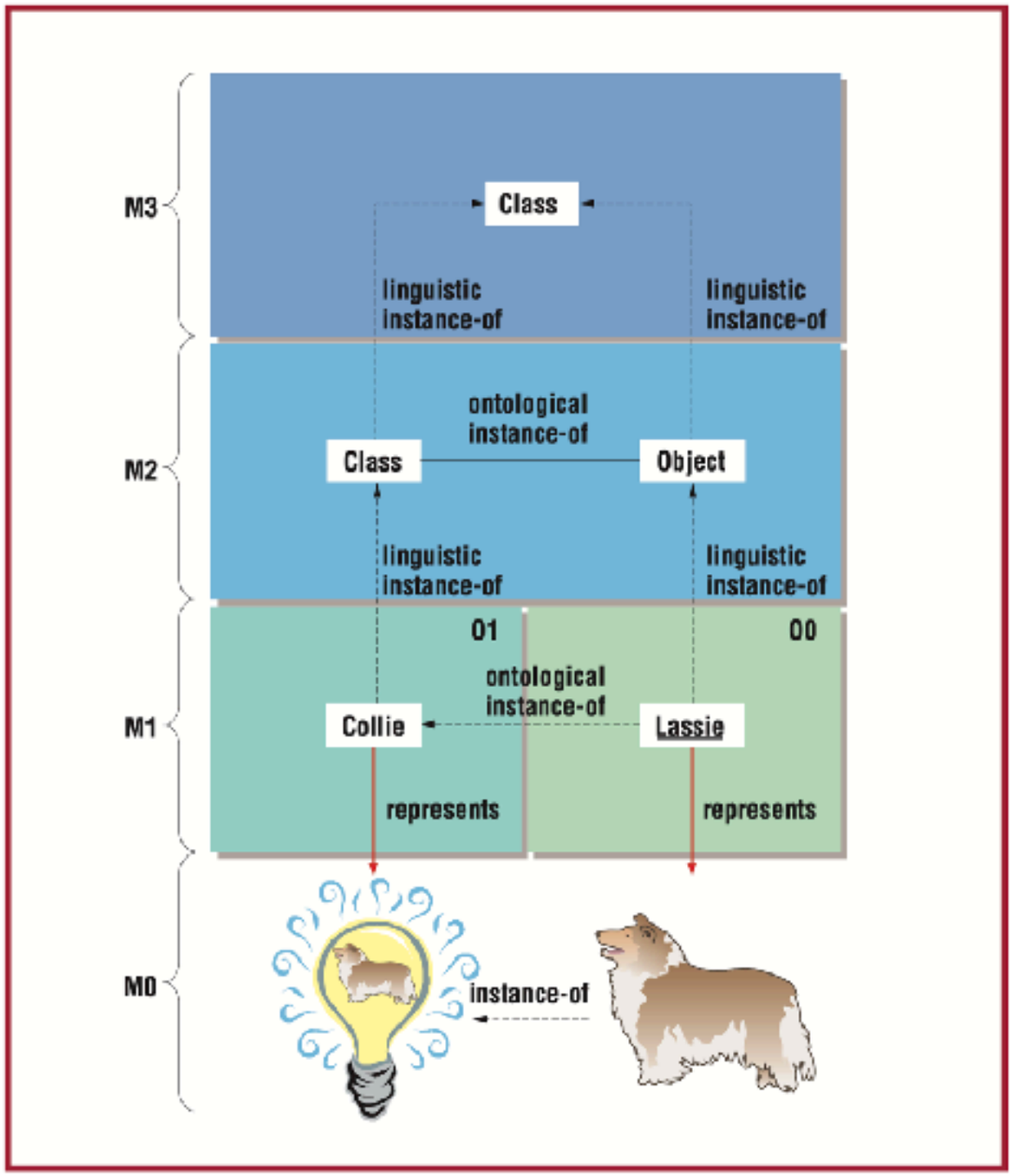
Megamodels on the Catwalk
Keynote at MODELSWARD 2021

Ralf Lämmel, ~~Facebook, London and~~ University of Koblenz-Landau, 2021-02-09

Just to be clear — no dogs today!



Source:
https://www.rose-hulman.edu/class/csse/csse490-mbse/Readings/MDD-Metamodeling.pdf

Megamodels are domain-specific models (like most or all models).
What's the domain supported by megamodels?
It depends! It may be, for example, the domain of model transformation or the domain of software technologies.

What is a megamodel anyways?
Roughly, a megamodel is a model whose elements are software artifacts (such as models or programs).
In fact, those artifacts aren't necessarily as concrete as actual models; they could be opaque, as in the case of libraries.
The relationships in a megamodel thus relate software artifacts (with conformance being the obvious example).
Hold on, megamodels also capture knowledge about the domain.
Thus, naturally, model elements and relationships also concern software concepts and software languages.
Megamodels come in many flavors: prescriptive, descriptive, executable, exemplified, renarratable, etc.

Megamodels serve abstraction (like most or all models).
What sort of abstraction do megamodels support?
Megamodels abstract by treating model elements for software artifacts effectively as variables.
Model elements for software concepts and languages are supposedly drawn from an appropriate ontology.
All that matters are the constraints on the model elements expressed by the relationships.
Thus, megamodels are like patterns of conglomerations of related artifacts, concepts, and languages.

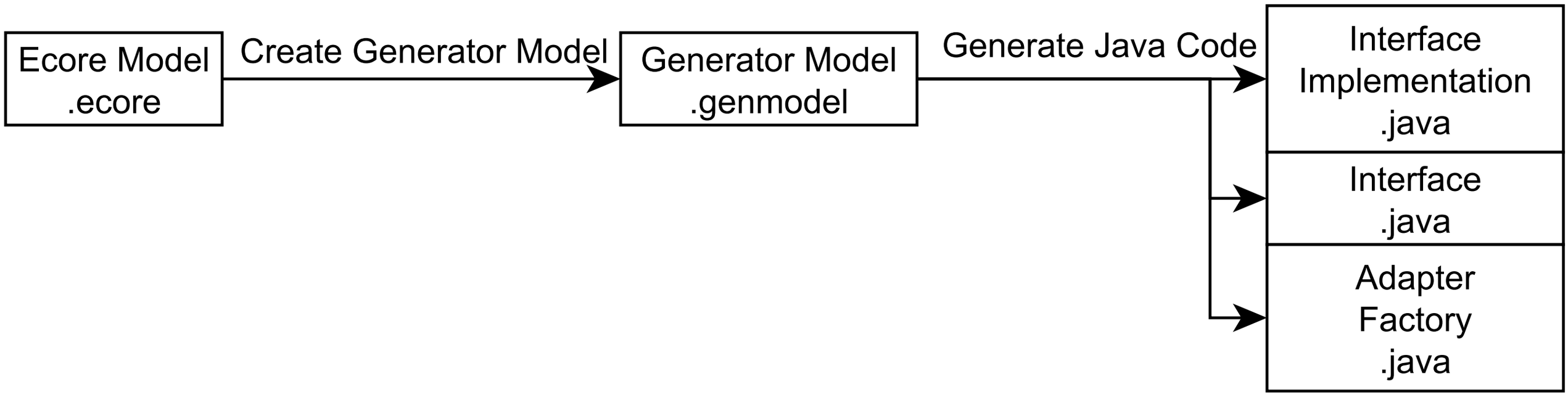
Outline

Time for a catwalk to organize the space of megamodeling.

Megamodeling is a niche, if you go by explicit mentioning of the paradigm in software engineering.
Megamodeling is omnipresent, if you acknowledge all related hacks and workarounds that are found in the wild.
In this talk, I also hint at where I saw some or where I wanted more megamodeling at Facebook in software development.
Let's discuss how megamodeling could be generalized and used more profoundly in software engineering.
To this end, we need to continue working on these premises:
i) Megamodeling languages are DSLs, subject to designated efforts in analysis, design, and implementation.
ii) Especially analysis involves ontology engineering for concepts, languages, types of artifacts, and relationships.
iii) The most important DSL semantics serves validation of megamodel instances against a megamodel.
iv) The alignment of megamodels and reality requires MSR-style information retrieval and reverse engineering.
v) What's the AST to classical software languages, that's the knowledge graph to megamodeling DSLs.

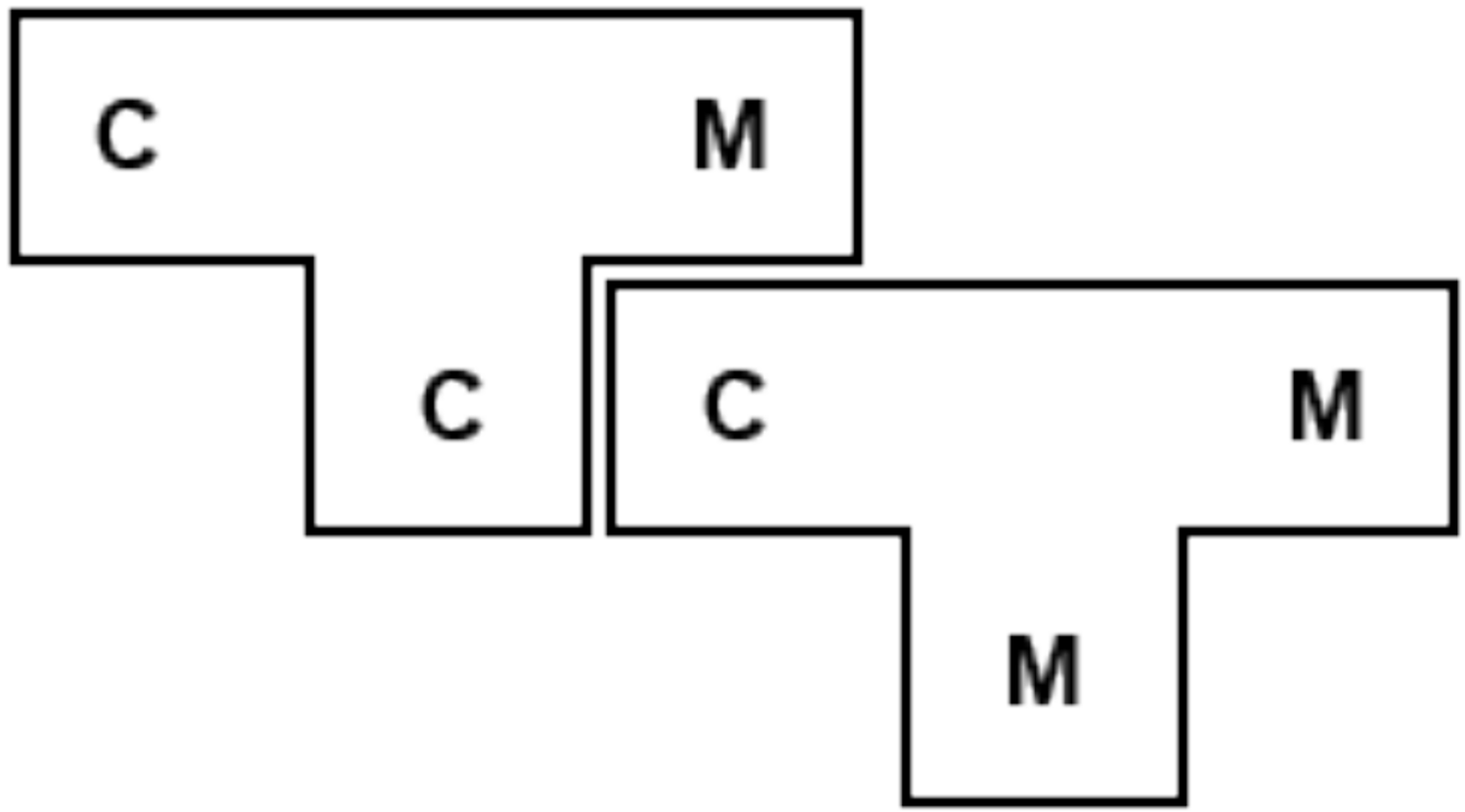

The term ‘megamodel’ lacks clarity in definition or demarcation.

A megamodel for EMF code generation

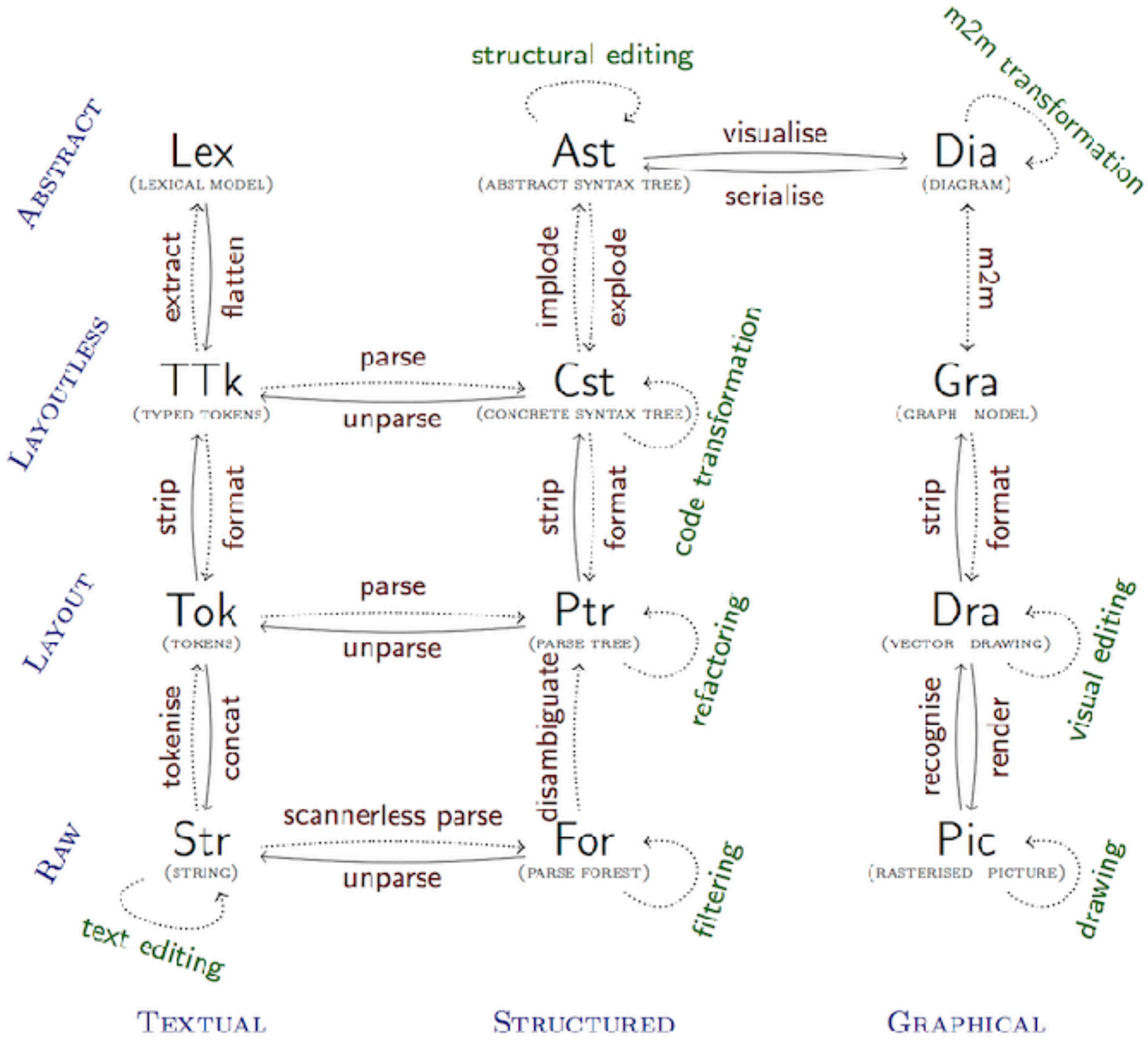


Source: Marcel Heinz, Johannes Härtel, Ralf Lämmel: **Reproducible Construction of Interconnected Technology Models for EMF Code Generation**. J. Object Technol. 19(2): 8:1-25 (2020). See also conference version: Johannes Härtel, Marcel Heinz, Ralf Lämmel: EMF Patterns of Usage on GitHub. ECMFA 2018: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

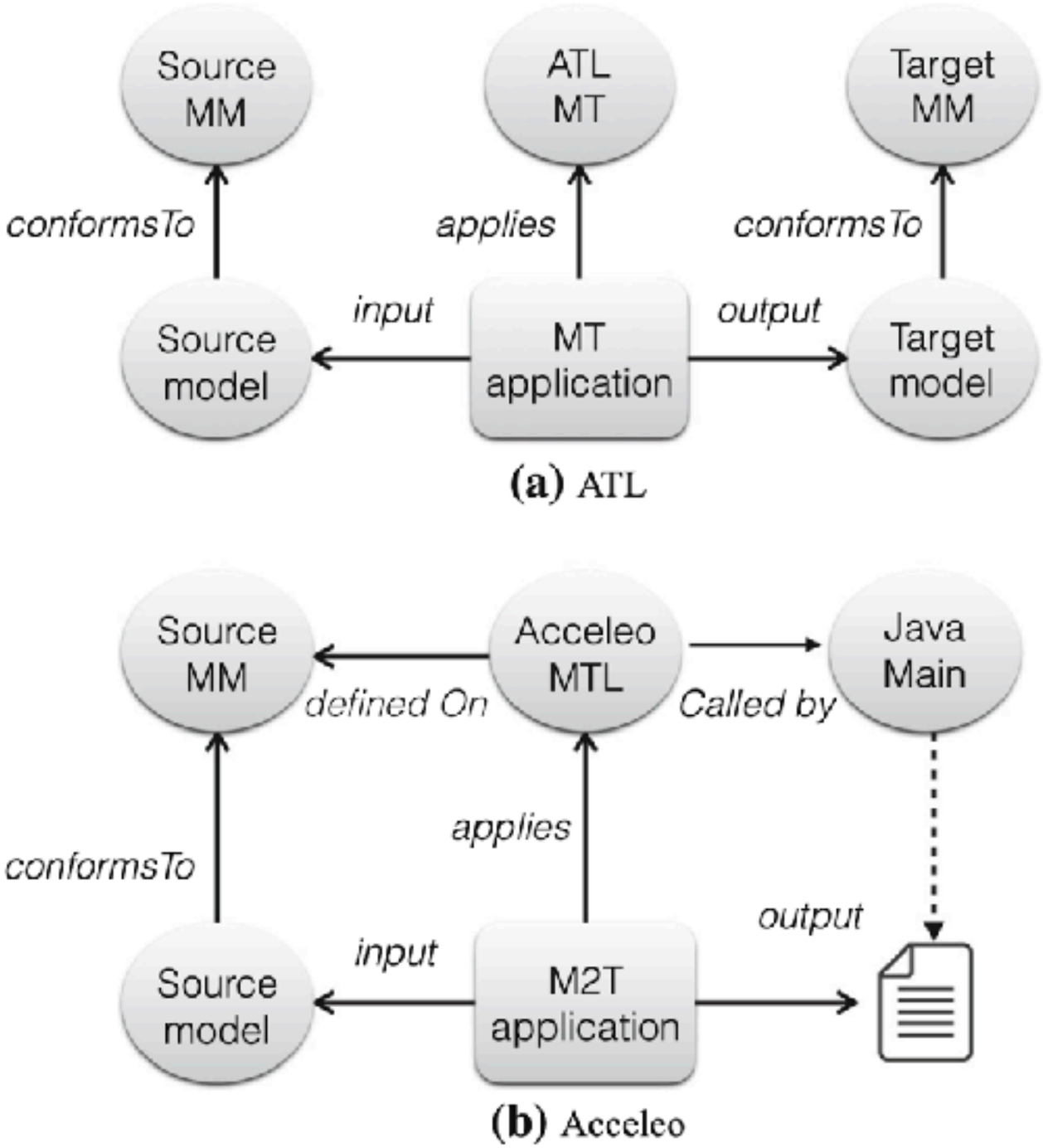
ANTLR.Notation ***domainOf*** *ANTLR.Generator*
Java ***rangeOf*** *ANTLR.Generator*

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) ↦ *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

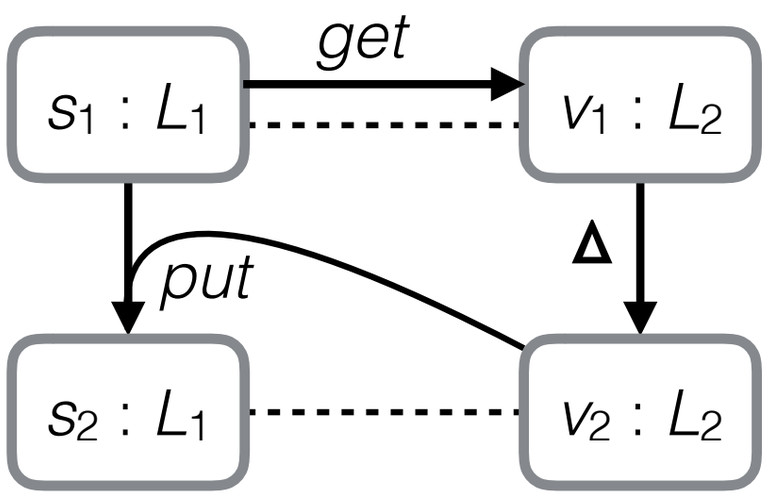
A megamodel for MT with ATL/Acceleo



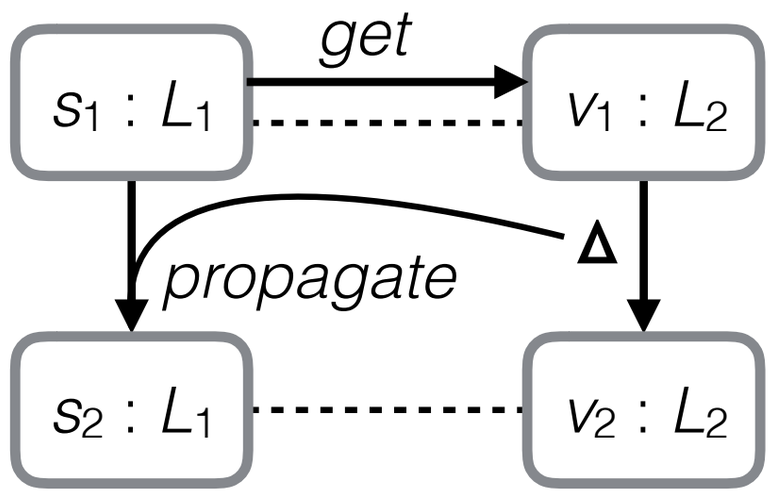
Source: Juri Di Rocco, Davide Di Ruscio, Johannes Härtel, Ludovico Iovino, Ralf Lämmel, Alfonso Pierantonio: **Understanding MDE projects: megamodels to the rescue for architecture recovery**. Softw. Syst. Model. 19(2): 401-423 (2020). See also conference version: Juri Di Rocco, Davide Di Ruscio, Johannes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

Megamodels for two basic BX patterns

State-based lenses



Delta-based lenses



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*

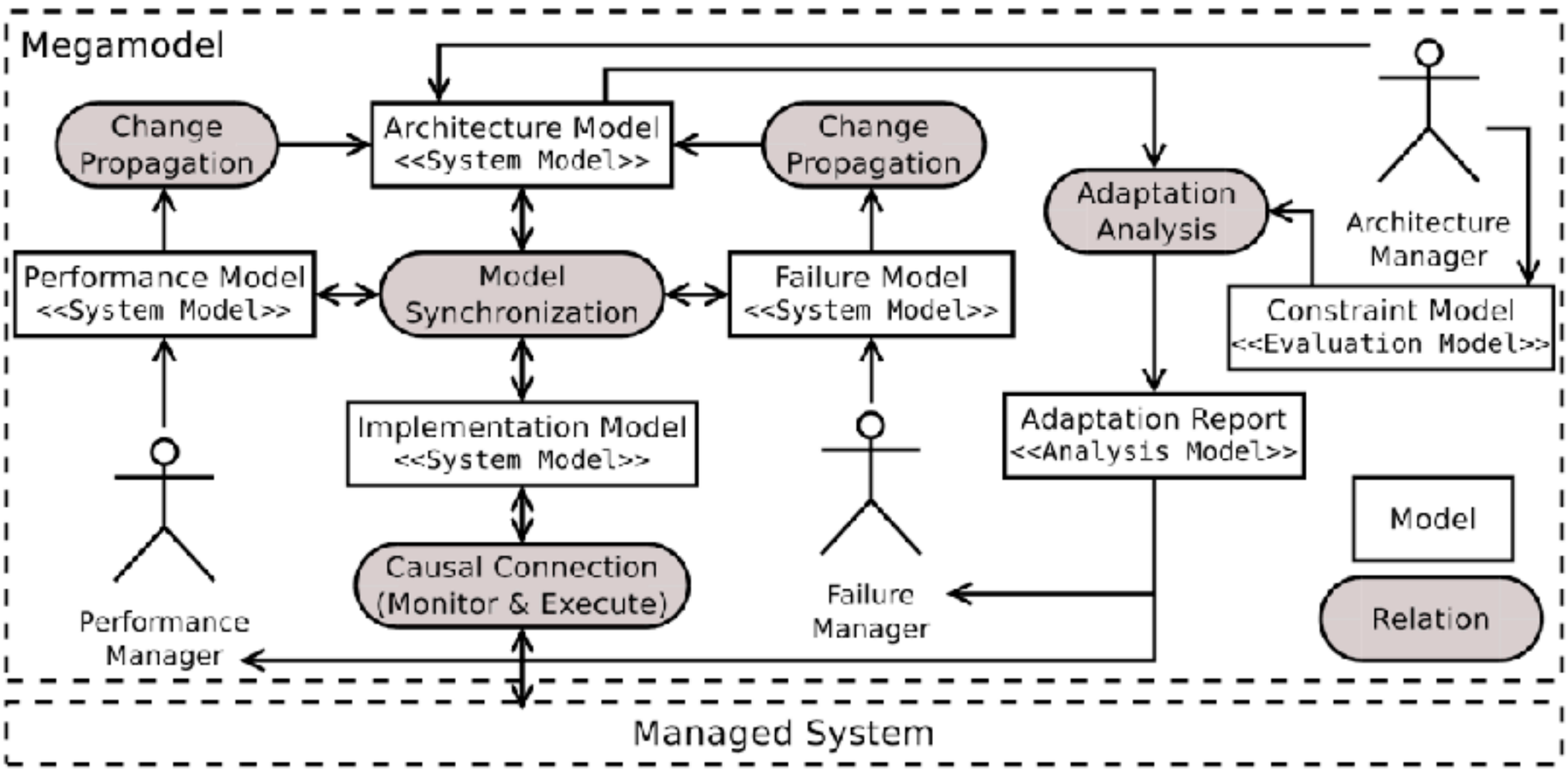
```
reuse cx.mapping [ mapping ↦ get ]
function get : L1 → L2
function put : L1 × L2 ↠ L1
axiom GetPut { ∀ s ∈ L1.
  put(s, get(s)) = s }
axiom PutGet { ∀ s1, s2 ∈ L1. ∀ v ∈ L2.
  put(s1, v) = s2 ⇒ get(s2) = v }
```

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)

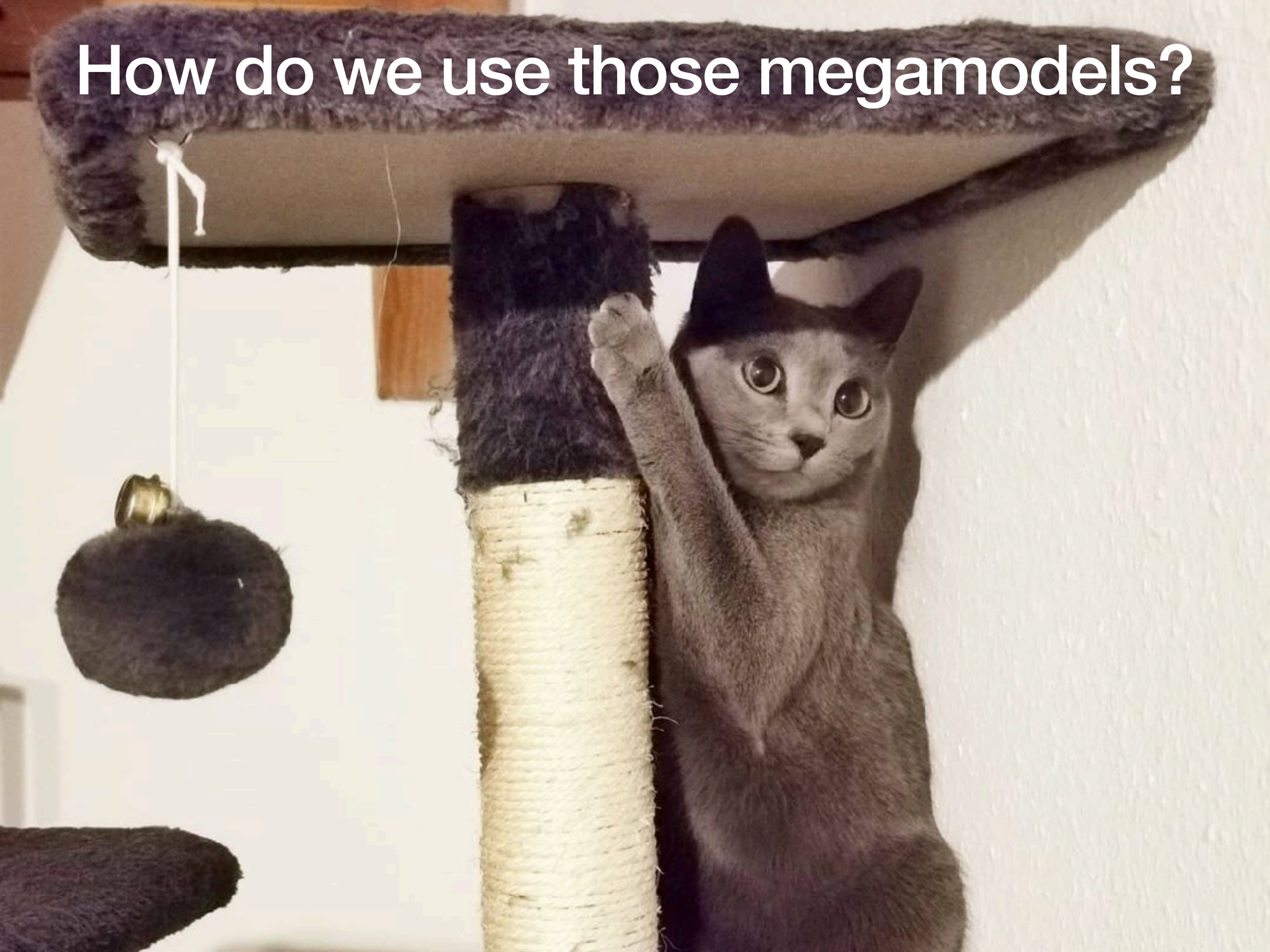


Source: https://arxiv.org/abs/1805.07396

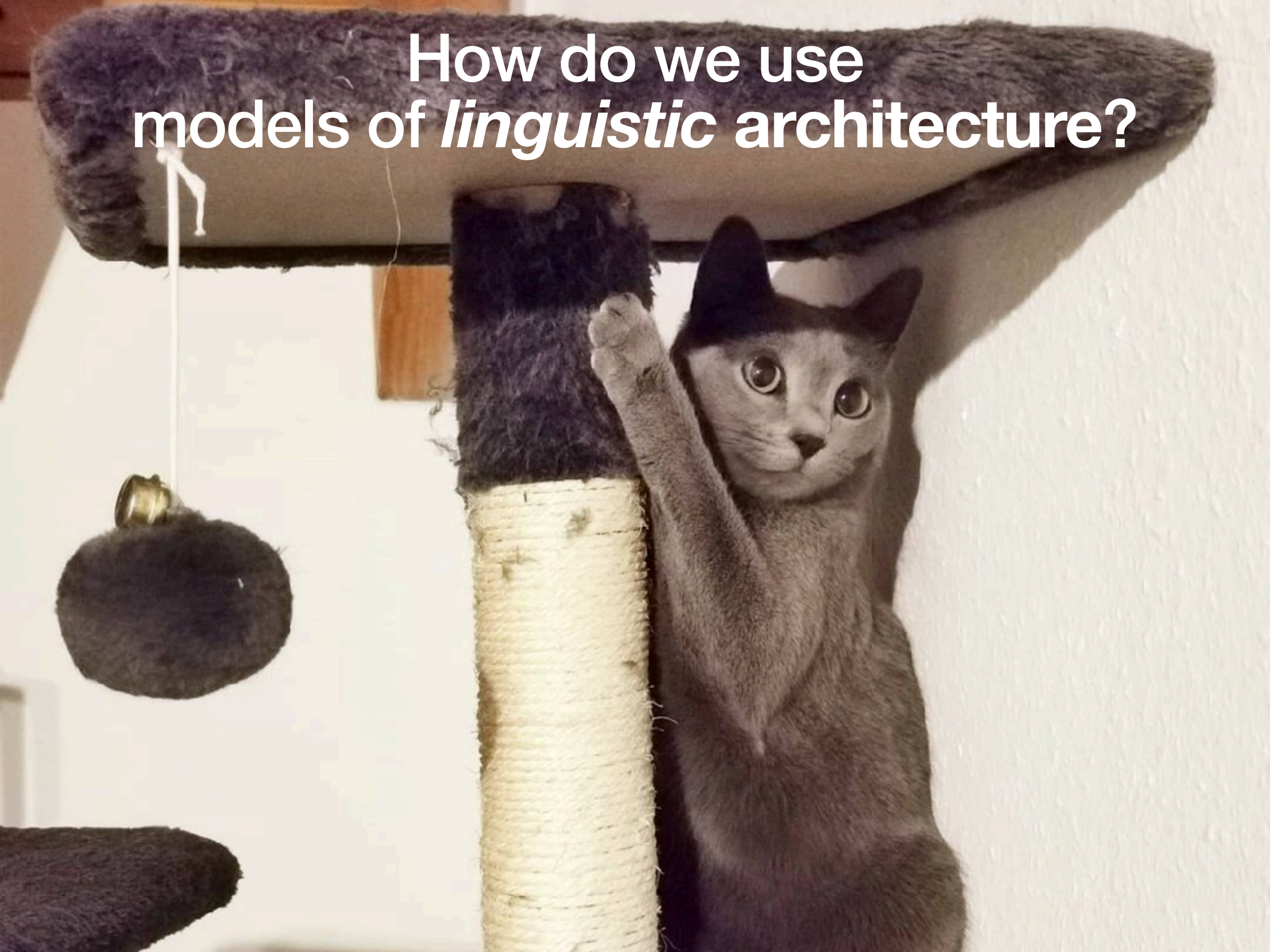
A lot of diversity!

- 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?
- …

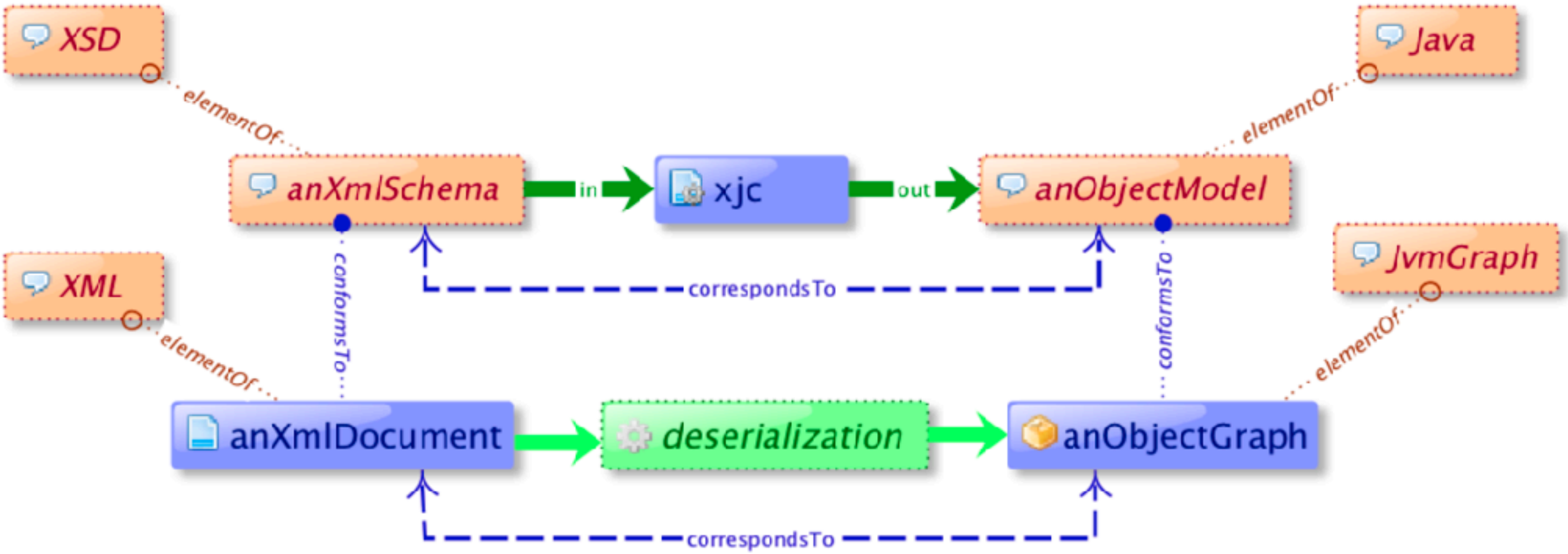
How do we use those megamodels?



How do we use models of *linguistic* architecture?

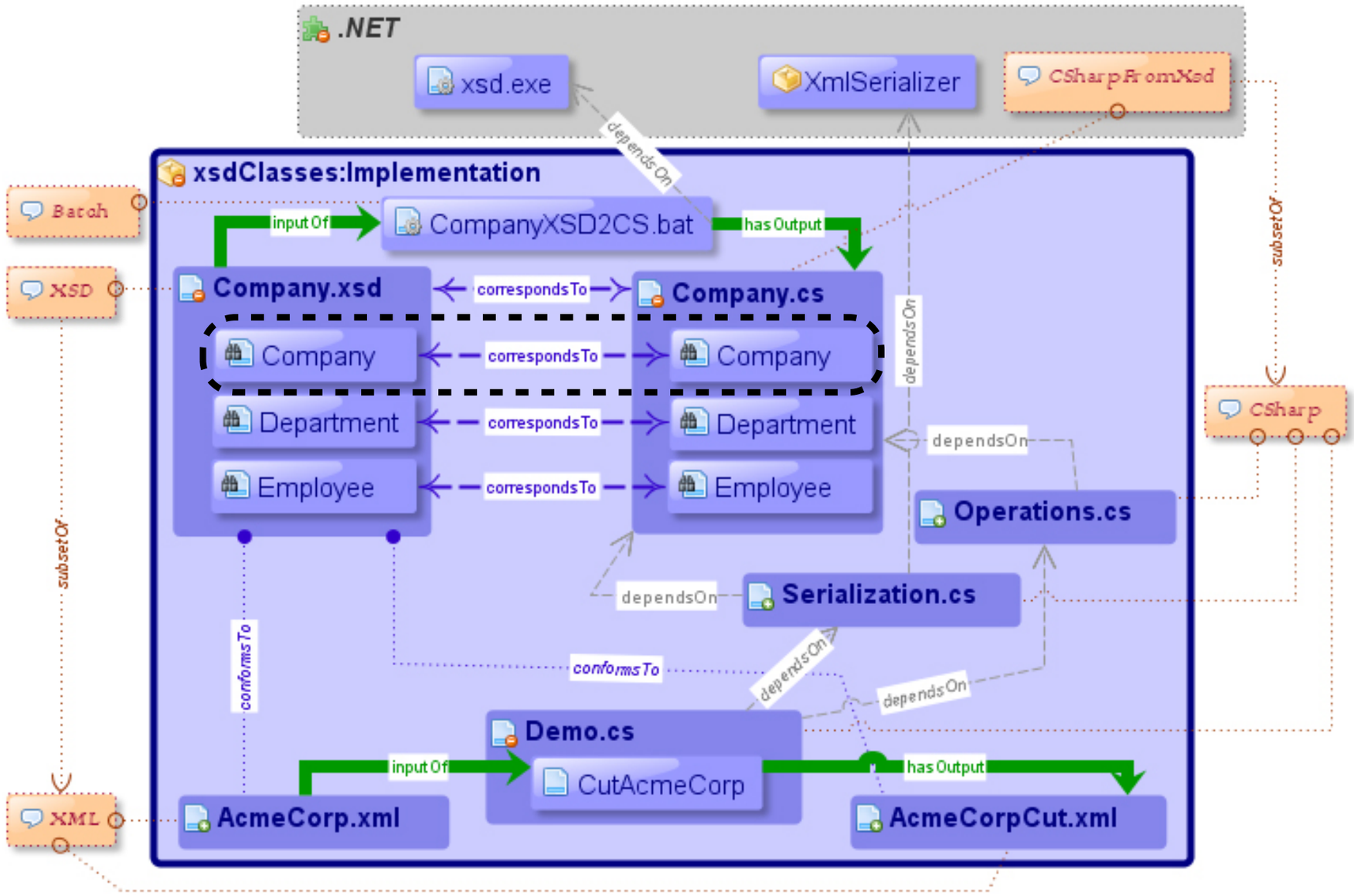


Linguistic architecture of XML-data binding in Java



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

… XML-data binding in C#



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

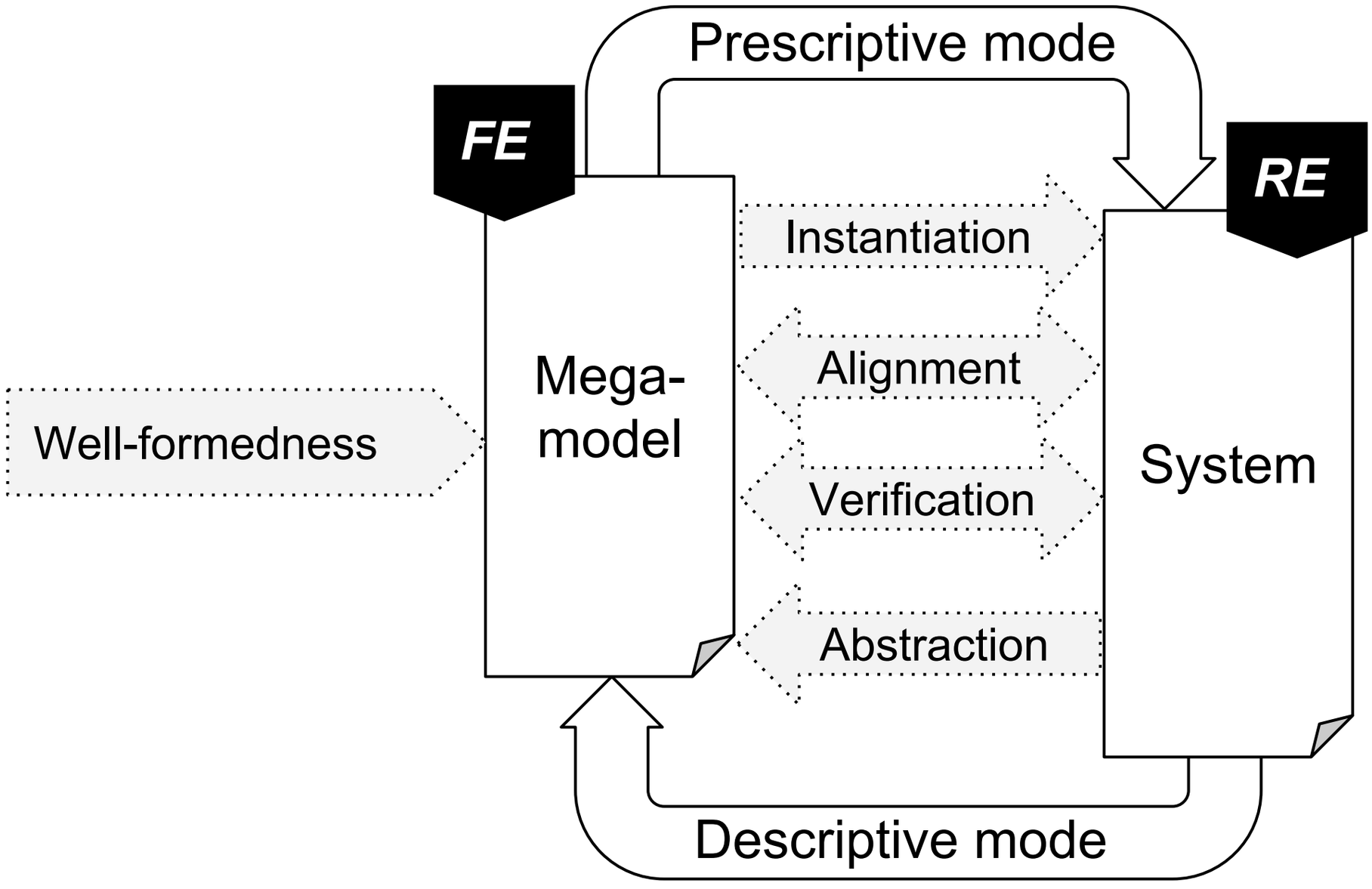
… XML-data binding in C#

```
_: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\"]" .
```

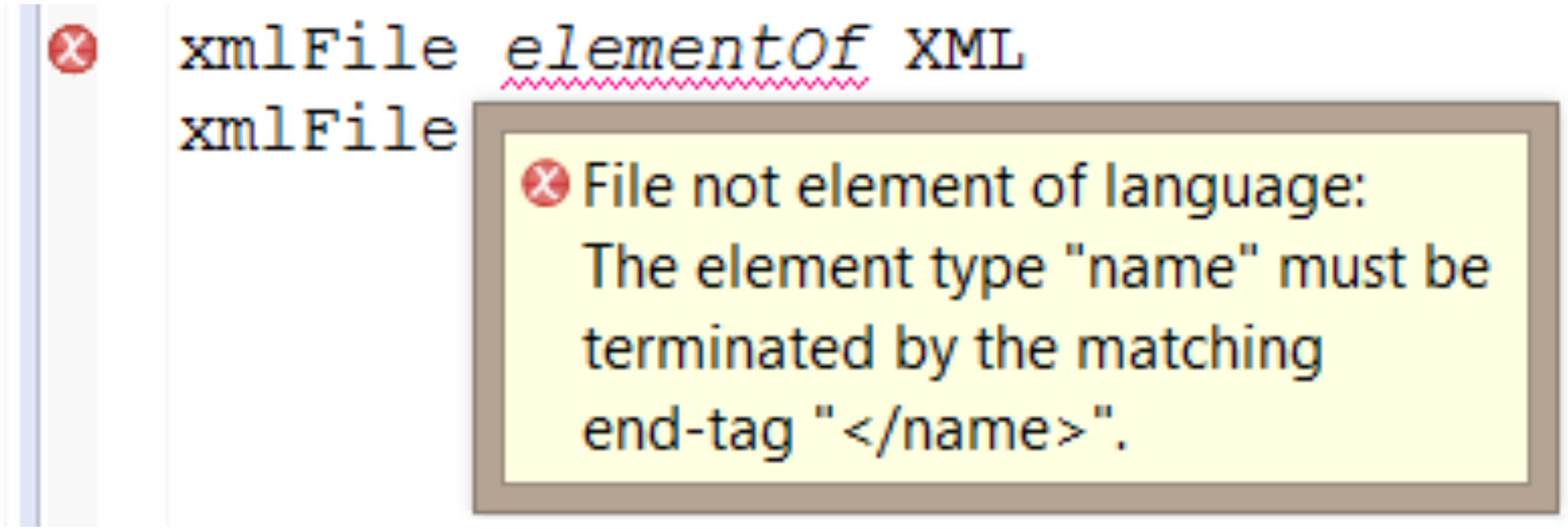
Source: Jean-Marie Favre, Ralf Lämmel, Andrei Varanovich: **Modeling the Linguistic Architecture of Software Products**. MoDELS 2012: 151-167

Linguistic architecture in a software development context



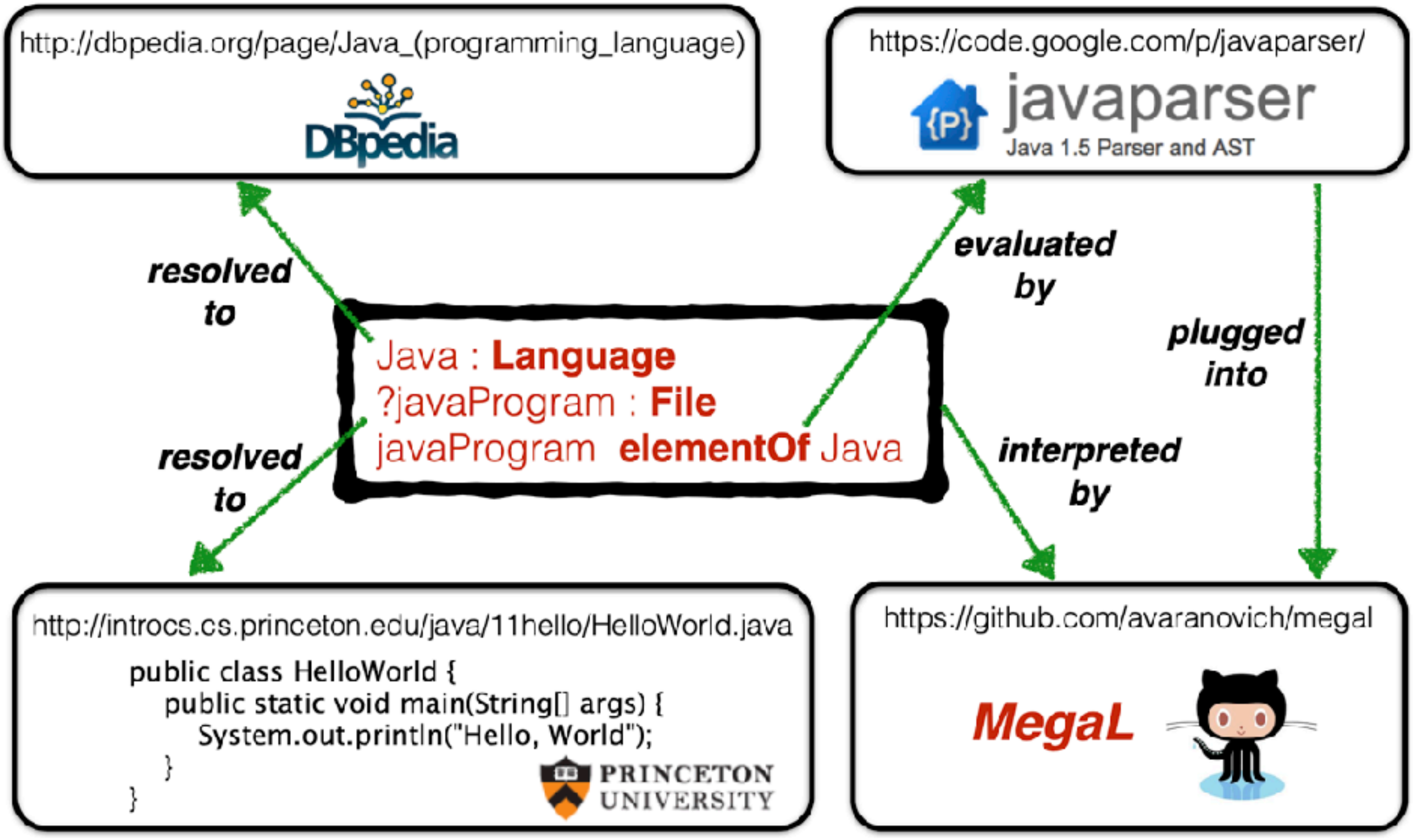
Source: Johannes Härtel, Lukas Härtel, Ralf Lämmel, Andrei Varanovich, Marcel Heinz: **Interconnected Linguistic Architecture**. Art Sci. Eng. Program. 1(1): 3 (2017)

Validation of models of linguistic architecture



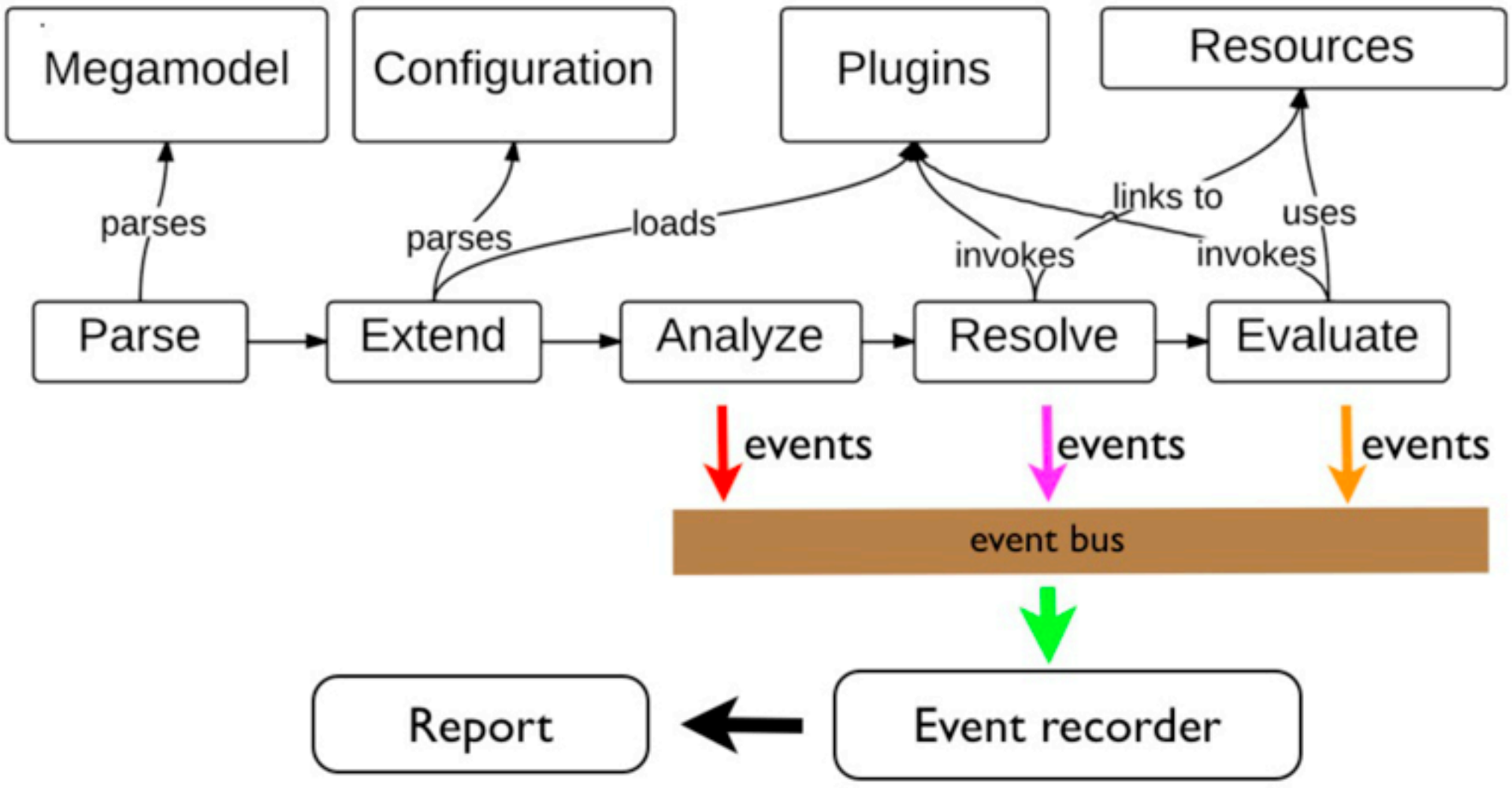
Source: Johannes Härtel, Lukas Härtel, Ralf Lämmel, Andrei Varanovich, Marcel Heinz: **Interconnected Linguistic Architecture**. Art Sci. Eng. Program. 1(1): 3 (2017)

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

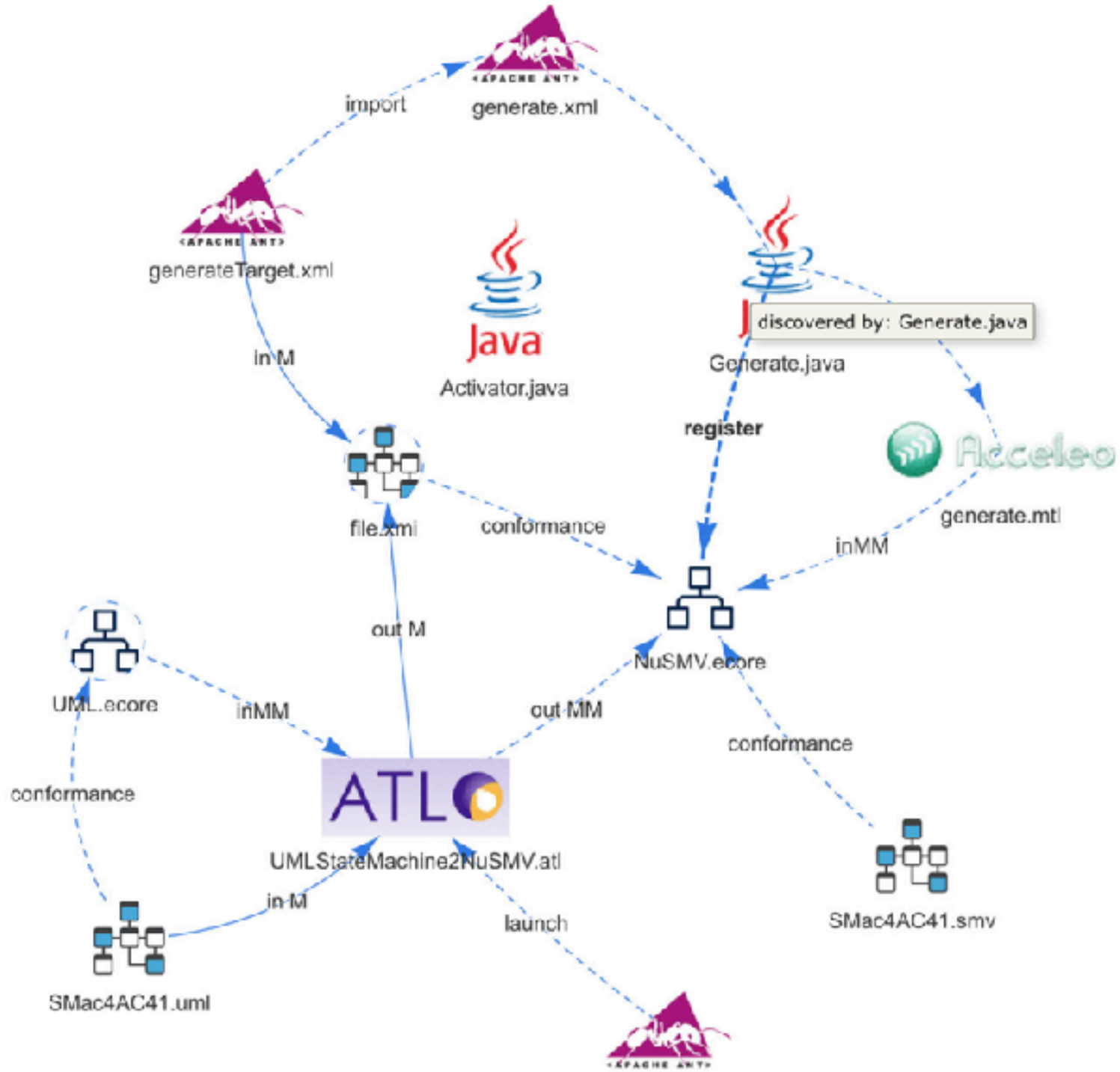
How do we build those megamodels?

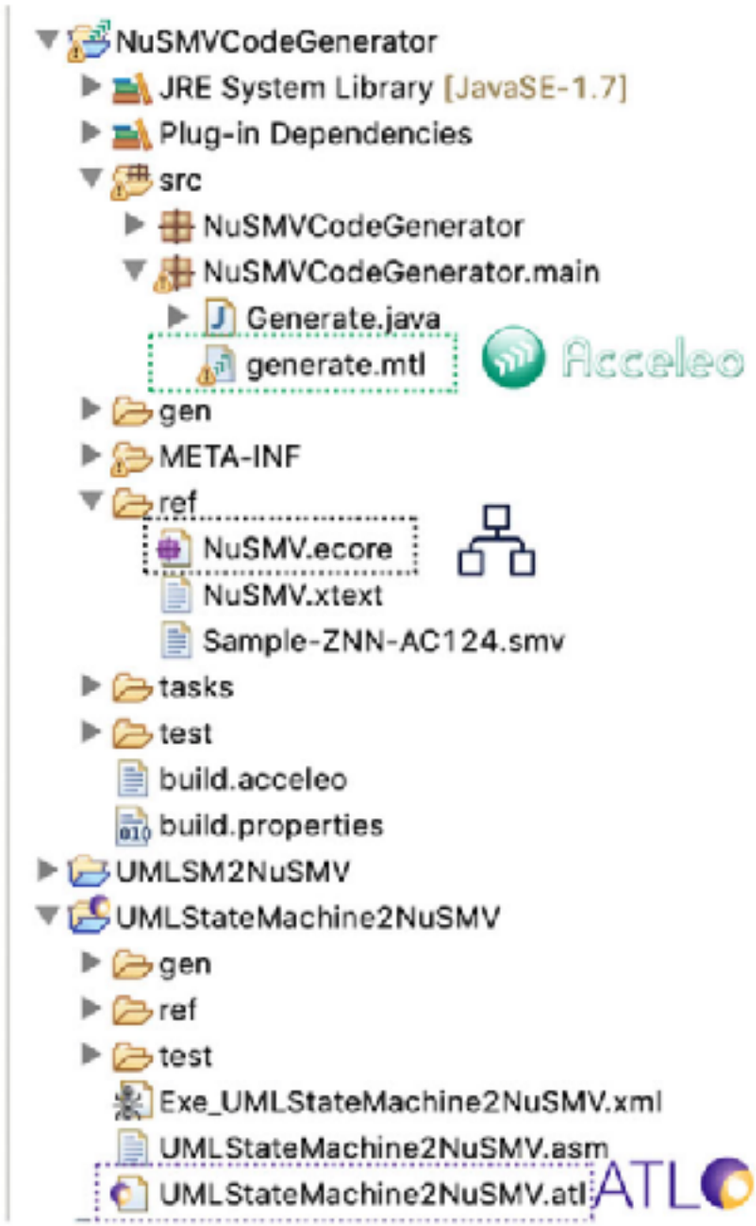
Discovery of entities and relationships

Id	Question	Relevant MegaL constructs
L1	Which languages can be identified?	Type *Language*
L2	Is one language contained in another?	Relationship *subsetOf*
A1	What artifacts participate in the scenario?	Type *Artifact*
A2	What is the language of each artifact?	Relationship *elementOf*
A3	Does an artifact conform to another artifact?	Relationship *conformsTo*
A4	Does an artifact define a language?	Relationship *defines*
F1	Is one artifact derived from another artifact?	Type *Function*
F2	What is domain and range of a function?	Function with domain & range
F3	How is a function applied?	Function application '$f(x) \mapsto y$'
F4	How is a function defined?	Relationship *defines*
R1	Are artifacts closely similar to each other?	Relationship *correspondsTo*
R2	Can a correspondence be structured?	Relationship *partOf*
R3	What causes a correspondence?	Function application '$f(x) \mapsto y$'
C1	Can the entity be described conceptually?	Type *Concept*
C2	Does the entity use the concept?	Relationship *uses*
C3	Does the entity help to use the concept	Relationship *facilitates*

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

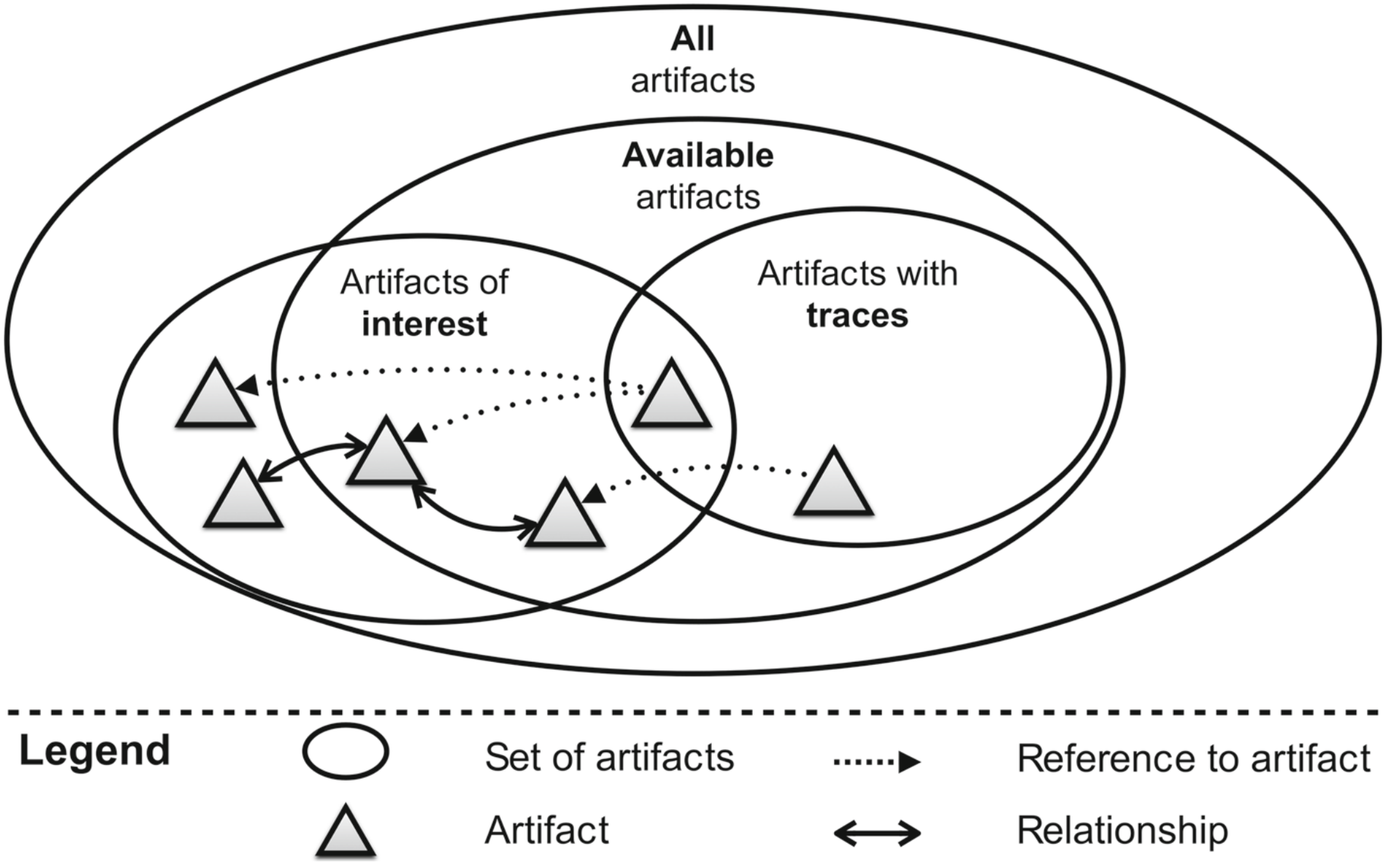
Recovered megamodel of an MDE project





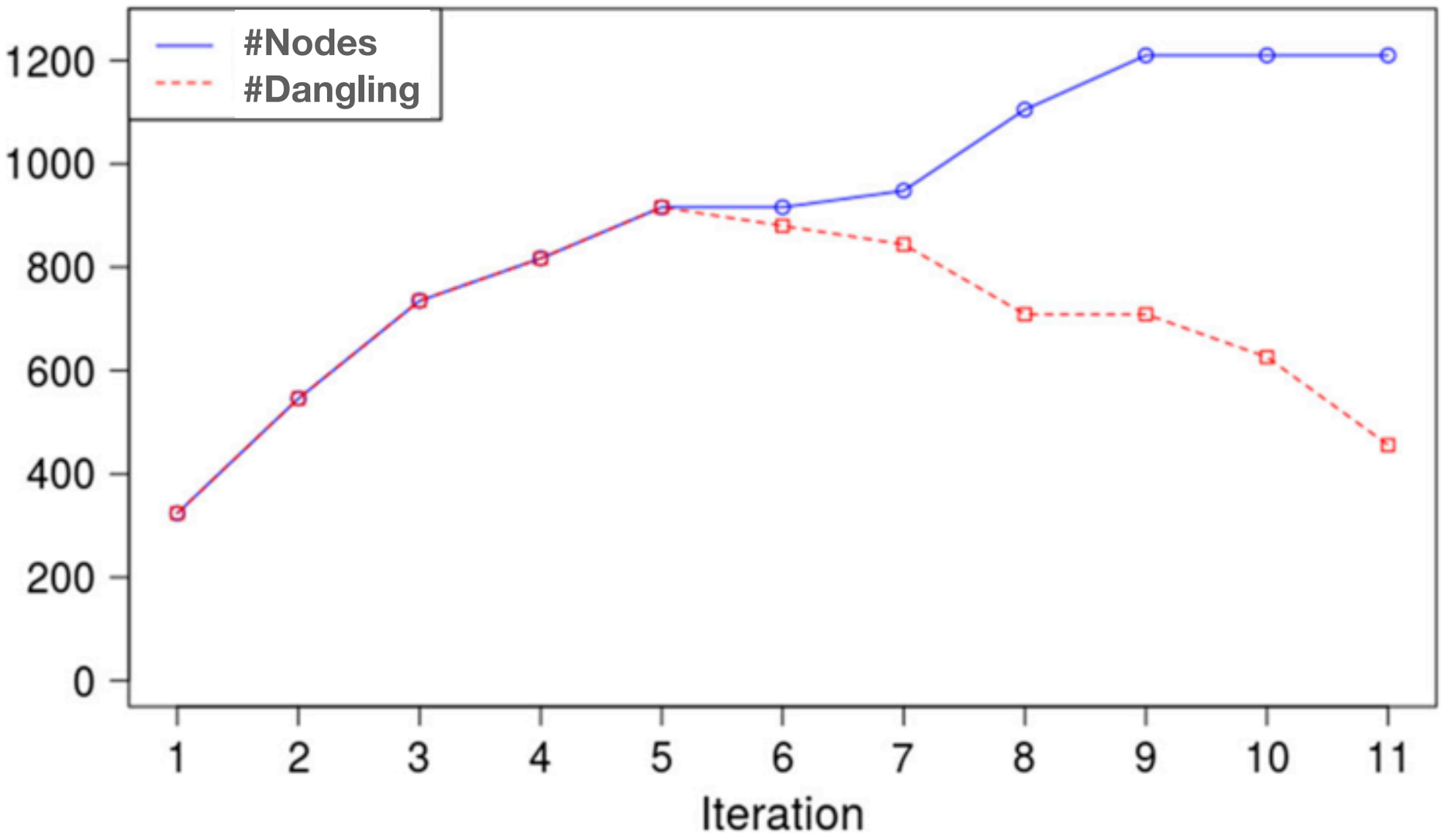
Source: Juri Di Rocco, Davide Di Ruscio, Johannes Härtel, Ludovico Iovino, Ralf Lämmel, Alfonso Pierantonio: **Understanding MDE projects: megamodels to the rescue for architecture recovery**. Softw. Syst. Model. 19(2): 401-423 (2020). See also conference version: Juri Di Rocco, Davide Di Ruscio, Johannes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

Heuristics-based architecture recovery



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

Heuristics-based architecture recovery



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

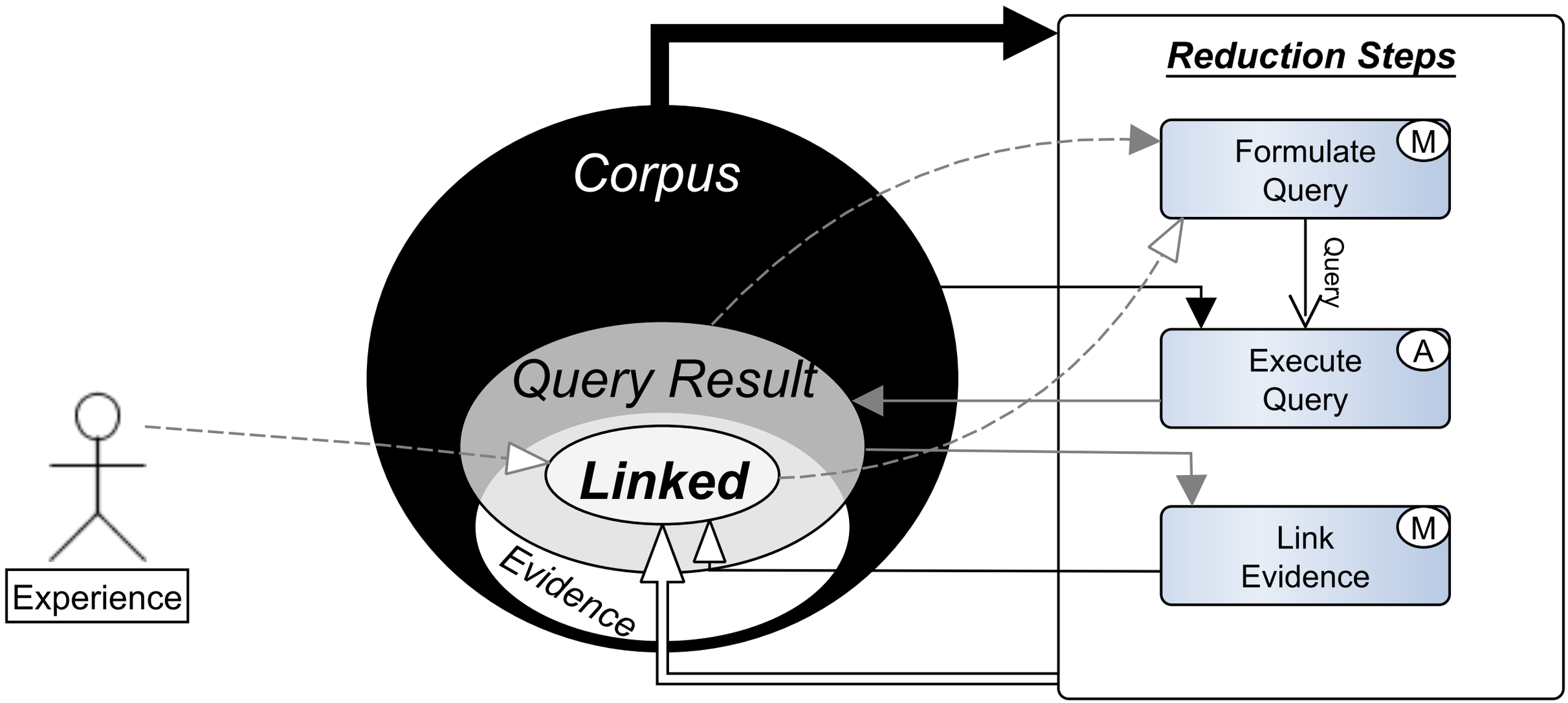
Heuristics-based architecture recovery

Iteration	Applied heuristics	#Nodes	#Edges	#Dangling nodes
1	EH	324	0	324
2	EH, AH	546	0	546
3	EH, AH, KH	735	0	735
4	EH, AH, KH, LH	817	0	817
5	EH, AH, KH, LH, ANH	916	0	916
6	EH, AH, KH, LH, ANH, APH	916	37	880
7	EH, AH, KH, LH, ANH, APH, LTH	948	212	844
8	EH, AH, KH, LH, ANH, APH, LTH, ANATLH	1105	831	709
9	EH, AH, KH, LH, ANH, APH, LTH, ANATLH, JH	1210	831	709
10	EH, AH, KH, LH, ANH, APH, LTH, ANATLH, JH, TOTEMH	1210	1039	626
11	EH, AH, KH, LH, ANH, APH, LTH, ANATLH, JH, TOTEMH, KM3ECOREH	1210	1112	456

EH EcoreHeuristic, *AH* ATLHeuristic, *KH* KM3Heuristic, *LH* LauncherHeuristic, *ANH* ANTHeuristic, *APH* ATLWithPathHeuristic, *LTH* LauncherATLHeuristic, *ANATLH* ANTWithATLHeuristic, *JH* JavaHeuristic, *TOTEMH* ATLWithTOTEMHeuristic, *KM3ECOREH* KM32ECOREHeuristic

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

Query-based construction of linked technology models



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

What’s the ontology behind megamodels?

Table 1: Entity types in relevant papers.

Paper	Artifact	Function	Record	System	Technology	Language	Inf. resource	Fragment	Collection	Trace	Concept	Others
[1]	x	x	x				x					x
[2]	x	x	x		x					x		x
[3]	x			x	x						x	x
[4]					x	x	x				x	x
[5]	x						x	x		x		x
[6]	x		x									
[7]	x	x	x									
[8]	x									x		
[9]	x						x		x			
[10]	**x**	**x**	x		x	x		x	x			
[11]	x	x	x							x		
[12]				x								x
[13]	x	x								x		x
[14]	x	x										

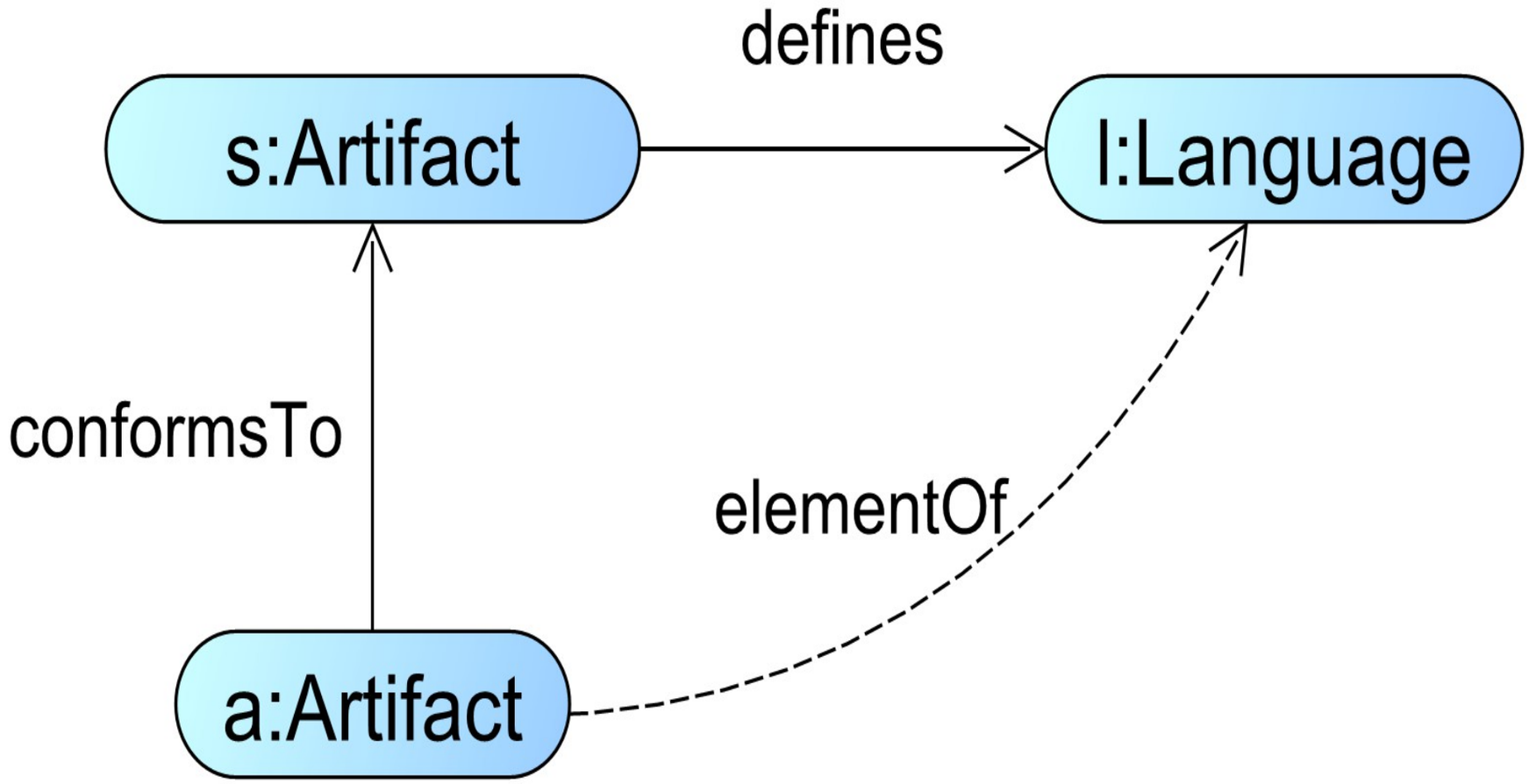
Source: Marcel Heinz, Ralf Lämmel, Andrei Varanovich: **Axioms of Linguistic Architecture**. MODELSWARD 2017: 478-486

Table 2: Relationship types in relevant papers.

Paper	Conformance	Definition	Correspondence	Implementation	Usage	Membership	Typing	Dependency	Abstract rel.	Others
[1]					x					x
[2]	x									
[3]	x	x	x	x	x			x		x
[4]				x	x		x	x		x
[5]				x			x			x
[6]						x	x		x	
[7]										x
[8]									x	
[9]		x							x	
[10]	x	x	x	x		x	x	x	x	
[11]	x	x					x		x	
[12]		x								x
[13]							x			
[14]	x								x	

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

Understanding Membership



- $\mathsf{elementOf}(a, l) \Rightarrow \mathsf{Artifact}(a) \wedge \mathsf{Language}(l)$...
- $\mathsf{elementOf}(a, l) \Leftarrow \exists s.\mathsf{defines}(s, l) \wedge \mathsf{conformsTo}(a, s)$.

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

Understanding Membership

- $\mathsf{Specification}(a) \Rightarrow \mathsf{Artifact}(a)$.
- $\mathsf{Language}(l) \Rightarrow \exists s.\mathsf{Specification}(s) \wedge \mathsf{defines}(s, l)$ ...
- $\mathsf{defines}(a, e) \Rightarrow \mathsf{Artifact}(a) \wedge \mathsf{Entity}(e)$.
- $\mathsf{conformsTo}(a, s) \Rightarrow \mathsf{Artifact}(a) \wedge \mathsf{Artifact}(s)$.
- $\mathsf{conformsTo}(a, s) \Leftarrow (\forall p_a.\mathsf{partOf}(p_a, a) \wedge \exists p_s.\mathsf{partOf}(p_s, s) \wedge \mathsf{conformsTo}(p_a, p_s)) \vee \exists t.\mathsf{defines}(s, t) \wedge \mathsf{elementOf}(a, t)$.

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

Classified entities on Wikipedia/Dbpedia

https://en.wikipedia.org/wiki/MATLAB ① **URL**

MATLAB (*matrix laboratory*) is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran and Python. ② **Summary**

MATLAB

L-shaped membrane logo[1]

MATLAB R2013a running on Windows 8

Developer(s)	MathWorks
Initial release	1984; 34 years ago
Stable release	R2018a / 15 March 2018; 3 months ago
Preview release	None [±]
Written in	C, C++, Java
Operating system	Windows, macOS, and Linux[2]
Platform	IA-32, x86-64
Type	Numerical computing
License	Proprietary commercial software
Website	mathworks.com/products/matlab

MATLAB

Paradigm	multi-paradigm: functional, imperative, procedural, object-oriented, array
Designed by	Cleve Moler
Developer	MathWorks
First appeared	late 1970s
Stable release	9.4 (R2018a) / March 14, 2018; 3 months ago
Preview release	None [±]
Typing discipline	dynamic, weak
Filename extensions	.m
Website	mathworks.com/products/matlab
Influenced by	APL · EISPACK · LINPACK · PL/0 · Speakeasy[3]
Influenced	Julia[4] · Octave[5] · Scilab[6]

MATLAB Programming at Wikibooks

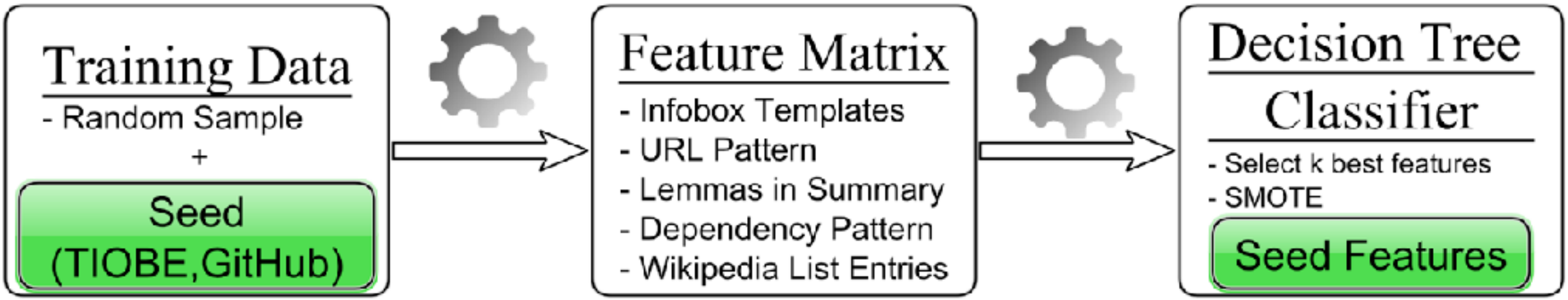
③ **Infoboxes**

④ **Category Graph**

Categories: Image processing software | Array programming languages | C software | Computer algebra system software for Linux | Computer algebra system software for MacOS | Computer algebra system software for Windows | Computer algebra systems | Computer vision software | Cross-platform software | Data mining and machine learning software | Data visualization software | Data-centric programming languages | Dynamically typed programming languages | Econometrics software | High-level programming languages | IRIX software | Linear algebra | Mathematical optimization software | Numerical analysis software for Linux | Numerical analysis software for MacOS | Numerical analysis software for Windows | Numerical linear algebra | Numerical programming languages | Numerical software | Parallel computing | Plotting software | Proprietary commercial software for Linux | Proprietary cross-platform software | Regression and curve fitting software | Software modeling language | Statistical programming languages | Time series software

Source: Marcel Heinz, Ralf Lämmel, Mathieu Acher: **Discovering Indicators for Classifying Wikipedia Articles in a Domain - A Case Study on Software Languages**. SEKE 2019: 541-706

ML approach to Wikipedia-based classification



Source: Marcel Heinz, Ralf Lämmel, Mathieu Acher: **Discovering Indicators for Classifying Wikipedia Articles in a Domain - A Case Study on Software Languages**. SEKE 2019: 541-706

I wish megamodeling was here.

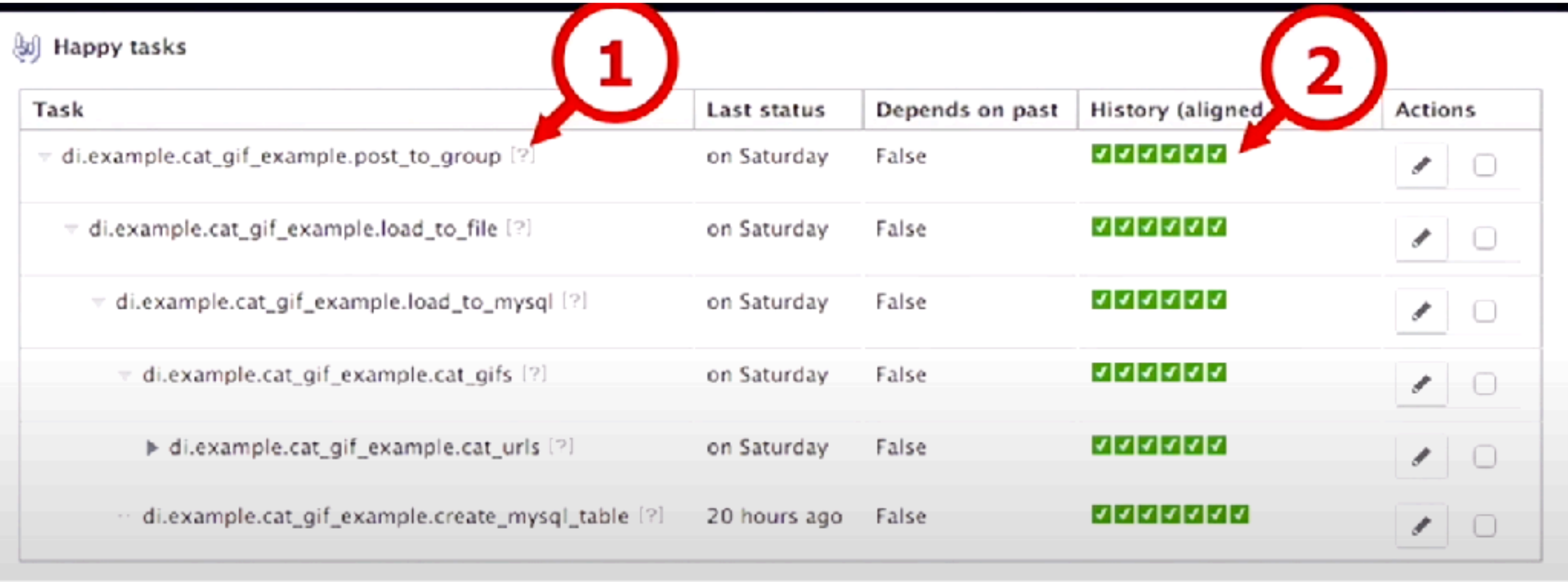
(At Facebook or such.)

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.

Data pipeline management (at Facebook)



1 — Nesting based on dependencies
2 — Historical (per-day) executions

Source: Mike Starr, Dataswarm, Youtube video

Fragmented DevOps

We can’t …

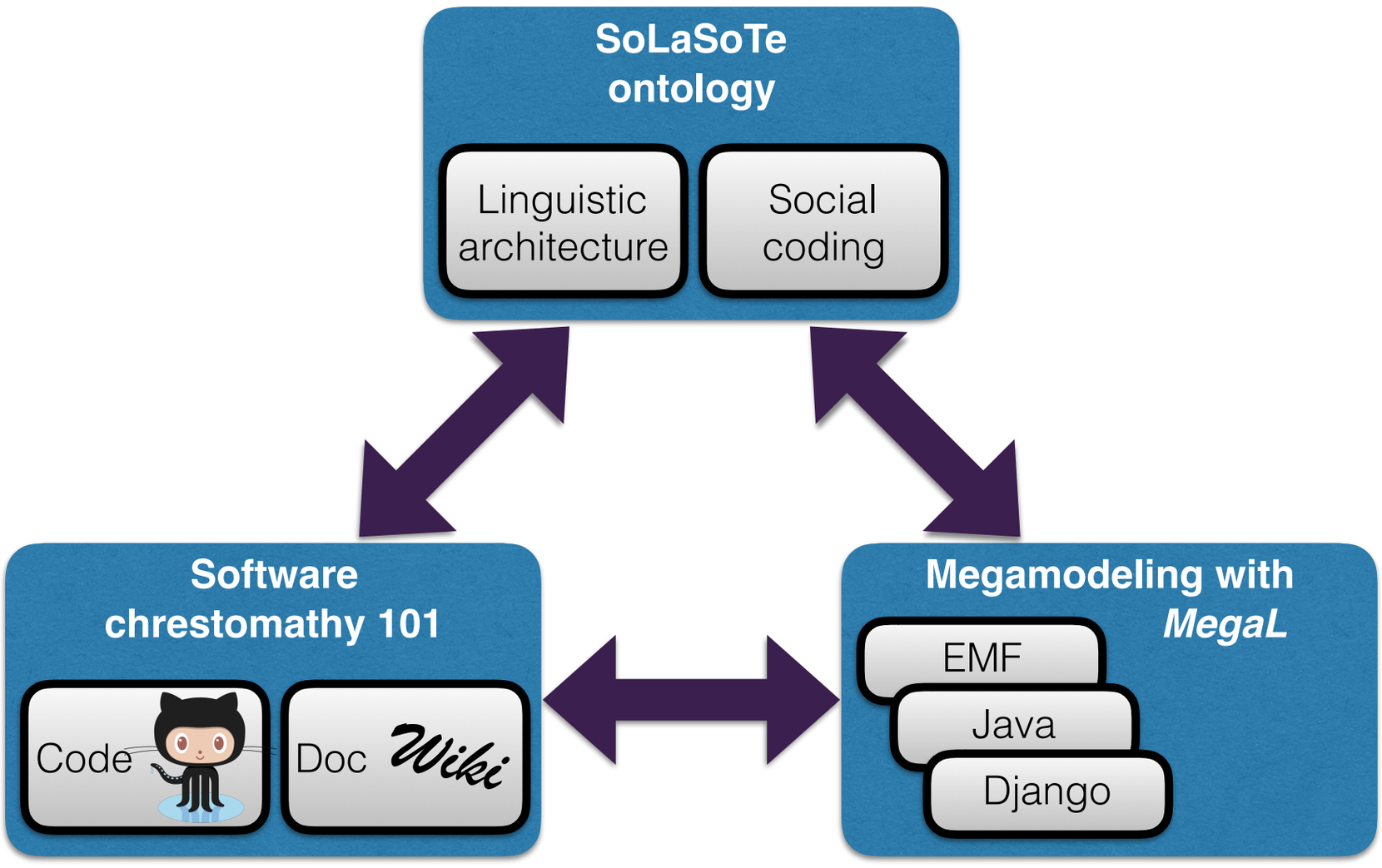
- Recover from blackout;
- Abstract from implementation layers;;
- Generate automatic documentation;
- Mine workflows from infra logging;
- Use models other than through UI/CLI;
- Handle cross-repo scope;
- …

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

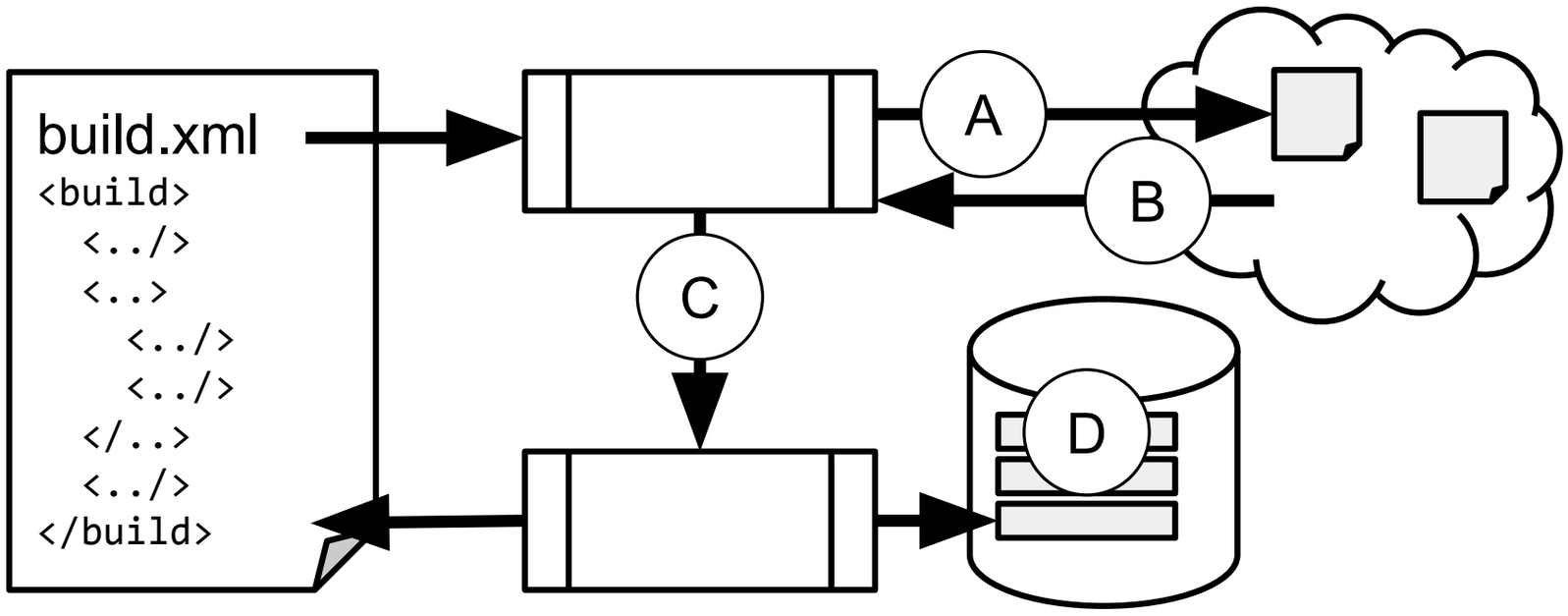
Support deep relationships

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

Thanks!
Let’s discuss.

Papers on megamodeling

- 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

- 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

- Marcel Heinz, Ralf Lämmel, Mathieu Acher: **Discovering Indicators for Classifying Wikipedia Articles in a Domain - A Case Study on Software Languages**. SEKE 2019: 541-706

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

- Ralf Lämmel: **Relationship Maintenance in Software Language Repositories**. Art Sci. Eng. Program. 1(1): 4 (2017)

- Juri Di Rocco, Davide Di Ruscio, Marcel Heinz, Ludovico Iovino, Ralf Lämmel, Alfonso Pierantonio: **Consistency Recovery in Interactive Modeling**. MODELS (Satellite Events) 2017: 116-122

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

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

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

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

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