A Traceability-Driven Approach to Model Transformation Testing

Nicholas D. Matragkas, Dimitrios S. Kolovos, Richard F. Paige and Thanos Zolotas
Department of Computer Science,
The University of York

Outline

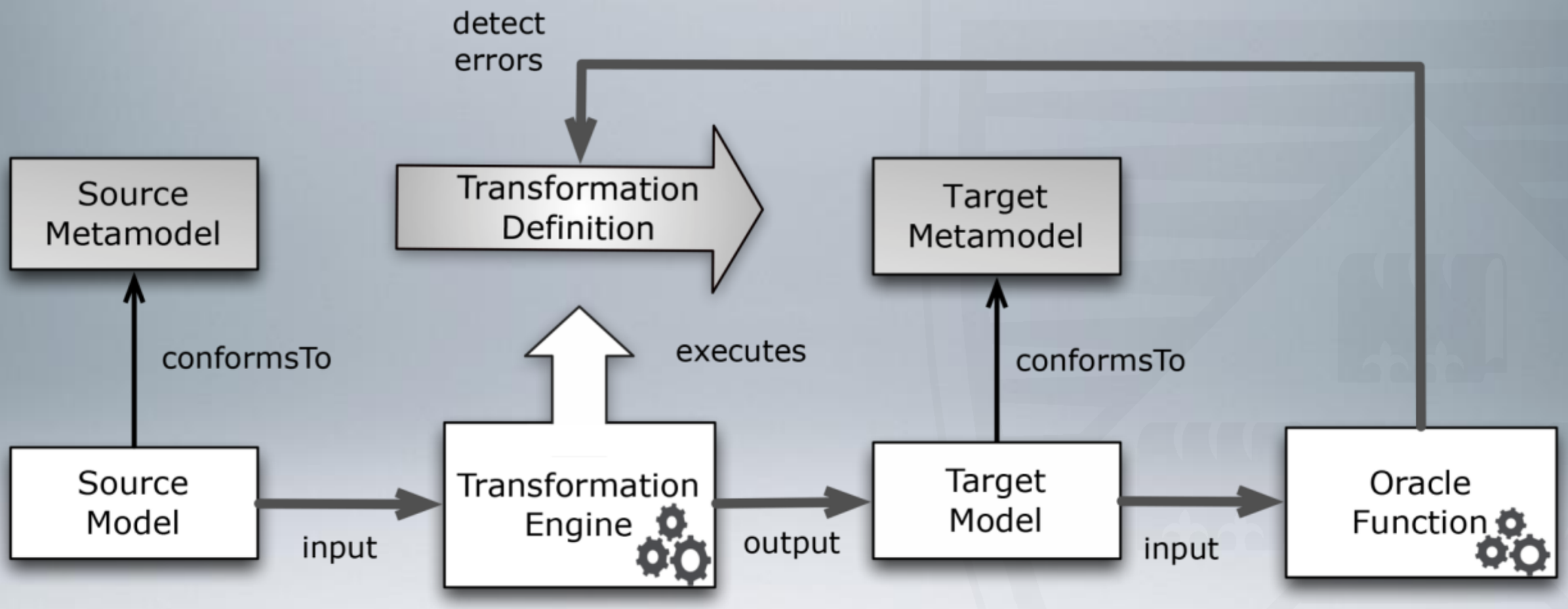
- Introduction
- Current Practice
  - Challenges
  - Motivation
- Proposed Approach
- Example
- Conclusions
- Future Work

Introduction

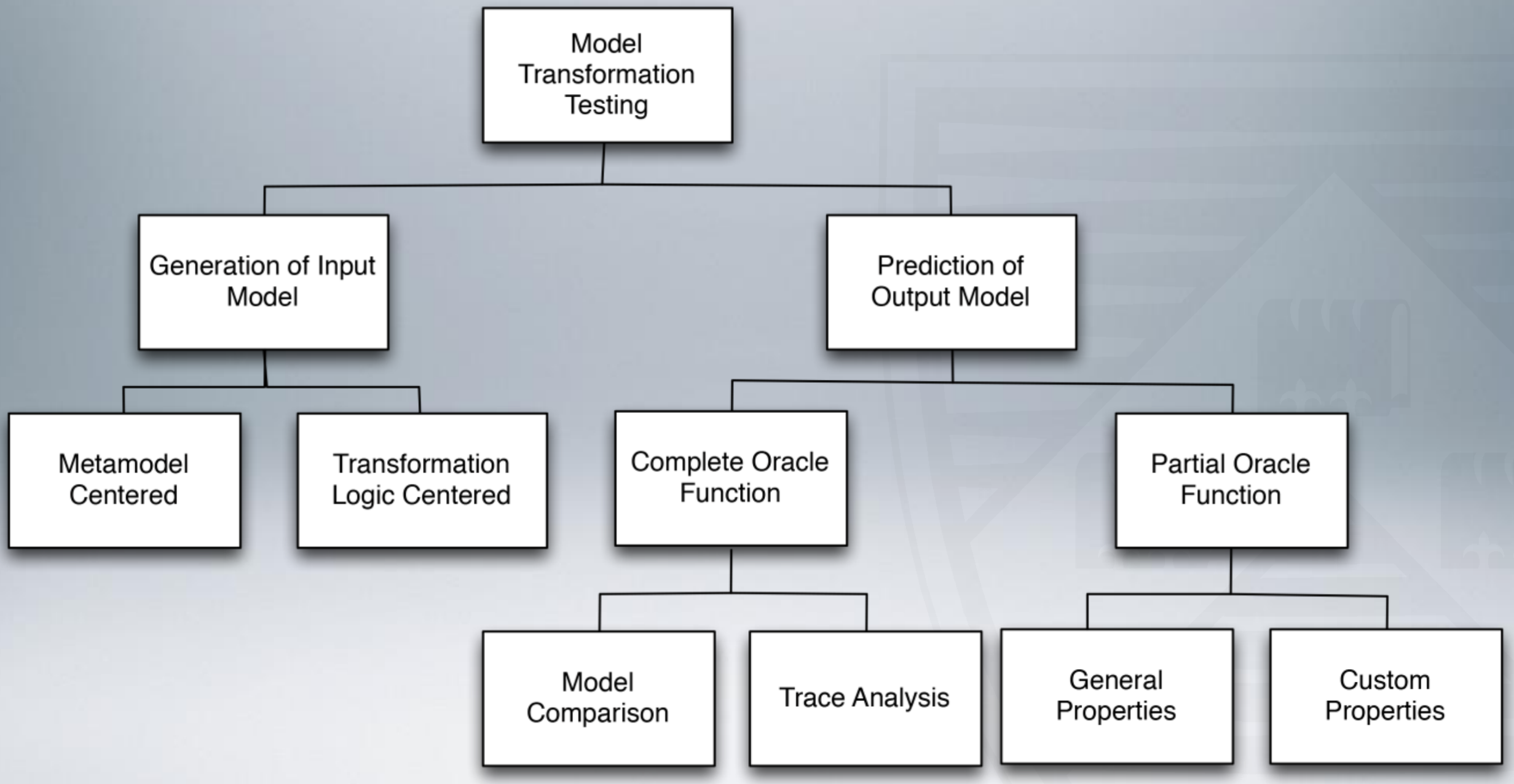
- Model transformations are the “heart” and “soul” of MDE

- Transformation testing is very important:
  - Assure the quality of model transformations
  - An error in transformations can have unpredictable results

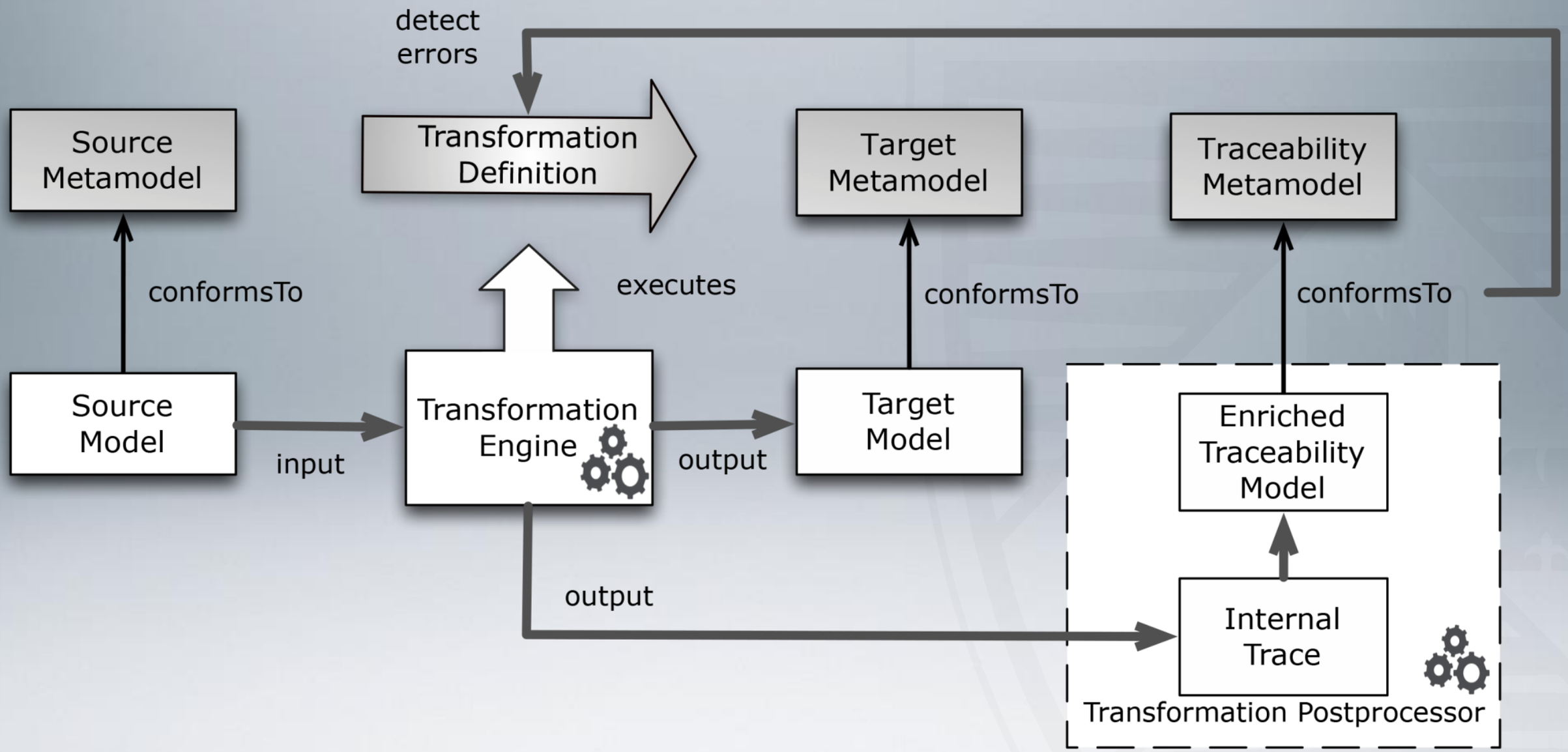
Model Transformation Testing



Classification



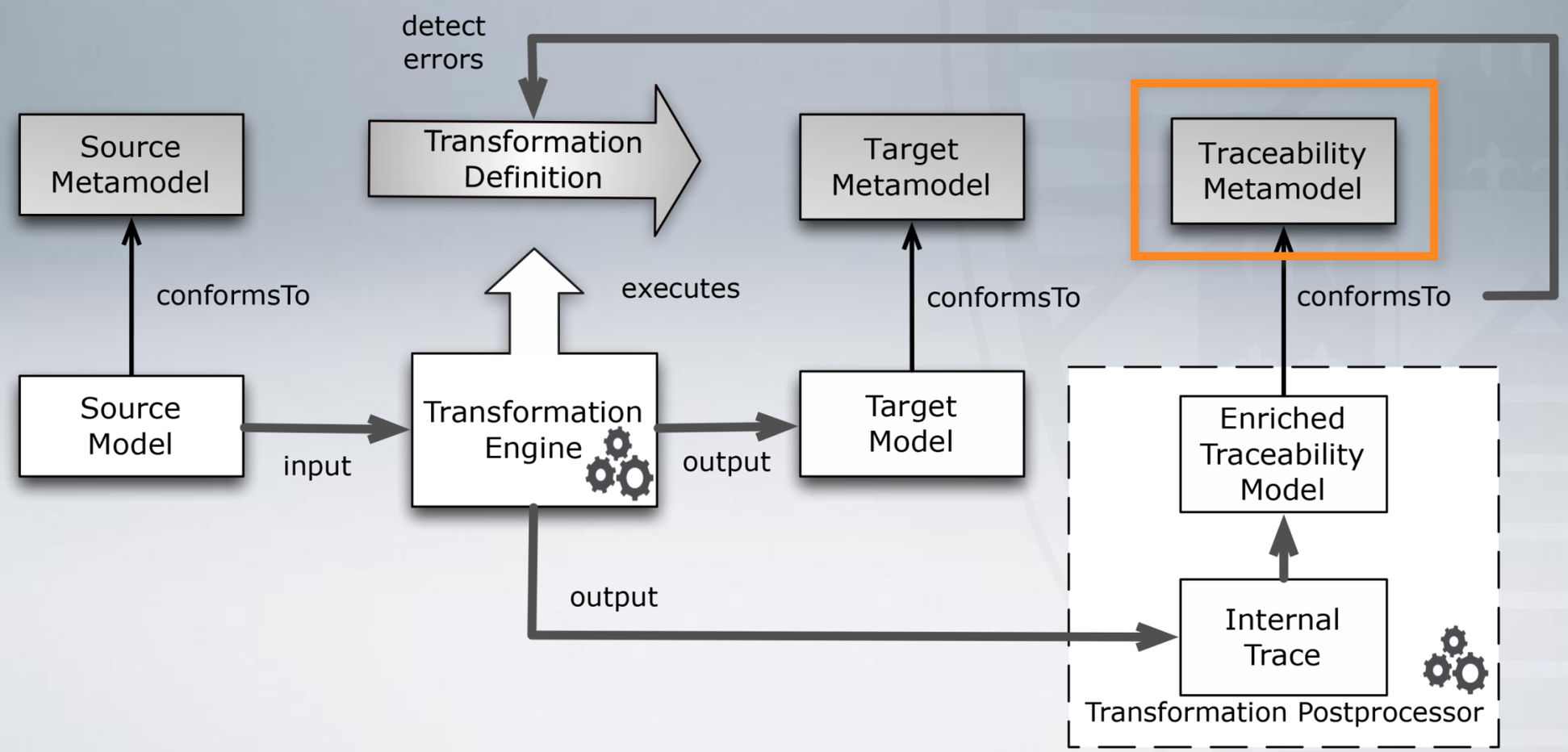
Proposed Approach



Proposed Approach

Step 1: Specify the domain-specific traceability metamodel

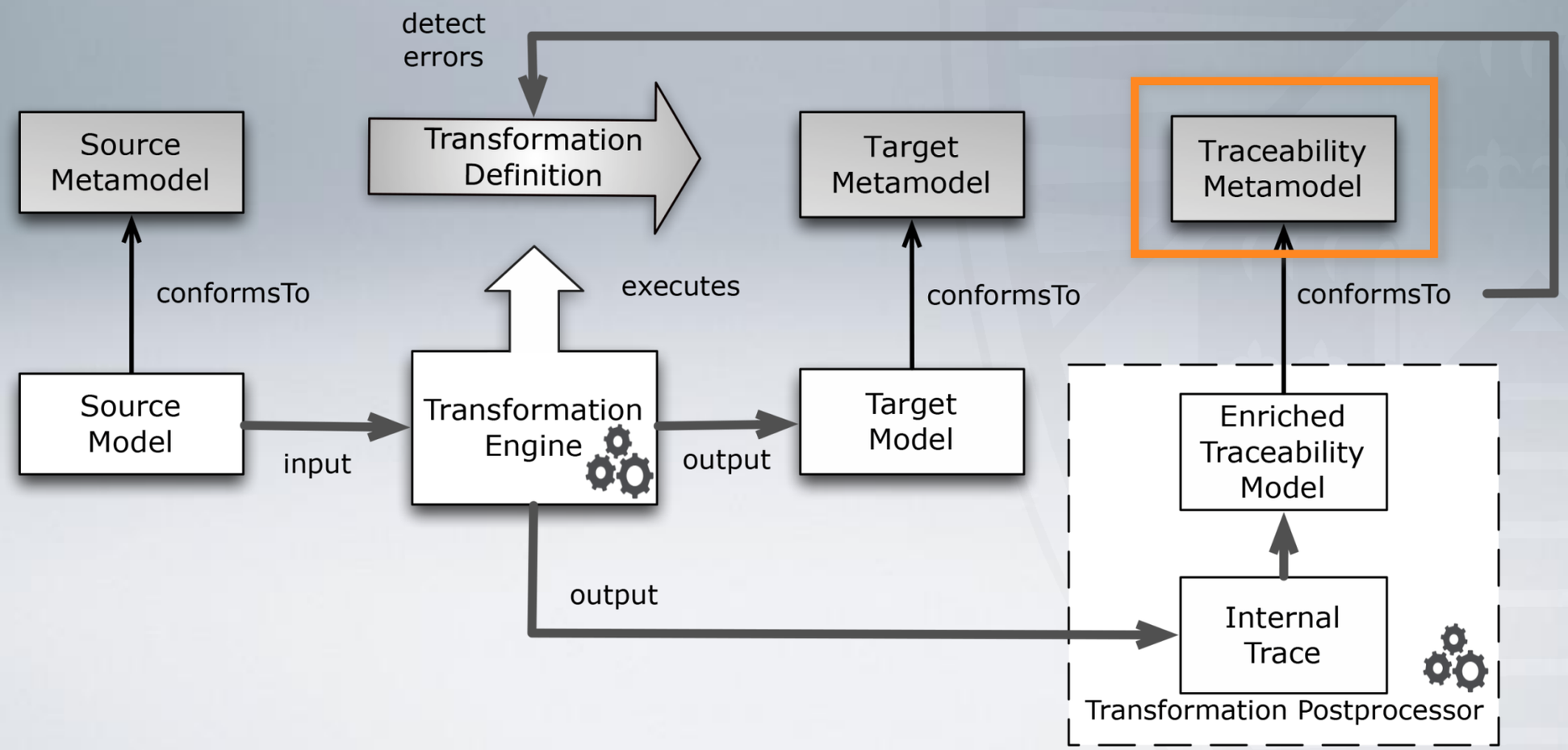
- Use different mapping languages (i.e. transML, Atlas Model Weaver, TML) to define correspondences.
  - This can be done even manually.



Proposed Approach

Step 1: Specify the domain-specific traceability metamodel

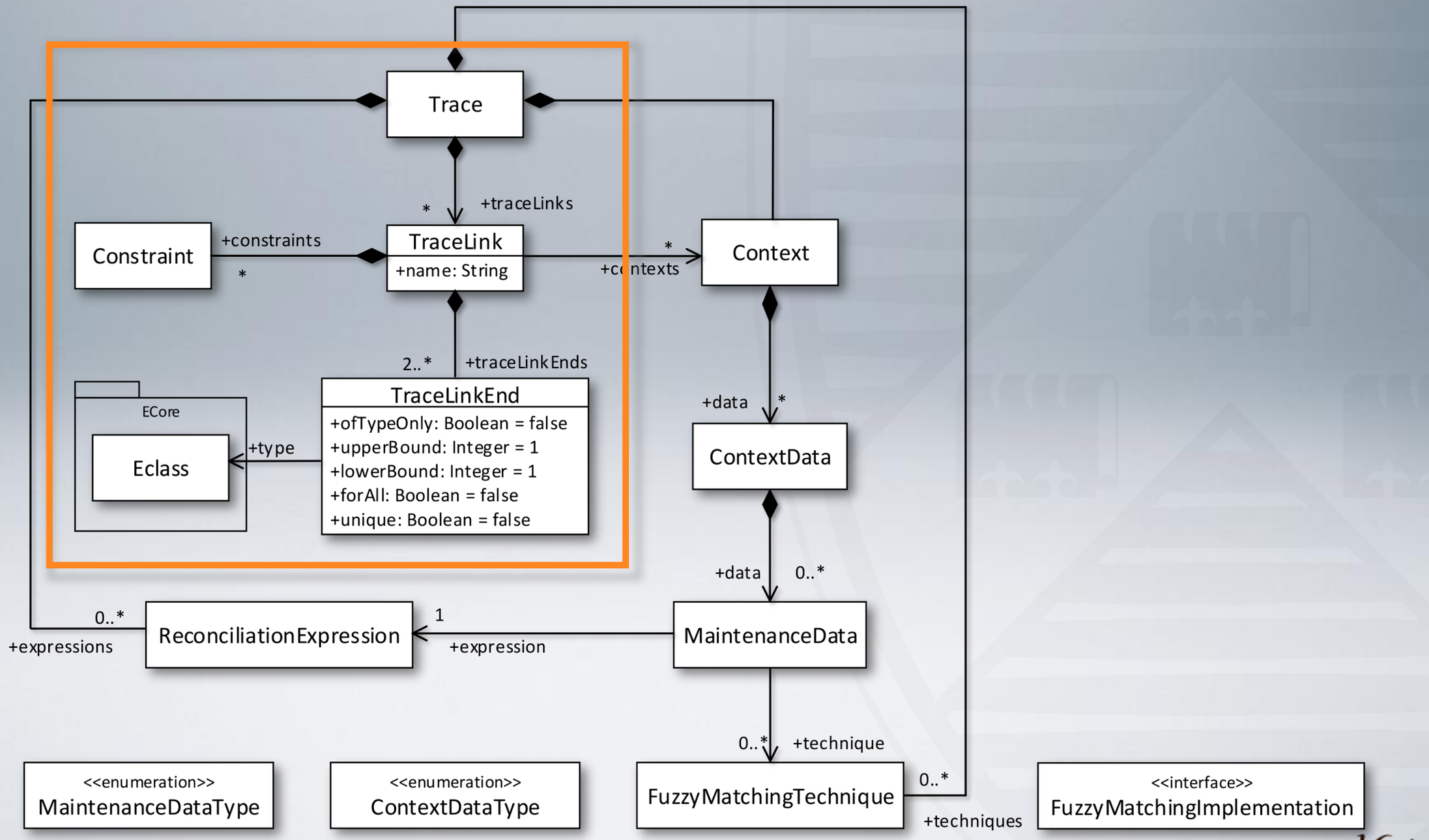
- The mappings have to be strongly typed and be accompanied by correctness constraints.



Proposed Approach

Step 1: Specify the domain-specific traceability metamodel

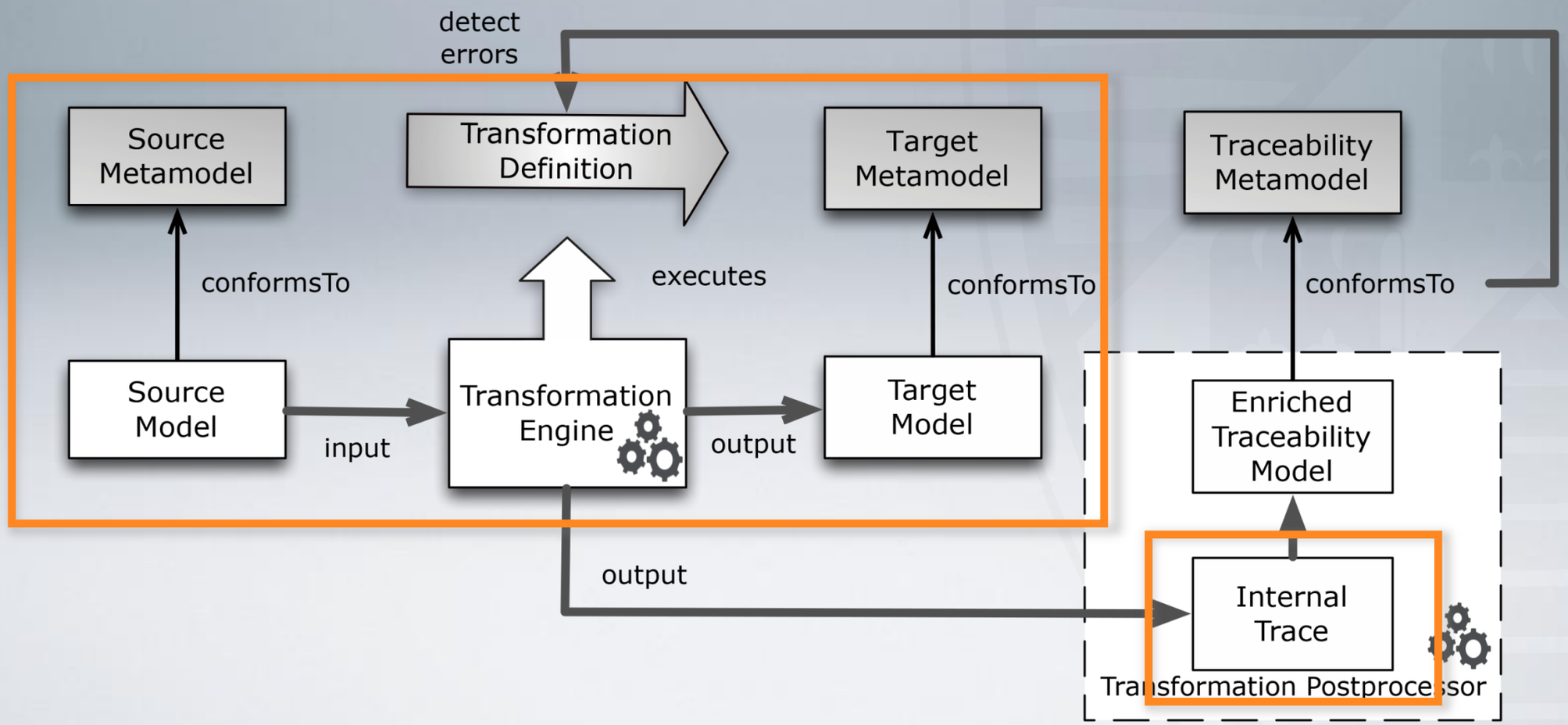
- In our current implementation we use TML.



Proposed Approach

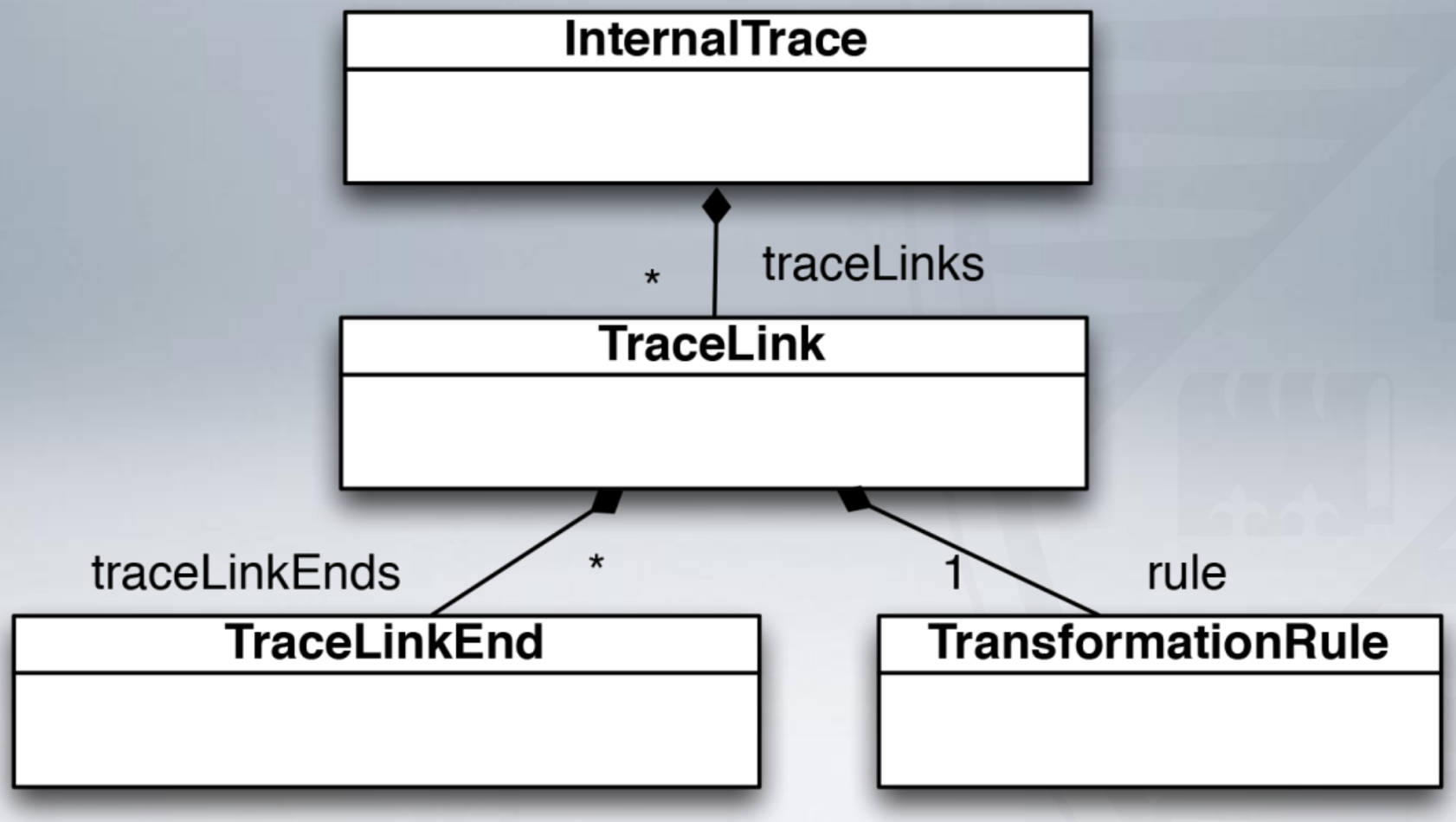
Step 2: Implement and execute the transformation over a set of input models

- Generation of the internal trace



Proposed Approach

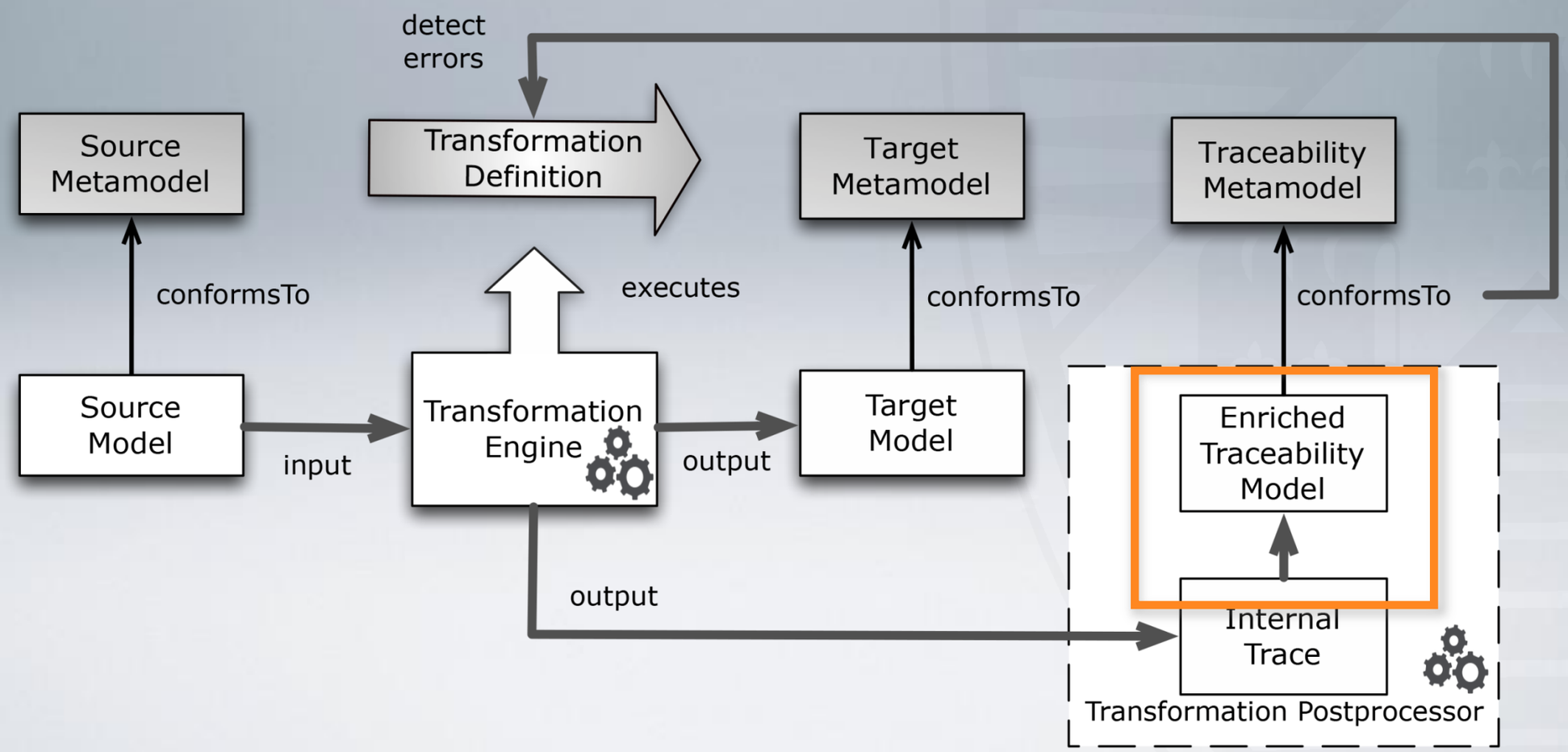
Internal Trace of ETL



Proposed Approach

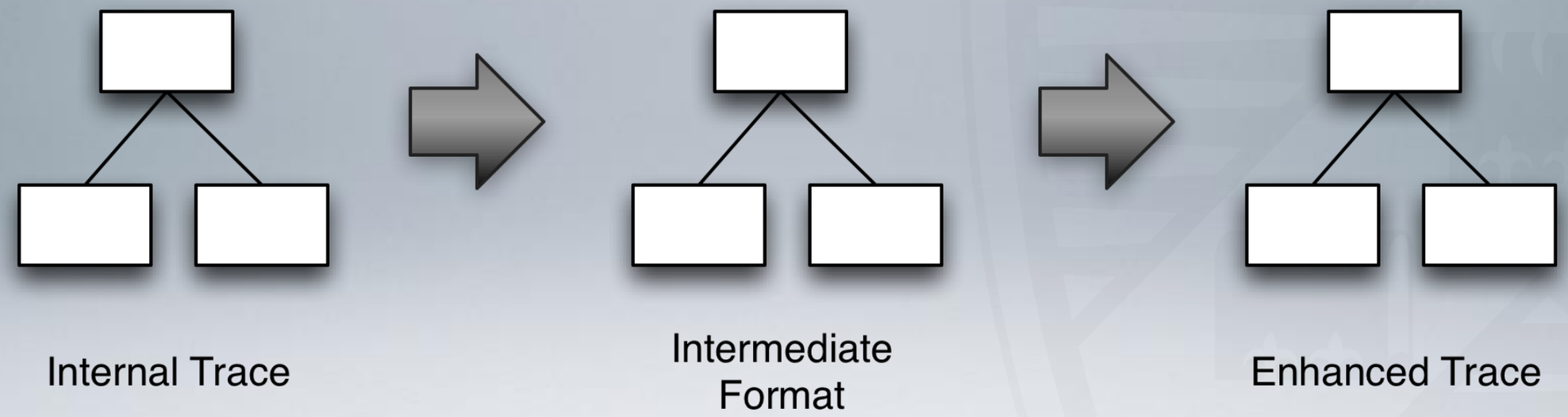
Step 3: Generate links with case-specific semantics (Enriched Traceability Model):

- Combine the Internal Trace (match link end types and cardinalities) with the Mappings specification



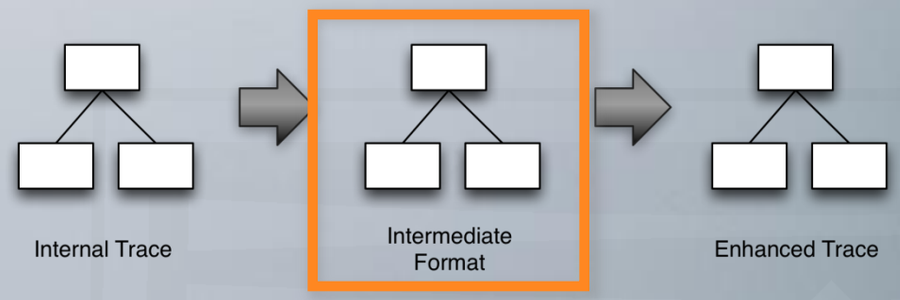
Proposed Approach

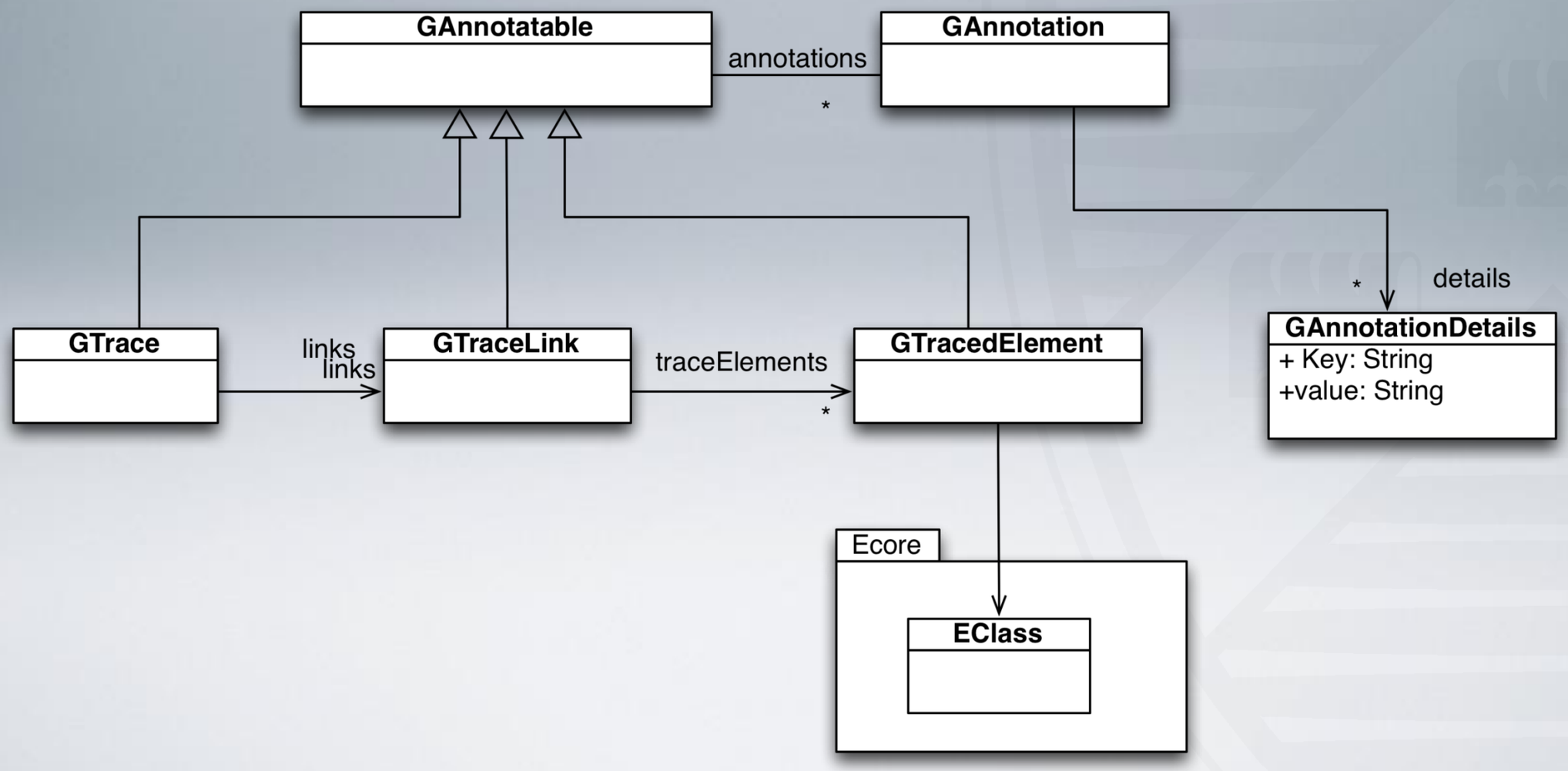
Step 3: It is a three step process



Proposed Approach

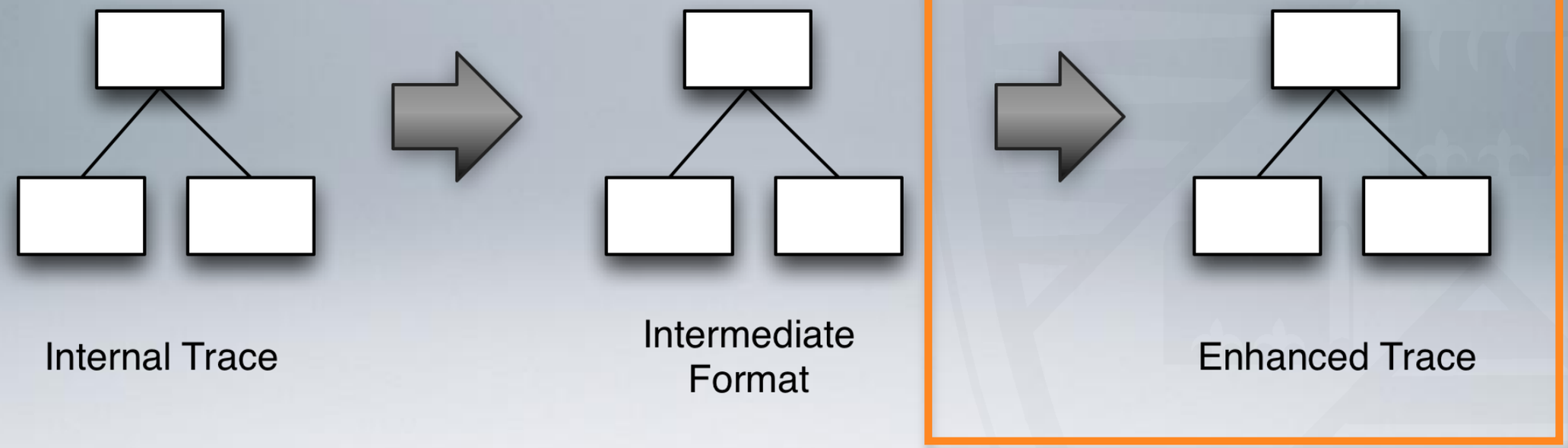
Step 3: Intermediate format





Proposed Approach

Step 3: It is a three step process



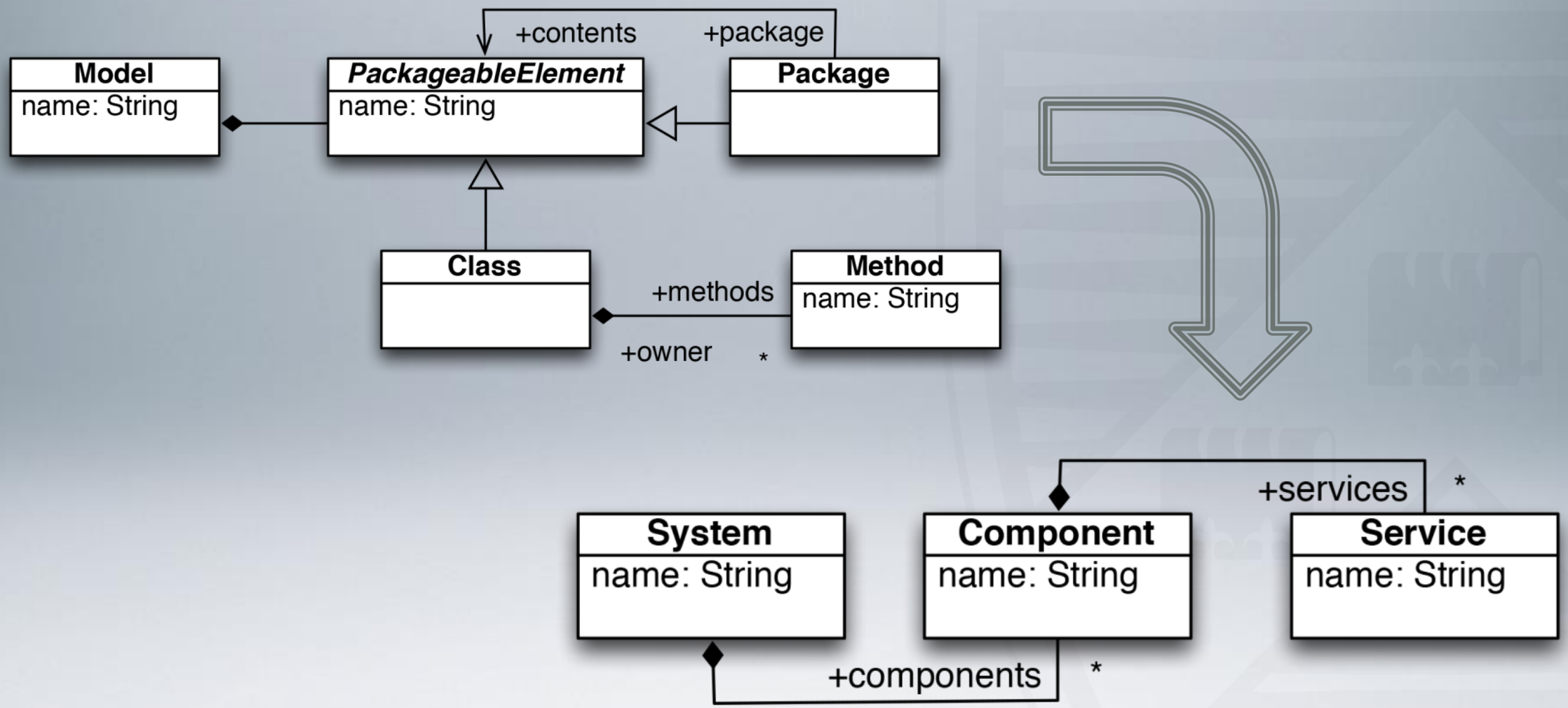
Proposed Approach

- Successful Transformation: If there are no errors in the transformation, then the Enriched Traceability Model should conform to the Traceability Metamodel

- Erroneous Transformation: If there are errors in the implementation of the transformation, then it will generate invalid mappings

Example

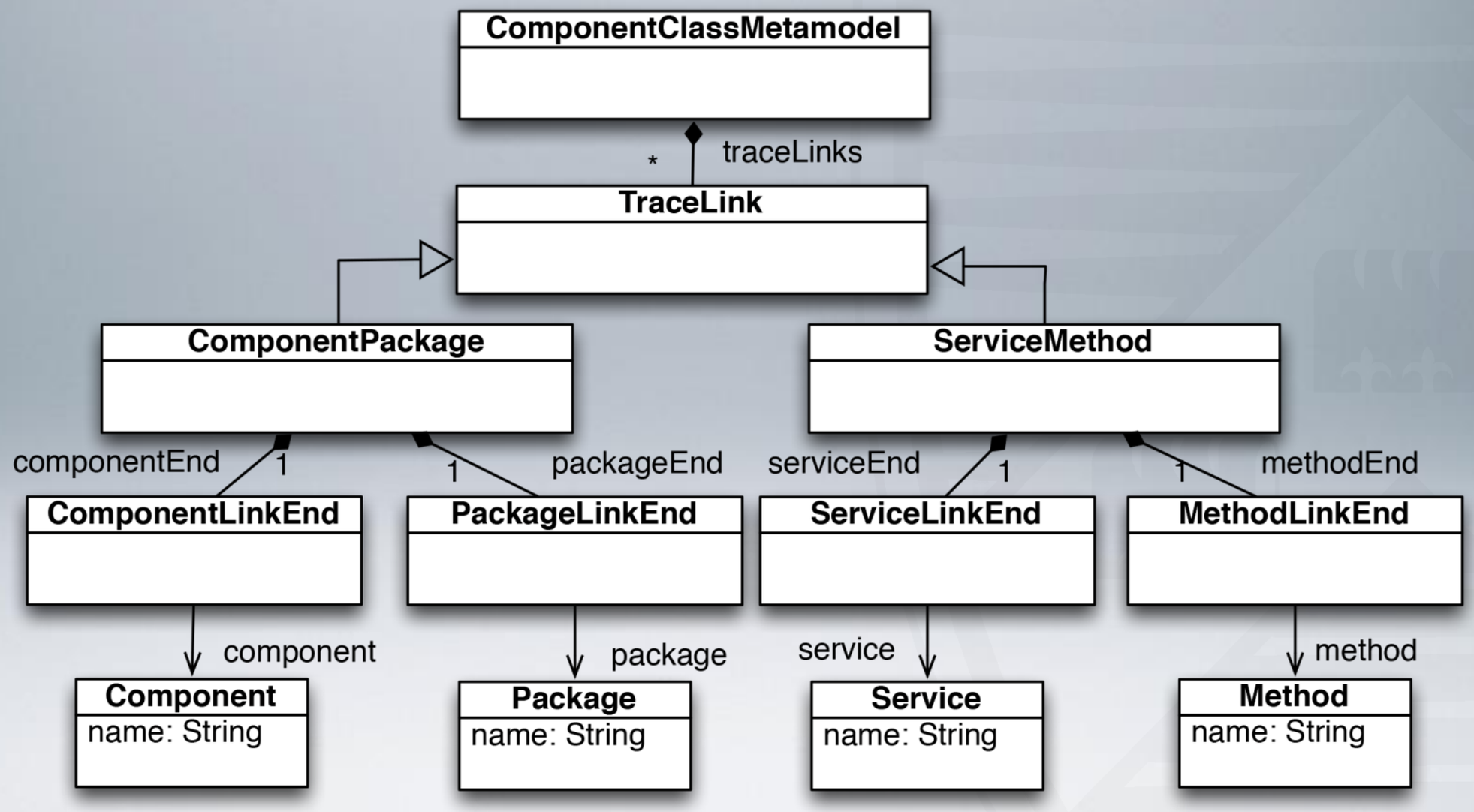
Example: Transform *Class* model to *Component* model



(C1) For each instance of Service there is exactly one instance of ServiceTraceMethodLink that links it with an instance of Method

Example

Step 1



Example

Step 2

```
@TML tracelinktype = "ComponentPackage"
rule Package2Component
transform p : ClassModel!Package to c: ComponentModel!Component {
        p.name = c.name;
}
```

```
@TML tracelinktype = "ServiceMethod"
rule Method2Service
transform m : ClassModel!Method to s: ComponentModel!Service {
        m.name = s.name;
}
```

Example

Step 3

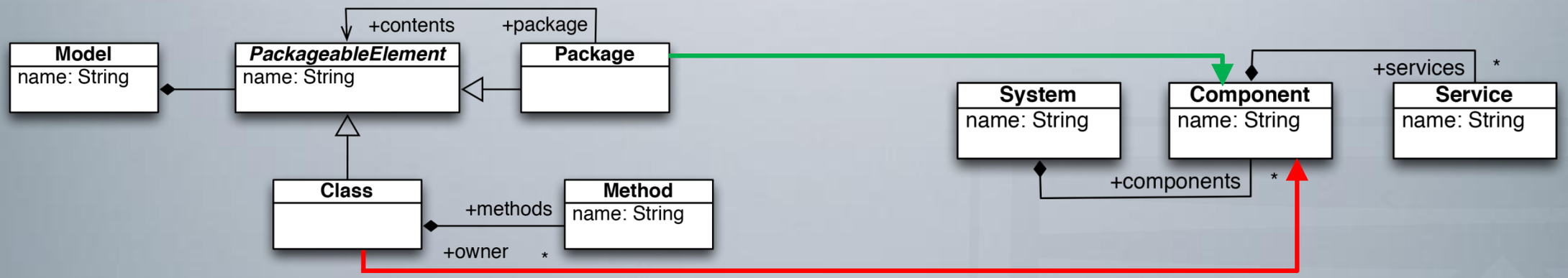
- Execute the transformation
  - If transformation is correct, then the enhanced traceability model that is generated, conforms to the Traceability Metamodel.
  - If not, then the transformation is incorrect.
  - In some cases the enhanced traceability model may conform to the metamodel but the transformation is wrong, due to the constraints.

- Examples of errors to follow

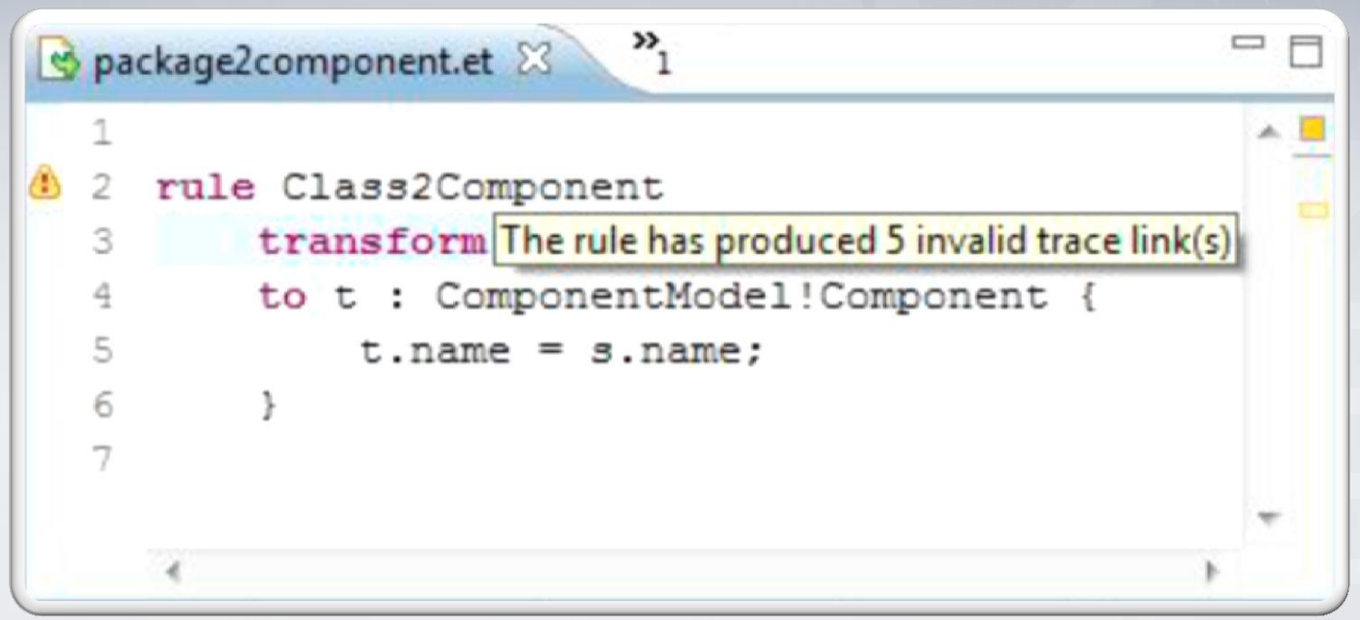
Examples of Errors

- The proposed approach detects only errors which generate invalid traces
- It can be ideally combined with other transformation testing approaches
- 3 example types of errors that the approach can detect are presented

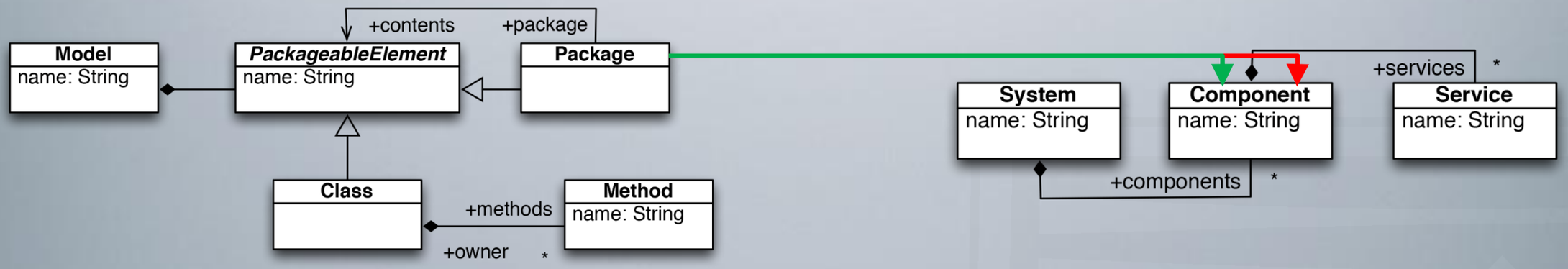
Error type I



- Engineer transforms erroneously Class to Component
- There’s no such link type, the transformation post-processor generates a warning



Error type II



- Engineer transforms erroneously one instance of Package to two of Component (wrong cardinalities)

- There’s a 1-to-1 relationship for this type of links in the Traceabilitiy Metamodel so a warning is generated

Error type III

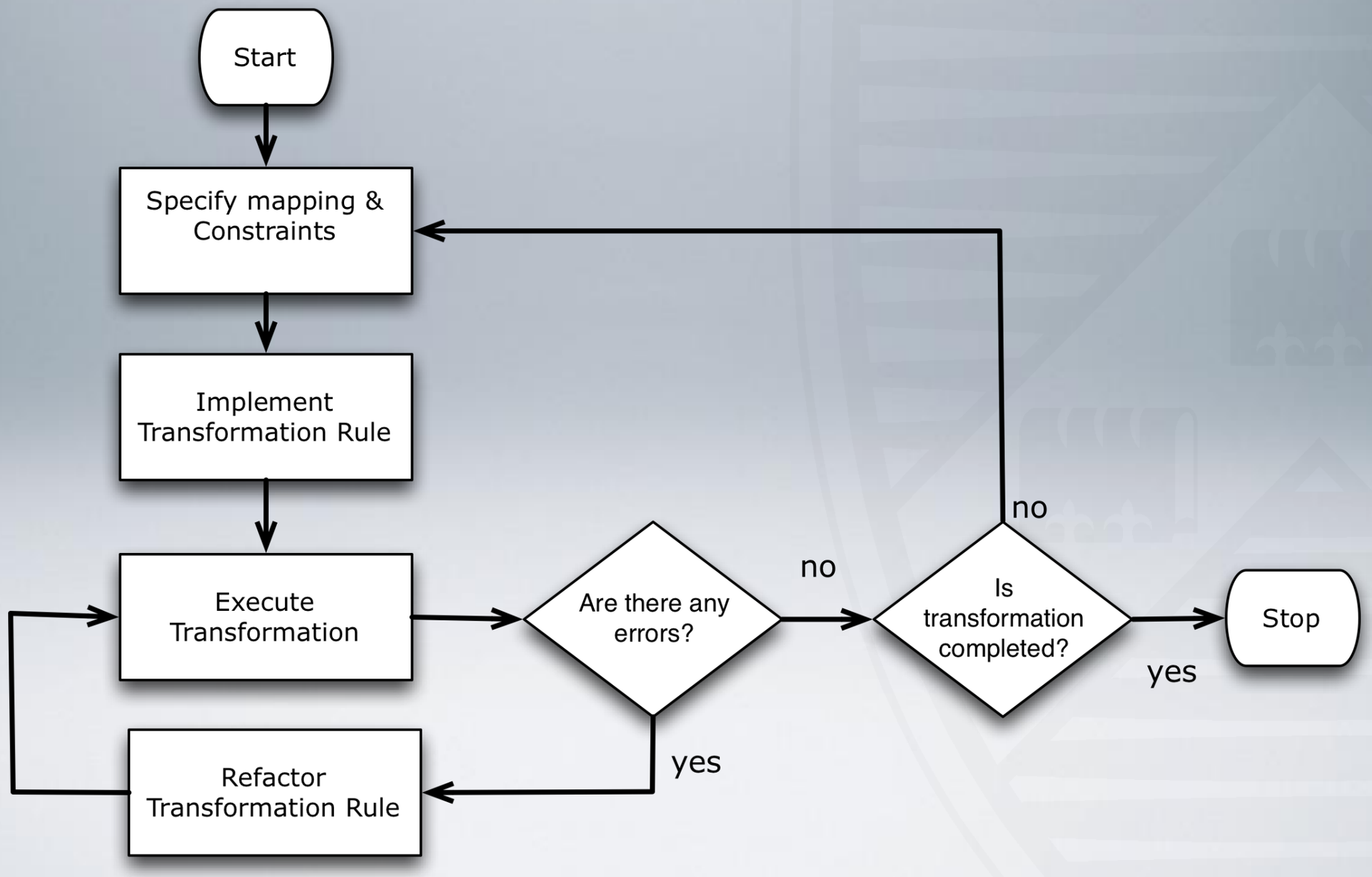
- Erroneous rules which create mappings, that conform to the traceability metamodel but they violate the correctness constraints

```
context Package {
  constraint OneForEachPackage{
    check : Package2ComponentTraceLink.all.exists(e|e.Package.target =
        self)
    message : 'No links of type Package2Component found for Package ' +
        self
  }
}
```

```
rule Class2Component
transform s : ClassModel!Package to t : ComponentModel!Component {
    guard : s.contents.select(c|c.isTypeOf(Class)).size()>0
    t.name = s.name;
  }
```

Incremental Development

- Incremental co-development of the transformation and its specification in a test-driven manner



Conclusions

- Proposal of a specification-conformance checking approach
  - relies on the traceability information to identify erroneous rules
  - support for hybrid transformation languages, which generate internal traces
- The incremental manner → decreased complexity
  - Transformation specifications and implementations can be done in more than one iterations.

Future Work

- Integration with other model transformation languages
  - e.g. ATL
- Investigate applicability to other model transformation paradigms
- Check how integration with other testing techniques improves the testing results

Questions