

Elements of SLEBOK

(Software Language Engineering (SLE)
Body of Knowledge (BOK))

Ralf Lämmel
Software Languages Team
University of Koblenz-Landau

<http://www.softlang.org/>

Let's just pretend this is not a

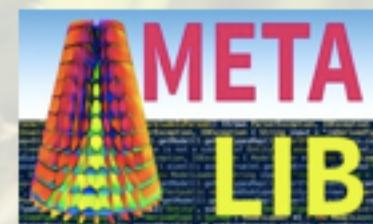
TABLE OF CONTENTS

- **What's a software language? (Java, UML, CSS, etc..)**
- **What's software language engineering (SLE)?**
- **What's a body of knowledge (BOK)?**
- **Why do we need a BOK for SLE?**
- **How do we go about the SLEBOK?**
- **Let's do it. <--- loads of stuff!**

(<https://pixabay.com/en/kittens-cat-cats-puppy-rush-555822/>)

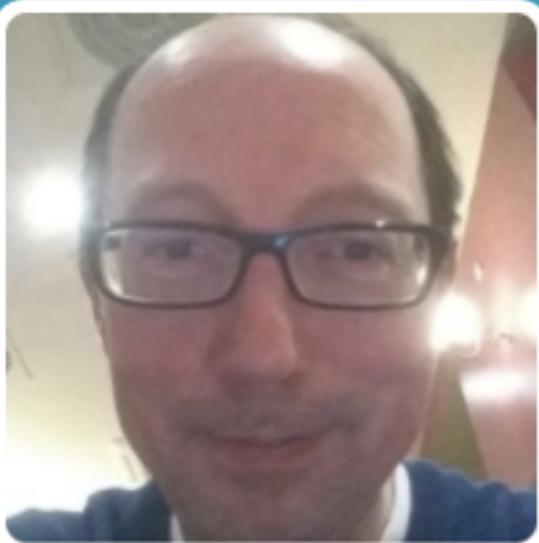
Highlights of this presentation

1. Language definition (syntax, semantics, types)
2. Metaprogramming
3. Metamodeling
4. Feature modeling
5. Megamodeling
6. Linked Open Data
7. Software chrestomathies
8. Software language repositories



[<https://pixabay.com/en/cat-kitten-tree-green-summer-1647775/>]

Contribute to the BOK!



Ralf Lämmel

@reallynotabba

My profile? Data mining should figure it out. Enough tweets and other traces at avail. I am pro-everything, except hate, bigotry, and PHP.

Totally obfuscated

[professor-fish.blogspot.de](#)

Joined March 2009

Twitter name: @reallynotabba

Hashtag: #slebok



[Tweet](#)



Course "Elements of Software Language Engineering Body of Knowledge" starting now #sle #bok #slebok #laquila #gssi <http://softlang.uni-koblenz.de/gssi17.pdf>

Book on selected SLEBOK foundations

<http://www.softlang.org/book>

Ralf Lämmel:

Software languages

Syntax, semantics, and metaprogramming

To appear, Springer, 2017



What's a software language?

“A software language is an ‘artificial language’ used by software engineers and other stakeholders in software development.“ [softlangbook]

Programming languages used in *softlangbook*

Language	Explanation
<i>Haskell</i> ¹	The functional programming language Haskell
<i>Java</i> ²	The Java programming language
<i>Python</i> ³	The dynamic programming language Python
<i>Prolog</i> ⁴	The logic programming language Prolog (specifically SWI-Prolog)

¹ Haskell language: <https://www.haskell.org/>

² Java language: [https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))

³ Python language: <https://www.python.org/>

⁴ Prolog language: <https://en.wikipedia.org/wiki/Prolog>

Other software languages used in *softlangbook*

Language	Explanation
<i>ANTLR</i> ⁵	The grammar notation of the ANTLR technology
<i>JSON</i> ⁶	The JavaScript Object Notation
<i>JSON Schema</i> ⁷	The JSON Schema language
<i>XML</i> ⁸	Extensible Markup Language
<i>XSD</i> ⁹	XML Schema Definition

⁵ ANTLR language: <http://www.antlr.org/>

⁶ JSON language: <https://en.wikipedia.org/wiki/JSON>

⁷ JSON Schema language: <http://json-schema.org/>

⁸ XML language: <https://en.wikipedia.org/wiki/XML>

⁹ XSD language: [https://en.wikipedia.org/wiki/XML_Schema_\(W3C\)](https://en.wikipedia.org/wiki/XML_Schema_(W3C))

Software languages mentioned in softlangbook

Language	Explanation
<i>Alloy</i> ¹⁰	The Alloy specification language
<i>CIL</i> ¹¹	Bytecode of .NET's CLR
<i>Common Log Format</i> ¹²	The NCSA Common log format
<i>DocBook</i> ¹³	The DocBook semantic markup language for documentation
<i>FOAF</i> ¹⁴	The Friend of a friend ontology
<i>INI file</i> ¹⁵	The INI file format
<i>Java bytecode</i> ¹⁶	Bytecode of the JVM
<i>make</i> ¹⁷	The make tool and its language
<i>OWL</i> ¹⁸	Web Ontology Language
<i>QTFF</i> ¹⁹	QuickTime File Format
<i>RDF</i> ²⁰	Resource Description Framework
<i>RDFS</i> ²¹	RDF Schema
<i>Scala</i> ²²	The functional OO programming language Scala
<i>Smalltalk</i> ²³	The OO, reflective programming language Smalltalk
<i>Sparql</i> ²⁴	SPARQL Protocol and RDF Query Language
<i>UML</i> ²⁵	The Unified Modeling Language
<i>XPath</i> ²⁶	The XML Path Language for querying
<i>XSLT</i> ²⁷	Extensible Stylesheet Language Transformations

¹⁰ Alloy language. <http://www.csail.mit.edu/alloy>

Software languages fabricated in softlangbook

BAL	Basic Assembly Language
BFPL	Basic Functional Programming Language
BGL	Basic Grammar Language
BIPL	Basic Imperative Programming Language
BML	Basic Machine Language
BSL	Basic Signature Language
BSTL	Basic Signature Transformation Language
BTL	Basic TAPL Language
BNL	Binary Number Language
EL	Expression Language
EFPL	Extended Functional Programming Language
EGL	Extended Grammar Language
E IPL	Extended Imperative Programming Language
ESL	Extended Signature Language
FSML	Finite State Machine Language
GBL	Graph-based Buddy Language
MML	MetaModeling Language
TBL	Tree-based Buddy Language
TLL	Typed Lambda Language
ULL	Untyped Lambda Language

Classification by nature of language elements

[softlangbook]

<i>Purpose (language)</i>	<i>Purpose (element)</i>	<i>Classifier</i>	<i>Example</i>
Programming	Program	Programming language	Java
Querying	Query	Query language	XPath
Transformation	Transformation	Transformation language	XSLT
Modeling	Model	Modeling language	UML
Specification	Specification	Specification language	Alloy
Data representation	Data	Data format	QTFF (<i>QuickTime file format</i>)
Documentation	Documentation	Documentation language	DocBook
Configuration	Configuration	Configuration language	INI file
Logging	Log	Log format	Common Log Format
...

General purpose vs. domain-specific (programming) language

Distinguishing characteristics [softlangbook]

Domain Only DSLs have a relatively small and well-defined domain.

Language size GPLs are large. DSLs are typically small.

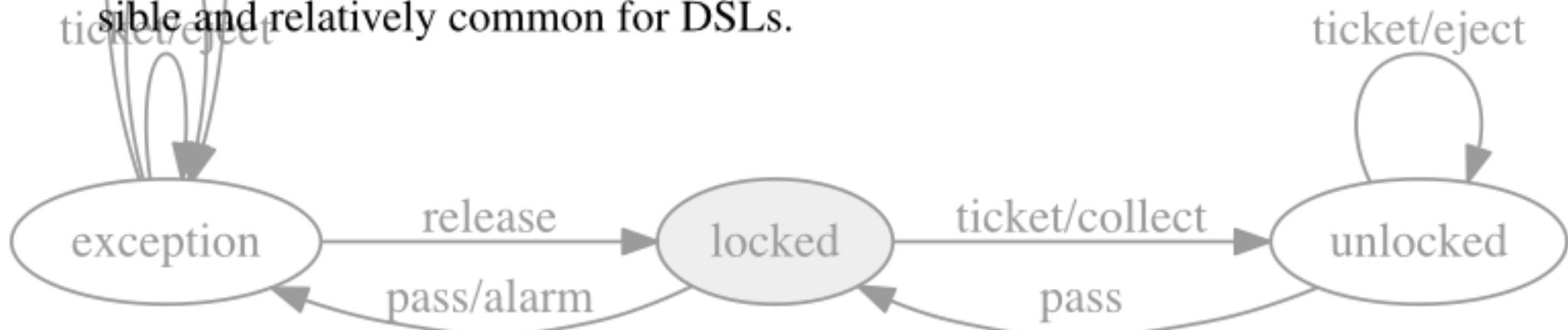
Turing completeness GPLs are whereas DSLs may not be Turing complete.

Lifespan GPLs live for years to decades. DSLs may live for months only.

Designed by ^{mute} GPLs are designed by gurus or committees. DSLs are designed by a few software engineers and domain experts.

Evolution GPLs evolve slowly. The evolution of DSLs is fast-paced.

Deprecation/incompatible changes This is almost impossible for GPLs; it is feasible and relatively common for DSLs.



Classification by ...

[softlangbook]

- **Representation**

- String language
- Tree language
- Graph language

- **Notation**

- Textual language
- Markup language
- Visual language

Classification by ...

[softlangbook]

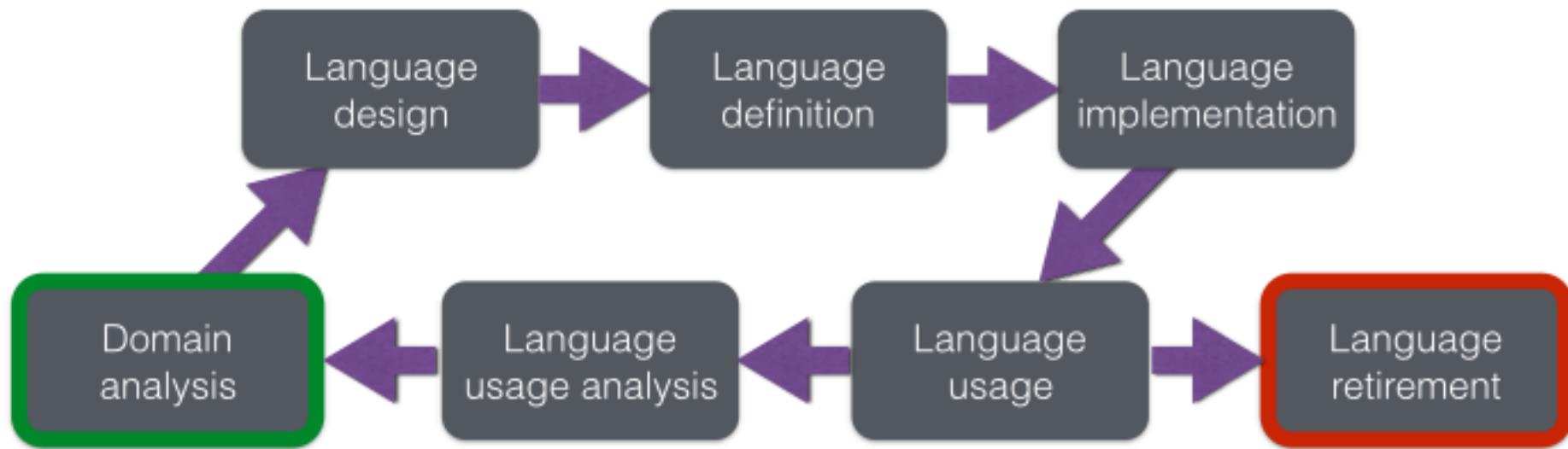
- **Degree of declarativeness**
 - **Non-declarative language**
 - **Declarative language**
 - **Rule-based language**
 - **Constraint-based language**

What's ***Software Language Engineering*** **(SLE)?**

“Software language engineering is the application of systematic, disciplined, and quantifiable approaches to the development (design, implementation, testing, deployment), use, and maintenance (evolution, recovery, and retirement) of these languages.” [<http://www.sleconf.org/2012/>]

The life cyle of a software language

[softlangbook]



Aspects of language definition

// Statements

```
symbol skip : → stmt ;
symbol assign : string × expr → stmt ;
symbol seq : stmt × stmt → stmt ;
symbol if : expr × stmt × stmt → stmt ;
symbol while : expr × stmt → stmt ;
```

// Expressions

```
symbol intconst : integer → expr ;
symbol var : string → expr ;
symbol unary : uop × expr → expr ;
symbol binary : bop × expr × expr → expr ;
```

$$m \vdash \text{skip} \rightarrow m$$

$$m \vdash e \rightarrow v$$

$$\frac{}{m \vdash \text{assign}(x, e) \rightarrow m[x \mapsto v]}$$

$$\frac{m_0 \vdash s_1 \rightarrow m_1 \quad m_1 \vdash s_2 \rightarrow m_2}{m_0 \vdash \text{seq}(s_1, s_2) \rightarrow m_2}$$

$$m \vdash e_0 \rightarrow \text{true} \quad m \vdash s_1 \rightarrow m'$$

$$\frac{m \vdash e_0 \rightarrow \text{true} \quad m \vdash s_1 \rightarrow m'}{m \vdash \text{if}(e_0, s_1, s_2) \rightarrow m'}$$

- Syntax
- Types
- Semantics
- Pragmatics

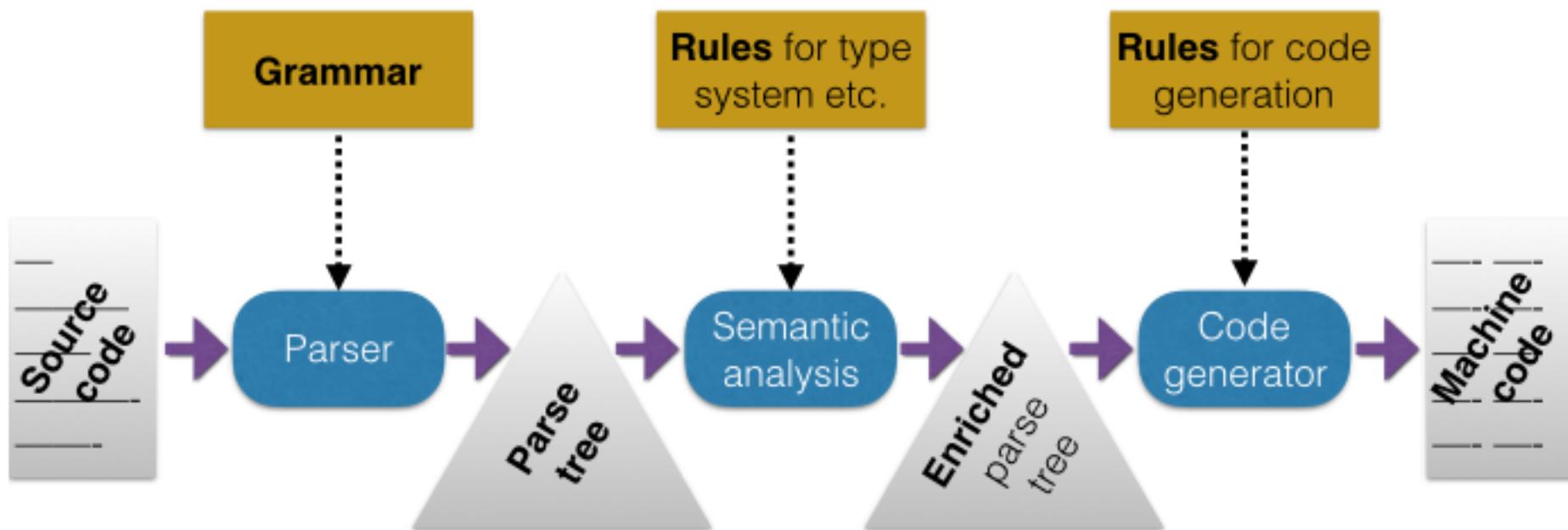
Aspects of language implementation

- Interpretation
- Type checking
- Compilation
- Formatting
- IDE
- ...

-- *Execution of statements*

```
execute :: Stmt → Store → Store
execute Skip m = m
execute (Assign x e) m = insert x (evaluate e m) m
execute (Seq s1 s2) m = execute s2 (execute s1 m)
execute (If e s1 s2) m = execute (if b then s1 else s2) m where Right b = evaluate e m
execute (While e s) m = execute (If e (Seq s (While e s)) Skip) m
```

Data flow in a compiler



Language processors

- Parser (text-to-model)
- Unparser (formatter, pretty printing, model-to-text ...)
- Preprocessor
- Software transformation (model-to-model transformation)
 - Exogenous transformation
 - Endogenous transformation
 - Horizontal transformation
 - Vertical transformation
- Software analysis or analyzer
 - Termination analysis
 - Performance analysis
 - Metrics analysis
 - Vocabulary analysis
 - Bug analysis
 - Usage analysis
- Software translator
- Software generator

(<https://pixabay.com/en/kittens-cat-cai-puppy-rush-555822/>)

What's a body of knowledge (BOK)?

A body of knowledge (BOK) is the set of concepts, terms, activities, etc. that make up a domain.

For comparison: SWEBOK

The Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) describes generally accepted knowledge about software engineering. Its 15 knowledge areas (KAs) summarize basic concepts and include a reference list pointing to more detailed information.

[<https://www.computer.org/web/swebok>]

Knowledge areas

1. SW requirements
2. SW design
3. SW construction
4. SW testing
5. SW maintenance
6. SW configuration management
7. SE management
8. SE process
9. SE models and methods
10. SW quality
11. SE professional practice
12. SE economics
- 13. Computing foundations**
14. Mathematical foundations
15. Engineering foundations

13. Computing foundations

...	
3. Programming Fundamentals	13-5
3.1. <i>The Programming Process</i>	13-5
3.2. <i>Programming Paradigms</i>	13-5
4. Programming Language Basics	13-6
4.1. <i>Programming Language Overview</i>	13-6
4.2. <i>Syntax and Semantics of Programming Languages</i>	13-6
4.3. <i>Low-Level Programming Languages</i>	13-7
4.4. <i>High-Level Programming Languages</i>	13-7
4.5. <i>Declarative vs. Imperative Programming Languages</i>	13-7
5. Debugging Tools and Techniques	13-8
5.1. <i>Types of Errors</i>	13-8
5.2. <i>Debugging Techniques</i>	13-8
5.3. <i>Debugging Tools</i>	13-8
...	
10. Compiler Basics	13-15
10.1. <i>Compiler/Interpreter Overview</i>	13-15
10.2. <i>Interpretation and Compilation</i>	13-15
10.3. <i>The Compilation Process</i>	13-15
...	

**SLE
relevance**

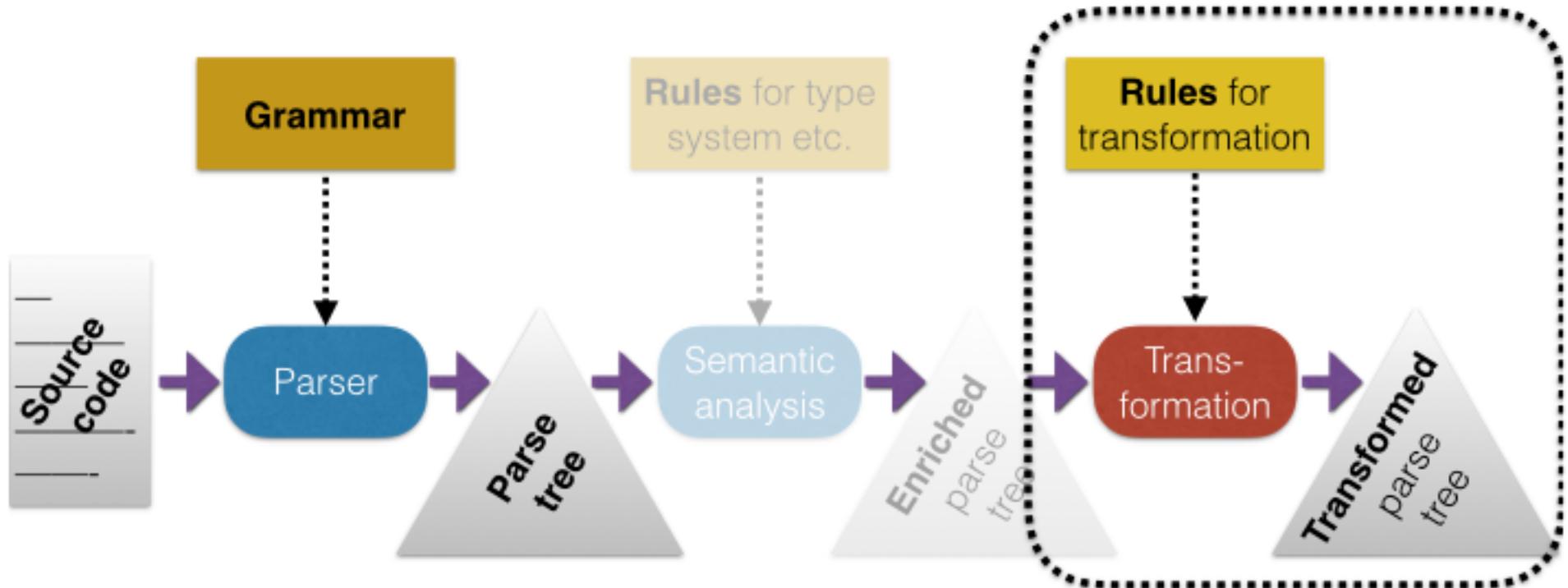
Why do we need a BOK for SLE? (... and what is it anyway?)

The Software Language Engineering (SLE) BOK (SLEBOK) concerns software languages, their definition, implementation, usage, evolution, etc.

Languages are everywhere in SE!

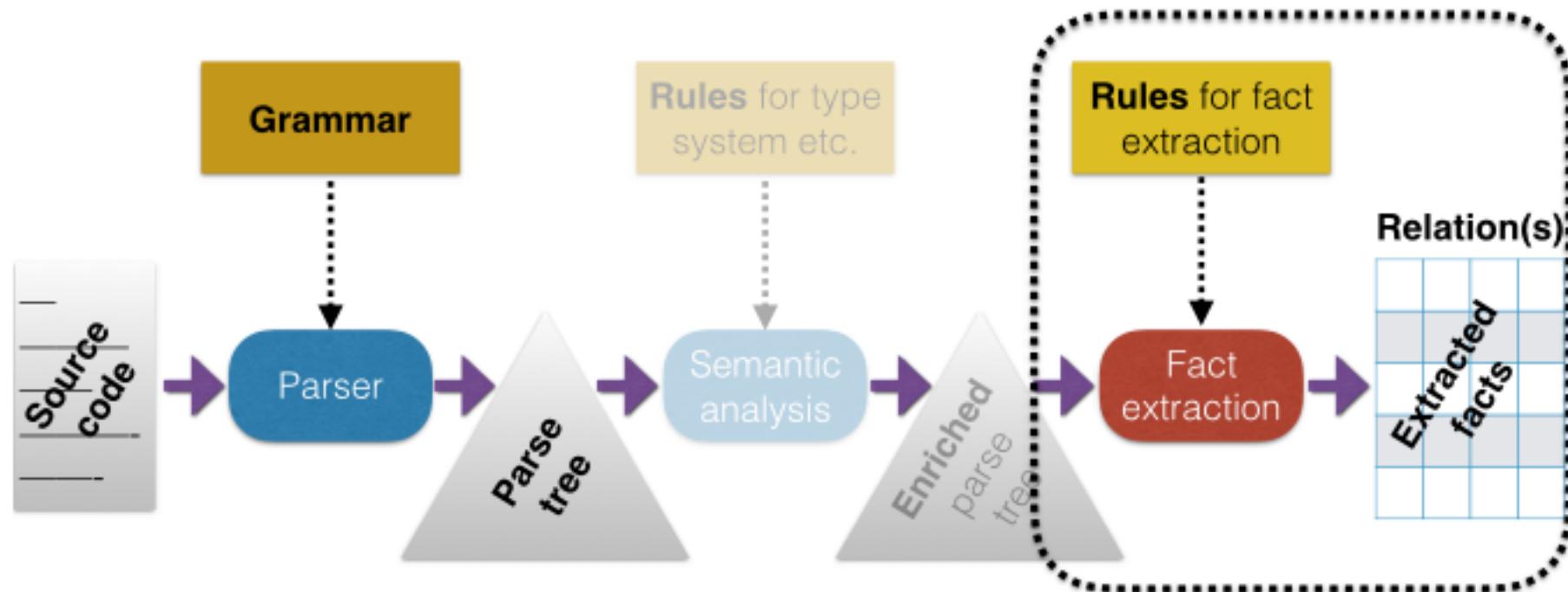
- SW **reengineering**
 - Refactoring
 - Migration
 - Wrapping
 - Traceability recovery
 - Design pattern detection
 - Change-impact analysis
 - Code smell
 - Software metric
- SW **reverse engineering**
- SW **analysis**
 - Program slicing
 - Feature location
 - Technological spaces
 - Model-driven engineering

Data flow in re-engineering tool



For instance: a method extraction refactoring in a Java IDE.

Data flow in reverse engineering tool



For instance: a call-graph extractor for architecture recovery.

Technological spaces

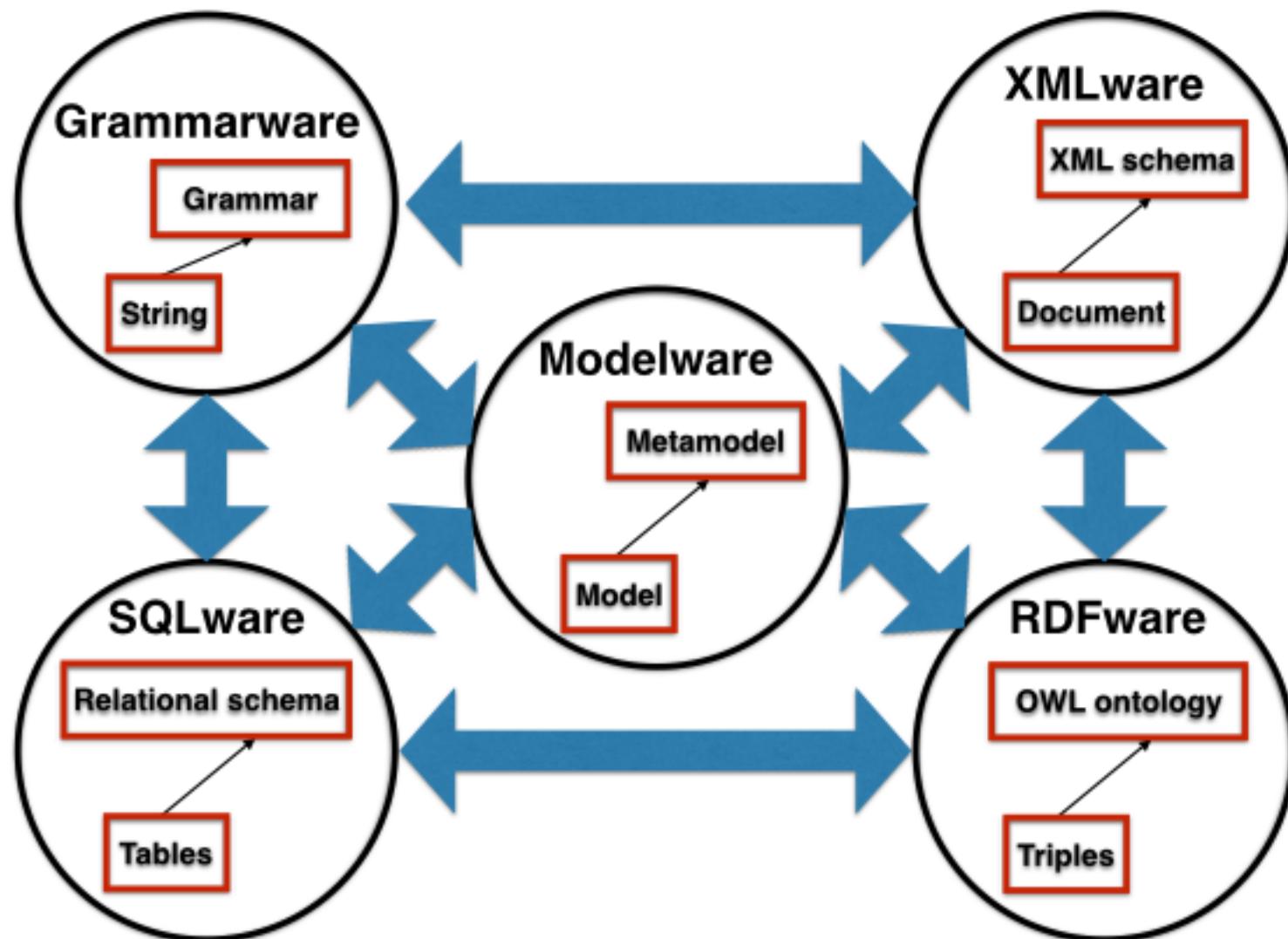
- **Spaces**

- Grammarware
- XMLware
- JSONware
- Modelware
- SQLware
- RDFware
- Objectware
- Javaware

- **Aspects**

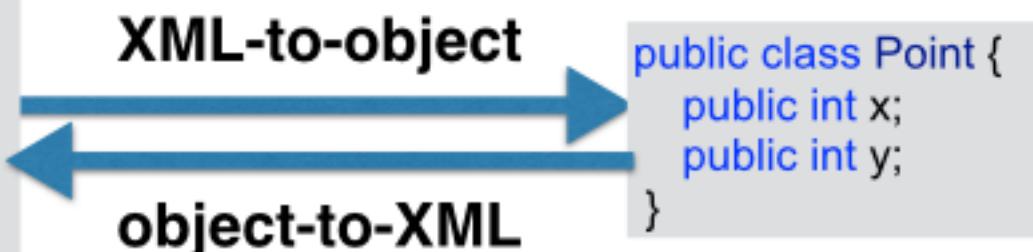
- Data models
- Schema languages
- Query languages
- Transformation languages
- PL integration
- Mapping
- Coupling

Technological spaces (Examples with ‘conformance’ relation)



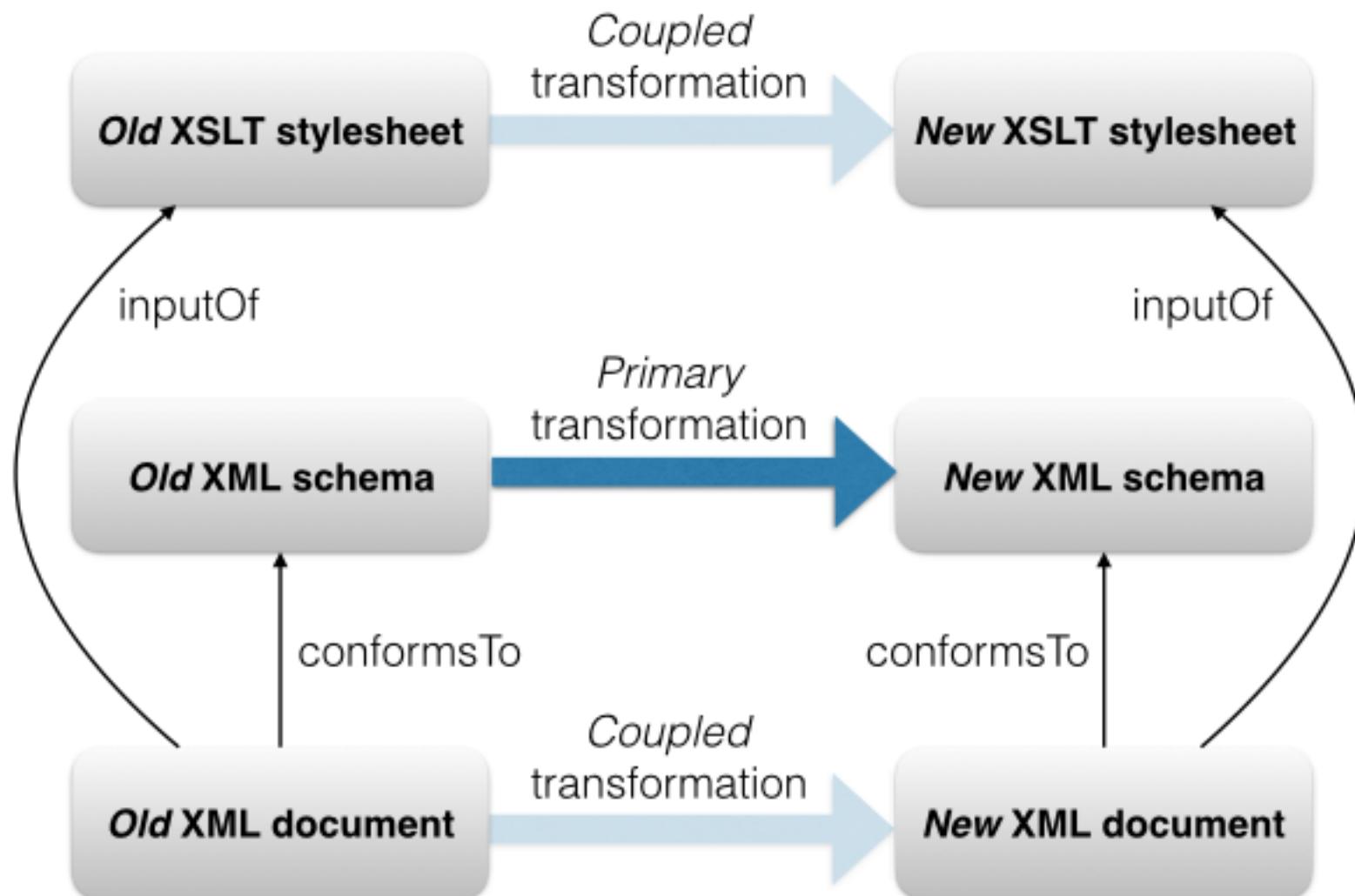
Mapping (Technological space travel)

```
<element name="point">
<complexType>
<sequence>
<element name="x" type="xs:int"/>
<element name="y" type="xs:int"/>
</sequence>
</complexType>
</element>
```



Coupling

(Recurrent complex transformation problem)



Model-driven engineering (A parallel universe)

- Modeling languages
- Domain-specific languages
- Model transformation
- Model evolution
- Model management
- Model comparison
- Model merging
- Model weaving
- Model synchronization
- Models@runtime
- Model co-evolution

Why do we need a BOK for SLE?

- Every software engineer deals with languages a lot.
- Language implementations form a special form of software.
- Wheels are reinvented in different technological spaces.
- Accidental complexity needs to be avoided.

(<https://pixabay.com/en/kitten-cat-baby-young-animal-408798/>)

https://twitter.com/grady_booch/status/835990696335040512



Grady Booch

@Grady_Booch

Follow



Hello, my name is Grady. I find @StackOverflow to be a wonderful aid to programming

And BTW I don't remember all the details of UML either.

Brian Fitzpatrick @therealfitz

Hi, my name is Fitz. I've been writing software since the early 90s and I still have to look up the correct syntax for join() in Python. twitter.com/harper/status/...

RETWEETS
117

LIKES
204



12:11 AM - 27 Feb 2017

10

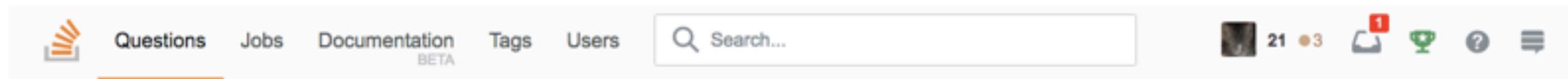
117

204



Stackoverflow
isn't the
Holy Grail.

[https://stackoverflow.com/questions/11828270/
how-to-exit-the-vim-editor](https://stackoverflow.com/questions/11828270/how-to-exit-the-vim-editor)



How to exit the Vim editor?

[Ask Question](#)

 I'm stuck and cannot escape. It says:

1657 "type :quit<Enter> to quit VIM"

 But when I type that it simply appears in the object body.

 vim vi

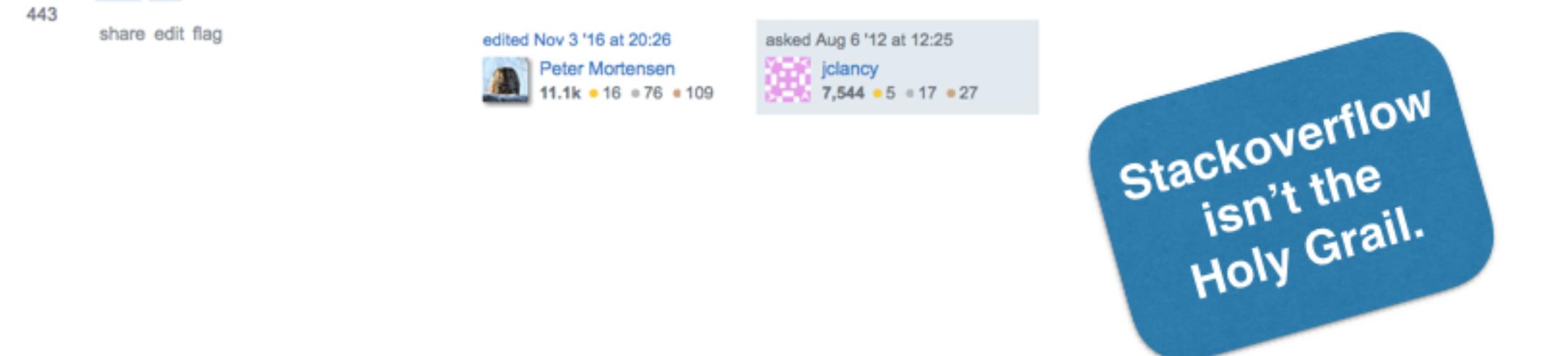
443 share edit flag

edited Nov 3 '16 at 20:26
 Peter Mortensen
11.1k ● 16 ● 76 ● 109

asked 4 years, 9 months ago

viewed 1055472 times

active today



How do we go about the SLEBOK?

- **Foremost, this is a community effort.**
- This is our specific focus though:
 - Aggregate & organize knowledge in *chrestomathies*
 - Use *megamodeling* for organization
 - Use *Linked Open Data* for presentation

This gives rise to what we would like to call an (advanced)
Software Language Repository (SLR)

[<https://pixabay.com/en/animal-cat-kitten-funny-yawning-339400/>]



20. – 25. August 2017, Dagstuhl Seminar 17342

SLEBOK: The Software Language Engineering Body of Knowledge

Over the last 10 years, the field of Software Language Engineering (SLE) has emerged based on a strong motivation to connect and integrate different research disciplines such as compiler construction, reverse engineering, software transformation, model-driven engineering, and ontologies. This Dagstuhl Seminar strives for directly promoting the further integration of said communities with the clear objective of assembling a Body of Knowledge on SLE (SLEBoK). The BoK features artefacts, definitions, methods, techniques, best practices, open challenges, case studies, teaching material, and other components that will afterwards help students, researchers, teachers, and practitioners to learn from, to better leverage, to better contribute to, and to better disseminate the intellectual contributions and practical tools and techniques coming from the SLE field.

What's a software chrestomathy?

chrestomathy

/krɛ'stɒməθi/ 

noun formal

a selection of passages from an author or authors, designed to help in learning a language.

[Google]

An example of a software chrestomathy

http://rosettacode.org/wiki/Rosetta_Code

Rosetta Code

Rosetta Code is a [programming chrestomathy](#) site. The idea is to present solutions to the same task in as many different languages as possible, to demonstrate how languages are similar and different, and to aid a person with a grounding in one approach to a problem in learning another. Rosetta Code currently has 850 [tasks](#), 198 [draft tasks](#), and is aware of 651 [languages](#), though we do not (and cannot) have solutions to every task in every language.

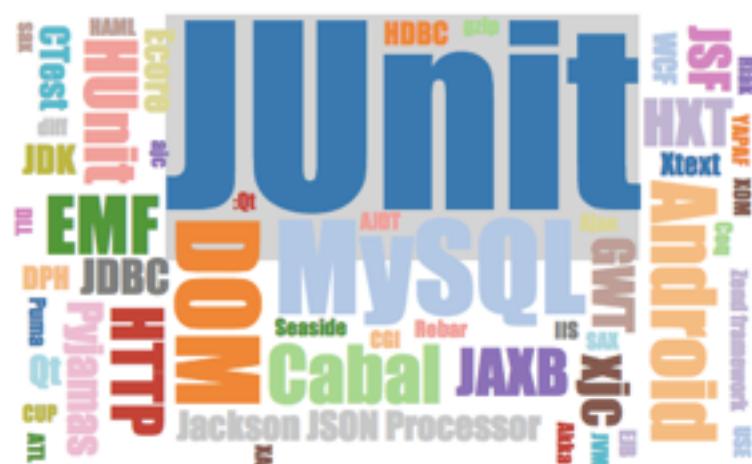


ROSETTACODE.ORG

Another example of a software chrestomathy

<https://101wiki.softlang.org/>

The project ‘101’ is an open knowledge resource covering software technologies, languages, and concepts. 101 targets programmers, software engineers, teachers, learners, and technologists; they can leverage 101 and they are encouraged to contribute to 101.

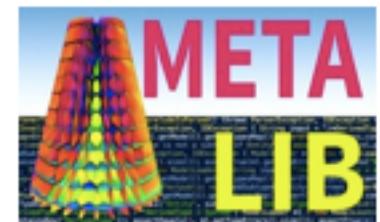


Characteristics of a software chrestomathy

- Community effort (for aggregation and evaluation)
- Requirement specification
- Multiplicity of languages
- Infrastructural support
- Revision and access control
- Quality assurance
- Rich metadata
- Process management

Outlook (regarding chrestomathies)

We will look at two software chrestomathies
that are specifically focused on being useful
for learning about software languages at the
level of implementation or metaprogramming.



What's a megamodel?

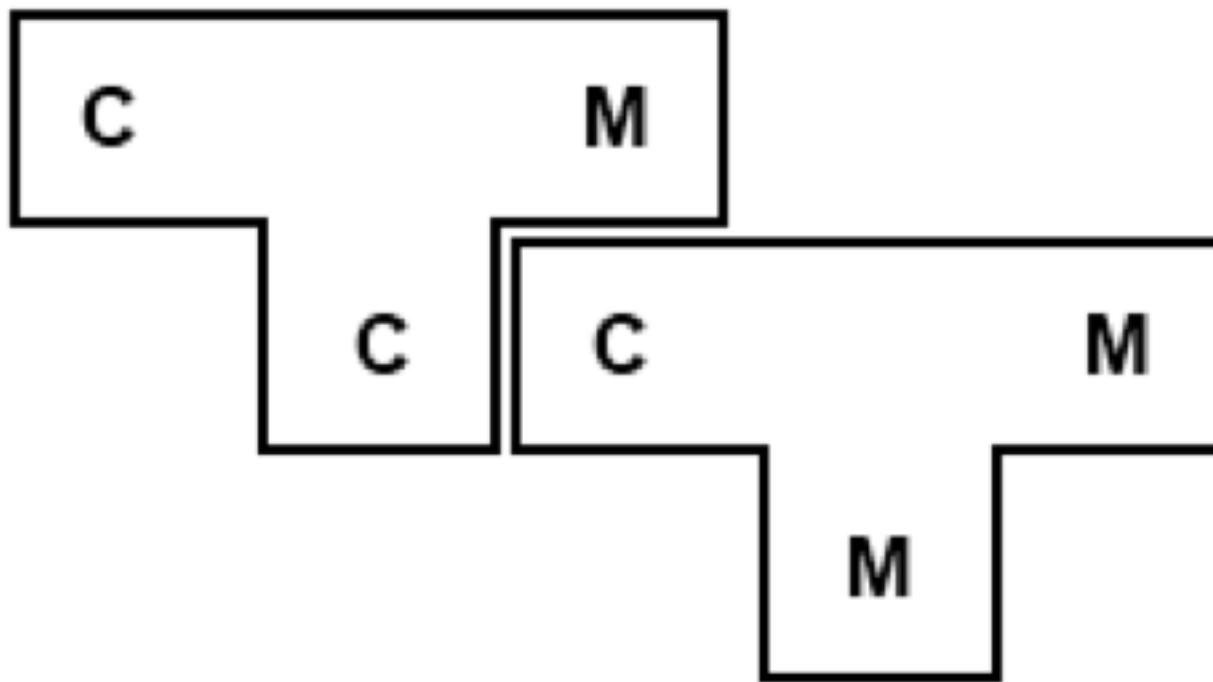
A model whose model elements are models.

The notion of model is to be interpreted broadly:

- **model, metamodel, model transformation,**
- **program, grammar, metaprogram,**
- **document, schema, transformation,**
- ...

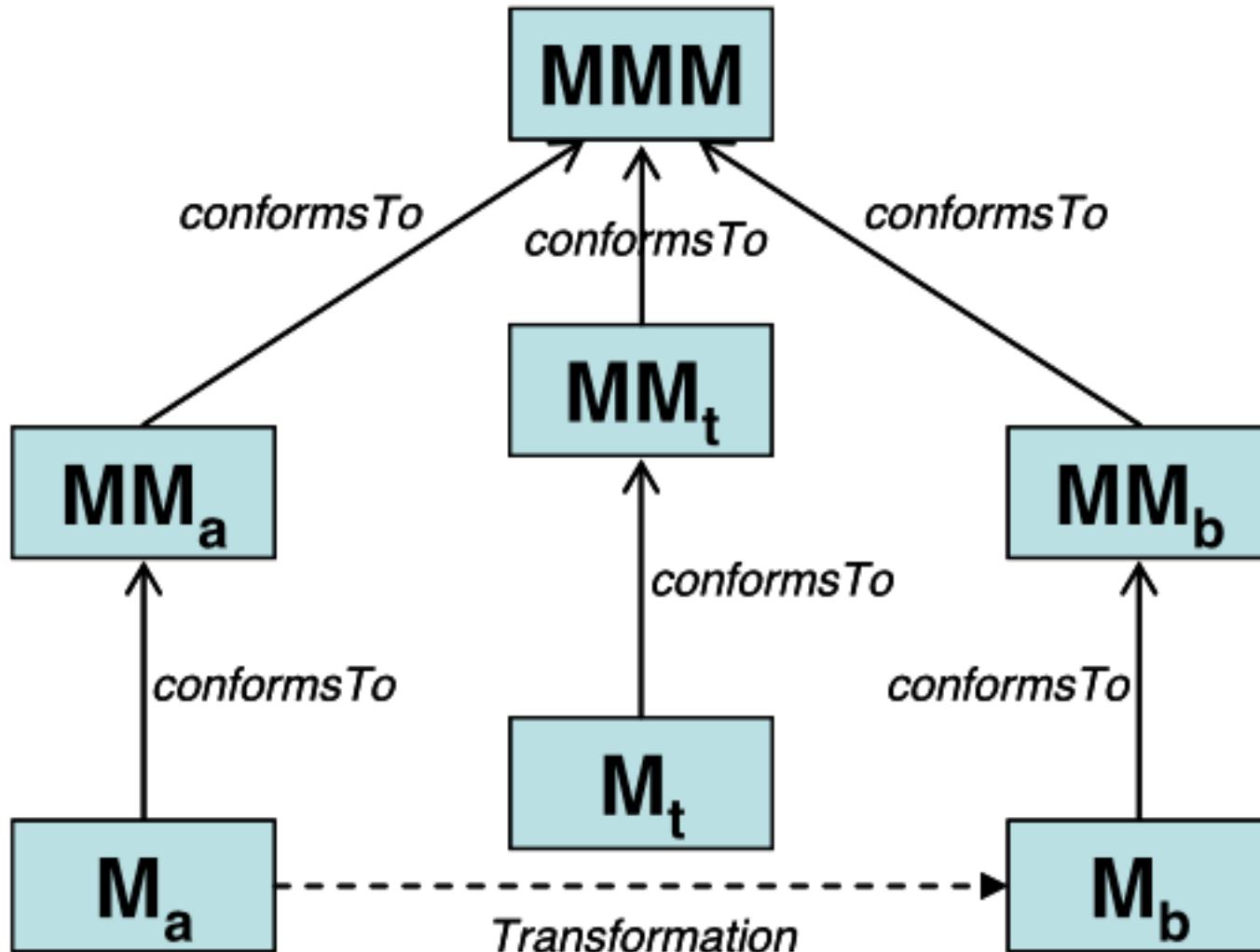
An example of a megamodel:

A tombstone diagram for bootstrapping a compiler



Another example of a megamodel:

Mechanics of an ATL-based transformation



What's *Linked Open Data* (LOD)?

“The Semantic Web isn't just about putting data on the web. It is about making links, so that a person or machine can explore the web of data. With linked data, when you have some of it, you can find other, related, data.” [<https://www.w3.org/DesignIssues/LinkedData.html>]

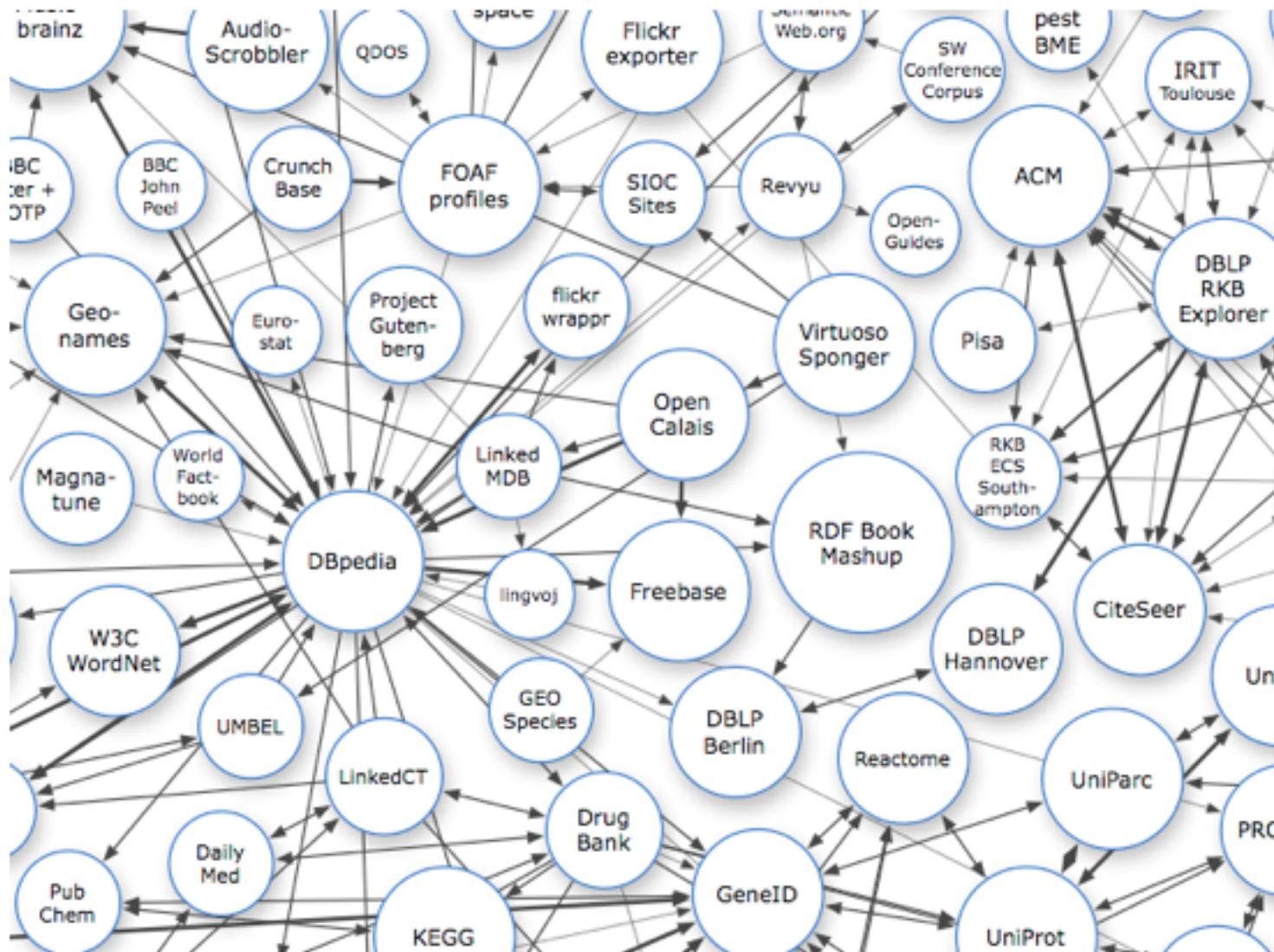
Tim Berners-Lee's LOD principles and stars

- Use URIs as names for things.
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).
- Include links to other URIs. so that they can discover more things.



[<https://www.w3.org/DesignIssues/LinkedData.html>]

<http://linkeddata.org/>



An example of LOD for a chrestomathy

<https://101wiki.softlang.org/resources>

About: language-haskell

An Entity of Type [functional_programming_language](#), from : [101companies.org](#)

[.json](#) [.xml](#) [.n3](#) [.ttl](#)

Properties (37)

Predicat	Object
isPrimaryTopicOf	wikipage-language-haskell
22-rdf-syntax-ns#type	functional_programming_language
22-rdf-syntax-ns#type	language
rdf-schema#label	Haskell
is uses of	contribution-dph

What's a *Software Language Repository* (SLR)?

A SLR is a repository with components for language processing (interpreters, translators, analyzers, transformers, pretty printers, etc.). SLRs are typically set up for developing and using metaprogramming systems, language workbenches, language definition frameworks, executable semantic frameworks, and modeling frameworks.

<https://github.com/about>

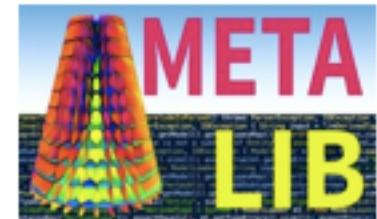
GitHub is how people build software

We're supporting a community where more than 22 million people learn, share, and work together to build software.

October 2007 First commit	San Francisco Headquarters	615 Employees worldwide	59+ million Projects hosted
------------------------------	-------------------------------	----------------------------	--------------------------------

Software Language Repositories

- Repos that go with systems/languages
 - ANTLR
 - Rascal
 - Spoofax
 - TXL
 - MoDisco
 - K semantic framework
 - PLT Redex
- Repo-oriented projects
 - MDEforge
 - SLPS
 - **YAS**
 - **MetaLib**



<https://github.com/antlr/grammars-v4>

The screenshot shows the GitHub repository page for `antlr/grammars-v4`. The page has a dark theme. At the top, there's a header with the repository name, a search bar, and user information for Ralf. Below the header are navigation links for 'This repository' (highlighted), 'Search', 'Pull requests', 'Issues', 'Marketplace', and 'Gist'. To the right are buttons for 'Watch', 'Star', 'Fork', and '778' (issues). A sidebar on the left shows 'Code' (selected), 'Issues 75', 'Pull requests 2', 'Projects 0', 'Wiki', and 'Insights'. The main content area starts with a brief description: 'Grammars written for ANTLR v4; expectation that the grammars are free of actions.' Below this are summary statistics: 1,848 commits, 2 branches, 2 releases, and 108 contributors. There are dropdowns for 'Branch: master' and 'New pull request', and buttons for 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The main feed lists recent commits:

Author	Commit Message	Time Ago
teverett	Merge pull request #728 from huan086/master	Latest commit f2a1a45 12 hours ago
abnf	antlr4test-maven-plugin.version	9 months ago
agc	Markdown: Correct headings, line breaks for better diff, better code ...	19 days ago
antlr3	minor antlr3 changes	a month ago
antlr4	antlr4: do not refer parser rules from lexer.	16 days ago
apex	\r\n -> \n for all *.g4 files.	14 days ago

<https://github.com/usethesource/rascal>

The screenshot shows a GitHub repository page for 'usethesource / rascal'. The URL in the address bar is <https://github.com/usethesource/rascal/tree/master/src/org/rascalimpl/library/demo>. The repository has 30 watchers, 135 stars, and 34 forks. The 'Code' tab is selected, showing a list of commits. The latest commit was by PaulKlint on Dec 2 2016, commenting out visualization related demos. Other commits include basic, common, lang, vis, Dominators.rsc, McCabe.rsc, Mod17.rsc, and Quine.rsc, all dated from 2013 to 2015.

Commit	Message	Date
basic	Cleaned-up and upgraded demo files	a year ago
common	Cleaned-up and upgraded demo files	a year ago
lang	Commented out visualization related demos	6 months ago
vis	Commented out visualization related demos	6 months ago
Dominators.rsc	issue #110: 2013 to 2015	2 years ago
McCabe.rsc	issue #110: 2013 to 2015	2 years ago
Mod17.rsc	fixed compilation error in Mod17 demo	11 months ago
Quine.rsc	issue #110: 2013 to 2015	2 years ago

<http://www.metaborg.org/>

The screenshot shows a web browser window with the address bar containing <http://www.metaborg.org/en/latest/>. The page title is "The Spoofax Language Workbench". The left sidebar has a blue header "Spoofax latest" and a search bar "Search docs". It lists several sections: "The Spoofax Language Workbench", "Examples", "Publications", a bolded "TUTORIALS" section containing "Installing Spoofax", "Creating a Language Project", "Using the API", and "Getting Support", and a "REFERENCE MANUAL" section containing "Syntax Definition with SDF3", "Static Semantics with NaBL2", "Transformation with Stratego", "Dynamic Semantics with DynSem", "Editor Services with ESV", "Language Testing with SPT", and "Building Languages". At the bottom of the sidebar are links for "Read the Docs" and "v: latest". The main content area shows the "The Spoofax Language Workbench" page with the heading "The Spoofax Language Workbench" and a paragraph about Spoofax being a platform for developing textual (domain-specific) programming languages. It lists several features: Meta-languages for high-level declarative language definition, an interactive environment for developing languages using these meta-languages, code generators that produce parsers, type checkers, compilers, interpreters, and other tools from language definitions, generation of full-featured Eclipse editor plugins from language definitions, generation of full-featured IntelliJ editor plugins from language definitions (experimental), and an API for programmatically combining the components of a language implementation. Below this is a paragraph stating that with Spoofax you can focus on the essence of language definition and ignore irrelevant implementation details. A horizontal line separates this from the footer.

The Spoofax Language Workbench

Spoofax is a platform for developing textual (domain-specific) programming languages. The platform provides the following ingredients:

- Meta-languages for high-level declarative language definition
- An interactive environment for developing languages using these meta-languages
- Code generators that produce parsers, type checkers, compilers, interpreters, and other tools from language definitions
- Generation of full-featured Eclipse editor plugins from language definitions
- Generation of full-featured IntelliJ editor plugins from language definitions (experimental)
- An API for programmatically combining the components of a language implementation

With Spoofax you can focus on the essence of language definition and ignore irrelevant implementation details.

Developing Software Languages

Ralf

TXL Home Page

www.txl.ca

What's new in the TXL Forum ?

NEW FreeTXL 10.6e

The TXL Programming Language

Source Transformation by Example

Welcome to the TXL Project web site. Here you will find everything about TXL - software, documentation, examples, support and more. Everything you need to take advantage of the best in source transformation systems!

» [About TXL](#)
What is TXL? What's it good for? What input languages can it handle? Who uses it? [Why's it called TXL?](#)

» [Documentation](#)
Reference manuals, user guides and learning materials for TXL. Publications about TXL and its applications.

» [Resources](#)
TXL World! The TXL grammar collection. Example applications. Useful rule sets and modules. Editor plugins for TXL.

» [NiCad Clone Detector](#)
Download NiCad4, a scalable, flexible code clone detection system based on TXL. **NEW** Current version NiCad 4.0 (February 2016)

» [Learn](#)
Introductory materials for learning TXL. The guided tour of TXL. The TXL Challenge. [How should I begin learning TXL?](#)

» [Download](#)
Download FreeTXL, a free and freely distributable TXL compiler / interpreter. Current version FreeTXL 10.6e (May 2017)

» [Support](#)
Having trouble? Check the FAQ. Ask a question. Report a bug or difficulty

TXL has grown with the support of NSERC, OCE, Esprit, CSEB, IBM Research and the IBM Center for Advanced Studies, and is presently supported by NSERC.

This website is maintained by members of the Software Technology Laboratory, Queen's University, Kingston, Canada, with the assistance of NSERC.
©2017 Queen's University at Kingston

FR

DE

IT

ES

<http://www.eclipse.org/MoDisco/>

The screenshot shows the MoDisco project page on the Eclipse website. At the top, there's a header with the Eclipse logo, a search bar labeled "Google Custom Search", and a "DONATE" button. Below the header, a navigation menu includes links for HOME, DOWNLOAD, DOCUMENTATION, SUPPORT, DEVELOPERS, and MORE. A breadcrumb trail at the top left indicates the current location: HOME / PROJECTS / MODISCO. The main content area features a large image of a disco ball with the "MoDisco" logo on it. To the right of the image are three main sections: "Documentation" (with a book icon), "Getting Involved" (with a people icon), and "Download" (with a green arrow icon). The "Download" section is highlighted with a yellow gradient background. Below these sections, there's a "Support" link with a question mark icon.

MoDisco

Legacy systems embrace a large number of technologies, making the development of tools to cope with legacy systems evolution a tedious and time consuming task. As modernization projects face with both technologies combination and various modernization situations, model-driven approaches and tools offer the requisite abstraction level to build up mature and flexible modernization solutions.

MoDisco provides an extensible framework to develop model-driven tools to support use-cases of existing software modernization :

About This Project

News

- [Graduation Time: MoDisco Neon Release \(v1.0.0\) is now available!](#)

Complete information and integrated downloads can be

<http://www.kframework.org/>

K Projects - K Framework Ralf

www.kframework.org/index.php/Projects

- [*C Semantics*](#)
Chucky Ellison
Defining a semantics of C in K
- [*Formal model-based language engineering in K*](#)
Dorel Lucanu, Vlad Rusu
Formal model-based language engineering using K
- [*Scheme Semantics*](#)
Patrick Meredith, Mark Hills
Defining a semantics of Scheme in K
- [*Haskell Semantics*](#)
David Lazar
Defining a semantics of Haskell in K
- [*Verilog Semantics*](#)
Patrick Meredith, Michael Katelyn
Defining a semantics of Verilog in K
- [*LLVM IR Semantics*](#)
David Lazar, Chucky Ellison
Defining a semantics of LLVM IR in K
- [*Esolang Semantics*](#)
Chucky Ellison, David Lazar

<https://github.com/racket/redex>

The screenshot shows a GitHub repository page for the 'racket / redex' project. The repository has 26 stars, 27 forks, and 14 open issues. The current branch is 'master'. The 'Code' tab is selected, showing the file 'beginner.rkt'. The file contains 937 lines (806 sloc) and is 26.9 KB in size. A commit by user 'samth' titled 'Remove extra dirs.' was made on Dec 2 2014, with a commit hash of e402d6f.

Branch: master | [redex](#) / [redex-examples](#) / [redex](#) / [examples](#) / [beginner.rkt](#) | [Find file](#) | [Copy path](#)

1 contributor

937 lines (806 sloc) | 26.9 KB | [Raw](#) | [Blame](#) | [History](#) | | |

```
1 #lang racket
2 #|
3
4 This is the semantics of Beginner Scheme, one of the
5 languages in DrRacket.
6
7 The first test case fails because the beginner spec
8 is broken for that program (ie, the model faithfully
9 reflects the (broken) spec).
10
11 |#
12
```

<http://www.mdeforge.org/>

The screenshot shows a web browser window with the URL www.mdeforge.org in the address bar. The page title is "MDEForge - MDE Forge". The header includes links for "Home Page", "About", "Publications", "API", and "Login". Below the header is a large banner image showing a complex network of white lines on a dark background, representing a modeling structure. The text "MDEForge Software-As-A-Service Modeling Platform" is visible at the bottom of the banner. The main content area features three circular icons with accompanying text: "Browse" (file folder icon), "Search" (magnifying glass icon), and "Use" (person icon). Each section has a brief description and a "Learn More" link.

MDEForge Software-As-A-Service Modeling Platform

Browse

Similarly to source code repositories, users can browse the repository in order to acquire knowledge from already developed modeling artifacts that might represent precious know-how to be conveyed to new modelers.

[Browse ▾](#)

Search

Users that have clear requirements about the wanted modeling artifacts are supported with advanced search facilities in order to find modeling artifacts that best fit the user needs.

[Learn Search ▾](#)

Use

Model management tools are available as software-as-a-service that can be remotely used without overwhelming the users with intricate and error-prone installation and configuration procedures.

[Learn Use ▾](#)

<http://slps.github.io/>

The screenshot shows a web browser window with the title bar "Software Language Processing X" and the address bar "slps.github.io". The main content area displays the SLPS logo (a stylized "S" and "L" followed by "P" and "S") and the text "Software Language Processing Suite". Below this, the word "Mission" is centered. A descriptive paragraph follows: "The project facilitates exposition and comparison of approaches and techniques on language processing in a way that is relevant for CS students, teachers, scientists, engineers and practitioners." A horizontal line separates this from a "Pages" section at the bottom. The "Pages" section contains a bulleted list of links:

- [Project page at GitHub](#) ([legacy project page at SourceForge](#))
- [List of recent commits](#) and [statistics on them](#) ([legacy subversion repository](#))
- [XBGF Language Manual](#)
- [SLPS Grammar Zoo](#) (Ada, C, C++, C#, Java, Modula, ...)
- [SLPS Grammar Tank](#) (BNF, EBNF, FL, TESCOL, ...)
- [TestMatch project](#) (comparing languages based on parsing generated test data)
- [Java grammars link repository](#)
- [FAQ \(legacy\)](#)
- [Other projects of @grammarware](#)

<http://www.softlang.org/yas>

The screenshot shows a web browser window with the title bar "YAS (Yet Another SLR (Software Language Repository))". The address bar contains the URL "www.softlang.org/yas". The main content area features a large orange header "YAS (Yet Another SLR (Software Language Repository))". Below it is a logo consisting of the letters "YAS" in a stylized font, where each letter is composed of overlapping colored segments (blue, red, yellow, orange). To the right of the logo is a section titled "Summary" which contains a detailed description of what YAS is. Further down is a "Links" section with four items, each preceded by a small blue folder icon. At the bottom of the page, there is footer text indicating the page revision and a link to stop watching the page.

YAS (Yet Another SLR (Software Language Repository))



Summary

A software language repository (SLR) is a software repository for software languages: definitions, implementations, language processors, and sample artifacts. **YAS** (Yet Another SLR) is an SLR targeting teaching and research on the foundations and engineering of software languages; it uses Haskell, Prolog, Java, and Python for implementing language processing functionality; YAS also exercises various other technologies for language processing, e.g., the ANTLR parser generator and the StringTemplate library for template processing. YAS is the codebase underlying the introductory textbook on software languages by this author. YAS relies on a megamodeling approach for build management and regression testing. In fact, YAS is the repository underlying the [Software Languages Book](#).

Links

- The YAS repository: ([GitHub](#))
- An emerging LOD view: ([.html](#))
- The Software Languages Book: ([.html](#))
- YAS' megamodeling language Ueber: ([.html](#))

page revision: 25, last edited: 9 May 2017, 18:01 (18 days ago)
[Stop watching this page \[?\]](#)

scalameta

meta

Case classes for object representation

Features: AST^-

Scala

Algebraic data type

```
package org.softlang.fsm

import scala.collection.immutable.Seq

package object AST {

    case class Fsm(states: Seq[State])

    case class State(initial: Boolean, id: String, transitions: Seq[Transition])

    case class Transition(event: String, action: Option[String], target: Option[String])

}
```

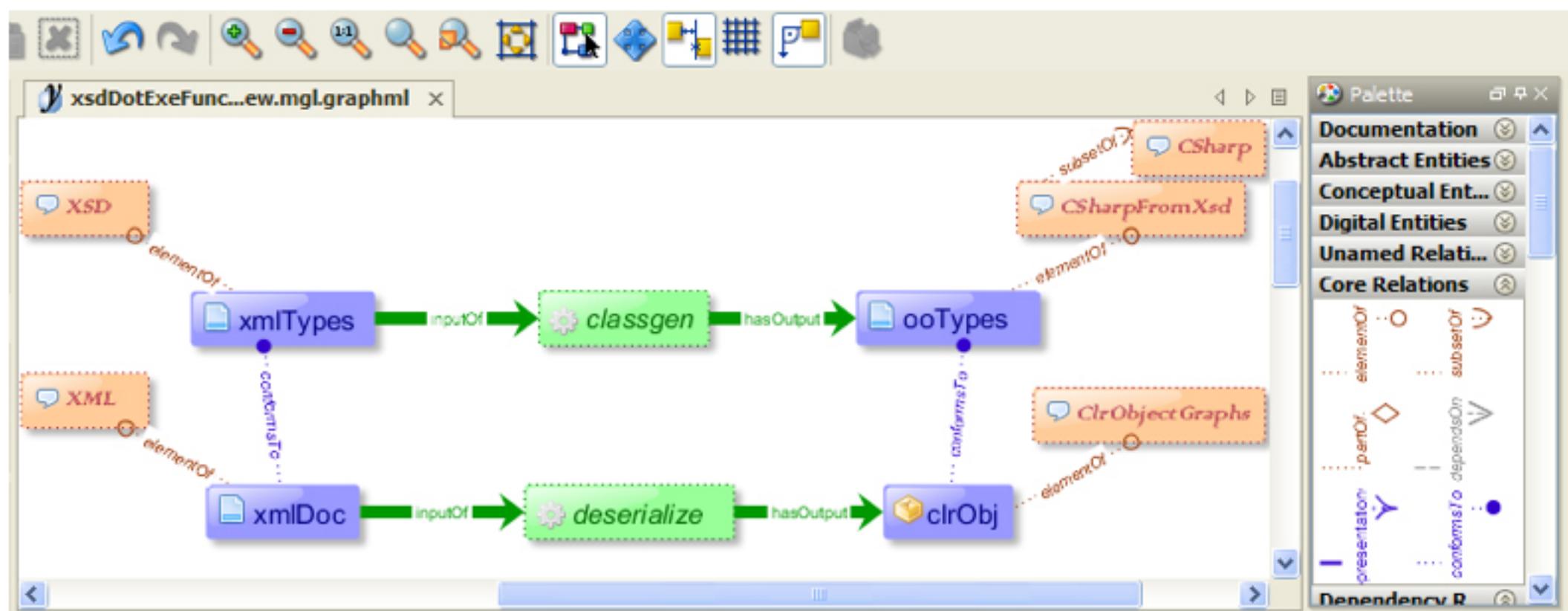
Megamodeling

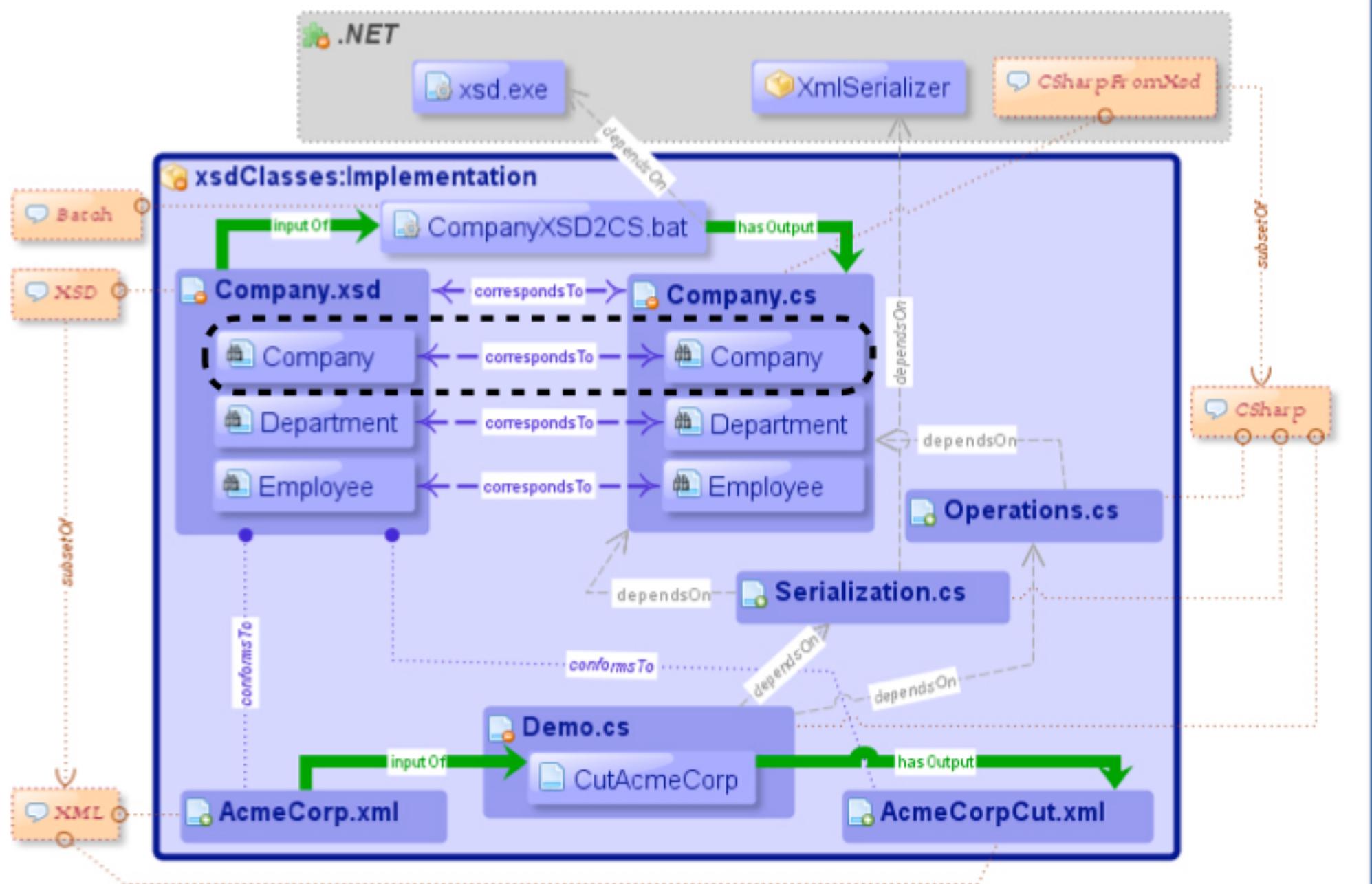
as an ontological approach
to **technology documentation**

Basic terminology

- Technology = Software technology, e.g.:
 - Web-application framework
 - O/R mapper
- Ontological documentation:
 - a form of **megamodeling**
 - also referred to as (model of) **linguistic architecture**

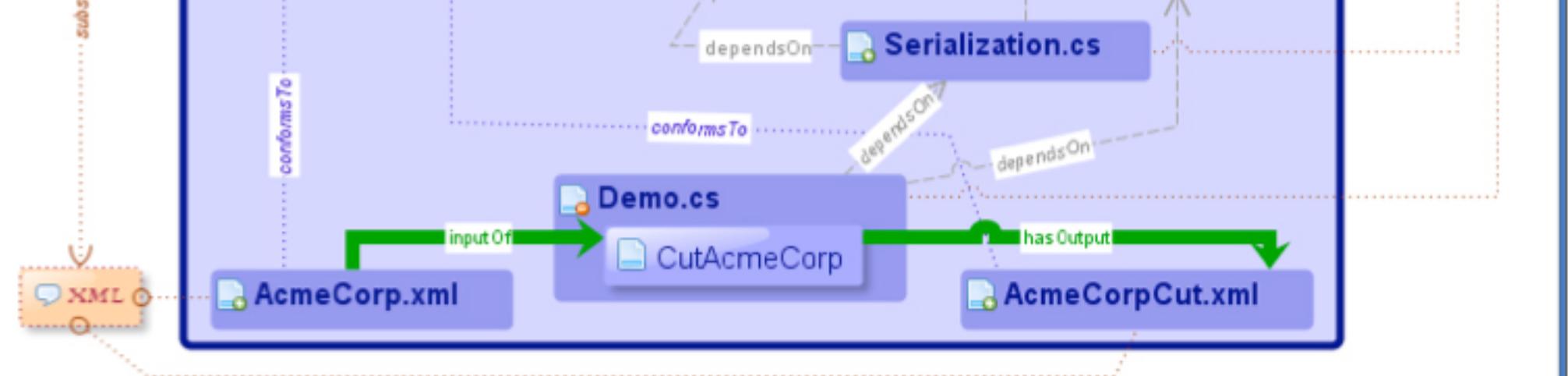
Linguistic architecture of xsd.exe





```
1. <?xml version="1.0" encoding="UTF-8" star-  
2. <xsd:schema xmlns="http://www.softlang.org/  
3.  
4.   <!-- schema information -->  
5.   <!-- schema information -->
```

```
21. [System.Diagnostics.DebuggerStepThroughAttribute]
22. [System.ComponentModel.DesignerCategoryAttribute("Code")]
23. [System.Xml.Serialization.XmlTypeAttribute(Namespace = "http://www.w3.org/2001/XMLSchema")]
24. [System.Xml.Serialization.XmlRootAttribute(Namespace = "http://www.w3.org/2001/XMLSchema",
```



```

1. <?xml version="1.0" encoding="UTF-8" star
2. <xs:schema xmlns="http://www.softlang.org
3.
4.     <xs:element name="Company">
5.         <xs:complexType>
6.             <xs:sequence>
7.                 <xs:element name="Name" type="xs:
8.                 <xs:element name="TopLevelDepartn
9.             </xs:sequence>
10.            </xs:complexType>
11.        </xs:element>
12.
13.        <xs:complexType name="Department">
14.            <xs:sequence>
15.                <xs:element name="Name" type="xs:st
16.                <xs:element name="Manager" type="En
17.                <xs:element name="SubDepartment" t
18.                <xs:element name="Employee" type="E
19.            </xs:sequence>
20.        </xs:complexType>
21.    
```

```

21. [System.Diagnostics.DebuggerStepThroughAt
22. [System.ComponentModel.DesignerCategoryAt
23. [System.Xml.Serialization.XmlTypeAttribut
24. [System.Xml.Serialization.XmlRootAttribut
25. public partial class Company {
26.
27.     private string nameField;
28.
29.     private Department[] topLevelDepartme
30.
31.     /// <remarks/>
32.     public string Name {
33.         get {
34.             return this.nameField;
35.         }
36.         set {
37.             this.nameField = value;
38.         }
39.     }
40.
41.     /// <remarks/>

```

Overarching research questions

- What are problems with classic documentation?
- How to ontologically structure documentation?
- How to address related needs of developers?
- What is the underlying vocabulary and ontology?
- What sort of tool support is necessary or helpful?
- How to actually renarrate such documentation?

An ontological documentation
for using EMF+Xtext+ATL
for the purpose of DSML implementation

Discovery of entities and relationships

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

Megamodel of the XML story

```
module XML import (Prelude) // Import basic vocabulary
XML : Language // Declare XML as a language entity
XSD : Language // and XSD (XML Schema), too
XSD subsetOf XML // Subset relationship on XSD and XML
xmlFile : Artifact // Declare artifact
xsdFiles : Artifact+ // Declare collection of artifacts
xmlFile elementOf XML // Assign language to artifact
xsdFiles elementOf XSD // Assign language to artifact
xmlFile conformsTo xsdFiles // XSD-based validation
```

Megamodel of the EMF story

```
module EMF import (Prelude)
Ecore : Language // As defined by metaMetaModel
Custom : Language // As defined by metaModel
metaModel : Artifact // A metamodel artifact
metaMetaModel : Artifact // The metamodel
metaModel elementOf Ecore
metaMetaModel elementOf Ecore
metaModel conformsTo metaMetaModel
metaMetaModel conformsTo metaMetaModel
metaModel defines Custom
metaMetaModel defines Ecore
```

Megamodel of the ATL story

```
module ATL import (EMF) // ATL depends on EMF
transformation : Custom → Custom // Function on DSL
input : Artifact // Input artifact
output : Artifact // Output artifact
input elementOf Custom // input (source) of transformation
output elementOf Custom // output (target) of transformation
transformation(input) ↣ output // Function application
ATL : Language // The ATL language
atlmodule : Artifact // An ATL transformation module
atlmodule elementOf ATL
atlmodule defines transformation // Semantics of ATL module
```

Megamodel of the Xtext story

```
module Xtext import (EMF) // Xtext integrates with EMF
Xtext : Language // Xtext language
grammar : Artifact // An artifact for the grammar
grammar elementOf Xtext // An Xtext grammar
EcoreWithoutOps : Language // Relevant subset of Ecore
EcoreWithoutOps subsetOf Ecore
metaModel elementOf EcoreWithoutOps // Restriction of import
metaModel correspondsTo grammar // Correspondence
generator : Xtext → EcoreWithoutOps // Generator function
generator(grammar) ↫ metaModel // Generator application
MWE2 : Language // Language for generator configuration
workflow : Artifact // Workflow artifact
workflow elementOf MWE2 // Workflow is written in MWE2
workflow defines generator // Workflow defines generator function
```

Megamodel of EMF Model API story

```
module EMFModelAPI import (
    EMF, // The EMF module is enhanced
    XML) // The XML module is needed for serialization
Java : Language // Java is a Language
EcoreJava : Language // A Java subset for EMF Model APIs
EcoreJava subsetOf Java // An EMF Model API is valid Java
EMFGenModel : Language // Language for the generator model
genModel : Artifact // Parameters of the generation
genModel elementOf EMFGenModel
genModel references metaModel // Referencing
EMFGenerator : EMFGenModel → EcoreJava
EMFGenerator(genModel) → javaFiles // Application of generator
CustomObjects : Language // Object graphs for Custom
Serialization : CustomObjects → Custom
Deserialization : Custom → CustomObjects
XMI : Language // Format for default persistence for EMF
XMI subsetOf XML // XMI is a subset of XML
```

XMI : Language // Format for default persistence for EMF

XMI subsetOf XML // XMI is a subset of XML

Custom subsetOf XMI // Custom uses default persistence

javaFiles : Artifact+ // The modeled/defined API

javaFiles elementOf EcoreJava

metaModel correspondsTo javaFiles // Close resemblance

javaFiles defines CustomObjects

javaFiles defines CustomSerialize

javaFiles defines CustomDeserialize

model : Artifact // A serialized artifact

model elementOf Custom // ... of Custom language

model conformsTo metaModel // Conformance to metamodel

objectGraph : Transient // A runtime artifact

objectGraph elementOf CustomObjects

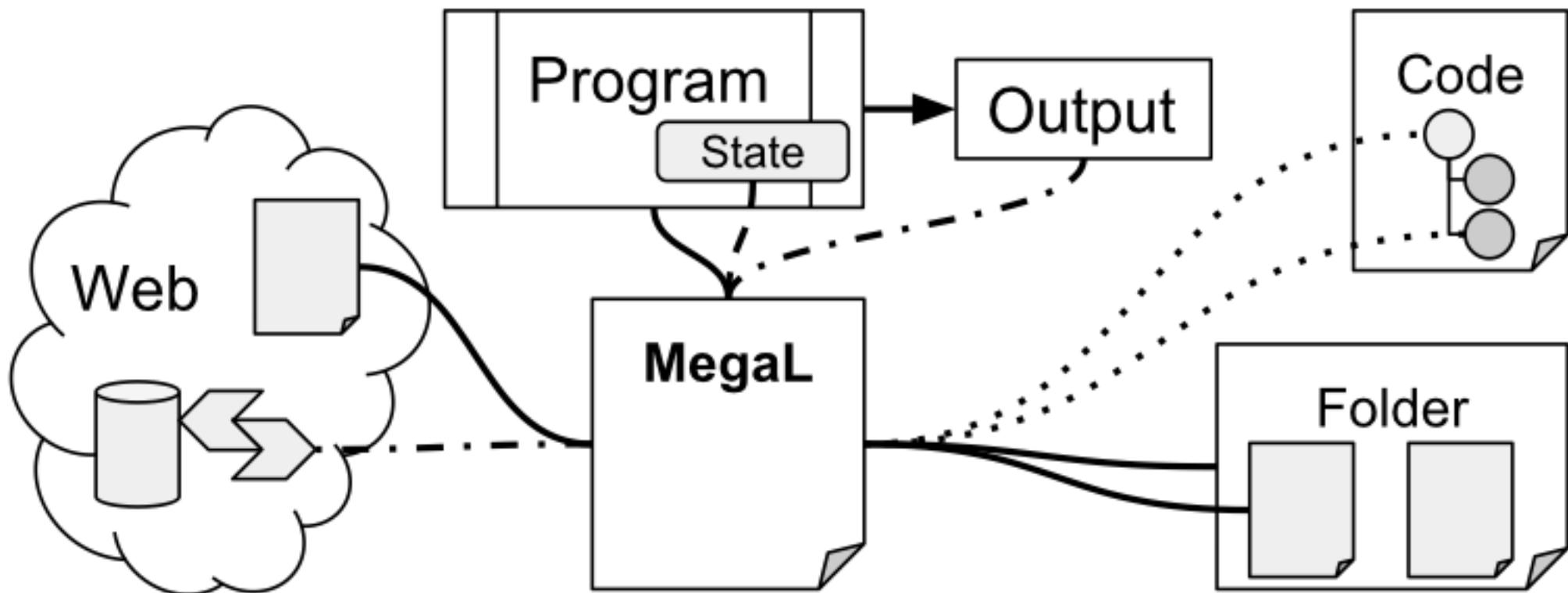
objectGraph conformsTo javaFiles // Conformance to Java classes

CustomSerialize(objectGraph) \mapsto model

CustomDeserialize(model) \mapsto objectGraph

Interconnected linguistic architecture

Interconnection of model and system



URI-based resolution

```
github://user/project/files/data.jar/content.xml/root/models/model#1
```

- GitHub repo
- File
- Archive content
- File again
- XML-based (XPath-like) selection

// module EMF continued

```
metaMetaModel = 'eclipse:/org.eclipse.emf.ecore/model/Ecore.ecore'
```

MegaL

Semantic annotations

'Identity' links to Wikipedia etc.

XML | <http://dbpedia.org/page/XML>

EMF | <https://eclipse.org/modeling/emf/>

// module XML continued

XML = '<http://dbpedia.org/page/XML>'

MegaL

// module EMFModelAPI continued

Persistence : Concept

Persistence =

'[http://dbpedia.org/page/Persistence_\(computer_science\)](http://dbpedia.org/page/Persistence_(computer_science))'

CustomSerialize facilitates Persistence

CustomDeserialize facilitates Persistence

Pluggable analyses



xmlFile *elementof* XML

xmlFile

✖ File not element of language:

The element type "name" must be terminated by the matching end-tag "</name>".

Plugin

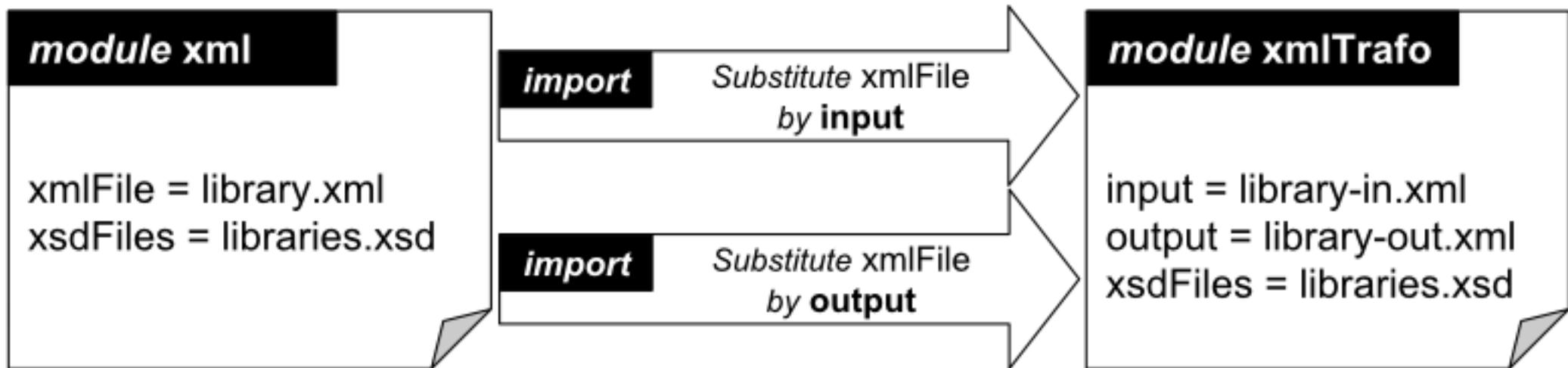
```
class XMLConformsToXSD extends MegaLEvaluator {  
    // Returns an evaluation report on the model element  
    protected Report<Void> evaluate(Relationship element) {  
        // Use SAX for validation; translate exceptions to report  
        ...  
    }  
}
```

Plugging in

```
conformsTo < Artifact * Artifact // Relationship type per prelude
ConformsToEvaluator : Plugin // Root plugin for conformance
ConformsToEvaluator = "classpath:ConformsToEvaluator"
conformsTo evaluatedBy ConformsToEvaluator
XMLConformsToXSD : Plugin // XML/XSD conformance
XMLConformsToXSD = "classpath:XMLConformsToXSD"
XMLConformsToXSD partOf ConformsToEvaluator
SAX : Technology // Semantic annotation of plugin
SAX = 'http://dbpedia.org/page/Simple_API_for_XML'
XMLConformsToXSD uses SAX
```

MegaL

Modularized models



A screenshot of a software interface showing the details of the **XMLAcceptor** plugin.

XMLAcceptor : Plugin

Bound to [classpath:plugins.jaxb.AcceptXML](#)

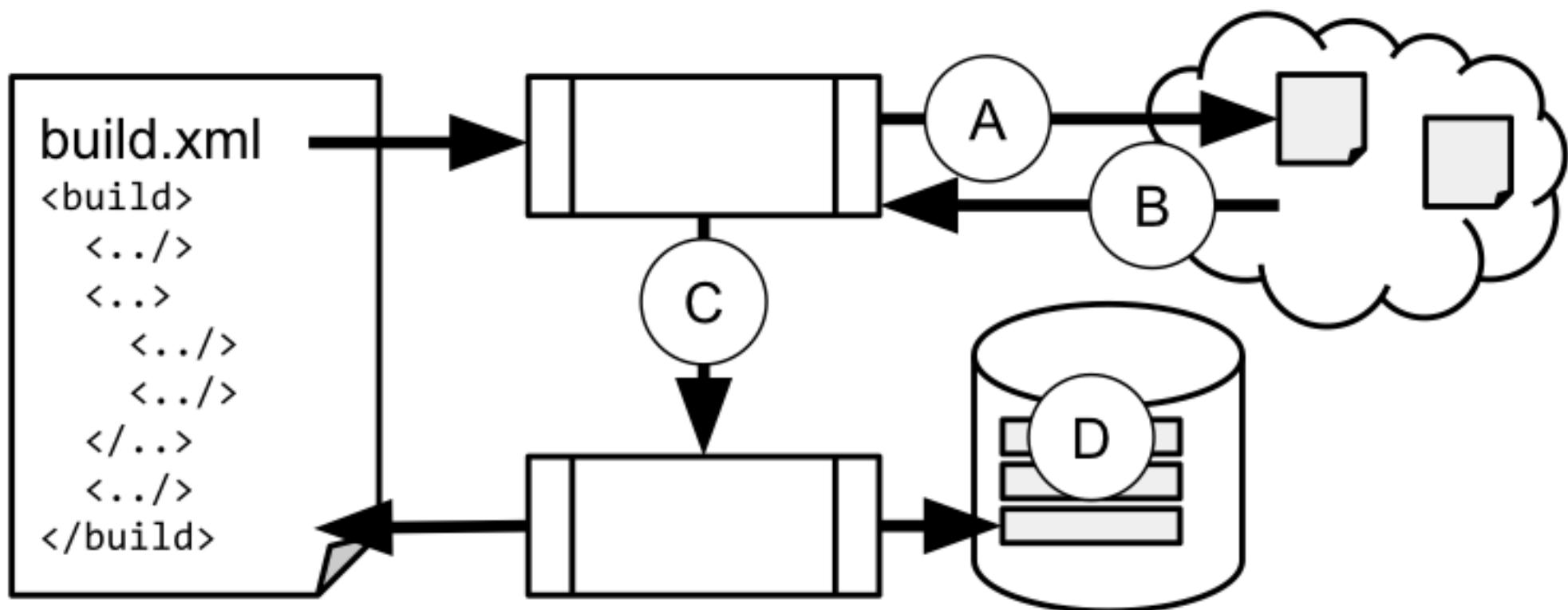
Relationships in defined document:

- **XMLAcceptor** partOf StringAcceptor

On the left, there is a tree view of relationships:

- ◆ VersionOfReasoner
- ◆ XML
- ◆ **XMLAcceptor** (highlighted)
- ◆ xmlFile
- ◆ XSD

Transient artifacts



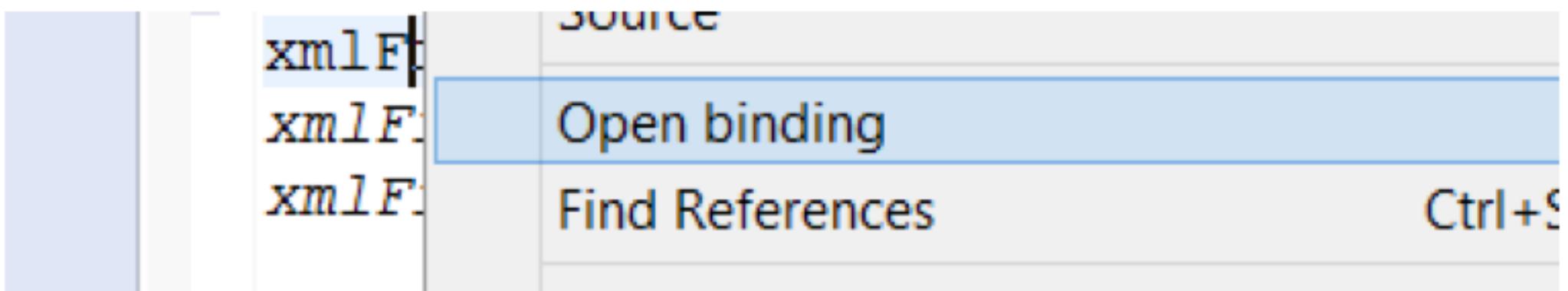
- A and B: web request and response
- C: piped program output
- D: transient data in memory or database

Model inference

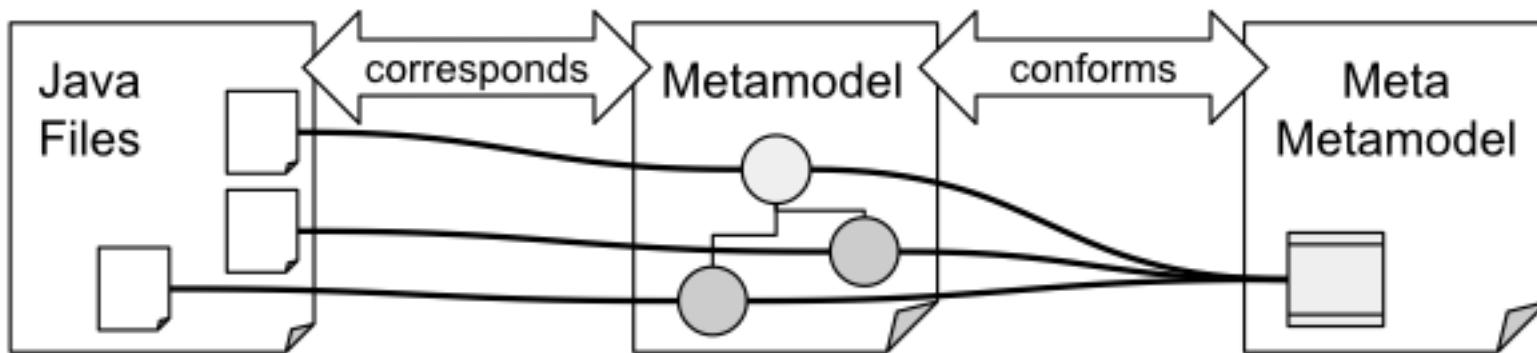
- Decomposition in parts
- Correspondence between parts
- Artifact bindings
- Subset relationship between languages

```
class EMFPartInerrer extends MegaInerrer {  
    // Returns an evaluation report and a model extension  
    protected Report<Megamodel> infer(Entity element) { ... }  
}
```

Explorable connections



Traceability links



xsdFiles

- /xs:schema/xs:complexType
- /xs:schema/xs:element#0
- /xs:schema/xs:element#1

xmlFile

- company/department#0
 - employee#0
 - address:Utrecht
 - name:Erik
 - salary:12345
- employee#1

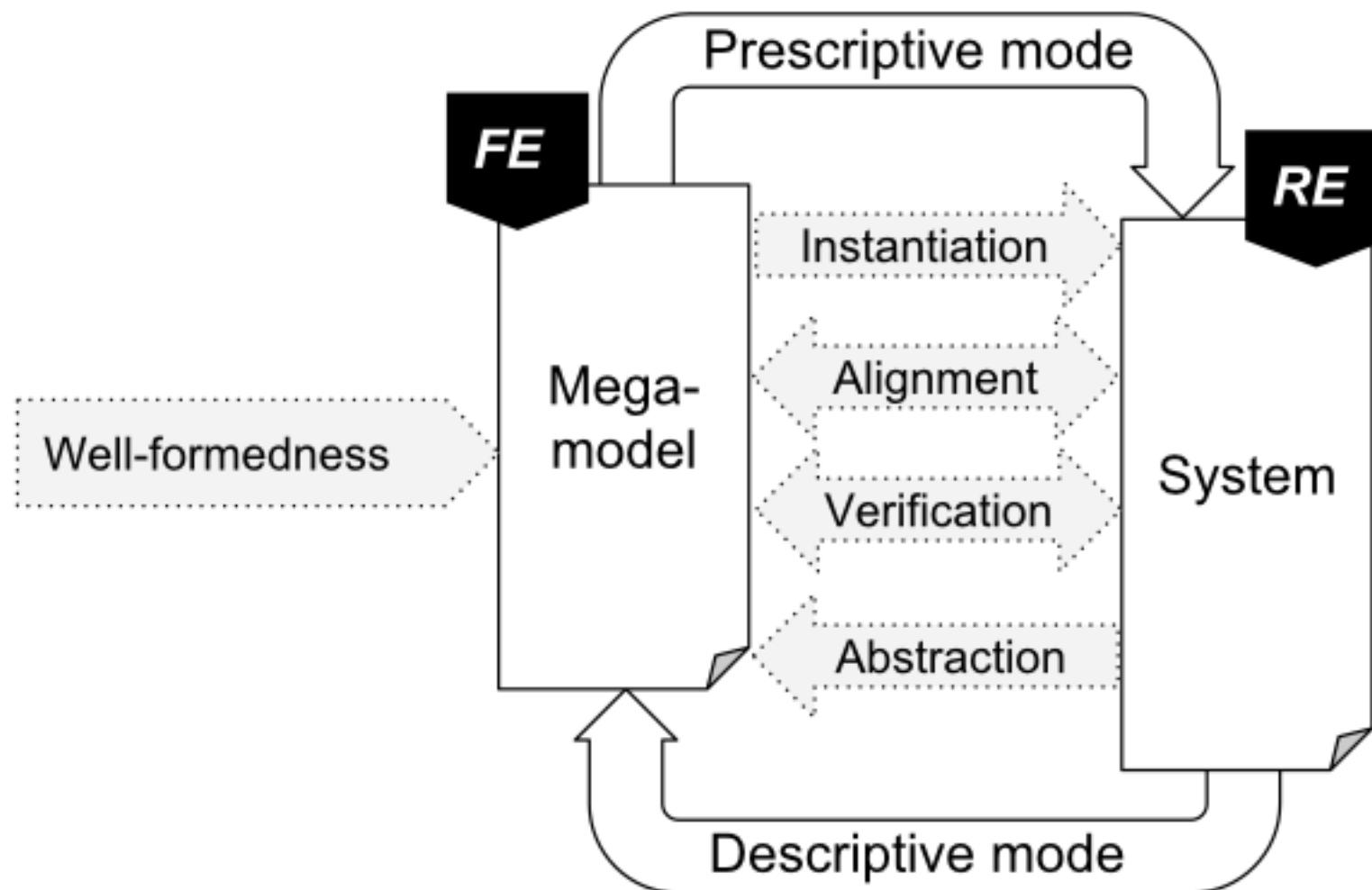
javaFiles

org/softlang/company/xjc/Employee.java	javaFiles
org/softlang/company/xjc/Company.java	
org/softlang/company/xjc/Department.java	
objectGraph	
org.softlang.company.xjc.Department@5fd1a6aa	
org.softlang.company.xjc.Employee@1a56a6c6	
Utrecht	
Erik	
12345.0	
org.softlang.company.xjc.Employee@748e432b	

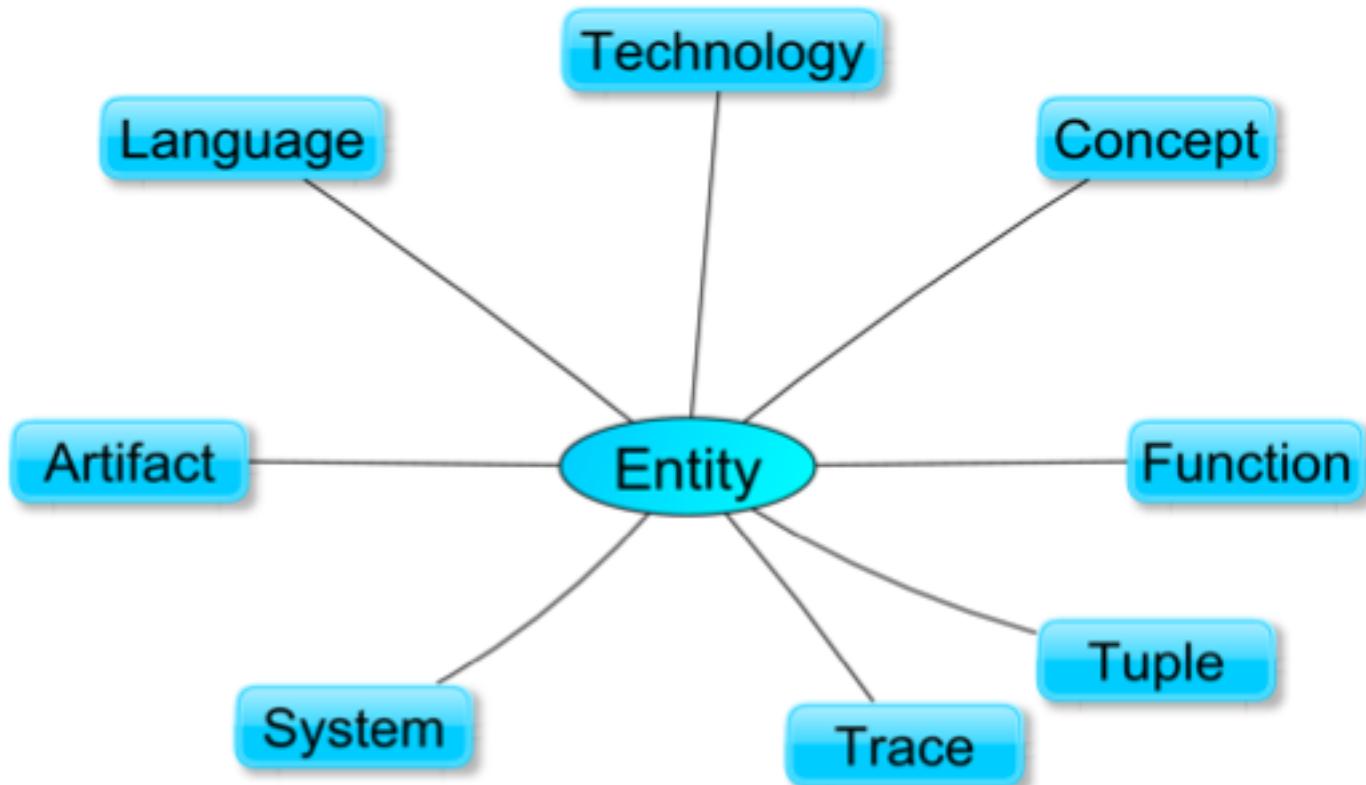
Literature survey

[30]	
[7]	
[25]	
[15]	
[11]	
[17]	
[12]	
[3]	
[4]	
([10])	
L_3		○		○			○		
L_2	⊗	×	⊗	×	×	×	×	○	
L_1		×			○	○	×	×	
<ul style="list-style-type: none">• L_1-L_3 maturity levels• ○ implementation• × demonstration		Traceability links 3.8	Artifact binding 3.1	Model inference 3.6	Pluggable analyses 3.3	Explorable connections 3.7	Modularized models 3.4	Semantic annotations 3.2	Transient artifacts 3.5

Linguistic architecture in forward and reverse engineering



Axioms of linguistic architecture



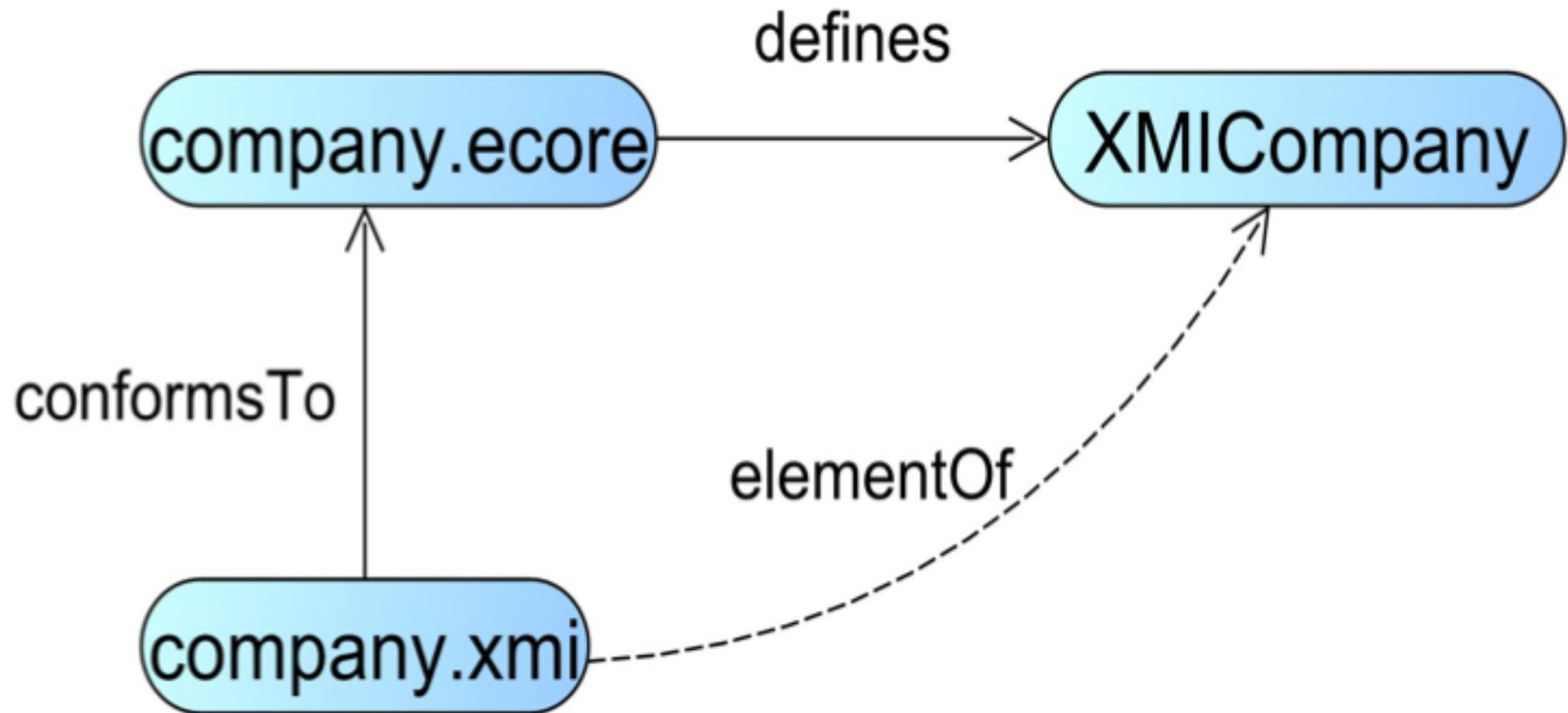
Entity types in megamodeling survey

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	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										

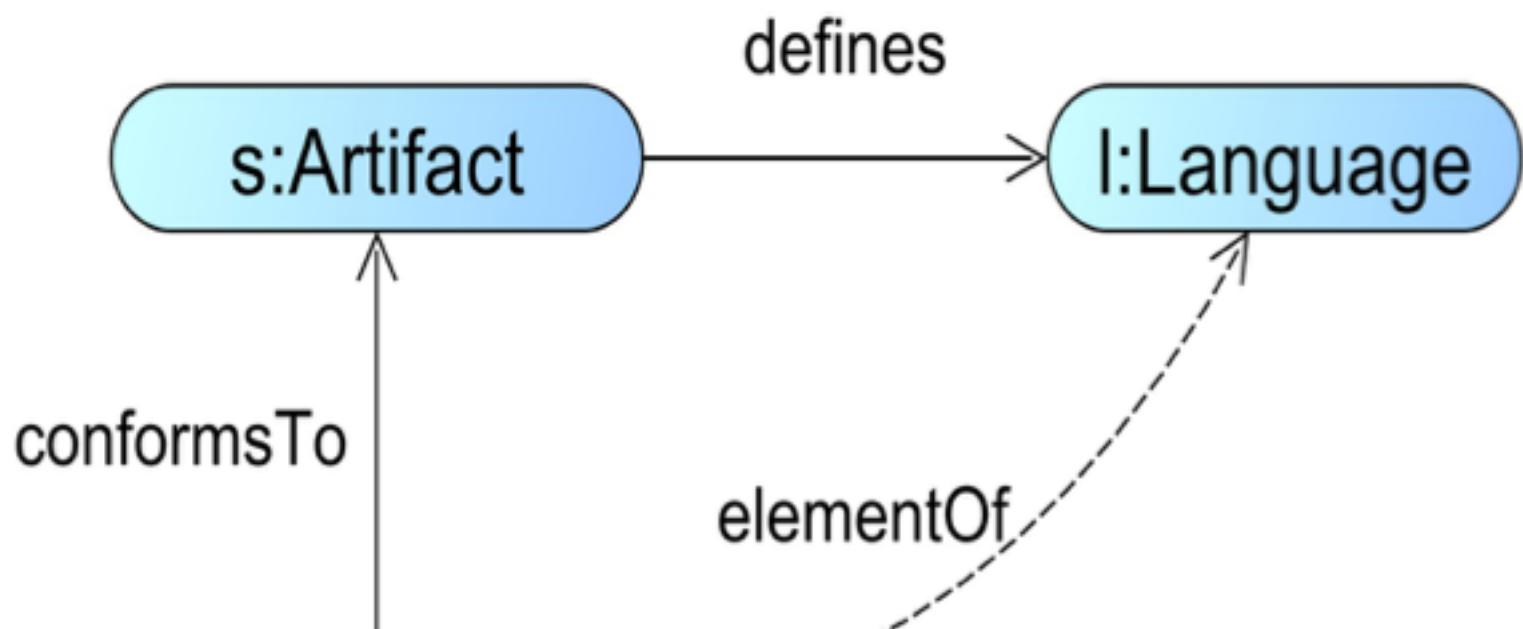
Relationship types in megamodeling survey

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		

Understanding Membership

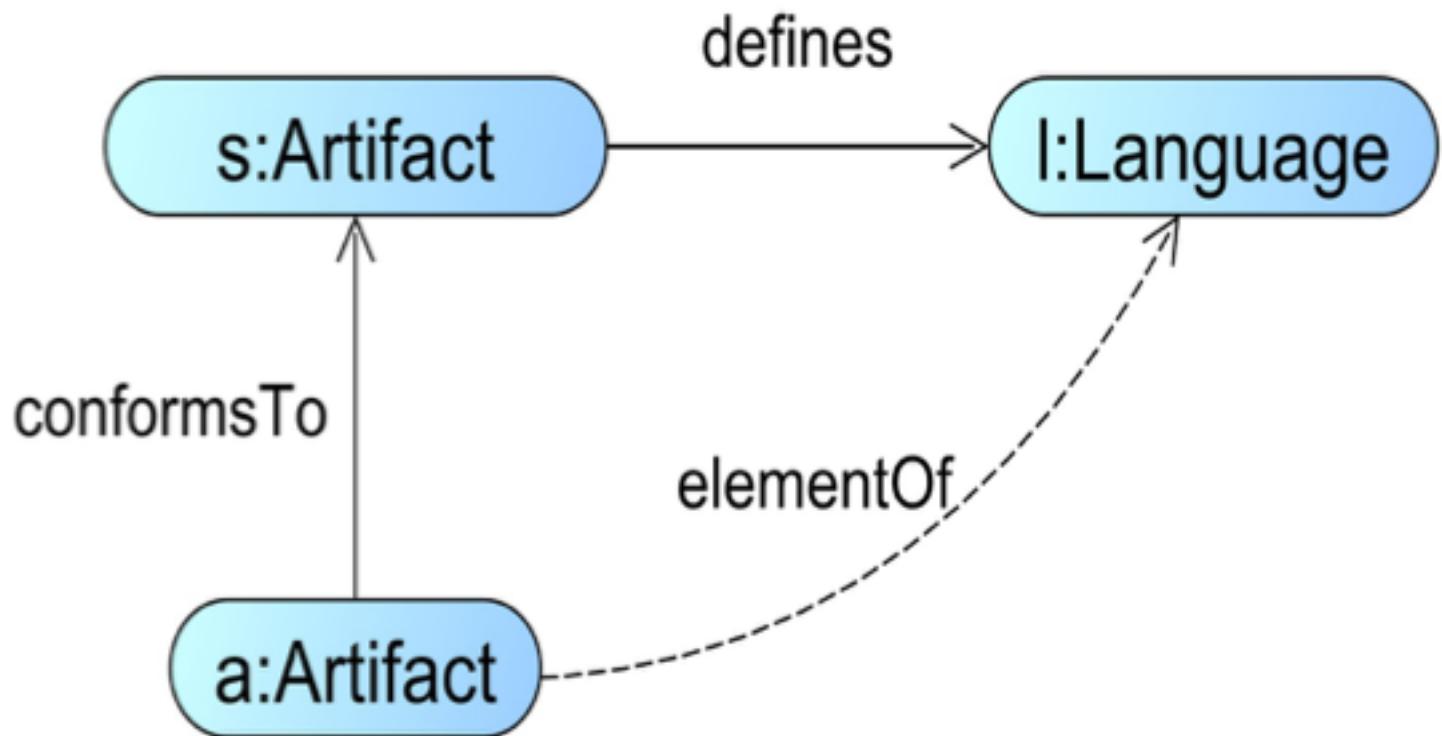


Understanding Membership



<u>s</u>	<u>a</u>
Grammar	Code
Schema	Instance
Metamodel	Model

Understanding Membership

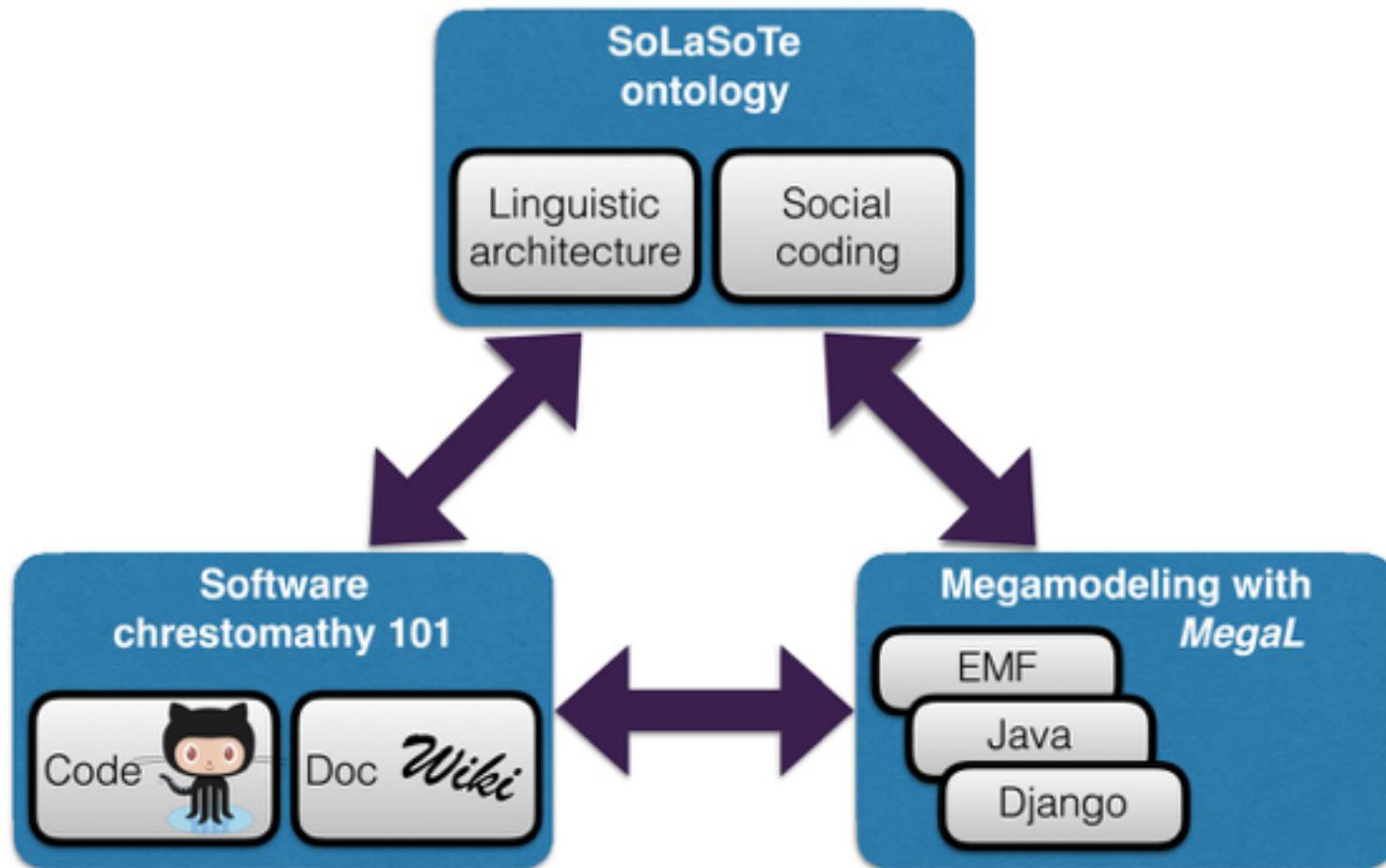


- ▶ $\text{elementOf}(a, l) \Rightarrow \text{Artifact}(a) \wedge \text{Language}(l) \dots$
- ▶ $\text{elementOf}(a, l) \Leftarrow \exists s. \text{defines}(s, l) \wedge \text{conformsTo}(a, s).$

Understanding Membership

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

Ontology engineering



Megamodeling

as means of
consistency management based on
build management and regression testing

YAS (Yet Another SLR (Software Language Repository))

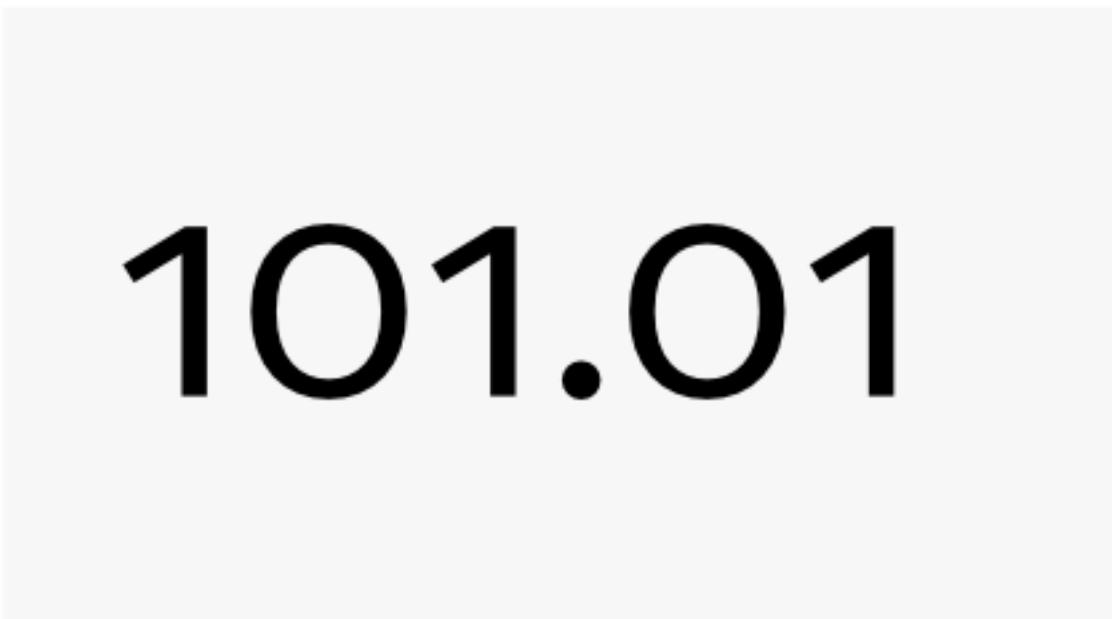
- 107 languages. (This includes different representation types.)
- 558 language-typed artifacts.
- 121 language-typed functions.
- 391 function applications.
- 252 Prolog modules.
- 171 Haskell modules.
- 111 Java classes.
- 19 Python scripts.

How to build and test such a repository?
How to understand all the dependencies?

Illustrative YAS artifacts



A binary number



101.01

BTW, everything is a file in a/this SLR.

2.1 Examples of languages

Figure 2 shows basic representation types in YAS and a few more specific software languages related to different aspects of a simple language BNL—*Binary Number Language*. The nodes in the figure denote languages including ‘formats’ (e.g., XML-based ones) or general ‘representation types’ (e.g., *text*). The directed edges (arrows) denote subset relationship for languages in a set-theoretical sense. For instance, language *bnl(text)* corresponds to the concrete textual syntax of BNL. Thus, language *text* can be viewed as the universe for text-based languages. We explain the various languages in the sequel.

Here is an example of a binary number represented as text, i.e., an element of *bnl(text)*:

Text resource *languages/BNL/samples/5comma25.bnl*

101.01

Language *bnl(json)* corresponds to the abstract, tree-based syntax of BNL using JSON for representation; here is the JSON representation of ‘101.01’:

Figure 2 shows basic representation types in YAS and a few more languages related to different aspects of a simple language BNL-Language. The nodes in the figure denote languages including ‘for based ones) or general ‘representation types’ (e.g., *text*). The directe denote subset relationship for languages in a set-theoretical sen language *bnl(text)* corresponds to the concrete textual syntax of BNI *text* can be viewed as the universe for text-based languages. We ex languages in the sequel.

Here is an example of a binary number represented as text, i.e *bnl(text)*:

Text resource *languages/BNL/samples/5comma25.bn*

```
101.01
```

Language *bnl(json)* corresponds to the abstract, tree-based synt JSON for representation; here is the JSON representation of ‘101.01

yas/5comma25.bnl at program · GitHub

Ralf

GitHub, Inc. [US] | https://github.com/softlang/yas/blob/programming17/languages/BNL/samples/5comma25.bnl

This repository Search Pull requests Issues Gist

softlang / yas Unwatch 1 Star 0 Fork 1

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

Branch: programming17 · [yas / languages / BNL / samples / 5comma25.bnl](#) Find file Copy path

rlaemmel Initial commit due to repo reorg 8ae805b on 12 Jun 2016

1 contributor

2 lines (1 sloc) | 7 Bytes Raw Blame History

1 101.01



Branch: [programming17](#) ▾[yas](#) / [languages](#) / [BNL](#) / [samples](#) / [5comma25.bn1](#)**rlaemmel** Initial commit due to repo reorg

1 contributor

2 lines (1 sloc) | 7 Bytes

1 **101.01**

: program x

<https://github.com/softlang/yas/blob/programming17/languages/BNL/samples/5comma25.bn>

/ Search

Pull requests Issues Gist

Issues 0

Pull requests 0

Projects 0

Wiki

Pulse

17 ▾

[yas / languages / BNL / samples / 5comma25.bn](#)

Commit due to repo reorg

7 Bytes

Decimal representation of 101.01

5.25.

A JSON-based AST of *101.01*

```
{  
  "bits": ["one", "zero", "one"],  
  "rest": ["zero", "one"]}  
}
```

Symbolic conversion from 101.01 to 5.25

```
2^(1+1+1-1)+(0+2^(1+1+1-1-1-1))+(0+2^(-1-1)).
```

An ANTLR grammar for binary numbers

```
grammar BnlEbnf;  
@header { package org.softlang; }  
number : bit+ ('.' bit+)? WS? EOF;  
bit : '0' | '1';  
WS : [ \t\n\r]+ ;
```

A BNF for binary numbers

```
[number] number : bits rest ;  
[single] bits : bit ;  
[many] bits : bit bits ;  
[zero] bit : '0' ;  
[one] bit : '1' ;  
[integer] rest : ;  
[rational] rest : '.' bits ;
```

An algebraic signature for binary numbers

```
symbol number: bits × rest → number ;  
symbol single: bit → bits ;  
symbol many: bit × bits → bits ;  
symbol zero: → bit ;  
symbol one: → bit ;  
symbol integer: → rest ;  
symbol rational: bits → rest ;
```

A DCG for binary-to-decimal number of conversion

101.01

$2^{(1+1+1-1)} + (0+2^{(1+1+1-1-1-1)}) + (0+2^{(-1-1)})$.

number(Val1+Val2) → bits(Len1-1, Len1, Val1), rest(Val2).

bits(Pos, 1, Val) → bit(Pos, Val).

bits(Poso, Len1+1, Val1+Val2) → bit(Poso, Val1), bits(Poso-1, Len1, Val2).

bit(_Pos, 0) → ['0'].

bit(Pos, 2^{Pos}) → ['1'].

rest(0) → [].

rest(Val) → [':'], bits(-1, _Len, Val).

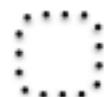
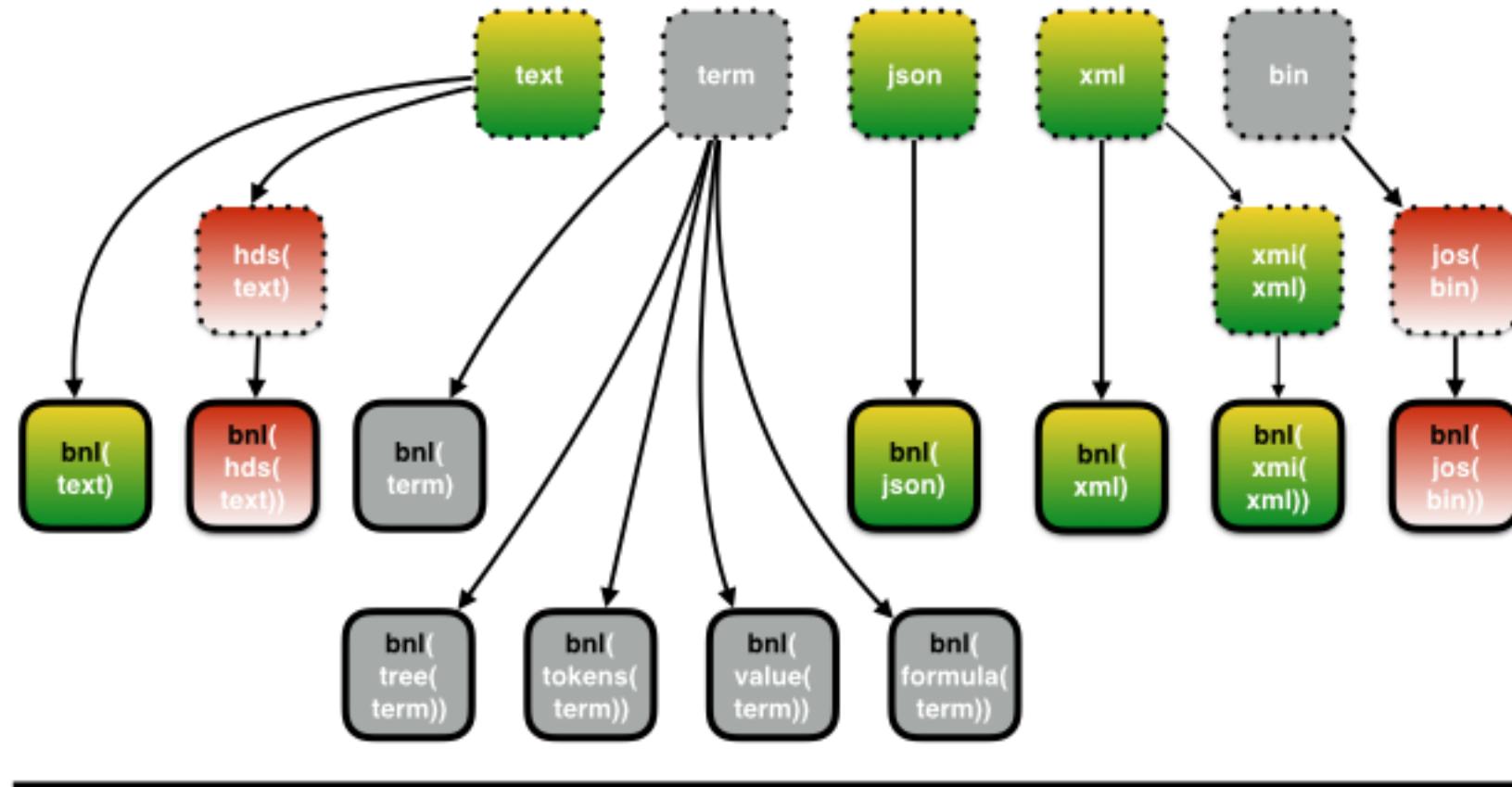
An evaluator

```
2^(1+1+1-1)+(0+2^(1+1+1-1-1-1))+(0+2^(-1-1)).
```

```
evaluate(F, V) ← V is F.
```

5.25.

Languages as types in an SLR



Representation type



Modeled language



Prolog/YAS format

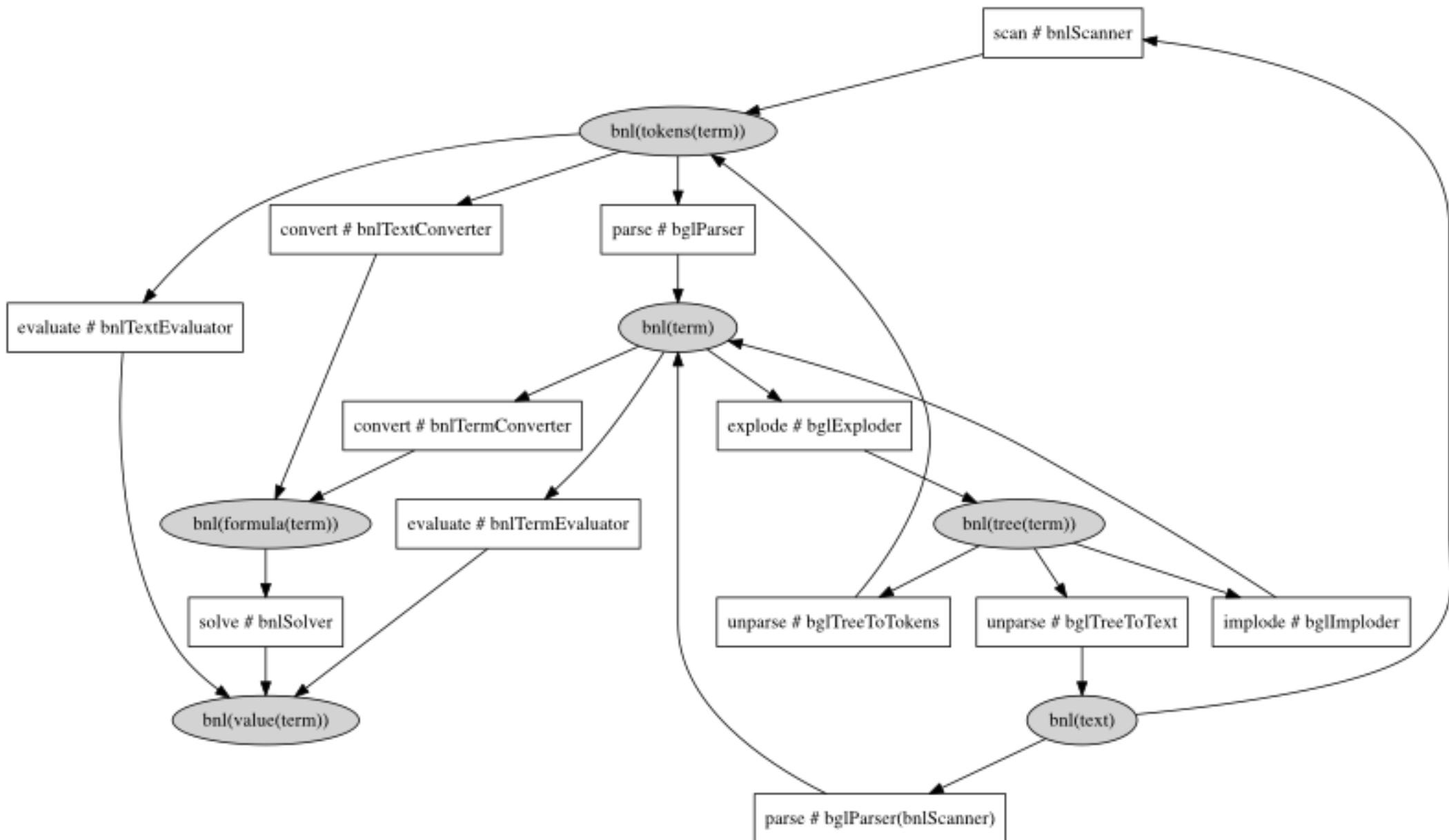


Interchange format

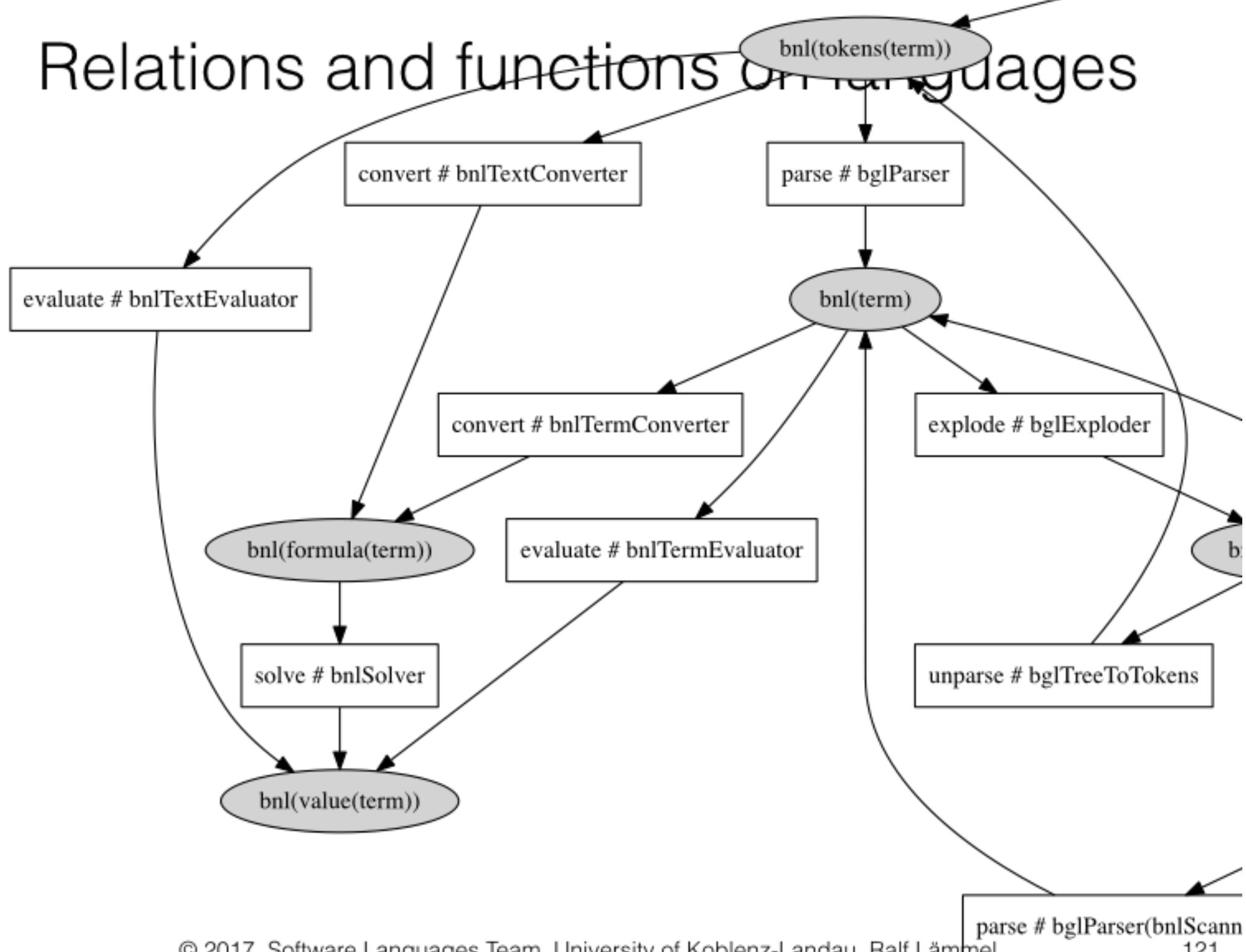


"Proprietary" format

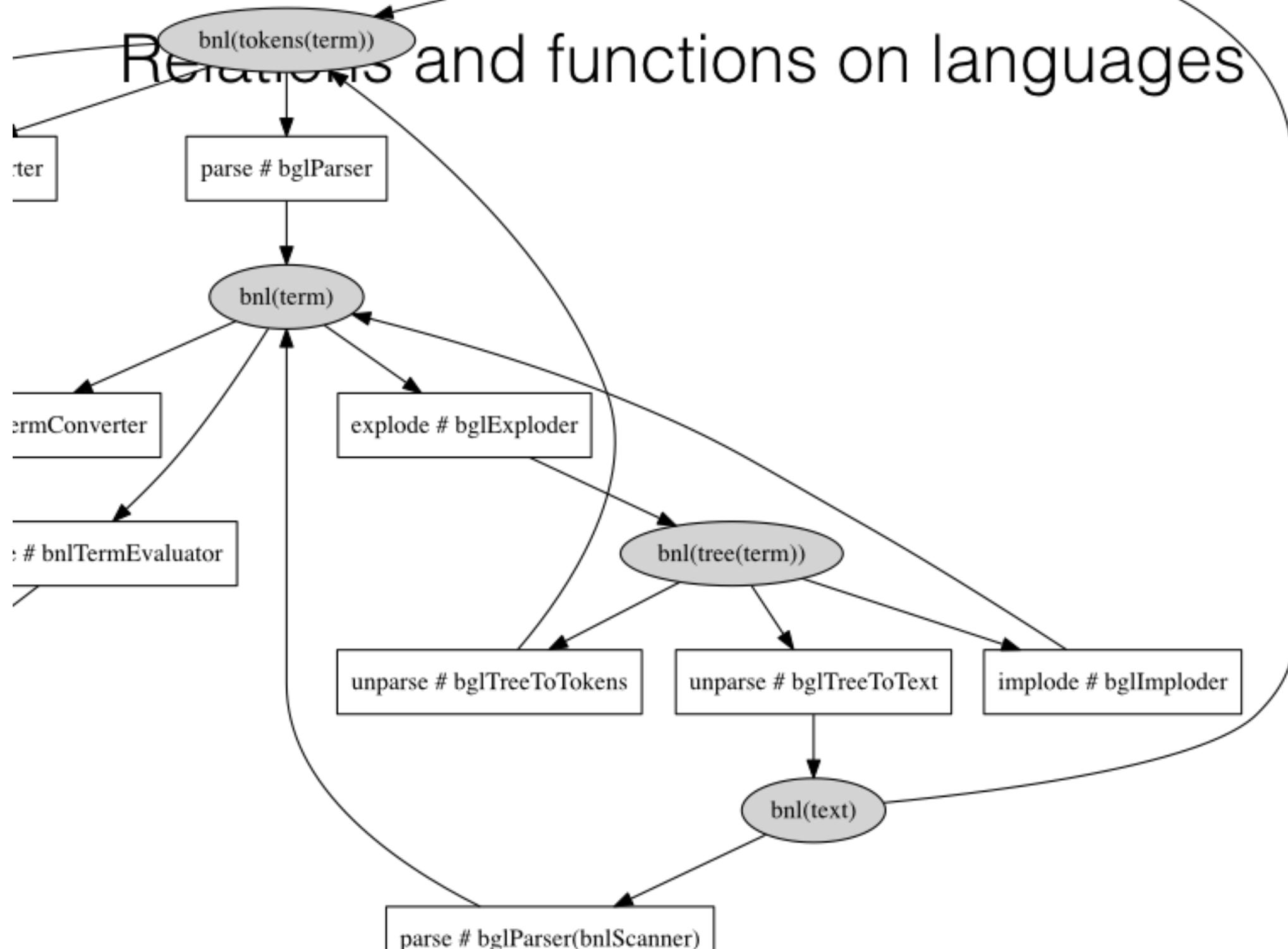
Relations and functions on languages



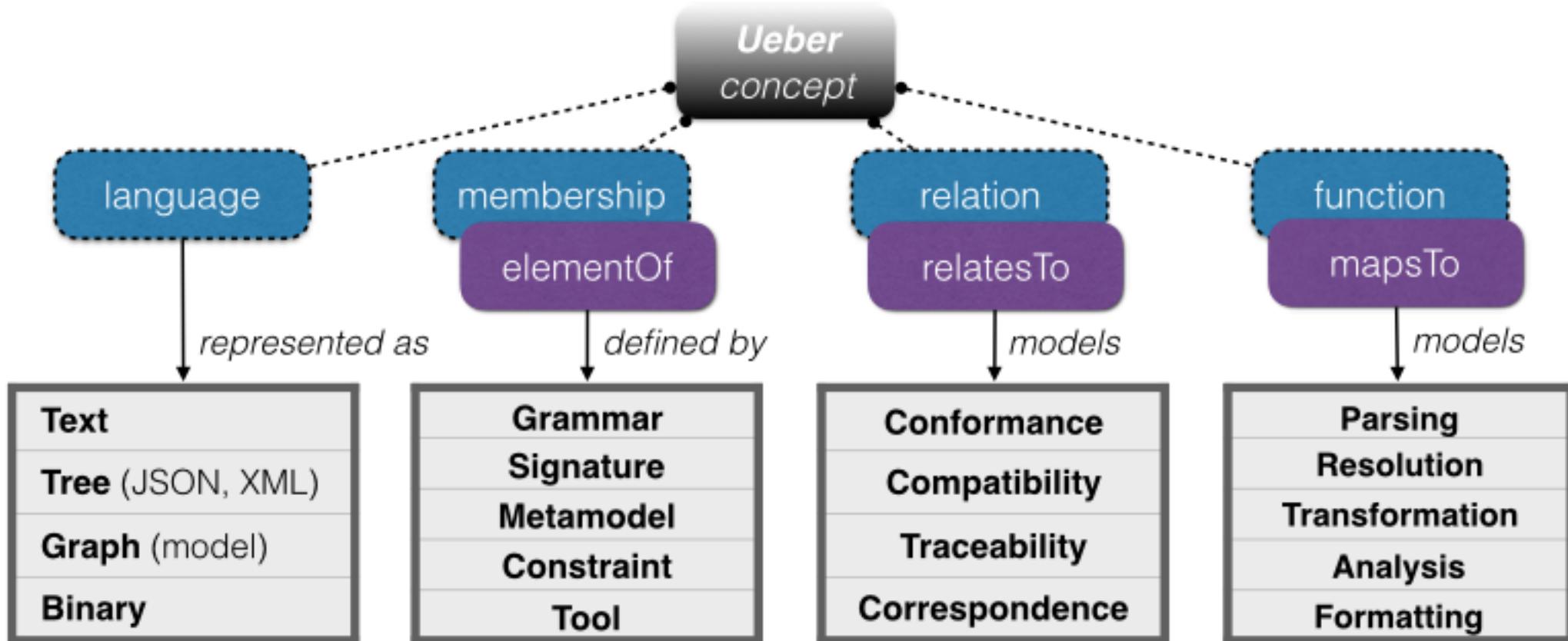
Relations and functions on languages



Relations and functions on languages



ueber's language concepts



Abstract syntax of ueber

```
type model = decl*;
symbol language : lang → decl ;
symbol elementOf : file × lang → decl ;
symbol notElementOf : file × lang → decl ;
symbol membership : lang × goal × file* → decl ;
symbol relation : rela × lang* × goal × file* → decl ;
symbol relatesTo : rela × file* → decl ;
symbol function : func × lang* × lang* × goal × file* → decl ;
symbol mapsTo : func × file* × file* → decl ;
symbol equivalence : lang × goal × file* → decl ;
symbol normalization : lang × goal × file* → decl ;
symbol macro : goal → decl ;
type file = string ;// filenames
type rela = string ;// names of relations
type func = string ;// names of functions
type lang = term ;// names of languages
type goal = term ;// Prolog literals
```

Language declarations

```
language(term).  
language(bnl(term)).
```

Element-of declarations

```
elementOf('languages/BNL/samples/5comma25.bnl', bnl(text)).
```

Membership functionality

```
membership(bnl(text),  
           bglTopDownAcceptor(bnlScanner), ['languages/BNL/cs.term']).
```

Conformance relation(ship)

```
relation(conformsTo, [term, bsl(term)], bslConformance, []).  
relatesTo(conformsTo,  
['languages/BNL/samples/5comma25.term',  
'languages/BNL/as.term']).
```

Parsing function (application)

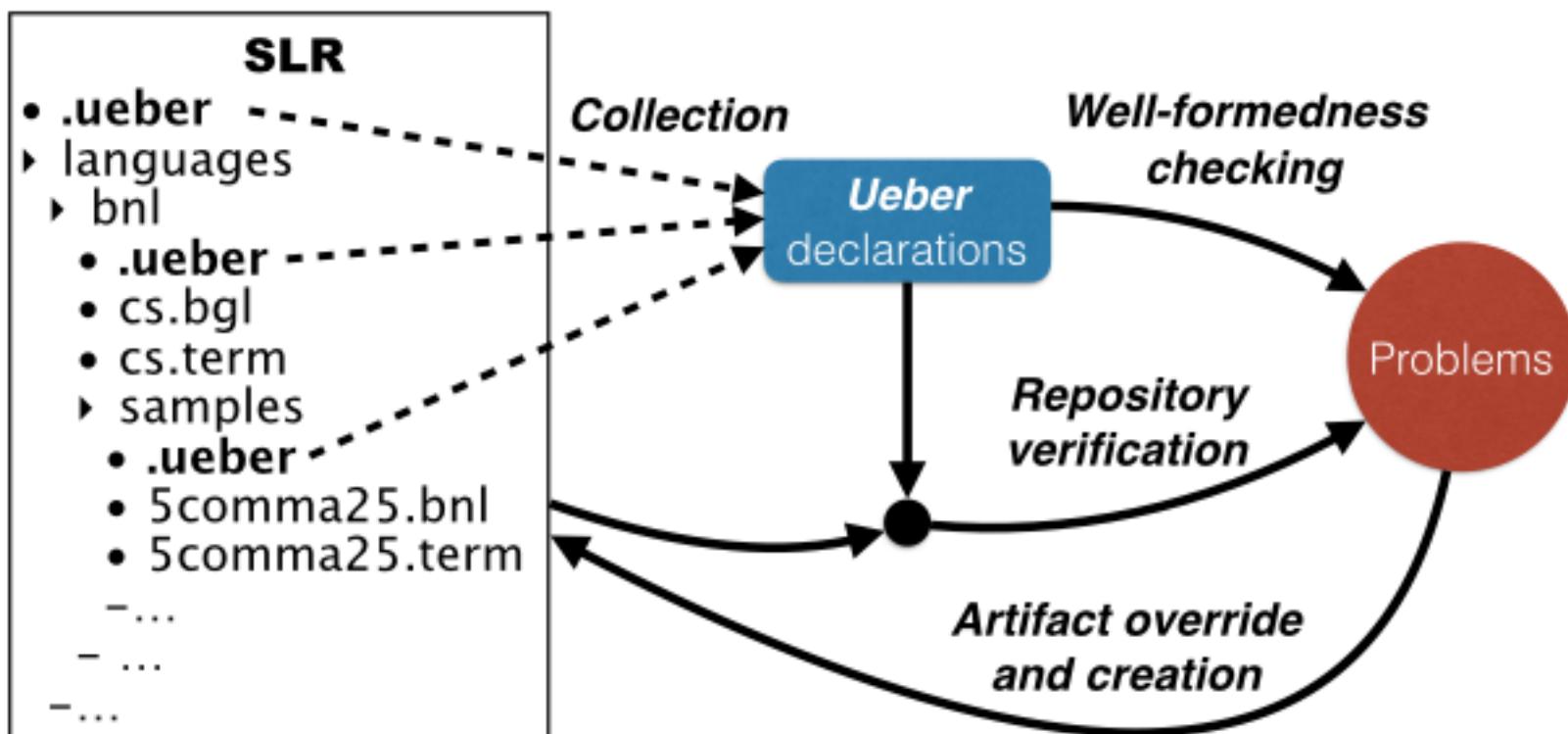
```
function(parse,  
[bnl(text)], [bnl(term)],  
bglTopDownParser(bnlScanner), ['languages/BNL/cs.term']).  
mapsTo(parse,  
['languages/BNL/samples/5comma25.bnl'],  
['languages/BNL/samples/5comma25.term']).
```

A macro for typeful function application

```
fx(Fun,FX,LX,FY,LY) ←  
ueber[  
    elementOf(FX, LX),  
    elementOf(FY, LY),  
    mapsTo(Fun, [FX], [FY]) ].
```

```
[  
    macro(fx(pp, 'text.ppl', ppl(term), 'text.txt', text)),  
    macro(fx(pp, 'vbox.ppl', ppl(term), 'vbox.txt', text)),  
    macro(fx(pp, 'vlist.ppl', ppl(term), 'vbox.txt', text)),  
    ...  
].
```

ueber's semantics



Output produced by the ueber processor

'Breaking changes to a language processor'

- Baseline *languages/PPL/tests/hseplist.txt*: **disagreeing**.
 - *mapsTo(pp,[languages/PPL/tests/hseplist.ppl],[....txt])*: **UNVERIFIED**.
-

'Development of a new test case'

- Baseline *languages/PPL/tests/indent.txt*: **missing**.
 - *elementOf(languages/PPL/tests/indent.txt,text)*: **UNVERIFIED**.
 - *mapsTo(pp,[languages/PPL/tests/indent.ppl],[....txt])*: **UNVERIFIED**.
-

'Modeling a new relationship'

- Overload *evaluate*: (*[languages/BNL/samples/5comma25.bnl]*) -> (*[....value]*): **missing**.
- *mapsTo(evaluate,[languages/BNL/samples/5comma25.bnl],[....value])*: **NOT OK**.
- *mapsTo(evaluate,[languages/BNL/samples/5comma25.bnl],[....value])*: **UNVERIFIED**.

ueber's FFI

	Prolog	Haskell	Java	Python
Implement relations & functions	Predicates	Main functions	Main methods	Scripts
Code location	Module auto loading	Automated module search path	Automated CLASSPATH	Automated PYTHONPATH
Compilation	N/A	On the fly	On the fly	N/A

■ Figure UEBER's integrated compile- and run-time.

	Prolog	Haskell	Java	Python
Parse text	DCG, ...	Parsec, ...	ANTLR, ...	ANTLR, ...
Represent trees	Prolog terms JSON, XML, ...	read/show conversion, JSON, XML	JSON, XML	str/repr conversion, JSON, XML
Represent graphs	Prolog terms	N/A	Serializable objects, XMI, ...	Serializable objects, ...

■ Figure Representation across different implementation languages.

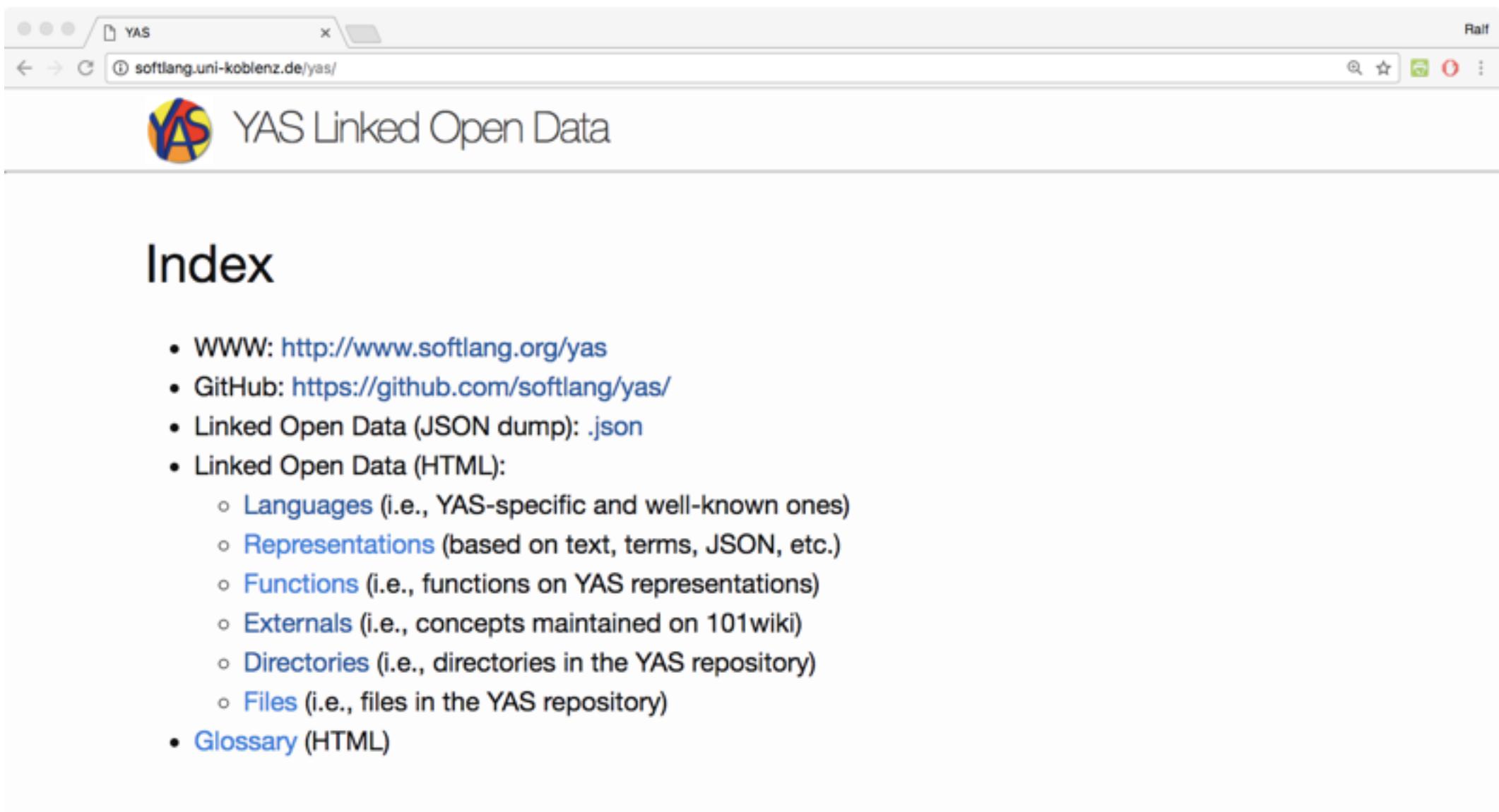
Advanced aspect of the YAS SLR

- Semantic annotations and LOD
- Incremental building
- Test-data generation
- Migration of a project to *ueber*
- Integration with version control
- Infer *ueber* declarations based on conventions
- A useful model of language variants



Linked Open Data for YAS

Demo of YAS LOD

A screenshot of a web browser window titled "YAS". The address bar shows the URL "softlang.uni-koblenz.de/yas/". The page content is titled "YAS Linked Open Data" and features a logo consisting of three overlapping colored circles (blue, yellow, red) followed by the acronym "YAS". Below the title, the word "Index" is prominently displayed. A bulleted list provides links to various resources:

- WWW: <http://www.softlang.org/yas>
- GitHub: <https://github.com/softlang/yas/>
- Linked Open Data (JSON dump): [.json](#)
- Linked Open Data (HTML):
 - Languages (i.e., YAS-specific and well-known ones)
 - Representations (based on text, terms, JSON, etc.)
 - Functions (i.e., functions on YAS representations)
 - Externals (i.e., concepts maintained on 101wiki)
 - Directories (i.e., directories in the YAS repository)
 - Files (i.e., files in the YAS repository)
- [Glossary](#) (HTML)

The screenshot shows a web browser window with the title bar "YAS languages". The address bar contains the URL "softlang.uni-koblenz.de/yas/languages.html". The page content features the YAS logo (a stylized "YAS" in blue, yellow, and red) and the text "YAS Linked Open Data".

Languages: YAS index

Language	YAS-specific?	Headline
ANTLR		The grammar language of the ANTLR technology
ASL	✓	A simple language for algebraic specification
BAL	✓	A trivial assembly language
BCL	✓	A format for CSTs for parsing with BGL grammars
BFPL	✓	A trivial first-order functional programming language
BGL	✓	A BNF-like notation for context-free grammars
BIPL	✓	A trivial imperative programming language
BL	✓	A trivial language for buddy relationships
BML	✓	A trivial machine language
BNL	✓	A trivial language for binary numbers

The screenshot shows a web browser window with the title "YAS language BIPL". The address bar contains the URL "softlang.uni-koblenz.de/yas/languages/bipl.html". The page content is as follows:

Language *BIPL*

GitHub

<https://github.com/softlang/yas/tree/master/languages/BIPL>

Expansion

Basic Imperative Programming Language

Headline

A trivial imperative programming language

Details

BIPL supports primitive types for integer and Boolean values; it does not support input, output, and procedures. BIPL may be viewed as small fragment of C.

The screenshot shows a web browser window with the following details:

- Title Bar:** YAS language BIPL
- Address Bar:** softlang.uni-koblenz.de/yas/languages/bipl.html
- User:** Ralf

The main content area displays the following sections:

Properties

- **this** *relatesTo* Language:C
- **this** *facilitates* Imperative programming
- **this** *subsetOf* Language:E IPL
- **this** *embeds* Language:EL

Representations

- bipl(text)
- bipl(term)

Components

- languages/BIPL/cs.egl (*Context-free grammar*)
- languages/BIPL/ls.egl (*Context-free grammar*)
- languages/BIPL/as.esl (*Algebraic signature*)
- languages/BIPL/.hinzu (*Annotation*)
- languages/BIPL/README.md (*Markup*)

The screenshot shows a web browser window with the following details:

- Title Bar:** YAS language BIPL
- Address Bar:** softlang.uni-koblenz.de/yas/languages/bipl.html
- User:** Ralf
- Content Area:** The page title is "Components". Below it is a bulleted list of components:

- [languages/BIPL/cs.egl](#) (*Context-free grammar*)
- [languages/BIPL/ls.egl](#) (*Context-free grammar*)
- [languages/BIPL/as.esl](#) (*Algebraic signature*)
- [languages/BIPL/.hinzu](#) (*Annotation*)
- [languages/BIPL/README.md](#) (*Markup*)
- [languages/BIPL/biplAbstract.pro](#) (*Logic program*)
- [languages/BIPL/.ueber](#) (*Megamodel*)
- [languages/BIPL/Haskell](#) (*Language implementation*)
- [languages/BIPL/Haskell/Language/BIPL](#) (*Language implementation*)
- [languages/BIPL/Haskell/Language/BIPL/Algebra](#) (*Language implementation*)
- [languages/BIPL/Haskell/Language/BIPL/Analysis](#) (*Program analysis*)
- [languages/BIPL/Haskell/Language/BIPL/CS](#) (*Interpreter*)
- [languages/BIPL/Haskell/Language/BIPL/DS](#) (*Interpreter*)
- [languages/BIPL/Haskell/Language/BIPL/Goto](#) (*Interpreter*)
- [languages/BIPL/Haskell/Language/BIPL/MonadicAlgebra](#) (*Language implementation*)
- [languages/BIPL/Haskell/Language/BIPL/Rename](#) (*Transformation*)

The screenshot shows a web browser window with the following details:

- Title Bar:** YAS file languages/BIPL/cs.egl
- Address Bar:** softlang.uni-koblenz.de/yas/files/languages-BIPL-cs.egl.html
- User:** Ralf

The main content area displays the following information:

YAS file *languages/BIPL/cs.egl*

GitHub

<https://github.com/softlang/yas/tree/master/languages/BIPL/cs.egl>

Representations

- `egl(text)`

Properties

- **this** *instanceOf* Context-free grammar
- **this** *defines* Context-free syntax

Dependencies

- `languages/BIPL: mapsTo`
 - Function `parse`
 - File *languages/BIPL/cs.egl*
 - File *languages/BIPL/cs.term*

yas/cs.egl at master · softlang/yas

Ralf

GitHub, Inc. [US] https://github.com/softlang/yas/blob/master/languages/BIPL/cs.egl

softlang / yas

Unwatch 2 Star 1 Fork 2

Code Issues 0 Pull requests 0 Projects 0 Wiki Settings Insights

Branch: master · yas / languages / BIPL / cs.egl Find file Copy path

 rlaemmel Initial commit due to repo reorg 0ae805b on Jun 12 2016

1 contributor

24 lines (22 sloc) | 730 Bytes Raw Blame History

```
1 // Statements
2 [skip] stmt : ';' ;
3 [assign] stmt : name '=' expr ';' ;
4 [block] stmt : '{' { stmt }* '}' ;
5 [if] stmt : 'if' '(' expr ')' stmt { 'else' stmt }? ;
6 [while] stmt : 'while' '(' expr ')' stmt ;
7
8 // Expressions
9 [or] expr : bexpr { '||' expr }? ;
10 [and] bexpr : cexpr { '&&' bexpr }? ;
11 [lt] cexpr : aexpr { '<' aexpr }? ;
12 [leq] cexpr : aexpr { '<=' aexpr }? ;
13 [eq] cexpr : aexpr { '==' aexpr }? ;
```

Hinzu

annotation language

Metadata for BIPL language

```
[  
  l([  
    id(bipl),  
    name('BIPL'),  
    expansion('Basic Imperative Programming Language'),  
    headline('A trivial imperative programming language'),  
    details('BIPL supports primitive types for integer and Boolean values; it does not  
support input, output, and procedures. BIPL may be viewed as small fragment of C.'),  
    relatesTo(extern('Language:C')),  
    facilitates(extern('Imperative programming')),  
    embeds(intern(el)),  
    subsetOf(intern(eipl))  
  ]),  
  r([  
    id(bipl(text)),  
    extension(bipl),  
    representationOf(bipl),  
    reason(succeed, [instanceOf(extern('Imperative program'))])  
  ])  
]
```

Abstract syntax of *Hinzu*

```
// Grouping of metadata items
type model = decl* ;
symbol l : item* -> decl ; // Languages
symbol r : item* -> decl ; // Language representations
symbol f : item* -> decl ; // Files in the repository
symbol d : item* -> decl ; // Directories in the repository

// General metadata items
symbol id : id -> item ; // Some decls need an id
symbol name : string -> item ; // Name of an entity such as a language
symbol expansion : string -> item ; // Expansion in case the name is an acronym
symbol headline : string -> item ; // One liner explanation of the entity
symbol details : string -> item ; // An optional paragraph for explanation of the entity

// Semantic links
symbol instanceOf : link -> item ; // An entity being an instance of a classifier
symbol sameAs : link -> item ; // Identity link
symbol similarTo : link -> item ; // Weak identity link
symbol relatesTo : link -> item ; // Degenerated identity link
symbol uses : link -> item ; // An entity uses another entity
symbol facilitates : link -> item ; // A language facilitates a concept
```

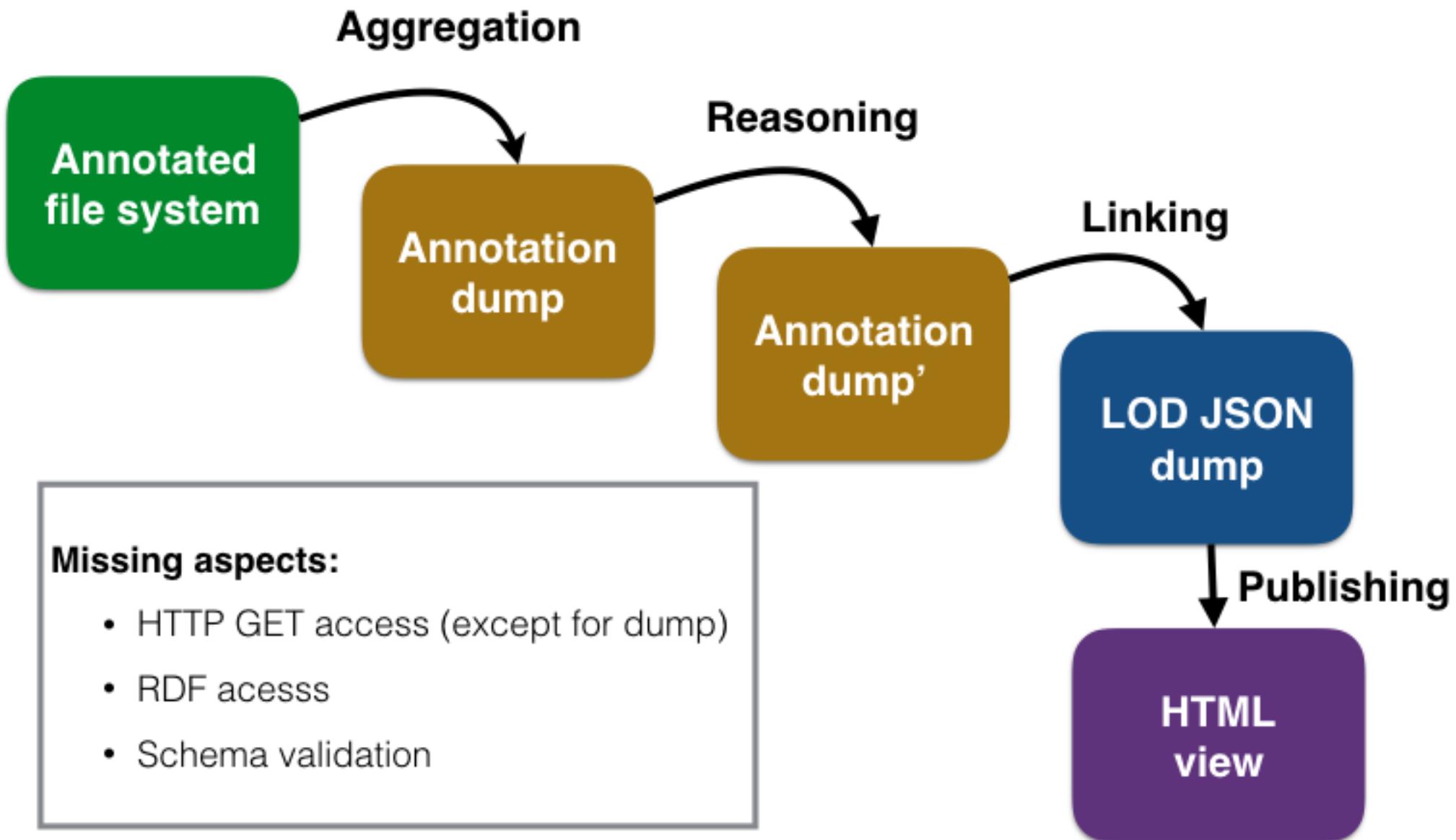
```
symbol facilitates : link -> item ; // A language facilitates a concept
symbol defines : link -> item ; // A file or directory defines a concept
symbol implements : link -> item ; // A file or directory implements a language
symbol supports : link -> item ; // An entity supports some entity
symbol subsetOf : link -> item ; // A language is a subset of another language
symbol supersetOf : link -> item ; // A language is a subset of another language
symbol embeds : link -> item ; // A language embeds another language
symbol dependsOn : link -> item ; // A language depends on another language
symbol linksTo : link -> item ; // Non-semantical link

// Representation-specific items
symbol extension : string -> item ; // Identify filename extension for a representation
symbol processor : goal -> item ; // Identify processor for a representation
symbol representationOf : id -> item ; // Associate representation with language
symbol reason : goal # item+ -> item ; // Guarded items for elements of representation

// Links
symbol intern : id -> link ; // Internal references
symbol extern : reluri -> link ; // External references

// Some synonyms for clarity's sake
type id = term ; // Must be string except for representations
type goal = term ; // Prolog literals
type reluri = string ; // Relative URIs (relative to 101wiki)
```

The LOD pipeline

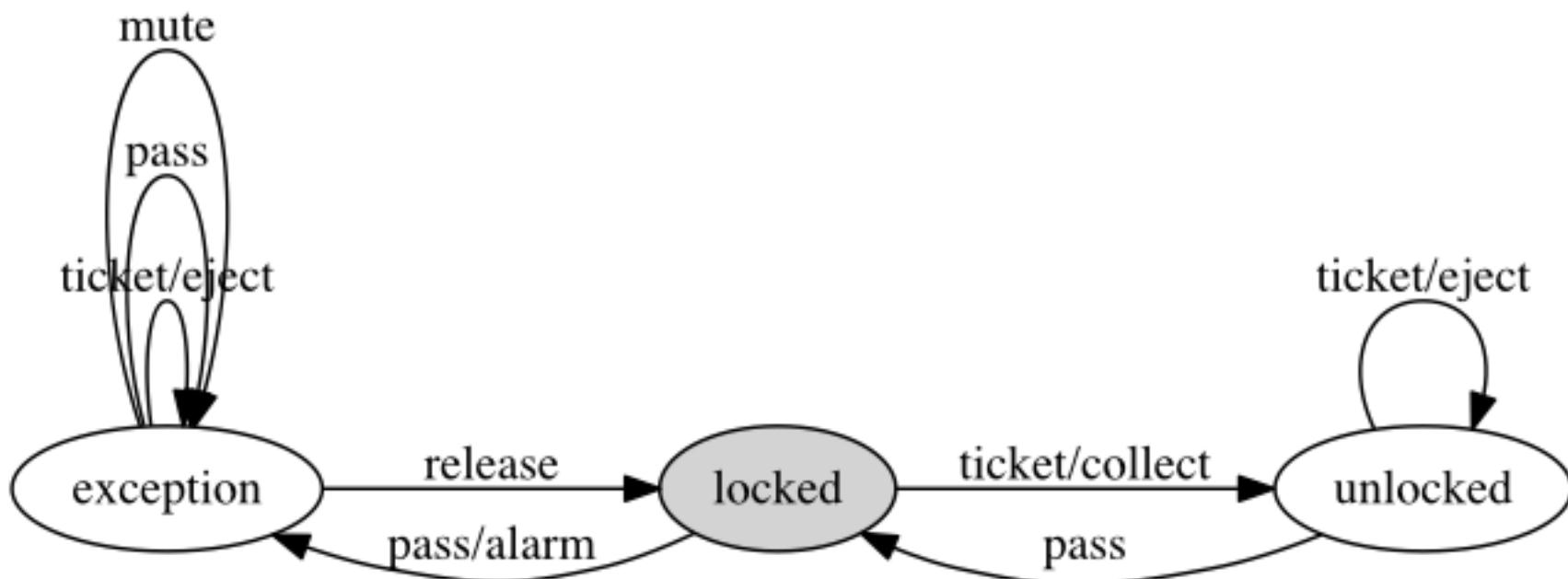


Metaprogramming Library (MetaLib)

Acknowledgement: This is joint work with Simon Schauß, Johannes Härtel, Kevin Klein, Wojciech Kwasnik (all SoftLang), and Thorsten Berger (Chalmers).

The Finite State Machine Language (FSML)

Visual FSML notation



Textual FSML notation

```
initial state locked {
    ticket/collect -> unlocked;
    pass/alarm -> exception;
}
state unlocked {
    ticket/eject;
    pass -> locked;
}
state exception {
    ticket/eject;
    pass;
    mute;
    release -> locked;
}
```

Textual FSML notation

```
initial state locked {
    ticket/collect -> unlocked;
    pass/alarm -> exception;
}
state unlocked {
    ticket/eject;
    pass -> locked;
}
state exception {
    ticket/eject;
    pass;
    mute;
    release -> locked;
}
```

Grammar of textual notation

```
fsm : {state}* ;
state : {'initial'}? 'state' stateid '{' {transition}* '}';
transition : event {'/' action}? {'->' stateid}? ';';
stateid : name ;
event : name ;
action : name ;
```

Signature of abstract syntax

```
type fsm = state* ;
type state = initial × stateid × transition* ;
type initial = boolean ;
type transition = event × action? × stateid ;
type stateid = string ;
type event = string ;
type action = string ;
```

Metamodel of abstract syntax

```
class fsm { part states : state* ; }
class state {
    value initial : boolean ;
    value stateid : string ;
    part transitions : transition* ;
}
class transition {
    value event : string ;
    value action : string? ;
    reference target : state ;
}
```

Small-step operational semantics

- | | |
|--|--------------------|
| $\langle \dots, \langle b, x, \langle \dots, \langle e, \langle a \rangle, x' \rangle, \dots \rangle \rangle, \dots \rangle \vdash \langle x, e \rangle \rightarrow \langle x', \langle a \rangle \rangle$ | [action] |
| $\langle \dots, \langle b, x, \langle \dots, \langle e, \langle \rangle, x' \rangle, \dots \rangle \rangle, \dots \rangle \vdash \langle x, e \rangle \rightarrow \langle x', \langle \rangle \rangle$ | [no-action] |

Well-formedness (a violation thereof)

```
initial state stateA { event1/action1 -> stateB; }
state stateB { }
state stateC { }
```

Generated C code

```
enum State {LOCKED,UNLOCKED,EXCEPTION,UNDEFINED};  
enum State initial = LOCKED;  
enum Event {TICKET,RELEASE,MUTE,PASS};  
void alarm() {}  
void eject() {}  
void collect() {}  
enum State next(enum State s, enum Event e) {  
    switch(s) {  
        case LOCKED:  
            switch(e) {  
                case TICKET: collect(); return UNLOCKED;  
                case PASS: alarm(); return EXCEPTION;  
                default: return UNDEFINED;  
            }  
        case UNLOCKED: ...  
        case EXCEPTION: ...  
        default: return UNDEFINED;  
    }  
}
```

MetaLib at github

The screenshot shows the GitHub repository page for 'softlang/metalib'. The repository name is 'softlang / metalib'. It has 109 commits, 3 branches, 0 releases, and 3 contributors. The latest commit was 13 seconds ago. The repository URL is <https://softlang.github.io/metalib>. The repository is public and has 5 watchers.

A library of metaprogramming experiments <https://softlang.github.io/metalib> Edit

metaprogramming Manage topics

109 commits 3 branches 0 releases 3 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

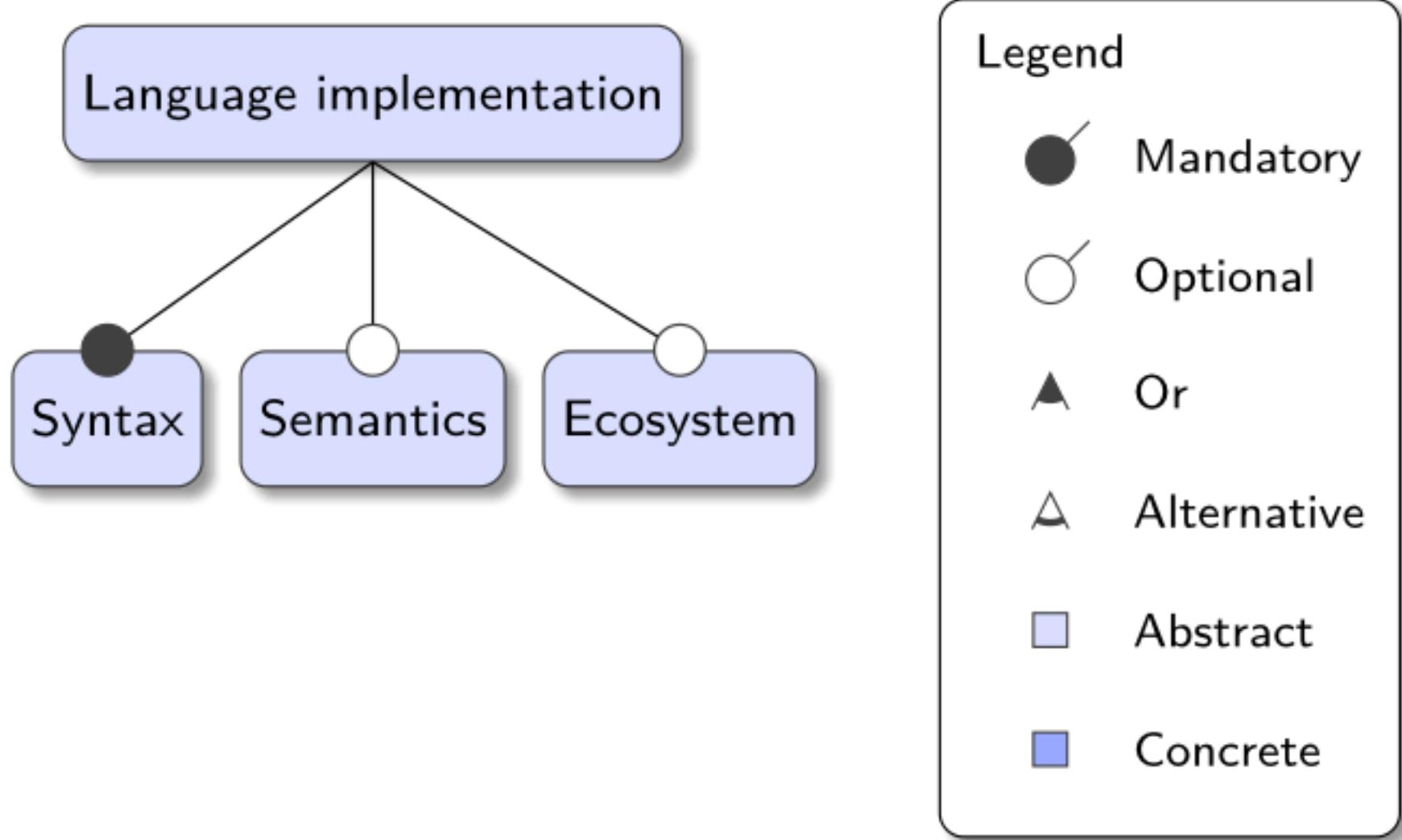
Author	Commit Message	Time Ago
sschauss [docs]	update haskell model to new structure	Latest commit 8725b13 13 seconds ago
docs	[docs] update haskell model to new structure	12 seconds ago
emf/fsml	changes 1234DD	10 days ago
haskell	[haskell] simplify run function	3 months ago
mps	[mps] rename structure input to event	4 months ago

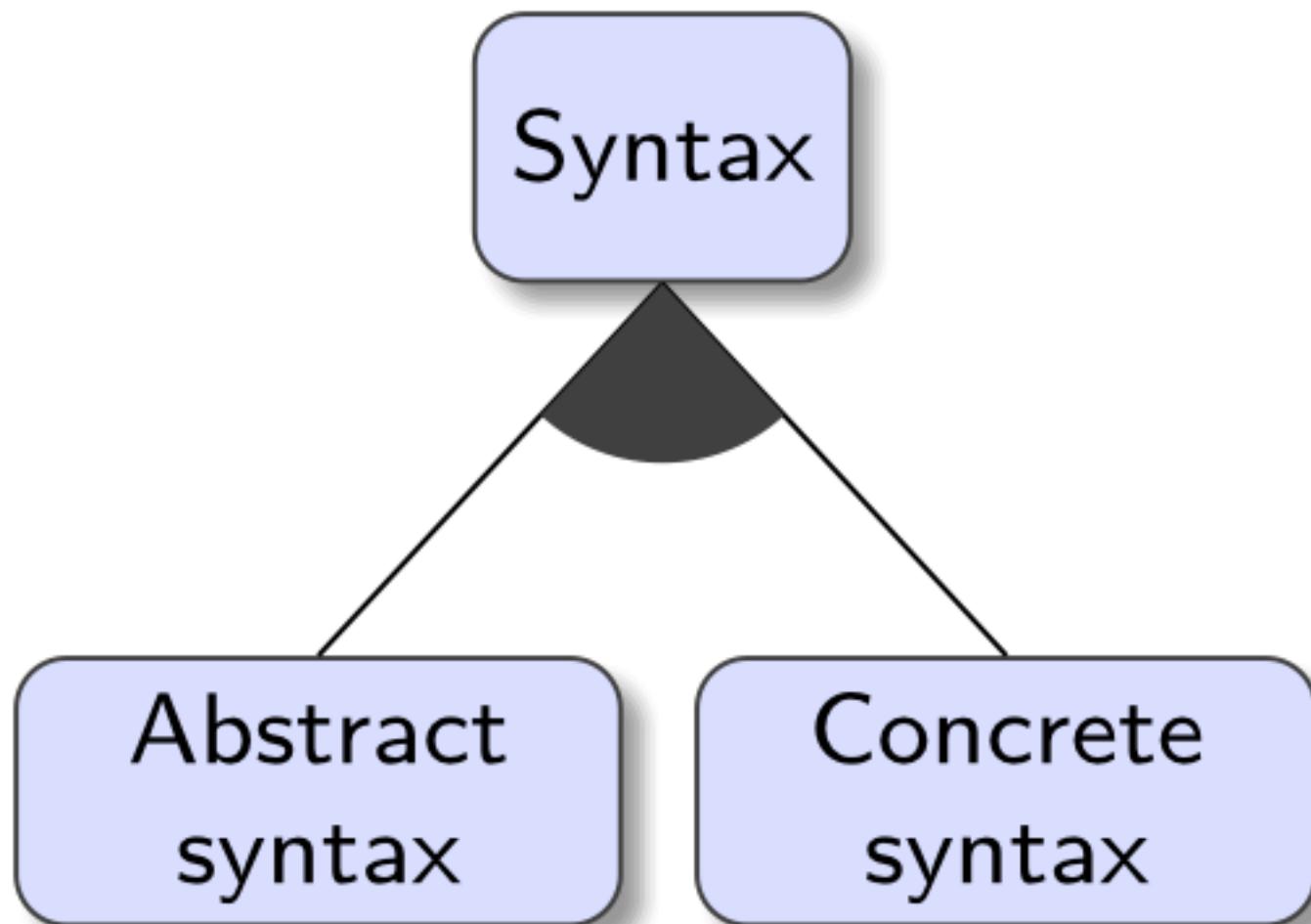
MetaLib at <https://softlang.github.io/metalib/>

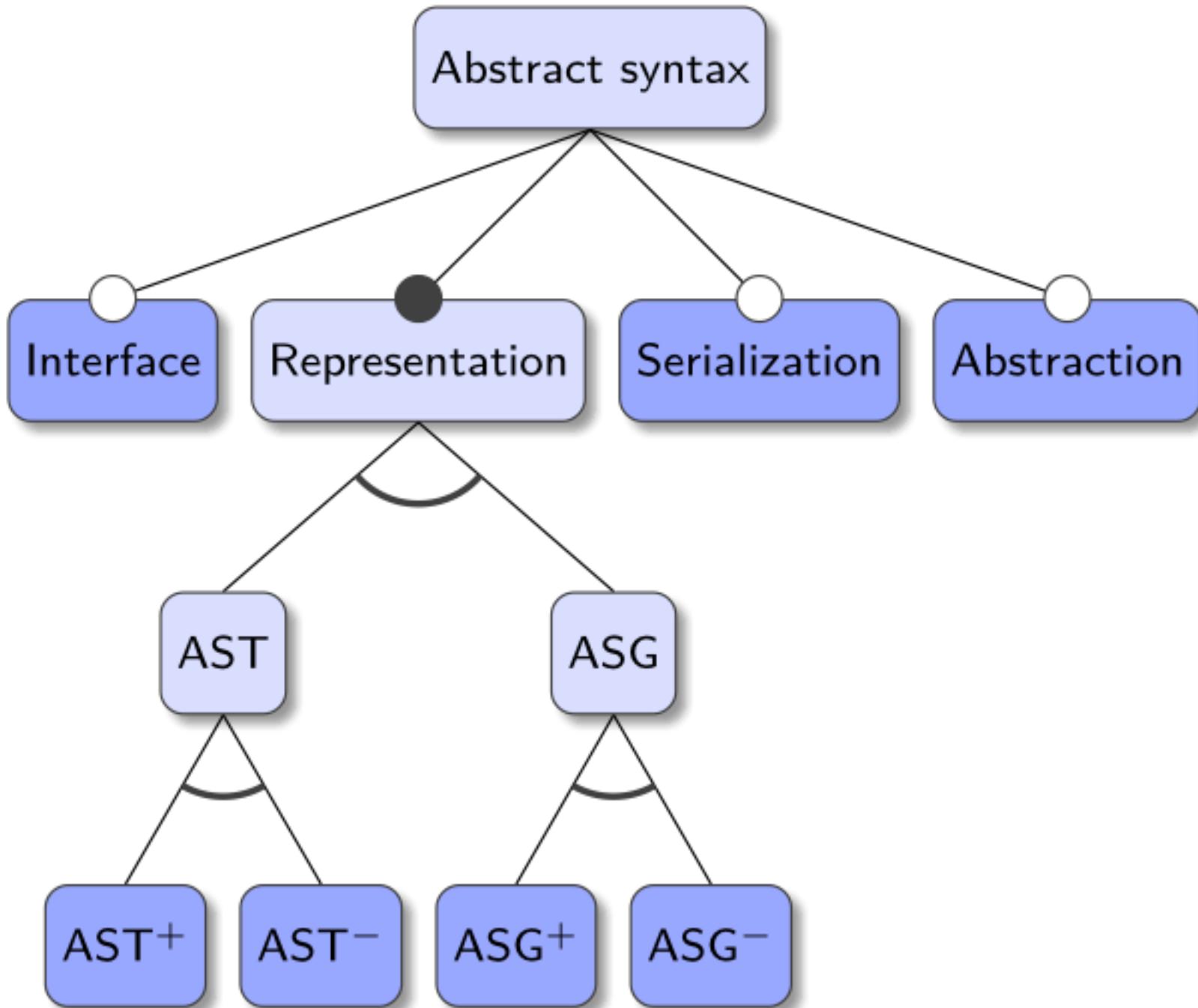
The screenshot shows a web browser window titled "metadocs" with the URL <https://softlang.github.io/metalib/javaExternal.html>. The page has a header "metalib" and a main title "External DSL style with ANTLR and Java". Below the title is a horizontal navigation bar with five colored sections: Perspectives (dark green), Features (green), Languages (yellow), Technologies (orange), and Concepts (red). The Perspectives section contains "meta", "object", and "test". The Features section contains "AST+", "Abstraction", "Concrete syntax", "Scanning", "Text-to-AST", and "Text-to-CST". The Languages section contains "ANTLR4" and "Java". The Technologies section contains "ANTLR4" and "JUnit". The Concepts section contains "Boilerplate code", "External DSL", "Functional constructor", "Getter", "Listener", "Parser generation", and "Setter". Below this is a search bar with the placeholder "Search metalib...". A modal dialog is open, showing a smaller navigation bar with "Perspectives" (dark green), "Features" (green), "Languages" (yellow), "Technologies" (orange), and "Concepts" (red). The "Features" section of the dialog is active, showing "object" and "Concrete syntax". The "Languages" section of the dialog is active, showing "sample.fsm". The "Languages" section of the main page is also active, showing "sample.fsm". The content area of the dialog displays the following state transition diagram:

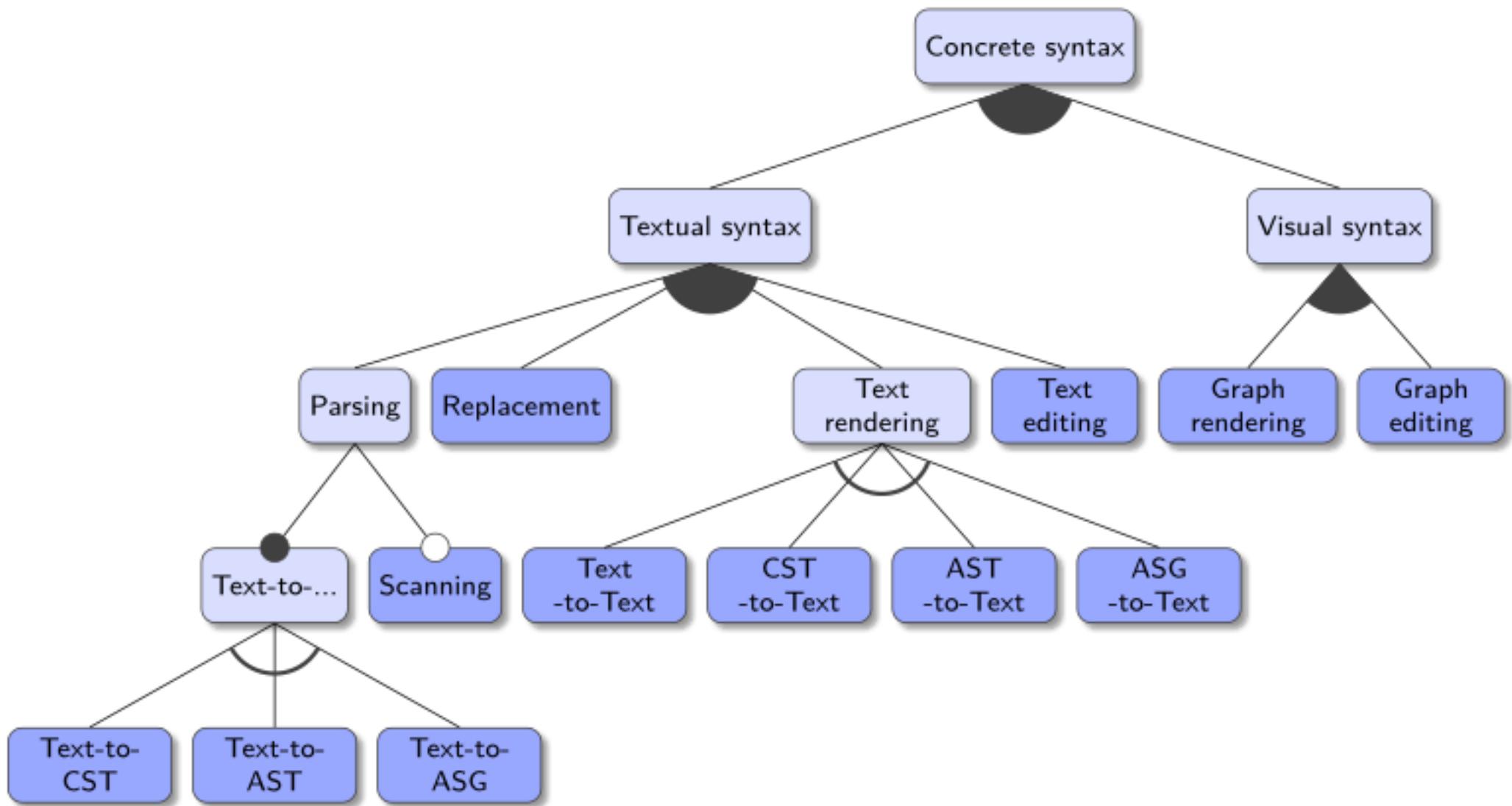
```
initial state locked {
    ticket/collect -> unlocked;
    pass/alarm -> exception;
}
state unlocked {
    ticket/eject;
    pass -> locked;
}
```

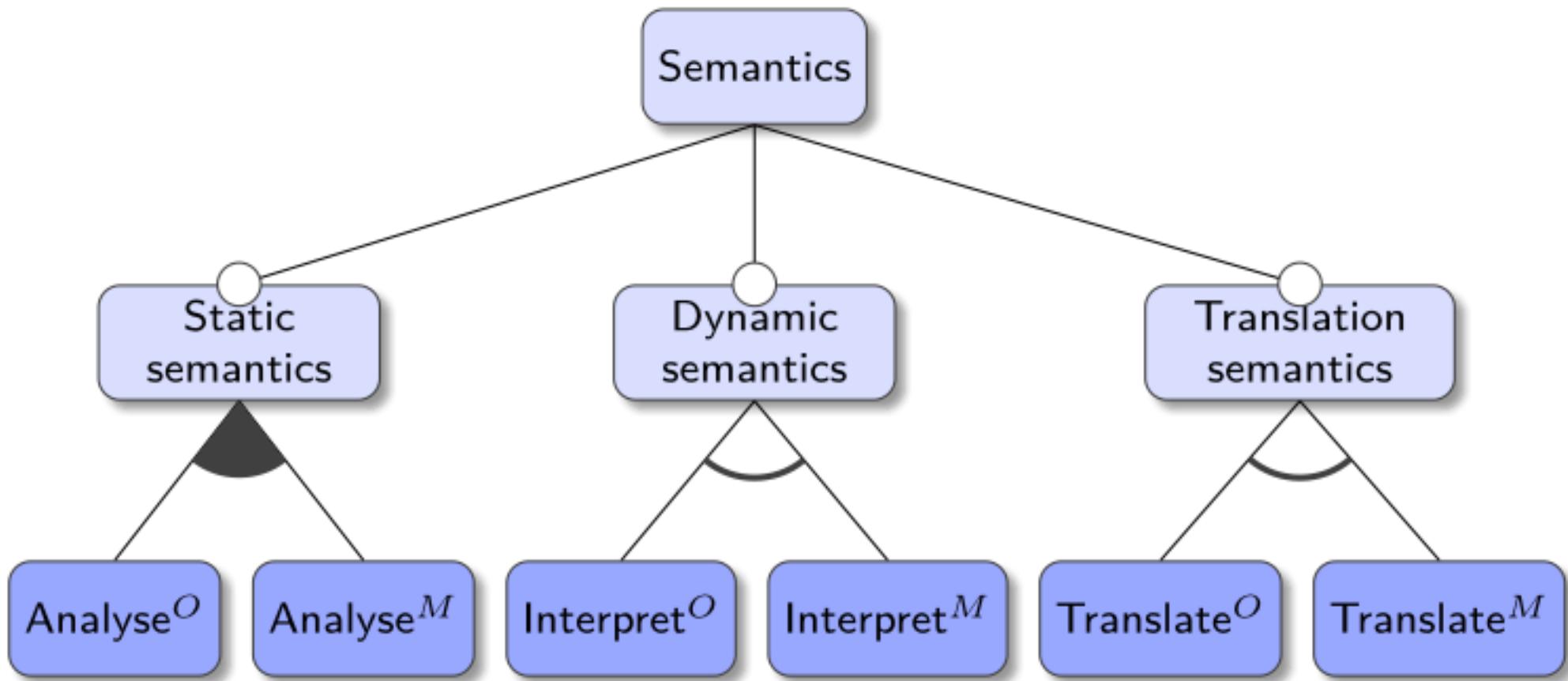
Feature model for metaprogramming implementation of FSML DSL

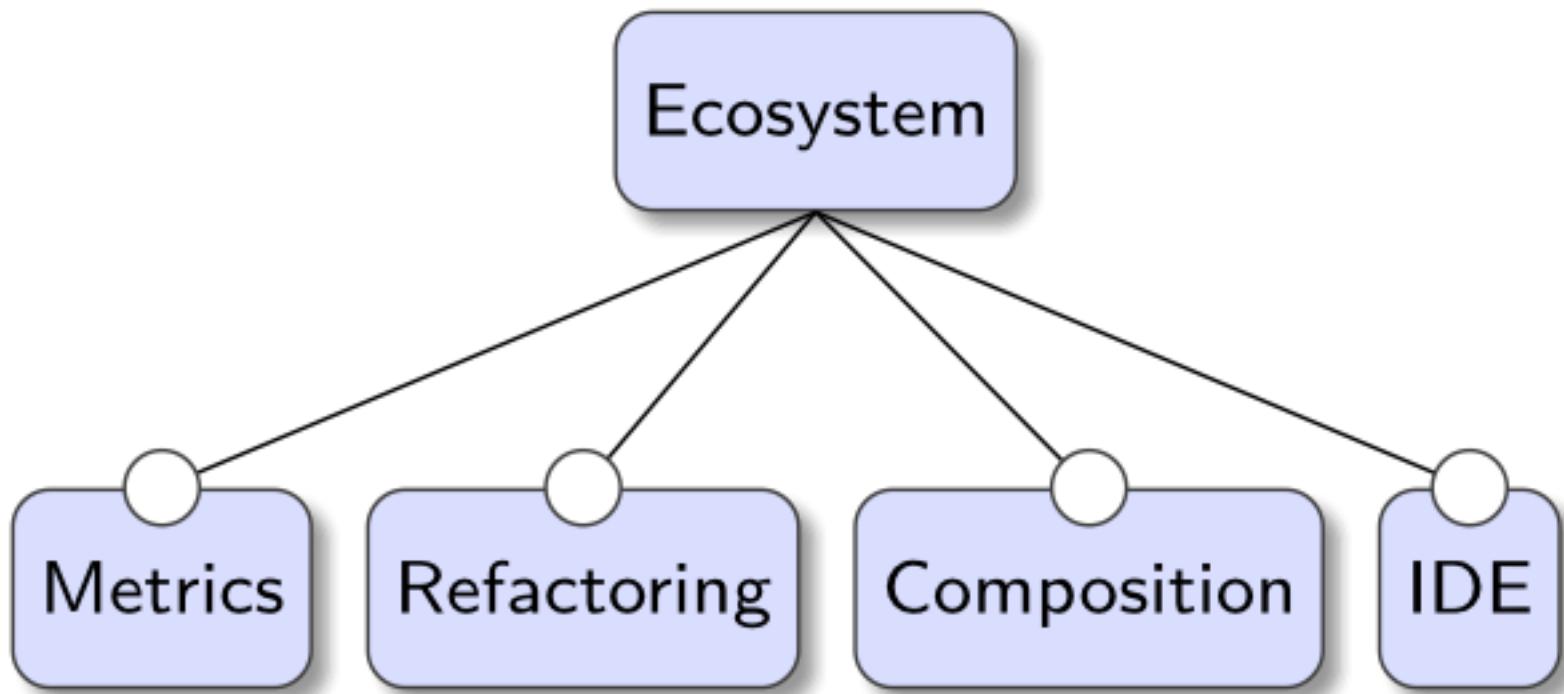












Model-based documentation

The screenshot shows a GitHub repository page for `softlang / metalib`. The URL in the address bar is `https://github.com/softlang/metalib/blob/master/docs/models/pythonExternal.json`. The repository has 5 stars, 0 forks, and 0 issues. The code tab is selected, showing the contents of `pythonExternal.json`. The JSON file contains the following data:

```
1 {  
2   "name": "pythonExternal",  
3   "headline": "Python external",  
4   "sections": [  
5     {  
6       "features": [  
7         "Concrete syntax"  
8       ],  
9       "perspective": "object",  
10      "languages": [],  
11      "type": "component",  
12      "concepts": [  
13        "External DSL"  
14      ]  
16    }  
17  ]  
18}  
19
```

Metamodel of documentation

```
// Documentation of contributions
class document {
    value name : string; // The name of the contribution
    value headline : string; // A one-liner explanation
    value baseuri : string; // Base URI for links
    part sections : section+; // Sections of the documentation
}

// Sections in a documentation
class section {
    value headline : string?; // Optional one-liner explanation
    part perspectives : perspective+; // Perspective of section
    value features : string+; // Features addressed by section
    value languages : string*; // Languages used
    value technologies : string*; // Technologies used
    value concepts : string*; // Concepts used
    part artifacts : artifact+; // Artifacts to be shown
}

// Perspectives of documentation
class perspective {
    value title : string; // Title of the perspective
    value description : string; // Description of the perspective
    value url : string; // URL of the perspective
}
```

```
// Perspectives of documentation
abstract class perspective { }

// Metaprogram, e.g., grammar or interpreter
class meta extends perspective { }

// Object program in different representations
class object extends perspective { }

// Data other than object programs, e.g., input
class data extends perspective { }

// Build-step for functionality
class build extends perspective { }

// Test for functionality
class test extends perspective { }

// Capture of behavior or appearance
class capture extends perspective { }
```

```
// Artifacts for projection
abstract class artifact {
    value link : string; // A relative URI
    value format : string; // MIME-like format type
}
```

```
// Nothing to show
class none extends artifact { }

// All to show
```

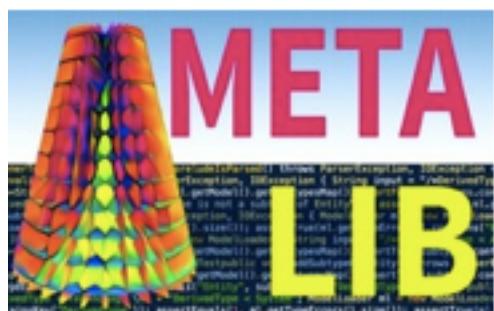
```
class data extends perspective { }
// Build-step for functionality
class build extends perspective { }
// Test for functionality
class test extends perspective { }
// Capture of behavior or appearance
class capture extends perspective { }

// Artifacts for projection
abstract class artifact {
    value link : string; // A relative URI
    value format : string; // MIME-like format type
}
// Nothing to show
class none extends artifact { }
// All to show
class all extends artifact { }
// A specific line range to show
class some extends artifact {
    value from : integer;
    value to : integer;
}
```

Online resources

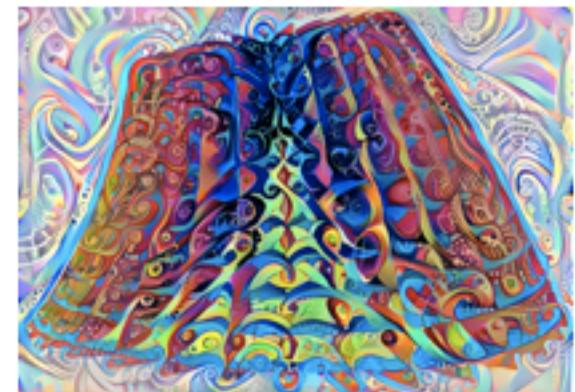


YAS (Yet Another SLR (Software Language Repository))
<http://www.softlang.org/yas>



MetaLib (Comparison of metaprogramming technologies)
www.softlang.org/metalib

The Software Languages Book
<http://www.softlang.org/book>



Publications

- <http://www.softlang.org/mega>
 - Modeling the Linguistic Architecture of Software Products
 - Interpretation of Linguistic Architecture
 - Interconnected Linguistic Architecture
 - Axioms of linguistic architecture
 - Coupled Software Transformations—Revisited
 - Relationship Maintenance in Software Language Repositories
 - Language Support for Megamodel Renarration
- Forthcoming
 - Systematic Comparison of Metaprogramming Systems
 - Linked Open Data for a Software Language Repository
 - Megamodeling-based Patterns of Co-evolution

Thank you!

