On the Quest for Flexible Modelling

Esther Guerra, Juan de Lara

MISO - Modelling & Software Engineering Research Group (miso.es)
Universidad Autónoma de Madrid (Spain)
Motivation

- Diverse nature of modelling, ranging:
  - from informal (e.g., for discussion)
  - to fully formal (e.g., for code generation)

- Most modelling tools only serve one of these extreme purposes:
  - create informal models or diagrams (imprecise)
  - build models fully conformant to the modelling language (rigid)

- MDE tools on the rigid side:
  - it hinders a wider adoption of MDE
  - unnecessarily complex solutions to some scenarios
Our claim: modelling tools need further flexibility
- cover different stages, purposes, and approaches
- explicit modelling process and conformance rules

In this presentation:
- requirements for flexible modelling tools
- application scenarios
- our proposal: meta-modelling language + explicit modelling process
- the Kite meta-modelling framework
Requirements and Scenarios
Requirements and scenarios
R1: Configurable inconsistency tolerance

R1: The user should be able to relax the model conformance rules.

Possibility to enable/disable:
- cardinality and integrity constraints
- type checking of field values
- objects with an abstract type
- objects with a non-existing type
Requirements and scenarios

R1: Configurable inconsistency tolerance

**R1**: The user should be able to relax the model conformance rules.

Scenarios:

- **model life-cycle**: from less to more strict rules
- **model migration, meta-model evolution**: incorrect models will “load”
- **meta-model testing**: partial, incorrect test models
- **test-driven meta-model development**: non-existing types and features
R2: The user should be able to dis(allow) information extension.

Possibility to have (or not):
- objects with no type (it is type-safe)
- typed objects with fields not in the object type
Requirements and scenarios

R2: Information extension

**R2:** The user should be able to dis(allow) information extension.

Scenarios:

- **data injection:** no meta-model upfront
- **language extension:** emergent features as untyped elements
- **auxiliary computation elements:** flags, clocks... as untyped elements
- **language creation:** creating types from untyped elements
Requirements and scenarios

R3: Configurable classification relation

R3: The user should be able to configure the classification relation.

Possibility to enable/disable:
- dynamic, multiple typing
- creation and a-posteriori types
- multiple meta-levels

Modelling tools typically support:
- single, static typing
- creation types
- two meta-levels
Requirements and scenarios

R3: Configurable classification relation

R3: The user should be able to configure the classification relation.

Scenarios:

- reuse of model operations: by allowing multiple typing

- joint instantiation of sets of classes: e.g., used in ontologies

- multi-level modelling: by allowing multiple meta-levels
R4: Configurable generalisation relation

R4: The user should be able to configure the generalisation relation.

Possibility to enable/disable:

- multiple inheritance
- generalisation between objects
Requirements and scenarios

R4: Configurable generalisation relation

**R4:** The user should be able to configure the generalisation relation.

Scenarios:

- **model libraries:** reusable by object inheritance
Requirements and scenarios

R5: Explicit and configurable modelling process

R5: The tool should allow defining and enacting modelling processes.

Modelling processes:

- phases, conformance rules
- order of object creation
- current modelling phase
  - manual
  - automatic
- process intent
  - meta-model creation (bottom-up, top-down, blended)
  - model creation

Diagram:

- Process
  - modelling process
  - process intent
  - explicit, generic, domain-specific, state change
  - bottom-up, top-down, blended
  - configurable, manual, automatic
  - model refinement
Requirements and scenarios

R5: Explicit and configurable modelling process

**R5**: The tool should allow defining and enacting modelling processes.

Scenarios:

- transition from informal to formal modelling
- modelling guidelines: e.g., in UML, classes first
R6: Fixes and refactorings may depend on the process intent.

For example, given a model error:

- bottom-up fixes modify the meta-model
- top-down fixes modify the model
- domain-specific fixes
Requirements and scenarios

R6: Process-aware extensible assistance

R6: Fixes and refactorings may depend on the process intent.

Scenarios:

- **model refinement**: model fixes and refactorings
- **live meta-model/model co-evolution**: model fixes and refactorings
- **bottom-up meta-modelling**: meta-model fixes
- **recommendation systems**
Our Proposal
Architecture

- meta-modelling language for flexible modelling
- explicit modelling process
Meta-modelling language

Basic modelling elements (R3, R4)

- Support for multiple levels (one class to represent types and instances)
- Models and objects can have features and constraints
- Instantiation cardinality vs Value cardinality
- References can have several targets, at any level
- Generalization at any meta-level (relation Object.super)

![Diagram of meta-modelling elements]
Support for multiple levels (one class to represent types and instances)
Models and objects can have features and constraints
Instantiation cardinality vs Value cardinality
References can have several targets, at any level
Generalization at any meta-level (relation Object.super)
Meta-modelling language
Basic modelling elements (R3, R4)

- Support for multiple levels (one class to represent types and instances)
- Models and objects can have features and constraints
- Instantiation cardinality vs Value cardinality
- References can have several targets, at any level
- Generalization at any meta-level (relation Object.super)

Diagram:

- **TypedElement**
  - name: String
  - lbound: int
  - ubound: int

- **ValuedElement**
  - lbound_v: int
  - ubound_v: int

- **Feature**
  - features
  - super
  - to

- **Integrity Constraint**
  - name: String
  - body: String

- **Context**
  - objects

- **Object**
  - super
  - object

- **Target**
  - to

- **Model**
  - objects

- **Attribute**
  - super

- **Reference**
  - super

- **Featured Element**
  - features
  - super

- **Meta-modelling language**
- **Basic modelling elements (R3, R4)**
- **Support for multiple levels (one class to represent types and instances)**
- **Models and objects can have features and constraints**
- **Instantiation cardinality vs Value cardinality**
- **References can have several targets, at any level**
- **Generalization at any meta-level (relation Object.super)**
Support for multiple levels (one class to represent types and instances)
Models and objects can have features and constraints
Instantiation cardinality vs Value cardinality
References can have several targets, at any level
Generalization at any meta-level (relation Object.super)
Support for multiple levels (one class to represent types and instances)
Models and objects can have features and constraints
Instantiation cardinality vs Value cardinality
References can have several targets, at any level
Generalization at any meta-level (relation Object.super)
Meta-modelling language

Flexible typing (R1, R2, R3)

- Explicit typing relation
  - Zero, one or more typings for an instance
  - Types can be assigned at creation time, or later
  - Re-typing (preserving the instance identity)
Meta-modelling language

Examples

1 Conference {
2  Author {}
3  Reviewer /1..*/ {}
4 }

5 MODELS :Conference /0..0/ {
6  amelia :Author :Reviewer {}
7  lateReviews {
8    ref who = amelia;
9  }
10 }
11 }
Meta-modelling language

Examples

```
Conference {
  Author {}
  Reviewer /1..*/ {} ——> instantiation cardinality
}

MODELS :Conference /0..0/ { ——> instantiation cardinality
  amelia :Author :Reviewer {}
  lateReviews {
    ref who = amelia;
  }
}
```
Meta-modelling language

Examples

```
1 Conference {
2  Author {}
3  Reviewer /1..*/ {}
4 }
5
6 MODELS :Conference /0..0/ {
7   amelia :Author :Reviewer {} —–> object with multiple types
8   lateReviews {
9     ref who = amelia;
10  }
11 }
```
Meta-modelling language

Examples

1 Conference {
2  Author {}
3  Reviewer /1../*/ {}
4 }
5
6 MODELS :Conference /0..0/ {
7   amelia :Author :Reviewer {}
8   lateReviews { −→ object with no types
9     ref who = amelia;
10 }
11 }
Meta-modelling language

Examples

1  ArtistTypes {  
2      Singer {  
3          att name : String;  
4          att stageName : String;  
5      }  
6  }  
7
8  SomeMusicians :ArtistTypes {  
9      tina :Singer {  
10          att name = "Anna Mae Bullock";  
11          att stageName = "Tina Turner";  
12      }  
13      joaquin :Singer {  
14          att realName (:name :stageName) = "Joaquin Pascual";  
15      }  
16  }
Explicit modelling process: phases, checks, transitions
- checks: predefined conformance rules, or custom-made ocl conditions
- transitions: manual or automatic, may define ocl conditions

Process intent: refinement, top-down, bottom-up, free
Quick fixes can be filtered by process intent
Conformance rules and quickfixes can be extended by users
Reified modelling process

Explicit modelling process (R5, R6)

- Explicit modelling process: phases, checks, transitions
  - checks: predefined conformance rules, or custom-made ocl conditions
  - transitions: manual or automatic, may define ocl conditions
- Process intent: refinement, top-down, bottom-up, free
  - Quick fixes can be filtered by process intent
  - Conformance rules and quickfixes can be extended by users
Explicit modelling process: phases, checks, transitions
- checks: predefined conformance rules, or custom-made ocl conditions
- transitions: manual or automatic, may define ocl conditions
Process intent: refinement, top-down, bottom-up, free
Quick fixes can be filtered by process intent
Conformance rules and quickfixes can be extended by users
Reified modelling process
Explicit modelling process (R5, R6)

- Explicit modelling process: phases, checks, transitions
  - checks: predefined conformance rules, or custom-made ocl conditions
  - transitions: manual or automatic, may define ocl conditions
- Process intent: refinement, top-down, bottom-up, free
- Quick fixes can be filtered by process intent
- Conformance rules and quickfixes can be extended by users
Reified modelling process
Further configuration options (R5, R6)

- Configuration of the meta-modelling language:
  - expresiveness: multiple typing, multiple levels, multiple inheritance
  - extensibility: untyped objects, untyped features
Reified modelling process

Examples

Intent: ModelRefinement
TransitionMode: Manual
Expressiveness: MultiTyping, MultiInheritance
Extensibility: UntypedObjects, UntypedFeatures

Typed checks: object types
Bounded checks: value cardinality, instance cardinality
Well-formed checks: type of feature values
Strict checks: integrity constraints

Draft
Typed
Bounded
Well-formed
Strict

Draft
Typed
Bounded
Well-formed
Strict

Fixes (refinement)
- remove type
- change to Author/Reviewer

Current Conference

MODELS :Conference {
  amelia :Author :Reviewer {}
  lateReviews :Comment {
    ref who : amelia;
  }
}

1 Conference {
  2 Author {}
  3 Reviewer /1..*/ {}
  4 }
  5 }
  6 MODELS :Conference {
  7 amelia :Author :Reviewer {}
  8 lateReviews :Comment {
  9 ref who : amelia;
  10 }
  11 }

Esther Guerra
On the Quest for Flexible Modelling
MoDELS 2018
Reified modelling process

Examples

**Intent:** BottomUp

**Transition Mode:** Manual

**Expressiveness:** Multinheritance

**Extensibility:** -

Free
- **Quick fixes**
  - createModelType

Object Types
- **checks:** object types
- createObjectType

Feature Types
- **checks:** feature types
  - createFeatType
  - createObject&FeatTypes

Value Types
- **checks:** type of feature values
  - createValueType
  - createFeat&ValueType
  - createObject&Feat&ValueType

Cardinalities
- **checks:** value cardinality, instance cardinality
- updateCardinality

Fixes (bottomup)
- Add new type Comment

1. Conference {
2.  Author {} 
3.  Reviewer /1..*/ {} 
4. } 
5. MODELS :Conference {
6.  amelia :Author :Reviewer {} 
7. } 
8. lateReviews :Comment {
9.  ref who : amelia; 
10. } 
11. }

Esther Guerra

On the Quest for Flexible Modelling

MoDELS 2018

18 / 23
Our prototype implementation Kite

- Kite is an eclipse textual editor for flexible modelling
- Based on EMF (for inter-operability), Xtext, and EVL (constraints)

Esther Guerra

On the Quest for Flexible Modelling
Conclusions
Summary

- Flexibility in modelling tools is useful in many scenarios

- List of requirements for flexible modelling tools:
  - flexible typing
  - explicit modelling process

- Initial proposal and implementation

- In the paper: review of existing flexible (meta-)modelling tools
  - support for flexibility is only partial
  - big gap on explicit modelling processes (opportunity!)
Next steps

- Improving Kite, e.g., DSL to define modelling processes
- Integration with further model management languages
- Extend reasoners to work with non-fully conformant models
- Explore others aspects of flexibility, like concrete syntax
- Meta-object protocols to extend meta-modelling facilities
On the Quest for Flexible Modelling

Esther Guerra, Juan de Lara
esther.guerra@uam.es

MISO - Modelling & Software Engineering Research Group (miso.es)
Universidad Autónoma de Madrid (Spain)

Comments? Questions?