Lecture series on
Software knowledge analytics

Megamodeling

(aka Linguistic Software Architecture)

Ralf Lämmel, Uni Koblenz, May 2022

Looking back at the outlook

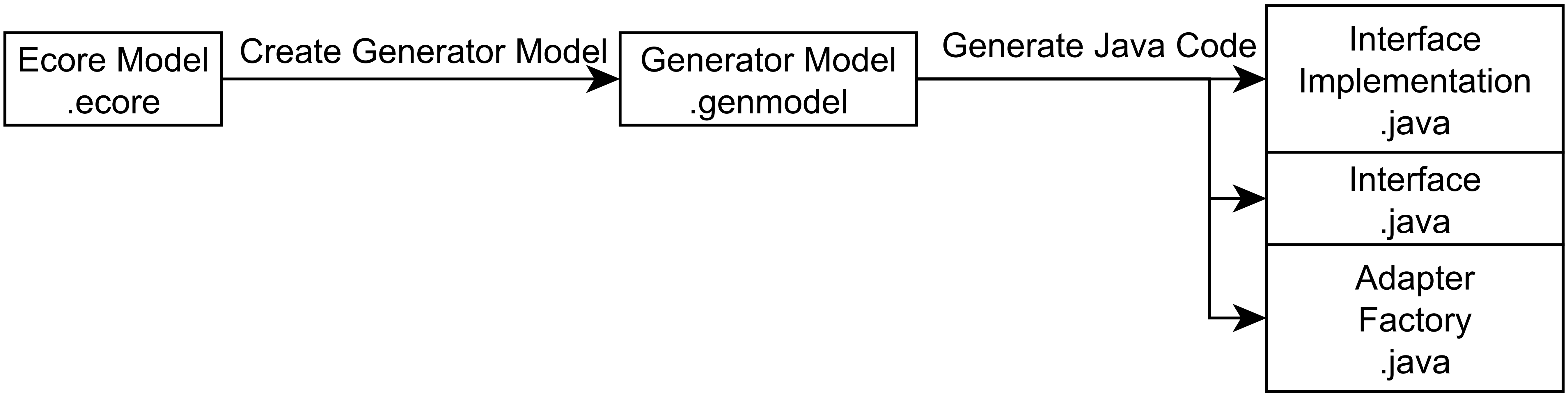
Technical topics for this lecture series

- API clustering — DONE
- Joint API usage — DONE
- Graph language proliferation — DONE (covered bits in inaugural lecture)
- Knowledge graph validation — SKIP (overall topic less context to analytics)
- Classifier discovery on Wikipedia — DONE (covered bits in inaugural lecture)
- Developer workflow modeling — DONE (covered bits in inaugural lecture)
- Linguistic architecture recovery — TODO (extending on bits from inaugural lecture)
- Simulation of MSR/ESE studies — TODO on Friday
- Multimodeling regression analysis — SKIP (not ready this time around)
- API developer profiles — DONE

Let's do a full-blown lecture on megamodeling

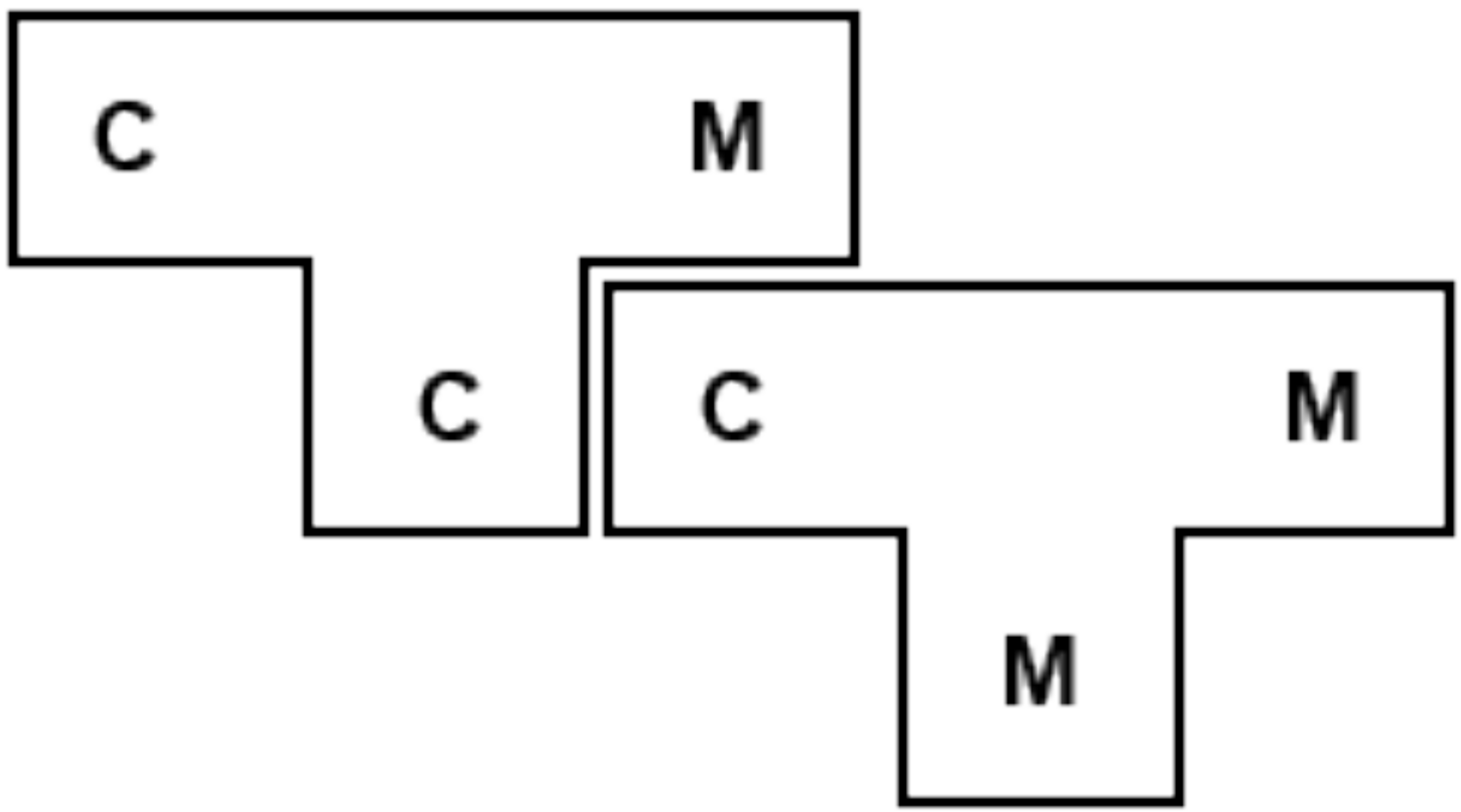

What's a
megamodel?

A megamodel for EMF code generation

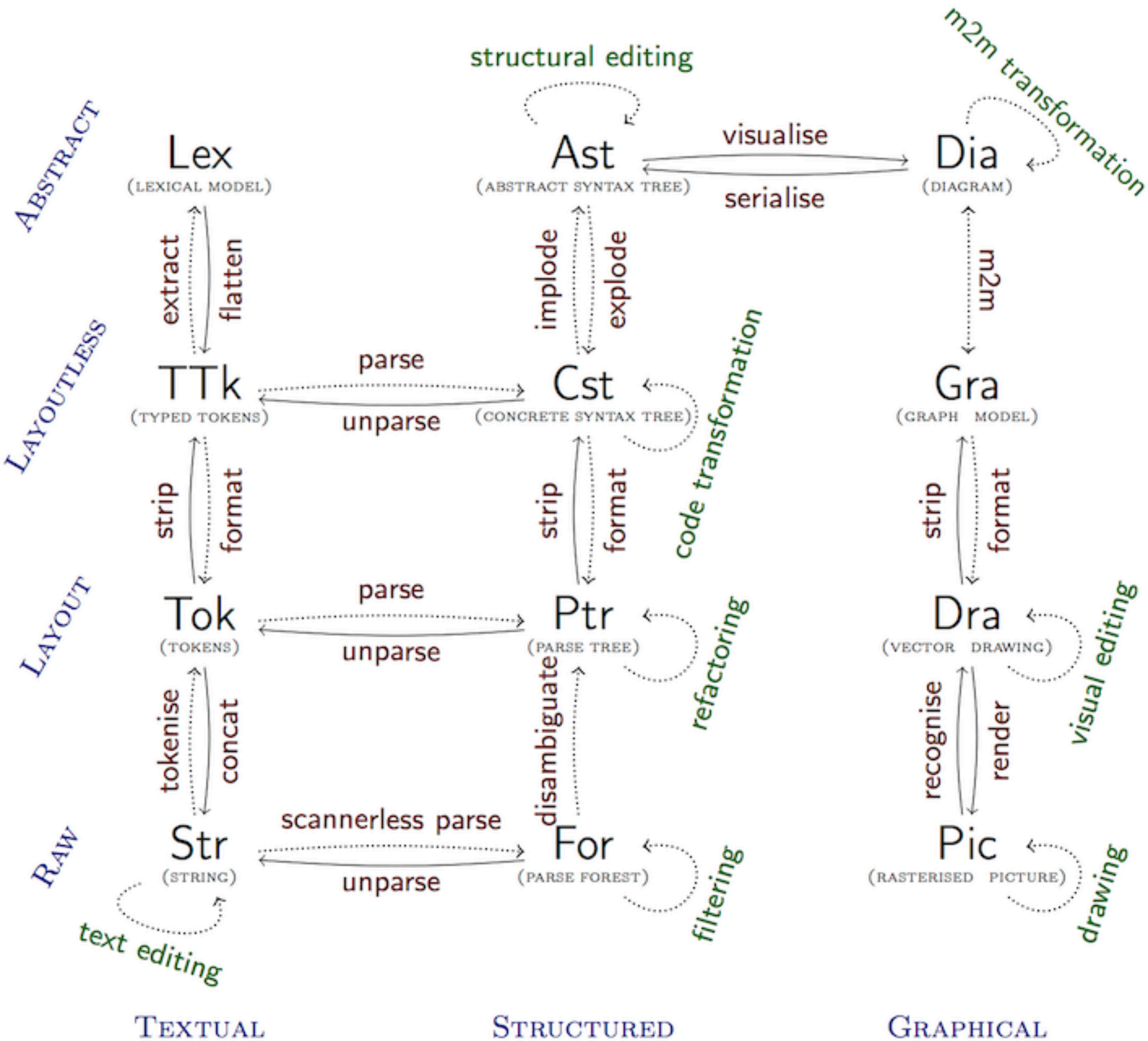


Source: Marcel Heinz, Johannes Härtel, Ralf Lämmel: **Reproducible Construction of Interconnected Technology Models for EMF Code Generation**. J. Object Technol. 19(2): 8:1-25 (2020). See also conference version: Johannes Härtel, Marcel Heinz, Ralf Lämmel: EMF Patterns of Usage on GitHub. ECMFA 2018: 216-234

A megamodel for compiler bootstrapping



A megamodel for parsing in a broad sense



Source: Vadim Zaytsev, Anya Helene Bagge: **Parsing in a Broad Sense.** MoDELS 2014: 50-67

A megamodel for ANTLR usage

ANTLR : ***Technology*** *// The technology as a conceptual entity*
Java : ***Language*** *// The language targeted by the parser generator*
ANTLR.Notation : ***Language*** *// The language of parser specifications*
ANTLR.Generator : ***Function*** *(ANTLR.Notation* $\rightarrow$ *Java)*
?aLanguage : ***Language*** *// Some language being modeled with ANTLR*
?aGrammar : ***File*** *// Some grammar defining the language at hand*
?aParser : ***File*** *// The generated parser for the language at hand*
?anInput : ***File*** *// Some sample input for the parser at hand*

ANTLR.Notation ***partOf*** *ANTLR // Notation is conceptual part of technology*
ANTLR.Generator ***partOf*** *ANTLR // Generator semantics as well*

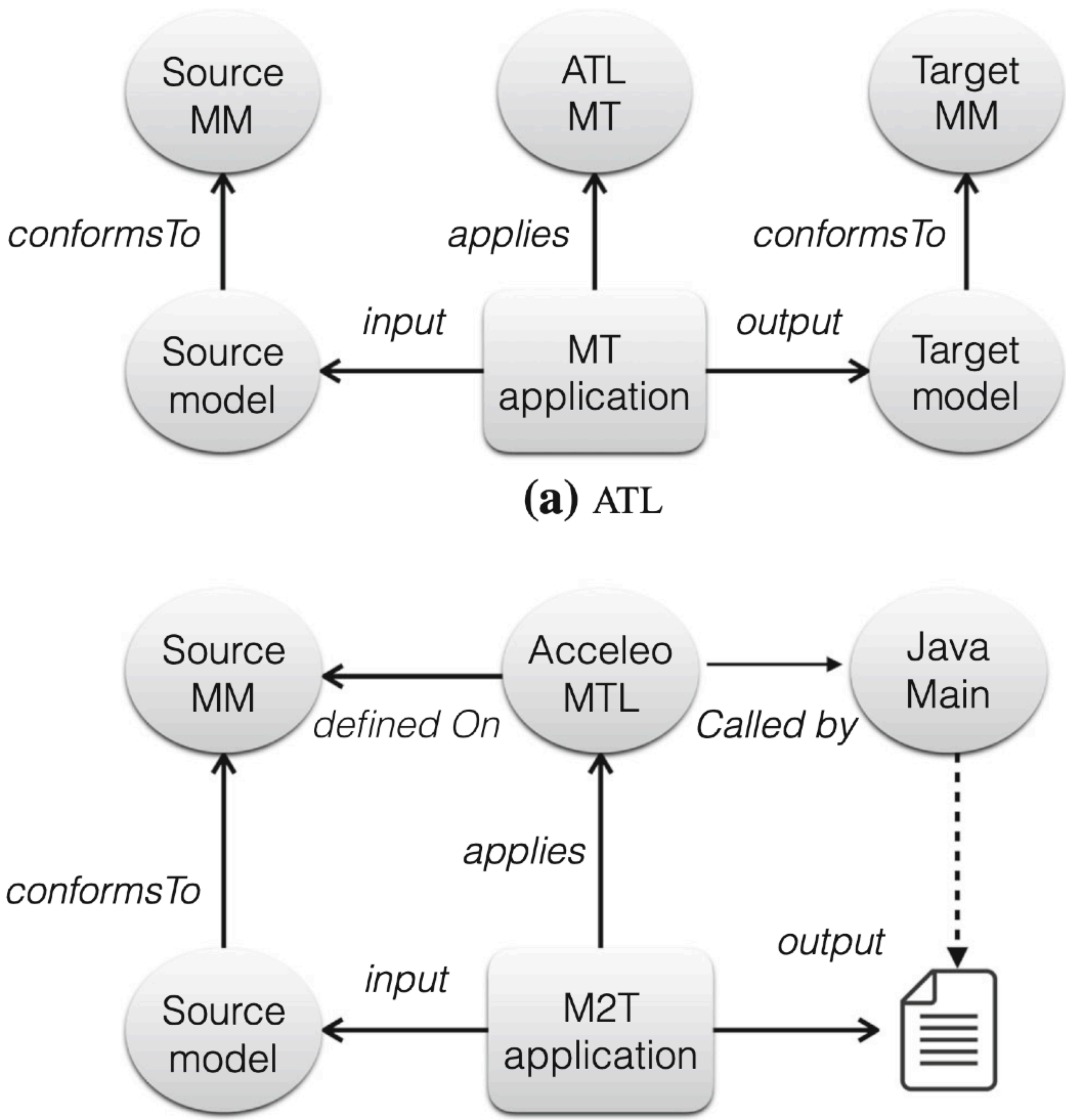
ANTLR.Notation ***domainOf*** *ANTLR.Generator*
Java ***rangeOf*** *ANTLR.Generator*

aGrammar ***elementOf*** *ANTLR.Notation // The grammar is given in ANTLR notation*
aGrammar ***defines*** *aLanguage // The grammar defines some language*
aParser ***elementOf*** *Java // Java is used for the generated parser*
ANTLR.Generator(aGrammar) $\mapsto$ *aParser // Generate parser from grammar*
anInput ***elementOf*** *aLanguage // Wanted! An element of the language*
anInput ***conformsTo*** *aGrammar // Conform also to the grammar*

ANTLR.GeneratorApp1 : ***FunctionApplication***
ANTLR.GeneratorApp1 ***elementOf*** *ANTLR.Generator*
aGrammar ***inputOf*** *ANTLR.GeneratorApp1*
aParser ***outputOf*** *ANTLR.GeneratorApp1*

Source: Ralf Lämmel, Andrei Varanovich: **Interpretation of Linguistic Architecture**. ECMFA 2014: 67-82

A megamodel for MT with ATL/Acceleo

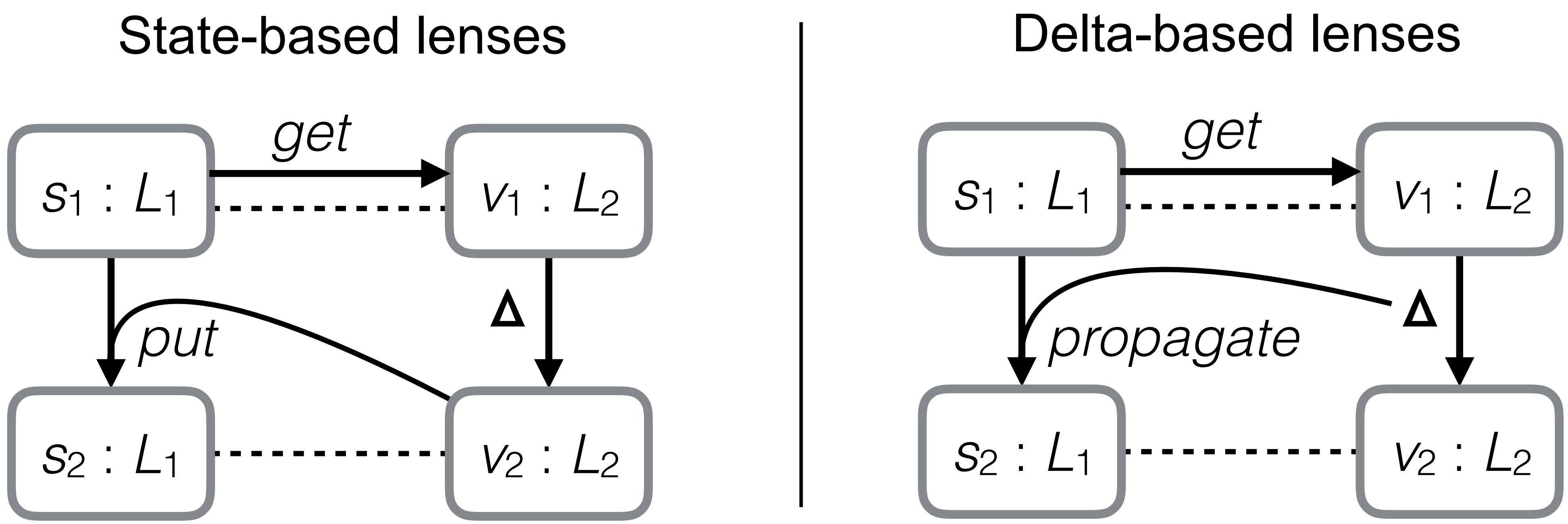


(a) ATL

(b) Acceleo

Source: Juri Di Rocco, Davide Di Ruscio, Johannes Härtel, Ludovico Iovino, Ralf Lämmel, Alfonso Pierantonio: **Understanding MDE projects: megamodels to the rescue for architecture recovery**. Softw. Syst. Model. 19(2): 401-423 (2020). See also conference version: Juri Di Rocco, Davide Di Ruscio, Johannes Härtel, Ludovico Iovino, Ralf Lämmel, Alfonso Pierantonio: Systematic Recovery of MDE Technology Usage. ICMT 2018: 110-126

Megamodels for two basic BX patterns



In the first (more basic) BX pattern, *get* maps a source to a view; *put* maps back a changed view to a source while taking into account the original source so that BX can go beyond bijective functions. In the second (more detailed) BX pattern, *put* has been replaced by a decomposition of differencing and change propagation.

Source: Ralf Lämmel: **Coupled software transformations revisited**. SLE 2016: 239-252

Megamodels for two basic BX patterns

LAL megamodel *bx.state*

```
reuse cx.mapping [ mapping ↦ get ]
function get : L₁ → L₂
function put : L₁ × L₂ ⇸ L₁
axiom GetPut { ∀s ∈ L₁.
  put(s, get(s)) = s }
axiom PutGet { ∀s₁, s₂ ∈ L₁. ∀v ∈ L₂.
  put(s₁, v) = s₂ ⇒ get(s₂) = v }
```

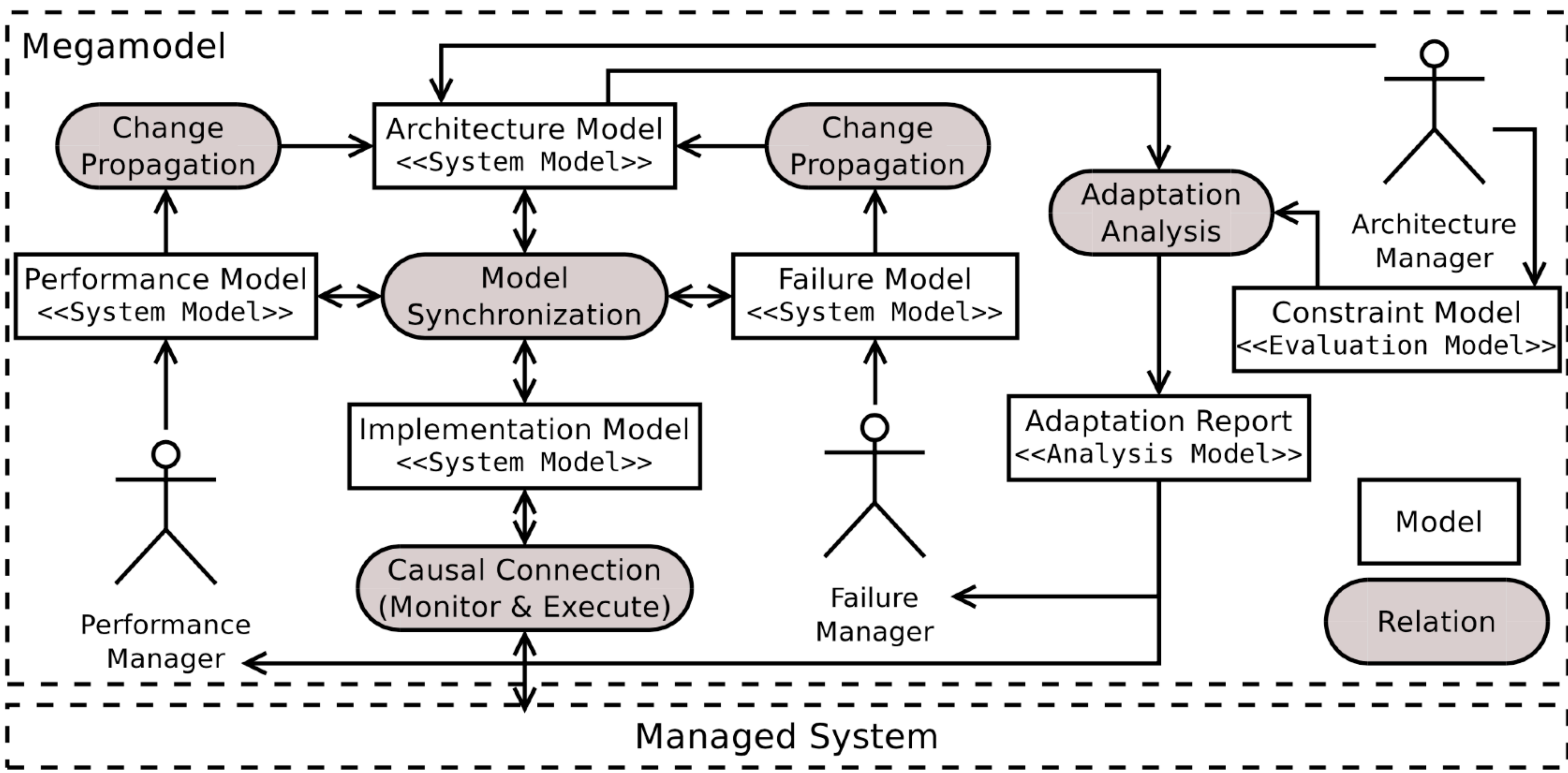
More on this later!

LAL megamodel *bx.delta*

```
reuse bx.state
reuse differencing [ L ↦ L₂, Any ↦ Any₂ ]
function propagate : L₁ × DiffL ⇸ L₁
axiom { ∀s₁, s₂ ∈ L₁. ∀v₁, v₂ ∈ L₂. ∀delta ∈ DiffL.
  get(s₁) = v₁
  ∧ diff(v₁, v₂) = delta
  ∧ propagate(s₁, delta) = s₂ ⇒
        put(s₁, v₂) = s₂ ∧ get(s₂) = v₂ }
```

Source: Ralf Lämmel: **Coupled software transformations revisited**. SLE 2016: 239-252

A megamodel for a self-adaptive software system (Models@Runtime)



Source: https://arxiv.org/abs/1805.07396

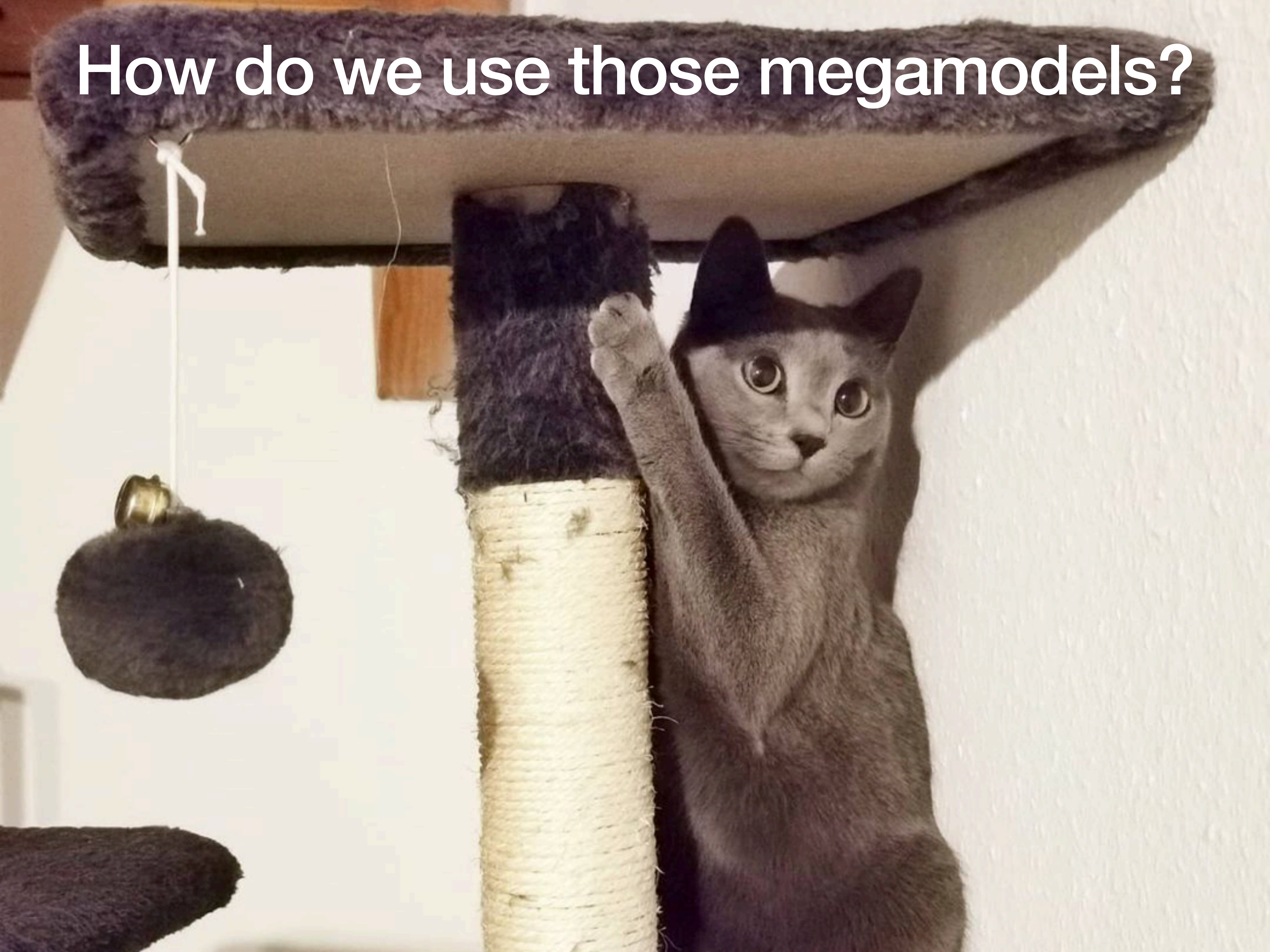
Megamodels in the wild

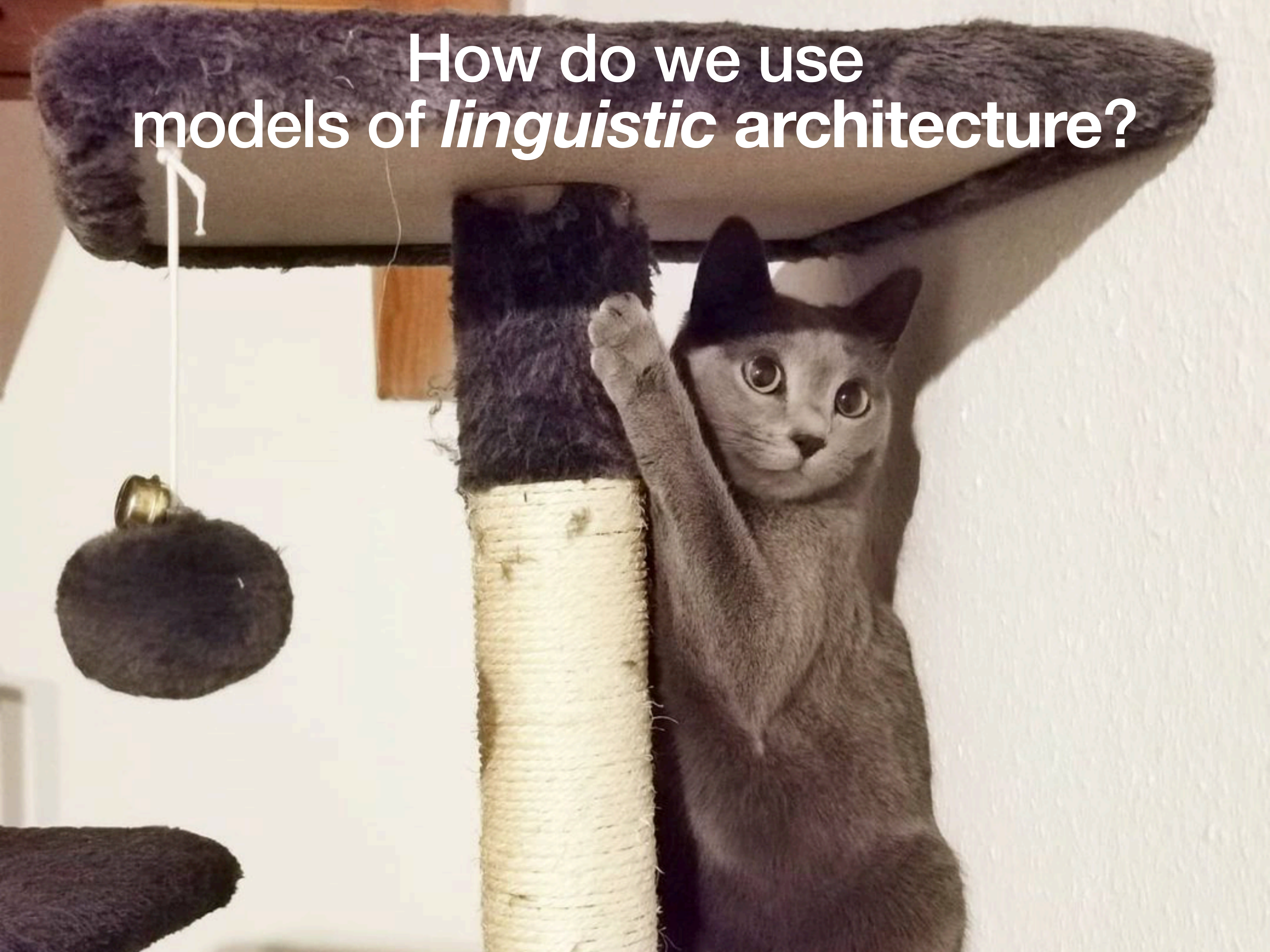
- Central service registry
- DB shard management
- ML workflow management
- Data pipeline management
- Configuration
- Package management
- Release management
- …

… basically some forms of DevOps through UI and CLI.

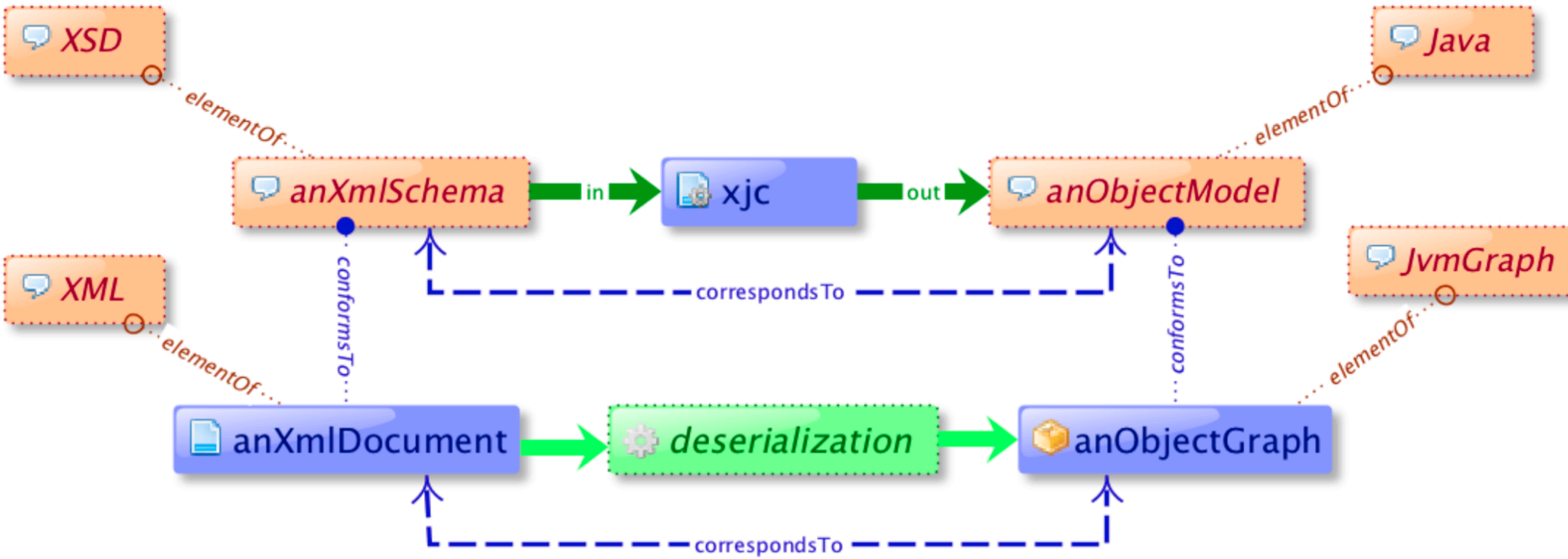
A lot of diversity!

- What are model elements (nodes)?
- What are relationships (edges)?
- What's the technical space, if not modelware?
- Is the model an abstraction?
- How to instantiate the model?
- How to validate the model?
- Does the model run?
- Is the model part of the system?
- …


How do we use those megamodels?

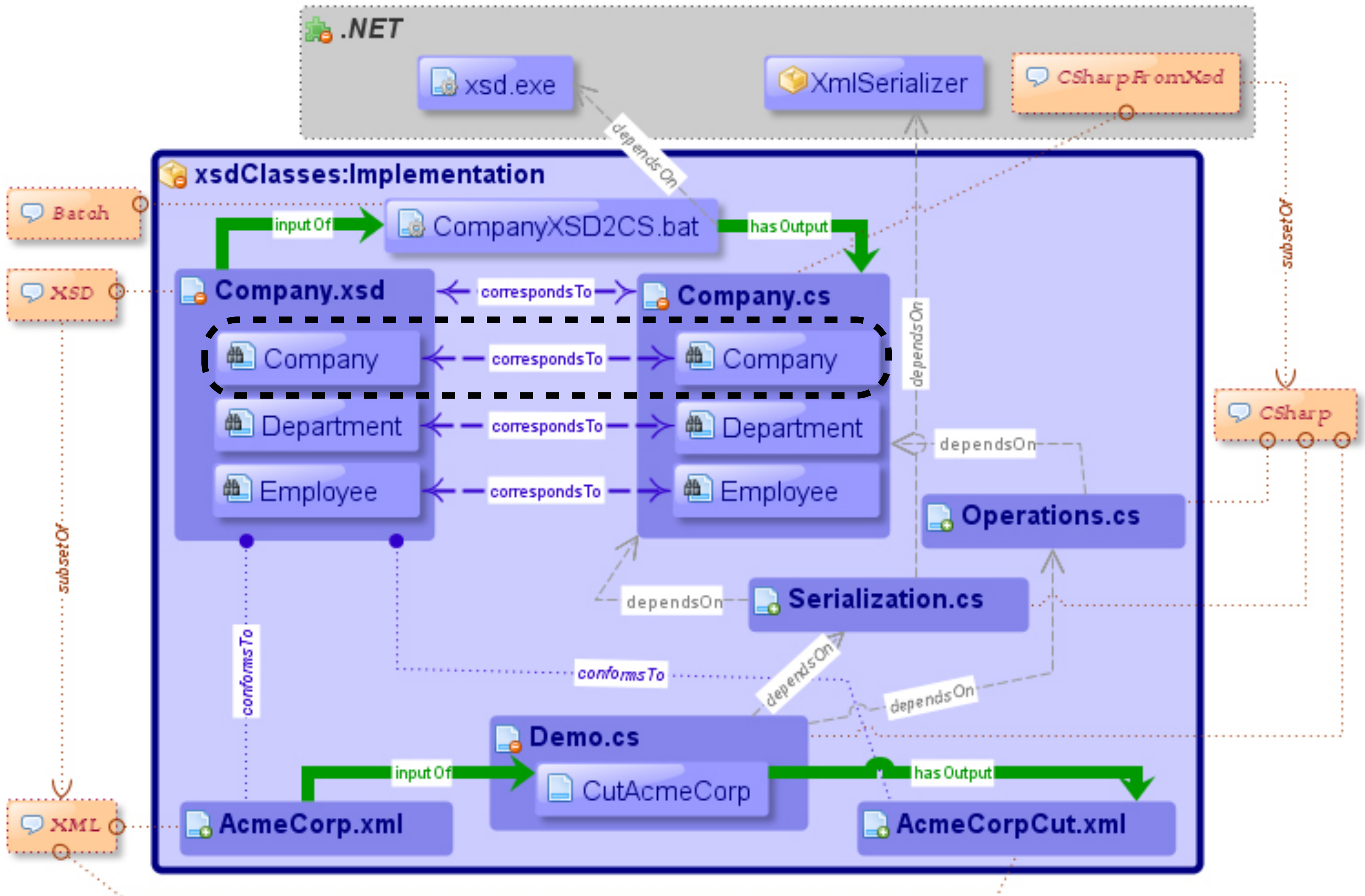

How do we use
models of *linguistic* architecture?

Linguistic architecture of XML-data binding in Java (A general megamodel — before “instantiation”)



Source: Jean-Marie Favre, Ralf Lämmel, Andrei Varanovich: Modeling the Linguistic Architecture of Software Products. MoDELS 2012: 151-167

… XML-data binding in C#



Source: Jean-Marie Favre, Ralf Lämmel, Andrei Varanovich: **Modeling the Linguistic Architecture of Software Products**. MoDELS 2012: 151-167

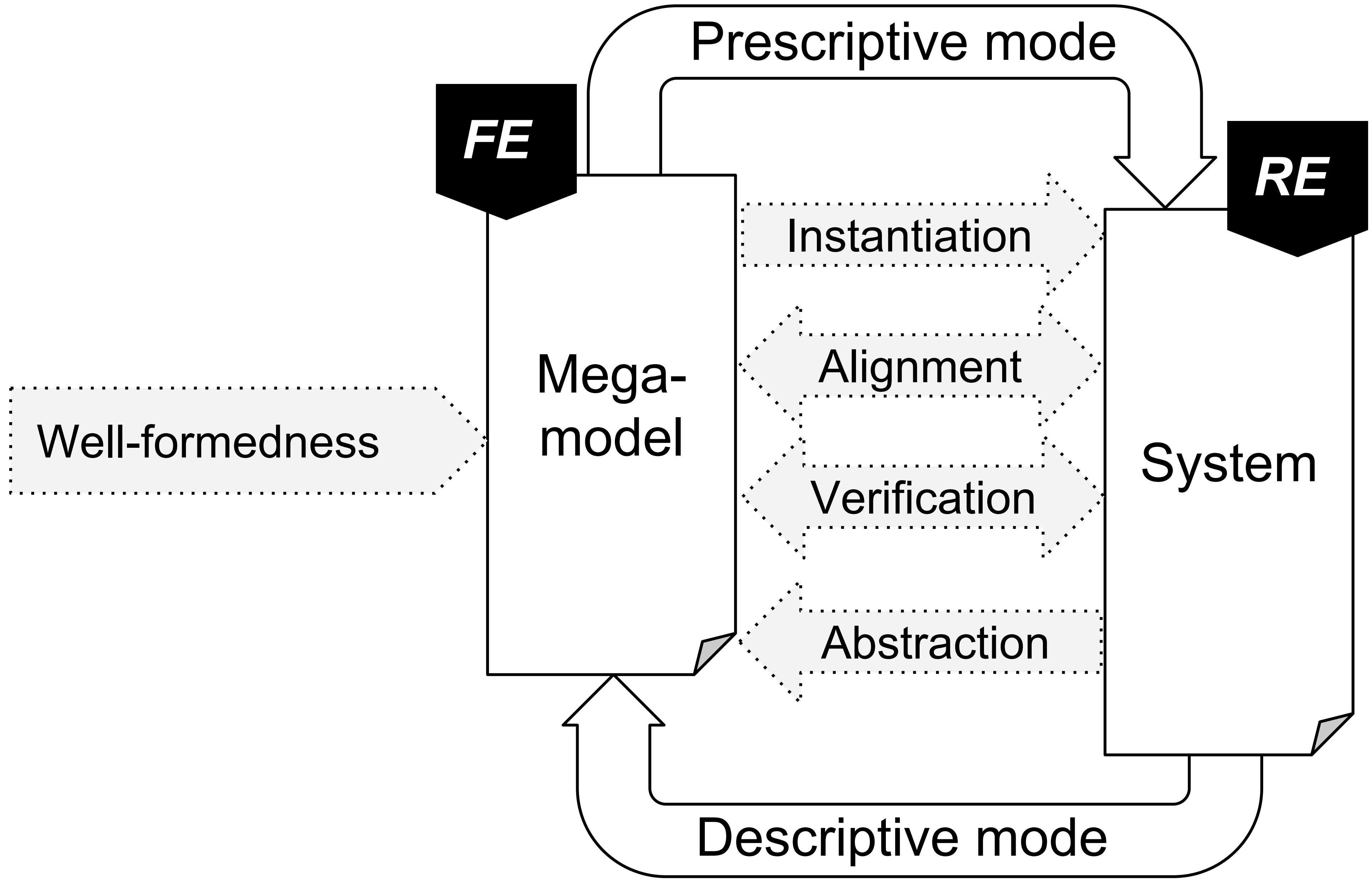
... XML-data binding in C#

```
_:xmlTypes rdf:type mgl:File .
_:xmlTypes rdfs:label "xmlTypes" .
_:xmlTypes mgl:elementOf lang:XSD .
_:xmlTypes mgl:inputOf _:classgen .

_:xmlDoc rdf:type mgl:File .
_:xmlDoc rdfs:label "xmlDoc" .
_:xmlDoc mgl:elementOf lang:XML .
_:xmlDoc mgl:conformsTo _:xmlTypes .
_:xmlDoc mgl:inputOf _:classgen .

_:classgen_app_1 rdf:type mgl:FunctionApplication .
_:classgen_app_1 rdfs:label "classgen" .
_:classgen_app_1 rdf:elementOf _:classgen .
_:classgen_app_1 rdf:hasOutput _:ooTypes .
```

```
_:CompanyDotXSD rdf:type mgl:File .
_:CompanyDotXSD rdfs:label "Company.xsd" .
_:CompanyDotXSD mgl:elementOf lang:XSD .
_:CompanyDotXSD mgl:inputOf _:CompanyXSD2CSDotBat .
_:CompanyElement mgl:partOf _:CompanyDotXSD .
_:CompanyElement rdf:type mgl:FileFragment .
_:CompanyElement rdfs:label "Company" .
... other fragments omitted ...

_:CompanyDotXSD mgl:partOf impl:xsdClasses .
_:CompanyDotXSD mgl:filename "./Company.xsd" .
_:CompanyElement mgl:xpathLocation
                    "//*[@name=\"Company\"]" .
```

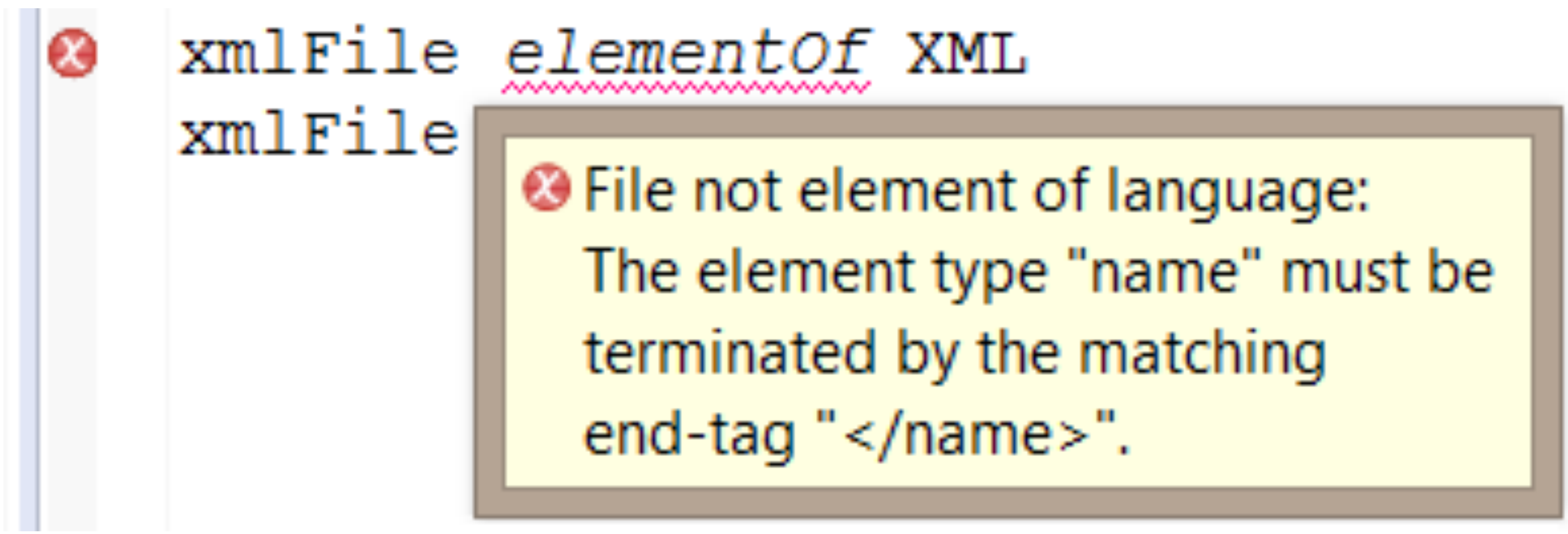
Source: Jean-Marie Favre, Ralf Lämmel, Andrei Varanovich: **Modeling the Linguistic Architecture of Software Products**. MoDELS 2012: 151-167

Linguistic architecture in a software development context



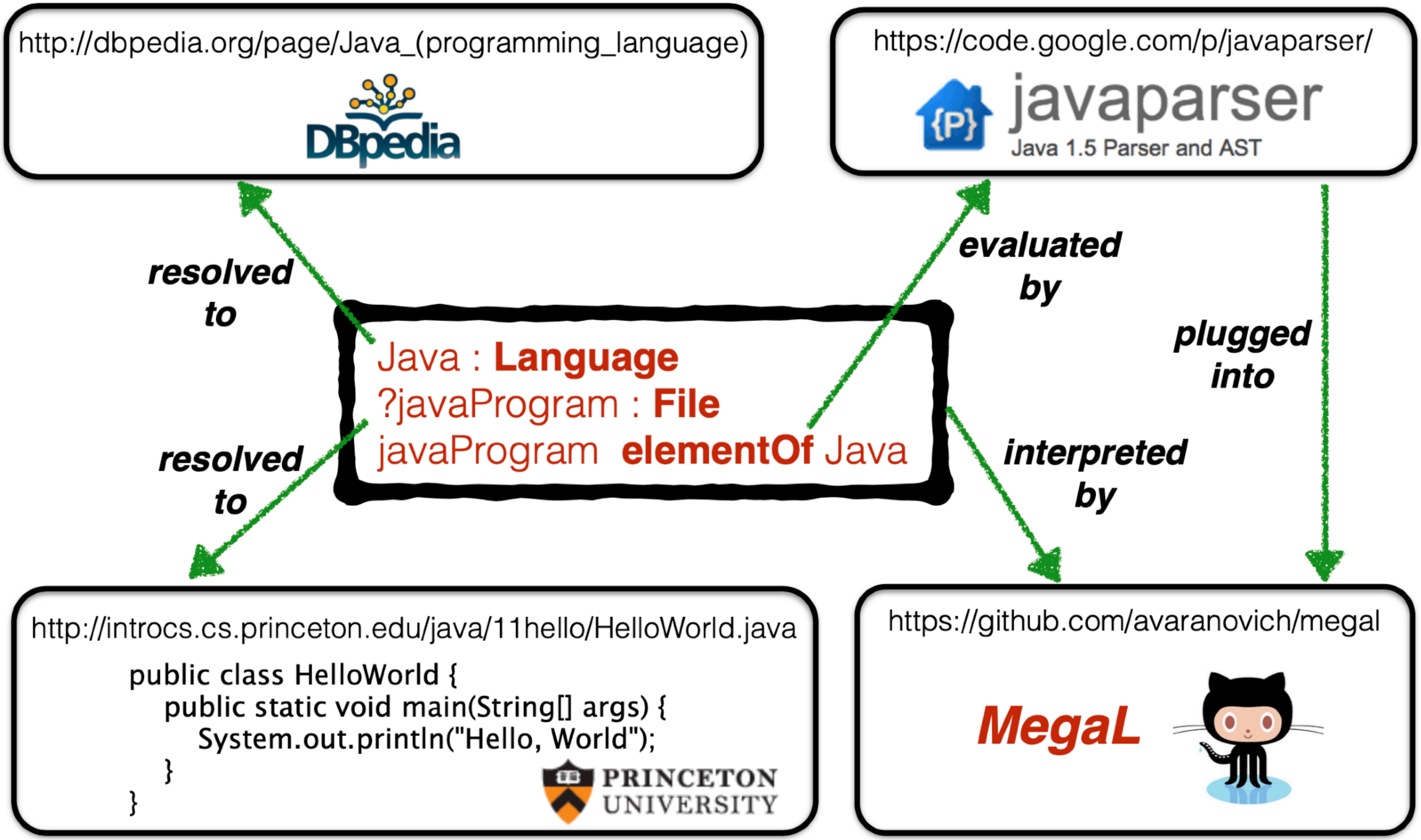
Source: Johannes Härtel, Lukas Härtel, Ralf Lämmel, Andrei Varanovich, Marcel Heinz: **Interconnected Linguistic Architecture**. Art Sci. Eng. Program. 1(1): 3 (2017)

Validation of models of linguistic architecture



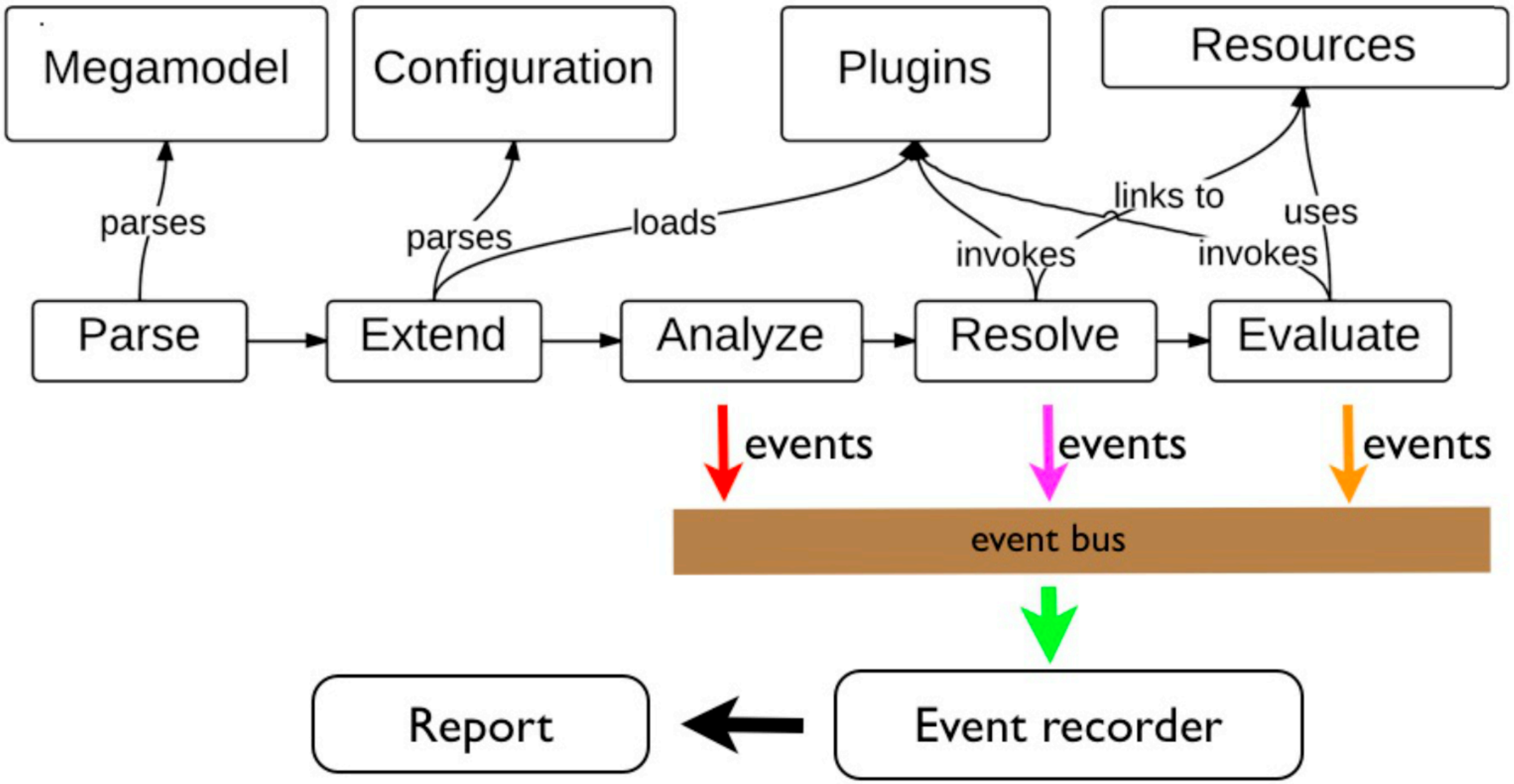
Source: Johannes Härtel, Lukas Härtel, Ralf Lämmel, Andrei Varanovich, Marcel Heinz: **Interconnected Linguistic Architecture**. Art Sci. Eng. Program. 1(1): 3 (2017)

Interpretation of models of linguistic architecture



Source: Ralf Lämmel, Andrei Varanovich: **Interpretation of Linguistic Architecture**. ECMFA 2014: 67-82

Processing models of linguistic architecture



Source: Ralf Lämmel, Andrei Varanovich: **Interpretation of Linguistic Architecture**. ECMFA 2014: 67-82

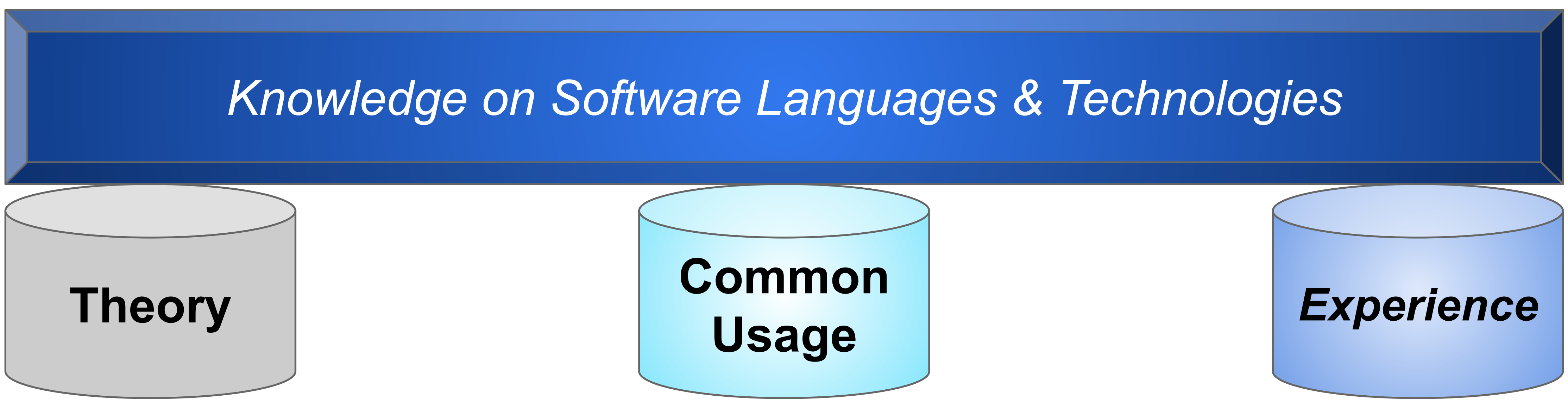
Knowledge Engineering for Software Languages and Software Technologies

Marcel Heinz

Universität Koblenz-Landau
Fachbereich 4 - Informatik
Softlang Team

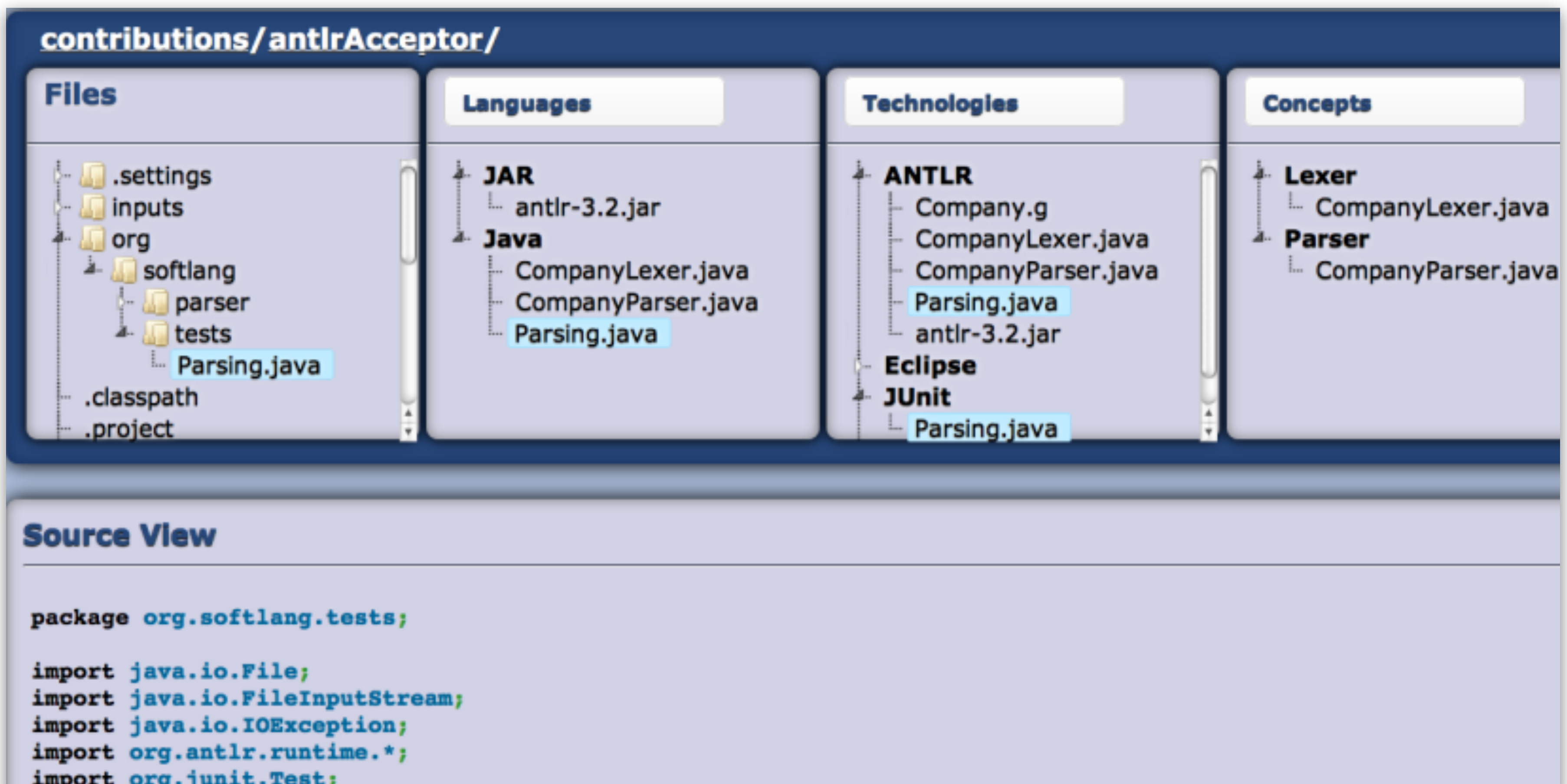
This part of the lecture is shamelessly based on Marcel Heinz' slides from his PhD defense. Thanks!

Motivation



Discovering and structuring knowledge resources
while assuring quality.

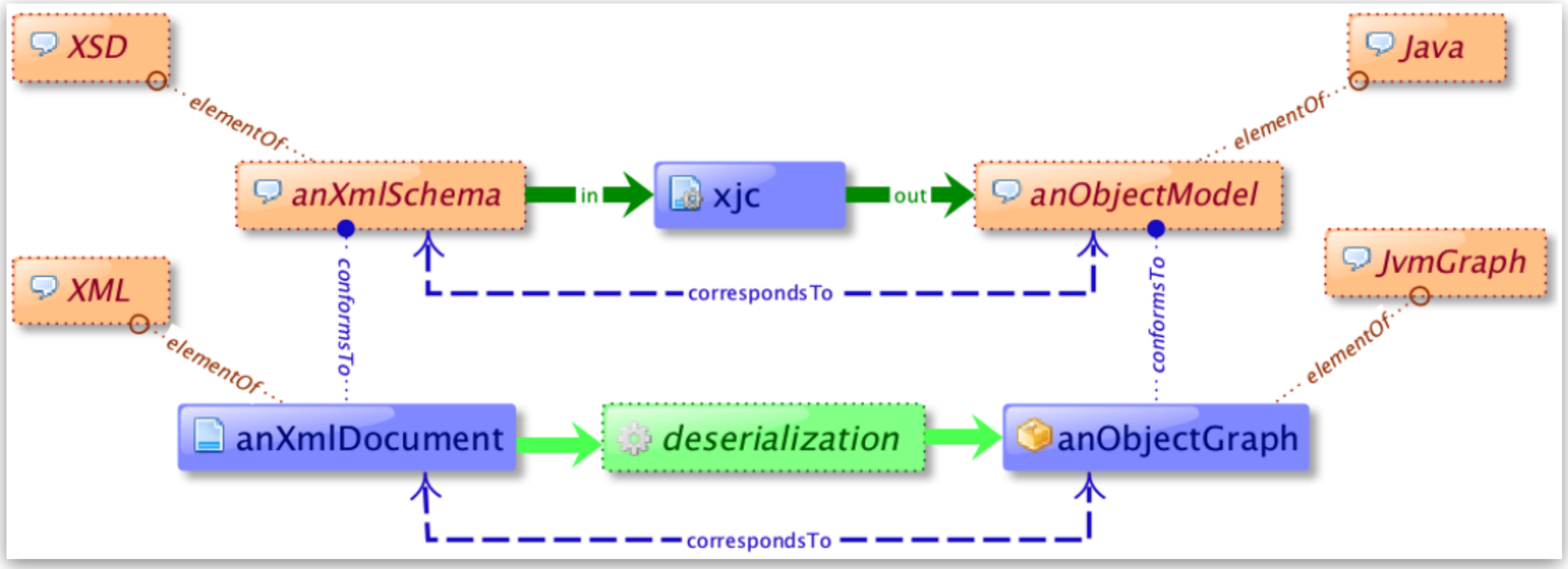
Background - Software Chrestomathy



Favre, J. M., Lämmel, R., Leinberger, M., Schmorleiz, T., & Varanovich, A. (2012, October). Linking documentation and source code in a software chrestomathy. In *2012 19th Working Conference on Reverse Engineering* (pp. 335-344). IEEE.

Background - Megamodels

Jean-Marie Favre, Ralf Lämmel, Andrei Varanovich: Modeling the Linguistic Architecture of Software Products. MoDELS 2012: 151-167



Research Contributions

(*) The overall methodology and framework was developed by Johannes Härtel. The case study is the actual contribution by this thesis.

Axioms of Linguistic Architecture

Systematic Mapping Study on Megamodeling Vocabulary.
The Core Ontology *SoLaSoTe.*
→ Case Study on EMF

Patterns of Usage on GitHub

Methodology & Framework to Mine Patterns of Technology Usage on GitHub.
→ Case Study on EMF (*)

Discovering Indicators for Classifying Wikipedia Articles in a Domain

Methodology to Discover Wikipedia Articles Relevant to a Single Domain Class.
→ Case Study on Software Languages

Reproducible Construction of Interconnected Technology Models

Methodology to Construct Reproducible Interconnected Technology Models.
→ Case Studies on EMF

Research Publications

Heinz, M., Lämmel, R., & Varanovich, A. *Axioms of Linguistic Architecture.* MODELSWARD 2017

Härtel, J., Heinz, M., & Lämmel, R. *EMF patterns of usage on GitHub*. ECMFA 2018

Heinz, M., Lämmel, R., & Acher, M. *Discovering Indicators for Classifying Wikipedia Articles in a Domain: A Case Study on Software Languages.* SEKE 2019

Heinz, M., Härtel, J., & Lämmel, R. *Reproducible Construction of Interconnected Technology Models for EMF Code Generation.* ECMFA 2020

Research Questions

Axioms of Linguistic Architecture

- What types of entities and relations are common in megamodeling literature?
- What modeling idioms exist for (language-centric) megamodels that can be formalized as axioms?

Systematic Mapping Study

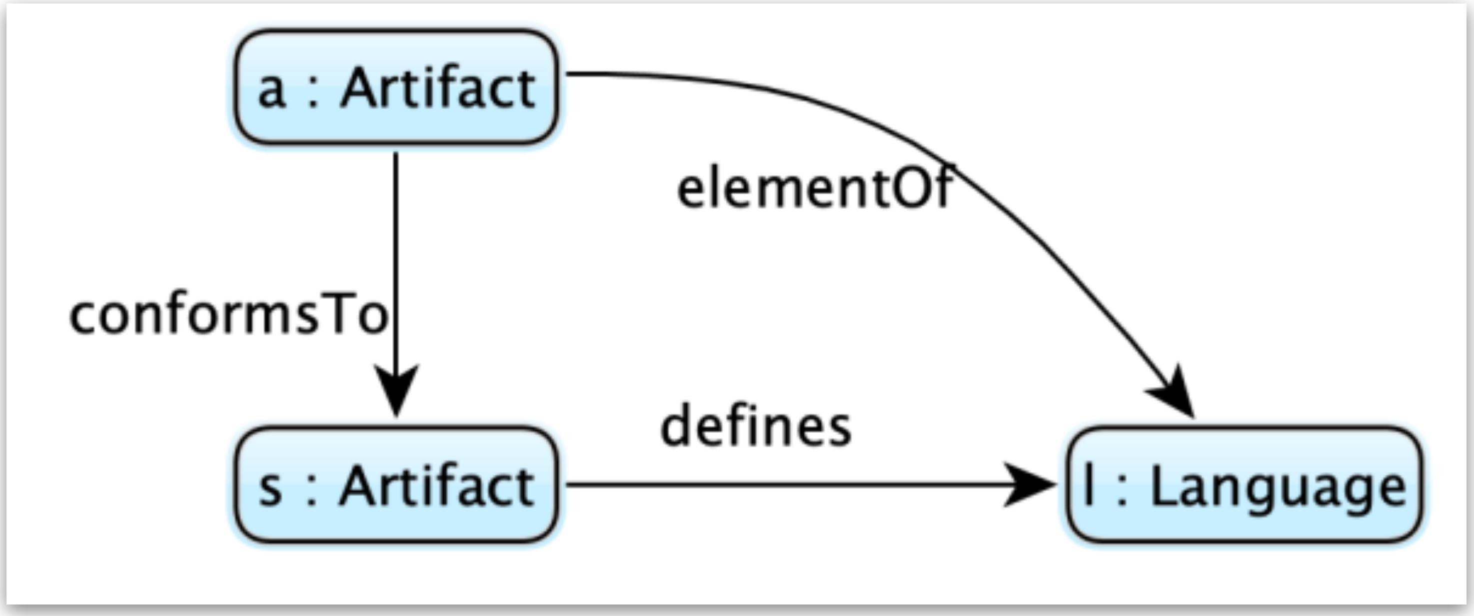
Paper	Keywords
[Favre et al., 2012c]	Artifact, File, Program, ObjectGraph, Resource
[Vogel and Giese, 2012]	Megamodel, Model
[Simmonds et al., 2015]	Terminal Model, metamodel, metametamodel, megamodel
[Favre, 2004]	Metamodel, Model
[Favre and Martinez, 2006]	PIM Metamodel, PSM Metamodel, ISM Metamodel
[Seibel et al., 2010]	Model, DynamicHierarchicalMegaModel
[Salay et al., 2015]	Model, Transable, MegaTransable, Megamodel
[Jouault et al., 2010]	WeavingModel,TraceModel,TerminalModel
[Barbero et al., 2008]	Megamodel, Model
[Kling et al., 2011]	Model, ReferenceModel, TerminalModel, MetaMetaModel, Meta-Model, MegaModel
[Méndez-Acuña et al., 2013]	MetaMegaModel, MetaModel, Model
[Vignaga et al., 2013]	Model, TextualEntity, TerminalModel, MegaModel, Reference-Model, MetametaModel, MetaModel
[Stevens, 2018]	Meta-Model, Model, Safety, Code, Tests
[Lämmel, 2016]	Artifact
[Sottet et al., 2018]	Model, Metamodel, Megamodel
[Toure et al., 2017]	Model, Metamodel, Megamodel

Table 3.2 – Vocabulary aligned with the type *Artifact*.

Competency Questions

Competency Questions as a Methodological Tool for Designing Ontologies.

- Which artifacts are elements of which software language?
- Which schema artifact can be used to validate an instance artifact?
- Which artifacts implement/define which language?



Axiomatization

State competency questions

- *What artifacts exist in the software?*
- *How do artifacts manifest?*

Develop axioms

Axioms	Prolog

$Artifact(e) \Rightarrow Entity(e).$
$Folder(a) \Rightarrow Artifact(a).$
$File(a) \Rightarrow Artifact(a).$
$Fragment(a) \Rightarrow Artifact(a).$
$Transient(a) \Rightarrow Artifact(a).$
	`entity(X) :- artifact(X).`
	`artifact(X) :- folder(X).`
	`artifact(X) :- file(X).`
	`artifact(X) :- fragment(X).`
	`artifact(X) :- transient(X).`

Validate based on EMF Case Study.

EMF

```
folder("org.eclipse.emf.ecore"). % metametamodel is a folder.
folder("com.example.po"). % the Java object model package is a folder.
file("org.eclipse.emf.ecore.EObject.java"). % the class EObject is a file.
file("org.eclipse.emf.ecore.EPackage.java"). % the class EPackage is a file.
file("SimplePO.ecore"). % the Ecore model for purchase orders is a file.
file("SimplePO.genmodel"). % the generator model is a file.
transient(christmas_order_object). % the purchase order object is a transient.
file("christmas_simplepo.xmi"). % the persisted purchase order object is a file.
fragment("SimplePO.ecore/PurchaseOrder"). % the PurchaseOrder EClass is a fragment.
```

Axiomatization

Formalize axiom.

Validate based on EMF Case Study.

Axioms	Prolog
$conformsTo(a, a') \Rightarrow Artifact(a) \wedge Artifact(a')$ $conformsTo(a, a') \Rightarrow (\exists l.defines(a', l) \wedge elementOf(a, l)) \vee (\forall p.partOf(p, a) \exists p'.partOf(p', a') \wedge conformsTo(p, p'))$.	`ok_relation(conforms_to(A1,A2)):- artifact(A1),artifact(A2),(( defines(A2,L),element_ofT(A1,L)); forall(part_of(P1,A1),( part_of(P2,A2), conforms_to(P1,P2)))).`

EMF

```
%Traceable Conformance
conforms_to("christmas_simplepo.xmi","SimplePO.ecore").
part_of("christmas_simplepo.xmi/PurchaseOrder","christmas_simplepo.xmi").
%part_of("SimplePO.ecore/PurchaseOrder","SimplePO.ecore"). is stated earlier.
conforms_to("christmas_simplepo.xmi/PurchaseOrder","SimplePO.ecore/PurchaseOrder").
element_of("christmas_simplepo.xmi/PurchaseOrder",purchase_order_xmi).
defines("SimplePO.ecore/PurchaseOrder",purchase_order_xmi).
```

Research Question

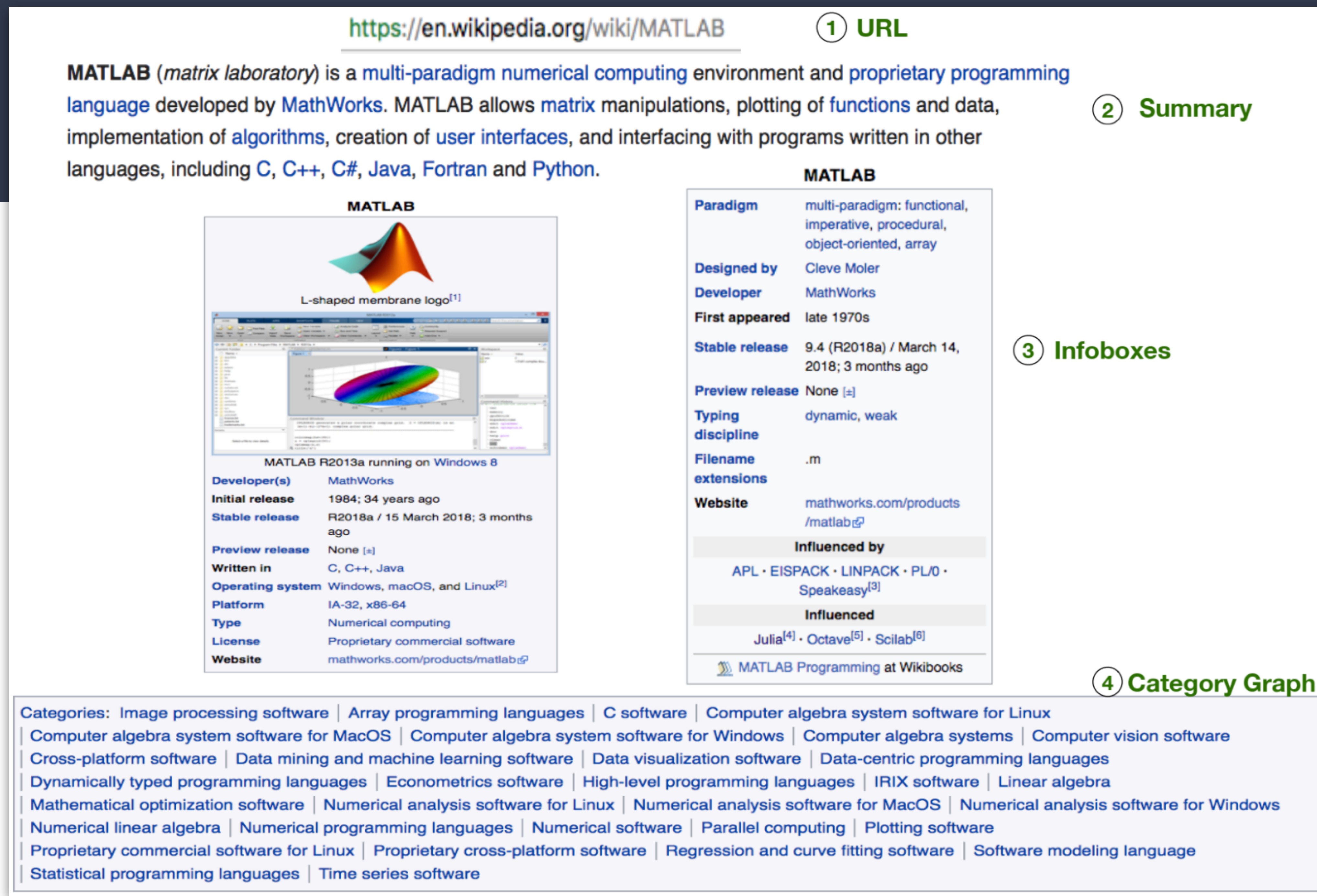
Discovering Indicators for Classifying Wikipedia Articles in a Domain

- *How can we classify* Wikipedia *articles by their relevance to a given domain when relevant articles are rare and multiple main topics are covered by articles?*

Motivation

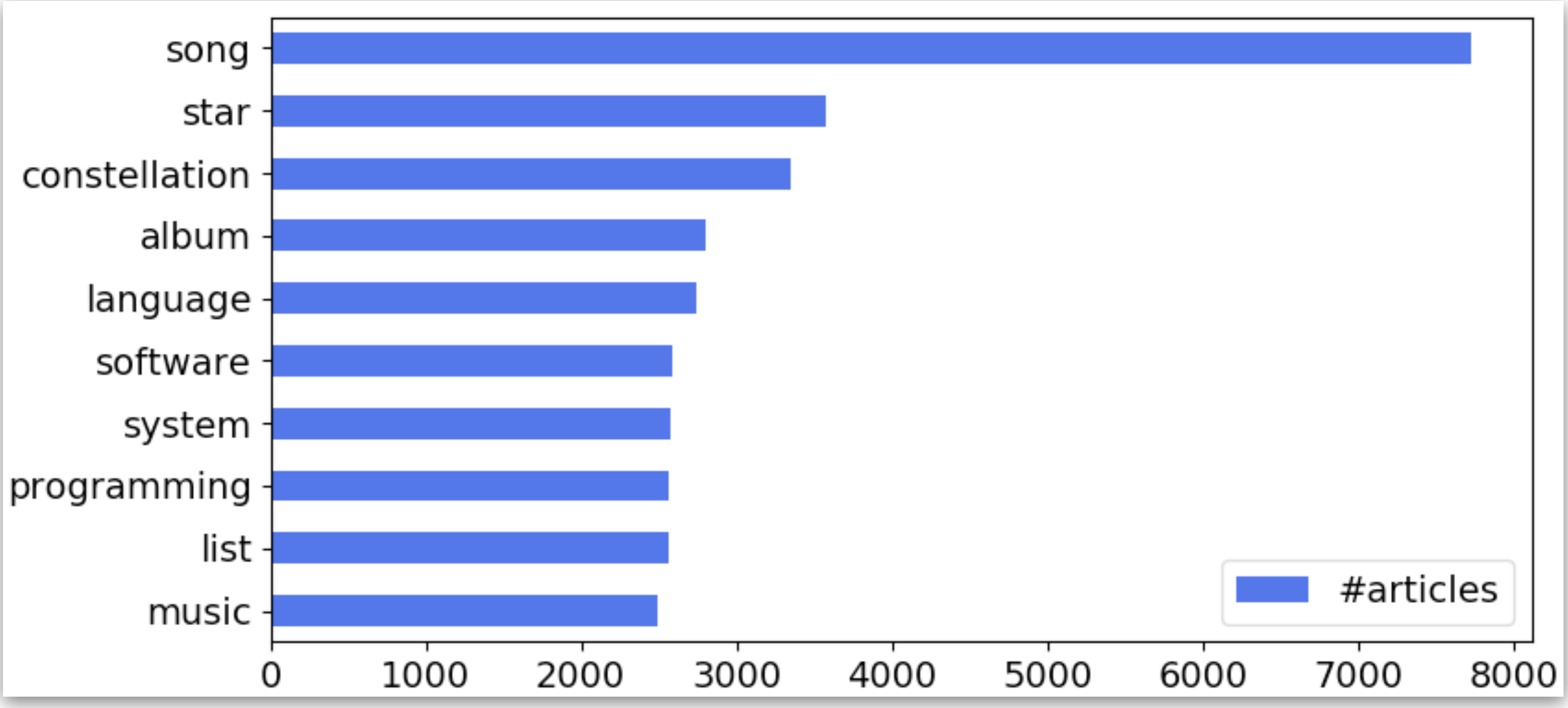
https://en.wikipedia.org/wiki/Wikipedia:Notability

*“When creating new content about a notable topic, editors should consider **how best to help readers understand it**.”*



Data Exploration

Frequent nouns in articles below the category ‘Computer languages’ with a maximum depth of seven.



Expert Survey to Reduce Subjectivity

Does the article Augmented_Backus–Naur_Form describe a software language?

We define a software language as a set of digital artifacts, for which syntax, type system, semantics, and pragmatics can be (in)formally defined, documented, and implemented. Thus, language categories such as programming languages or file formats are included.

Click on the link and skim through the article. You have to decide.

Hints

- Looking at the title alone can be misleading.
- Remember that one article can describe many topics.
- A casual mention of a software language is not enough.
- The first paragraph should clarify that a software language is a major topic.

Not logged in Talk Contributions Create account Log in

Wikipedia
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia

Article Talk Read Edit View history Search Wikipedia

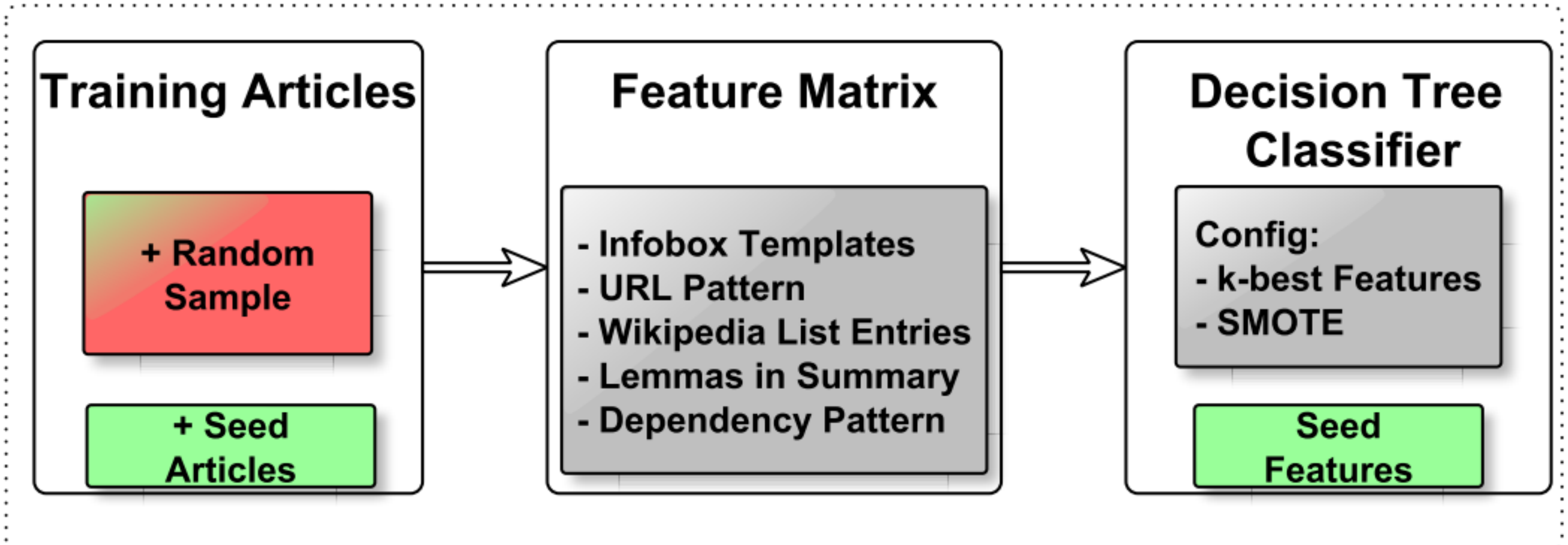
Augmented Backus–Naur form

From Wikipedia, the free encyclopedia
(Redirected from Augmented Backus–Naur Form)

In computer science, **augmented Backus–Naur form** (**ABNF**) is a metalanguage based on Backus–Naur form (BNF), but consisting of its own syntax and derivation rules. The motive principle for ABNF is to describe a formal system of a language to be used as a bidirectional communications protocol. It is defined by *Internet Standard 68* ("STD 68", type case sic), which as of December 2010 is RFC 5234, and it often

- ◯ Yes
- ◯ No

Methodology & Result



301 seed articles based on GitHub and TIOBE Index
→ 2797 articles on software languages.

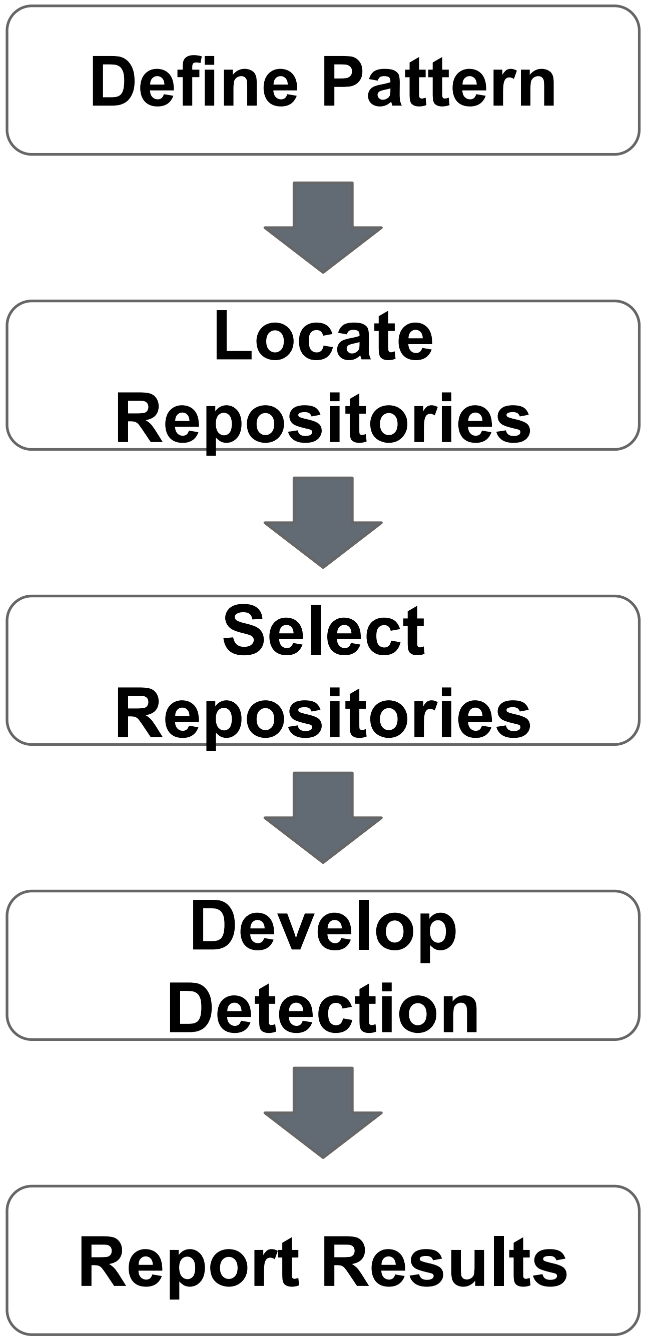
With k=23, the learned classifier performs with an f1-score of 0.7, balanced accuracy of 0.9, recall of 0.81 and specificity of 0.99.

Research Question

Patterns of Usage on GitHub

- *How can we locate traces of technology usage on GitHub?*

Methodology Overview



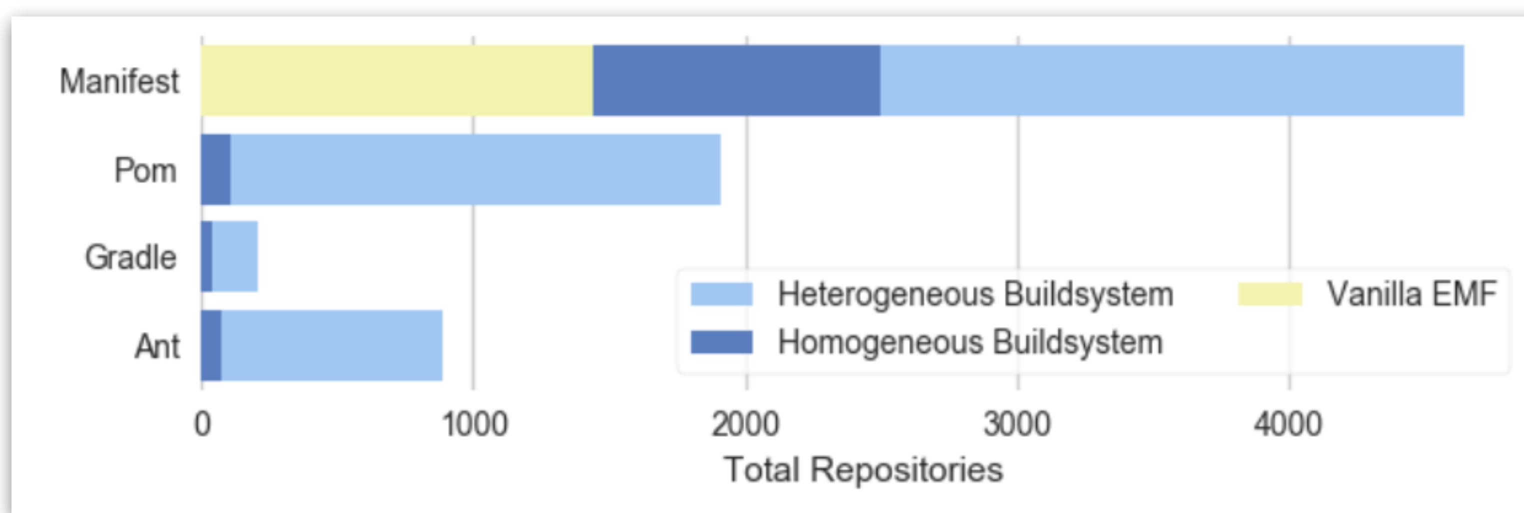
Case Study on EMF

Locating Repositories

Evidence	Query	Extension
Java Model	"extends EObject {"	java
Ecore Model	GenModel	ecore
Generator Model	EClass	genmodel

Table 5.3 – Queries for locating repositories through GitHub API.

Selecting Repositories

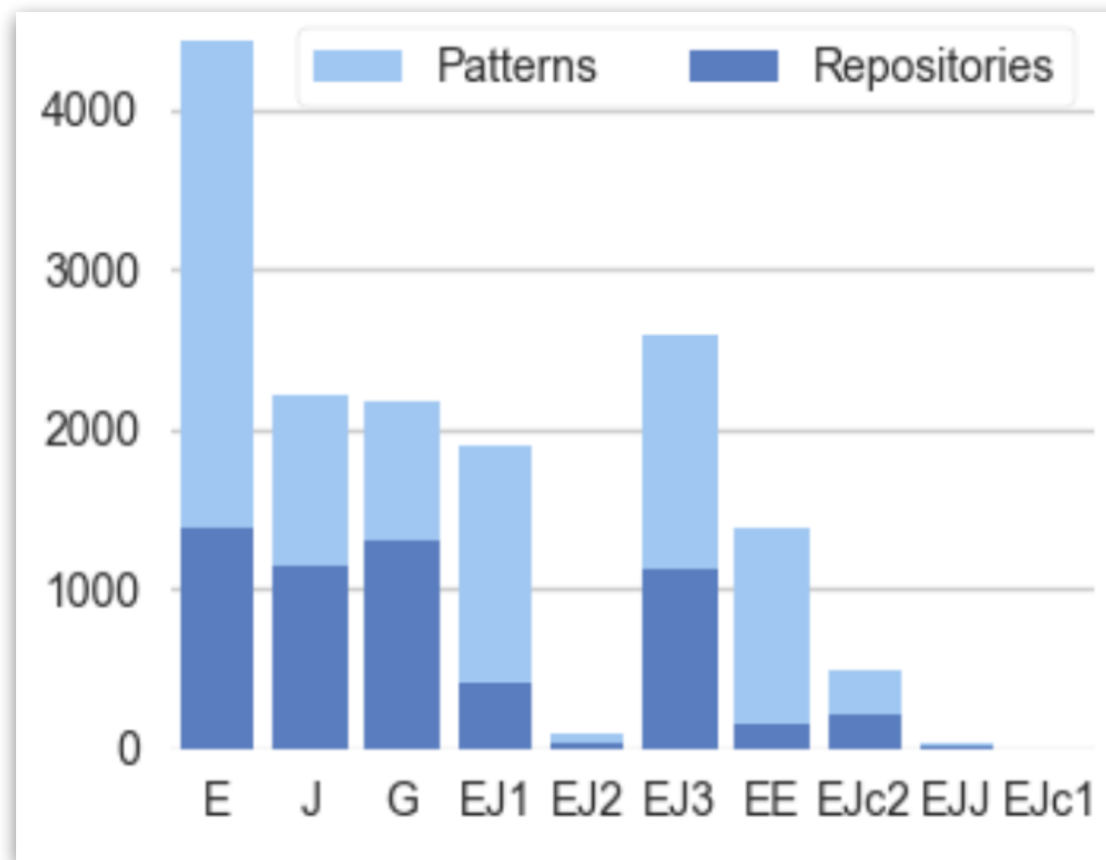


Case Study on EMF

Develop Detection

```
(?x sl:manifestsAs sl:File)
(?x sl:elementOf sl:XML)
Extension(?x,"ecore") →
 (?x sl:elementOf sl:Ecore).
```

Report Results

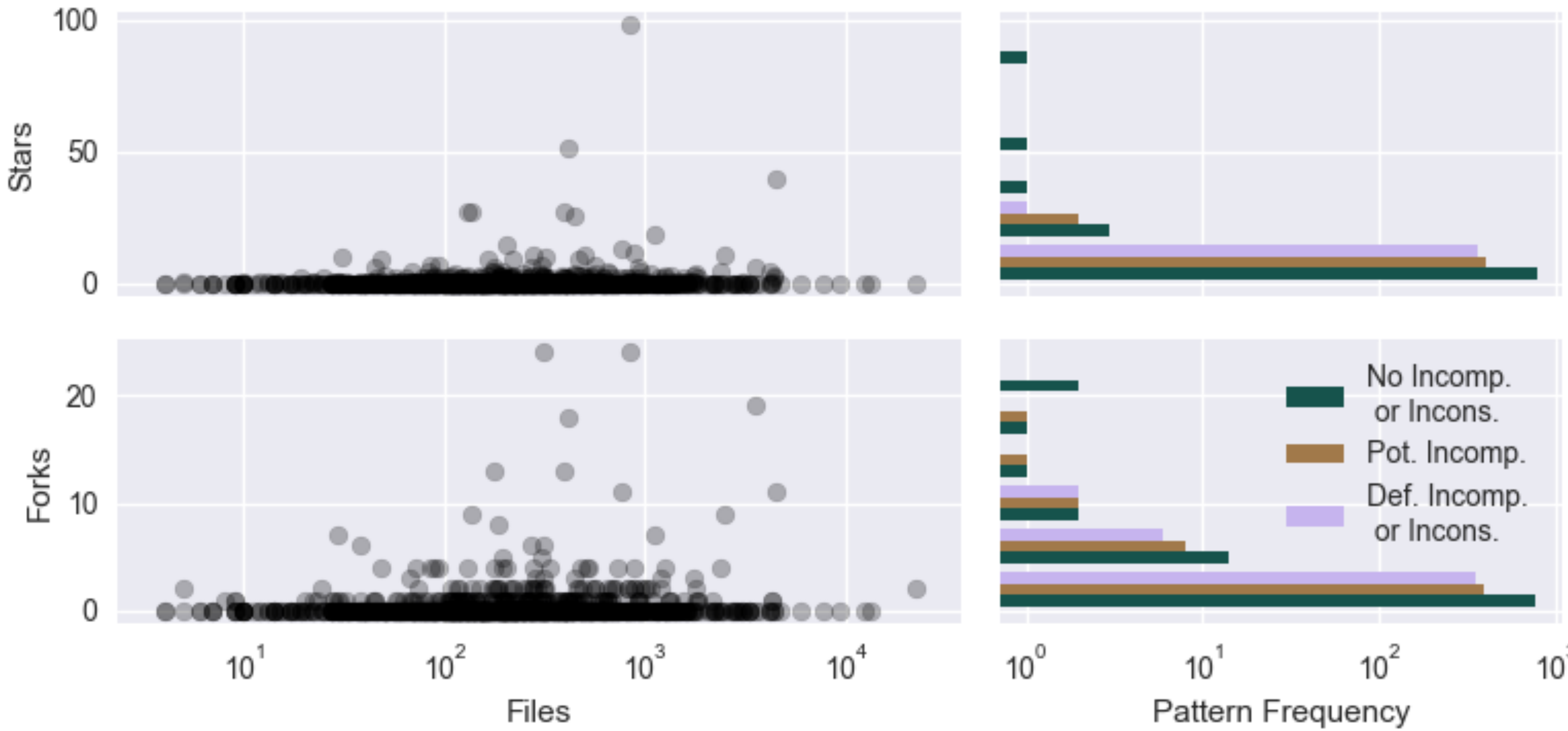


Defined Pattern

Id	Cls.	Artifacts	Description and cause
Single artifact patterns			
E	Pres.	• Ecore Pkg.	The presence of an Ecore Pkg. in '.ecore' files as root or subpackage.
J	Pres.	• Java Pkg.	The presence of a Java Pkg.
G	Pres.	• Genmodel Pkg.	The presence of a Genmodel Pkg. in '.genmodel' files as root or subpackage.
C	Pres.	• Customized Java Pkg.	The presence of a Java Pkg. with customized interface or implementation.
Double artifact patterns			
EJ1	Pot. In-comp.	• Ecore Pkg. • Java Pkg. (m[a]) [a] Missing	A Java Pkg. cannot be found for a given nsURI as extracted from some Ecore Pkg. This is only a potential incompleteness, because a Java Pkg. could be potentially derived, if no customization is intended.

Mining GitHub

Repository quality



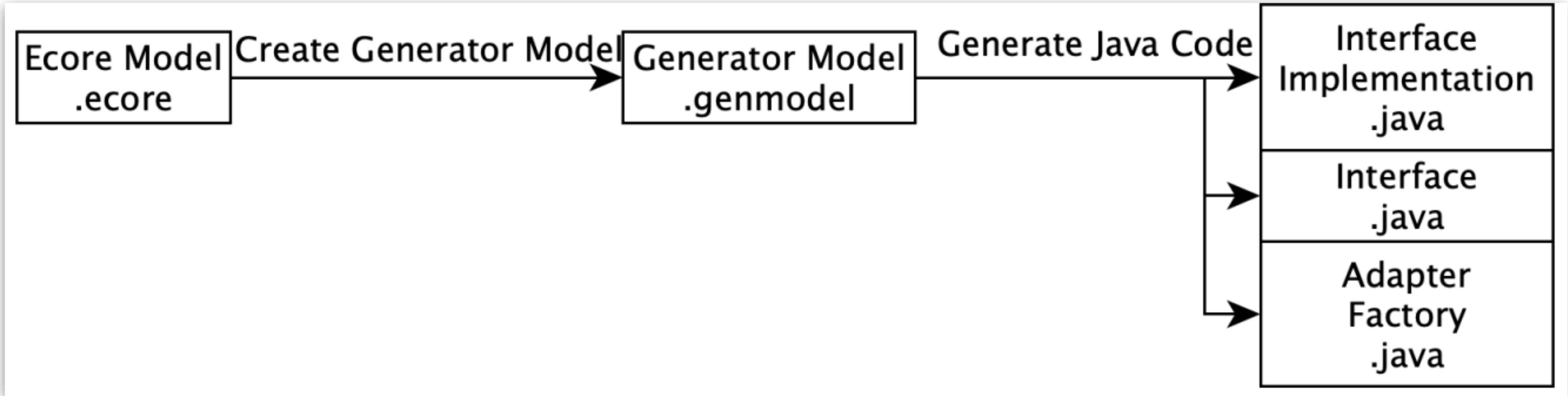
Research Question

Reproducible Construction of Interconnected Technology Models

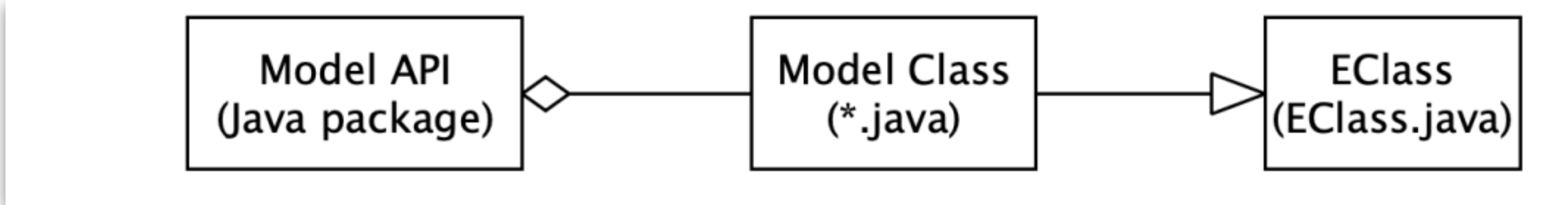
- *How can we construct a technology model in a reproducible manner so that it is interconnected with existing textual explanations and code examples?*

Motivation

Common Usage



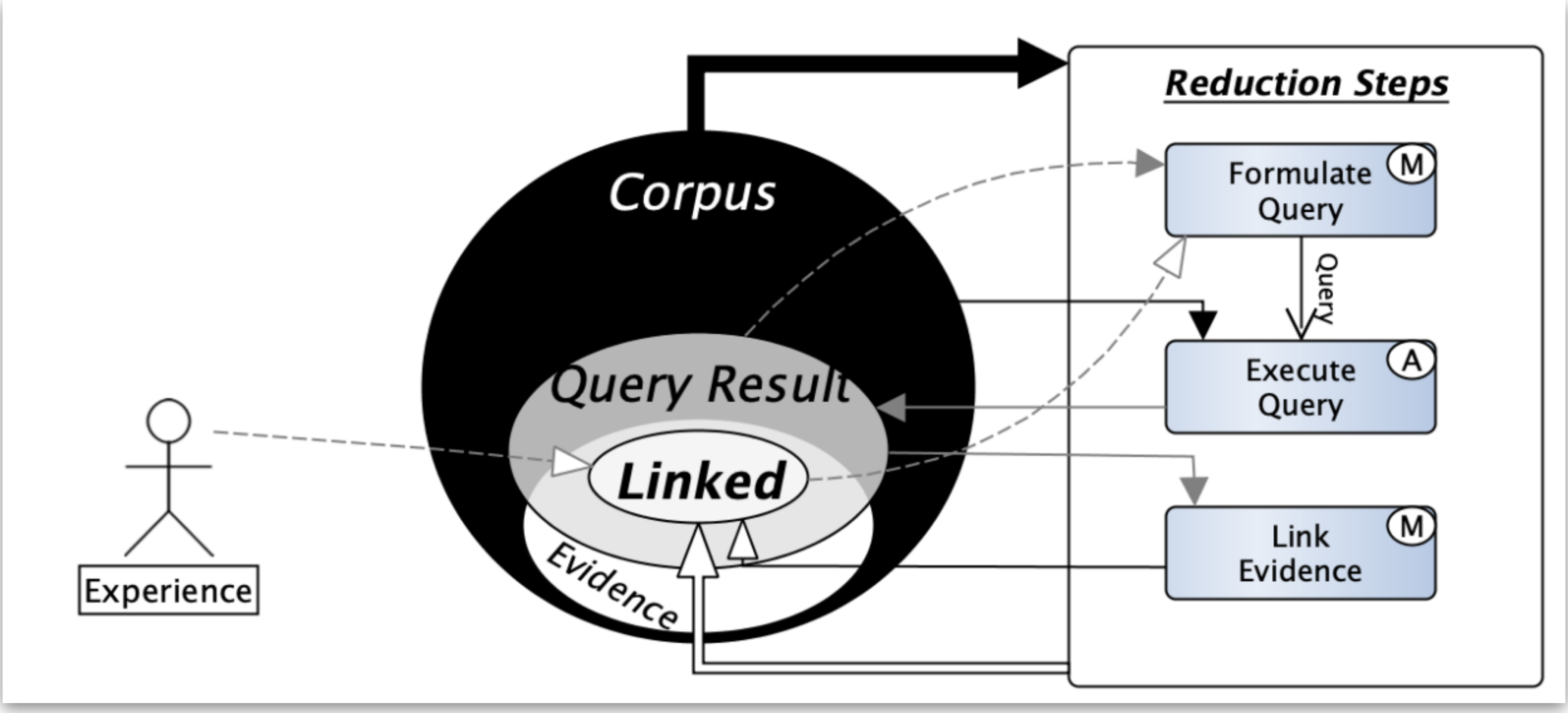
Misconception



Methodology for Reducing a Corpus to Linked Evidence

A corpus can be:

- *Developer literature*.
- *Scientific literature*.
- *Demo Projects*.
- *Wild Projects*.



Reproducibility of Technology Models
— Process textbook —

ID	***Step***	***In***	***Out***	***Automation***
1	Formulate Query	*Experience*: Name 'Ecore model'	*Query*: 'Ecore'	M
2	Execute Query	*Corpus Resource*: Table of Content *Query*: See **1**	*Query Results*: Subsection 2.3.1, Subsection 2.3.5 Subsection 4.2.4 ...	A
3	Link Evidence	*Query results:* See **2**	*Linked*: Subsection 2.3.1	M
...	...	...	...	...

Reproducibility of Technology Models
— Process sample code —

ID	*Step*	*In*	*Out*	*Automation*
1	Formulate Query	*Experience*: search for file endings .java, .genmodel, .ecore	*Query*: see Listing 6.1 + '.java' query	M
2	Execute Query	*Corpus Resource*: Project PrimerPO *Query*: See **1**	*Query Results*: PrimerPO.ecore PrimerPO.genmodel Item.java .. PPOPackage.java ...	A
3	Link Evidence	*Query results:* See **2**	*Linked*: see Table 6.6	M
...	...	...	...	...

Table 6.5 – Excerpt of the reduction step protocol for the demo project PrimerPO. '.java' files returned by the query are manually filtered. For instance, *PPOPackage* does not exemplify any modeled type.

Links from model elements to code

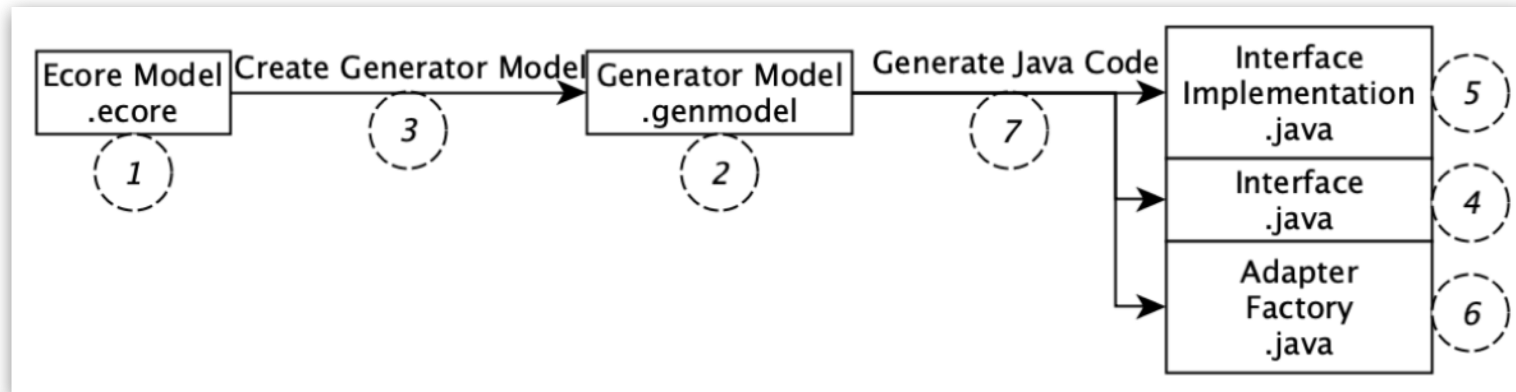
Type/**R**elation	*Links*	*Rationale*
T:EM	<:/model/PrimerPO.ecore>	Ecore Model Query (see Listing 6.1)
T:GM	<:/model/PrimerPO.genmodel>	Generator Model Query (see Listing 6.1)
R:EtoG	(<:/model/PrimerPO.ecore>, <:/model/PrimerPO.genmodel>)	Foreign Model Query (see Listing 2)
T:Int	<:/src/ppo/Item.java>,..	Extends Queries (see Figure 6.5)
T:Impl	<:/src/ppo/impl/ItemImpl.java>,..	Implements Queries (see Figure 6.5)
T:AF	<:/src/ppo/util/PpoAdapterFactory.java>	Package Reference Query (see Figure 6.5)
R:GtoJ	(<:/model/PrimerPO.genmodel>, <:/src/ppo/Item.java>), .. , (<:/model/PrimerPO.genmodel>, <:/src/ppo/impl/ItemImpl.java>), .. , (<:/model/PrimerPO.genmodel>, <:/src/ppo/util/PpoAdapterFactory.java>)	Reference Queries (see Figure 6.5)

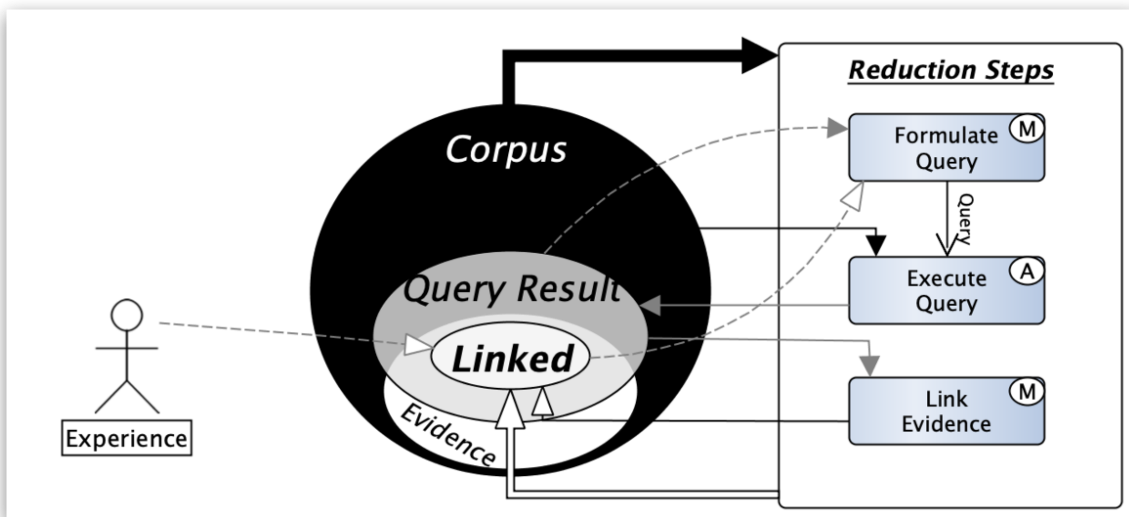
Reproducibility of Technology Models
— Process paper collection —

Type /**R**elation	*Links* [1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	*Rationale*
T:EM	1	1	1	3	2	2	2	1	1	2	Ecore Model Query (see Listing 6.1)
T:GM	1	2	1	5	2	2	2	1	1	2	Generator Model Query (see Listing 6.1)
R:EtoG	1	1		5	2	1	2	1	1	2	Foreign Model Query (see Listing 2)
T:Int	111	2	2	12				46	6	5	Extend EObject Queries (see Figure 6.5)
T:Impl	82	2	2	15				46	6	4	Implements Queries (see Figure 6.5)
T:AF	3	1	1	4				1	1	2	Package Reference Query (see Figure 6.5)
R:GtoJ	167	5		38				93	13	4	Reference Query (see Figure 6.5)

Case Study Results

Common Usage

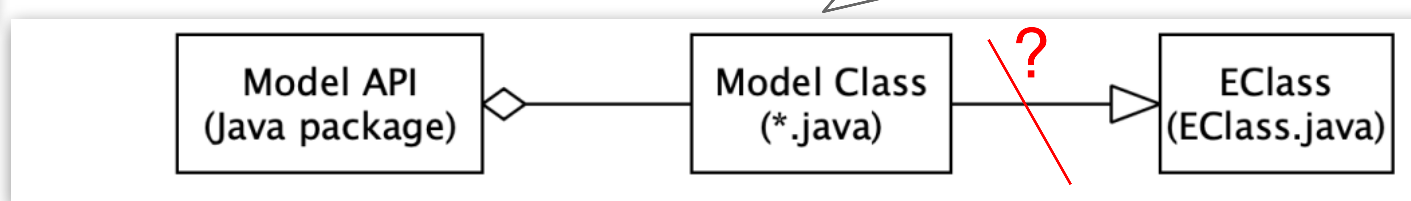




Many interconnected textual explanations and code examples.

Rare interconnected textual explanations and code examples.

Misconception



Threats to Validity

- Our contributions may be biased by our modeling experience.
- Our case studies only focus on EMF in depth.
- The selection of resources (e.g., on GitHub) may not be representative.

Conclusion

- Discovering and structuring knowledge based on **literature studies, Wikipedia mining,** and **GitHub mining** while assuring quality.
- Coverage on different technologies is needed for further investigations.
- **Prototyping** and **internal validation** with other domain experts have been conducted.
- **External validation** in terms of quantitative research is needed to discuss quality dimensions such as usefulness of technology models to professional software engineers.

Megamodeling

Coupled Software Transformations

Ralf Lämmel

A long time ago (at an unknown workshop (SET'04)) …

Coupled Software Transformations

— Extended Abstract —

Ralf Lämmel

VUA & CWI, Amsterdam, The Netherlands

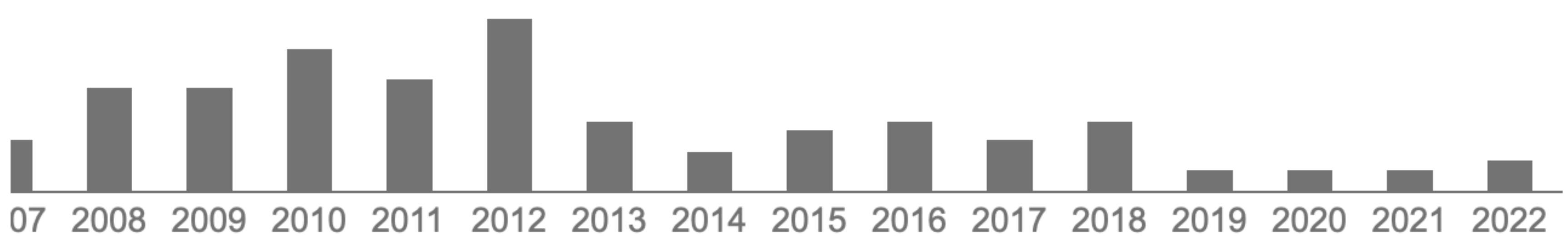
Problems with the past:

- CX (or BX) has developed ever since.
- We don't like figures without meaning anymore.
- Things shall be illustrated, validated, reproducible.

Find the bug in this Google Scholar page

Coupled software transformations revisited

Authors	Ralf Lämmel
Publication date	2016/10/20
Book	Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering
Pages	239-252
Description	We revisit the notion of coupled software transformations (CX) which is concerned with keeping collections of software artifacts consistent in response to changes of individual artifacts. We model scenarios of CX while we abstract from technological spaces and application domains. Our objective is to mediate between universal consistency properties of CX and test-driven validation of concrete (illustrative) CX implementations. To this end, we leverage an emerging megamodeling language LAL which is based on many-and order-sorted predicate logic with support for reuse by inlining modulo substitution. We provide a simple translation semantics for LAL so that formulae can be rendered as test cases on appropriate interpretations of the megamodel elements. Our approach has been implemented and validated in logic programming; this includes the executable language definition of LAL and test-case execution …
Total citations	Cited by 131



Today (SLE 2016)

Coupled Software Transformations—Revisited

Ralf Lämmel

Software Languages Team, http://www.softlang.org/
University of Koblenz-Landau
Germany

“Everything” is linked to the repo!

…

3.1 Languages

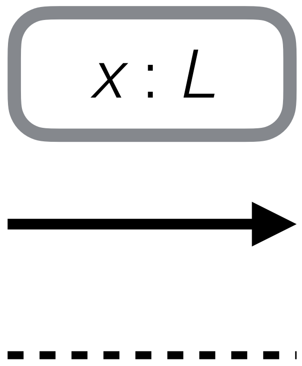
Let us express that a language L is a subset of a suitable universe Any (such as ‘all’ strings, trees, or gr

LAL megamodel *language*

```
sort Any // The universe to draw elements from
sort L ⊆ Any // A language as a subset of the universe
```

…

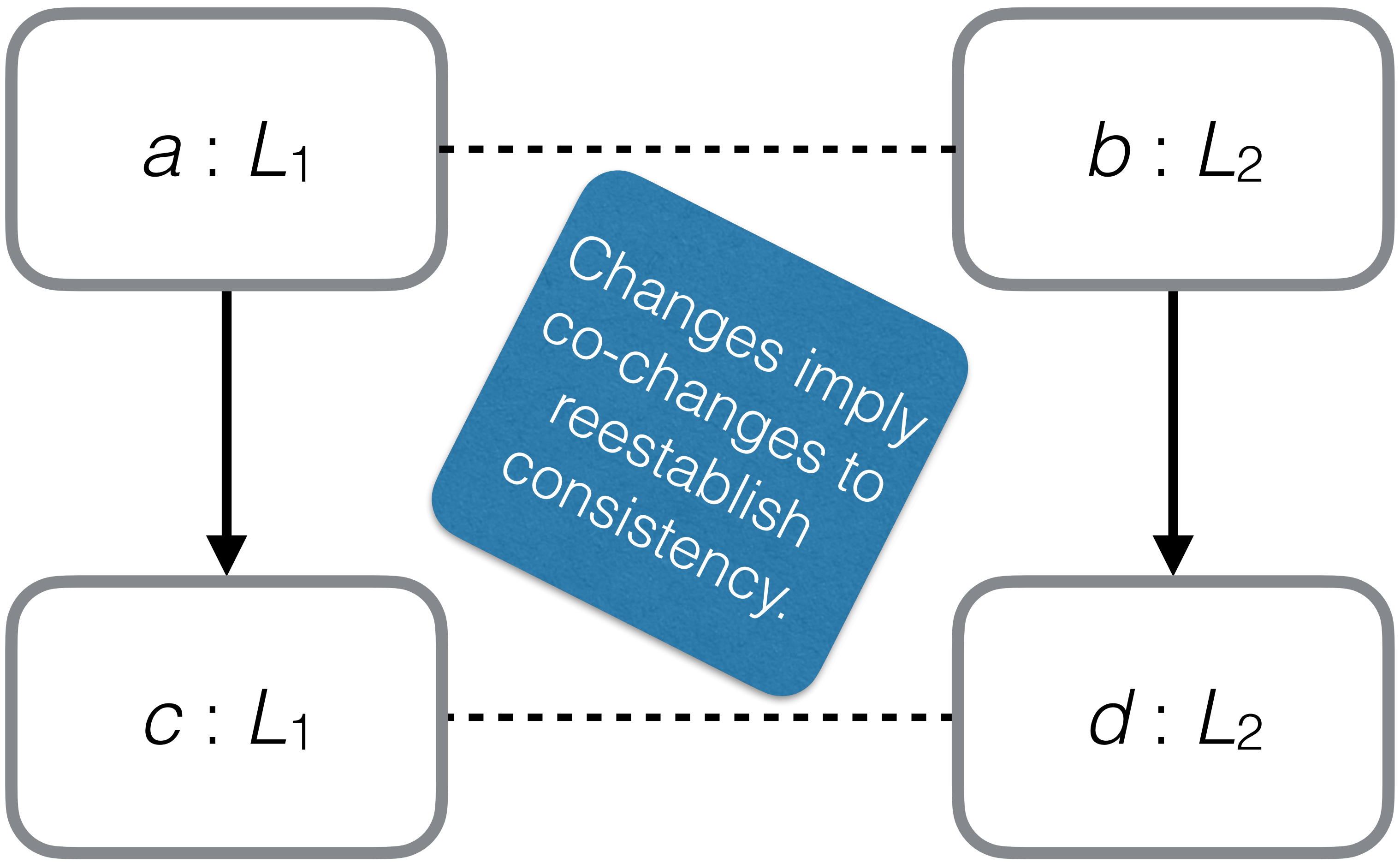
What’s a *coupled transformation* (CX)?



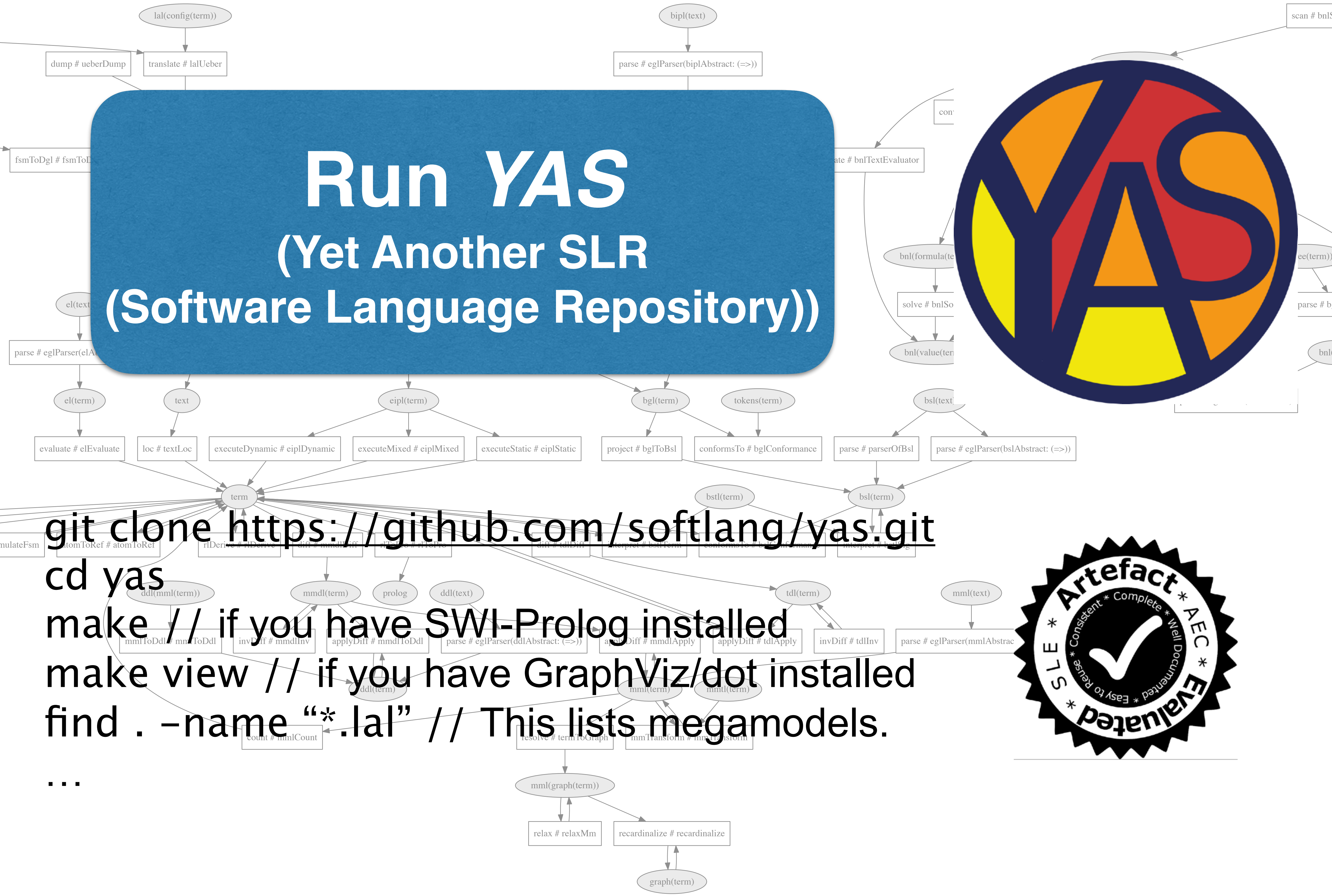
Artifacts ‘typed’ by languages

Transformation, often in the sense of **evolution**

Consistency, e.g., conformance



- **<u>What</u> are we doing?**
  - Model ‘patterns’ of CX.
  - Capture properties of transformations.
  - Instantiate ‘patterns’ as test cases.
- **<u>Why</u> are we doing it?**
  - Provide a CX chrestomathy (‘useful for learning …’).
  - Introduce a logic-based form of testable megamodels.
- **<u>How</u> are we doing it?**
  - Design a domain-specific predicate logic.
  - Design and implement a logic-based test framework.
  - Implement CX examples in Prolog (so it happens).



Run *YAS*

(Yet Another SLR (Software Language Repository))

git clone https://github.com/softlang/yas.git
cd yas
make // if you have SWI-Prolog installed
make view // if you have GraphViz/dot installed
find . -name "*.lal" // This lists megamodels.
...

How do the *megamodels* look like?

```
sort Any // The universe to draw elements from
sort L ⊆ Any // A language as a subset of the universe
```

LAL megamodel language

```
reuse language [ L ↦ MathML, Any ↦ XML ]
link MathML to 'https://www.w3.org/TR/MathML3'
link XML to 'https://www.w3.org/XML'
```

LAL megamodel language.mathml

```
reuse language // The defined language
reuse language [ L ↦ DefL, Any ↦ DefAny ]
constant defL : DefL // The language definition
relation conformsTo : Any × DefL
axiom { ∀x ∈ Any. x ∈ L ⇔ conformsTo(x, defL) }
```

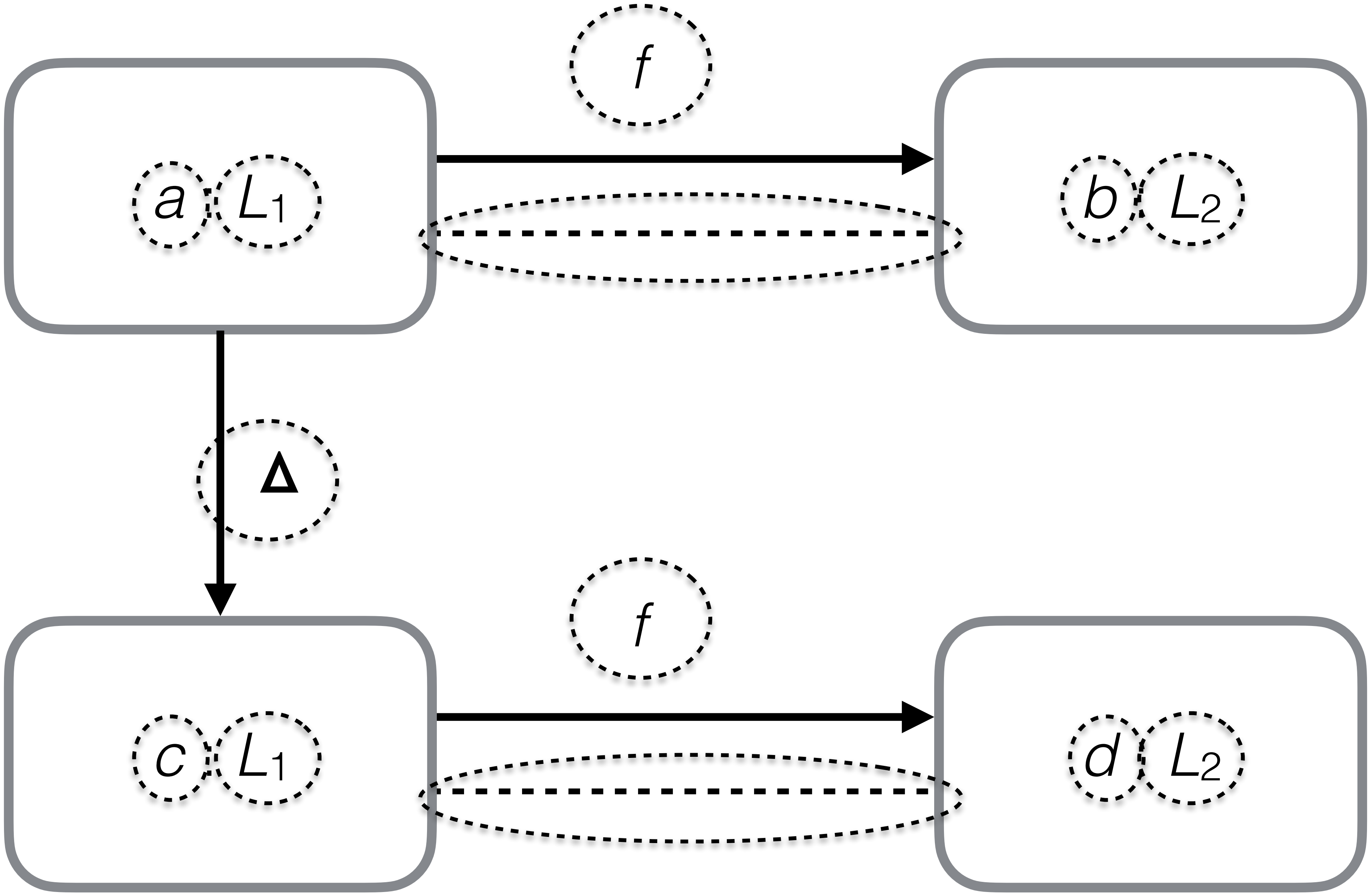
LAL megamodel conformance

```
reuse conformance [
    Any ↦ XML, DefAny ↦ XML,
    L ↦ MathML, DefL ↦ XSD, defL ↦ MathMLSchema ]
link XML to 'https://www.w3.org/XML'
link XSD to 'https://www.w3.org/XML/Schema'
link MathML to 'https://www.w3.org/TR/MathML3'
link MathMLSchema to 'https://www.w3.org/Math/XMLSchema'
```

LAL megamodel conformance.mathml

The ‘pattern’ of CX by *mapping*

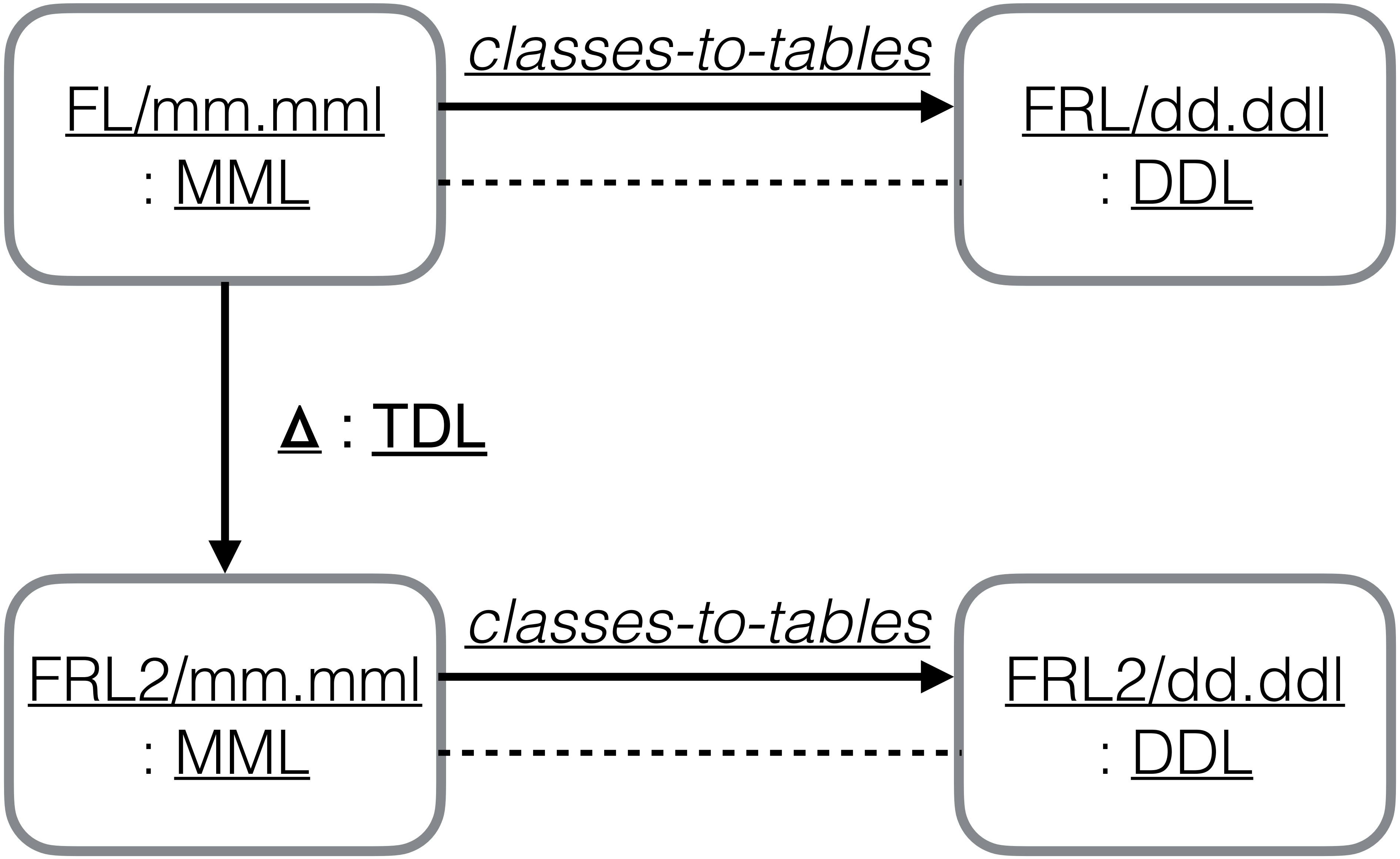
Let’s instantiate the pattern!



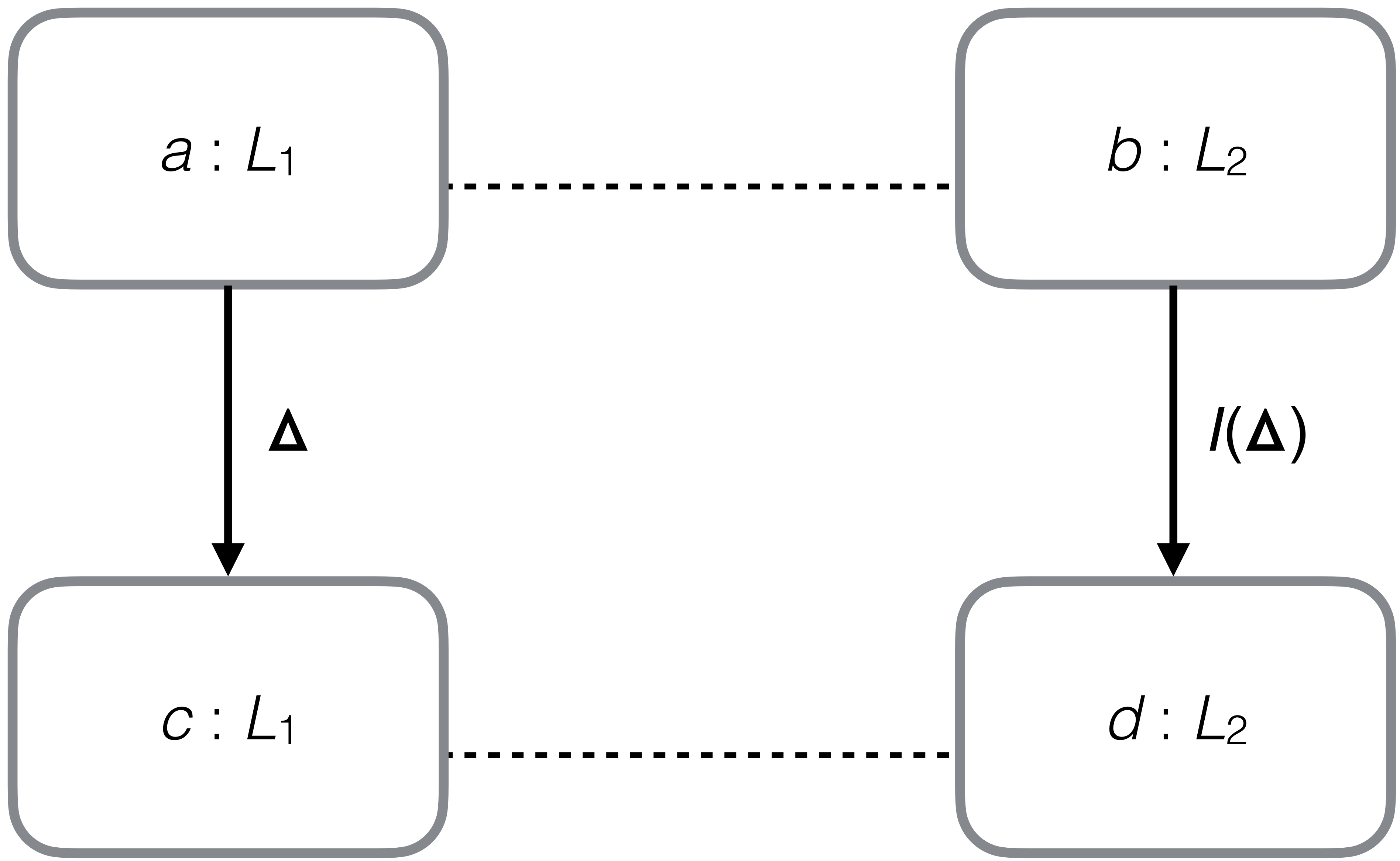
An 'instance' of CX by *mapping*

Everything is linked to artifacts!

FL — Family Language
MML — Metamodeling Language
DDL — Data Definition Language
TDL — Term Difference Language

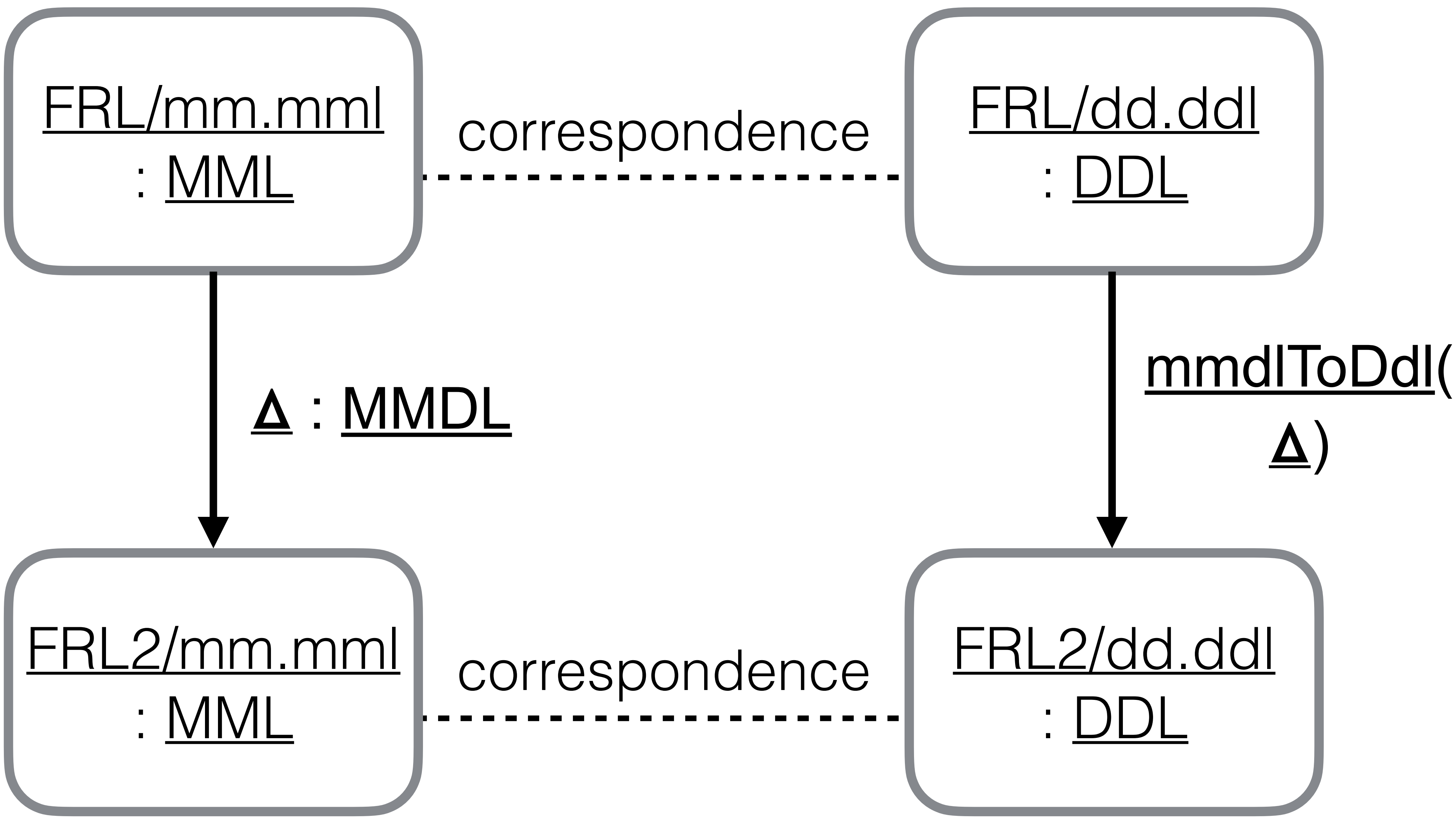


The ‘pattern’ of CX by *incremental mapping*

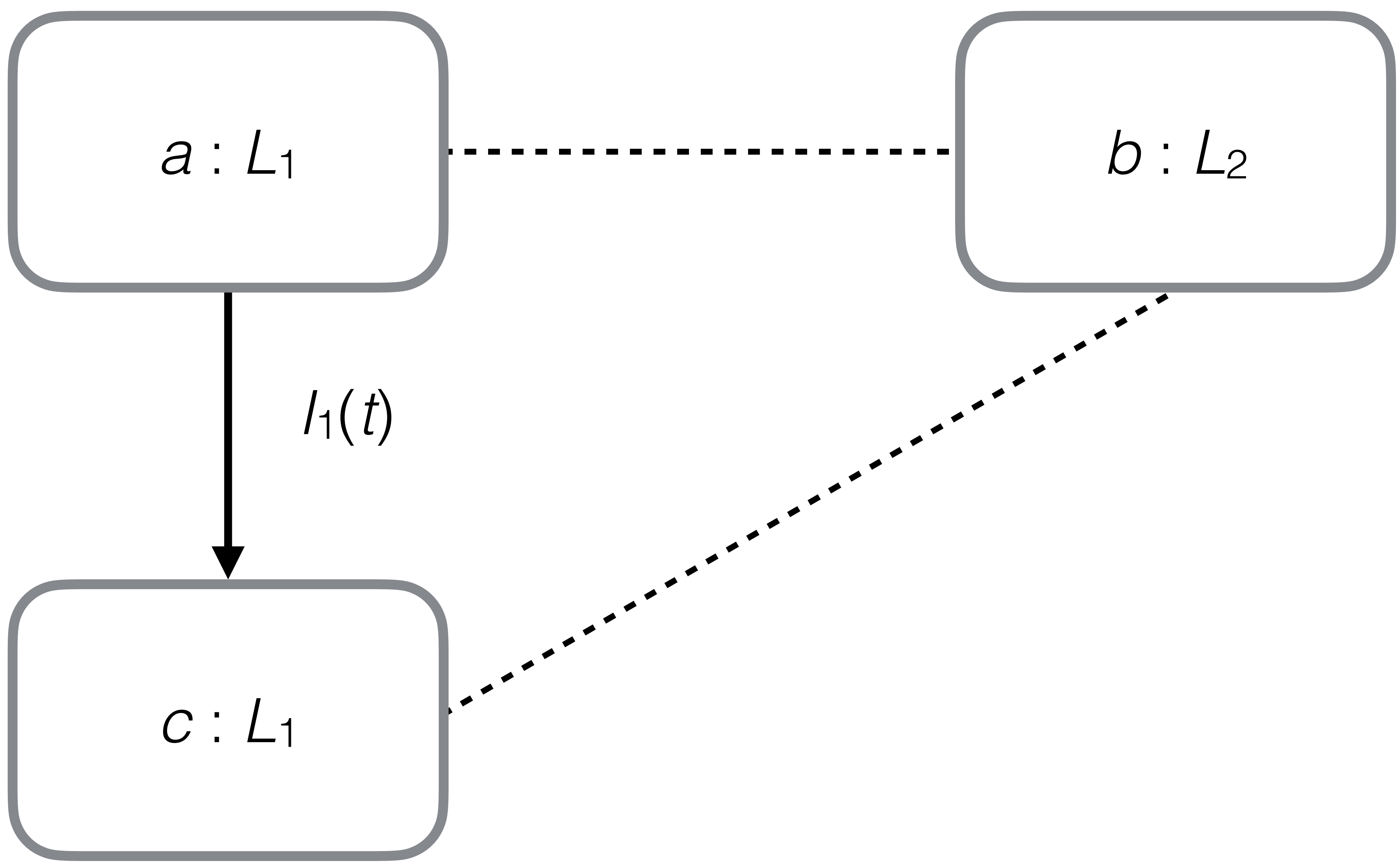


An ‘instance’ of CX by *incremental mapping*

FRL — Family ... Language
MML — Metamodeling Language
DDL — Data Definition Language
MMDL — Metamodel Difference Language

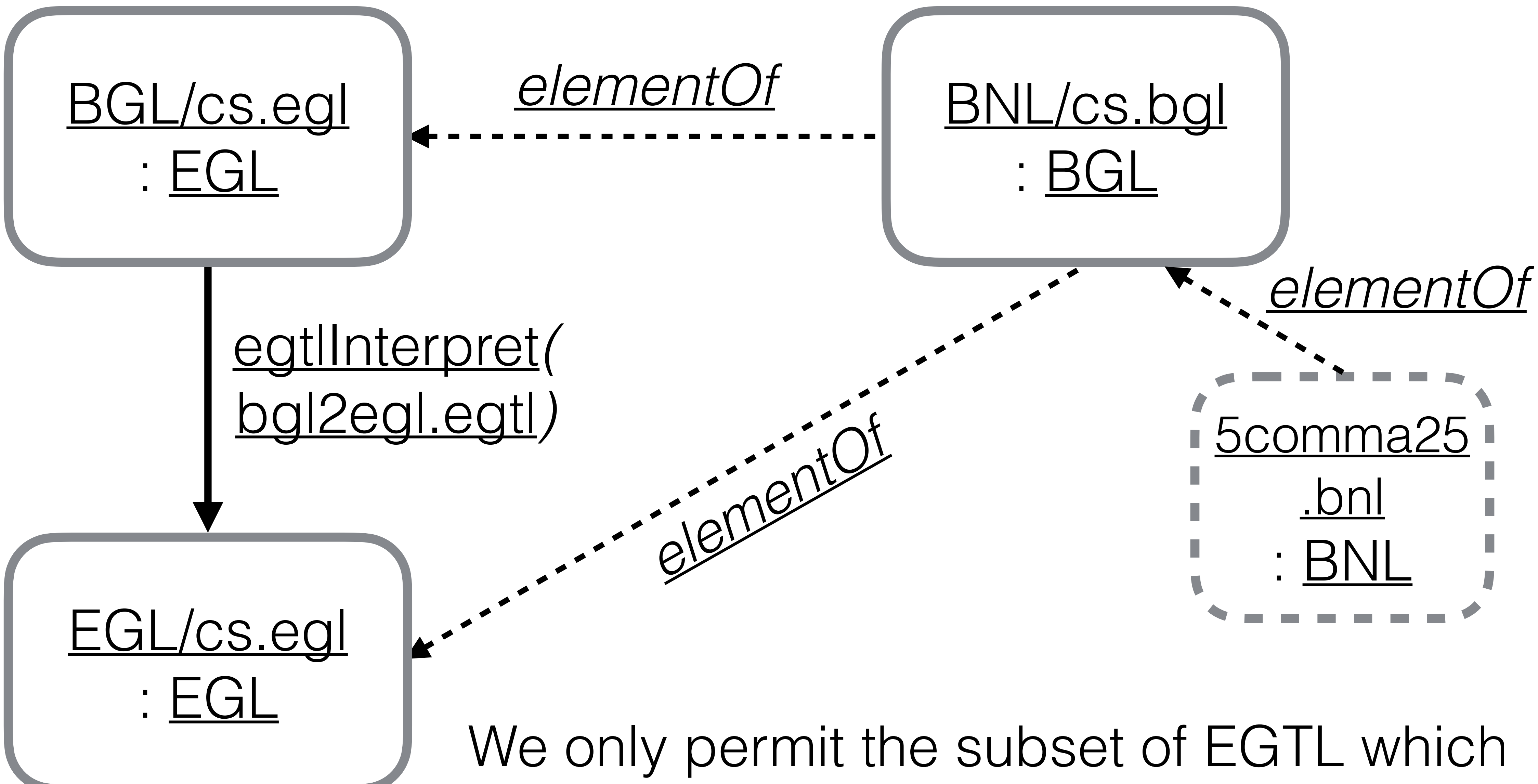


The ‘pattern’ of CX by *invariant consistency*



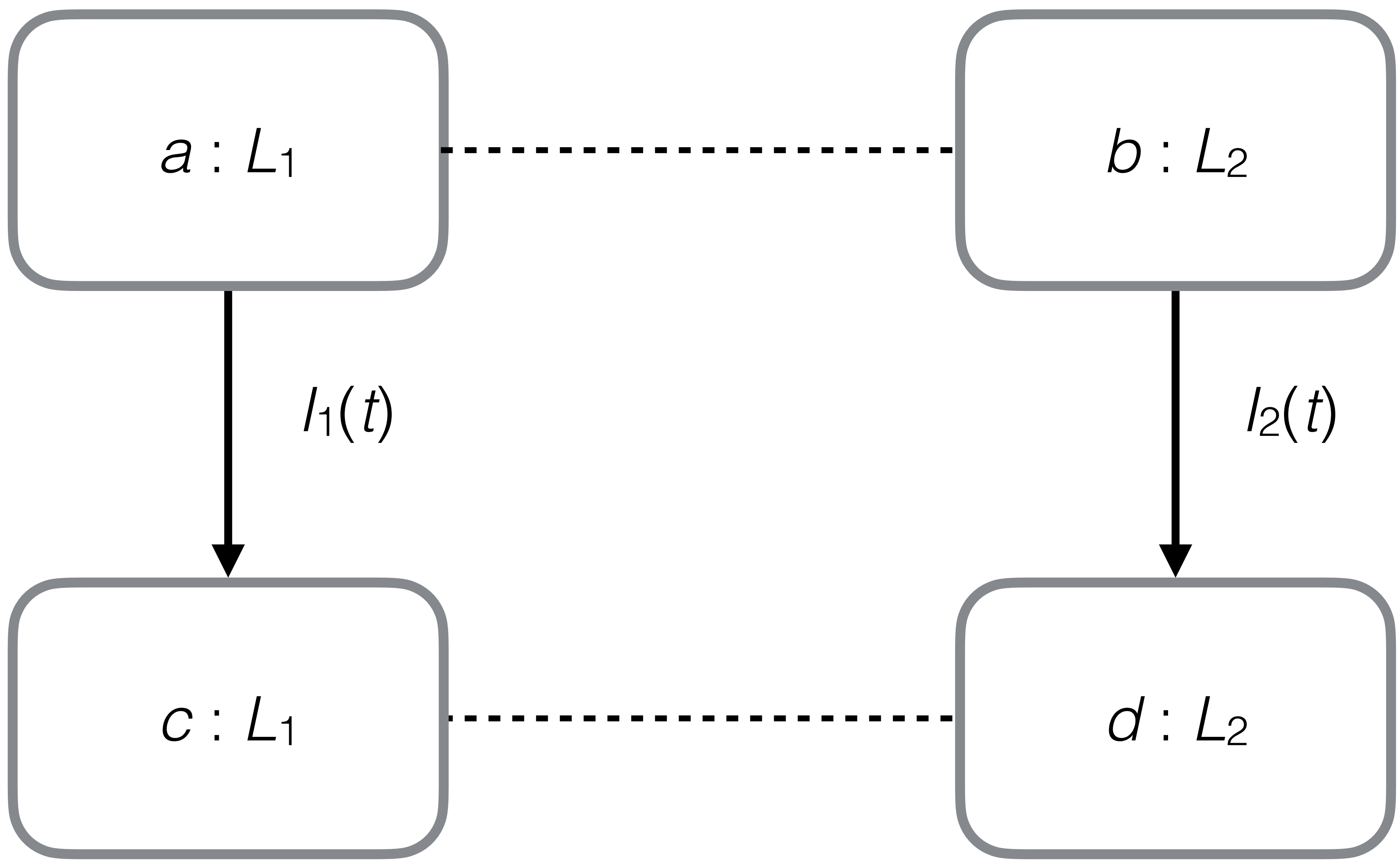
An 'instance' of CX by *invariant consistency*

BNL — Binary Number Language
BGL — Basic Grammar Language
EGL — Extended Grammar Language
EGTL — Extended Grammar Transformation Language



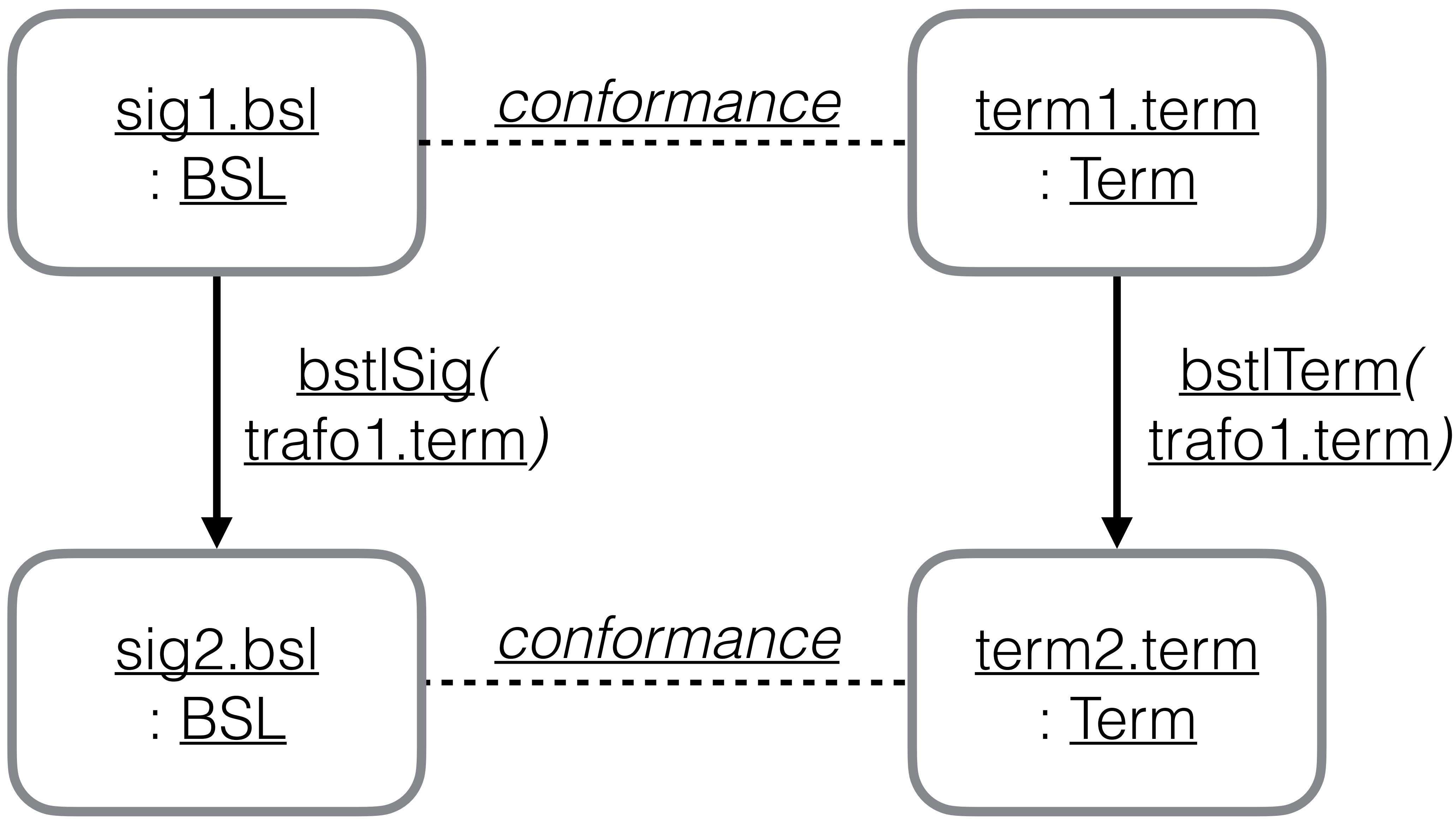
We only permit the subset of EGTL which serves language extension. See here.

The ‘pattern’ of CX by *co-transformation*

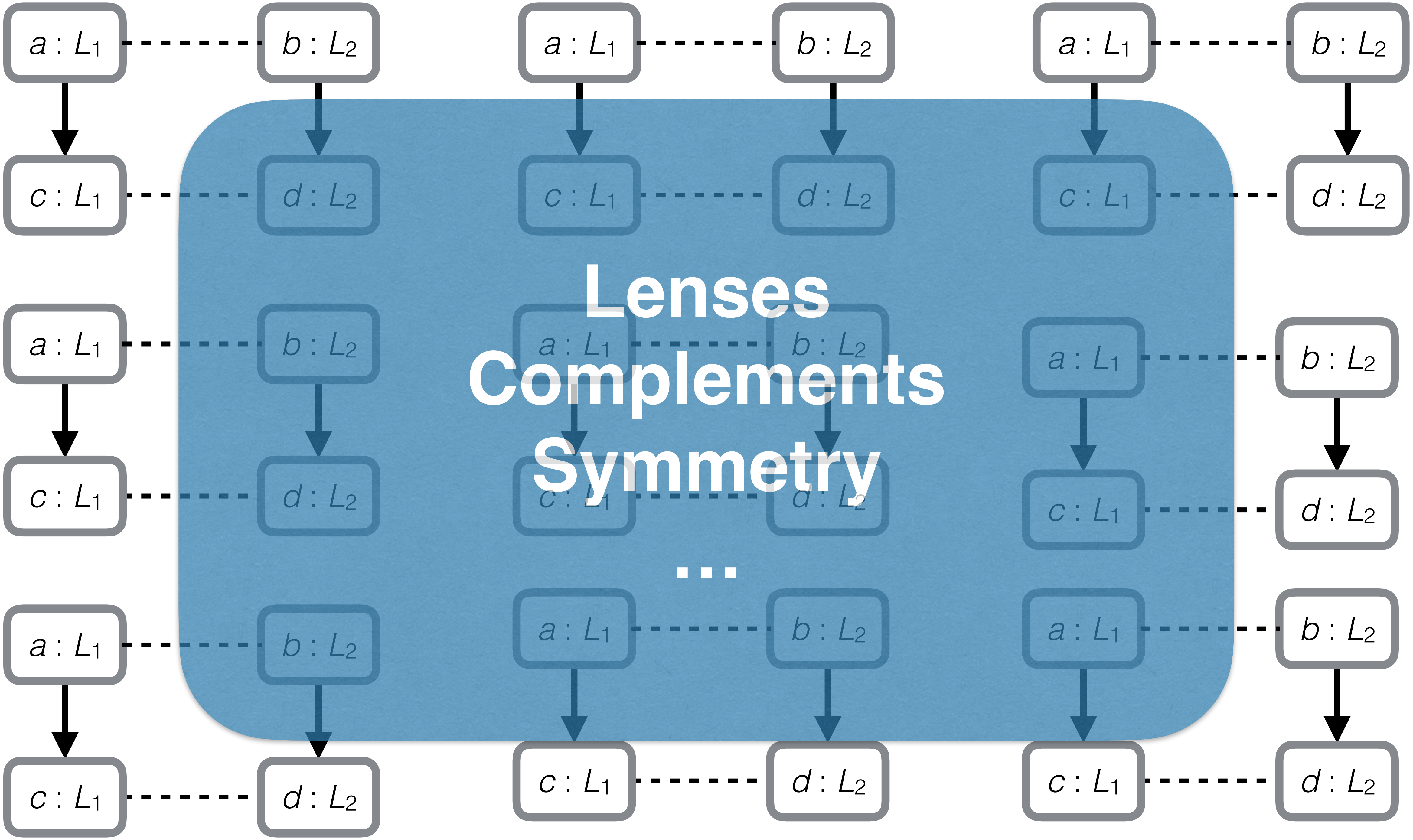


An ‘instance’ of CX by *co-transformation*

BSL — Basic Signature Language
Term — Terms conforming to signature
BSTL — Basic Signature Transformation Language

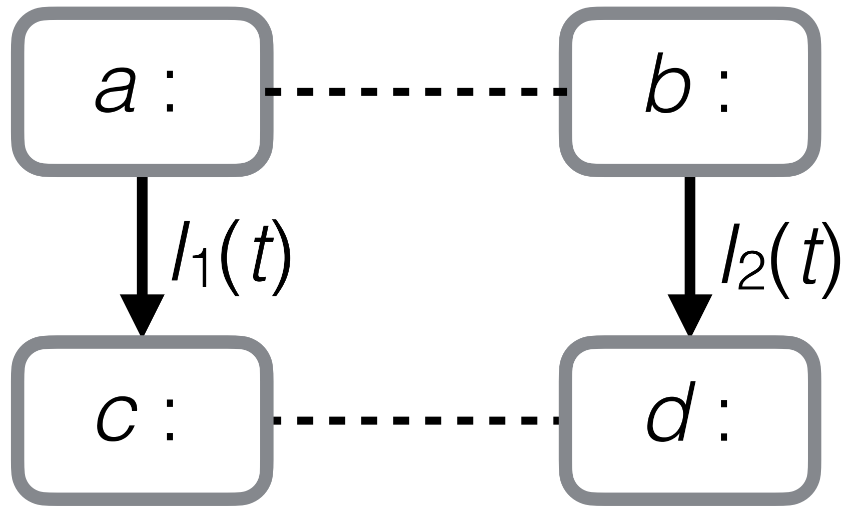


More CX



…

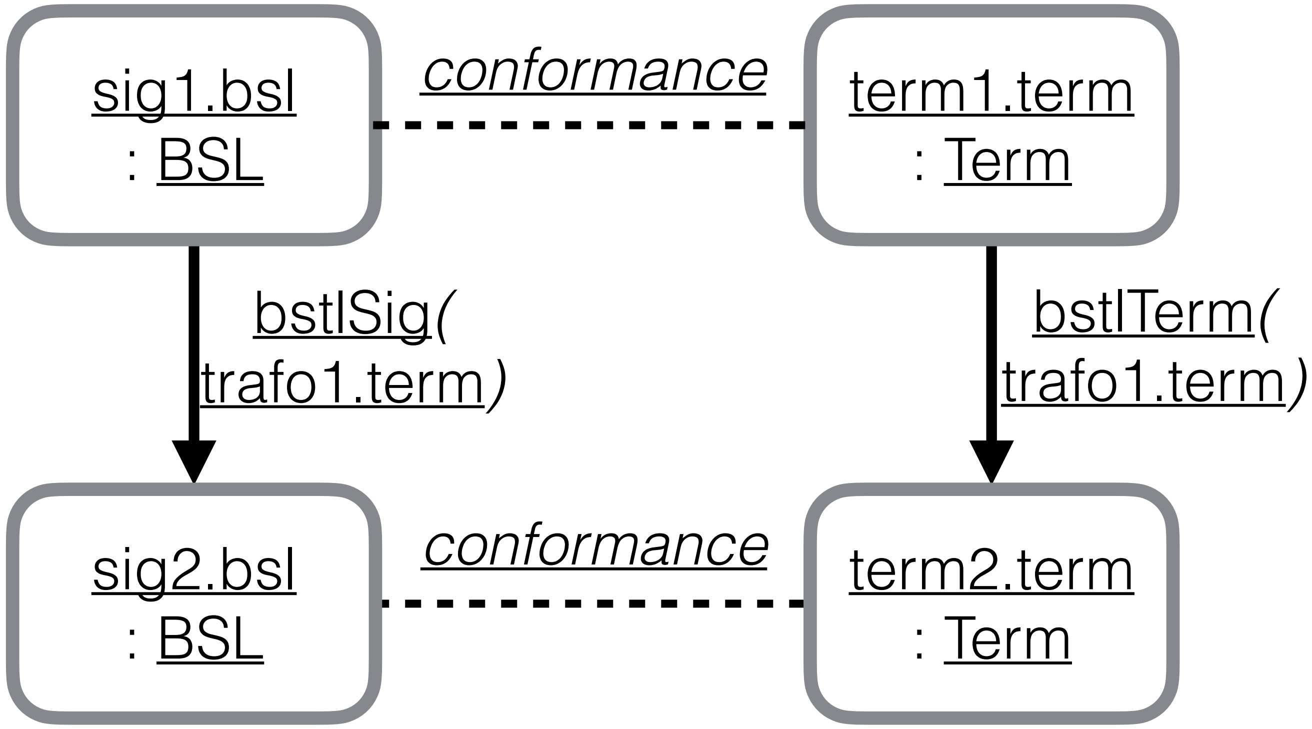
Higher level megamodel for CX by *co-transformation*



LAL megamodel cx.cotransformation

reuse coupling
reuse interpretation [$L_2 \mapsto L_1$, $Any_2 \mapsto Any_1$]
reuse interpretation [$L_1 \mapsto L_2$, $Any_1 \mapsto Any_2$]
axiom consistency { $\forall$ t $\in$ XL. $\forall$ a, c $\in L_1$. $\forall$ b, d $\in L_2$.
  consistent(a, b)
  $\wedge$ interpret(t, a) = c
  $\wedge$ interpret(t, b) = d $\Rightarrow$ consistent(c, d) }

Lower level megamodel CX by *co-transformation*



Ueber megamodel BSTL/tests/trafo1.ueber

```
[ elementOf('trafo1.term',bstl(term)),
  elementOf('term1.term',term),
  elementOf('term2.term',term),
  elementOf('sig1.term',bsl(term)),
  elementOf('sig2.term',bsl(term)),
  relatesTo(conformsTo,['term1.term','sig1.term']),
  mapsTo(interpret,['trafo1.term','term1.term'],['term2.term']),
  mapsTo(interpret,['trafo1.term','sig1.term'],['sig2.term']),
  relatesTo(conformsTo,['term2.term','sig2.term']) ].
```

Megamodel compilation for CX by *co-transformation*

LAL megamodel

```
reuse coupling
reuse interpretation [ L2 ↦ L1, Any2 ↦ Any1 ]
reuse interpretation [ L1 ↦ L2, Any1 ↦ Any2 ]
axiom consistency { ∀t ∈ XL. ∀a, c ∈ L1. ∀b, d ∈ L2.
  consistent(a, b)
  ∧ interpret(t, a) = c
  ∧ interpret(t, b) = d ⇒ consistent(c, d) }
```

Ueber megamodel

```
[ elementOf('trafo1.term',bstl(term)),
  elementOf('term1.term',term),
  elementOf('term2.term',term),
  elementOf('sig1.term',bsl(term)),
  elementOf('sig2.term',bsl(term)),
  relatesTo(conformsTo,['term1.term','sig1.term']),
  mapsTo(interpret,['trafo1.term','term1.term'],['term2.term']),
  mapsTo(interpret,['trafo1.term','sig1.term'],['sig2.term']),
  relatesTo(conformsTo,['term2.term','sig2.term']) ].
```

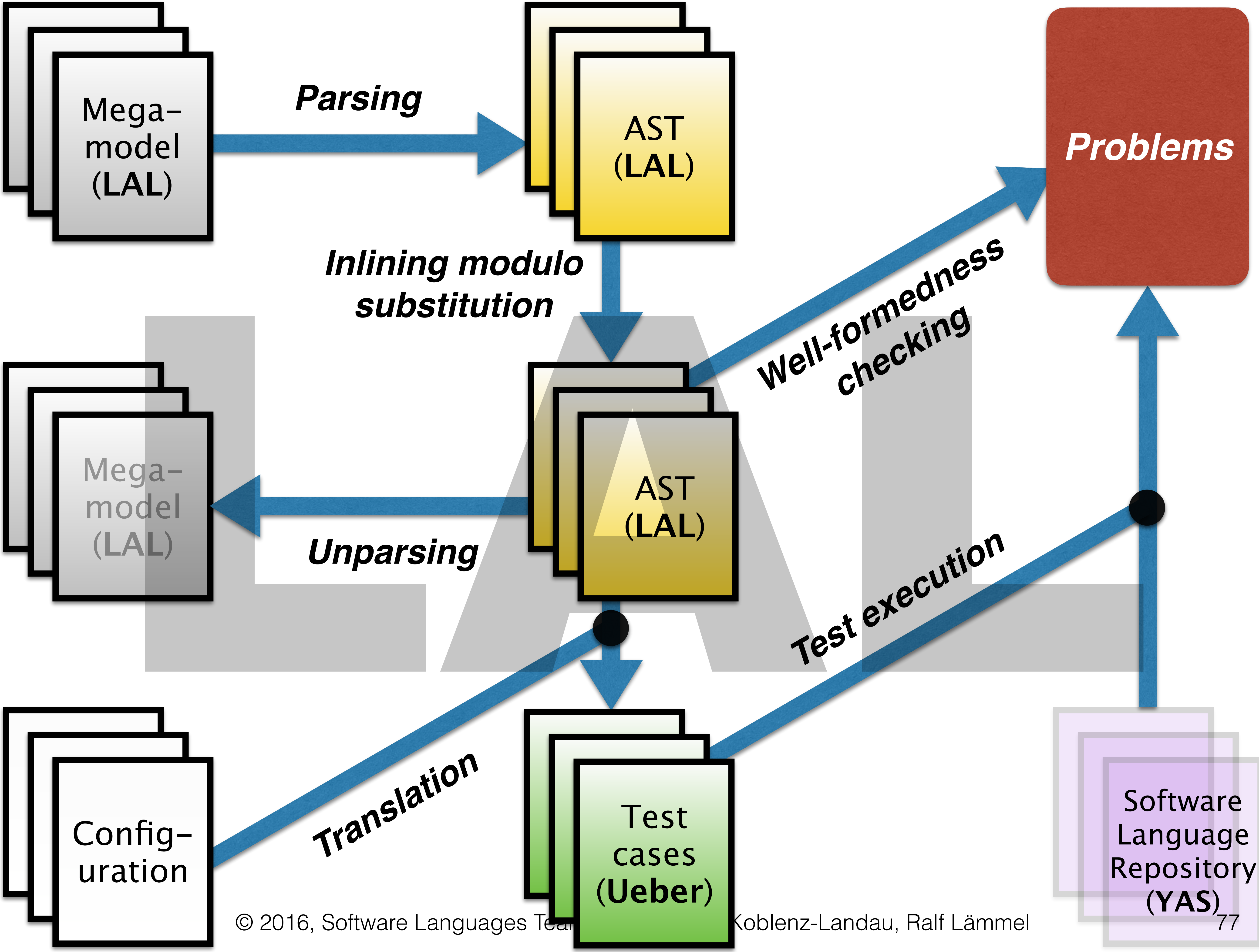
Configuration of compilation from higher to lower level megamodel

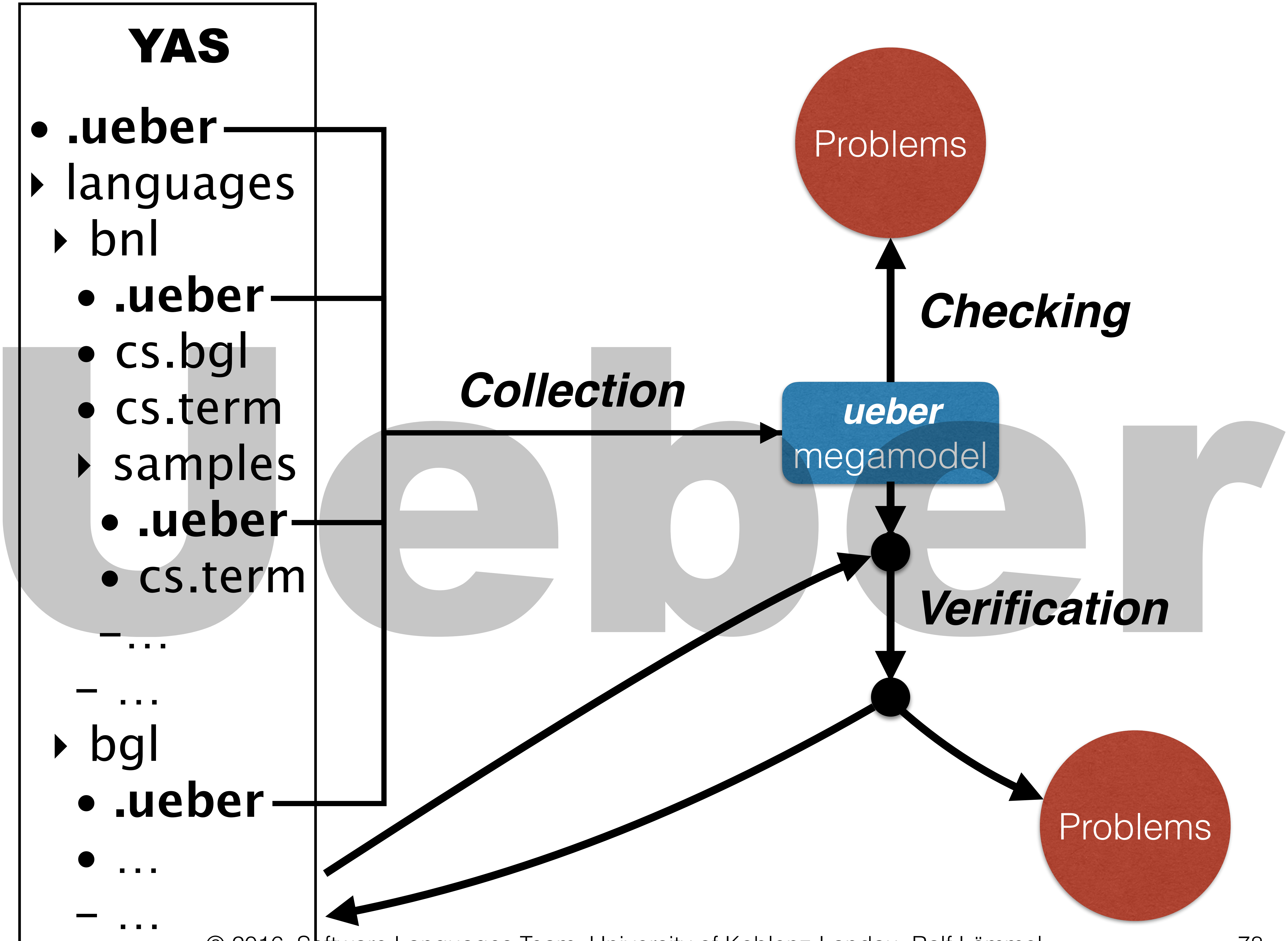
LAL configuration cx.cotransformation

```
[ sort('L1', term),
  sort('Any1', term),
  sort('L2', bsl(term)),
  sort('Any2', term),
  sort('XL', bstl(term)),
  sort('XAny', term),
  relation(consistent, conformsTo),
  axiom(consistency, [
    (t, 'trafo1.term'),
    (a, 'term1.term'),
    (b, 'sig1.term'),
    (c, 'term2.term'),
    (d, 'sig2.term') ])].
```

Summary of megamodel compilation

- A limited subset of predicate logic is considered.
- Forall becomes exists
- Implication becomes conjunction
- …
- Instantiate languages, artifacts, functions, relations.
- Rely on interpretations at low level.


Mega-
model
(LAL)
Parsing
AST
(LAL)
Problems
Inlining modulo
substitution
Well-formedness
checking
Mega-
model
(LAL)
AST
(LAL)
Unparsing
Test execution
Config-
uration
Translation
Test
cases
(Ueber)
Software
Language
Repository
(YAS)


YAS
• .ueber
‣ languages
‣ bnl
• .ueber
• cs.bgl
• cs.term
‣ samples
• .ueber
• cs.term
– …
– …
‣ bgl
• .ueber
• …
– …
Collection
ueber
megamodel
Checking
Problems
Verification
Problems
ueber


Call to arms!

Enjoy an SLE view on megamodeling

- i) **Megamodeling languages are DSLs**, subject to designated efforts in analysis, design, and implementation. (How to fight **fragmentation**?)

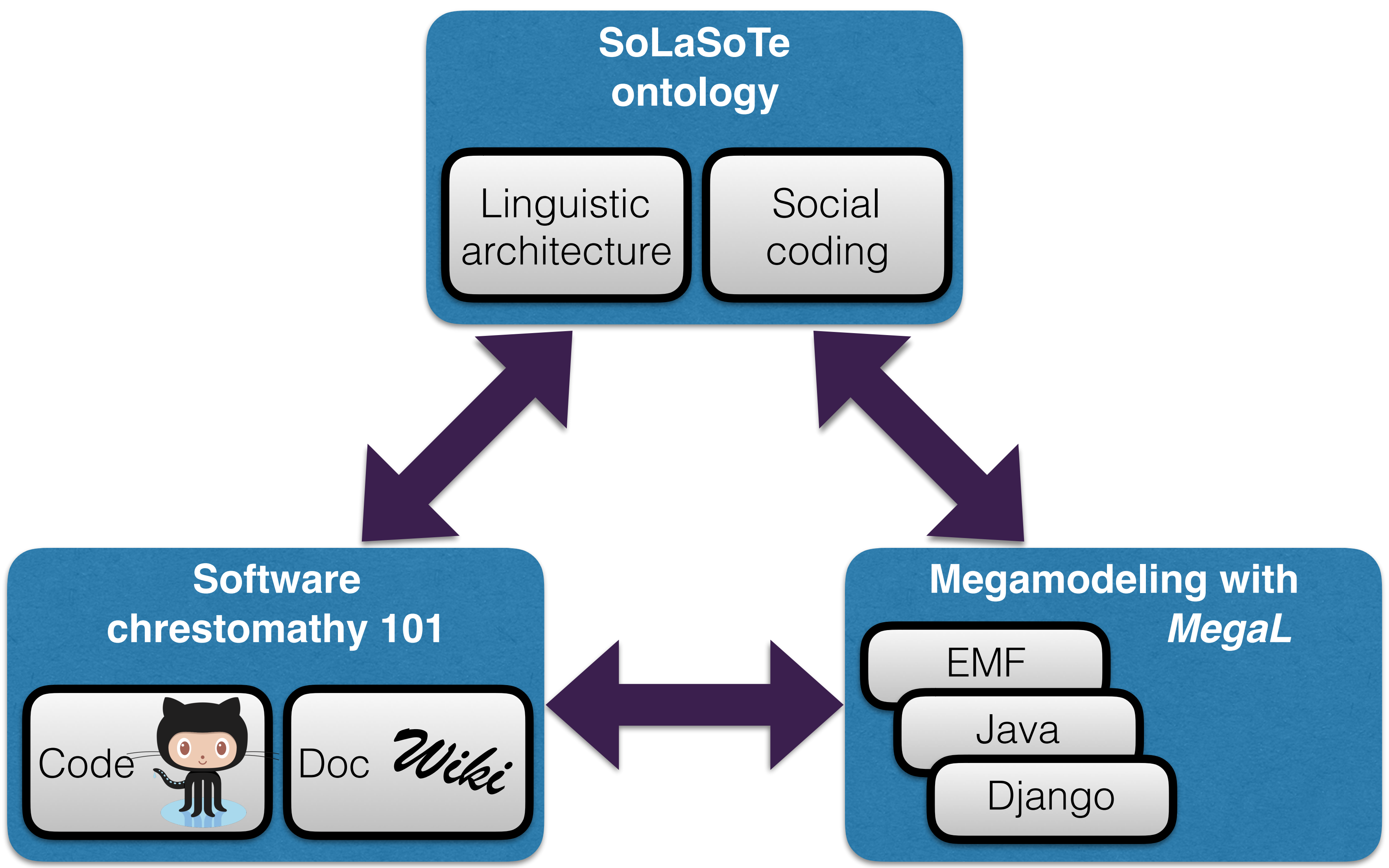
- ii) Especially analysis involves **ontology engineering** for concepts, languages, types of artifacts, and relationships. (How to organize such an effort? **Dagstuhl?**)

- iii) The basic DSL semantics serves **validation of megamodel instances.** (How to rework **technological spaces** to support such megamodeling seamlessly.)

- iv) The alignment of megamodels and reality requires **MSR**-style information retrieval and reverse engineering. (See basic ideas in our recent papers.)

- v) What's the AST to classical software languages, that's the **knowledge graph** to megamodeling DSLs. (Build a system / a knowledge graph that can be used by developers.)

Combine ontologies and chrestomathies in a megamodeling context



Source: Marcel Heinz, Ralf Lämmel, Andrei Varanovich: **Axioms of Linguistic Architecture**. MODELSWARD 2017: 478-486

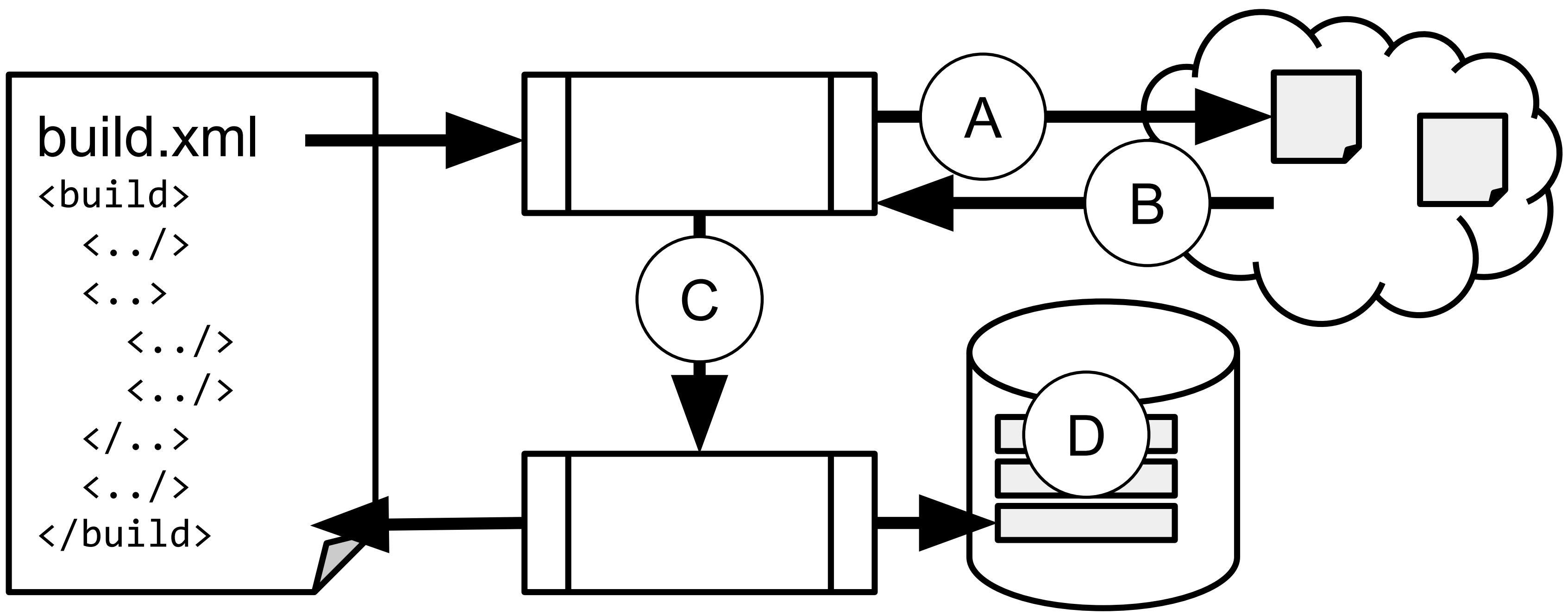
Support deep relationships

˅ xsdFiles	javaFiles
/xs:schema/xs:complexType	org/softlang/company/xjc/Employee.java
/xs:schema/xs:element#0	org/softlang/company/xjc/Company.java
/xs:schema/xs:element#1	org/softlang/company/xjc/Department.java
˅ xmlFile	objectGraph
˅ company/department#0	org.softlang.company.xjc.Department@5fd1a6aa
˅ employee#0	org.softlang.company.xjc.Employee@1a56a6c6
address:Utrecht	Utrecht
name:Erik	Erik
salary:12345	12345.0
› employee#1	org.softlang.company.xjc.Employee@748e432b

Explorable trace links in MegaL/Xtext+IDE for an extended XML story with involvement of XML-data binding, i.e., Java-class generation from an XML schema. The trace at the top shows similarity of XSD schema versus Java classes. The trace below shows similarity of XML document versus Java object (past deserialization). The indented rows are fragments (part of) the files. Fragmented URIs are used where applicable. Similar traces arise in the EMF story with generation and serialization of Sec. 2.

Source: Johannes Härtel, Lukas Härtel, Ralf Lämmel, Andrei Varanovich, Marcel Heinz: **Interconnected Linguistic Architecture**. Art Sci. Eng. Program. 1(1): 3 (2017)

Support transients in megamodels



A depiction of data flow and related transient states. A and B represent web request and response, respectively, C depicts piping of program output, and D shows transient data in memory or database.

Source: Johannes Härtel, Lukas Härtel, Ralf Lämmel, Andrei Varanovich, Marcel Heinz: **Interconnected Linguistic Architecture**. Art Sci. Eng. Program. 1(1): 3 (2017)

Embrace principles of interconnection

	Traceability links	Artifact binding	Model inference	Pluggable analyses	Explorable connections	Modularized models	Semantic annotations	Transient artifacts
[30]		·		●		●		·
[7]	●	·		·				
[25]	●	●	●	●		·		
[15]	·			·		·		
[11]	●	·			·			
[17]	●	·	●		·			
[12]	●	·	●	·				
[3]	●	·	●		·			
[4]	●	●	●	·	·			
([10])	●	●	·	·	●		●	

Source: Johannes Härtel, Lukas Härtel, Ralf Lämmel, Andrei Varanovich, Marcel Heinz: **Interconnected Linguistic Architecture**. Art Sci. Eng. Program. 1(1): 3 (2017)

Enable renarration of megamodels

Consider the following megamodel (in fact, megamodeling pattern) of a file and a language being related such that the former (in terms of its content) is an element of the latter.

```
[Label="File with language", Operator="Addition"]
+ ?aLanguage : Language // some language
+ ?aFile : File // some file
+ aFile elementOf aLanguage // associate language with file
```

In a next step, let us instantiate the language parameter to actually commit to the specific language *Java*. Thus:

```
[Label="A Java file", Operator="Instantiation"]
+ Java : Language // pick a specific language
+ aFile elementOf Java // associate the file with Java
- ?aLanguage : Language // removal of language parameter
- aFile elementOf aLanguage // removal of reference to language parameter
```

Source: Ralf Lämmel, Vadim Zaytsev: **Language Support for Megamodel Renarration**. XM@MoDELS 2013: 36-45

END OF SLIDE DECK