Understanding Programming Technologies by Analogy, Examples, and Abstraction (AOSD 2011 tutorial)

Ralf Lämmel
Software Language Engineer
University of Koblenz-Landau

Follow @meganalysis
Acknowledgement

Co-developers
- Thomas Schmorleiz
- Andrei Varanovich

Scientific collaboration
- Dragan Gasevic
- Jean-Marie Favre

See here for detailed credits: 101companies:Contributors
Introduction
We have a problem.

Thanks to Jean-Marie Favre for this excellent slide!
Today’s Issues

- Silos of knowledge
- Combining technologies
- Complexity of technologies
- Entering a new space
- Teaching technologies?
Why would you study computer science, if your ultimate destiny is to get lost in space and technology?
Popular opinion 1

Practice is just inherently complex. University (say, theory or research) should not bother.
Popular opinion 2

Practice is just incidentally complex.
University must not bother.
Practice is just incredibly complex.
University can not bother.
(As yet) unpopular opinion

Practice is just amazingly complex and does not go away. University and research must, should, and can help.
Today’s Issues

• Silos of knowledge
• Combining technologies
• Complexity of technologies
• Entering a new space
• Teaching technologies?

Our Approach

• Analogy
• Examples
• Abstraction
# Analogy, examples, abstraction

<table>
<thead>
<tr>
<th>Modelware</th>
<th>XMLware</th>
<th>Ontoware</th>
<th>Dataware</th>
<th>Grammarware</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOF</td>
<td>XSD</td>
<td>RDFS</td>
<td>SQL.DDL</td>
<td>EBNF</td>
</tr>
<tr>
<td>OCL</td>
<td>XPath</td>
<td>SPARQL</td>
<td>SQL</td>
<td>TXL</td>
</tr>
<tr>
<td>QVT</td>
<td>XQuery</td>
<td>Protégé</td>
<td>MySQL</td>
<td>ASF</td>
</tr>
<tr>
<td>ArgoUML</td>
<td>XSLT</td>
<td>Topbeard</td>
<td>Oracle</td>
<td>MetaEnv.</td>
</tr>
<tr>
<td>Rose</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MoDELS</td>
<td>XML</td>
<td>ICSW</td>
<td>VLDB</td>
<td></td>
</tr>
<tr>
<td>ECMDA</td>
<td>VLDB</td>
<td>ESWC</td>
<td>SIGMOD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Analogy in space travel

- modelware
- XMLware
- Dataware
- Ontware
- javaware

- XMI
- JMI
- EMF.gen
- JAXB
- JDOM
- Sesame
- Teneo
- JDBC
- Hibernate
- JPA
- Jena
- Hibernate
- JPA
Analogy, examples, abstraction

<table>
<thead>
<tr>
<th>Department</th>
<th>Manager</th>
<th>Address</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>Craig</td>
<td>Redmond</td>
<td>123456</td>
</tr>
<tr>
<td></td>
<td>Erik</td>
<td>Utrecht</td>
<td>12345</td>
</tr>
<tr>
<td></td>
<td>Ralf</td>
<td>Koblenz</td>
<td>1234</td>
</tr>
<tr>
<td>Development</td>
<td>Ray</td>
<td>Redmond</td>
<td>234567</td>
</tr>
</tbody>
</table>

Total salaries

Cut salaries

Store companies

Navigate companies
Functionality on companies

Total salaries in XQuery

```xml
<result>
    {sum(//salary)}
</result>
```

Cut salaries in SQL DML

```sql
UPDATE employee
SET salary = salary / 2;
```
Variation points for examples

X vs. O vs. R vs. $\lambda$ etc.
Static typing vs. dynamic typing
Textual vs. abstract vs. visual syntax
GPPL vs. DSL vs. embedding vs. API
Instance- vs. operation-based mapping
Type checking vs. inference vs. reasoning
Code first vs. schema first vs. mapping only
In-memory processing vs. push vs. pull parsing
Pure vs. impure transformations (or in between)
Code vs. generative vs. model-driven vs. mapping
What’s the essence of technology xyz?

What’s the ontology of programming technologies?
Remember *Tombstone diagrams*?

"Used for describing complicated processes for bootstrapping, porting, and self-compiling of compilers, interpreters, and macro-processors."

http://en.wikipedia.org/wiki/T-diagram
Abstraction with megamodels

An XSLT transformation
What’s the ontology we need?

Class(a:Person partial)
Class(a:Academic partial a:Person)
Class(a:Happy partial a:Person)
Class(a:Lecturer partial a:Academic)
Class(a:Professor partial a:Academic)
Class(a:Student partial a:Person)
ObjectProperty(a:hasFriend)
ObjectProperty(a:isFriendOf inverseOf(a:hasFriend))
DisjointClasses(a:Student a:Academic)

Individual(a:arthur type(a:Student) type(a:Happy))
Individual(a:bob type(a:Student) type(complementOf(a:Happy)))
Individual(a:charlie type(a:Professor) type(a:Happy))
Individual(a:diane type(a:Professor) type(complementOf(a:Happy)))

http://owl.man.ac.uk/tutorial/
Abstraction with an ontology

Capability
- Access control: the capability to control access to data and resources within programs
- Distribution: the capability to distribute programs (objects) over computers in a network
- Indexing: the capability for access to keyed and ordered records
- Interaction: the capability of interactions between the user and the system
- Logging: the capability of logging certain events along program execution
- Mapping: the capability of bridging technical spaces
  - O/R mapping: the capability of bridging the technical spaces objectware and tupleware
  - O/X mapping: the capability of bridging the technical spaces objectware and XMLware
  - R/X mapping: the capability of bridging the technical spaces tupleware and XMLware
- Parallelism: the capability to execute a program in parallel
- Parsing: the capability of analyzing software artifacts in terms of their concrete syntax
- Persistence: the capability to maintain program data beyond the runtime of the program
- Serialization: the capability of converting program data into a format for storage or transmission
- Streaming: the capability for processing data in a stream as opposed to in-memory
Aside: a related course on Programming (Techniques and) Technologies

- The Expression Problem
- The Visitor Design Pattern
- Parsing
- XML Processing
- XML Validation
- XML Data Binding
- Database Access
- O/R Mapping
- Model View Controller

- More Design Patterns
- Reflection
- Aspect-Oriented Programming
- Functional OO Programming
- Combinator Libraries
- Generic Programming
- Programming with Threads
- Distributed Programming
- WebService Programming
101companies system
The 101companies system (in the sequence: just the "system") is a conceived system in the application domain of human resources. The present specification is meant to be informal and liberal; it should facilitate different implementations of the system with different programming technologies and techniques, and with different feature sets. The system is concerned with companies, departments, managers, and employees, and it supports functionality for totaling salaries, cutting salaries, computing other data, and checking data in some ways. The system may also be subject to additional capabilities---similar to non-functional requirements, e.g.: serialization, persistence, or logging. The following feature model breaks down all required or optional features of the system.
Feature model

101feature

- 101basics
  - 101feature:Company a data model for companies
  - 101feature:Cut cut all salaries in half
  - 101feature:Total total all salaries in a company
- 101capabilities
  - 101feature:Interaction interaction with companies though a user interface
  - 101feature:Logging logging for mutations of companies
  - 101feature:Persistence persistence for companies
  - 101feature:Serialization serialization for companies
- 101extras
  - 101feature:Depth determine depth of department nesting
  - 101feature:Mentoring associate employees with mentors
  - 101feature:Precedence check that salaries increase with rank in hierarchy
What's a company?

- A company is structured as follows:
  - There is a name.
  - There is any number of (possibly nested) departments.
- Each department is structured as follows.
  - There is a name.
  - There is any number of employees.
  - There is a manager as a special employee.
  - There is any number of (possibly nested) sub-departments.
- Employees are characterized by name, salary, and possibly other properties.
- The idea is that each employee can serve only in one position in the company.
Functionality on companies

Total salaries in XQuery

\[
\text{<result>}
\text{
\{sum(//salary)\}
\text{<result>}
\]

Cut salaries in SQL DML

UPDATE employee
SET salary = salary / 2;
Some options for `101feature:serialization`

- Use “Object streams”.
- Convert objects into XML tree.
- Use XML-data binding.
- Use even a persistence technology.
- Use uniform read/show (from/toString).
101companies implementations
Implementation *java*
Keywords “java”

- POJO
- Containers
- Composite pattern
- Subtyping
- Virtual methods
- Object serialization
- Marker interface pattern
Implementation *java*

```
org
  softlang
  company
    Company.java
    Department.java
    Employee.java
    Person.java
    Subunit.java
  java
  tests
    Basics.java
    Serialization.java
README
sampleCompany.ser
```
Implementation aspectJ
Keywords “aspectJ”

- Modularity
- Inter-type declarations
- Pointcuts, Advice, Logging
Implementation *aspectJ*

- org
  - softlang
    - aspectj
      - Cut.aj
      - Depth.aj
      - Logging.aj
    - tests
      - SampleCompany.java
      - Tests.java
      - Total.aj
  - company
    - Company.java
    - Department.java
    - Employee.java
    - Person.java
    - Subunit.java
- README
Implementation *haskell*
Keywords "haskell"

- Algebraic datatypes
- Pattern matching
- Structural induction
- Purity vs. impurity
Implementation *haskell*

- baseline
- Company.hs
- Cut.hs
- Main.hs
- Makefile
- README
- SampleCompany.hs
- Total.hs
Implementation $\text{syb}$
Keywords “syb”

- Boilerplate code
- Uniform traversal
- Polymorphism
- Code generation
Generic traversal with SYB

One-layer traversal

Deep traversal
Implementation syb

- baseline
- Company.hs
- Cut.hs
- Depth.hs
- Main.hs
- Makefile
- README
- SampleCompany.hs
- Total.hs
Implementation dom
Keywords “dom”

- Technical space Xmlware
- Trees vs. graphs
- Generic vs. problem-specific representation
- DOM API with its axes
- API vs. API implementation
Implementation `dom`

```
org
  softlang
  dom
    Cut.java
    DOMUtilities.java
    Tests.java
    Total.java
    README
    sampleCompany.xml
```
Implementation jaxb
Keywords “jaxb”

- XML data binding = O/X mapping
- Bridging technical spaces
- Impedance mismatch
- Code generation
- Annotations
- ...

Keywords: JAXB, XML data binding, O/X mapping, Bridging technical spaces, Impedance mismatch, Code generation, Annotations, ...
XML Schema → Binding Compiler → Schema-Derived Classes & Interfaces → Application → JAXB API → Content Objects

XML Document → marshal → JAXB API → unmarshal → Application

http://www.oracle.com/technetwork/articles/javase/index-140168.html
Implementation jaxb
101companies ontology
[−] Base

[+] 101feature

[+] 101implementation

[+] Language

[+] Technology

[+] Term
[-] Base
[+] 101feature
[+] 101implementation
[+] Language
[+] Technology
[+] Term
<table>
<thead>
<tr>
<th>101implementation</th>
<th>antlr</th>
<th>java</th>
</tr>
</thead>
<tbody>
<tr>
<td>antlr2</td>
<td>antlr2</td>
<td>jaxb</td>
</tr>
<tr>
<td>antlr3</td>
<td>antlr3</td>
<td>jaxb2</td>
</tr>
<tr>
<td>aspectJ</td>
<td>aspectJ</td>
<td>jdbc</td>
</tr>
<tr>
<td>atl</td>
<td>atl</td>
<td>jdbc2</td>
</tr>
<tr>
<td>atl2</td>
<td>atl2</td>
<td>jdom</td>
</tr>
<tr>
<td>atl3</td>
<td>atl3</td>
<td>jena</td>
</tr>
<tr>
<td>clojure</td>
<td>clojure</td>
<td>jena2</td>
</tr>
<tr>
<td>csharp</td>
<td>csharp</td>
<td>kiama</td>
</tr>
<tr>
<td>csharpEntityFramework</td>
<td>csharpEntityFramework</td>
<td>mobl</td>
</tr>
<tr>
<td>dom</td>
<td>dom</td>
<td>mps</td>
</tr>
<tr>
<td>emf</td>
<td>emf</td>
<td>prolog</td>
</tr>
<tr>
<td>fsharp</td>
<td>fsharp</td>
<td>ruby</td>
</tr>
<tr>
<td>gwt</td>
<td>gwt</td>
<td>sax</td>
</tr>
<tr>
<td>haskell</td>
<td>haskell</td>
<td>scala</td>
</tr>
<tr>
<td>hibernate</td>
<td>hibernate</td>
<td>sql</td>
</tr>
<tr>
<td>hibernate2</td>
<td>hibernate2</td>
<td>sqlServer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>swing</td>
</tr>
</tbody>
</table>
[-] Base
[+] 101feature
[+] 101implementation
[+] Language
[+] Technology
[+] Term
Language

Domain-specific language

Format language
  ASCII
  Unicode
  XML

Markup language

Programming language
  Aspect-oriented programming language
  Functional programming language
    FSharp
    Haskell
    Haskell 98
    JavaScript
    Scala
    XQuery
    XSLT
  Multi-paradigm programming language
  OO programming language

Scripting language

Unspecific language

XML language
Languages cited by implementations

XML  Java
CSS  XSD  AspectJ  Haskell
HTML  CSharp  Javascript  Scala  XQuery  XSLT  XPath
SQL
Languages cited anywhere

XML
SQL
XPath
CSharp
HTML
Java
XQuery
Haskell
Javascript
XSD
XSLT
AspectJ
Scala
[−] Base
[+] 101feature
[+] 101implementation
[+] Language
[+] Technology
[+] Term
[−] Technology
  [+] Data technology
  [+] IDE
[−] Language processor
  [−] Language implementation
    [+] Compiler
    [+] Executor
    [+] Interpreter
  [×] Parser
  [×] Pretty printer
  [×] Program analyzer
  [+] Program generator
  [×] Program transformer
[+] Platform
[+] Reuseware
[+] Runtime
[×] Tool
[+] Toolkit
Technologies cited by implementations

JSE JDK
JAXB JUnit
Eclipse JAXP DOM
Hibernet

HSQldb Web
Xalan-Java Ajax
GWT XOM GHCi Google
SAX xjc
javac AspectJ Object
xsltproc Kiama Toolkit
BaseX Saxon NUnit
Technologies cited anywhere

JSE
JDK
Hibernate
JAXP
Eclipse
JUnit
NET
JAXB
XOM
Google
Saxon
Studio
Xalan-Java
GHC
GHci

Java
DOM
AJDT
JDOM
xjc
HSQldb
SAX
Compiler
GWT
Ajax
Toolkit
Streams
Web
Sample
Edition
Kiama
Standard
CSharp

© 2010-2011, Ralf Lämmel and 101companies
[−] Base
[+] 101feature
[+] 101implementation
[+] Language
[+] Technology
[+] Term
[−] **Capability**
[−] **Mapping**
  - O/R mapping
  - O/X mapping
  - R/X mapping

**Access control**

**Distribution**

**Indexing**

**Interaction**

**Logging**

**Parallelism**

**Parsing**

**Persistence**

**Serialization**

**Streaming**
Domain

Application domain

ECommerce

Financial exchange

Health care

Human resources

Programming domain

Data programming

Database programming

Distributed programming

GUI programming

XML programming
[−] **Term**

[+] **Architectural component**

[+] **Capability**

[+] **Computation**

[+] **Design pattern**

[+] **Domain**

[+] **Language concept**

[+] **Language construct**

[+] **Program structure**

[+] **Programming**

[+] **Programming paradigm**

[+] **Programming technique**

[+] **Technical space**
Program structure

Component
Function
Method
Module
Object model
Package
[-] Term
[+] Architectural component
[+] Capability
[+] Computation
[+] Design pattern
[+] Domain
[+] Language concept
[+] Language construct
[+] Program structure
[+] Programming
[+] Programming paradigm
[+] Programming technique
[+] Technical space
Programming
Aspect-oriented programming
Functional programming
OO programming

Programming paradigm
Aspect-oriented programming paradigm
Functional programming paradigm
OO programming paradigm
[−] **Term**

[+] **Architectural component**

[+] **Capability**

[+] **Computation**

[+] **Design pattern**

[+] **Domain**

[+] **Language concept**

[+] **Language construct**

[+] **Program structure**

[+] **Programming**

[+] **Programming paradigm**

[+] **Programming technique**

[+] **Technical space**
Demo

http://101companies.uni-koblenz.de
101companies: About

Welcome to the 101companies wiki. This is an effort to organize and understand programming technologies. Please follow @meganalysis on Twitter.

101companies pages

- 101companies: Ontology: the 101companies ontology
- 101companies: System: the specification for the 101companies system
- 101companies: Implementations: implementations of the 101companies system
- 101companies: Development: the development corner for this wiki
- 101companies: Invitation: call for contributors of all kinds
- 101companies: Administration: administrators with contact information
- 101companies: Contributors: contributors of all kinds ("hall of fame")
- 101companies: Resources: papers, slide decks, and other resources
- 101companies: License: Creative Commons Deed License for wiki
101companies: **System**

**Intent**

--- A liberal specification of the 101companies system ---

**Specification**

The 101companies system (in the sequence: just the "system") is a conceived system in the application domain of human resources. The present specification is meant to be informal and liberal; it should facilitate different implementations of the system with different programming technologies and techniques, and with different feature sets. The system is concerned with companies, departments, managers, and employees, and it supports functionality for totaling salaries, cutting salaries, computing other data, and checking data in some ways. The system may also be subject to additional capabilities---similar to non-functional requirements, e.g.: serialization, persistence, or logging. The following feature model breaks down all required or optional features of the system.

**Feature model**

[−] **101feature**

[−] **101basics**

*Company*

*Cut*

*Total*

[+] **101capabilities**

[+] **101extras**
101companies: Implementations

List of implementations

The pages for the implementations are members of the Wiki category 101implementation.

The pages were automatically derived from the README files from the repository of implementations.

[−] 101implementation
   antlr
   antlr2
   antlr3
   aspectJ
   atl
   atl2
   atl3
   clojure
   csharp
   csharpEntityFramework
   dom
   emf
   fsharp
   gwt
   haskell
   hibernate
   ...

101companies:Contributors

Want to contribute?
See the 101companies:Invitation page.

Contributors to the 101companies effort

- See Category:101contributor for a list of contributors to the 101companies:Implementations.
- See Special:ListUsers for a list of contributors to the 101companies wiki.
Specific 101companies:Implementations point out designated contributors and those should be contacted if necessary.

History of the 101companies effort

...
Category: 101contributor

This wiki category includes all pages of contributors of 101companies:Implementations. See also a detailed description of contributors to the 101companies effort.

Ontology

[−] 101contributor
Andrei Varanovich
Betim Sojeva
Ralf Lämmel
Thiago Tonelli Bartolomei
Thomas Schmorleiz
Tony Sloane
Vadim Zaytsev
Zef Hemel
101implementation: syb

Intent
--- An implementation in Haskell using SYB ---

Implementations
There is more basic, functional programming-based implementation in Haskell: 101implementation:haskell. The present implementation exercises an advanced programming technique: SYB style of generic programming.

Languages
- Haskell including SYB-related extensions

Technologies
- GHCi: Haskell interpreter used for testing (Version 6.12.3)

Features

Motivation
Several of the 101companies features essentially involve traversal over arbitrarily nested data. For instance, totaling all salaries requires the location of salary subterms at all levels in the term. Hence, support for traversals can significantly help here. In fact, SYB style of generic programming can be put to work, and the corresponding function implementations become drastically more concise.
101companies: Ontology

All concepts and instances of the ontology are rooted in these base concepts:

- **Language**: classification of software languages
- **Technology**: classification of programming technologies
- **Term**: concepts and instances for characterization
- **101feature**: features of the 101companies information system
- **101implementation**: implementations of the 101companies information system

MediaWiki is used in a certain way to represent the ontology.

Category tree

There is also a separate view available with a [fully expanded category tree](#) for the ontology.

Base concepts

**Concept Language**

Classification

[+] Language

...
Domain-specific capabilities
Domain-specific capabilities

- Domain-specific formatting, e.g.:
  - prettify links to be more category-aware
  - normalize and highlight “intent”

- Focus on domain ontology; redirect elsewhere for more.

- Provide domain-specific backlist-based indexes.

- Provide navigation information based on document type/structure

- Extend API for external tools, e.g., based on category tree

- Import README files into Wiki incl. indexing for contributors et al.
Ambiguous terminology
Unambiguous terminology

Intent
--- The capability for processing data in a stream as opposed to in-memory ---

Links

Citations

Technologies
- SAX

Remaining ontology
- XML processing
- Push-based XML parsing

Retrieved from "http://101companies.uni-koblenz.de/index.php/Streaming"
Category: Capability
Information overload (“Haskell”)

Haskell (programming language)

Haskell (pronounced /hæskəl/) is a standardized, general-purpose purely functional programming language, with non-strict semantics and strong static typing. It is named after logician Haskell Curry. In Haskell, “a function is a first-class citizen” of the programming language. As a functional programming language, the primary control construct is the function. The language is rooted in the observations of Haskell Curry and his intellectual descendants, that “a proof is a program; the formula it proves is a type for the program.”

Haskell 1.0

The first version of Haskell (version 1.0) was defined in 1990. The committee’s efforts resulted in a series of language definitions.

Haskell 98

In late 1997, the series culminated in Haskell 98, intended to specify a stable, minimal, portable version of the language and an accompanying standard library for Haskell, as a base for future extensions.

Haskell Prime

In early 2006, the process of defining a successor to the Haskell 98 standard, informally named “Haskell Prime” (Haskell Prime), was begun. This is an ongoing incremental process to revise the language definition, producing a new revision once per year. The first revision, named Haskell 2010, was announced in November 2009.

Haskell 2010

Haskell 2010 adds the Foreign Function Interface (FFI) to Haskell, allowing for bindings to other programming languages, fixes some syntax issues (changes in the formal grammar) and bans so-called “plus-k-patterns”, that is, definitions of the form  \( \text{let x = \ldots} \) in  \( \text{\ldots} \)  \( \text{\ldots} \)  \( \text{\ldots} \) are no longer allowed. It introduces the Language-Pragma-Syntax-Extension which allows for designating a haskell source as Haskell 2010 or requiring certain extensions to the Haskell language. The names of the extensions introduced in Haskell 2010 are DoInImport, HierarchicalModules, EmptyDataDeclarations, FixityResolution, ForeignFunctionInterface, LineCommentSyntax, PatternGuards, RelaxedDependencyAnalysis, LanguagePragma, NoPlusKPatterns.

Features

Main article: Haskell features

See also: Glasgow Haskell Compiler#Extensions to Haskell

Haskell features lazy evaluation, pattern matching, list comprehensions, types and type polymorphism. It is a purely functional language, which means that in general, functions in Haskell do not have side effects. There is a distinct type for representing side effects, orthogonal to the type of functions. A pure function may return a side effect which is subsequently executed, modeling the impure effects of other languages.

Haskell has a strong, static type system based on Hindley–Milner type inference. Haskell’s principal innovation in this area is to add type classes, which were originally conceived as a principled way to add overloading to the language, but have since found many more uses.

The type which represents side effects is an example of a monad. Monads are a general framework which can model different kinds of computation, including error handling, nondeterminism, parsing, and software transaction memory. Monads are defined as ordinary datatypes, but Haskell provides some syntactic sugar for their use.

The language has an open, published specification and multiple implementations exist.
Tailored information

Language:Haskell

From 101companies

Intent

--- the functional programming language ---

Resources

- haskell.org (http://www.haskell.org/haskellwiki/Haskell) (Home page)
- Haskell (http://en.wikipedia.org/wiki/Haskell_(programming_language)) (Wikipedia)

Citations

Technologies

- GHC
- GHCi

Languages

- Haskell 98

Implementations

- 101implementation:haskell
- 101implementation:syb

Remaining ontology

- Unspecific language

Retrieved from "http://101companies.uni-koblenz.de/index.php/Language:Haskell"
Categories: Functional programming language | Unspecific language
Navigation and indexing

Technology: GHCi

Discussion
GHCi is part of the GHC distribution.

Citations

Implementations
- 101implementation:haskell
- 101implementation:syb

Category: Interpreter
Megamodeling
What’s a megamodel?
What’s a megamodel?

Follow @meganalysis
A megamodel is model.
A megamodel is not a metamodel.
Modeling-biased definition:

A megamodel shows the relationships between models, metamodels, and model transformations.
Do we need megamodeling?
Empirical mega modeling

(ANTLR)

Company.g

cOMPANY : ...

sample.Company

cOMPANY "meganalysis" {
    ...
}

accept (or parse)

A megamodel

Meganalysis company
Specific megamodelling

\[ w \in L(G_{\text{Company}}) \]

...
We need to be more precise and more abstract.
Precise **mega**modeling

The *megamodel* meta*model*  

E ... “element of”  
C ... “conforms to”  
R ... “represents”  

ANTLR  

Company.g  

sample.Company  

meganalysis  

R  

E  

C
Basic megamodelling idioms
Entities

- Technologies
- Languages
- Elements of languages
  - Programs
  - Schemas
- Functions (arising through interpretation, for example)
- ...
Relationships

- E: elementOf
- C: conformsTo
- D: dependsOn
- S: subsetOf
- R: represents
- Function application
Languages

Java

Haskell
Artifacts and "elementOf"

HelloWorld.java

E

Java

HelloWorld.hs

E

Haskell
Conformance

```
HelloWorld.java  C  Java.g

HelloWorld.hs  C  Haskell.g
```
Application
Compilation
Interpretation

Haskell Expression
Interpretation cont’d
Examples
Interpretation of a Haskell program
Execution of a compiler Haskell program
Abstract from interpreter/compiler
An XSLT transformation
Abbreviated XSLT transformation
Transformation with DOM

Java program → javac → Bytecode → java → Transformation → Input → Output

Java → JAXP DOM → DOM → XML

Language
Abbreviated transformation with DOM
Transformation with JAXB
Wrap-up
BOF session

When? Today, 5pm

Where? Please see registration desk.

What?

- Show off with your favorite AOP technology
- Review 101companies approach
- Improve implementations and wiki
- Add new implementations
- Coordinate future efforts
TODO / Plans

- Publish over overall idea and megamodeling approach.
- Upscale ontology in a course next semester.
- Upload additional implementations.
- Tutorials at other PL/SE events.
- Set up regular online meetings.
Acknowledgement

Co-developers
- Thomas Schmorleiz
- Andrei Varanovich

Scientific collaboration
- Dragan Gasevic
- Jean-Marie Favre

See here for detailed credits: 101companies:Contributors
Thanks!

Thanks!