



# The Evolutionary Life of Metamodels

---

Alfonso Pierantonio

SWEN / Software Engineering Research Group

Università degli Studi dell'Aquila, Italy

The Jjodel Project

# Conceptual Foundation

---



Metamodels evolve like living systems

Co-evolution is not an afterthought but a first-class, continuous capability

The talk demonstrates how reflective platforms such as Jjodel make this evolution live and seamless

# Evolution in Software

---

Dealing with the inevitable

**TABLE I**  
**LAWS OF PROGRAM EVOLUTION**

---

**I. *Continuing Change***

A program that is used and that as an implementation of its specification reflects some other reality, undergoes continual change or becomes progressively less useful. The change or decay process continues until it is judged more cost effective to replace the system with a recreated version.

**II. *Increasing Complexity***

As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it.

**III. *The Fundamental Law of Program Evolution***

Program evolution is subject to a dynamics which makes the programming process, and hence measures of global project and system attributes, self-regulating with statistically determinable trends and invariances.

**IV. *Conservation of Organizational Stability (Invariant Work Rate)***

During the active life of a program the global activity rate in a programming project is statistically invariant.

**V. *Conservation of Familiarity (Perceived Complexity)***

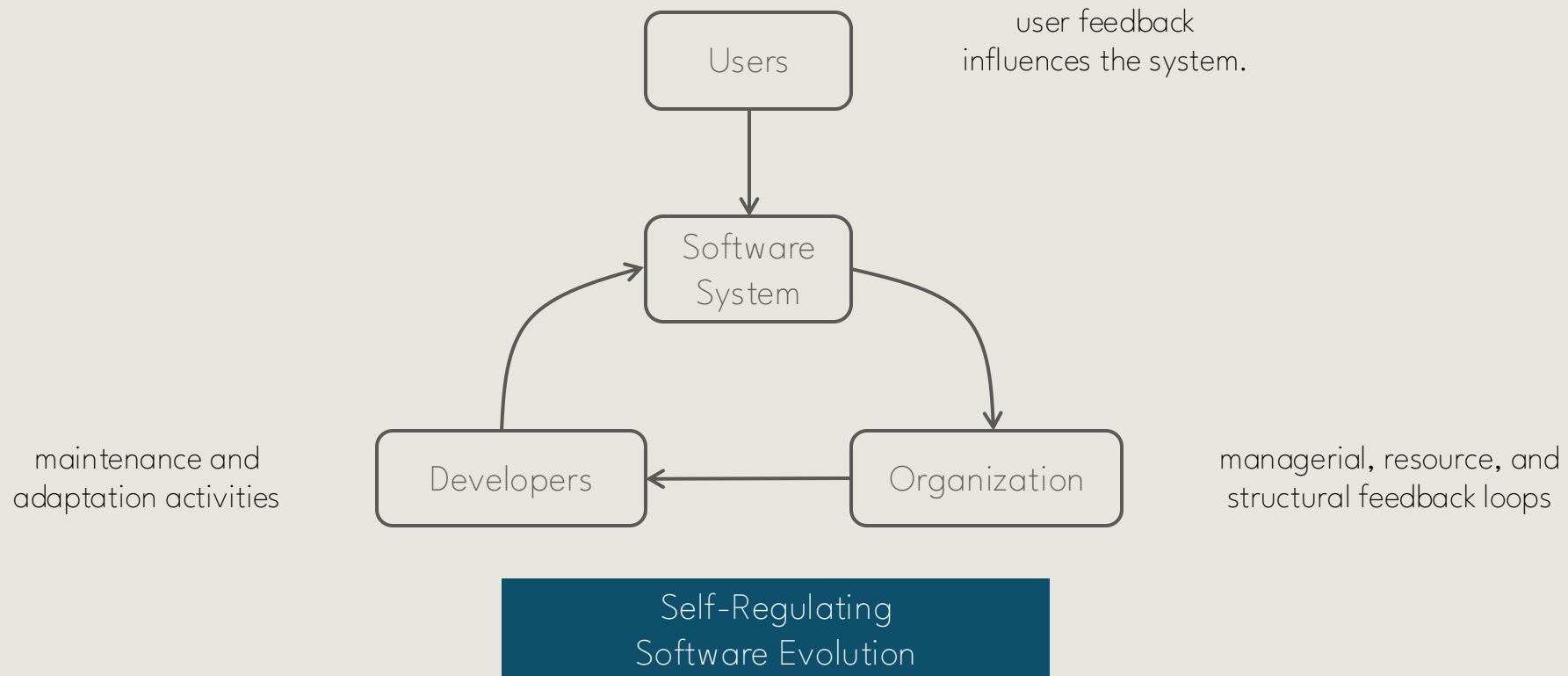
During the active life of a program the release content (changes, additions, deletions) of the successive releases of an evolving program is statistically invariant.

---

## Lehman's Laws

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. Proceedings of the IEEE, 68(9), 1060-1076.

Lehman's Laws show that software evolution is not chaos but a living process: continuous, complex, and self-regulated



# Software Evolution is Inevitable



---

Software lives in a changing world: evolution is a law of nature, not a management choice

The environment never stops changing

- Software does not exist in isolation, it encodes assumptions about the world (users, devices, laws, and business logic)
- When any of those assumptions change, the code that depends on them becomes partially false

The only way to restore correctness is to evolve

# Entropy and Complexity Accumulate



---

- Every change introduces small inconsistencies, technical debt, architectural drift
- Without periodic restructuring, complexity grows

Therefore, even systems with stable requirements must evolve to fight entropy through refactoring and rejuvenation

# Human Understanding Evolves



Requirements are not discovered once: they mature as users and designers learn

- As knowledge increases, abstractions shift: we see new insights, new patterns, new generalizations
- Evolution is therefore also cognitive adaptation: our internal model of the system refines, and the software must follow



# Cognitive Challenges of Evolution

---

Understanding how we think  
about change



Software reflects the shared  
knowledge system formed by the  
user's and designer's cognitive  
schemata of the domain

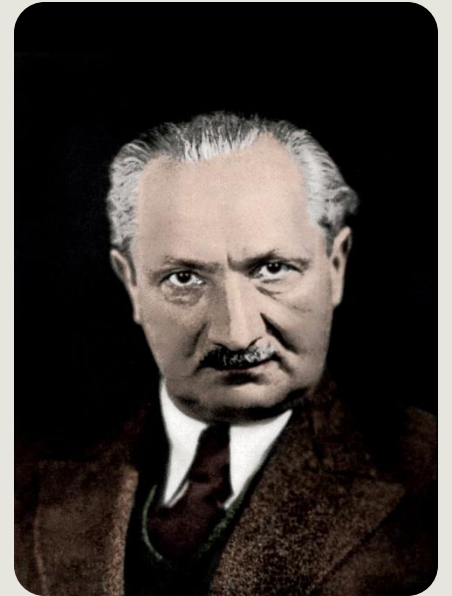
# Technology is a Model of Revealing

---

Martin Heidegger (1889–1976) provides the philosophical foundations to technology

«Technology is not just a means to an end,  
but a mode of revealing»

–In his philosophy, tools are not merely objects, but **mediators** that shape human engagement with the world.



# How Change Reconfigures Understanding



---

Technology is a way of revealing, it discloses how we understand and act in the world

In software, every evolution reconfigures this revealing: changing what is visible, thinkable, and doable

Evolution is cognitively demanding because each change reconfigures our understanding of the software and its abstractions



Martin Heidegger (1889 – 1976)

Phenomenology, Philosophy of Technology

Directly defines **tool transparency** (ready-to-hand)



Jean Piaget (1896–1980)

Developmental and Cognitive Psychology

Defined the concept of **cognitive schema**



A tool is transparent when users have developed a stable cognitive schema: every evolution reopens that schema, demanding new understanding



# A Flushing System

---

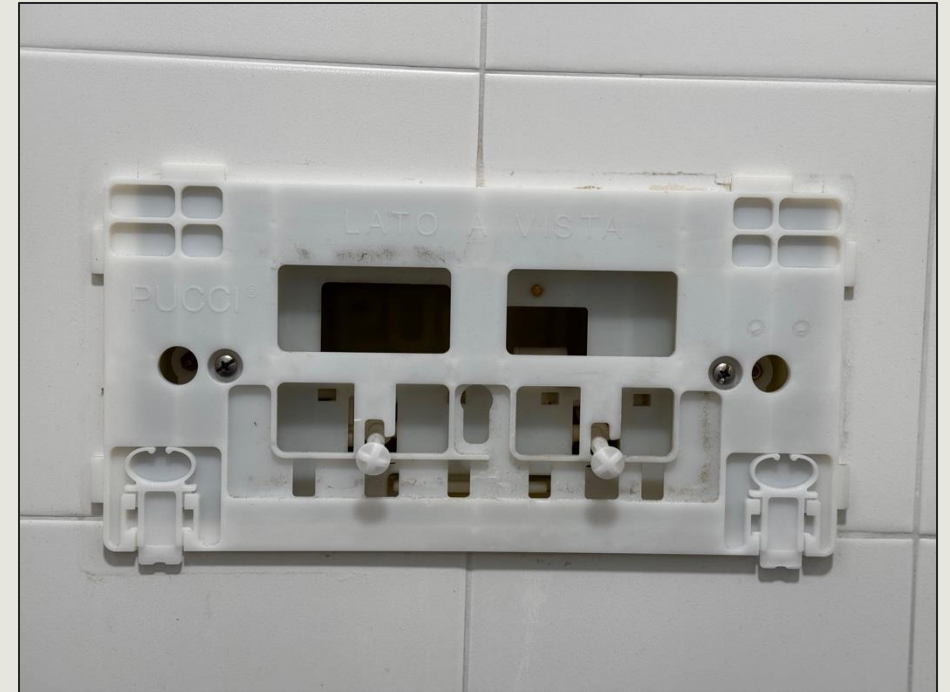
The interface for flushing the toilet is immediately clear and intuitive

- The buttons are designed so that users interact with them without conscious thought
- The flushing system is ready-to-hand !



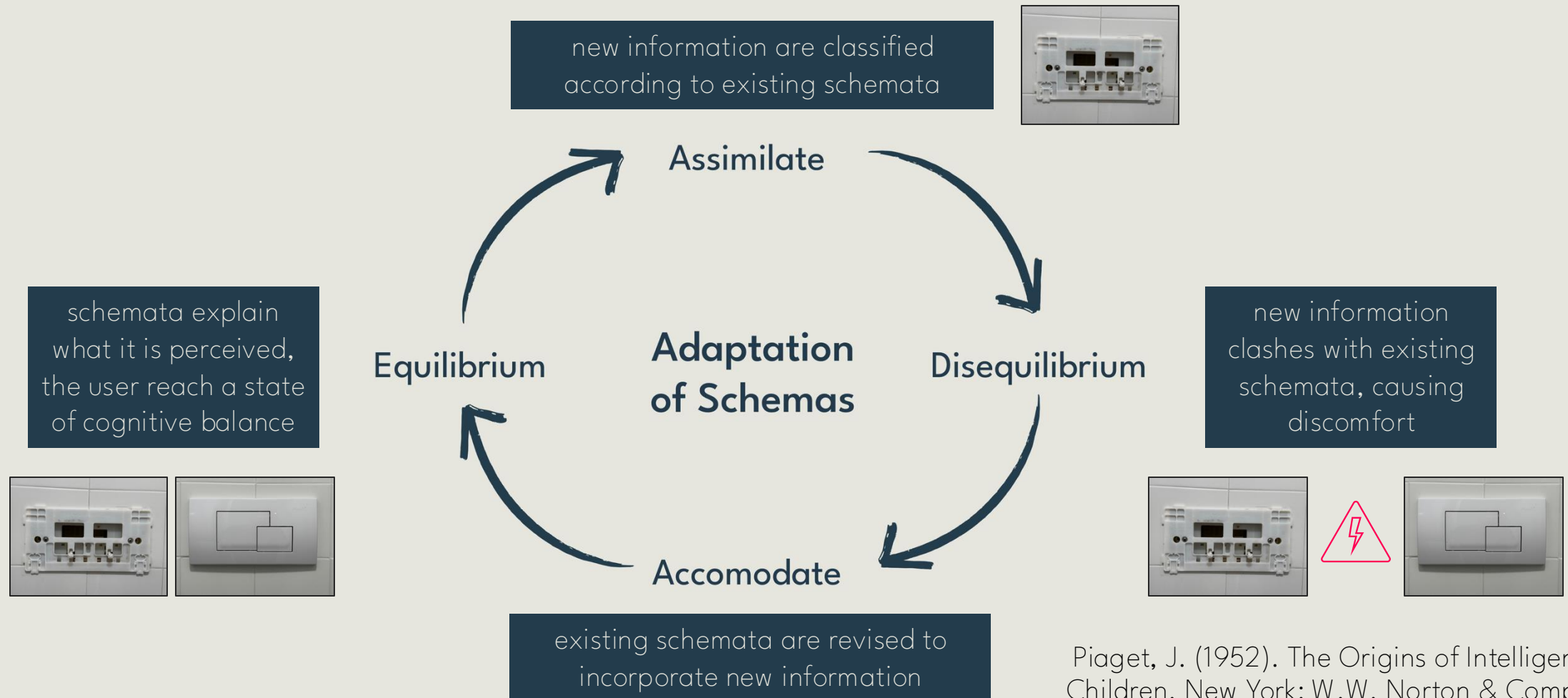
# When Tools Disrupt the Task

- The operation is less intuitive and requires reflection and understanding how to proceed
- The focus is no longer on the task but on **how to operate** the system
- As a consequence, user must **adapt** their cognitive schemata



# The Process of Adaptation

Intellectual growth is a process of adaptation to the world



Piaget, J. (1952). The Origins of Intelligence in Children. New York: W.W. Norton & Company.



# Metamodels

---

# What is a Metamodel?



---

A metamodel formalizes the body of knowledge that characterizes a domain

- It specifies which concepts exist, how they relate, and how they can be composed into valid models
- Every model conforms to its metamodel, just as every sentence conforms to the grammar of its language

# What is a Model?



---

A model is a simplified, cost-effective representation of reality, created to understand, predict, or control aspects of a system

- It abstracts away irrelevant details while preserving those that are essential for a specific purpose
- Models help us reason about complex phenomena by making them tractable and communicable
- Every model is defined according to the concepts and constraints specified by its metamodel

# Metamodels do not live in Isolation



---

Metamodels are the primary assets in Model-Driven Engineering



Metamodel

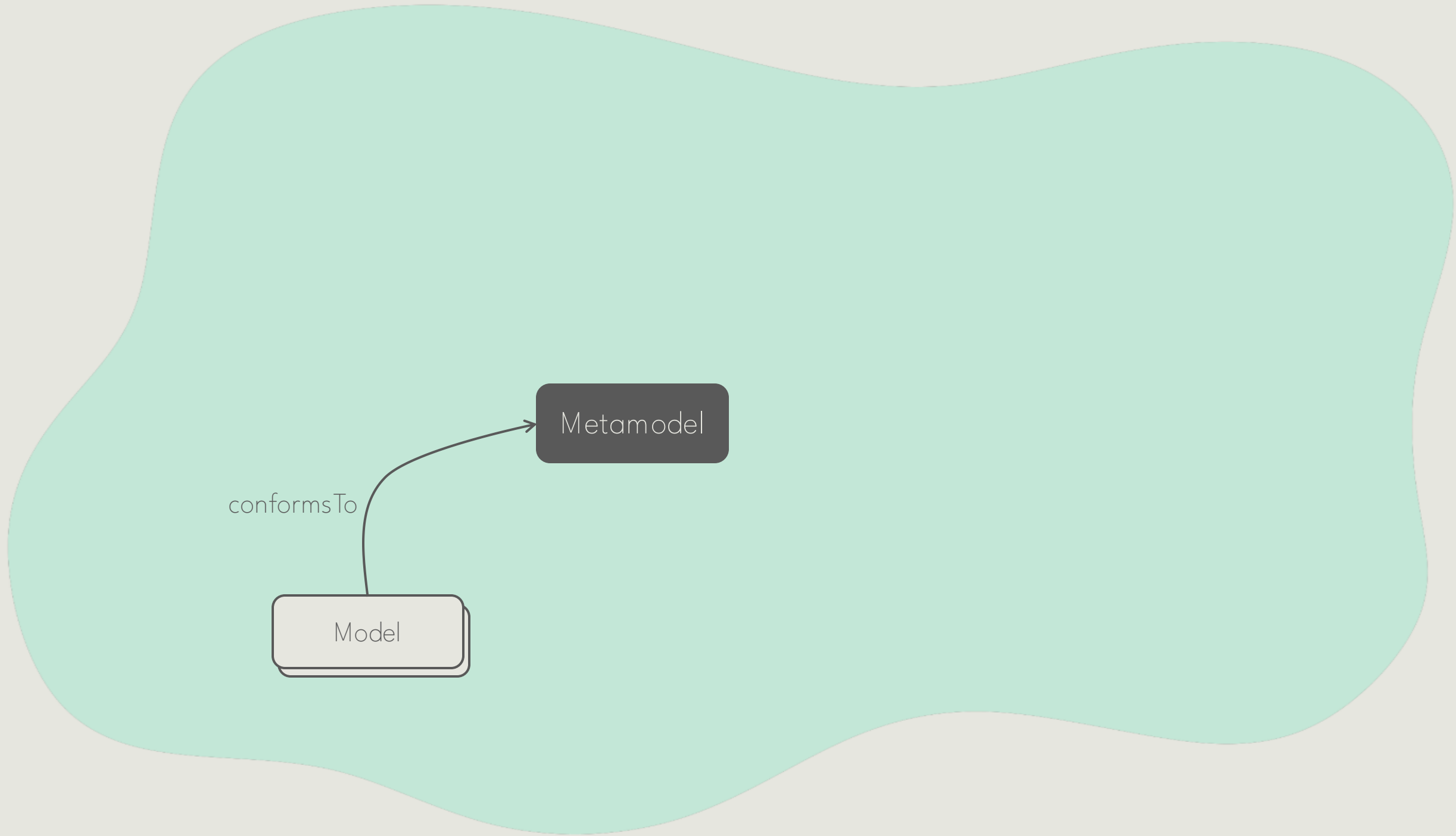
# Metamodels do not live in Isolation



---

Metamodels are primary assets in Model-Driven Engineering, defining the structure of related artifacts

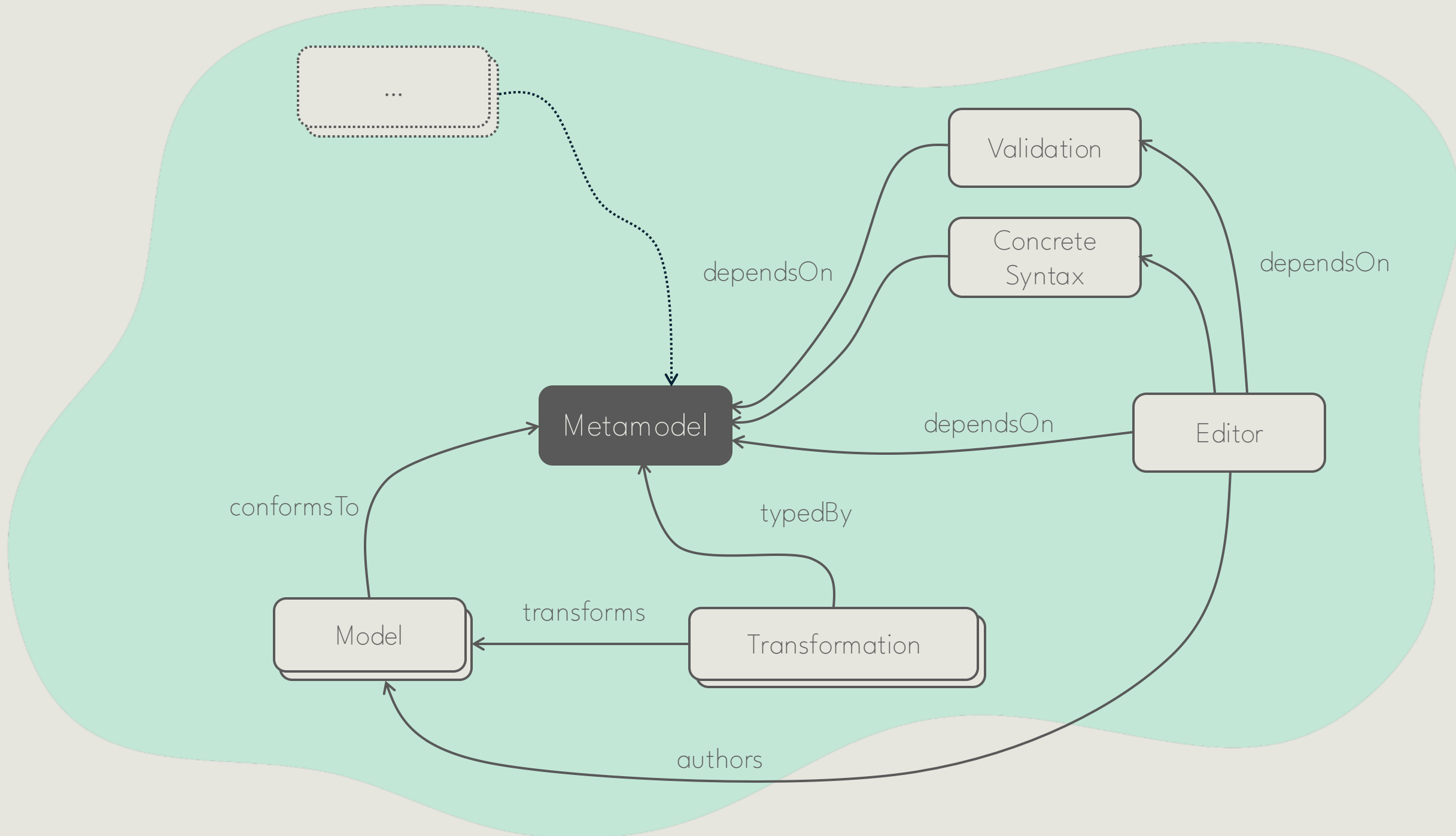
- A range of related artifacts interlaced with explicit or implicit correspondences
- Models, transformations, editors, and supporting tools can be regarded as part of an ecosystem pursuing a common scope



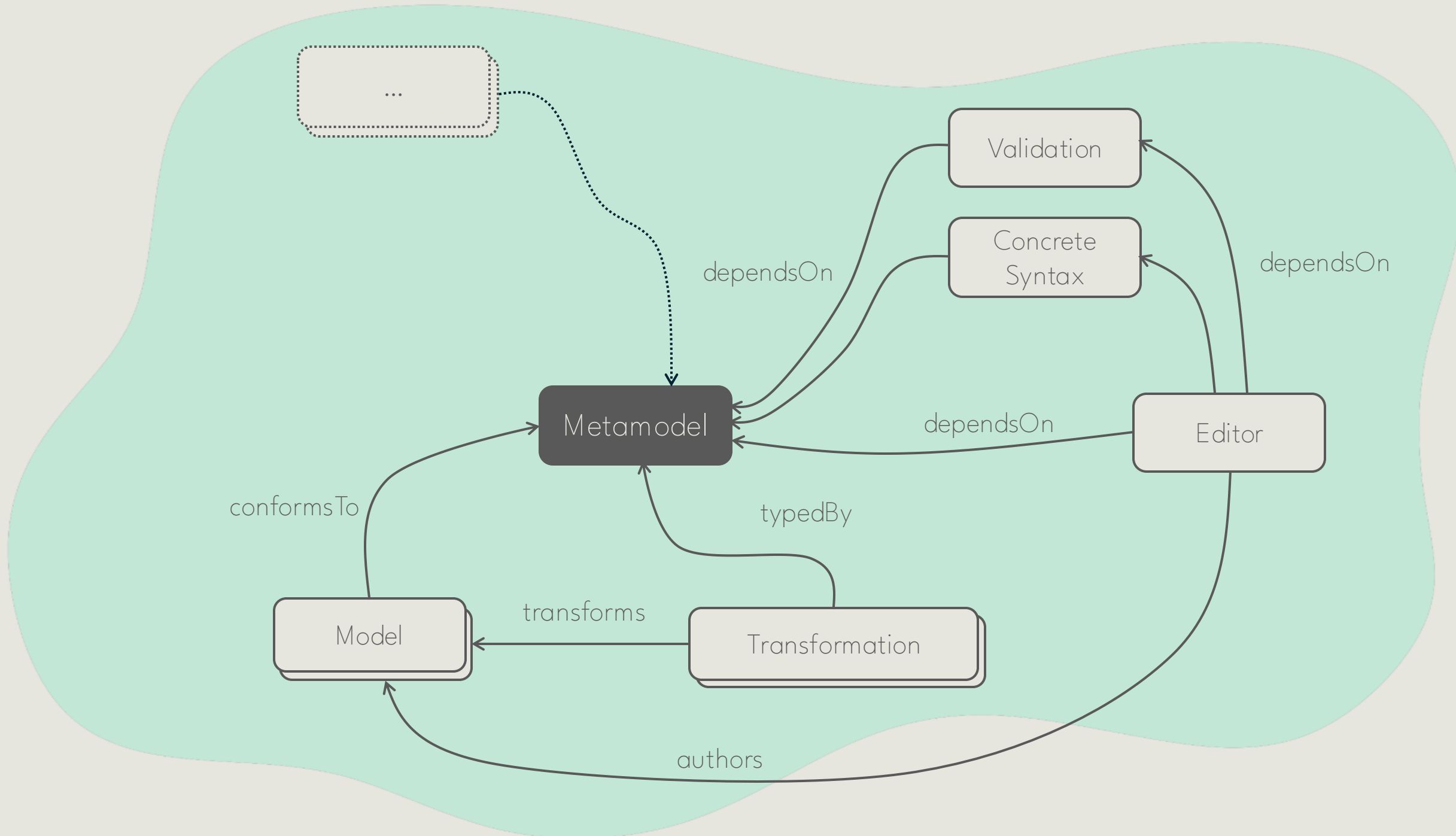
Model

Metamodel

conformsTo

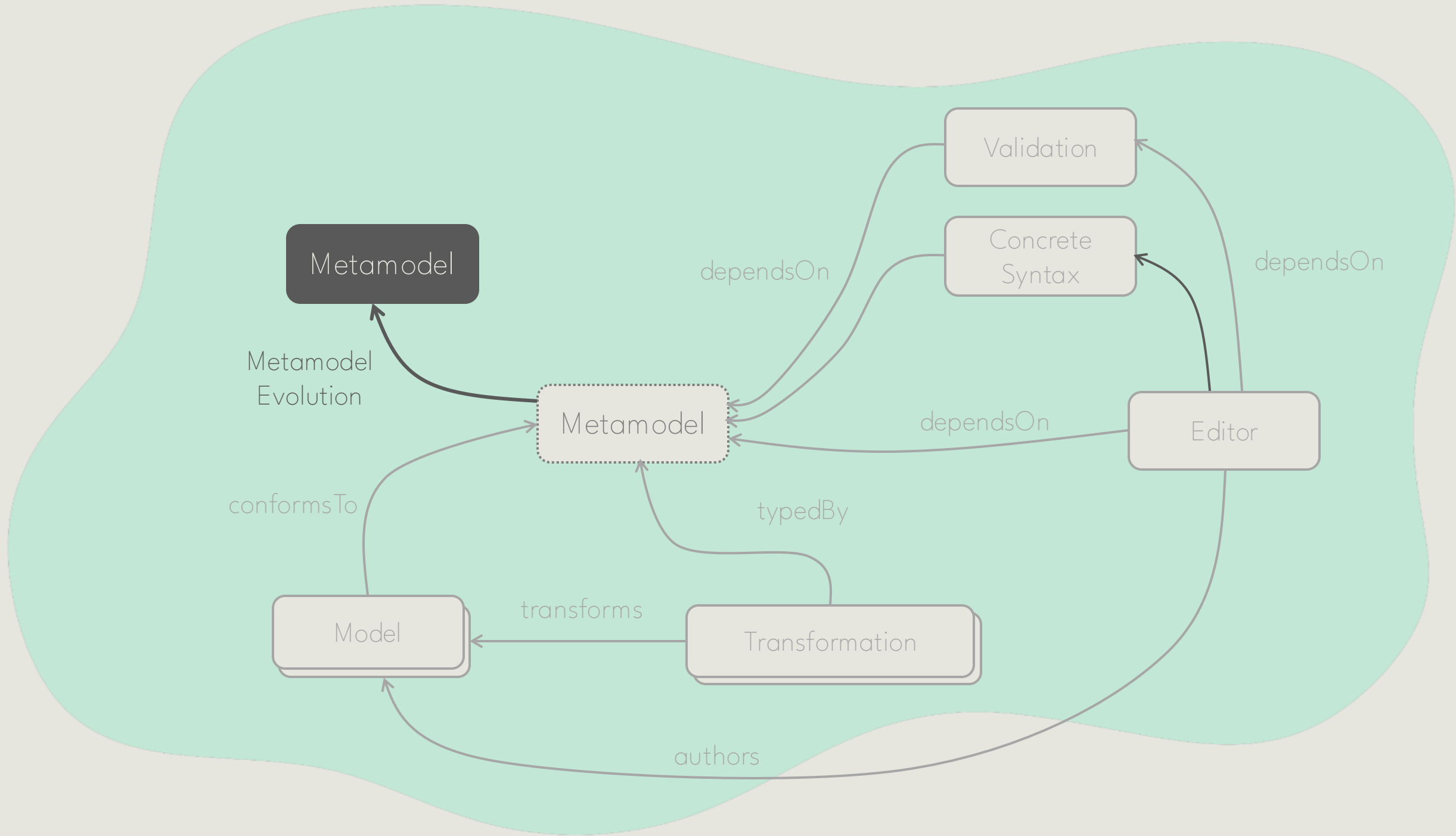


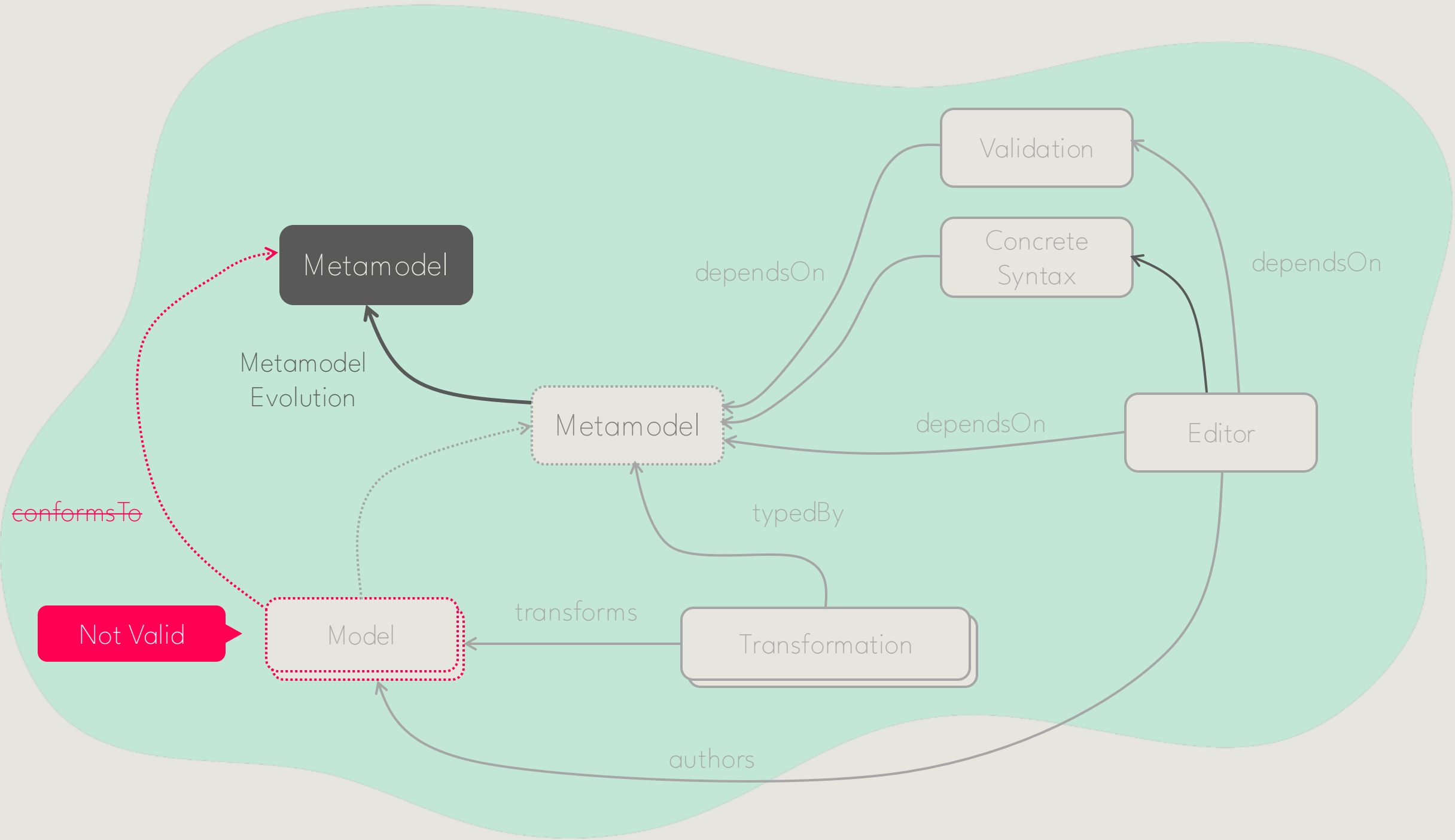


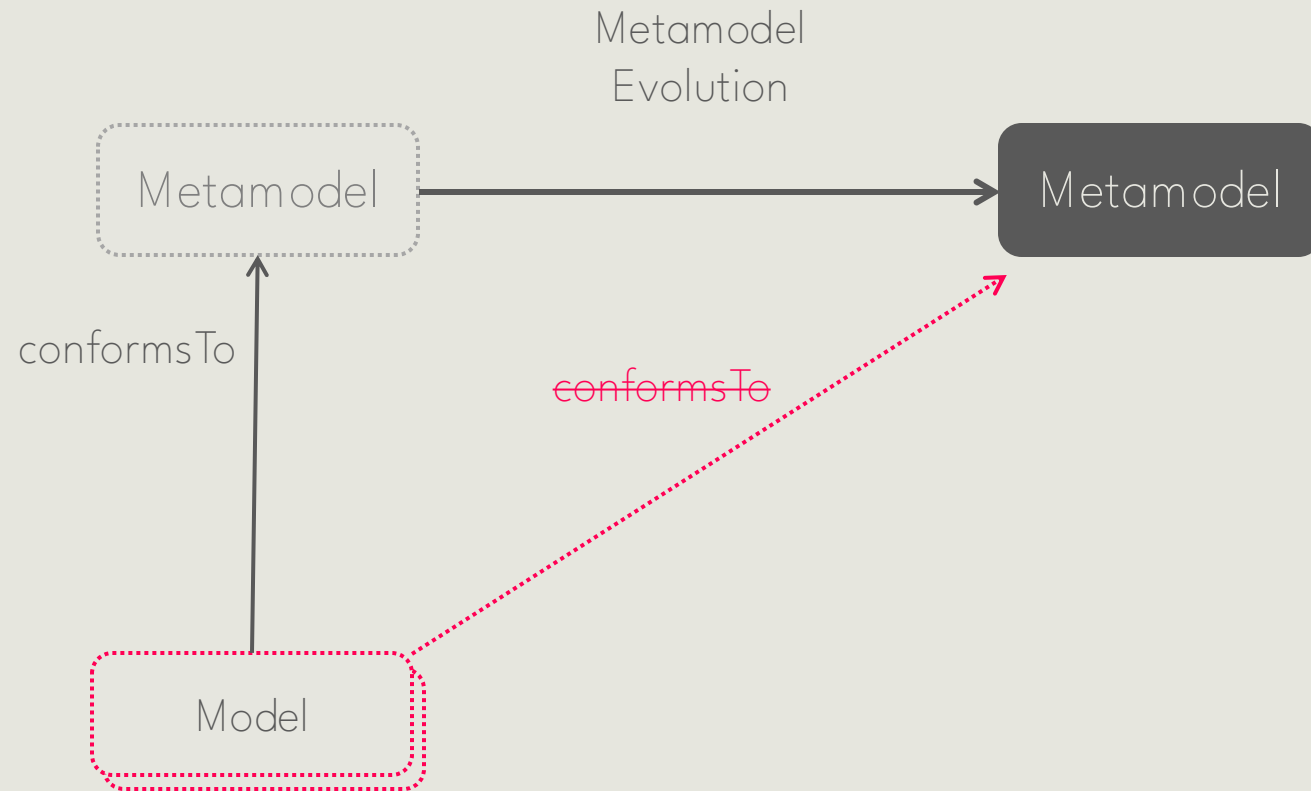


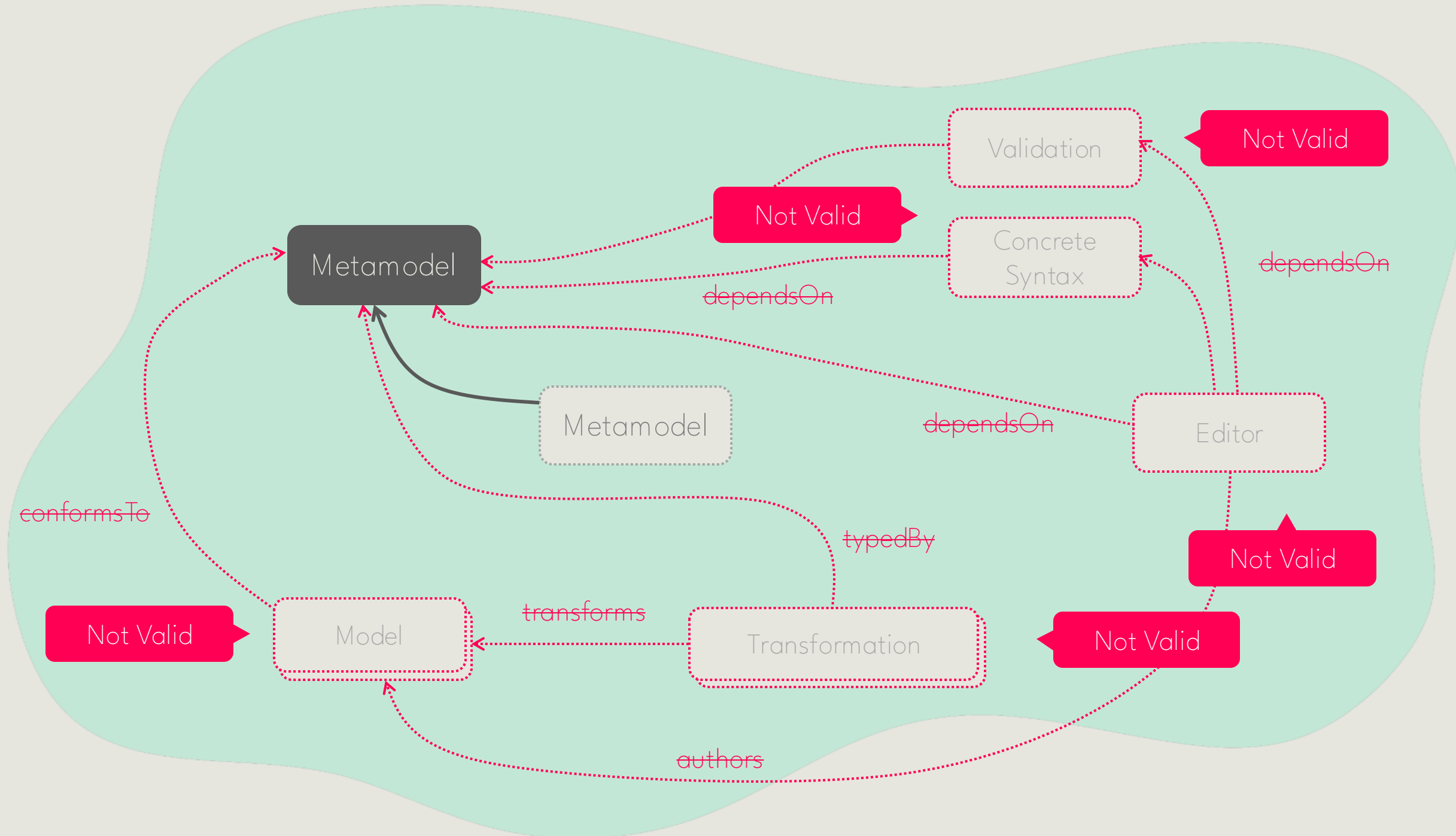
Software Evolution is Inevitable  
Metamodel Evolution too

A Change in a Metamodel does not stop at its Definition  
It ripples through the Entire Ecosystem









# Co-Evolution

---

Restoring Validity



# Co-evolution in MDE: 15 Years of Investigation



---

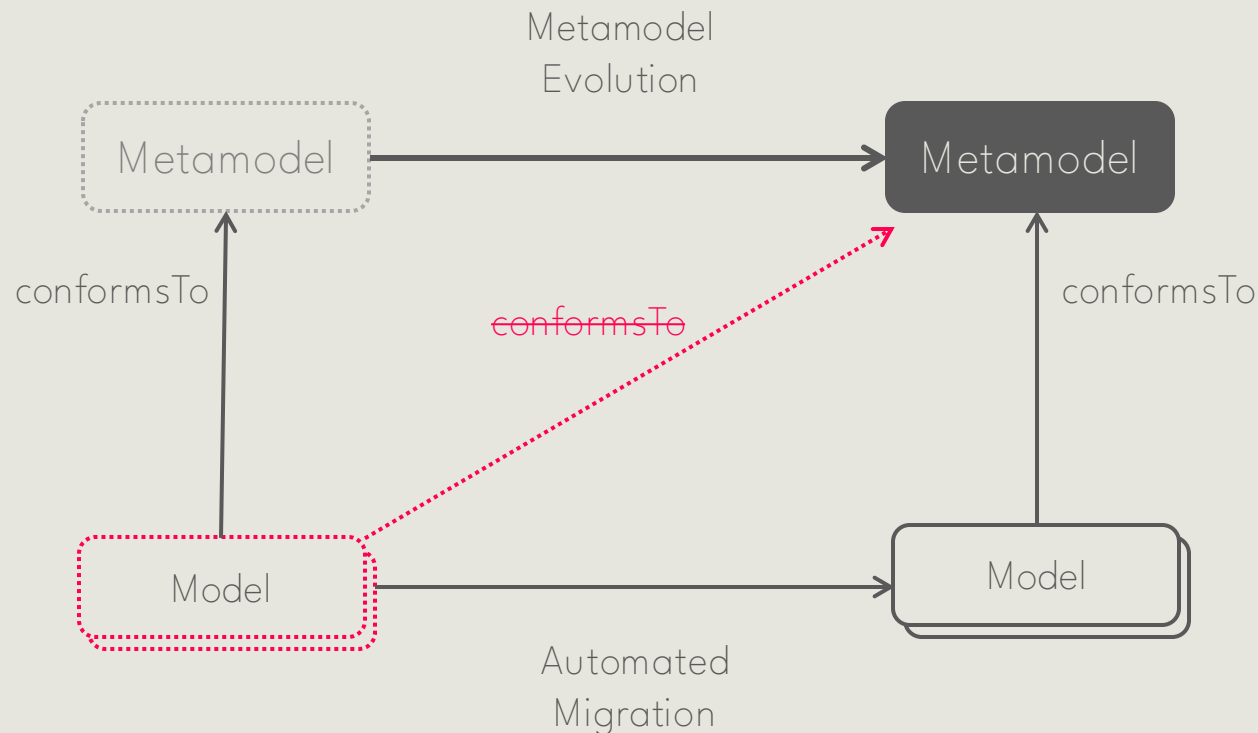
Initial work (late 2000s)

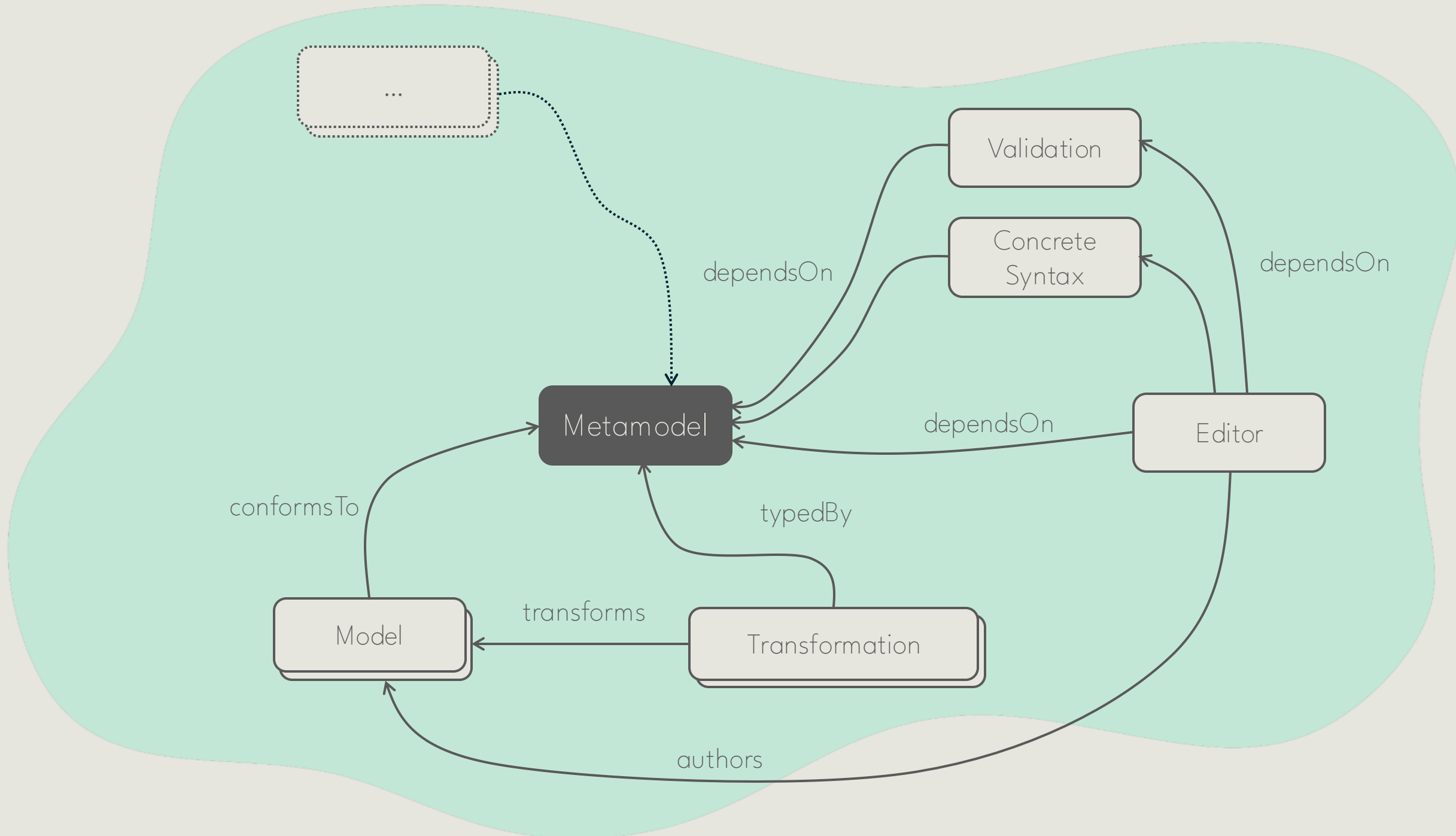
- Wachsmuth (2007), Cicchetti et al. (2008), Herrmannsdoerfer et al. (2009), Kolovos et al. (2010), Di Ruscio et al. (2012)
- Hebig et al. (2016): Survey

Over a decade and a half of research has transformed co-evolution from a repair mechanism into a foundation for adaptive, resilient modeling ecosystems

# Co-Evolution

Co-evolution refers to the **coordinated adaptation** of models and related artifacts when their metamodel evolves, to maintain conformance, validity, and semantic integrity





# Why It Matters?



---

A change in a metamodel can invalidate the models and related artifacts that depend on it

- Manual adaptation is error-prone and costly, leading to inconsistencies and information loss
- Automated co-evolution minimizes disruption and preserves model integrity

# Classification of Metamodel Changes



Metamodel changes can be classified according to their impact on the artifacts

1. Non-breaking changes: do not affect model conformance, eg adding optional elements
2. Breaking but resolvable changes: break conformance but can be automatically adapted, eg renaming or restructuring
3. Breaking and unresolvable changes: require human intervention e.g., adding mandatory properties)

# Classification of Metamodel Changes



---

The classification depends on the artifact kind: what matters for models, might not be relevant for other category of artifacts

- For instance, adding optional elements in models are non-breaking, whereas in editors and transformations compromises their coverage
- In editors, additional information may be needed



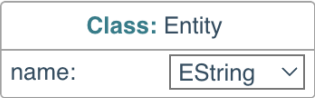
Metamodel Evolution

ERD v1

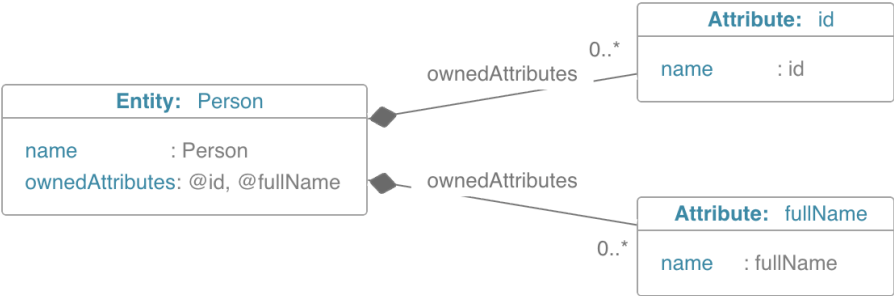
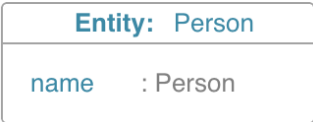


ERD v2

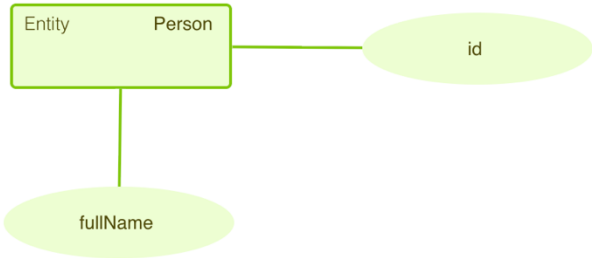
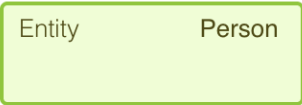
Metamodel



Model  
(abstract syntax)



Model  
(concrete syntax)





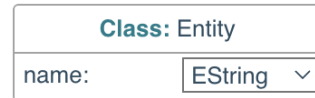
## Entity Relationship Diagram Metamodel

ERD v1

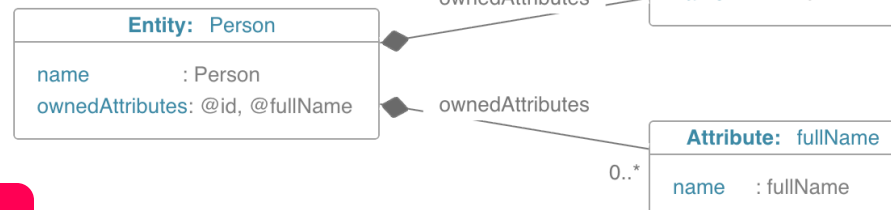
Metamodel Evolution

ERD v2

Metamodel



Model  
(abstract syntax)



Adding an optional element  
Non-breaking Change

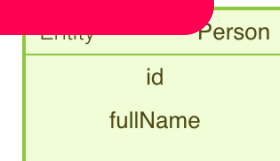
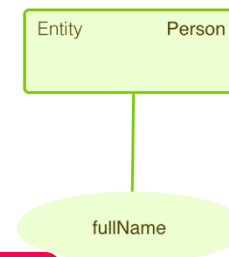
Instance unaffected

Logic Data Model

Over time model can evolve

Model  
(concrete syntax)

Conceptual Data Model



Syntax unaffected



# Approaches

# Approaches



---

They are mainly characterized by

- Category of Artifacts, eg Models, Transformations, Syntax/Editor, Validation
- Technique: State-based vs Operation-based Differencing

# State-Based Differencing

---

“What changed?”

State-based differencing  
supports reconstruction of  
evolution

# Operation-Based Differencing

---

“How did it change?”

Operation-based differencing  
enables automation of co-  
evolution

Kolovos, D. S., Di Ruscio, D., Pierantonio, A., & Paige, R. F.  
Different models for model matching: An analysis of approaches  
to support model differencing. In 2009 ICSE Workshop on  
Comparison and Versioning of Software Models

## State-Based

---

- Compares snapshots
- Differences in declarative format
- Post-hoc Analysis
- Descriptive
- Supports batch co-evolution

## Operation-Based

---

- Tracks edits
- Differences in procedural terms
- Live or recorded process
- Prescriptive
- Enables live co-evolution

Kolovos, D. S., Di Ruscio, D., Pierantonio, A., & Paige, R. F.  
Different models for model matching: An analysis of approaches  
to support model differencing. In 2009 ICSE Workshop on  
Comparison and Versioning of Software Models

# A State-based Approach



Effective co-evolution requires

- Detecting and classifying change types
- Decomposing differences into parallel independent (safe) and dependent (conflicting) sets
- Applying automated or guided transformations accordingly

# A State-based Approach



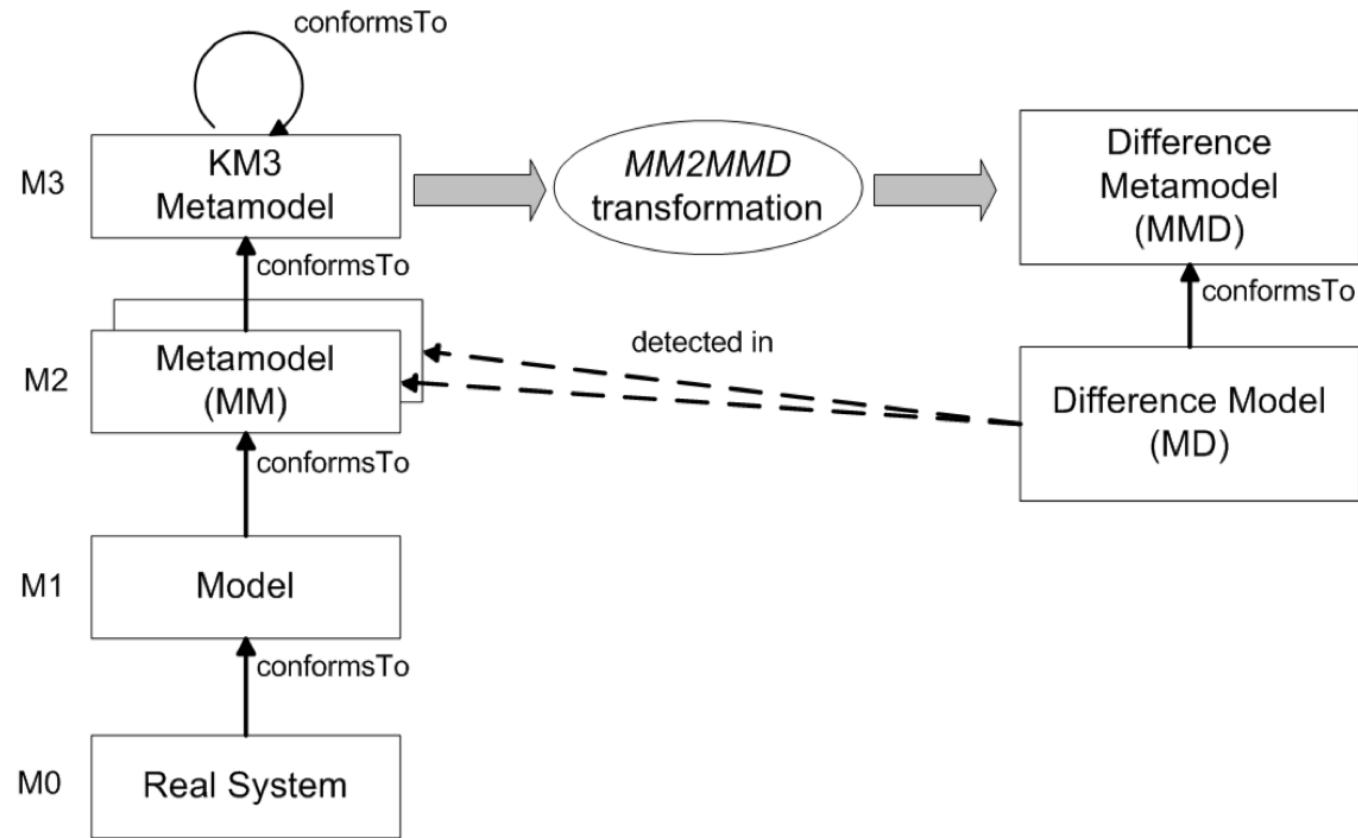
---

## Understanding the Changes

- Additive: new elements added (metaclass, property, superclass)
- Subtractive: elements removed (metaclass or property deletion)
- Updative: elements renamed, moved, or restructured

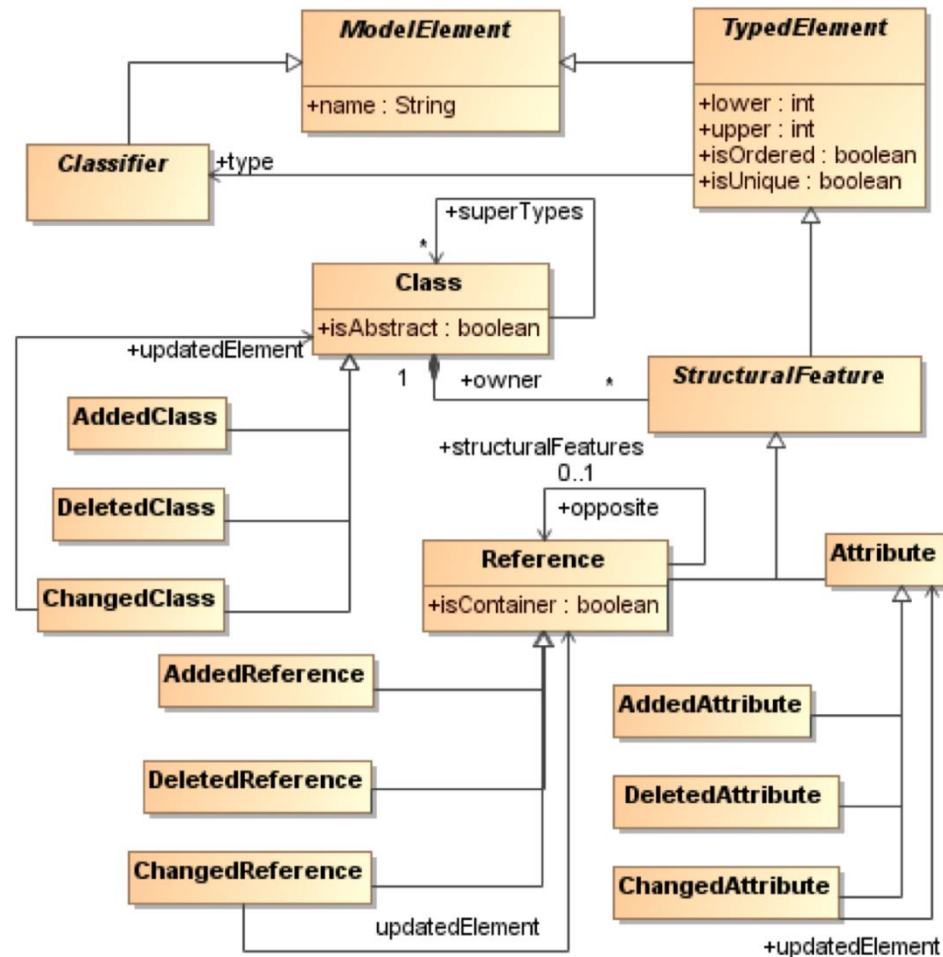
In order to automatize the artifact migration, a formal representation of differences is needed regardless whether declarative or procedural

# A State-Based Approach



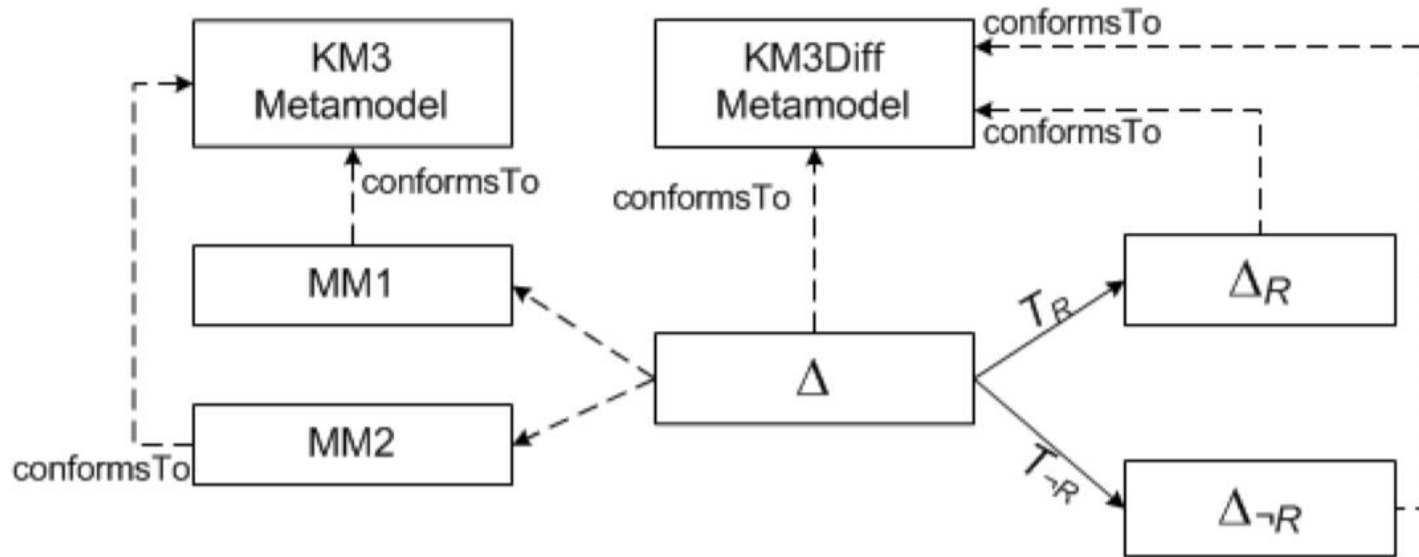
# A State-Based Approach

Difference  
Metamodel





# A State-Based Approach



Resolvable

Non Resolvable

# Batch vs Live Co-Evolution

---



Aspect	Batch Co-evolution	Live Co-evolution
Timing	After evolution	During evolution
Mode	Offline / post-hoc	Online / continuous
Basis	State-based differencing	Operation-based or event-driven
User role	Execute migration	Experience adaptation
Goal	Restore consistency	Preserve consistency
Paradigm	Can be declarative	Mainly heuristic-based

# Evaluating Co-evolution in Modeling Ecosystems



A framework for evaluating tool support for co-evolution of modeling languages, tools, and models

- The study exposes limits of batch-style (post-hoc) co-evolution
- Jjodel is positioned as a step toward live co-evolution
  - Reflective and self-descriptive
  - Supports in-situ adaptation
  - Reduces user effort and disruption during evolution

Tolvanen, J. P., Kelly, S., Di Rocco, J., Pierantonio, A., & Tinella, G. (2025). A framework for evaluating tool support for co-evolution of modeling languages, tools and models. *Software and Systems Modeling*, 24(2)

# Live Co-Evolution in Jodel

# Jjodel: A Reflective Cloud Platform for Model-Driven Engineering



Jjodel is a cloud-based reflective platform that simplifies MDE while maintaining expressiveness and power

- Designed to bridge the gap between usability and complexity in both educational and industrial contexts
- Integrated SaaS workbench for creating and evolving domain-specific languages (DSLs)



Bucchiarone, A., Di Rocco, J., Di Vincenzo, D. Pierantonio, A.  
Modeling in Jjodel: towards bridging complexity and usability  
in model-driven engineering. *Softw Syst Model* (2025).  
<https://doi.org/10.1007/s10270-025-01324-y>

# Demo



We will create an Entity Relationship Diagram (ERD) metamodel

- It is a rather simple metamodel
- The idea is to give a flavor of what live co-evolution is about and how it works in Jjodel

# Lehman's Laws Rivisted for MDE








---

Lehman's laws remind us that evolution is inevitable, co-evolution and live adaptation can make it sustainable

# Lehman's Laws Rivisted for MDE



---

-  Continuing Change, metamodels and models must co-evolve to stay relevant
-  Increasing Complexity, Co-evolution mechanisms aim to contain and manage this complexity
-  Fundamental Law of Evolution, Modeling ecosystems exhibit similar feedback loops between metamodels, models, and tools
-  Conservation of Organizational Stability, Teams sustain an invariant rhythm of adaptation in MDE ecosystems
-  Conservation of Familiarity, Live co-evolution supports this by adapting tools and models seamlessly, without disruptive migrations



# Conclusions



Despite co-evolution has been investigated for over 15 years a definitive solution is not at sight, although tools like Jodel, MetaEdit+, and MPS provide excellent support to it

A basic prerequisite for co-evolution whether batch or live is having at least an overarching architecture for efficiently managing dependencies, reflectiveness is an advantage

—An interesting perspective is represented by DesignSpaces (Egyed et al. 2018)



Thank you!



Right

Bottom

Up

Left

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. Proceedings of the IEEE, 68(9), 1060-1076.