Declarative Software Development

Distilled Tutorial

Ralf Lämmel

Software Languages Team, University of Koblenz Landau

Joint work with Andrei Varanovich, Martin Leinberger, Thomas Schmorleiz, Jean-Marie Favre
Declarative Software Development?
How the title emerged …

• **Ralf**: Here is my title: „Towards an ontology for software development“.

• **Olivier**: May I suggest the title „Towards a Declarative Ontology for Software Development“?

• **Ralf**: Any ontology is declarative. What about this title: „Declarative knowledge about software development“?

• **Olivier**: „Declarative Software Development“ would come across as more radical, possibly even novel.

• **Ralf**: Thanks for radicalizing me. I go with your proposal.
What’s NOT meant by declarative software development …

- (Exclusive) use of declarative PLs in development
- Use of formal methods instead
- Use of MDE and model transformations instead
Declarative software development: 
The focus is on **understanding**

Linked Software Data

Software ontology

- **Thing**
- **Language**
- **Technology**
- **Concept**

Software analysis

uses(S,’JUnit’) ← imports(S, ’org.junit’).

Technology models

- An XML schema
- An object model

© 2014 Software Languages Team, University of Koblenz-Landau
Declarative software development: The focus is on **understanding**

**Linked Software Data** with 101explorer and 101triples

- **The SoLaSoTe ontology**
  - **Thing**
  - **Language**
  - **Technology**
  - **Concept**

- **Chrestomathy**
  - 101

- **The MegaL language**
  - An XML schema
    - map to
  - An object model

- **The 101worker infrastructure**
  - uses(S,’JUnit’)
  - imports(S,’org.junit’).
Why is this important?

...because of the technology plethora
The software chrestomathy

101 companies

A quick intro
The 101 companies project

Company X: Swing + JDBC

Company Y: SWT + Hibernate

Company Z: GWT + MongoDB

... 

Different implementations of the same system varying languages, technologies, and concepts.

For what it matters, it’s an HRMS: a human resources management system.
Data & functionality of 101’s system

- **Total**: Sum up all salaries
- **Increase**: Increase salaries of all employees
- **Cut**: Cut all salaries in half
- **Persistence**: Persist companies
- **Editing**: GUI support for editing companies
- …
A Prolog-based implementation

% Total all salaries in a company
total(X,R) :- collect(getSalary,X,L), sum(L,R).

% Helper for salary extraction
getSalary(employee(_,_,S),S).

% Higher—order traversal for accumulation
collect(P,X,L) :-
  apply(P,[X,Y]) ->
  L = [Y];
  X =.. [-|Xs],
  maplist(collect(P),Xs,Yss),
  append(Yss,L).

Scrap Your Boilerplate—Prologically!
PPDP 2009
# 101’s features

<table>
<thead>
<tr>
<th>feature</th>
<th>headline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>Sum up the salaries of all employees.</td>
</tr>
<tr>
<td>Cut</td>
<td>Cut the salaries of all employees in half.</td>
</tr>
<tr>
<td>Hierarchical_company</td>
<td>A data model for companies with nested departments</td>
</tr>
<tr>
<td>Parsing</td>
<td>Parse an external format for company data</td>
</tr>
<tr>
<td>Closed_serialization</td>
<td>Closed serialization for company data</td>
</tr>
<tr>
<td>Unparsing</td>
<td>Unparse company data to an external format</td>
</tr>
<tr>
<td>Browsing</td>
<td>UI support for browsing company data</td>
</tr>
<tr>
<td>Depth</td>
<td>Compute the nesting depth of departments</td>
</tr>
<tr>
<td>Editing</td>
<td>UI support for editing company data</td>
</tr>
<tr>
<td>Web_UI</td>
<td>A web-based user interface</td>
</tr>
<tr>
<td>Distribution</td>
<td>Distribution of company data and operations</td>
</tr>
<tr>
<td>Company</td>
<td>Basic company structure</td>
</tr>
<tr>
<td>Flat_company</td>
<td>A data model for flat companies</td>
</tr>
<tr>
<td>Persistence</td>
<td>Persistence for company data</td>
</tr>
<tr>
<td>Mapping</td>
<td>Mapping company data across technological spaces</td>
</tr>
<tr>
<td>Open_serialization</td>
<td>Open serialization for company data</td>
</tr>
</tbody>
</table>
101’s implementations

<table>
<thead>
<tr>
<th>contribution</th>
<th>headline</th>
</tr>
</thead>
<tbody>
<tr>
<td>haskellEngineer</td>
<td>Basic software engineering for Haskell</td>
</tr>
<tr>
<td>haskellComposition</td>
<td>Data composition in Haskell with algebraic data types</td>
</tr>
<tr>
<td>mysqlMany</td>
<td>A MySQL database with SQL scripts</td>
</tr>
<tr>
<td>javaComposition</td>
<td>Object composition in Java</td>
</tr>
<tr>
<td>antlrAcceptor</td>
<td>An ANTLR-based acceptor for textual syntax</td>
</tr>
<tr>
<td>antlrLexer</td>
<td>Lexer-based text processing with ANTLR</td>
</tr>
<tr>
<td>javaInheritance</td>
<td>Class inheritance in Java</td>
</tr>
<tr>
<td>antlrParser</td>
<td>An ANTLR-based parser with semantic actions</td>
</tr>
<tr>
<td>antlrObjects</td>
<td>ANTLR-based object-text mapping for Java</td>
</tr>
<tr>
<td>antlrTrees</td>
<td>Parsing and tree walking with ANTLR</td>
</tr>
<tr>
<td>jdom</td>
<td>XML processing with Java’s JDOM API</td>
</tr>
<tr>
<td>jaxbComposition</td>
<td>Object-XML mapping with JAXB of the Java platform</td>
</tr>
</tbody>
</table>
The 101 companies project

- 101repo: GitHub-based source code repo
- 101wiki: Semantic wiki-based documentation
- 101worker: Infrastructure for automated analysis
- 101explorer: Linked Data explorer
- 101triples: Linked Data SPARQL endpoint
- SoLaSoTe: Underlying ontology
- MegaL: Language for technology models
The software ontology

SoLaSoTe

A quick intro
Types of SoLaSoTe’s individuals

<table>
<thead>
<tr>
<th>type</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>onto:Concept</td>
<td>Software concepts</td>
</tr>
<tr>
<td>onto:Contribution</td>
<td>Contributions to the 101 project</td>
</tr>
<tr>
<td>onto:Contributor</td>
<td>Contributors to the 101 project</td>
</tr>
<tr>
<td>onto:Course</td>
<td>Courses on programming and software engineering</td>
</tr>
<tr>
<td>onto:Document</td>
<td>Documents in a broad sense</td>
</tr>
<tr>
<td>onto:Feature</td>
<td>Software features</td>
</tr>
<tr>
<td>onto:Language</td>
<td>Software languages</td>
</tr>
<tr>
<td>onto:Script</td>
<td>Scripts as units of a course</td>
</tr>
<tr>
<td>onto:Technology</td>
<td>Software technologies</td>
</tr>
<tr>
<td>onto:Theme</td>
<td>Containers of contributions</td>
</tr>
<tr>
<td>onto:Vocabulary</td>
<td>Containers of terms</td>
</tr>
</tbody>
</table>
For instance: software languages

<table>
<thead>
<tr>
<th>language</th>
<th>headline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>An OO programming language</td>
</tr>
<tr>
<td>Haskell</td>
<td>A purely-functional programming language</td>
</tr>
<tr>
<td>XML</td>
<td>The extensible markup language</td>
</tr>
<tr>
<td>JavaScript</td>
<td>A multi-paradigm programming language for the web et al.</td>
</tr>
<tr>
<td>JSON</td>
<td>The JavaScript Object Notation for data exchange</td>
</tr>
<tr>
<td>SQL</td>
<td>Data definition and manipulation for relational databases</td>
</tr>
<tr>
<td>Python</td>
<td>A multi-paradigm programming language</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
For instance: software technologies

<table>
<thead>
<tr>
<th>technology</th>
<th>headline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradle</td>
<td>A build tool inspired by Ant and Maven</td>
</tr>
<tr>
<td>JUnit</td>
<td>A framework for unit testing for Java</td>
</tr>
<tr>
<td>Eclipse</td>
<td>An IDE for Java with a plug-in system</td>
</tr>
<tr>
<td>.NET</td>
<td>A library and runtime for programming languages on Windows</td>
</tr>
<tr>
<td>ANTLR</td>
<td>A parser generator with various language processing capabilities</td>
</tr>
<tr>
<td>GHC</td>
<td>A Haskell compiler</td>
</tr>
<tr>
<td>MySQL</td>
<td>A relational database management system</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
For instance: software concepts

<table>
<thead>
<tr>
<th>concept</th>
<th>headline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web_programming</td>
<td>The domain of web application development</td>
</tr>
<tr>
<td>Algebraic_data_type</td>
<td>A type for the construction of terms</td>
</tr>
<tr>
<td>OO_programming</td>
<td>The object-oriented programming paradigm</td>
</tr>
<tr>
<td>Functional_programming</td>
<td>The functional programming paradigm</td>
</tr>
<tr>
<td>API</td>
<td>An interface for reusable functionality</td>
</tr>
<tr>
<td>Type_class</td>
<td>An abstraction mechanism for polymorphism</td>
</tr>
<tr>
<td>Software_system</td>
<td>A system of intercommunicating software components</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
A SPARQL query sorting software concepts by popularity

```
SELECT ?concept ?headline (COUNT(?subject) AS ?count)
WHERE {
  ?concept a onto:Concept .
  ?concept onto:hasHeadline ?headline .
}
GROUP BY ?concept ?headline
ORDER BY DESC(?count)
```

Thus, the realization of SoLaSoTe depends on RDF, RDFS (OWL), and SPARQL.
Technology modeling with **MegaL**

A quick intro

Technology models

- An XML schema
- An object model
A classic „technology model“ for bootstrapping a compiler

http://en.wikipedia.org/wiki/Tombstone_diagram
A general notion of technology model

• Technology models are „ER models“.
• Entities of interest
  ‣ Software technologies and parts thereof, e.g., Hibernate
  ‣ Software languages, e.g., SQL
  ‣ Software artifacts, e.g., O/R mapping file
  ‣ Software concepts, e.g., persistence
  ‣ ...
• Relationships of interest
  ‣ Conformance
  ‣ Transformation
  ‣ ...

© 2014 Software Languages Team, University of Koblenz-Landau
A technology model for JAXB
(XML-data binding of the Java platform)

Part 1: Technology break-down and concepts
A technology model for JAXB (XML-data binding of the Java platform)

Part 2: Type-level mapping
A technology model for JAXB
(XML-data binding of the Java platform)

Part 3:
Instance-level mapping
A technology model for JAXB
(XML-data binding of the Java platform)

Part 4: Conformance
Software analysis with 101worker (incl. 101meta)

A quick intro

Software analysis
imports(S,'org.junit')
⇔ uses(S, 'JUnit')
101meta — a rule-based language for metadata inference

\[
\text{uses}(C, \text{lang}:\text{Haskell}) \iff \\
\quad \text{contrib}(C), \\
\quad \text{filePartOf}(F, C), \\
\quad \text{suffix}(F, \text{'.hs'}). \\
\]

\[
\text{uses}(C, \text{tech}:\text{JUnit}) \iff \\
\quad \text{contrib}(C), \\
\quad \text{filePartOf}(F, C), \\
\quad \text{imports}(F, \text{'org.junit'}). \\
\]

\[
\text{implements}(C, \text{feature}:\text{Total}) \iff \\
\quad \text{contrib}(C), \\
\quad \text{filePartOf}(F, C), \\
\quad \text{tokenPartOf}('\text{total}', F). \\
\]
Comparison of feature implementations across languages, technologies, and styles

Involved analyses:
- NCLOC metrics
- Feature detection
- Comparison of detected and declared features

Haskell "wins"
Linked Software Data with $101\text{explorer}$ and $101\text{triples}$
Use and enrich *Linked Data* principles:

- Use URIs as names for things, e.g., languages.
- Make URIs HTTP resolvable.
- Provide useful info, e.g., schemas or source code.
- Provide data for humans & software: HTML, JSON, RDF.
- Include links to other resources, e.g., GitHub or 101wiki.
Exploration of software data

A video going through these stages:

• 101’s contribution \texttt{haskellStarter} on \texttt{101wiki}

• Ditto’s contribution source code on \texttt{101repo}

• Various related resources on \texttt{101explorer}

• The Haskell entity on \texttt{101triples}

• An illustrative query on SPARQL endpoint

https://www.youtube.com/watch?v=0MkzWMPp1ll
The software ontology

SoLaSoTe

Deep(er) dive
SoLaToSe aspects

- Classification (instanceOf, isA)
- Relationships (uses, supports, ...)
- Containers (themes, vocabularies, ...)
- Systems (101)
- Other resources (sameAs, ...)
- Validation of the ontology
Classification

Is 'Java' a language?

true

What are the supertypes of 'Java'?

onto:OO_programming_language
onto:Programming_language

What are popular classifiers?

- Algebraic_data_type
- OO_programming
- Functional_programming
- API
- Software_system
- Client
- Web_browser
- ...
### Predicates for relationships

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>basedOn</td>
<td>Reuse of systems</td>
</tr>
<tr>
<td>carries</td>
<td>Tagging of entities</td>
</tr>
<tr>
<td>dependsOn</td>
<td>Dependence</td>
</tr>
<tr>
<td>designedBy</td>
<td>Designer of a component</td>
</tr>
<tr>
<td>developedBy</td>
<td>Developer of a component</td>
</tr>
<tr>
<td>illustrates</td>
<td>Chrestomathy</td>
</tr>
<tr>
<td>implements</td>
<td>Systems implementing descriptions</td>
</tr>
<tr>
<td>linksTo</td>
<td>Non-specific link to external resource</td>
</tr>
<tr>
<td>memberOf</td>
<td>Membership relationship</td>
</tr>
<tr>
<td>mentions</td>
<td>Nonspecific mentions</td>
</tr>
<tr>
<td>moreComplexThan</td>
<td>Comparison of complexity levels</td>
</tr>
<tr>
<td>partOf</td>
<td>Whole-part relationship</td>
</tr>
<tr>
<td>profile</td>
<td>Web page with content</td>
</tr>
<tr>
<td>reviewedBy</td>
<td>Reviewer of an entity</td>
</tr>
<tr>
<td>sameAs</td>
<td>Equivalence relative to external resource</td>
</tr>
<tr>
<td>similarTo</td>
<td>Similarity relation</td>
</tr>
<tr>
<td>supports</td>
<td>Instruments</td>
</tr>
<tr>
<td>uses</td>
<td>Use of instruments</td>
</tr>
<tr>
<td>varies</td>
<td>Similarity of variations</td>
</tr>
</tbody>
</table>

---

For example:
- feature: `Hierarchical_company`
- moreComplexThan
- feature: `Flat_company`

For example:
- tech: `ghc`
- partOf
- tech: `Haskell_platform`

For example:
- tech: `Ruby_on_Rails`
- supports
- concept: `REST`
## Predicates for relationships

<table>
<thead>
<tr>
<th>predicate</th>
<th>domain</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>onto:basedOn</td>
<td>onto:System</td>
<td>onto:System</td>
</tr>
<tr>
<td>onto:carries</td>
<td>onto:Entity</td>
<td>onto:Tag</td>
</tr>
<tr>
<td>onto:dependsOn</td>
<td>onto:Entity</td>
<td>onto:Entity</td>
</tr>
<tr>
<td>onto:designedBy</td>
<td>onto:Entity</td>
<td>foaf:Person</td>
</tr>
<tr>
<td>onto:developedBy</td>
<td>onto:Entity</td>
<td>foaf:Person</td>
</tr>
<tr>
<td>onto:illustrates</td>
<td>onto:Description</td>
<td>onto:Instrument</td>
</tr>
<tr>
<td>onto:implements</td>
<td>onto:System</td>
<td>onto:Description</td>
</tr>
<tr>
<td>onto:linksTo</td>
<td>onto:Entity</td>
<td>rdfs:Literal</td>
</tr>
<tr>
<td>onto:memberOf</td>
<td>onto:Entity</td>
<td>onto:Container</td>
</tr>
<tr>
<td>onto:mentions</td>
<td>onto:Entity</td>
<td>onto:Entity</td>
</tr>
<tr>
<td>onto:moreComplexThan</td>
<td>onto:Entity</td>
<td>onto:Entity</td>
</tr>
<tr>
<td>onto:partOf</td>
<td>onto:Entity</td>
<td>onto:Entity</td>
</tr>
<tr>
<td>onto:profile</td>
<td>onto:Contributor</td>
<td>rdfs:Literal</td>
</tr>
<tr>
<td>onto:reviewedBy</td>
<td>onto:Entity</td>
<td>foaf:Person</td>
</tr>
<tr>
<td>onto:sameAs</td>
<td>onto:Entity</td>
<td>rdfs:Literal</td>
</tr>
<tr>
<td>onto:similarTo</td>
<td>onto:Entity</td>
<td>rdfs:Literal</td>
</tr>
<tr>
<td>onto:supports</td>
<td>onto:Instrument</td>
<td>onto:Instrument</td>
</tr>
<tr>
<td>onto:uses</td>
<td>onto:System</td>
<td>onto:Instrument</td>
</tr>
<tr>
<td>onto:varies</td>
<td>onto:System</td>
<td>onto:System</td>
</tr>
</tbody>
</table>
Query: **Arrange lecture in a course**

```
SELECT DISTINCT
  ?course
  (COUNT(?prerequisites) AS ?count)
WHERE {
  ?course onto:memberOf course:Lambda_in_Koblenz .
  OPTIONAL { ?course onto:dependsOn+ ?prerequisites }
}
GROUP BY ?course
ORDER BY ?count
```

Arrange scripts (lectures) in an order that respect the lecture dependencies.
**RDFS’ inference vs. SPARQL’s validation**

```
onto:contribUsesLang
    rdfs:type rdfs:Property ;
    rdfs:subPropertyOf onto:uses ;
    rdfs:comment "Use of languages by contributions" ;
    rdfs:domain onto:Contribution;
    rdfs:range onto:Language .
```

**Semantics:** if a resource is the subject of `contribUsesLang`, then it is of type `Contribution`.

```
SELECT ?x {
    FILTER NOT EXISTS { ?x sesame:directType onto:Contribution }
}
```

**Semantics:** find resources that are subjects of `contribUsesLang` without being of declared type `Contribution`.

© 2014 Software Languages Team, University of Koblenz-Landau
**OWL’s consistency vs. SPARQL’s validation**

**OWL**

```xml
<owl:AllDisjointClasses>
<owl:members rdf:parseType="Collection">
  <owl:class rdf:about="http://101companies.org/ontology#Language"/>
  <owl:class rdf:about="http://101companies.org/ontology#Technology"/>
  <owl:class rdf:about="http://101companies.org/ontology#Concept"/>
  ...
</owl:members>
</owl:AllDisjointClasses>
```

Merely a declaration of a consistency requirement without standardized reporting semantics.

**SPARQL**

```sparql
SELECT ?entity ?t1 ?t2 {
  ?entity a ?t1 .
  ?entity a ?t2 .
  ?t1 rdfs:subClassOf onto:Entity .
  ?t2 rdfs:subClassOf onto:Entity .
  FILTER NOT EXISTS { ?t1 a onto:Classifier } .
  FILTER NOT EXISTS { ?t2 a onto:Classifier } .
}
```

An operational query for entities with more than one entity type.
Future research

• Community process for ontology evolution
• Ontology integration with Wikipedia et al.
• Help developers (akin to StackOverflow)
• Programming models for Linked Data
• Validation / mapping for ontologies
• More advanced reasoning than in SoLaSoTe
Further reading