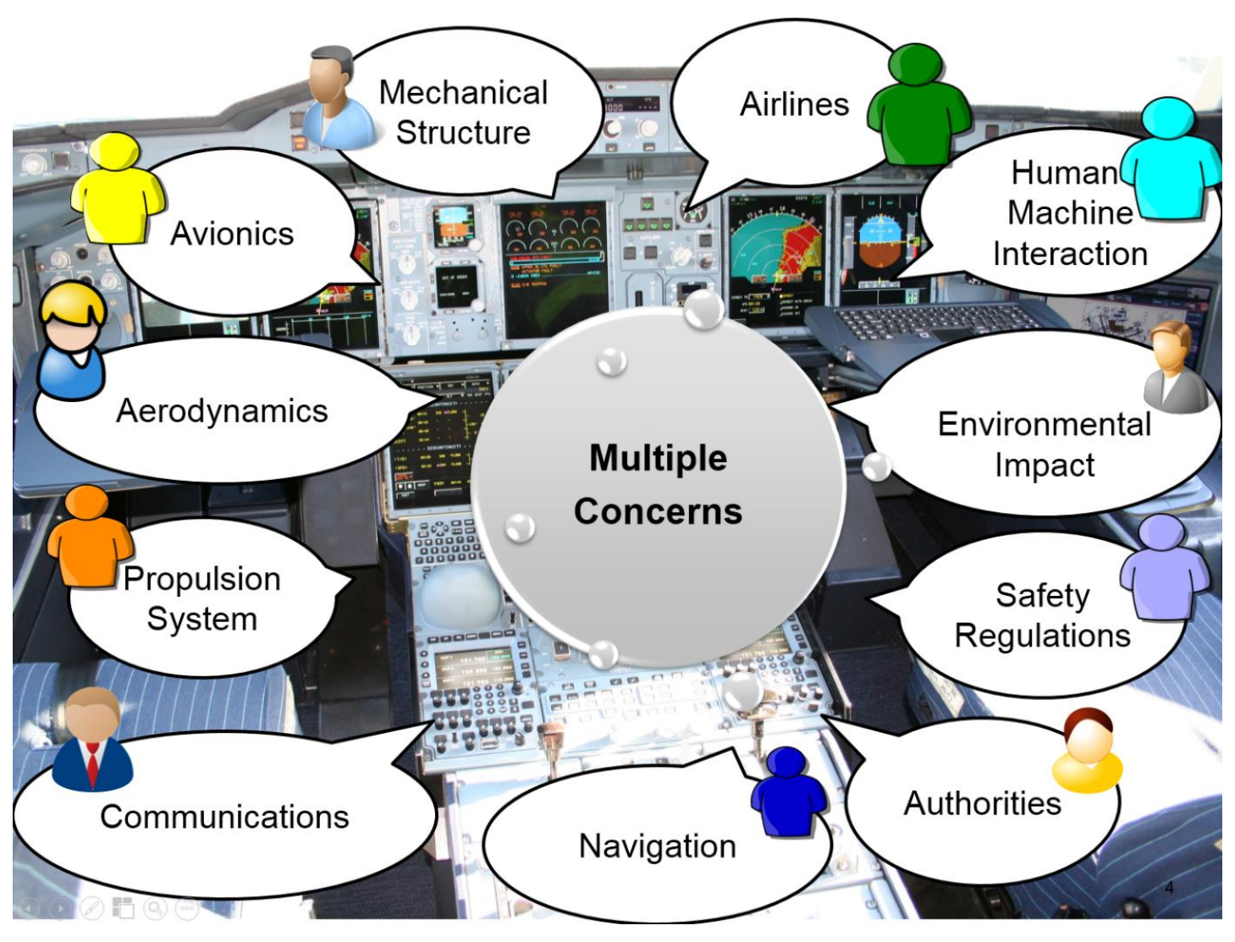
Towards Language Interfaces for DSL Integration with Melange

Thomas Degueule – INRIA, France

CONTEXT & MOTIVATION

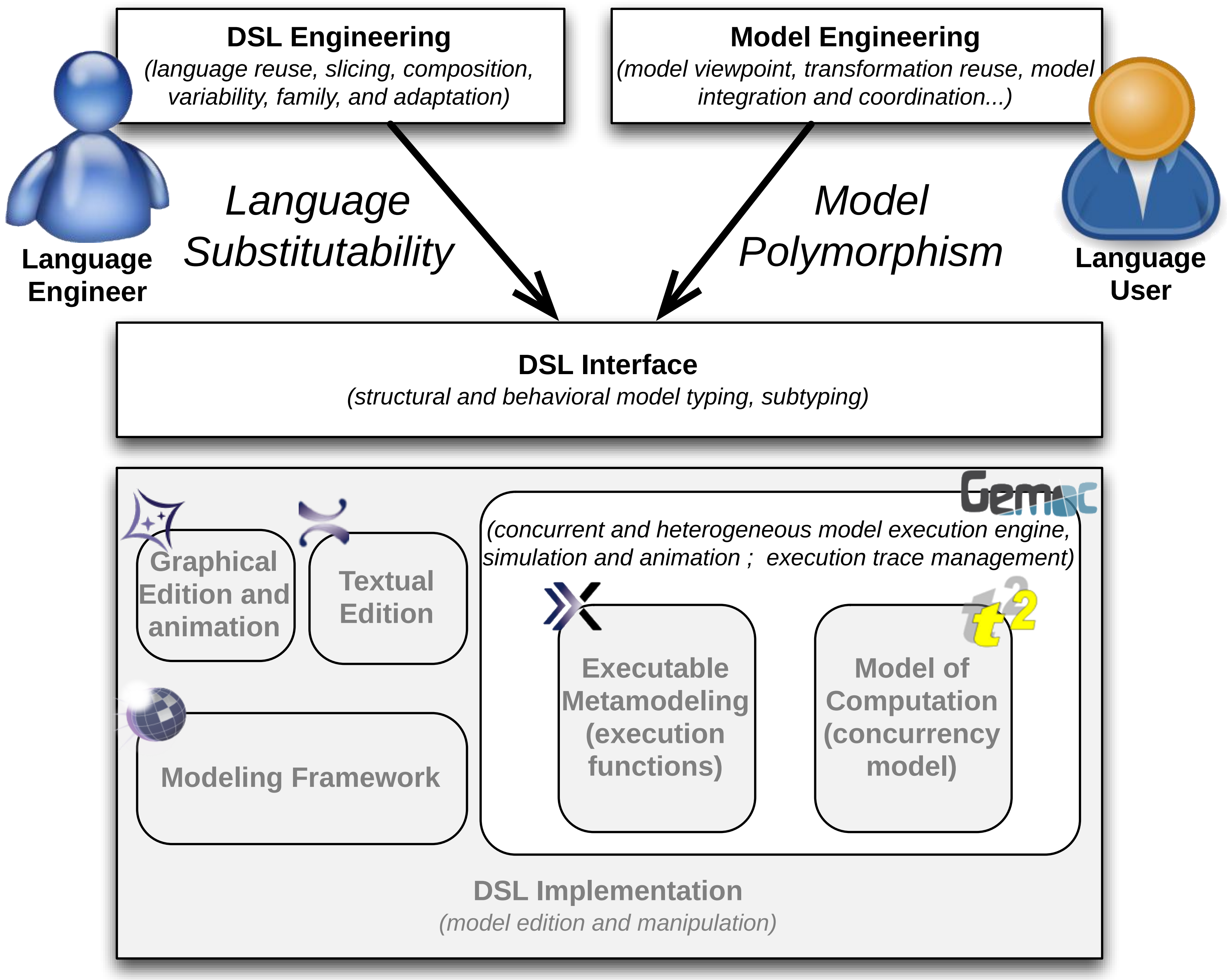
- Model-Driven Engineering (MDE) proposes to address each aspect of a system with dedicated DSLs closely tied to the needs of stakeholders
- DSLs evolve as the experts understanding of the domain evolve, and may eventually be replaced with alternatives DSLs
- The definition of a DSL and its tooling is costly considering its limited audience
- The lack of abstraction and genericity in the manipulation of languages and models hinders evolution, maintenability and reusability capabilities



Multiple stakeholders use multiple, constantly evolving DSLs to address multiple concerns

LANGUAGE INTERFACES

- Language interfaces enhance abstraction and genericity
  - Abstract the intrinsic complexity of language implementation
  - Expose meaningful information
  - Concerning an aspect of a language (e.g. abstract syntax)
  - For a specific purpose (e.g. composition or reuse)
  - In an appropriate formalism (e.g. a metamodel)
- Binding relation between language implementations and interfaces
- Ease the definition of operators between the interfaces



DSL Engineering
(language reuse, slicing, composition, variability, family, and adaptation)

Model Engineering
(model viewpoint, transformation reuse, model integration and coordination...)

Language Engineer

Language Substitutability

Model Polymorphism

Language User

DSL Interface
(structural and behavioral model typing, subtyping)

Graphical Edition and animation

Textual Edition

Modeling Framework

(concurrent and heterogeneous model execution engine, simulation and animation ; execution trace management)

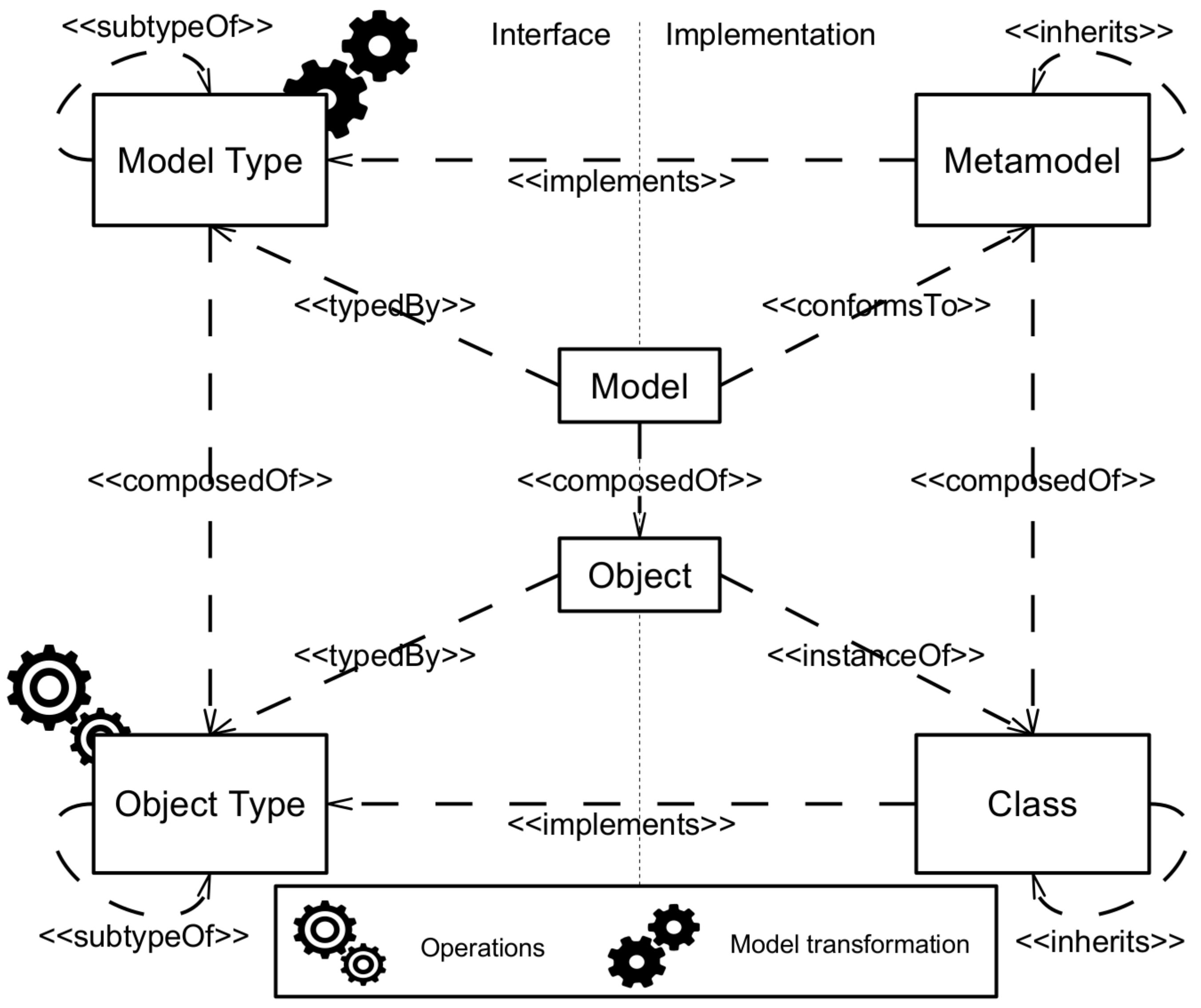
Executable Metamodeling (execution functions)

Model of Computation (concurrency model)

DSL Implementation
(model edition and manipulation)

MODEL TYPE: A STRUCTURAL INTERFACE

- MDE strongly relies on the conformance relation which hinders reuse
- Model types as an explicit typing interface on top of DSLs metamodels
- Provides model substitutability and polymorphism
- Leveraging type group polymorphism and structural typing



Interface

Implementation

Model Type

Metamodel

Model

Object

Object Type

Class

<<subtypeOf>>

<<inherits>>

<<implements>>

<<typedBy>>

<<conformsTo>>

<<composedOf>>

<<composedOf>>

<<composedOf>>

<<typedBy>>

<<instanceOf>>

<<implements>>

<<subtypeOf>>

<<inherits>>

Operations

Model transformation

MELANGE

- A language-based, model-oriented programming language
- Models as first-class, typed citizens
- Model-oriented type system providing model polymorphism
- Handy operators for language engineering (inheritance, merge, slicing, aspect weaving, etc.)
- Seamlessly integrated with the Eclipse Modeling Framework ecosystem

```
// Language and interface definition
modeltype FsmMT {
  ecore “FSM.ecore”
}

language ExecFsm implements FsmMT {
  ecore “FSM.ecore”
  with ExecutableFSM
  with ExecutableState
  with ExecutableTransition
  exactType ExecFsmMT
}

language TimedFsm inherits ExecFsm {
  ecore “TimedFsm.ecore”
  with TimedTransition
  exactType TimedFsmMT
}
```

```
// Generic model manipulation
transformation flatten(FsmMT m) {
  m.ownedStates.forEach[...]
}
transformation execute(ExecFsmMT m){
  // Dynamically binded to the
  // appropriate execution semantics
  // (with/without time constraints)
  m.root.execute(“input-word”)
}
main() {
  val m1 = ExecFsm.load(“…”)
  val m2 = TimedFsm.load(“…”, FsmMT)
  val m3 = new TimedFsm
  flatten.call(m1)
  flatten.call(m2)
  execute.call(m1)
  val m4 = m1 as FsmMT
}
```

EXPERIMENTS & FUTURE WORK

- Families of syntactically and semantically diverse languages (e.g. FSM)
- Integrated in the ANR GEMOC project to support language extension and model polymorphism in the context of heterogeneous model execution and coordination
- Experienced in the Clarity project to design an executable extension of the Capella system engineering language
- Experienced in the ITEA2 MERgE project to extend the UML language with domain-specific metrics for evaluation of architecture variants

- Model types as a support for viewpoints engineering. Investigated for designing task-oriented viewpoints that span multiple DSLs
- Model types as explicit required and provided interfaces for the design and composition of language units
- Generic meta-programming through the reuse of generic analyses on close programming languages

Get it, try it, hack it! – http://melange-lang.org