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!
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


© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
Contribute to the BOK!

Twitter name: @reallynotabba
Hashtag: #slebok

---

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

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

<table>
<thead>
<tr>
<th>Language</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haskell¹</td>
<td>The functional programming language Haskell</td>
</tr>
<tr>
<td>Java²</td>
<td>The Java programming language</td>
</tr>
<tr>
<td>Python³</td>
<td>The dynamic programming language Python</td>
</tr>
<tr>
<td>Prolog⁴</td>
<td>The logic programming language Prolog (specifically SWI-Prolog)</td>
</tr>
</tbody>
</table>

¹ Haskell language: [https://www.haskell.org/](https://www.haskell.org/)
³ Python language: [https://www.python.org/](https://www.python.org/)
⁴ Prolog language: [https://en.wikipedia.org/wiki/Prolog](https://en.wikipedia.org/wiki/Prolog)
Other **software languages** used in *softlangbook*

<table>
<thead>
<tr>
<th>Language</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ANTLR</strong></td>
<td>The grammar notation of the ANTLR technology</td>
</tr>
<tr>
<td><strong>JSON</strong></td>
<td>The JavaScript Object Notation</td>
</tr>
<tr>
<td><strong>JSON Schema</strong></td>
<td>The JSON Schema language</td>
</tr>
<tr>
<td><strong>XML</strong></td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td><strong>XSD</strong></td>
<td>XML Schema Definition</td>
</tr>
</tbody>
</table>

---

5. [ANTLR language](http://www.antlr.org/)
6. [JSON language](https://en.wikipedia.org/wiki/JSON)
7. [JSON Schema language](http://json-schema.org/)
## Software languages mentioned in *softlangbook*

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

---

<sup>10</sup> [Alloy language](http://alloy.mit.edu/alloy/)

<sup>11</sup> [CIL](http://www.cil.com/)

<sup>12</sup> [Common Log Format](http://commonlog.org/)

<sup>13</sup> [DocBook](http://docbook.org/)

<sup>14</sup> [FOAF](http://xmlns.com/foaf/0.1/)

<sup>15</sup> [INI File Format](http://en.wikipedia.org/wiki/INI_file)

<sup>16</sup> [Java bytecode](http://en.wikipedia.org/wiki/Java_BYTECODE)

<sup>17</sup> [make](http://www.wolips.com/)

<sup>18</sup> [OWL](http://en.wikipedia.org/wiki/Web_Ontology_Language)

<sup>19</sup> [QTFF](http://en.wikipedia.org/wiki/QuickTime_File_Format)

<sup>20</sup> [RDF](http://w3.org/RDF/)

<sup>21</sup> [RDFS](http://www.w3.org/2000/01/rdf-schema/)

<sup>22</sup> [Scala](http://www.scala-lang.org/)

<sup>23</sup> [Smalltalk](http://www.smalltalk.com/)

<sup>24</sup> [Sparql](http://www.w3.org/TR/rdf-sparql-query/)

<sup>25</sup> [UML](http://en.wikipedia.org/wiki/Unified_Modeling_Language)

<sup>26</sup> [XPath](http://www.w3.org/TR/xpath/)

<sup>27</sup> [XSLT](http://www.w3.org/TR/xslt/)
**Software languages fabricated in softlangbook**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAL</td>
<td>Basic Assembly Language</td>
</tr>
<tr>
<td>BFPL</td>
<td>Basic Functional Programming Language</td>
</tr>
<tr>
<td>BGL</td>
<td>Basic Grammar Language</td>
</tr>
<tr>
<td>BIPL</td>
<td>Basic Imperative Programming Language</td>
</tr>
<tr>
<td>BML</td>
<td>Basic Machine Language</td>
</tr>
<tr>
<td>BSL</td>
<td>Basic Signature Language</td>
</tr>
<tr>
<td>BSTL</td>
<td>Basic Signature Transformation Language</td>
</tr>
<tr>
<td>BTL</td>
<td>Basic TAPL Language</td>
</tr>
<tr>
<td>BNL</td>
<td>Binary Number Language</td>
</tr>
<tr>
<td>EL</td>
<td>Expression Language</td>
</tr>
<tr>
<td>EFPL</td>
<td>Extended Functional Programming Language</td>
</tr>
<tr>
<td>EGL</td>
<td>Extended Grammar Language</td>
</tr>
<tr>
<td>EIPL</td>
<td>Extended Imperative Programming Language</td>
</tr>
<tr>
<td>ESL</td>
<td>Extended Signature Language</td>
</tr>
<tr>
<td>FSML</td>
<td>Finite State Machine Language</td>
</tr>
<tr>
<td>GBL</td>
<td>Graph-based Buddy Language</td>
</tr>
<tr>
<td>MML</td>
<td>MetaModeling Language</td>
</tr>
<tr>
<td>TBL</td>
<td>Tree-based Buddy Language</td>
</tr>
<tr>
<td>TLL</td>
<td>Typed Lambda Language</td>
</tr>
<tr>
<td>ULL</td>
<td>Untyped Lambda Language</td>
</tr>
</tbody>
</table>
# Classification by nature of language elements

*softlangbook*

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

\[
\begin{align*}
  m & \vdash \text{skip} \rightarrow m \\
  m & \vdash e \rightarrow v \\
  m & \vdash \text{assign}(x, e) \rightarrow m[x \leftarrow v] \\
  m_0 & \vdash s_1 \rightarrow m_1 \quad m_1 & \vdash s_2 \rightarrow m_2 \\
  m_0 & \vdash \text{seq}(s_1, s_2) \rightarrow m_2 \\
  m & \vdash e_0 \rightarrow \text{true} \quad m & \vdash s_1 \rightarrow m' \\
  m & \vdash \text{if}(e_0, s_1, s_2) \rightarrow m'
\end{align*}
\]
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

Grammar

Rules for type system etc.

Rules for code generation

Source code

Parser

Parse tree

Semantic analysis

Enriched parse tree

Code generator

Machine code
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


© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
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
   3.1. The Programming Process
   3.2. Programming Paradigms

4. Programming Language Basics
   4.1. Programming Language Overview
   4.2. Syntax and Semantics of Programming Languages
   4.3. Low-Level Programming Languages
   4.4. High-Level Programming Languages
   4.5. Declarative vs. Imperative Programming Languages

5. Debugging Tools and Techniques
   5.1. Types of Errors
   5.2. Debugging Techniques
   5.3. Debugging Tools

10. Compiler Basics
    10.1. Compiler/Interpreter Overview
    10.2. Interpretation and Compilation
    10.3. The Compilation Process
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
- SW reverse engineering
- SW analysis
  - Program slicing
  - Feature location
- Traceability recovery
- Design pattern detection
- Change-impact analysis
- Code smell
- Software metric
- 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)

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

XML-to-object

public class Point {
  public int x;
  public int y;
}

object-to-XML
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.
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/...
How to exit the Vim editor?

I'm stuck and cannot escape. It says:

"type :quit<Enter> to quit VIM"

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


© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
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
/krɛˈstɔməθi/  

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.
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,
• ...

© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
An example of a megamodel:

A tombstone diagram for bootstrapping a compiler
Another example of a megamodel:
Mechanics of an ATL-based transformation

© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
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]
An example of LOD for a chrestomathy

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

**About: language-haskell**

An Entity of Typ `functional_programming_language`, from : 101companies.org

<table>
<thead>
<tr>
<th>Predicat</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>isPrimaryTopicOf</td>
<td>wikipage-language-haskell</td>
</tr>
<tr>
<td>22-rdf-syntax-ns#type</td>
<td>functional_programming_language</td>
</tr>
<tr>
<td>22-rdf-syntax-ns#type</td>
<td>language</td>
</tr>
<tr>
<td>rdf-schema#label</td>
<td>Haskell</td>
</tr>
<tr>
<td>is uses of</td>
<td>contribution-dph</td>
</tr>
</tbody>
</table>
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.
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.

<table>
<thead>
<tr>
<th>October 2007</th>
<th>San Francisco</th>
<th>615</th>
<th>59+ million</th>
</tr>
</thead>
<tbody>
<tr>
<td>First commit</td>
<td>Headquarters</td>
<td>Employees worldwide</td>
<td>Projects hosted</td>
</tr>
</tbody>
</table>
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

Grammars written for ANTLR v4; expectation that the grammars are free of actions.
https://github.com/usethesource/rascal
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 produces 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

© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
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, Espirit, CSER, IBM Research and the IBM Center for Advanced Studies, and is presently supported by NSERC.
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:
http://www.kframework.org/

- **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 Katelman
  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

```racket
#lang racket
#
This is the semantics of Beginner Scheme, one of the languages in DrRacket.

The first test case fails because the beginner spec is broken for that program (ie, the model faithfully reflects the (broken) spec).
```
http://slps.github.io/

Software Language Processing Suite

Mission

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.

Pages

- 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
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](http://www.softlang.org/yas)
- An emerging LOD view: [.html](http://www.softlang.org/yas)
- The Software Languages Book: [.html](http://www.softlang.org/yas)
- YAS' megamodeling language Ueber: [.html](http://www.softlang.org/yas)

page revision: 25, last edited: 9 May 2017, 18:01 (18 days ago)

Stop watching this page [?]

© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
Case classes for object representation

Features: AST-

Scala Algebraic data type

```scala
package org.softlang.fsml

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
```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.softlang.org"

class="Company"

declaration="Company">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Name" type="xs:string">
            <xs:element name="TopLevelDepartment">
                </xs:complexType>
            <xs:element name="Manager" type="Employee">
                <xs:element name="SubDepartment" type="Employee">
                    <xs:element name="Employee" type="Employee">
                        </xs:complexType>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```
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

<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 Language</td>
</tr>
<tr>
<td>L2</td>
<td>Is one language contained in another?</td>
<td>Relationship subsetOf</td>
</tr>
<tr>
<td>A1</td>
<td>What artifacts participate in the scenario?</td>
<td>Type Artifact</td>
</tr>
<tr>
<td>A2</td>
<td>What is the language of each artifact?</td>
<td>Relationship elementOf</td>
</tr>
<tr>
<td>A3</td>
<td>Does an artifact conform to another artifact?</td>
<td>Relationship conformsTo</td>
</tr>
<tr>
<td>A4</td>
<td>Does an artifact define a language?</td>
<td>Relationship defines</td>
</tr>
<tr>
<td>F1</td>
<td>Is one artifact derived from another artifact?</td>
<td>Type Function</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 defines</td>
</tr>
<tr>
<td>R1</td>
<td>Are artifacts closely similar to each other?</td>
<td>Relationship correspondsTo</td>
</tr>
<tr>
<td>R2</td>
<td>Can a correspondence be structured?</td>
<td>Relationship partOf</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 Concept</td>
</tr>
<tr>
<td>C2</td>
<td>Does the entity use the concept?</td>
<td>Relationship uses</td>
</tr>
<tr>
<td>C3</td>
<td>Does the entity help to use the concept?</td>
<td>Relationship facilitates</td>
</tr>
</tbody>
</table>
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 metametamodel
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
module EMFModelAPI import (EMF, // The EMF module is enhancedXML) // The XML module is needed for serializationJava : Language // Java is a LanguageEcoreJava : Language // A Java subset for EMF Model APIsEcoreJava subsetOf Java // An EMF Model API is valid JavaEMFGeneratorModel : Language // Language for the generator modelgenModel : Artifact // Parameters of the generationgenModel elementOf EMFGeneratorModelgenModel references metaModel // ReferencingEMFGenerator : EMFGeneratorModel → EcoreJavaEMFGenerator(genModel) → javaFiles // Application of generatorCustomObjects : Language // Object graphs for Custom serialization : CustomObjects → Custom Deserialization : Custom → CustomObjectsXMI : Language // Format for default persistence for EMFXML subsetOf XML // XML 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) ➔ model
CustomDeserialize(model) ➔ objectGraph
Interconnected linguistic architecture
Interconnection of model and system
URI-based resolution

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

// module EMF continued
metaMetaModel = 'eclipse:/org.eclipse.emf.ecore/model/Ecore.ecore'
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'

// module EMFModelAPI continued
Persistence : Concept
Persistence =
  'http://dbpedia.org/page/Persistence_(computer_science)'
CustomSerialize facilitates Persistence
CustomDeserialize facilitates Persistence
Pluggable analyses

xmlFile `elementType` 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
        ...
    }
}
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
Modularized models

module xml

xmlFile = library.xml
xsdFiles = libraries.xsd

import Substitute xmlFile by input

input = library-in.xml
output = library-out.xml
xsdFiles = libraries.xsd

module xmlTrafo

import Substitute xmlFile by output

XMLAcceptor : Plugin
Bound to classpath:plugins.jaxb.AcceptXML

Relationships in defined document:

• XMLAcceptor partOf StringAcceptor
XML: XMLAcceptor, ?XMLAcceptor, XMLAcceptor

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 EMFPartInferrer extends MegaLInferrer {
    // Returns an evaluation report and a model extension
    protected Report<Megamodel> infer(Entity element) {
        ...
    }
}
Explorable connections
## Traceability links

![Diagram showing traceability links between Java files, a metamodel, and a meta-metamodel.](image)

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

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

### javaFiles
- `org/softlang/company/xjc/Employee.java`
- `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`

© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
# Literature survey

| [30] |                |                |                |                |                |
| [7]  |                |                |                |                |                |
| [25] |                |                |                |                |                |
| [15] |                |                |                |                |                |
| [11] |                |                |                |                |                |
| [17] |                |                |                |                |                |
| [12] |                |                |                |                |                |
| [3]  |                |                |                |                |                |
| [4]  |                |                |                |                |                |
| ([10]) |            |                |                |                |                |

| $L_3$ | $L_2$ | $L_1$ |                |                |                |
|-------|-------|-------|                |                |                |
|       |       |       | ○                | ○                | ○                |
|       |       |       | ×                | ×                | ×                |
|       |       |       | ○                | ○                | ×                |
|       |       |       | ×                | ×                | ×                |

- $L_1$-$L_3$ maturity levels
- ○ implementation
- × demonstration

<table>
<thead>
<tr>
<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>3.8</td>
<td>3.1</td>
<td>3.6</td>
<td>3.3</td>
<td>3.7</td>
<td>3.4</td>
<td>3.2</td>
<td>3.5</td>
</tr>
</tbody>
</table>
Linguistic architecture in forward and reverse engineering
Axioms of linguistic architecture
Entity types in megamodeling survey

<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>x</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></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>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[6]</td>
<td></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>
</tr>
<tr>
<td>[7]</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[8]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[9]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[10]</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[11]</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[12]</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[13]</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></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>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Relationship types in megamodeling survey

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[3]</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[4]</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[5]</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>[6]</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[7]</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>[8]</td>
<td>x</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></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>[11]</td>
<td>x</td>
<td></td>
<td></td>
<td></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>
</tr>
<tr>
<td>[13]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[14]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
Understanding Membership

commpany.ecore

defines

XMLCompany

conformsTo

company.xmi

elementOf
Understanding Membership

\[ s:Artifact \quad \text{defines} \quad l:Language \]

\[ a:Artifact \quad \text{conformsTo} \]

\[ \quad \text{elementOf} \]

<table>
<thead>
<tr>
<th>s</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grammar</td>
<td>Code</td>
</tr>
<tr>
<td>Schema</td>
<td>Instance</td>
</tr>
<tr>
<td>Metamodel</td>
<td>Model</td>
</tr>
</tbody>
</table>
Understanding Membership

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

- Specification(a) ⇒ Artifact(a).
- Language(l) ⇒ ∃s. Specification(s) ∧ defines(s, l) ...
- defines(a, e) ⇒ Artifact(a) ∧ Entity(e).
- conformsTo(a, s) ⇒ Artifact(a) ∧ Artifact(s).
- conformsTo(a, s) ⇔ (∀p_a. partOf(p_a, a) ∧ ∃p_s. partOf(p_s, s) ∧ conformsTo(p_a, p_s)) ∧ ∃t. defines(s, t) ∧ 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.
Illustrative YAS artifacts
A binary number

101.01

BTW, everything is a file in a/this SLR.
2 The YAS software language repository

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’:

© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
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 BNL 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.bnl

101.01

Language bnl(json) corresponds to the abstract, tree-based synt JSON for representation; here is the JSON representation of ‘ı0ı.0ı
Branch: programming17  

yas / languages / BNL / samples / 5comma25.bnl

riaemmel Initial commit due to repo reorg  

0ae805b on 12 Jun 2016

1 contributor

2 lines (1 sloc)  7 Bytes

1 101.01
Branch: programming17

yas / languages / BNL / samples / 5comma25.bnl

ralmme Initial commit due to repo reorg

1 contributor

2 lines (1 sloc) | 7 Bytes

1 101.01
commit due to repo reorg

7 Bytes
Decimal representation of 101.01

5.25.
A JSON-based AST of \texttt{101.01}

\begin{verbatim}
{
    "bits": ["one", "zero", "one"],
    "rest": ["zero", "one"]
}
\end{verbatim}
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 ;
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

\[2^{(1+1+1-1)+ (0+2^{(1+1+1-1-1)})} + (0+2^{(-1-1)})].\]

\[
\begin{align*}
\text{number}(\text{Val1}+\text{Val2}) & \longrightarrow \text{bits}(\text{Len1}-1, \text{Len1}, \text{Val1}), \text{rest}(\text{Val2}). \\
\text{bits}(\text{Pos}, 1, \text{Val}) & \longrightarrow \text{bit}(\text{Pos}, \text{Val}). \\
\text{bits}(\text{Poso}, \text{Len1}+1, \text{Val1}+\text{Val2}) & \longrightarrow \text{bit}(\text{Poso}, \text{Val1}), \text{bits}(\text{Poso}-1, \text{Len1}, \text{Val2}). \\
\text{bit}(\_\text{Pos}, 0) & \longrightarrow ['0']. \\
\text{bit}(\text{Pos}, 2^{\_\text{Pos}}) & \longrightarrow ['1']. \\
\text{rest}(0) & \longrightarrow [\cdot]. \\
\text{rest}(\text{Val}) & \longrightarrow ['\cdot'], \text{bits}(\_1, _\text{Len}, \text{Val}).
\end{align*}
\]
An evaluator

\[2^{(1+1+1-1)} + (0+2^{(1+1+1-1-1)}) + (0+2^{(-1-1)})].\]

evolve(F, V) ⇔ V is F.

5.25.
Languages as types in an SLR

- Representation type
- Modeled language

- Prolog/YAS format
- Interchange format
- “Proprietary” format

© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
Relations and functions on languages
Relations and functions on languages

\[ \text{bnl}(\text{tokens(term)}) \]

- parse \# bglParser

\[ \text{bnl}(\text{term}) \]

- termConverter
- explode \# bglExploder
- : \# bnlTermEvaluator

\[ \text{bnl}(\text{tree(term)}) \]

- unparsing # bglTreeToTokens
- unparsing # bglTreeToText
- imploding # bglImploder

\[ \text{bnl}(\text{text}) \]

- parse \# bglParser(bnlScanner)
ueber's language concepts

- **Language** represented as:
  - Text
  - Tree (JSON, XML)
  - Graph (model)
  - Binary

- **Membership** defined by:
  - Grammar
  - Signature
  - Metamodel
  - Constraint
  - Tool

- **Relation** models:
  - Conformance
  - Compatibility
  - Traceability
  - Correspondence

- **Function** models:
  - Parsing
  - Resolution
  - Transformation
  - Analysis
  - Formatting

© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
Abstract syntax of \textit{ueber}

```plaintext
type model = decl*;
symbol language : lang \rightarrow decl;
symbol elementOf : file \times lang \rightarrow decl;
symbol notElementOf : file \times lang \rightarrow decl;
symbol membership : lang \times goal \times file* \rightarrow decl;
symbol relation : rela \times lang* \times goal \times file* \rightarrow decl;
symbol relatesTo : rela \times file* \rightarrow decl;
symbol function : func \times lang* \times lang* \times goal \times file* \rightarrow decl;
symbol mapsTo : func \times file* \times file* \rightarrow decl;
symbol equivalence : lang \times goal \times file* \rightarrow decl;
symbol normalization : lang \times goal \times file* \rightarrow decl;
symbol macro : goal \rightarrow decl;

\textbf{type} \hspace{1em} \textbf{file} = \textbf{string} ; \hspace{1em} // \textbf{filenames}
\textbf{type} \hspace{1em} \textbf{rela} = \textbf{string} ; \hspace{1em} // \textbf{names of relations}
\textbf{type} \hspace{1em} \textbf{func} = \textbf{string} ; \hspace{1em} // \textbf{names of functions}
\textbf{type} \hspace{1em} \textbf{lang} = \textbf{term} ; \hspace{1em} // \textbf{names of languages}
\textbf{type} \hspace{1em} \textbf{goal} = \textbf{term} ; \hspace{1em} // \textbf{Prolog literals}
```

© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
Language declarations

\texttt{language(term).}
\texttt{language(bnl(term)).}
Element-of declarations

\texttt{elementOf('languages/BNL/samples/5comma25.bnl', bnl(text))}.
Membership functionality

\[
\text{membership}(\text{bnl(text)}, \text{bglTopDownAcceptor(\text{bnlScanner}), ["languages/BNL/cs.term")].}
\]
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

\[
\text{fx}_y(\text{Fun}, \text{FX}, \text{LX}, \text{FY}, \text{LY}) \Leftarrow \text{ueber}([ \\
\text{elementOf}(\text{FX}, \text{LX}), \\
\text{elementOf}(\text{FY}, \text{LY}), \\
\text{mapsTo}(\text{Fun}, [\text{FX}], [\text{FY}]))]).
\]

\[
[ \\
\text{macro}\text{fx}_y(pp, 'text.ppl', ppl(term), 'text.txt', text)), \\
\text{macro}\text{fx}_y(pp, 'vbox.ppl', ppl(term), 'vbox.txt', text)), \\
\text{macro}\text{fx}_y(pp, 'vlist.ppl', ppl(term), 'vbox.txt', text)), \\
... \\
].
\]
ueber's semantics

SLR
- .ueber
  - languages
  - bnl
    - .ueber
    - cs.bgl
    - cs.term
  - samples
    - .ueber
      - 5comma25.bnl
      - 5comma25.term
    - ...
    - ...
    - ...

Collection

Well-formedness checking

Ueber declarations

Repository verification

Problems

Artifact override and creation
‘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

<table>
<thead>
<tr>
<th>Implement relations &amp; functions</th>
<th>Prolog</th>
<th>Haskell</th>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicates</td>
<td></td>
<td>Main functions</td>
<td>Main methods</td>
<td>Scripts</td>
</tr>
<tr>
<td>Code location</td>
<td>Module auto loading</td>
<td>Automated module search path</td>
<td>Automated CLASSPATH</td>
<td>Automated PYTHONPATH</td>
</tr>
<tr>
<td>Compilation</td>
<td>N/A</td>
<td>On the fly</td>
<td>On the fly</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Figure**  
ueber’s integrated compile- and run-time.

<table>
<thead>
<tr>
<th>Parse text</th>
<th>Prolog</th>
<th>Haskell</th>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCG, ...</td>
<td></td>
<td>Parsec, ...</td>
<td>ANTLR, ...</td>
<td>ANTLR, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Represent trees</th>
<th>Prolog terms</th>
<th>read/show conversion, JSON, XML</th>
<th>JSON, XML</th>
<th>str/repr conversion, JSON, XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON, XML</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 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
Index

- 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)
Languages: YAS index

<table>
<thead>
<tr>
<th>Language</th>
<th>YAS-specific?</th>
<th>Headline</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANTLR</td>
<td></td>
<td>The grammar language of the ANTLR technology</td>
</tr>
<tr>
<td>ASL</td>
<td>✔</td>
<td>A simple language for algebraic specification</td>
</tr>
<tr>
<td>BAL</td>
<td>✔</td>
<td>A trivial assembly language</td>
</tr>
<tr>
<td>BCL</td>
<td>✔</td>
<td>A format for CSTs for parsing with BGL grammars</td>
</tr>
<tr>
<td>BFPL</td>
<td>✔</td>
<td>A trivial first-order functional programming language</td>
</tr>
<tr>
<td>BGL</td>
<td>✔</td>
<td>A BNF-like notation for context-free grammars</td>
</tr>
<tr>
<td>BIPL</td>
<td>✔</td>
<td>A trivial imperative programming language</td>
</tr>
<tr>
<td>BL</td>
<td>✔</td>
<td>A trivial language for buddy relationships</td>
</tr>
<tr>
<td>BML</td>
<td>✔</td>
<td>A trivial machine language</td>
</tr>
<tr>
<td>BNL</td>
<td>✔</td>
<td>A trivial language for binary numbers</td>
</tr>
</tbody>
</table>
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.
Properties

- this relatesTo Language:C
- this facilitates Imperative programming
- this subsetOf Language:EIPL
- 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)
Components

- languages/BIPL/cs.eql (Context-free grammar)
- languages/BIPL/is.eql (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)
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`
// Statements
[skip] stmt : ':'
[assign] stmt : name '==' expr ':'
[block] stmt : '{' [stmt] '}'
[if] stmt : 'if' '(' expr ')' stmt { 'else' stmt }?
[while] stmt : 'while' '(' expr ')' stmt

// Expressions
[or] expr : bexpr { '||' bexpr }?
[and] bexpr : cexpr { '&&' bexpr }?
[lt] cexpr : aexpr { '<' aexpr }?
[leq] cexpr : aexpr { '<=' aexpr }?
[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

Aggregation

Annotated file system

Annotation dump

Reasoning

Annotation dump

Linking

LOD JSON dump

Publishing

HTML view

Missing aspects:
- HTTP GET access (except for dump)
- RDF access
- Schema validation
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

exception

mute
pass
ticket/eject

release
pass/alarm
locked

ticket/collect
pass
unlocked

ticket/eject
initial state locked {
    ticket/collect $\rightarrow$ unlocked;
    pass/alarm $\rightarrow$ exception;
}

state unlocked {
    ticket/eject;
    pass $\rightarrow$ locked;
}

state exception {
    ticket/eject;
    pass;
    mute;
    release $\rightarrow$ locked;
}
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

```plaintext
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 \times stateid \times transition*;
type initial = boolean;
type transition = event \times action? \times stateid;
type stateid = string;
type event = string;
type action = string;
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

\[ \ldots, \langle b, x, \ldots, \langle e, \langle a \rangle, x' \rangle, \ldots \rangle \ldots \vdash \langle x, e \rangle \rightarrow \langle x', \langle a \rangle \rangle \]  

\[ \ldots, \langle b, x, \ldots, \langle e, \langle \rangle, x' \rangle, \ldots \rangle \ldots \vdash \langle x, e \rangle \rightarrow \langle x', \langle \rangle \rangle \]  

[action]  

[no-action]
Well-formedness
(a violation thereof)

\begin{verbatim}
initial state stateA { event1/action1 \rightarrow stateB; }
state stateB { }
state stateC { }
\end{verbatim}
Generated C code

```c
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

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

- 109 commits
- 3 branches
- 0 releases
- 3 contributors

Branch: master  New pull request

- 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/

External DSL style with ANTLR and Java

<table>
<thead>
<tr>
<th>Perspectives</th>
<th>Features</th>
<th>Languages</th>
<th>Technologies</th>
<th>Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>meta</td>
<td>AST+</td>
<td>ANTLR4</td>
<td>ANTLR4</td>
<td>Boilerplate code</td>
</tr>
<tr>
<td>object</td>
<td>Abstraction</td>
<td>Java</td>
<td>JUnit</td>
<td>External DSL</td>
</tr>
<tr>
<td>test</td>
<td>Concrete syntax</td>
<td></td>
<td></td>
<td>Functional constructor</td>
</tr>
<tr>
<td></td>
<td>Scanning</td>
<td></td>
<td></td>
<td>Getter</td>
</tr>
<tr>
<td></td>
<td>Text-to-AST</td>
<td></td>
<td></td>
<td>Listener</td>
</tr>
<tr>
<td></td>
<td>Text-to-CST</td>
<td></td>
<td></td>
<td>Parser generation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Perspectives</th>
<th>Features</th>
<th>Languages</th>
<th>Technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Concrete syntax</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```java
initial state locked {
    ticket/collect -> unlocked;
    pass/alarm -> exception;
} state unlocked {
    ticket/eject;
    pass -> locked;
}
```
Feature model for metaprogramming implementation of FSML DSL
Syntax

Abstract syntax

Concrete syntax
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
abstract class 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

© 2017, Software Languages Team, University of Koblenz-Landau, Ralf Lämmel
Publications

- [http://www.softlang.org/mega](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!