Just to be clear — no dogs today!
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.

Time for a catwalk to organize the space of megamodeling.

Megamodelling is a niche, if you go by explicit mentioning of the paradigm in software engineering.
Megamodelling 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 megamodelling at Facebook in software development.
Let's discuss how megamodelling could be generalized and used more profoundly in software engineering.
To this end, we need to continue working on these premises:

i) Megamodelling 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 megamodelling DSLs.
The term ‘megamodel’ lacks clarity in definition or demarcation.
A megamodel for EMF code generation

A megamodel for compiler bootstrapping
A megamodel for parsing in a broad sense

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)

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

ANTLR-grammar elementOf ANTLR-Notation // The grammar is given in ANTLR notation
ANTLR-grammar defines aLanguage // The grammar defines some language
ANTLR-parser elementOf Java // Java is used for the generated parser
ANTLR-generator(aGrammar) → aParser // Generate parser from grammar
ANTLR-input elementOf aLanguage // Wanted! An element of the language
ANTLR-input conformsTo aGrammar // Conform also to the grammar

ANTLR-generatorApp1 : **FunctionApplication**
ANTLR-generatorApp1 elementOf ANTLR-generator
ANTLR-generatorApp1 inputOf ANTLR-generatorApp1
ANTLR-generatorApp1 outputOf ANTLR-generatorApp1

Source: Ralf Lämmel, Andrei Varanovich: **Interpretation of Linguistic Architecture**. ECMFA 2014: 67-82
A megamodel for MT with ATL/Acceleo

Megamodels for two basic BX patterns

In the first (more basic) BX pattern, \textit{get} maps a source to a view; \textit{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, \textit{put} has been replaced by a decomposition of differencing and change propagation.

Source: Ralf Lämmel: \textbf{Coupled software transformations revisited}. SLE 2016: 239-252
Megamodels for two basic BX patterns

LAL megamodel $bx.state$

```plaintext
reuse cx.mapping [ mapping $\mapsto$ get ]
function get : $L_1 \rightarrow L_2$
function put : $L_1 \times L_2 \rightarrow L_1$
axiom GetPut { $\forall s \in L_1$.
    put(s, get(s)) = s }
axiom PutGet { $\forall s_1, s_2 \in L_1$. $\forall v \in L_2$.
    put(s_1, v) = s_2 $\Rightarrow$ get(s_2) = v }
```

LAL megamodel $bx.delta$

```plaintext
reuse $bx.state$
reuse differencing [ $L \mapsto L_2$, Any $\mapsto$ Any_2 ]
function propagate : $L_1 \times$ DiffL $\rightarrow L_1$
axiom { $\forall s_1, s_2 \in L_1$. $\forall v_1, v_2 \in L_2$. $\forall$ delta $\in$ DiffL.
    get(s_1) = v_1
    \land$ diff(v_1, v_2) = delta
    \land propagate(s_1, delta) = s_2 $\Rightarrow$
    put(s_1, v_2) = s_2 $\land$ get(s_2) = v_2 }
```

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#

... XML-data binding in C#

Linguistic architecture in a software development context

Validation of models of linguistic architecture

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](https://example.com). ECMFA 2014: 67-82
How do we build those megamodels?
# Discovery of entities and relationships

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

Recovered megamodel of an MDE project

Artifacts in a MDE project

In the case of ATL-based model transformation, artifacts of interest are clearly the ATL transformations themselves, but also source and target models for transformations as well as metamodels for conformance.

Artifacts with traces are those (available) artifacts (of interest or not) in which we may locate traces to artifacts (mainly references). Subject to a classification of the artifacts with traces, these artifacts may be interpreted as (encoding) relationships between artifacts. Finally, the recovery approach may also involve virtual artifacts; but in this sense that the artifacts are not really thought to be part of the repository (available or not), but they are computed, much in the sense of transient artifacts, but only for the purpose of discovering artifact types and relationships.

The overall assumption is that we may identify artifacts of interest by examining algorithmically available artifacts and we may identify relationships between artifacts by examining, again, algorithmically available artifacts on the grounds of technology-specific patterns for traces; we may introduce (in rare cases) virtual artifacts along the way.

4.2 Relationships to be recovered

Figure 4 identifies ‘abstract’ artifacts of interest with relationships for the running example of ATL and Acceleo. In particular, in Fig. 4 at the source and target models, the corresponding metamodels (MMs), the actual ATL model transformation (MT), and the application thereof. In Fig. 4b, the source model is the input, conforming to the source metamodel, and the Acceleo module (M2T) is executed to get the output, which can be any textual format file or code, depending on the target platform. An Acceleo module is usually called by a corresponding main Java file containing references to the module specification.

We also show relationships between these artifacts that need to be recovered. Relationships between artifacts, e.g., conformance and transformation application in the example, can be identified in different ways:

4.2.1 Trace-based identification

Based on the type of referring artifact (e.g., an ANT file), based also on the details of reference (e.g., the argument position...
Heuristics-based architecture recovery

Heuristics-based architecture recovery

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

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

Query-based construction of linked technology models

Query-based construction of linked technology models involves several manual and automated steps. First, a promising corpus that contains evidence needs to be selected. Then, a query is developed. Queries are formulated and executed to reduce the search scope within a selected corpus. Here, a query is any tool-based reduction of the corpus to candidates for linked evidence, for example, by searching for an artifact type’s name using `grep`. Queries can be formulated based on patterns identified in previously linked evidence and returned query results. The query results are then manually inspected to confirm concise textual explanations and idiomatic code examples and link them as evidence. The detection of evidence is a continuous process. Textual explanations can be helpful to identify code examples and vice versa. Hence, we do not intend to enforce any order in which the different corpora are processed. They can be processed in an interleaving manner.

Figure 2 – Manually ('M') or automatically ('A') executed steps reduce a corpus to linked evidence. Resources are related to steps by input and output edges, whose color hints at whether it is unknown (black), query-related (gray), or linked (white).

The degree of manual effort for executing the methodology depends on the given experience. If concise textual explanations and idiomatic code examples can be linked without any querying effort, the effort is at the minimum. If no query can be formulated from the beginning, the effort is at the maximum. Not every resource that can serve as evidence for an increment may be returned by a developed query, especially, within a restricted time window; and not every query result can serve as evidence. We focus on what is in between: Concise textual explanations and idiomatic code examples that are selected from systematically refined query results and then linked. To assure the reproducibility of evidence, queries are shared as well. In the example-driven evaluation, we emphasize reproducibility by providing reduction step protocols in which we record input and output of each executed reduction step. This way, a reduction step protocol instantiates the reduction procedure from Figure 2.

When the construction of a technology model is based on an unvaried or non-representative corpus of resources, it is prone to errors, in particular, misconception (See Section 4.2). Linking evidence helps in raising the confidence in an interconnected technology model. By developing queries, varied and representative evidence can be linked. Sharing the queries makes the recovery of evidence reproducible and facilitates linking evidence in any additional corpus. Thus, we assume that systematically interconnecting a varied and representative set of resources is more robust and may help with constructing an interconnected technology model more quickly, when compared to a less systematic approach.

What’s the ontology behind megamodels?
### Table 1: Entity types in relevant papers.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Artifact</th>
<th>Function</th>
<th>Record</th>
<th>System</th>
<th>Technology</th>
<th>Language</th>
<th>Inf. resource</th>
<th>Fragment</th>
<th>Collection</th>
<th>Trace</th>
<th>Concept</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[2]</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[3]</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[4]</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[5]</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[6]</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[7]</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[8]</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[9]</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[10]</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[11]</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[12]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[13]</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[14]</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Source: Marcel Heinz, Ralf Lämmel, Andrei Varanovich: **Axioms of Linguistic Architecture**. MODELSWARD 2017: 478-486
## Table 2: Relationship types in relevant papers.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Conformance</th>
<th>Definition</th>
<th>Correspondence</th>
<th>Implementation</th>
<th>Usage</th>
<th>Membership</th>
<th>Typing</th>
<th>Dependency</th>
<th>Abstract rel.</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[2]</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[6]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[7]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[8]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[9]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[10]</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[11]</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[12]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[13]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[14]</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Understanding Membership

- \(\text{elementOf}(a, l) \Rightarrow \text{Artifact}(a) \land \text{Language}(l)\)
- \(\text{elementOf}(a, l) \iff \exists s. \text{defines}(s, l) \land \text{conformsTo}(a, s)\)

Understanding Membership

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

Source: Marcel Heinz, Ralf Lämmel, Andrei Varanovich: [Axioms of Linguistic Architecture](https://doi.org/10.1007/978-3-319-63509-0_32). MODELSWARD 2017: 478-486
Classified entities on Wikipedia/Dbpedia

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 megamodelling context

Support deep relationships

<table>
<thead>
<tr>
<th>xsdFiles</th>
<th>javaFiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>/xs:schema/xs:complexType</td>
<td>org/softlang/company/xjc/Employee.java</td>
</tr>
<tr>
<td>/xs:schema/xs:element#0</td>
<td>org/softlang/company/xjc/Company.java</td>
</tr>
<tr>
<td>/xs:schema/xs:element#1</td>
<td>org/softlang/company/xjc/Department.java</td>
</tr>
<tr>
<td>xmlFile</td>
<td>objectGraph</td>
</tr>
<tr>
<td>company/department#0</td>
<td>org.softlang.company.xjc.Department@5fd1a6aa</td>
</tr>
<tr>
<td>employee#0</td>
<td>org.softlang.company.xjc.Employee@1a56a6c6</td>
</tr>
<tr>
<td>address:Utrecht</td>
<td>Utrecht</td>
</tr>
<tr>
<td>name:Erik</td>
<td>Erik</td>
</tr>
<tr>
<td>salary:12345</td>
<td>12345.0</td>
</tr>
<tr>
<td>employee#1</td>
<td>org/softlang/company/xjc.Employee@748e432b</td>
</tr>
</tbody>
</table>

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.

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.

## Embrace principles of interconnection

<table>
<thead>
<tr>
<th></th>
<th>Traceability links</th>
<th>Artifact binding</th>
<th>Model inference</th>
<th>Pluggable analyses</th>
<th>Explorable connections</th>
<th>Modularized models</th>
<th>Semantic annotations</th>
<th>Transient artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>[30]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[7]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[25]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[15]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[11]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[17]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[12]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>([10])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

\[
\text{Label} = "File \ with \ language", \ \text{Operator} = "Addition"
\]

+ ?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:

\[
\text{Label} = "A \ Java \ file", \ \text{Operator} = "Instantiation"
\]

+ 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 megamodelling


• 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


• 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