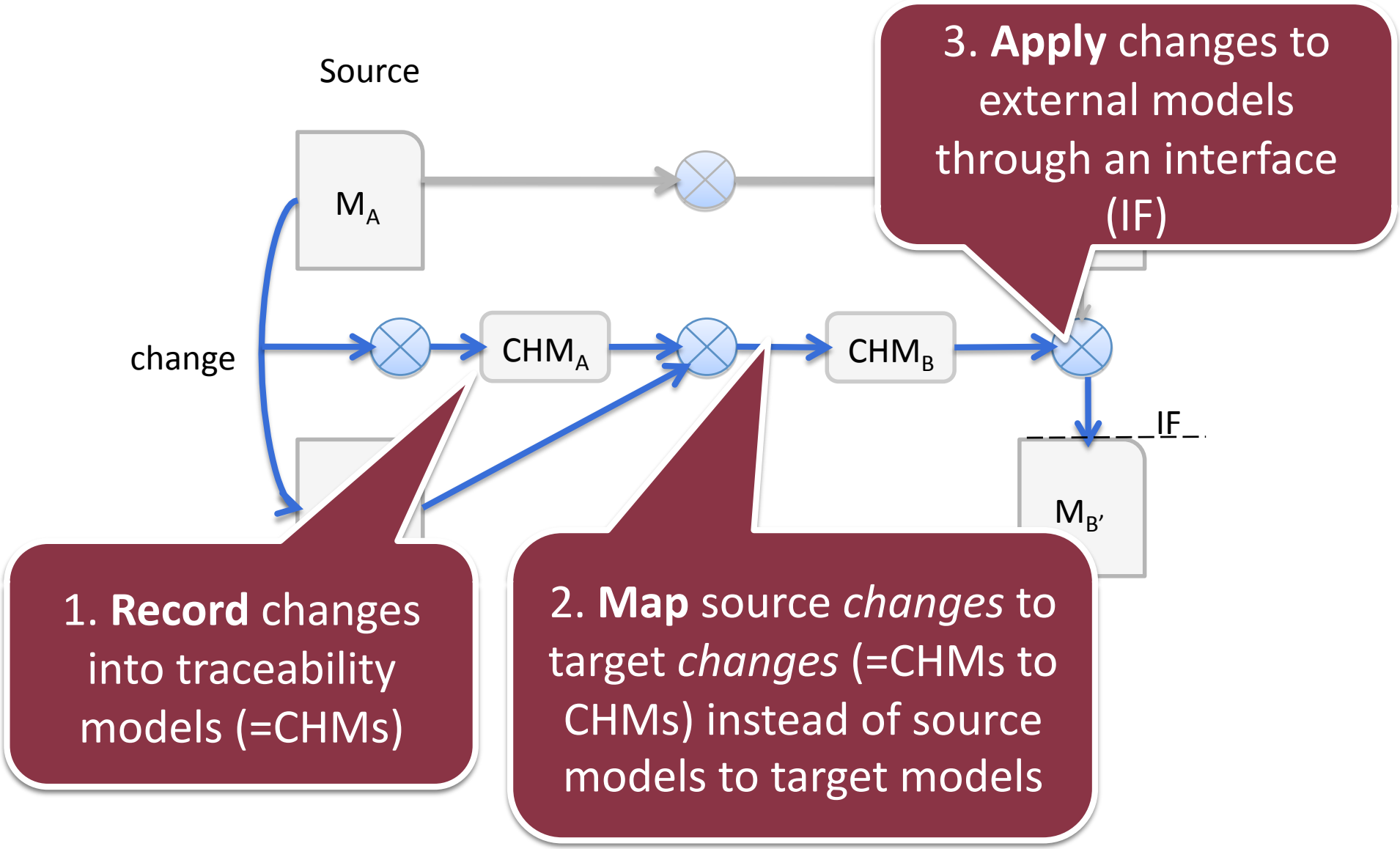
Case study and challenges

- 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



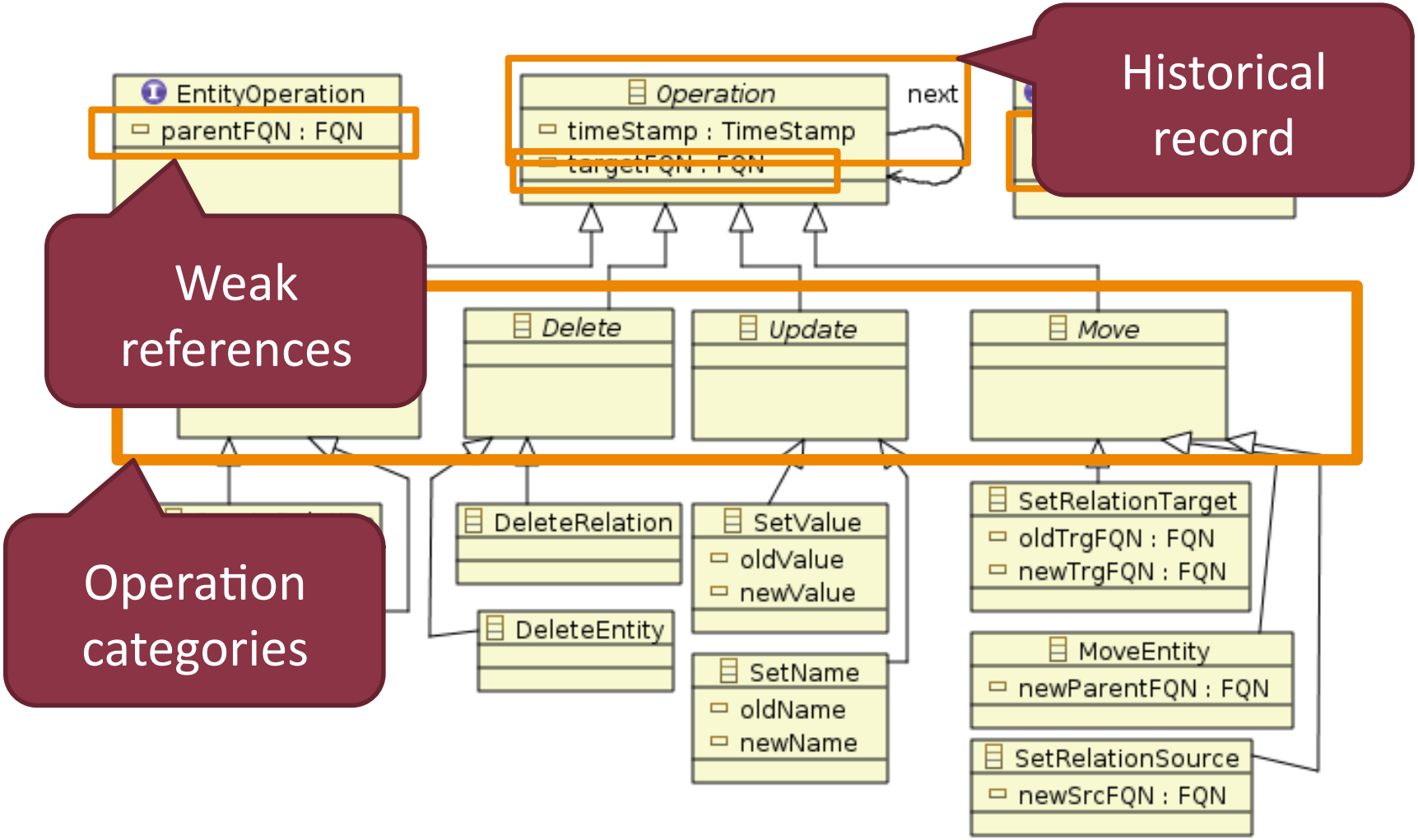
Change history models

- Traceability models

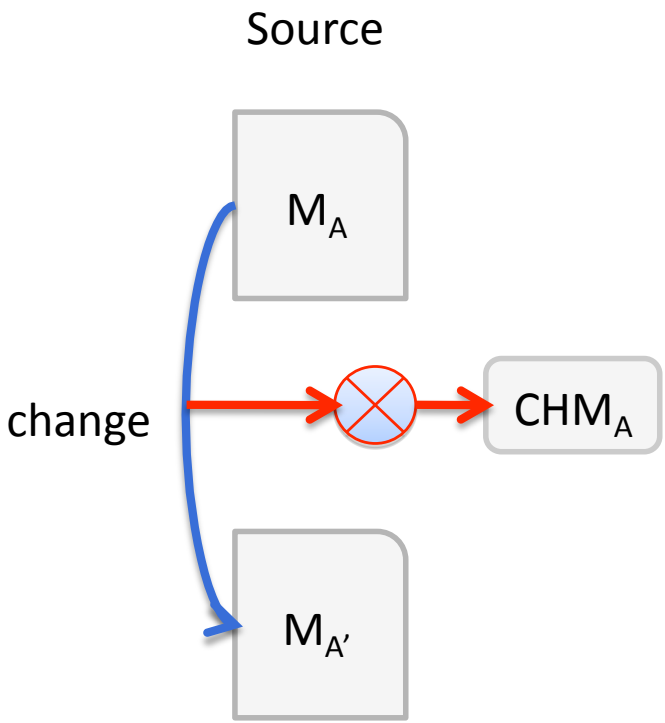
  CHM_A CHM_B

  - 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

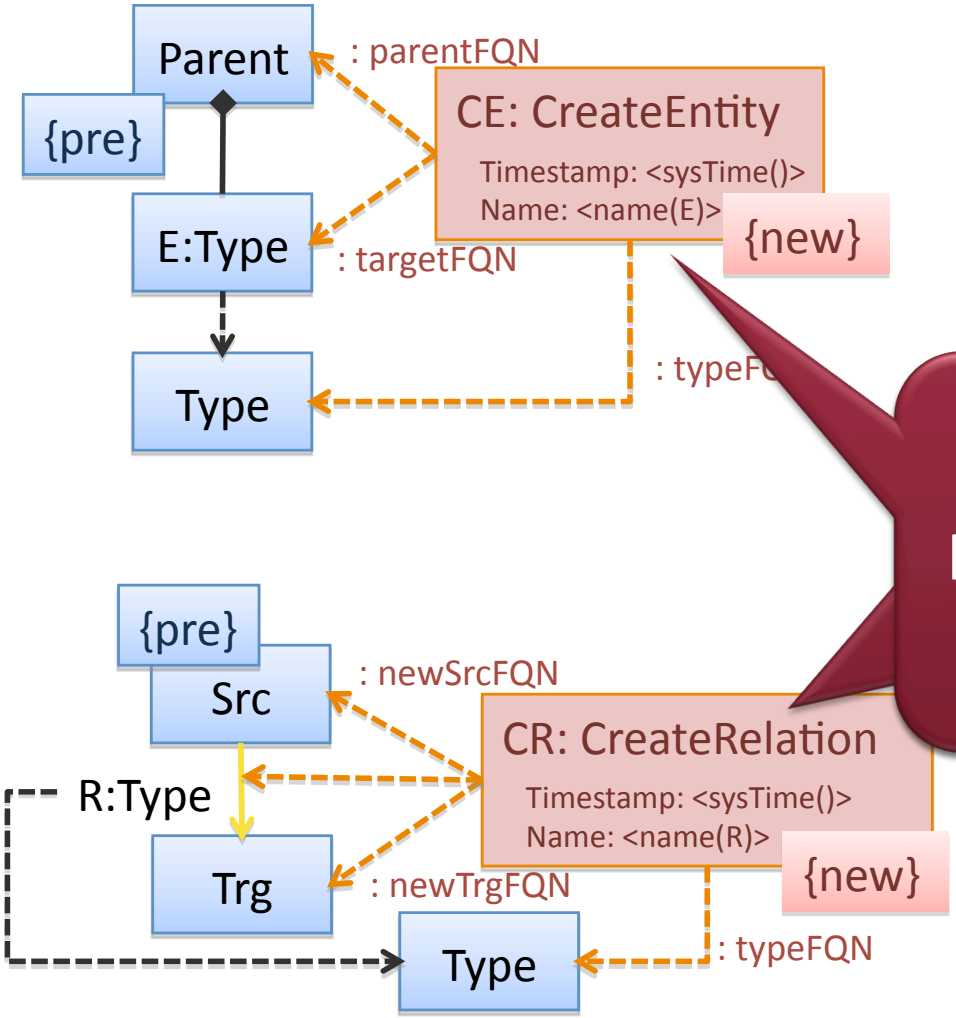


Generation of CHMs

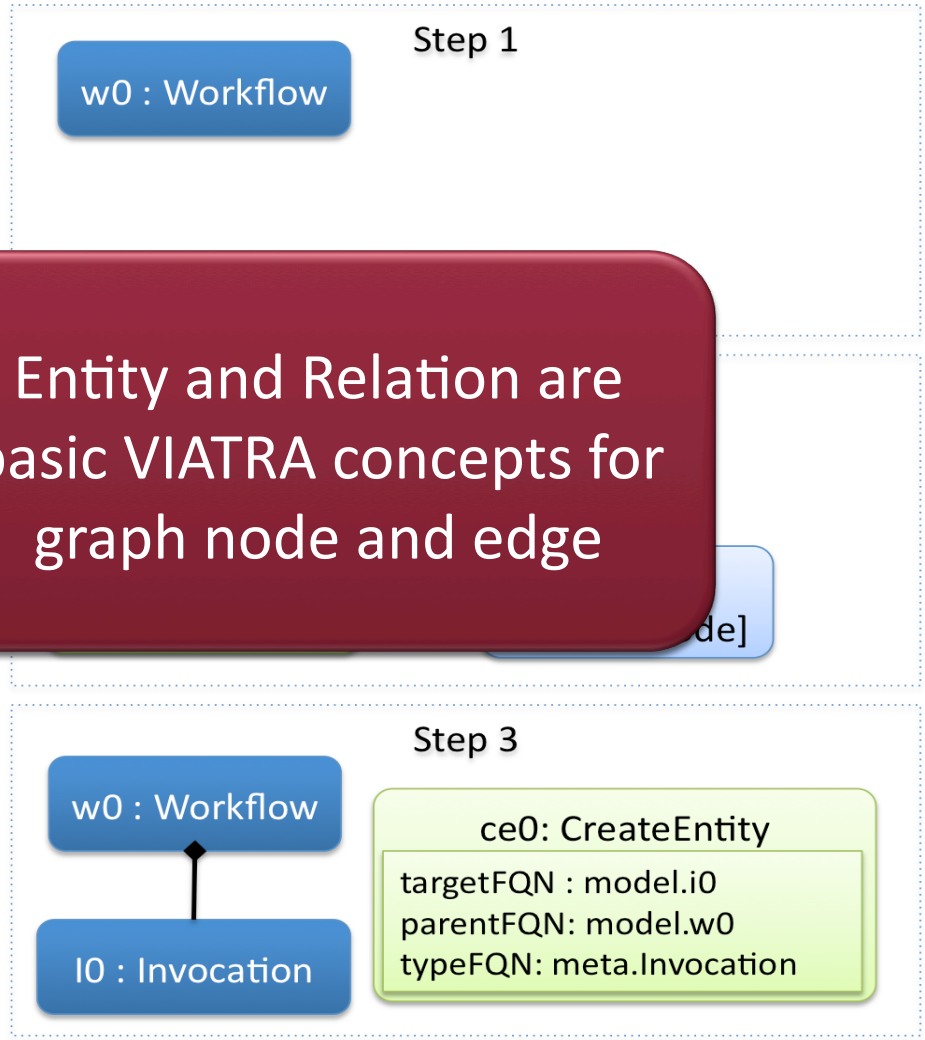


- Live transformations
  - Editor-independent!
- Generate trace model snippets as the user is editing the model
  - Timestamps
  - Contextual references

Generation of CHMs: Generic example

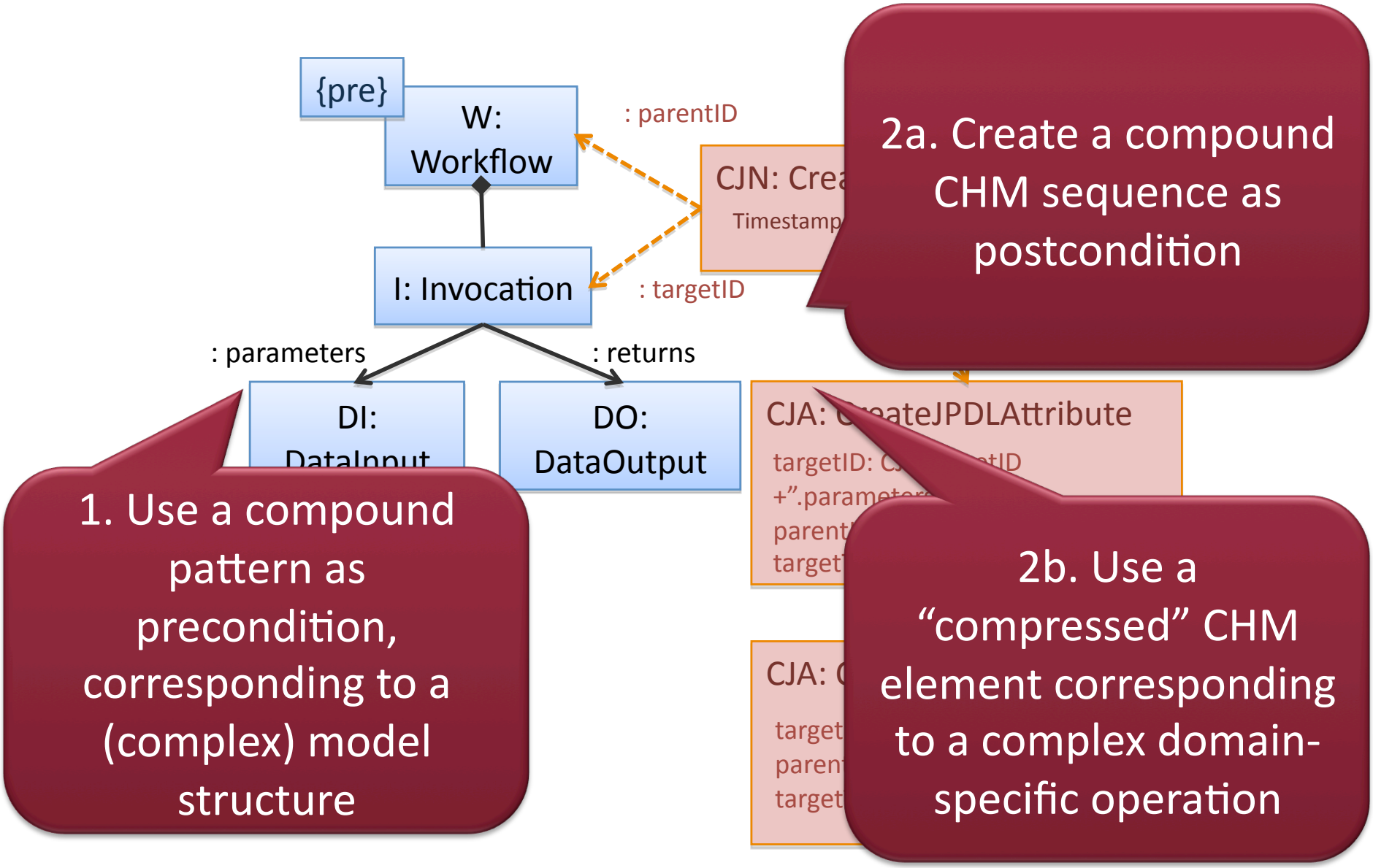


Sample execution sequence:

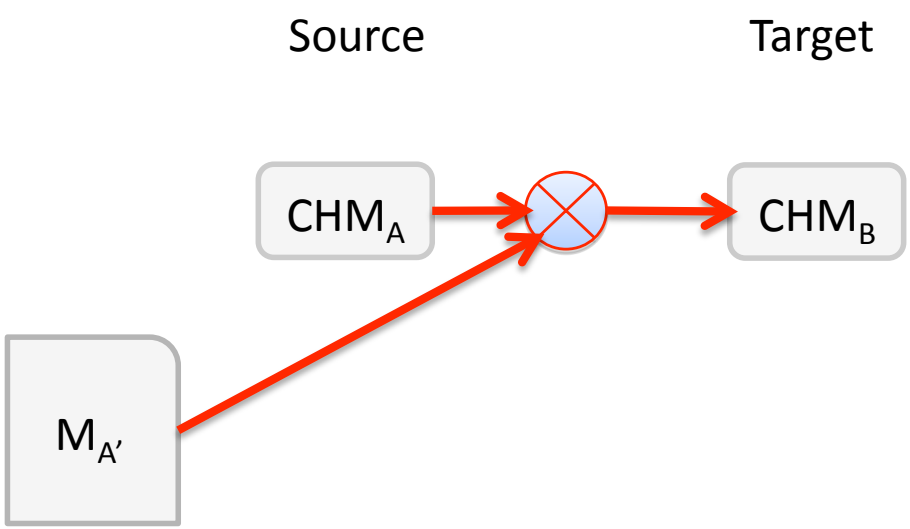


Entity and Relation are basic VIATRA concepts for graph node and edge

Generating domain-specific CHMs

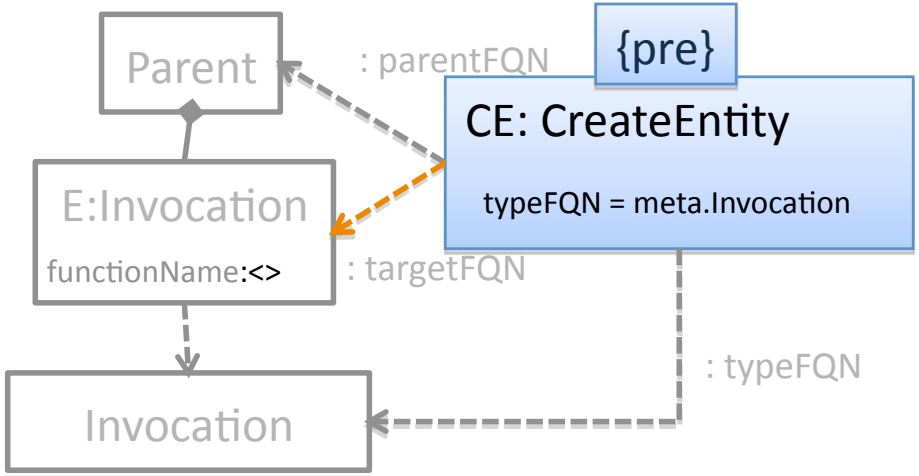


Change-driven transformations

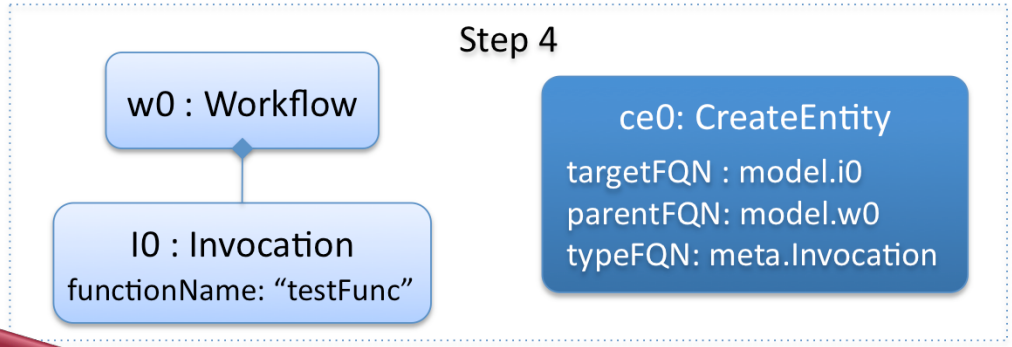


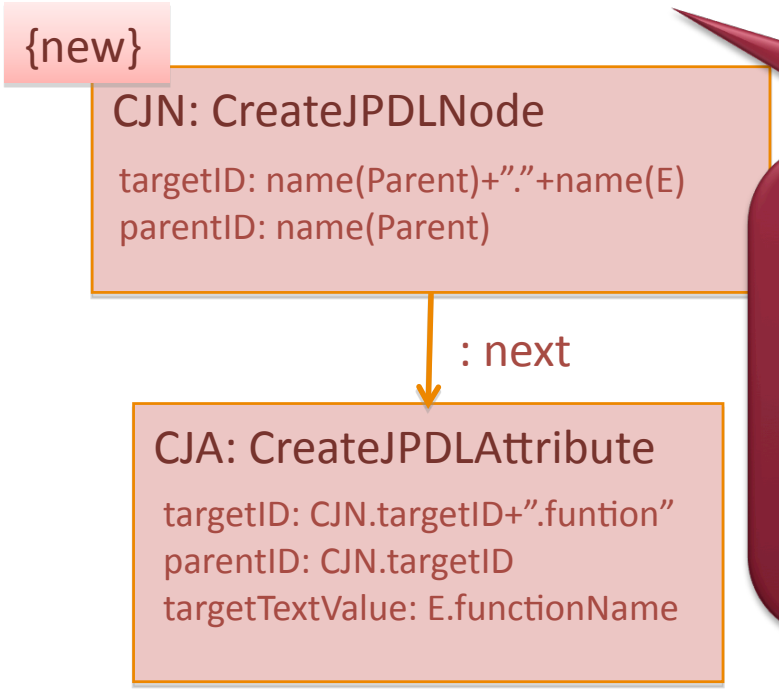
- Input:
  - Changes of the source model
- Output
  - Corresponding changes of the target model
- May be formulated as:
  - Live transformation
  - Batch transformation
- Granularity?
  - “one-to-one”
  - “n-to-m”

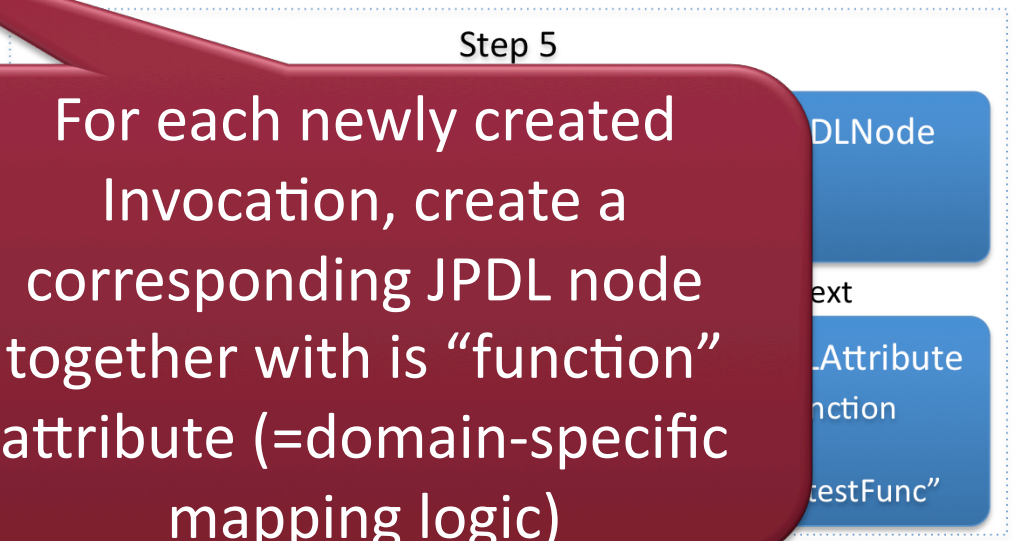
Mapping CHMs



Sample execution continued:

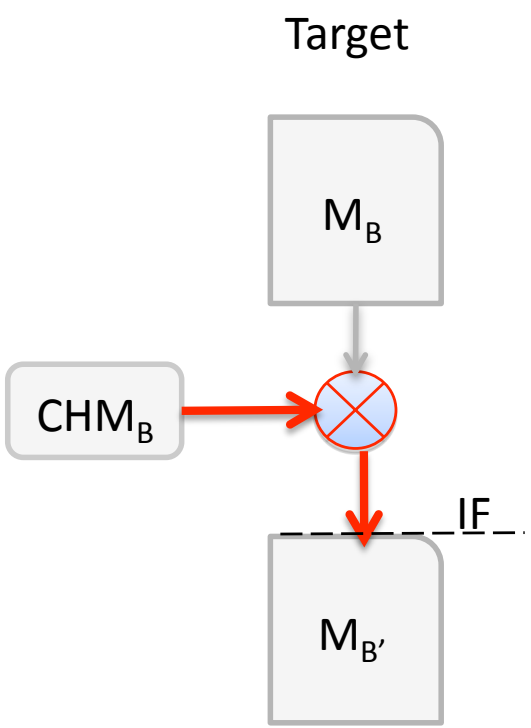






For each newly created Invocation, create a corresponding JPDL node together with is "function" attribute (=domain-specific mapping logic)

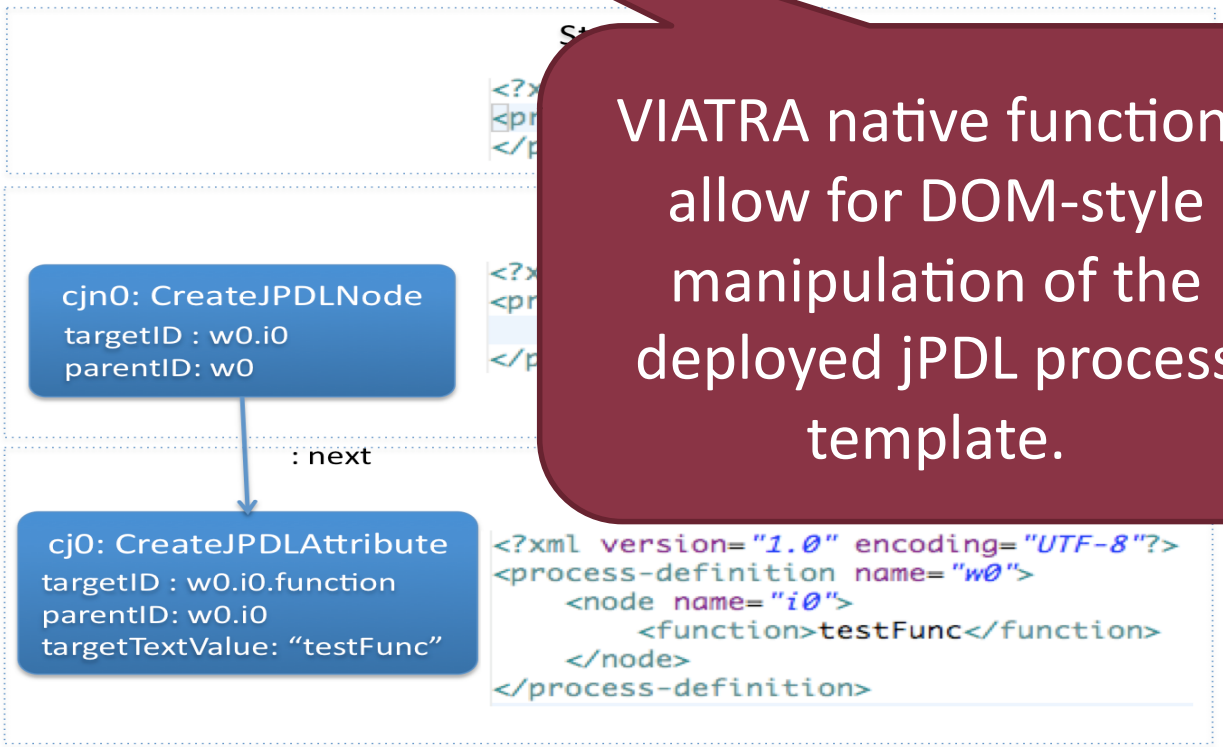
Applying CHMs to external models



- Applying CHMs = model "interpretation"
- External models are manipulated through a (service) interface
  - VIATRA: "native functions"

Manipulating non-materialized models with VIATRA

```
// create JPDL Node -- Step 7
let TargetNode = createElement(getElementById(CJN.parentID),CJN.targetID),
    Result0 = addElement(TargetNode, JPDL_DOM) in
      println("Debug created JPDL Node:"+Result0);
// create JPDL Attribute -- Step 8
let TargetAttrNode = createElement(getElementById(CJA.parentID),CJA.targetID),
    Result1 = setContent(TargetAttrNode,ref(CJN.targetFQN).functionName),
    Result2 = addElement(TargetAttrNode,JPDL_DOM) in
      println("Debug created JPDL Attribute:"+Result2);
```



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 (→ (re)verification, change impact analysis)
  - Model merging (~operational merging)
- Thank you for your attention.