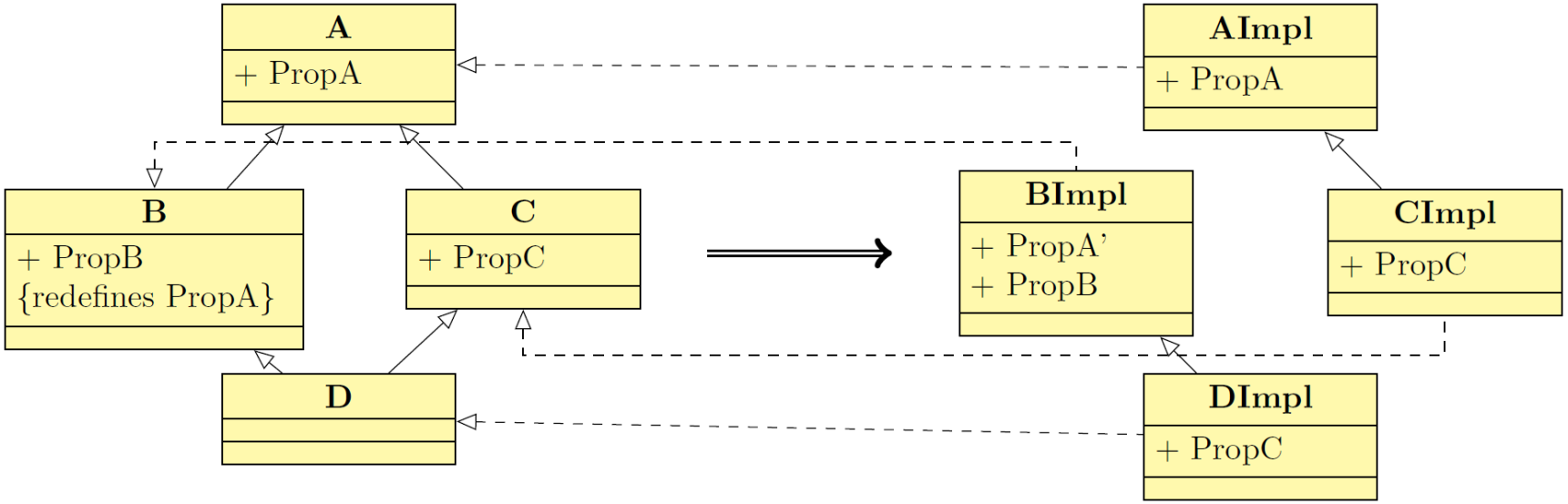
The TTC 2017 Live Contest on Transformation Reuse in the Presence of Multiple Inheritance and Redefinitions

Georg Hinkel

Reuse in Model Transformations

- Model transformations are the „heart-and-soul“ of MDE [SK03]
  - Increasing adoption of MDE ➔ Increasing need for modularity and reuse
  - Current reuse facilities seem not satisfactory [WKK+12, KSW+13]
- Reuse only advantageous if reused code is substantial
  - Reuse introduces a dependency
  - Not viable to reuse a rule that only copies a name…
- Code Generation for Refinements as a reuse problem [HGB+17]

Code Generation for Multiple Inheritance and Redefinitions



- Code generator maps metaclass to interface + default implementation as class
  - Multiple inheritance is resolved to single inheritance + replication of features
- Refinements influence the set of possible base types to inherit from
- Choice of base types independent of representation for attributes/references

Finding a base class

Algorithm 1 Find implementation base class

function AllFeatures(c) **return** $\bigcup_{c \preceq c_b} c_b.eStructuralFeatures$

function Refinements(c) **return** $\{g | f \in c.eStructuralFeatures, f \xrightarrow{\ll refines \gg} g\}$

function Edge(c_s, c_t) **return** $c_s \preceq c_t \vee (\text{Refinements}(c_s) \cap \text{AllFeatures}(c_t) \neq \emptyset \wedge c_t \not\preceq c_s)$

function FindBaseClass(c)
  $shadows \leftarrow \text{Refinements}(c)$
  $ancestors \leftarrow \text{TransitiveHull}(c, cl \mapsto cl.eSuperTypes)$
  for all $layer$ in ReverseTopologicalOrder($ancestors$, Edge) **do**
    if $|layer| = 1 \wedge layer \neq \{c\} \wedge shadows \cap \text{AllFeatures}(layer[0]) = \emptyset$ **then return** $layer[0]$
    for all l in $layer$ **do**
      $shadows \leftarrow shadows \cup \text{Refinements}(l)$
  return $\perp$

- Algorithm for selecting an appropriate base class provided
  - Reversed topological sort, for instance using the algorithm of Tarjan [Tar72]
  - Java Implementation of topological sort included in case resources

Task: Create two code generators

- Code Generator A: Represent references as properties backed with a field
  - Non-refined reference: Generate property and backing field
  - Refined reference: Generate property that accesses refining property
- Code Generator B: Represent references as methods that lazy load the references from a database
  - Non-refined reference: Call resolve
  - Refined reference: Call method for refining reference
  - All classes must inherit (directly or indirectly) from DBObject
- Choice of base class independent of concrete representation of attributes and references ➔ Reuse very important

Example Output: Code Generator A

Code Generator A:

Properties backed with a field

```
1  interface A
2    abstract property PropA : E
3
4  class AImpl : A
5    field _PropA : E
6    property PropA : E
7      get: this._PropA
8      set: (this._PropA = value)
9
10 interface B : A
11   abstract property PropB : E
12
13 class BImpl : B
14   field _PropB : E
15   property PropA : E
16     get: this.PropB
17     set: (this.PropB = value)
18
19   property PropB : E
20     get: this._PropB
21     set: (this._PropB = value)
```

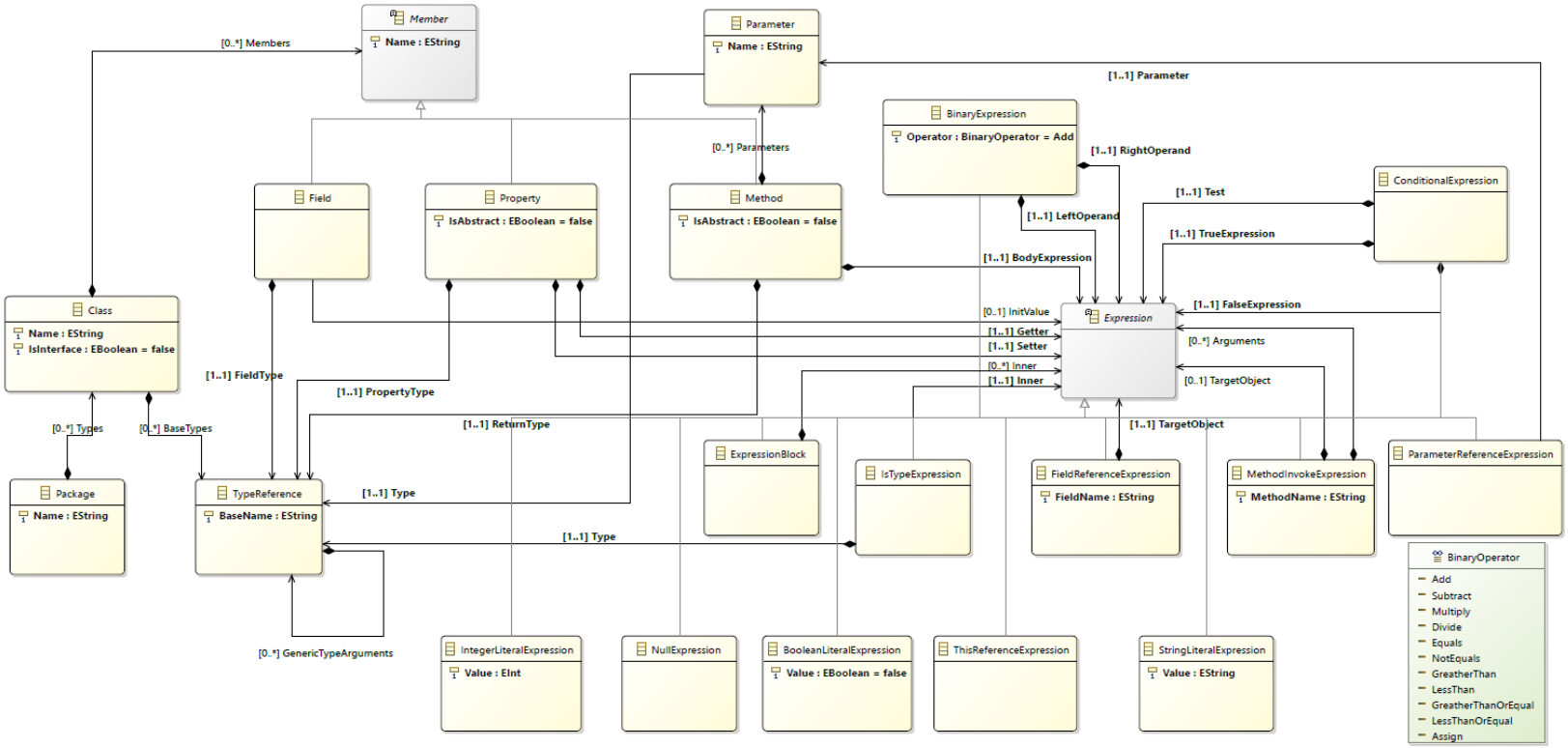
Code Generator B:

Lazy Loading from a database

```
1  interface A
2    abstract method get_PropA()
3
4  class AImpl : DBObject, A
5    method get_PropA()
6      resolve(this, 'PropA')
7
8  interface B : A
9    abstract method get_PropB()
10
11 class BImpl : B
12   method get_PropA()
13     this.get_PropB()
14
15   method get_PropB()
16     resolve(this, 'PropB')
```

Metamodels

- Input: Modified version of Ecore that supports refinements
- Output: Either text or code model



Limitations

- Example instances contain only:
  - One instance of EPackage
  - Multiple instances of EClass with multiple base types
  - Multiple instances of EReference that have multiplicity 1
- All test models can be assumed correct (only use redefinitions where allowed)

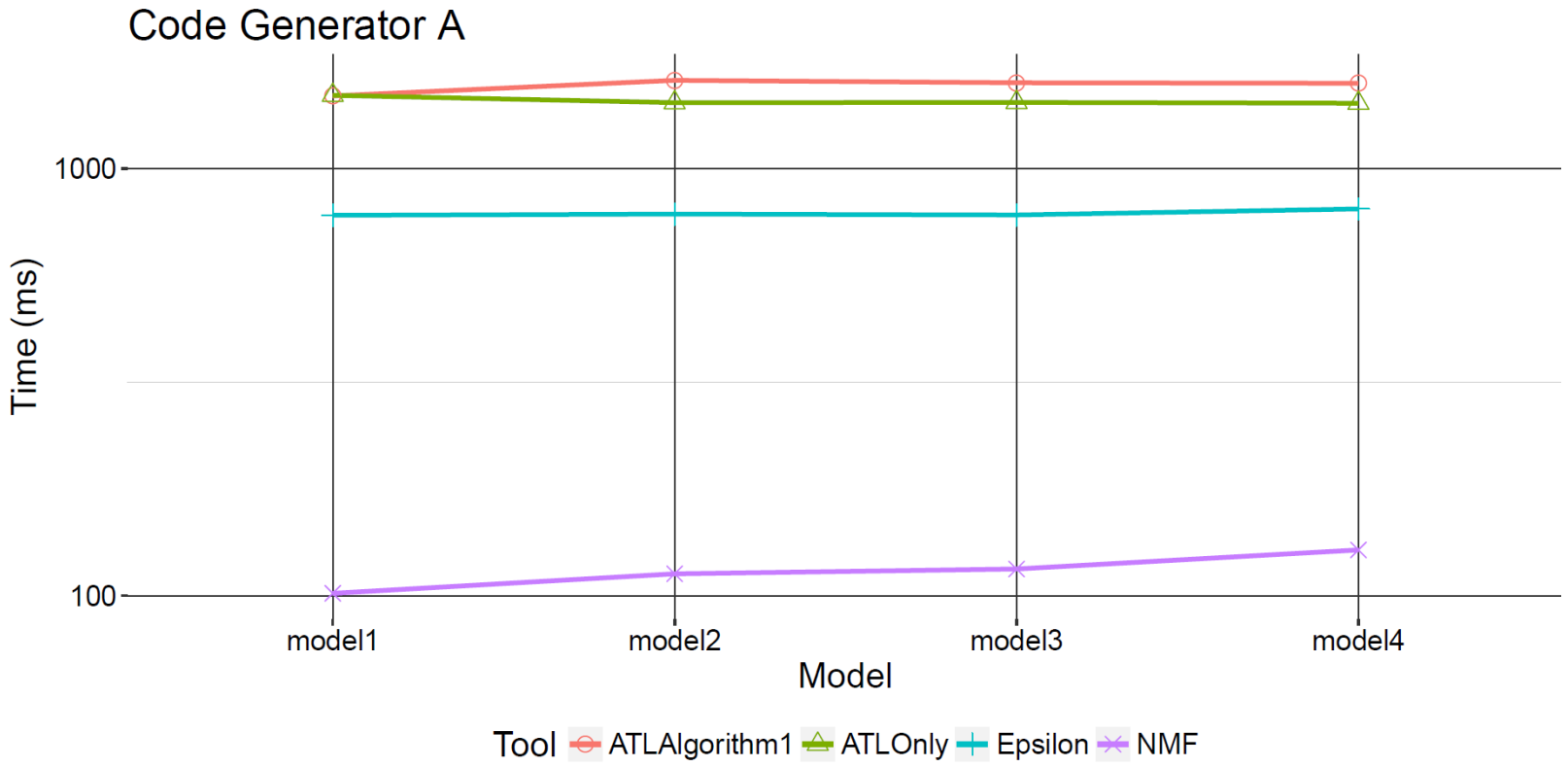
Resources

- Benchmark framework and resources available online
  - Case description
  - Benchmark Framework
  - Metamodels
  - Example Models
  - Expected Results
  - Reference Solution in NTL [Hin13]
  - Java implementation of Topological sort
    - usage not mandatory

- http://github.com/georghinkel/ttc2017LiveContest
- To submit a solution, clone the repo and create a Pull Request before Thursday 23:59:59 CET

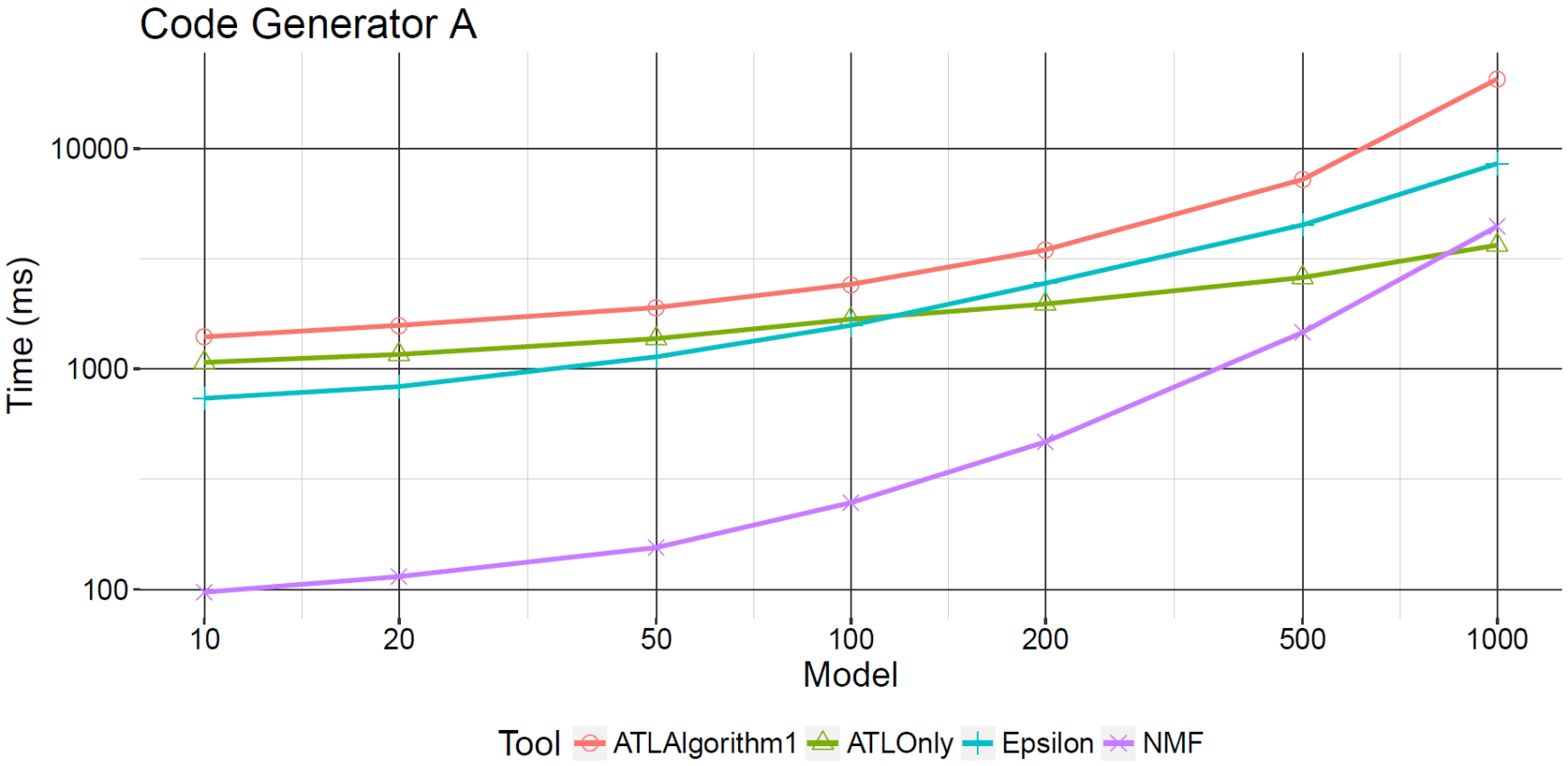
Performance Results I

- Performance for the Test Models



Performance Results II

- Performance for model4 multiplied n times



Conclusion

- Reuse in Model Transformations
  - Reuse complex logic
  - Define and reuse transformation skeletons
- Use case: Code generation in the presence of multiple inheritance and refinements
- Benchmark framework with many resources available
- http://github.com/georghinkel/ttc2017LiveContest
- To submit a solution, clone the repo and create a Pull Request before Thursday 23:59:59 CET

hinkel@fzi.de

THANK YOU FOR YOUR ATTENTION

References

[SK03] S. Sendall and W. Kozaczynski, “Model transformation the heart and soul of model-driven software development,” Tech. Rep., 2003.

[WKK+12] M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger, “Fact or fiction–reuse in rule-based model-to-model transformation languages,” in Theory and Practice of Model Transformations, Springer, 2012, pp. 280–295.

[KSW+13] A. Kusel, J. Schönböck, M. Wimmer, G. Kappel, W. Retschitzegger, and W. Schwinger, “Reuse in model-to-model transformation languages: Are we there yet?” Software & Systems Modeling, pp. 1–36, 2013.

[HGB+17] G. Hinkel, T. Goldschmidt, E. Burger, and R. Reussner, “Using Internal Domain-Specific Languages to inherit Tool Support and Modularity for Model Transformations,” Software & Systems Modeling, pp. 1–27, 2017.

[Hin13] G. Hinkel, “An approach to maintainable model transformations using an internal DSL,” Master’s thesis, Karlsruhe Institute of Technology, 2013.

[Tar72] R. Tarjan, “Depth-first search and linear graph algorithms,” SIAM journal on computing, vol. 1, no. 2, pp. 146–160, 1972.