Requirements Engineering for BX

Richard Paige

Dept of Computer Science, University of York

@richpaige

Contents

- Why consider requirements for transformations and BX?
- What are requirements for BX?
- Requirements engineering processes for BX.
- Requirements specification languages for BX.

Motivation

- A number of BX languages and tools have been proposed (see last lecture).
  - Each tool has its own characteristics, strengths, weaknesses, etc.
- In a number of transformation applications, the requirements for BX are “discovered” late.
  - Sometimes requiring substantial rework!
- A better understanding of requirements engineering for BX can help mapping tools/ approaches to problems, and mapping problems to BX.

Selected Literature

R. Eramo et al, “Representing uncertainty in bidirectional transformations”

R. Eramo et al, “Towards a taxonomy for bidirectional transformations”

N. Macedo et al, “Least-change bidirectional model transformations with QVT-R and ATL”

S. Hidaka et al, “Feature-based classification of bidirectional transformation approaches”

E. Guerra et al, “Engineering model transformations with *trans*ML”

S. Tehrani et al, “Requirements engineering in model transformation development”

Requirements and BX

General Questions

- What are some general questions to be addressed when we engineer a BX?
  1. What needs to be transformed into what?
  2. What mechanisms can be used for building the BX? (includes theory, tools, techniques)
  3. What are the application domains for the BX?
  4. What are the specific characteristics of the BX (e.g, what patterns are appropriate to use)?
  5. What are the quality requirements (eg., performance) for the BX?
  6. What are the success criteria for the BX?

General Properties for BX

- Size: is the BX small (e.g., a single reversible refactoring) or large (e.g., a reversible codegen)?
- Level of automation: fully automated, human-in-the-loop?
- Visualisation: how is the BX, its results, and its input presented to users?
- Level of industry application
- Maturity level: is there a tool, is it theoretical?

Functional Requirements

- Define what a BX must, should or could provide.
- Some examples.
- Correctness:
  - when the BX is run in the forward direction, the target model must be well formed
  - when the BX is run in the reverse, the source model must be well formed
  - (well formedness usually defined in terms of conformance to metamodel/constraints evaluate to true).

Functional Requirements

- Inconsistency tolerance:
  - Should be able to support incomplete or inconsistent artefacts (e.g., temporarily inconsistent models)
- Modularity: should provide the ability to compose transformations into new ones.
- Traceability: should support generation of links (correspondence model) between source and target models
  - as well as between the steps of a transformation process (chaining)

Functional Requirements

- Change propagation: should provide support for propagating changes in a model from one direction to another.
- Incrementality: should support the ability to update target models based on only the changes made in the source models.
- Uniqueness: could support the ability to generate a unique solution to a BX problem (cf JTL).

Functional Requirements

- Termination: should support the definition of terminating transformation executions.
- Mechanisms/styles: must support a transformation style, i.e., declarative, operational or hybrid.
  - Will vary in terms of what they make implicit, e.g., navigation of source model, creation of target model, order of rule execution, etc

Non-Functional Requirements

- Specifies criteria with which we can judge the quality of a BX.
- Examples.
  - Extensibility and modifiability
  - Usability (tricky!)
  - Robustness: can the BX manage invalid models and deal with corresponding errors?
  - Conciseness
  - Interoperability: connection with other (especially non-MDE) tools
  - Verifiability and validity.

Requirements Engineering Processes for BX

Key Concepts

- What are the typical stages of a RE process for BX?
- What are the key artefacts that are involved?
- Who are the stakeholders?
- What problems may arise?
- What techniques can be applied?

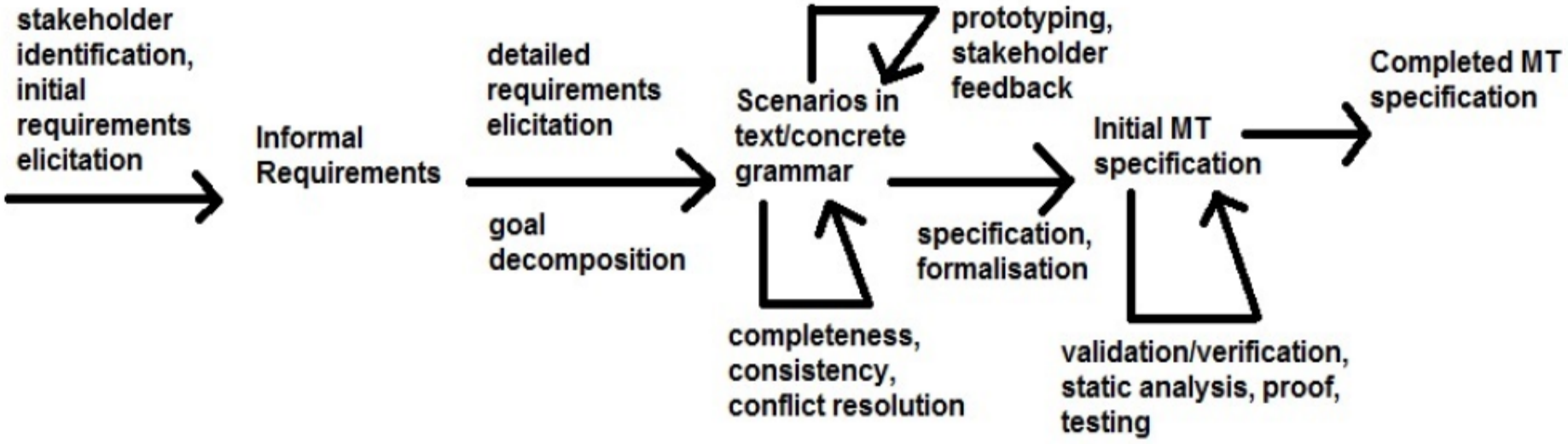
Typical RE Stages

- Domain analysis and elicitation:
  - Who are your stakeholders?
  - Gather information from users, customers, etc on the system domain and system requirements.
- Evaluation and negotiation:
  - Identify imprecision, conflicts, omissions and redundancies in these “informal” requirements
  - Resolve these (if possible and appropriate) via negotiation and consultation

Typical RE Stages

- Specification:
  - Document the formal requirements in a specification (more later)
  - Often the basis for a contract between developers and customers
- Validation & Verification:
  - Check the specification for consistency, completeness and acceptability to stakeholders

RE Processes for BX



- From Tehrani et al, ICMT 2016

RE Processes for BX

- Previous diagram: a typical RE process.
- Points of note:
  - Use of BX scenarios as a concrete mechanism for driving the development of a specification.
  - Distinction between *local* and *global* requirements.
  - Local requirements: mapping, rewriting, defining correspondences
  - Global requirements: properties of an entire model, e.g., a measure of complexity is reduced by running a BX, performance obligations, information hiding.

BX Elicitation

- Observation/ethnographic methods: observe current (possibly) manual BX process.
  - E.g., consistency between an Excel spreadsheet and a SysML requirements diagram.
- Unstructured interviews: ask open-ended questions about domain, current BX process.
  - Useful for transformation goals, e.g., “ensure refactorings are applied to both source and target within 10ms”, “preserve performance properties of models”

BX Elicitation

- Questions for unstructured interviews?
  - Size range of source/target models? (what kind of infrastructure should be supported? – led us to use NoSQL for a couple of projects)
  - Formats for encodings of models and BX? (explored binary encodings instead of text/XMI)
  - Assumptions about source/target models? (e.g., always available, being incrementally updated)
    - Who checks that assumptions are met?
  - Read, write, read-write?
  - Confidentiality restrictions?

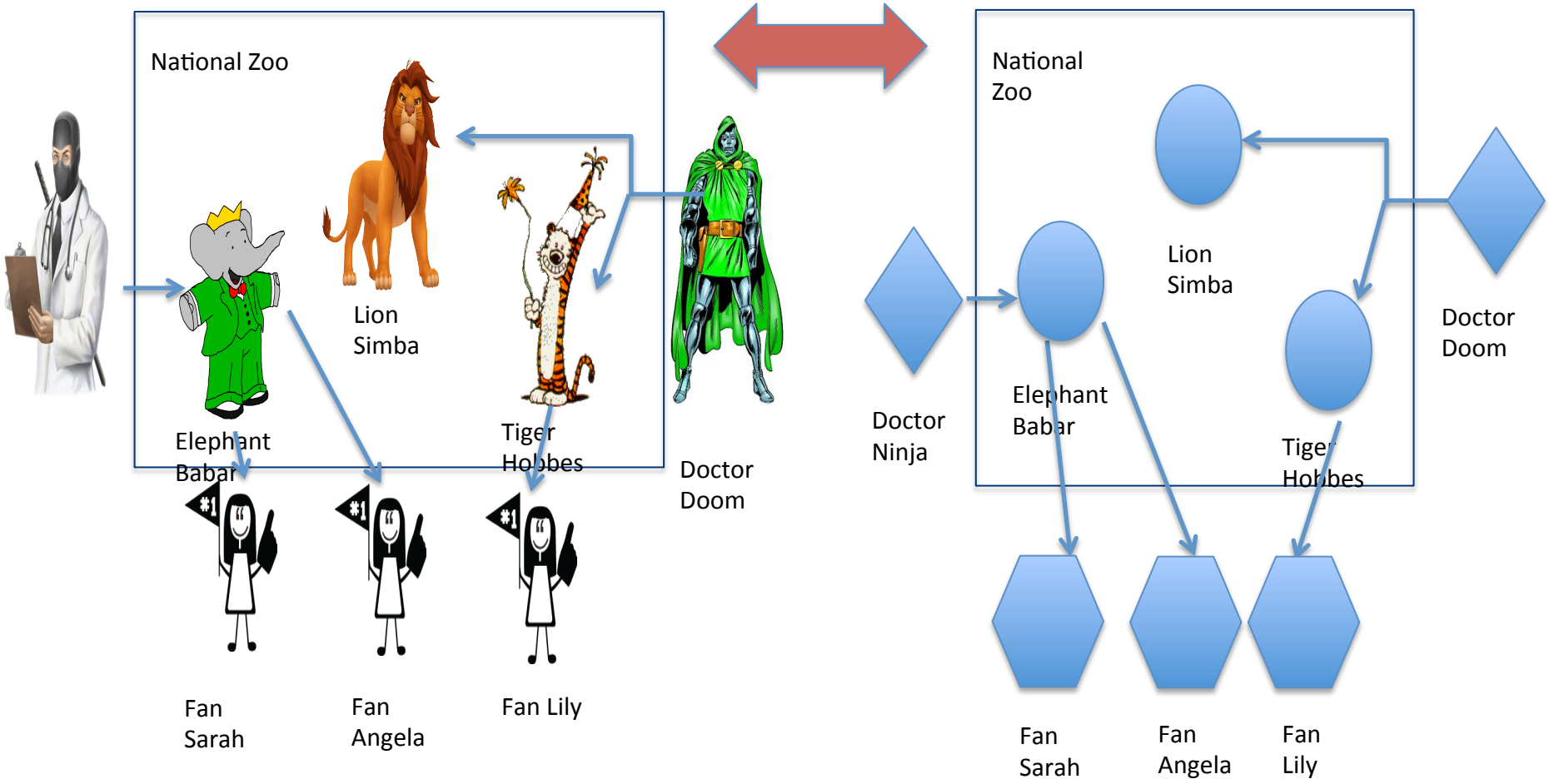
BX Elicitation

- Structured interviews:
  - Preloaded questions about the domain and BX.
  - Can focus on a checklist based around a requirements pattern catalogue.
  - Examples:
    - Global functional requirements: hippocraticness, synchronisation, semantic preservation, completeness (i.e., are all entities and language features covered?)
    - Local non-functional requirements: specific rule satisfies a time bound

BX Elicitation

- Scenario-based analysis:
  - Scenarios used to capture different required transformation processing cases.
  - Can use a concrete scenario language (e.g., CNL) with sketches of sample models.
  - Example: refactor object-oriented design
    - define a success measure to improve OO structure, e.g., increase cohesion.
    - then decompose the scenario into specific cases addressing individual examples of poor structure, where specific update transformations can be applied

Another BX Elicitation Example



BX Evaluation

- Techniques:
  - Prototyping (how easy is this with our current BX tools?)
  - Goal-oriented analysis (link between goal-based approaches and BX???)
  - Further scenarios (testing) for specific poorly understood corner cases

BX Requirements Specification

- Wide range of techniques, including use of controlled natural language, UML, OCL.
- We will explore the *transML* family of languages shortly.
  - Also worth looking at DSL-Maps, see paper by Pescador at ASE'16.
- Supplementing with natural language can be very useful.

BX Requirements V&V

- More details later, but general techniques include:
  - BX requirements inspection
  - Testing
  - Checklists
  - Static analysis
  - Proof/model checking
- Example: a requirement for semantic property preservation
  - can be refined into a set of checks that individual BX rules maintain an invariant.

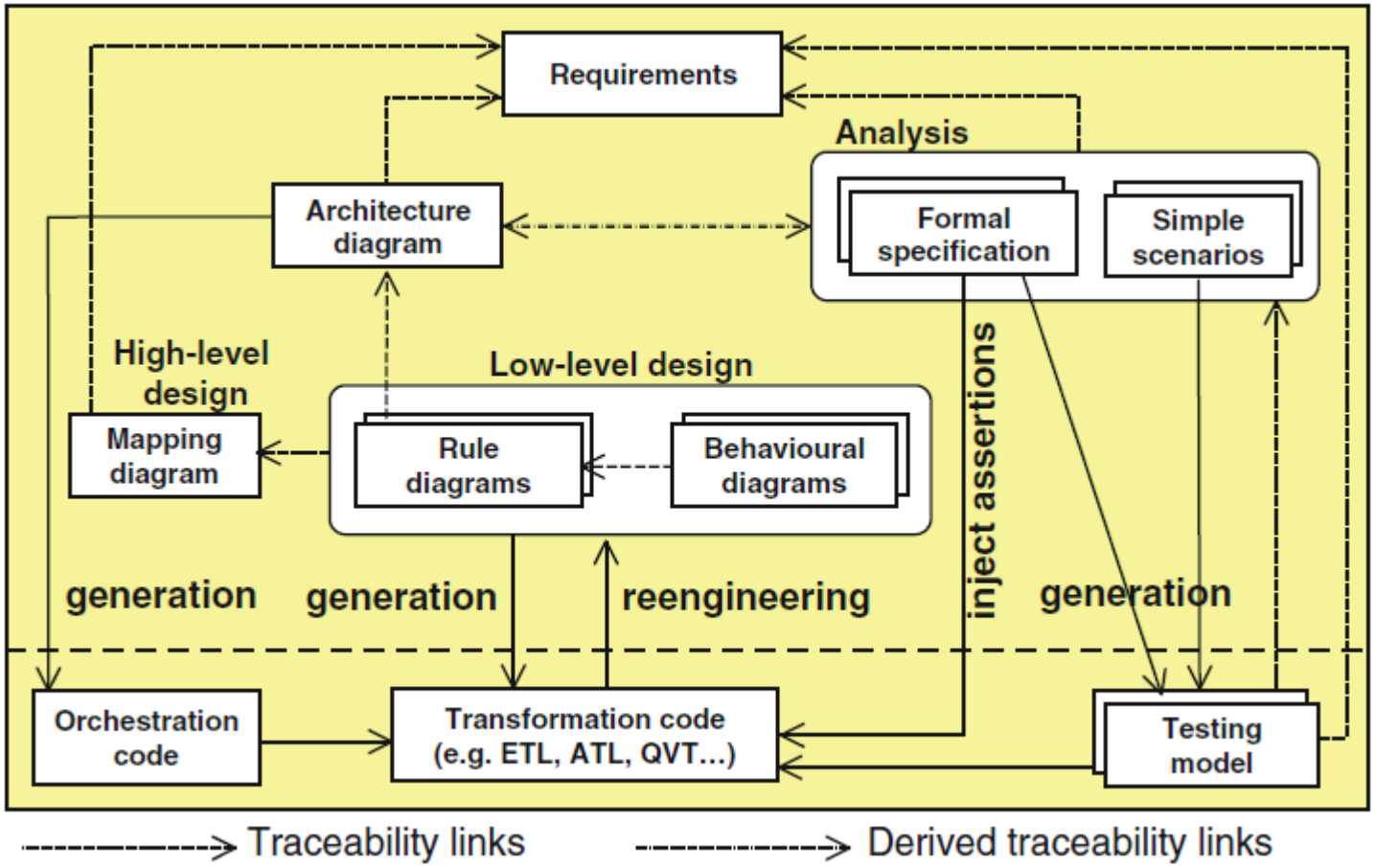
BX Requirements Problems

- What problems might we encounter in RE for BX?
  - Unrealistic requirements: address this by frequent short iterations
  - Changing requirements: check requirements at the start of each iteration; proper contracting.
  - Conflicts: capture trade-offs and negotiate
  - Uncertainty: identify, resolve, refine, negotiate
- Anything new under the sun? (Ecclesiastes 1.9)

*trans*ML

- A family of languages to support the lifecycle of transformation development.
  - Not just BX.
- Can be used with any transformation implementation language.
  - Experience of QVTo, EOL, ETL, ATL.
- Here we focus on the requirements support, later on architecture, design and testing.

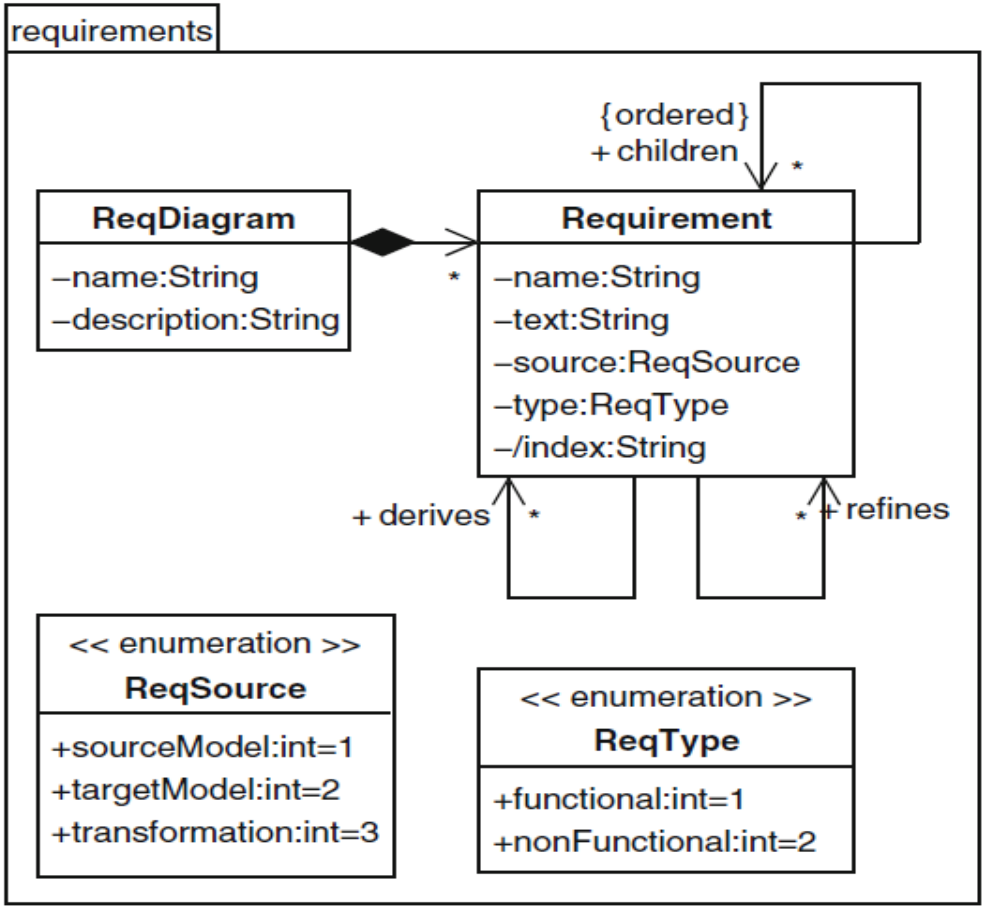
*trans*ML


Requirements
Analysis
Architecture diagram
Formal specification
Simple scenarios
High-level design
Low-level design
Mapping diagram
Rule diagrams
Behavioural diagrams
inject assertions
generation
generation
reengineering
generation
Orchestration code
Transformation code (e.g. ETL, ATL, QVT...)
Testing model
Traceability links
Derived traceability links

*trans*ML Requirements

- Can use any of the aforementioned RE techniques for elicitation and negotiation.
- *trans*ML includes a diagram representation of (BX) requirements.
  - To support forward traceability.
  - Based on SysML requirement diagrams.

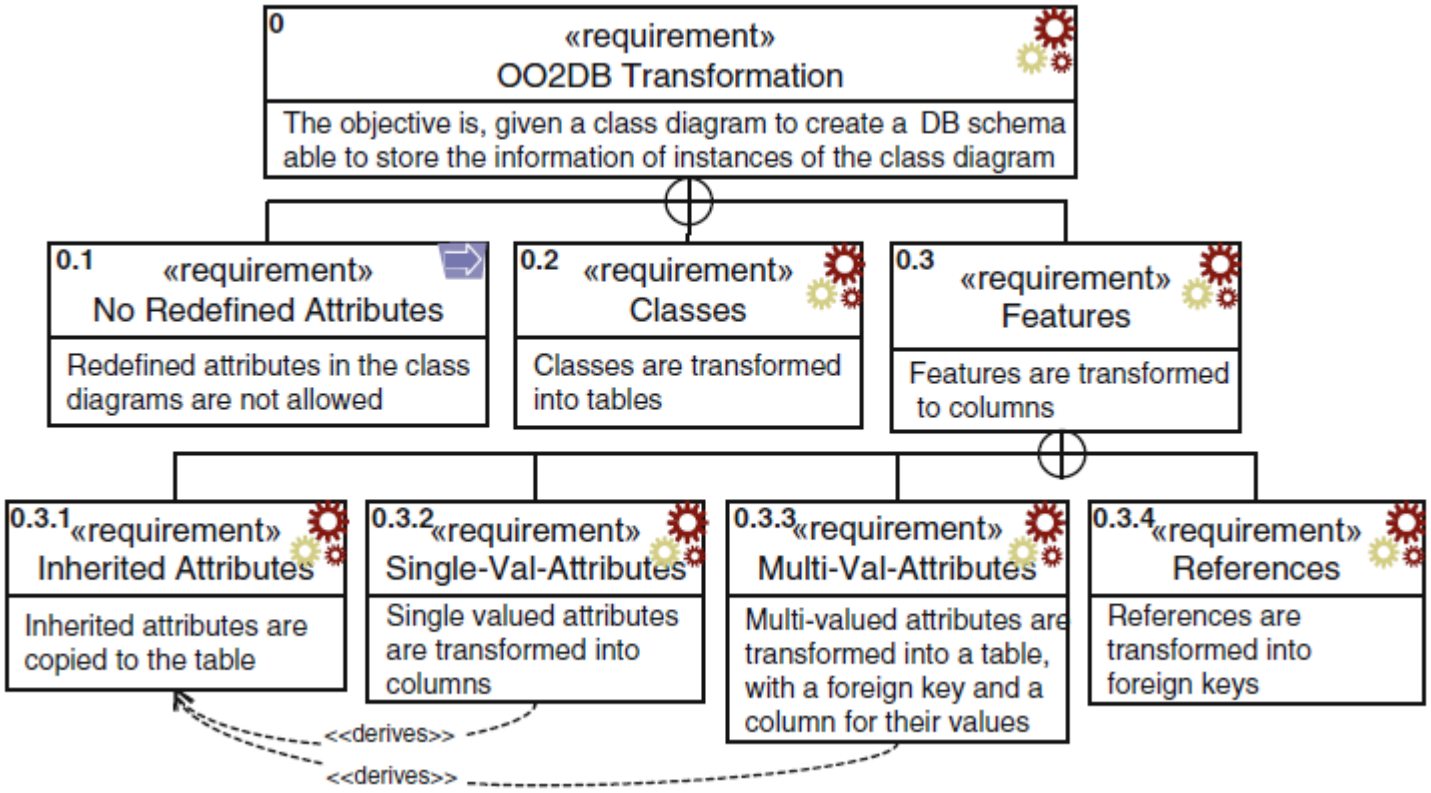
*trans*ML Requirements Metamodel



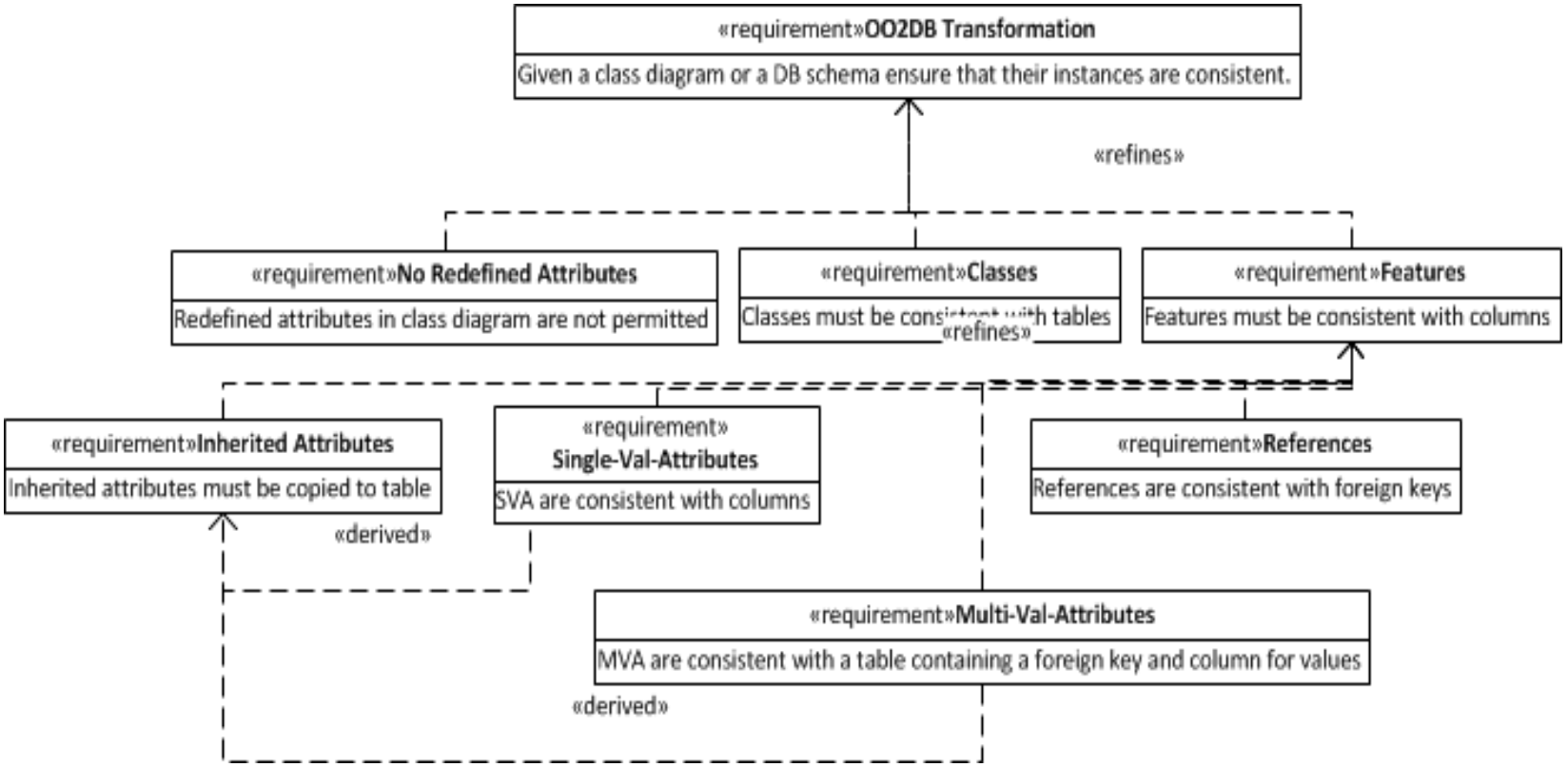
*trans*ML Requirements Metamodel

- Explicitly supports hierarchical decomposition, classification, refinement and traceability.
  - Classification is *dual*: are they functional/not? Are they requirements of {input model, output model, transformation itself}.
  - Not all valid input instances need to be processed or regenerated.
  - Not all valid output instances need to be generated or in scope.

*trans*ML Example Model



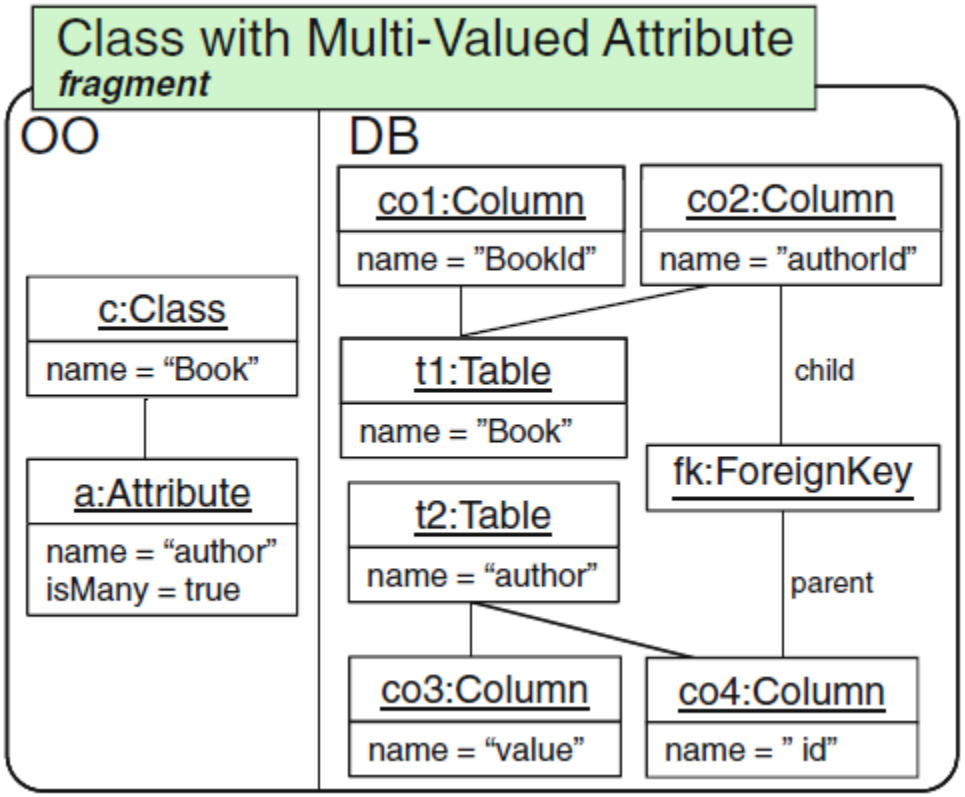
*trans*ML Example Model



*trans*ML Scenarios

- Saw scenarios earlier: how concrete examples of source/target models are transformed.
- *trans*ML supports a dedicated *transformation case* language:
  - How examples are to be related.
  - Applicable to models or model fragments.
- Used to reason about what a transformation should do.
- Used as input to transformation-by-example approaches.
  - BX research gap here?
  - Fitness functions based on consistency relations? Need to be made measurable?

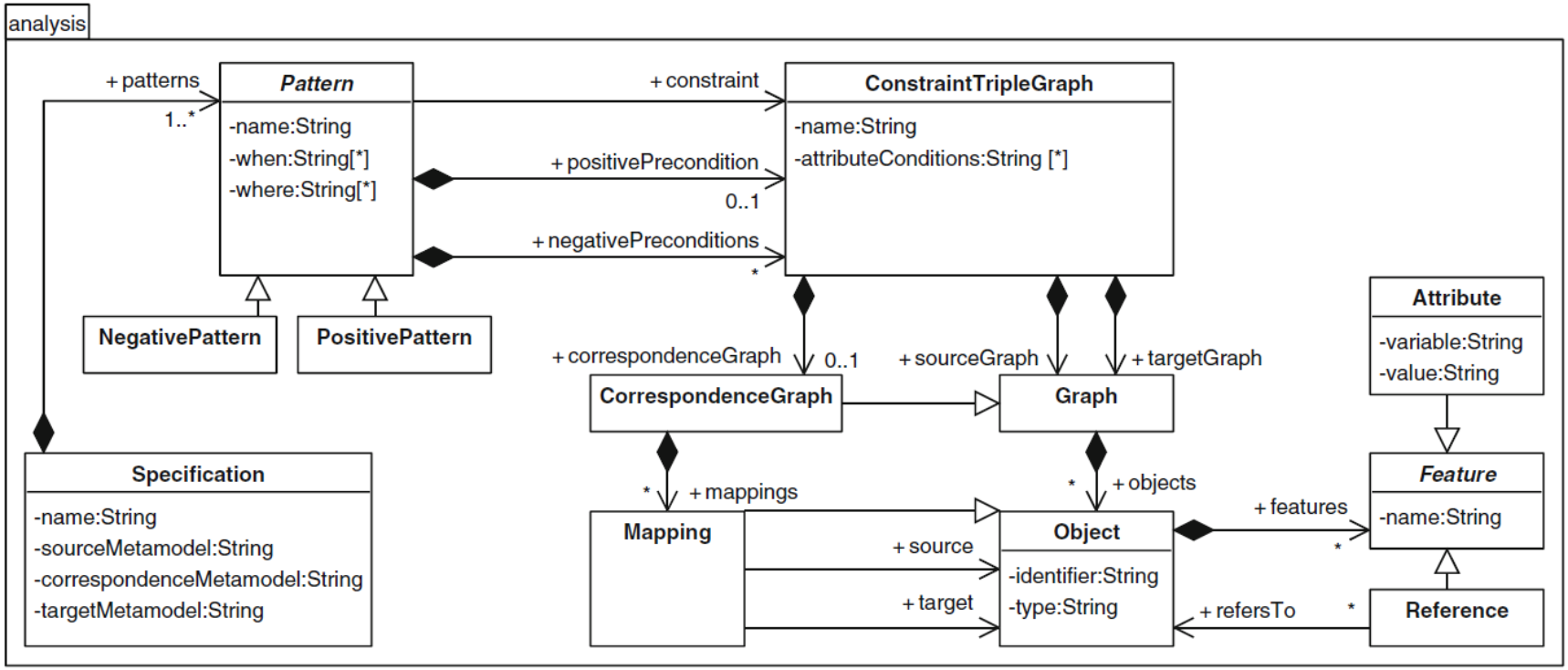
*trans*ML Example Case



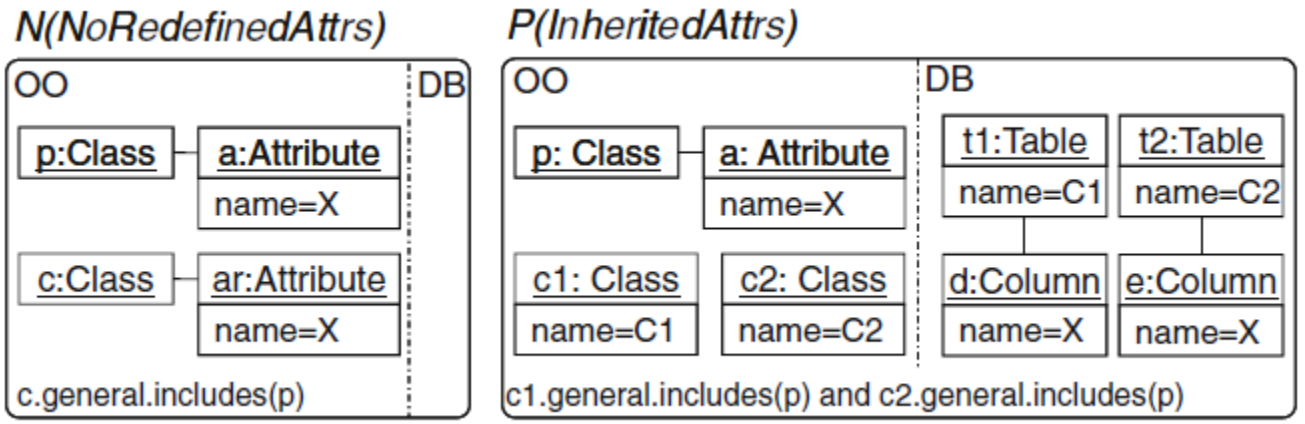
*trans*ML Requirement Specification

- Visual formal language to describe what a transformation has to do
  - specify correctness properties; specify restrictions on source/target models.
- Uses declarative patterns to express allowed and forbidden relations.
- Patterns have a graphical part (*ConstraintTripleGraph)* and can include conditions on attribute values and constraints (using EOL).

*trans*ML Requirements Specification Metamodel



*trans*ML Example Patterns



Continuing

- Architecture and design
  - Relevant architectural styles and patterns for BX.
- *trans*ML support for high level design (e.g., mappings) and low level design (e.g., rule structure diagrams)
- Verification and validation:
  - Generating unidirectional transformations
  - Verification with Hoare logic