Verstehen von Technologien und Sprachen

http://www.softlang.org/course:techonto

Prof. Dr. R. Lämmel und Marcel Heinz (MSc)
AG Softwaresprachen
Universität Koblenz-Landau
http://www.softlang.org/startseite

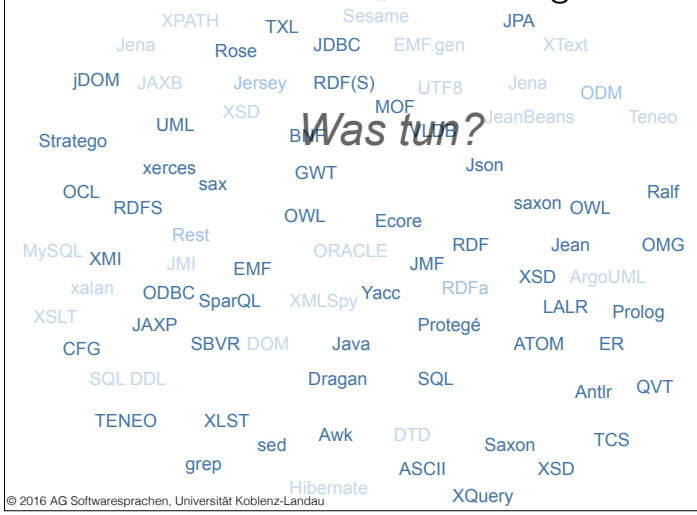
Danksagung an weitere Mitarbeiter und Studierende aus der Arbeitsgruppe: Andrei Varanovich, Lukas Härtel, Johannes Härtel.

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4 Informatik

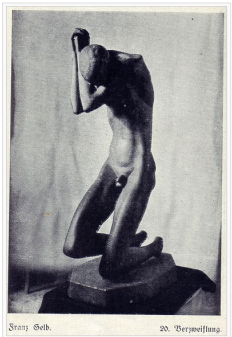
softlang

Das Problem der „Technologienfülle“



Konsequenzen des Problems der „Technologienfülle“

- Vendor lock-in
- Expertisenkorrision
- Technologienermüdung
- Job-Security für Entwickler?
- Kosten der Technologieeinführung
- …



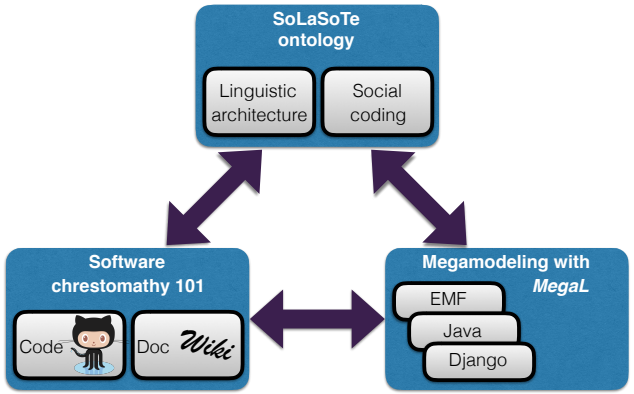
Franz Gelb: Verzweiflung, Kunststein, vor 1921. Abbildung aus dem Katalog "Das badische Kunstschaffen", Karlsruhe 1930

Quelle: http://de.wikipedia.org/wiki/Franz_Gelb

Lösungsansatz der Arbeitsgruppe für das Problem der „Technologienfülle“

- Illustrative Softwaresystemchen (101companies)
- Ontologie zur Erfassung der Konzepte (SoLaSoTe)
- Modellierung von Softwaretechnologien (MegaL)

Lösungsansatz der Arbeitsgruppe für das Problem der „Technologienfülle“



http://www.softlang.org/course:techonto

Gliederung

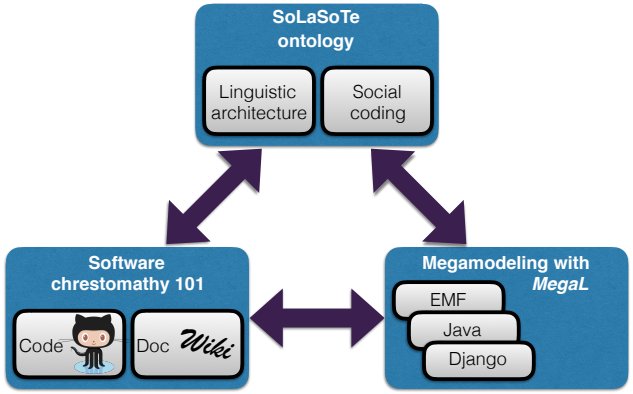
1. Willkommen **09:00-09:10**
2. **Befragung der Teilnehmer** **09:10-09:20**
3. Überblick über 101, SoLaSoTe und MegaL **09:20-10:15**
4. Klassifizierung von Sprachen, Technologien, u.ä. **10:30-11:15**
5. Beispielhafte Technologiemodelle **11:15-12:15**
6. Eine Kern-Ontologie zur Technologiemodellierung **13:00-13:45**
7. Weitere beispielhafte Technologiemodelle **14:00-14:45**
8. Prozess- und Werkzeugunterstützung **14:45-15:30**

http://www.softlang.org/course:techonto

Gliederung

1. Willkommen **09:00-09:10**
2. Befragung der Teilnehmer **09:10-09:20**
3. **Überblick über 101, SoLaSoTe und MegaL** **09:20-10:15**
4. Klassifizierung von Sprachen, Technologien, u.ä. **10:30-11:15**
5. Beispielhafte Technologiemodelle **11:15-12:15**
6. Eine Kern-Ontologie zur Technologiemodellierung **13:00-13:45**
7. Weitere beispielhafte Technologiemodelle **14:00-14:45**
8. Prozess- und Werkzeugunterstützung **14:45-15:30**

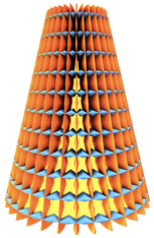
Lösungsansatz der Arbeitsgruppe für das Problem der „Technologienfülle“



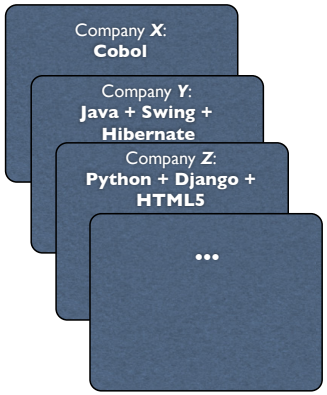
Die Software-Chrestomathie 101

101companies.org

Ein Gemeinschaftsprojekt zur Schaffung einer Wissensbasis über Softwaretechnologien und -sprachen sowie -konzepten auf der Basis (u.a.) der wiederkehrenden Implementation eines Systems für das Personalwesen.



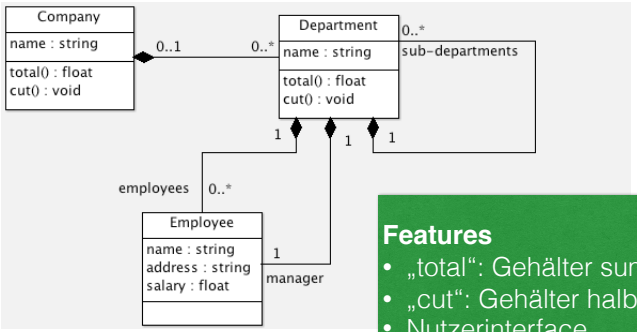
Warum der Name „*101companies*“?



Aus dem Englischen:

„101 ways of doing something“

Ein Informationssystem für das Personalwesen



Features

- „total“: Gehälter summieren
- „cut“: Gehälter halbieren
- Nutzerinterface
- Persistenz in der Datenbank
- …

Demo

http://101companies.org/wiki/Contribution:cobol

Headline

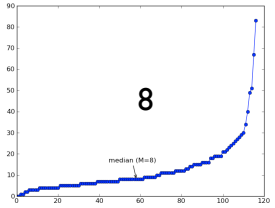
A simple implementation in Language:Cobol

Motivation

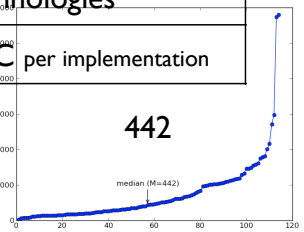
Cobol is a language that readily comes with means of data modeling and storage (persistence). Hence, it provides a good fit for the basic features and persistence.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. Total

ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
  FILE-CONTROL.
   COPY "copybooks/employee.fc".

DATA DIVISION.
 FILE SECTION.
  COPY "copybooks/employee.fd".

 WORKING-STORAGE SECTION.
  COPY "copybooks/file-status.ws".
  78 MEGANALYSIS    VALUE "meganalysis".
  01 TOTAL          PIC 9(9)V99.
```

Ein paar Zahlen zum Projekt „101“



#Files per implementation	Technologies
Languages	LOC per implementation



http://101companies.org/

What's 101?

Languages
101 aggregates information about software languages such as Haskell, XML, and SQL.

Technologies
101 organizes software technologies such as APIs, IDEs, build systems, and what have you.

Concepts
101 associates conceptual knowledge to various entities incl. languages and technologies.

Contributions
101 is a community project which relies on contributions ("code"!) "to explore all variables.

Contributors
101 relies on many contributors who code, document, and develop infrastructure.

Features
Contributions are assumed to exercise interesting features of the 101companies system.

Themes
The many implementations of 101 are organized in manageable, focused themes.

Courses
101 is leveraged in university lectures and professional training. Please join in!

Namespaces
There is a longer list of namespaces serving as major containers of knowledge items.

Wie benutzen wir 101?

* Moderne und praxisrelevante Kurskonzepte an der Universität:
  - Fortgeschrittene Programmierung
  - Einführung in die Funktionale Programmierung
* Material für IT-Schulungen
  - Einführung in die Objektorientierte Programmierung
  - Einführung in relationale Datenbanken
  - NoSQL-Technologien
  - Web-Programmierung
* Wissensbasis und Korpus für wissenschaftliche Untersuchungen

Die SoLaSoTe-Ontologie

http://www.softlang.org/solasote

- Entity types
  - System
  - Artifact
  - Fragment
  - Reference
  - Language
  - Technology
  - Function
  - Concept
- Relationship types
  - Membership
  - Conformance
  - Part-of
  - Definition
  - Implementation
  - Correspondence
  - Usage
  - Facilitation

Die SoLaSoTe-Ontologie

http://www.softlang.org/solasote

- Entity types
  - System *myWebApp*
  - Artifact *.java, .jar, .py, .html, …*
  - Fragment *methods, attributes, statement blocks, …*
  - Reference *URL, import, …*
  - Language *Python, HTML5, CSS, JavaScript, …*
  - Technology *Django, ASP.NET, Hibernate, …*
  - Function *Implemented behaviors*
  - Concept *MVC, Request, Response, REST, …*

Die SoLaSoTe-Ontologie

http://www.softlang.org/solasote

- Relationship types
  - Membership HelloWorld.java elementOf Java
  - Conformance *doc.xml conformsTo schema.xsd*
  - Part-of *javac partOf JDK*
  - Definition *java.g4 defines Java*
  - Implementation *method1 implements function2*
  - Correspondence schema.xsd correspondsTo Model.java
  - Usage *myWebApp uses Hibernate*
  - Facilitation *Hibernate facilitates O/R-mapping*

Technologiemodellierung mit MegaL

http://www.softlang.org/megal

Andere Arten von Modellen:

* Datenmodelle — z.B. ein XML-Schema
* Strukturmodelle — z.B. ein Klassendiagramm
* Verhaltensmodelle — z.B. ein Zustandsdiagramm
* Unternehmensmodelle—z.B. ein OrgChart
* Netzwerkmodelle—z.b. das OSI-Modell

What's the coolest language?

- Cobol
- Python
- Haskell
- Java

Vote

Datenmodell einer Wahl-Anwendung (ähnlich Doodle)

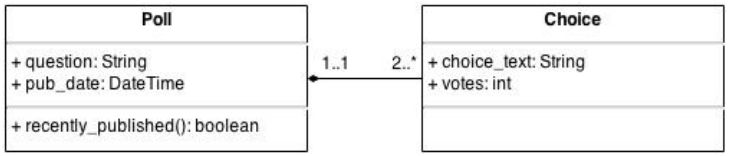
```
CREATE TABLE "polls_poll" (
    "id" integer NOT NULL PRIMARY KEY,
    "question" varchar(200) NOT NULL,
    "pub_date" datetime NOT NULL
)
;
CREATE TABLE "polls_choice" (
    "id" integer NOT NULL PRIMARY KEY,
    "poll_id" integer NOT NULL REFERENCES "polls_poll" ("id"),
    "choice" varchar(200) NOT NULL,
    "votes" integer NOT NULL
)
;
```

What's the coolest language?

- Cobol
- Python
- Haskell
- Java

Vote

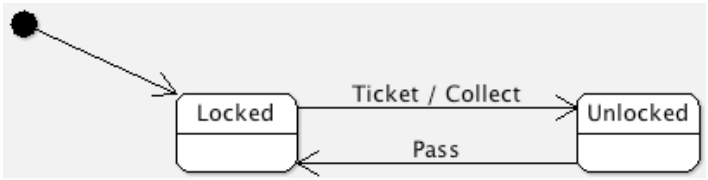
Strukturmodell einer Wahl-Anwendung (ähnlich Doodle)



Definition von Klassen, Attributen und Beziehungen sowie Kardinalitäten

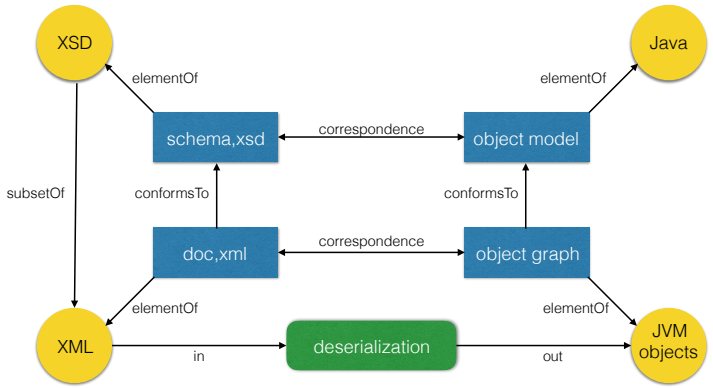
Quelle: http://commons.wikimedia.org/wiki/File:Drehtür%C3%B6r_Kaiserbad_Karlovy_Vary.jpg

Verhaltensmodell einer Drehtür in einem Hotel



Definition von Zuständen, Übergängen, Eingaben und Ausgaben.

Modell für eine Serialisierungstechnologie



http://www.softlang.org/course:techonto

Gliederung

1.	Willkommen	**09:00-09:10**
2.	Befragung der Teilnehmer	**09:10-09:20**
3.	Überblick über 101, SoLaSoTe und MegaL	**09:20-10:15**
4.	**Klassifizierung von Sprachen, Technologien, u.ä.**	**10:30-11:15**
5.	Beispielhafte Technologiemodelle	**11:15-12:15**
6.	Eine Kern-Ontologie zur Technologiemodellierung	**13:00-13:45**
7.	Weitere beispielhafte Technologiemodelle	**14:00-14:45**
8.	Prozess- und Werkzeugunterstützung	**14:45-15:30**

Was ist eine Softwaresprache?
(Computersprachen = Softwaresprachen!?)

Deutsches Wikipedia: "Computersprachen sind formale Sprachen, die zur Interaktion zwischen Menschen und Computern entwickelt wurden."

What is a computer language?

English Wikipedia:

Programming language, a formal language designed to communicate instructions to a machine, particularly a computer

Command language, a language used to control the tasks of the computer itself, such as starting other programs

Machine code or machine language, a set of instructions executed directly by a computer's central processing unit

Markup language, a grammar for annotating a document in a way that is syntactically distinguishable from the text

Style sheet language, is a computer language that expresses the presentation of structured documents

What is a computer language?

English Wikipedia :

Configuration language, a language used to write configuration files

Construction language, a general category that includes configuration languages, toolkit languages, and programming languages

Query language, used to make queries in databases and information systems

Modeling language, a formal language used to express information or knowledge, often for use in computer system design

Data serialization format.

What is a computer language?

A computer language is any language designed or used for automatic "information processing", i.e. data and process handling and management.

Anureev, I. S., et al. "On the problem of computer language classification." Joint NCC&IIS Bulletin, Series Computer Science 27 (2008): 1-20.

Ansätze zur Klassifizierung

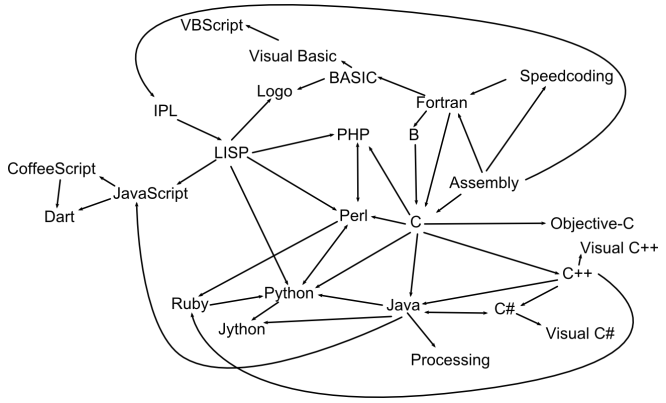
- Chronologische Klassifizierung mit Annotationen zu den Autoren, dem Zweck, etc.
- GPL versus DSL (Domäne)
- Paradigmen
- Mehrdimensionale Klassifizierung
- Artifakt (z.B. Programm versus Model)
- Zweck (z.B. Programmierung versus Modellierung)
- Technologieräume

Geschichtliche Betrachtung

- ~1950-65: Imperative Programmiersprachen für die Neumann-Architektur.
- Ende der 1960-er Jahre: Einführung von Spezifikations- und Datenrepräsentationssprachen. Weitere Paradigmen kamen dazu: die deklarative Programmierung (z.B. APL), die logische Programmierung und OO.

Quelle: http://thomasinterestingblog.wordpress.com/2011/11/26/the-family-tree-of-programming-languages/

Der Familienbaum für Programmiersprachen



Geschichtliche Betrachtung

- In den 1990-er Jahren: Neue Arten von Sprachen, z.B.: Wissenrepräsentationssprachen, Sprachen für parallele, nebenläufige, verteilte Ausführung sowie für Multi-Agenten-Systeme.
- Heutzutage gibt es “unzählige” Spezialsprachen (Domain-specific languages). Man unterscheidet auch gern Spezialsprachen in der Programmierung (DSL) versus der Modellierung (DSML).

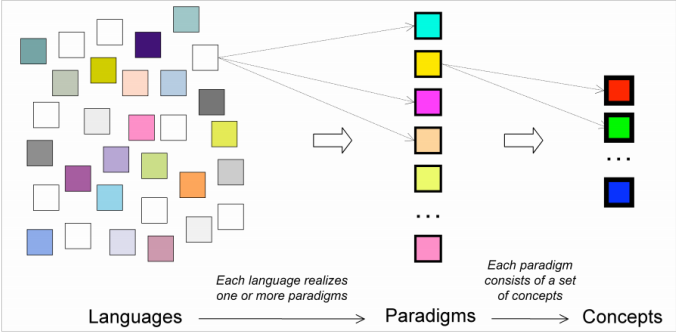
Programmierparadigmen

Prozeduren und Variablen

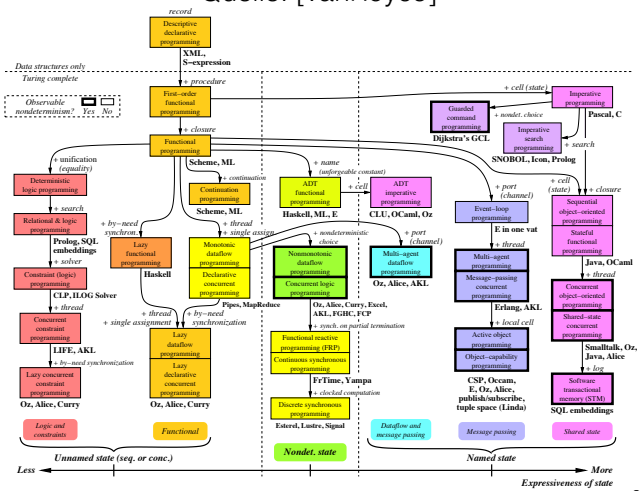
- Prozedurale (imperative) Programmierung
- Funktionale Programmierung — Mathematische Funktionen
- Logische Programmierung — Mathematische Logik
- Objektorientierte Programmierung — Objekte
- Multi-Paradigmen-Programmierung

Zugrundeliegende Prinzipien charakterisieren Programmierparadigmen.

Programmierparadigmen



Taxonomy of programming paradigms.
Quelle: [vanRoy09]



Artifakt-basierte Klassifizierung

Artifact	***Language***	***Example***
Program	*Programming language*	*Java*
Query	*Query language*	*XPath*
Transformation	*Transformation language*	*XSLT*
Model	*Modeling language*	*UML*
Specification	*Specification language*	*Alloy*
Data	*Data format*	*QTFF (QuickTime file format)*
Documentation	*Documentation language*	*DocBook*
Configuration	*Configuration language*	*INI file*
Log	*Log format*	*Common Log Format*
...	...	...

Zweck-basierte Klassifizierung

Artifact	***Purpose***	***Language***
Program	*Programming*	*Programming language*
Query	*Querying*	*Query language*
Transformation	*Transformation*	*Transformation language*
Model	*Modeling*	*Modeling language*
Specification	*Specification*	*Specification language*
Data	*Data storage/representation*	*Data format*
Documentation	*Documentation*	*Documentation language*
Configuration	*Configuration*	*Configuration language*
Log	*Logging*	*Log format*
...	...	...

Quellen zu Programmiersprachen

Books and research papers

Information retrieval technology is necessary

Wikipedia

WibiTaxonomy

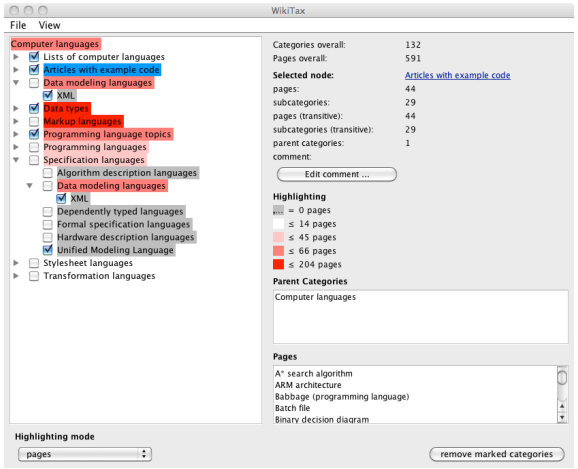
Wikidata

Dbpedia

BabelNet

~~Computer Language Classification Knowledge Portal~~

Ein kleiner Ausflug: WikiTax — ein Tool der Arbeitsgruppe zur Untersuchung und Bereinigung von Wikipedia-Kategorien



Reduzierte Subkategorien von Wikipedia's Kategorie "Computer Language"

Category	**Subcategories**
Data modeling languages	–
Markup languages	*Declarative markup languages, GIS file formats, Knowledge representation languages, Lightweight markup languages, Mathematical markup languages, Musical markup languages, Page description markup languages, Playlist markup languages, User interface markup languages, Vector graphics markup languages, Web syndication formats, XML markup languages*
Programming languages	*.NET programming languages, Agent-based programming languages, Agent-oriented programming languages, Concatenative programming languages, Concurrent programming languages, Data-structured programming languages, Declarative programming languages, Dependently typed languages, Domain-specific programming languages, Dynamic programming languages, Extensible syntax programming languages, Formula manipulation languages, Function-level languages, Functional languages, High Integrity Programming Language, High-level programming languages, ICL programming languages, Intensional programming languages, Low-level programming languages, Multi-paradigm programming languages, Nondeterministic programming languages, Object-based programming languages, Pattern matching programming languages, Procedural programming languages, Process termination functions, Prototype-based programming languages, Reactive programming languages, Secure programming languages, Set theoretic programming languages, Statically typed programming languages, Synchronous programming languages, Term-rewriting programming languages, Text-oriented programming languages, Tree programming languages, Visual programming languages, XML-based programming languages*
Specification languages	*Algorithm description languages, Dependently typed languages, Formal specification languages, Hardware description languages*
Stylesheet languages	–
Transformation languages	*Macro programming languages*

ages	*.NET programming languages, Agent-based programming languages, Agent-oriented programming languages, Concatenative programming languages, Concurrent programming languages, Data-structured programming languages, Declarative programming languages, Dependently typed languages, Domain-specific programming languages, Dynamic programming languages, Extensible syntax programming languages, Formula manipulation languages, Function-level languages, Functional languages, High Integrity Programming Language, High-level programming languages, ICL programming languages, Intensional programming languages, Low-level programming languages, Multi-paradigm programming languages, Nondeterministic programming languages, Object-based programming languages, Pattern matching programming languages, Procedural programming languages, Process termination functions, Prototype-based programming languages, Reactive programming languages, Secure programming languages, Set theoretic programming languages, Statically typed programming languages, Synchronous programming languages, Term-rewriting programming languages*

GPL versus DSL

- DSL: Domain-specific language
  - Domain-specific programming language (DSL)
    - Z.B.: Parsec (Eine Kombinatorbibliothek zum Parsen)
  - Domain-specific modeling language (DSML)
    - Z.B.: UML-Zustandsdiagramme

Apache Camel™

The Apache Software Foundation http://www.apache.org/

http://camel.apache.org/

Apache Camel ™ is a versatile open-source integration framework based on known Enterprise Integration Patterns.

Camel empowers you to define routing and mediation rules in a variety of domain-specific languages, including a Java-based Fluent API, Spring or Blueprint XML Configuration files, and a Scala DSL. This means you get smart completion of routing rules in your IDE, whether in a Java, Scala or XML editor.

Apache Camel uses URIs to work directly with any kind of Transport or messaging model such as HTTP, ActiveMQ, JMS, JBI, SCA, MINA or CXF, as well as pluggable Components and Data Format options. Apache Camel is a small library with minimal dependencies for easy embedding in any Java application. Apache Camel lets you work with the same API regardless which kind of Transport is used - so learn the API once and you can interact with all the Components provided out-of-box.

Apache Camel provides support for Bean Binding and seamless integration with popular frameworks such as CDI, Spring, Blueprint and Guice. Camel also has extensive support for unit testing your routes.

Case ID	Problem domain	Solution domain/ generation target
1	Telecom services	Configuration scripts
2	Insurance products	J2EE
3	Business processes	Rule engine language
4	Industrial automation	3 GL
5	Platform installation	XML
6	Medical device configuration	XML
7	Machine control	3 GL
8	Call processing	CPL
9	Geographic Information System	3 GL, propriety rule language, data structures
10	SIM card profiles	Configuration scripts and parameters
11	Phone switch services	CPL, Voice XML, 3 GL
12	eCommerce marketplaces	J2EE, XML
13	SIM card applications	3 GL
14	Applications in microcontroller	8-bit assembler
15	Household appliance features	3 GL
16	Smartphone UI applications	Scripting language
17	ERP configuration	3 GL
18	ERP configuration	3 GL
19	Handheld device applications	3 GL
20	Phone UI applications	C
21	Phone UI applications	C++
22	Phone UI applications	C
23	Phone UI applications	C++

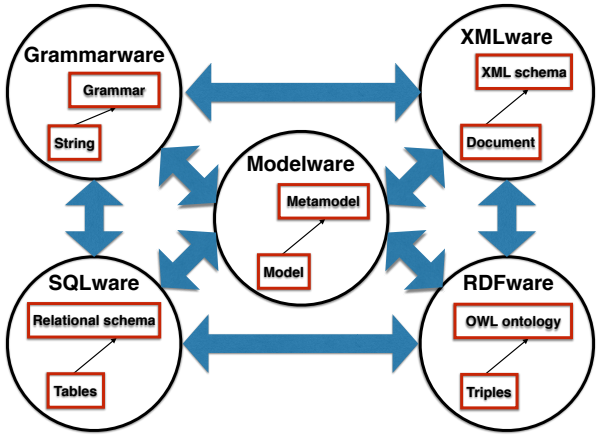
"Defining Domain-Specific Modeling Languages: Collected Experiences" by Janne Luoma, Steven Kelly, Juha-Pekka Tolvanen

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.6595&rep=rep1&type=pdf

GPLs versus DSLs
Quelle: [VBD+13]

	GPLs	DSLs
Domain	large and complex	smaller and well-defined
Language size	large	small
Turing completeness	always	often not
User-defined abstractions	sophisticated	limited
Execution	via intermediate GPL	native
Lifespan	years to decades	months to years (driven by context)
Designed by	guru or committee	a few engineers and domain experts
User community	large, anonymous and widespread	small, accessible and local
Evolution	slow, often standardized	fast-paced
Deprecation/incompatible changes	almost impossible	feasible

Technologieräume



Technological space
= Technology and community context in software engineering

- UMLware — Mainstream software modeling
- MDEware — Model Driven Engineering
- Javaware — Mainstream OO programming
- XMLware — Interoperability in data exchange
- SQLware — Mainstream database management
- Pythonware — Scripting and frameworks
- RDFware — Semantic Web and Linked Data
- JSONware — Modern interoperability
- COBOLware — Legacy programming technology

Technological Spaces: an Initial Appraisal•

Ivan Kurtev [1], Jean Bézivin [2], Mehmet Aksit [1]

[1] Software Engineering Group (TRESE), University of Twente, The Netherlands
{kurtev, aksit}@cs.utwente.nl

[2] Faculty of Sciences, University of Nantes, France
bezivin@sciences.univ-nantes.fr

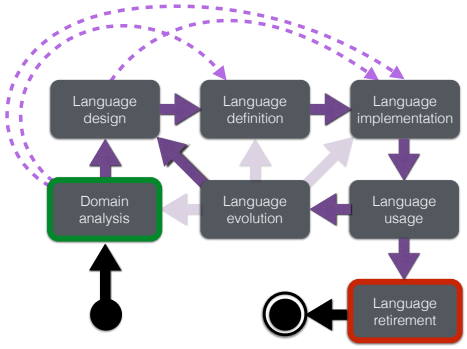
A technological space is **a working context with a set of associated concepts, body of knowledge,tools, required skills, and possibilities**. It is often associated to a given user community with shared know-how, educational support, common literature and even workshop and conference meetings. It is at the same time a zone of established expertise and ongoing research and a repository for abstract and concrete resources.

Bestandteile eines Technologieraumes

- Anwendungsszenarien
- Programmiersprachen
- Formate zur Datenrepräsentation
- Entwicklungswerkzeuge wie Compiler und IDEs
- Laufzeitwerkzeuge wie Bibliotheken oder Rahmenwerke
- Anfrage- und Transformationssprachen
- Lehrbücher und andere Wissensquellen
- Konferenzen und andere “Community resources”
- …

Weitere Technologieräume

Klassifizierung im Kontext: Der Lebenszyklus von Sprachen



Klassifizierung von Technologien

- Laufzeit
  - Framework (Rahmenwerk)
  - API (Schnittstelle)
  - Library (Bibliothek)
  - …
- Entwurfszeit
  - Compiler
  - Analyzer (Analysetool)
  - Generator
  - IDE
  - …
- Ein-/Ausgabe
  - Transformation
    - Homogen (Refactoring)
    - Heterogen (Übersetzung)
  - Überprüfung
  - Analyse

Klassifizierung von Artifakten

- Manifestation
  - File
  - Folder
  - Request
  - Response
  - Object
- Access/reference
  - URL
  - URI
  - UUID
  - Fragment
  - Archive element
  - Repo element
- Role
  - Input, Output
  - Program
  - Grammar
  - Model, metamodel
  - Model, view, controller
- Representation
  - Text (string)
  - Tree (XML, JSON)
  - Graph (objects, …)

Weitere Dimensionen

- Programmiertechniken
  - Rekursion
  - Iteration
  - Verteilung
  - Parsing
  - Formatting
  - Serialisierung
  - …
- Entwurfsmuster
  - MVC
  - Beobachter
  - Kompositum
  - Besucher
  - …
- Protocol
  - Three-way handshake
  - HTTP
  - Rest
  - …

Offene Fragen

- Welches sind die wichtigen Eigenschaften bzw. Attribute bzw. Dimensionen für eine Klassifizierung?
- Wie kann man auf der Basis existierender Quellen einen “Gold Standard” / eine Referenz erstellen und diese Referenz fortwährend mit gemeinschaftlicher Beteiligung warten?

Ein Ausflug: Vergleichende Betrachtung von Sprachen auf der Basis von Chrestomathien

Eine **Chrestomathie** (griechisch χρηστομαθία, zu χρηστός *chrestós* ‚nützlich‘ und μανθάνω *manthánо* mit dem Infinitiv Aorist μαθεῖν *matheín* ‚lernen‘) ist eine Zusammenstellung von Texten oder Textauszügen – hauptsächlich aus Prosaschriften – zu didaktischen Zwecken.[1]

Programmchrestomathien

- Hello World: http://www.roesler-ac.de/wolfram/hello.htm
- 99 bottles of beer: http://99-bottles-of-beer.net/
- Factorial: http://www.willamette.edu/~fruehr/haskell/evolution.html
- Fibonacci sequence: http://cubbi.com/fibonacci.html
- OO shapes: http://www.angelfire.com/tx4/cus/shapes/
- Literate programs: http://en.literateprograms.org/
- Rosetta Code: http://rosettacode.org/
- 101companies: http://101companies.org/

Siehe auch http://rosettacode.org/wiki/Help:Similar_Sites

Hello World

http://www.roesler-ac.de/wolfram/hello.htm

The Hello World Collection

"Hello World" is the first program one usually writes when learning a new programming language. The first Hello World program appeared in chapter 1.1 of the first edition of Kernighan & Ritchie's original book about C, "The C Programming Language", in 1978 and read like this:

```
main() {
    printf("hello, world\n");
}
```

Since then, Hello World has been implemented in just about every programming language on the planet. This collection includes **441 Hello World programs** in many more-or-less well known programming languages, plus **64 human languages**.

The programs in this collection are intended to be as minimal as possible in the respective language. They are meant to demonstrate how to output Hello World as simply as possible, not to show off language features. For a collection of programs that tell more about what programming in the languages actually is like, have a look at the 99 Bottles of Beer collection.

99 bottles of beer

Lyrics of the song 99 Bottles of Beer

99 bottles of beer on the wall, 99 bottles of beer.
Take one down and pass it around, 98 bottles of beer on the wall.

98 bottles of beer on the wall, 98 bottles of beer.
Take one down and pass it around, 97 bottles of beer on the wall.

97 bottles of beer on the wall, 97 bottles of beer.
Take one down and pass it around, 96 bottles of beer on the wall.

96 bottles of beer on the wall, 96 bottles of beer.
Take one down and pass it around, 95 bottles of beer on the wall.

95 bottles of beer on the wall, 95 bottles of beer.
Take one down and pass it around, 94 bottles of beer on the wall.

94 bottles of beer on the wall, 94 bottles of beer.
Take one down and pass it around, 93 bottles of beer on the wall.

A Java version

http://www.99-bottles-of-beer.net/language-java-4.html?PHPSESSID=32269dee0fedac31dc90739c31ab45f9

```
class bottles {
  public static void main(String args[]) {
    String s = "s";
    for (int beers=99; beers>-1;) {
      System.out.print(beers + " bottle" + s + " of beer on the wall, ");
      System.out.println(beers + " bottle" + s + " of beer, ");
      if (beers==0) {
        System.out.print("Go to the store, buy some more, ");
        System.out.println("99 bottles of beer on the wall.\n");
        System.exit(0);
      } else
        System.out.print("Take one down, pass it around, ");
      s = (--beers == 1) ? "" : "s";
      System.out.println(beers + " bottle" + s + " of beer on the wall.\n");
    }
  }
}
```

http://www.softlang.org/course:techonto

Gliederung

1. Willkommen **09:00-09:10**
2. Befragung der Teilnehmer **09:10-09:20**
3. Überblick über 101, SoLaSoTe und MegaL **09:20-10:15**
4. Klassifizierung von Sprachen, Technologien, u.ä. **10:30-11:15**
5. **Beispielhafte Technologiemodelle** **11:15-12:15**
6. Eine Kern-Ontologie zur Technologiemodellierung **13:00-13:45**
7. Weitere beispielhafte Technologiemodelle **14:00-14:45**
8. Prozess- und Werkzeugunterstützung **14:45-15:30**

Beispielhafte Technologiemodelle

- Kompilation mit dem javac Compiler für Java
- Einstellen von Daten in eine MySQL-Datenbank
- XML/object-Mapping mit der JAXB-Technologie für Java

Wir benutzen die textuelle Notation der MegaL-Sprache für die Megamodellierung. MegaL wird in der AG Softwaresprachen entwickelt.

Compilation with Java's javac compiler

http://en.wikipedia.org/wiki/Javac

Explanation

Given a source file (a Java program), the compiler produces bytecode (a .class file). (The bytecode could be executed directly by the JVM (Java Virtual Machine); execution is not covered by the following megamodel.)

Compilation with Java's javac compiler

Illustration

```
$ ls HelloWorld.*
HelloWorld.java
$ more HelloWorld.java
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, World");
    }

}
$ javac HelloWorld.java
$ ls HelloWorld.*
HelloWorld.class
HelloWorld.java
```

Compilation with Java's javac compiler

- Java : Language — the programming language Java
- JavaByteCode : Language — the bytecode language of Java/JVM
- Compiler < Technology
- javac : Compiler — https://en.wikipedia.org/wiki/Javac
- aJavaProgram : Artifact — e.g., HelloWorld.java (placeholder)
- aJavaByteCodeProgram : Artifact — e.g., HelloWorld.class (placeholder)
- aJavaProgram ∈ Java
- aJavaByteCodeProgram ∈ JavaByteCode
- compilation : Java → JavaByteCode
- javac implements compilation
- compilation(aJavaProgram) ↦ aJavaByteCodeProgram

Model

Let's look at the language elements of MegaL in some detail.

Compilation with Java's javac compiler

- **Java : Language**
- **JavaByteCode : Language**
- Compiler < Technology
- **javac : Compiler**
- **aJavaProgram : Artifact** — e.g., HelloWorld.java
- **aJavaByteCodeProgram : Artifact** — e.g., HelloWorld.class
- aJavaProgram ∈ Java
- aJavaByteCodeProgram ∈ JavaByteCode
- **compilation : Java → JavaByteCode**
- javac implements compilation
- compilation(aJavaProgram) ↦ aJavaByteCodeProgram

„Java" is an entity of type „Language".

„javac" is an entity of type „Compiler".

MegaL explained: **Entity declarations**

compilation is an function entity.

Compilation with Java's javac compiler

- Java : Language
- JavaByteCode : Language
- **Compiler < Technology**
- javac : Compiler
- aJavaProgram : Artifact — e.g., HelloWorld.java
- aJavaByteCodeProgram : Artifact — e.g., HelloWorld.class
- aJavaProgram ∈ Java
- aJavaByteCodeProgram ∈ JavaByteCode
- compilation : Java → JavaByteCode
- javac implements compilation
- compilation(aJavaProgram) ↦ aJavaByteCodeProgram

A new subtype

MegaL explained: **Entity TYPE declarations**

Compilation with Java's javac compiler

- Java : Language
- JavaByteCode : Language
- Compiler < Technology
- javac : Compiler
- aJavaProgram : Artifact — e.g., HelloWorld.java
- aJavaByteCodeProgram : Artifact — e.g., HelloWorld.class
- **aJavaProgram ∈ Java**
- **aJavaByteCodeProgram ∈ JavaByteCode**
- compilation : Java → JavaByteCode
- **javac implements compilation**
- **compilation(aJavaProgram) ↦ aJavaByteCodeProgram**

MegaL explained: **Relationship declarations**

A technology implements a function

A function application expresses data flow

Population of a relational database

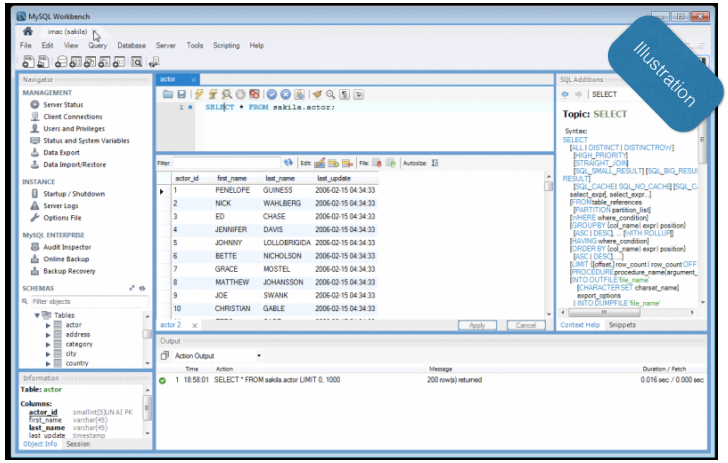
Explanation

The RDBMS shell (a workbench or monitor) is used to populate a relational database. To this end, the database schema (some CREATE TABLE statements) are executed, followed by another script with sample data (some INSERT statements).

We will be using MySQL as the RDBMS.
https://www.mysql.com/

https://www.mysql.com/products/workbench/

Illustration



Illustration

```
DROP TABLE IF EXISTS employee;
DROP TABLE IF EXISTS department;

# Departments

CREATE TABLE IF NOT EXISTS department (
        id INTEGER AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(100) NOT NULL,
        did INTEGER,
        FOREIGN KEY (did) REFERENCES department(id) ON DELETE CASCADE ON UPDATE CASCADE
);

# Employees

CREATE TABLE IF NOT EXISTS employee (
        id INTEGER AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(50) NOT NULL,
        address VARCHAR(50) NOT NULL,
        salary DOUBLE NOT NULL,
        manager BOOL NOT NULL,
        did INTEGER NOT NULL,
        FOREIGN KEY (did) REFERENCES department(id) ON DELETE CASCADE ON UPDATE CASCADE
);
```


Illustration

```
# Departments

INSERT INTO department (name) VALUES ("Research"); -- deptId = 1
INSERT INTO department (name) VALUES ("Development"); -- deptId = 2
INSERT INTO department (name,did) VALUES ("Dev1",2); -- deptId = 3
INSERT INTO department (name,did) VALUES ("Dev1.1",3); -- deptId = 4

# Employees

INSERT INTO employee (name, address, salary, manager, did)
SELECT "Craig", "Redmond", 123456, true, 1
UNION ALL
SELECT "Ray", "Redmond", 234567, true, 2
UNION ALL
SELECT "Klaus", "Boston", 23456, true, 3
UNION ALL
SELECT "Karl", "Riga", 2345, true, 4
UNION ALL
SELECT "Erik", "Utrecht", 12345, false, 1
UNION ALL
SELECT "Ralf", "Koblenz", 1234, false, 1
UNION ALL
SELECT "Joe", "Wifi City", 2344, false, 4;
```

Population of a MySQL database

- Data definition language < Language
- Data manipulation language < Language
- SQL : Language
- SQL DDL : Data definition language
- SQL DML : Data manipulation language
- SQL DDL ⊆ SQL —- all CREATE TABLE st.
- SQL DML ⊆ SQL —- all CRUD statements
- RDBMS < Technology
- MySQL : RDBMS
- IDE < Technology
- MySQL Workbench : IDE

- createStmts : Artifact
- insertStmts : Artifact
- createStmts ∈ SQL DDL
- insertStmts ∈ SQL DML
- db_1, db_2, db_3: Artifact
- DbImage : Language
- db_1, db_2, db_3 ∈ DbImage
- execution : SQL × DbImage → DbImage
- MySQL Workbench implements execution
- execution(createStmts, db_1) ↦ db_2
- execution(insertStmts, db_2) ↦ db_3

Model summary

Population of a MySQL database

- **Data definition language < Language**
- **Data manipulation language < Language**
- **SQL : Language**
- **SQL DDL : Data definition language**
- **SQL DML : Data manipulation language**
- **SQL DDL ⊆ SQL**
- **SQL DML ⊆ SQL**
- RDBMS < Technology
- MySQL : RDBMS
- IDE < Technology
- MySQL Workbench : IDE

- createStmts : Artifact
- insertStmts : Artifact
- createStmts ∈ SQL DDL
- insertStmts ∈ SQL DML
- db_1, db_2, db_3: Artifact
- DbImage : Language
- db_1, db_2, db_3 ∈ DbImage
- execution : SQL × DbImage → DbImage
- MySQL Workbench implements execution
- execution(createStmts, db_1) ↦ db_2
- execution(insertStmts, db_2) ↦ db_3

Taking apart SQL into DDL and DML

Population of a MySQL database

- Data definition language < Language
- Data manipulation language < Language
- SQL : Language
- SQL DDL : Data definition language
- SQL DML : Data manipulation language
- SQL DDL ⊆ SQL
- SQL DML ⊆ SQL
- **RDBMS < Technology**
- **MySQL : RDBMS**
- **IDE < Technology**
- **MySQL Workbench : IDE**

- createStmts : Artifact
- insertStmts : Artifact
- createStmts ∈ SQL DDL
- insertStmts ∈ SQL DML
- db_1, db_2, db_3: Artifact
- DbImage : Language
- db_1, db_2, db_3 ∈ DbImage
- execution : SQL × DbImage → DbImage
- MySQL Workbench implements execution
- execution(createStmts, db_1) ↦ db_2
- execution(insertStmts, db_2) ↦ db_3

Identification of involved technologies

Population of a MySQL database

- Data definition language < Language
- Data manipulation language < Language
- SQL : Language
- SQL DDL : Data definition language
- SQL DML : Data manipulation language
- SQL DDL ⊆ SQL
- SQL DML ⊆ SQL
- RDBMS < Technology
- MySQL : RDBMS
- IDE < Technology
- MySQL Workbench : IDE

- **createStmts : Artifact**
- **insertStmts : Artifact**
- **createStmts ∈ SQL DDL**
- **insertStmts ∈ SQL DML**
- **db_1, db_2, db_3: Artifact** — state of database
- **DbImage : Language** — MySQL format
- **db_1, db_2, db_3 ∈ DbImage**
- execution : SQL × DbImage → DbImage
- MySQL Workbench implements execution
- execution(createStmts, db_1) ↦ db_2
- execution(insertStmts, db_2) ↦ db_3

Identification of involved artifacts, also subject to a language for the database image

Population of a MySQL database

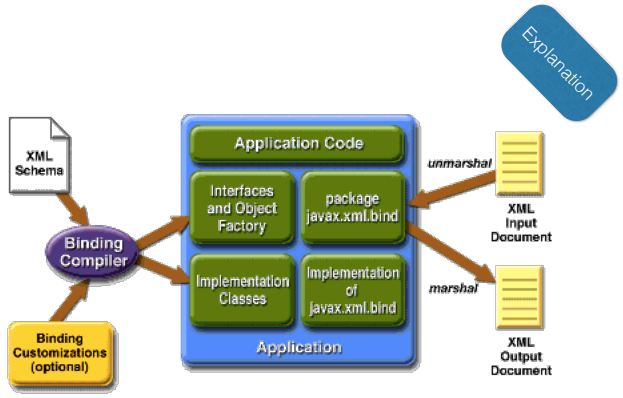
- Data definition language < Language
- Data manipulation language < Language
- SQL : Language
- SQL DDL : Data definition language
- SQL DML : Data manipulation language
- SQL DDL ∈ SQL
- SQL DML ∈ SQL
- RDBMS < Technology
- MySQL : RDBMS
- IDE < Technology
- MySQL Workbench : IDE

Script execution with the help of the workbench resulting in changed images

- createStmts : Artifact
- insertStmts : Artifact
- createStmts ∈ SQL DDL
- insertStmts ∈ SQL DML
- db_1, db_2, db_3: Artifact
- DbImage : Language
- db_1, db_2, db_3 ∈ DbImage
- **execution : SQL × DbImage → DbImage**
- **MySQL Workbench implements execution**
- **execution(createStmts, db_1) ↦ db_2**
- **execution(insertStmts, db_2) ↦ db_3**

XML data binding with Java's JAXB technology

http://en.wikipedia.org/wiki/Java_Architecture_for_XML_Binding



Source: http://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/JAXBWorks2.html

A megamodel for JAXB (XML-data binding of the Java platform)

- Platform < Technology
- XML data binding technology < Technology
- Code generator < Technology
- Library < Technology — or framework or API

The types of technologies involved

- Java platform : Platform
- Java : Language
- Java platform implements Java

The overall Java platform which JAXB is part of

- JAXB : XML data binding technology
- JAXB partOf Java platform
- bindingCompiler : Code generator
- bindingFramework : Library
- bindingCompiler partOf JAXB
- bindingFramework partOf JAXB

The relevant parts of JAXB
https://jaxb.java.net/2.2.4/docs/xjc.html
https://docs.oracle.com/javaee/5/api/javax/xml/bind/package-summary.html

Part 1/5: Technology break-down

A megamodel for JAXB (XML-data binding of the Java platform)

- XML : Language — all of XML
- XSD : Language — XML schema
- XSD ⊆ XML — schemas are written in XML

XSD is the XML schema language; it happens to be an XML language itself.

- JAXB.Java : Language
- JAXB.Java ⊆ Java
- generation : XSD → JAXB.Java

We introduce the name „JAXB.Java" to refer to the specific Java subset that is used by JAXB's code generator.

- xmlTypes : File+
- xmlTypes ∈ XSD
- javaClasses : File+
- javaClasses ∈ JAXB.Java
- bindingCompiler implements generation
- generation(xmlTypes) ↦ javaClasses

The relevant artifacts, i.e., XML types as input for code generation and Java classes as output. (The types and classes may be distributed over multiple files — thus the „+".)

Part 2/5: Type-level mapping including data flow

XML schema for “companies”

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema xmlns="http://www.company.softlang.org/company.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="http://www.company.softlang.org/company.xsd">

    <xs:element name="company">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="name"/>
                <xs:element maxOccurs="unbounded" minOccurs="0" ref="department"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="department">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="name"/>
                <xs:element name="manager" type="employee"/>
                <xs:element maxOccurs="unbounded" minOccurs="0" ref="department"/>
                <xs:element maxOccurs="unbounded" minOccurs="0" name="employee" type="employee"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="employee">
        <xs:sequence>
            <xs:element ref="name"/>
            <xs:element ref="address"/>
            <xs:element ref="salary"/>
        </xs:sequence>
    </xs:complexType>

    ...
```


Generated Java code for “companies” (simplified)

```java
public class Company {
    protected String name;
    protected List<Department> department;
    public String getName() {  return name;  }
    public void setName(String value) {  this.name = value; }
    public List<Department> getDepartment() {
        if (department == null) {
            department = new ArrayList<Department>();
        }
        return this.department;
    }
}
```


Generated Java code for “companies”

```java
//
// This file was generated by the JavaTM Architecture for XML
Binding(JAXB) Reference Implementation, v2.2.4-2
// See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/
xml/jaxb</a>
// Any modifications to this file will be lost upon recompilation of the
source schema.
// Generated on: 2015.07.01 at 12:14:10 PM CEST
//


package org.softlang.company.xjc;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for anonymous complex type.
 *
 * <p>The following schema fragment specifies the expected
content contained within this class.
 *
 * <pre>
 * &lt;complexType>
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}
anyType">
 *       &lt;sequence>
 *         &lt;element ref="{http://www.company.softlang.org/
company.xsd}name"/>
 *         &lt;element ref="{http://www.company.softlang.org/
company.xsd}department" maxOccurs="unbounded"
minOccurs="0"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "name",
    "department"
})
@XmlRootElement(name = "company")
public class Company {

    @XmlElement(required = true)
    protected String name;
    protected List<Department> department;

    /**
     * Gets the value of the name property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getName() {
        return name;
    }

    /**
     * Sets the value of the name property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setName(String value) {
        this.name = value;
    }

    /**
     * Gets the value of the department property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for
the department property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getDepartment().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link Department }
     *
     *
     */
    public List<Department> getDepartment() {
        if (department == null) {
            department = new ArrayList<Department>();
        }
        return this.department;
    }

}
```



```java
//
// This file was generated by the JavaTM Architecture for XML Binding(JAXB) Reference Implementation, v2.2.4-2
// See <a href="http://java.sun.com/xml/jaxb">http://java.sun.com/xml/jaxb</a>
// Any modifications to this file will be lost upon recompilation of the source schema.
// Generated on: 2015.07.01 at 12:14:10 PM CEST
//


package org.softlang.company.xjc;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for anonymous complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this class.
 *
 * <pre>
 * &lt;complexType>
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element ref="{http://www.company.softlang.org/company.xsd}name"/>
 *         &lt;element ref="{http://www.company.softlang.org/company.xsd}department" maxOccurs="unbounded"
minOccurs="0"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
```

The comment is the “watermark” by the binding compiler.

These imports also concern the binding framework.

```
/**
 * <p>Java class for anonymous complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this class.
 *
 * <pre>
 * &lt;complexType>
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element ref="{http://www.company.softlang.org/company.xsd}name"/>
 *         &lt;element ref="{http://www.company.softlang.org/company.xsd}department" maxOccurs="unbounded"
minOccurs="0"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "name",
    "department"
})
@XmlRootElement(name = "company")
public class Company {

    @XmlElement(required = true)
    protected String name;
    protected List<Department> department;

    /**
     * Gets the value of the name property.
```

The Java comment captures the underlying schema types.


```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "name",
    "department"
})
@XmlRootElement(name = "company")
public class Company {

    @XmlElement(required = true)
    protected String name;
    protected List<Department> department;

    /**
     * Gets the value of the name property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getName() {
        return name;
    }

    /**
     * Sets the value of the name property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setName(String value) {
        this.name = value;
    }
```

These annotations are used for validation and serialization.


```
    @XmlElement(required = true)
    protected String name;
    protected List<Department> department;

    /**
     * Gets the value of the name property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getName() {
        return name;
    }

    /**
     * Sets the value of the name property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setName(String value) {
        this.name = value;
    }

    /**
     * Gets the value of the department property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
```



```
    public void setName(String value) {
        this.name = value;
    }

    /**
     * Gets the value of the department property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the department property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getDepartment().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link Department }
     *
     *
     */
    public List<Department> getDepartment() {
        if (department == null) {
            department = new ArrayList<Department>();
        }
        return this.department;
    }

}
```

A megamodel for JAXB (XML-data binding of the Java platform)

- Annotation : Concept — here: Java annotations used as metadata for Java code
- bindingFramework facilitates Annotation — binding framework exports JAXB annotations
- javaClasses uses Annotation — generated classes use JAXB annotations
- Serialization : Concept — here: object serialization (aka un-/marshalling)
- bindingFramework facilitates Serialization — binding framework used for serialization
- Code generation : Concept — here: the generation of Java code from XML schemas
- bindingCompiler facilitates Code generation — binding compiler used for code generation
- bindingCompiler uses Code generation — binding compiler uses code generation itself
- Validation : Concept — schema-based validation or conformance checking
- JAXB facilitates Validation — Validation is a byproduct of XML-data binding

Part 3/5: Involved concepts

A megamodel for JAXB (XML-data binding of the Java platform)

- JVM.ObjectGraphs : Language
- anObjectGraph : Transient
- anObjectGraph ∈ JVM.ObjectGraphs

The „language“ for run-time object graphs

The type „Transient“ conveys that we face a run-time artifact.

- anXmlDoc : File
- anXmlDoc ∈ XML
- unmarshalling : XML → JVM.ObjectGraphs
- unmarshalling(anXmlDoc) ↦ anObjectGraph

We take an XML document as input and somehow invoke „unmarshalling“ to retrieve a run-time object graph.

- application : File+
- application ∈ Java
- aMethodCall : Fragment
- aMethodCall partOf application
- aMethodCall uses bindingFramework
- (aMethodCall refersTo bindingFramework)
- aMethodCall defines unmarshalling

The unmarshalling function arises as the meaning of a code fragment that is part of the application that uses JAXB. That code clearly uses and refers to the JAXB library.

Part 4/5: Instance-level mapping including data flow

Java code of application for *de-serializing* “companies”

```
public static Company deserializeCompany(File input)
throws JAXBException
{
   initializeJaxbContext();
   Unmarshaller unMarshaller = jaxbContext.createUnmarshaller();
   return (Company) unMarshaller.unmarshal(input);
}
```

aMethodCall

Java code of application for ***serializing*** “companies”

```
public static void serializeCompany(File output, Company c)
throws
        JAXBException,
        FileNotFoundException,
        XMLStreamException
{
   initializeJaxbContext();
   OutputStream os = new FileOutputStream(output);
   Marshaller marshaller = jaxbContext.createMarshaller();
   XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();
   XMLStreamWriter writer = outputFactory.createXMLStreamWriter(os);
   marshaller.marshal(c, writer);
}
```

A megamodel for JAXB (XML-data binding of the Java platform)

- anXmlDoc conformsTo xmlTypes

  Not an arbitrary document — rather one that conforms to the given schema!

- anObjectGraph conformsTo javaClasses

  Not an arbitrary object graph — rather one whose class is part of the given classes!

- xmlTypes correspondsTo javaClasses

  XML schema types and Java generated classes are very "similar" in structure — we call this correspondence.

- anXmlDoc correspondsTo anObjectGraph

  XML document and object graph obtained by deserialization are also very "similar" in structure — this is another instance of correspondence.

Part 5/5: Conformance and correspondence

XML schema-to-generated Java classes correspondence

XML schema

```
<xs:schema xmlns="http://www.company.softlang.org/
    <xs:element name="company">
    <xs:element name="department">
    <xs:complexType name="employee">
    <xs:element name="name" type="xs:string"/>
    <xs:element name="address" type="xs:string"/>
    <xs:element name="salary" type="xs:double"/>
</xs:schema>
```

There is a definition of companies, departments, and employees on both sides. The structures aren't completely equal, of course.

Java

- Company.java
- Department.java
- Employee.java
- ObjectFactory.java

XML schema-to-generated Java classes correspondence

XML schema

```
<xs:element name="company">
<xs:element name="department">
<xs:complexType name="employee">
    <xs:sequence>
        <xs:element ref="name"/>
        <xs:element ref="address"/>
        <xs:element ref="salary"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="name" type="xs:string"/>
<xs:element name="address" type="xs:string"/>
<xs:element name="salary" type="xs:double"/>
```

Java

- Company.java
- Department.java
- Employee.java
  - Employee
    - address
    - name
    - salary
    - getAddress() : String
    - getName() : String
    - getSalary() : double
    - setAddress(String) : void
    - setName(String) : void
    - setSalary(double) : void
- ObjectFactory.java

The similarity continues at the level of the individual types with interesting differences. For instance, we face getters and setters in Java.

Summary of megamodeling

- **Entities** in software development
  - e.g.: Java, Python, J2EE, Django, Testing, Inheritance
- **Entity types** in software development
  - e.g.: Language, Technology, Artifact, Concept
- **Relationships** in software development
  - e.g.:
    - HelloWorld.java ∈ Java
    - Django *uses* Python
- **Relationship types** in software development
  - e.g., „∈“ or „uses“

Think of relationships as edges in a graph with entities as the nodes.

Entity types

- **Predefined base types**
  - **Language** — conceptual entities (possibly thought of as sets) for languages
  - **Technology** — conceptual entities for technologies
  - **Artifact** — „manifested“ / „physical“ entities, e.g., a file
  - **System** — a conglomeration of artifacts making up a system
  - **Function** — mathematical functions on languages or actions
  - **Concept** — programming techniques or other concepts in software development

One could introduce more base types*!*

- **People** — human beings such as developers, stakeholders, etc.
- **Organization** — enterprises and other kinds organizations
- …

Before introducing a new base type, make sure that it is not better taken care of as a subtype of a predefined entity type.

Base entity type **Language**

- Definition:
  - *An artificial language used in software development*
- Subtypes of *Language*
  - Programming language: *Java, Python, Ruby*, …
  - Query language: *XPath, SQL, XQuery*, …
  - Transformation language: *XSLT, SQL, ATL*, …
  - Modeling language: *UML, SDL, BPMN*, …

More subtypes of **Language**

- Hypertext language: HTML, …
- Markup language: XML, …
- Configuration language
- Annotation language
- Template language
- …

Base entity type **Technology**

- Definition:
  - *A tool (in a very general sense) used in software development*
- Subtypes of *Technology*
  - API and library: *JDOM, JQuery, Swing*, Tkinter, Twitter API, …
  - Framework: *JPA, Hibernate, Spring, Django*, …
  - IDE: *Visual Studio, Eclipse, NetBeans*, …
  - Platform: *.NET, Android, J2EE, Java (platform), JRE*, …
  - Language processor: *javac, python, gcc*, …

More subtypes of **Technology**

- Server, e.g., Web server
- Web browser
- Plugin
- Office software
- Operating system
- Package portal
- Package manager
- App store
- …

Base entity type **Artifact**

- Definition:
  - *A „manifested" / „physical" entity in a software system*
- Subtypes of *Artifact* — they all concern „**representation**"*!*
  - **File**: files in the common sense of an operating system
  - **Folder**: folders as nested collections of files and folders
  - **Resource**: artifacts addressable / retrievable by URI/URL
  - **Transient**: artifacts arising „temporarily" by the execution of software
  - **Fragment**: artifacts being part of an artifact

Base entity type **System**

- Definition:
  - *A deployed, implemented, or designed software system*
- Subtype of *System*
  - Information system
  - Web application
  - Web service
  - Mobile app

Entity type **Fragment**

- Definition:
  - *A part of a software artifact*
- Examples
  - Methods in a class of a Python script
  - A method call in a Python script
  - A CREATE TABLE statement in a SQL/DDL script

Source code as a nested container

```
def search(l, x):
    return searchInRange(l, x, 0, len(l)-1)

def searchInRange(l, x, min, max):
    if min>max:
        return False
    else:
        middle = min+(max-min)/2
        if x > l[middle]:
            # Search in right half
            return searchInRange(l, x, middle+1, max)
        elif x < l[middle]:
            # Search in left half
            return searchInRange(l, x, min, middle-1)
        else:
            # Found in the middle
            return True
```


„Conceptual“ subtypes of **Artifact**

- **Specification** — e.g., the Java Language Specification
- **Standard** — e.g., software and systems engineering standards
  https://standards.ieee.org/findstds/standard/software_and_systems_engineering.html
- **Request** — e.g., an HTTP REST request
- **Response** — e.g., a response to the said request
- **Model** — in the sense of metamodeling and MDE
- **Metamodel** — in the sense of metamodeling and MDE
- **...**

It may happen that an artifact-like entity has two types — a representational one and a conceptual one, e.g.:

aRequest : Transient, Request

More generally, an entity may be of multiple types, but the base entity types are disjoint.

Base entity type **Function**

- Definition:
  - *A function on **languages** for domain and range defined by an artifact or implemented by a technology*
- Examples
  - The I/O behavior of a program
  - The I/O behavior of a tool as part of a technology

Base entity type **Concept**

- Definition:
  - A concept *from the broad domain of software development*
- Subtypes of *Concept*
  - Programming technique: iteration, recursion, etc.
  - Modeling principle: inheritance, composition, etc.
  - Design pattern: Composite, Visitor, etc.
  - Feature: persistence, etc.
  - Protocol: HTTP, etc.

Relationship symbols

We collect the symbols from the examples.

- ∈ — membership relationship for languages
- defines — something defining a language or a function
- implements — something implementing a language or a function
- ↦ — function application (data flow)
- ⊆ — subset relationship on languages
- partOf — part-of relationship (composition)
- uses — usage of languages, technologies, and concepts
- facilitates — facilitation (support for) usage
- refersTo — encoded references to entities
- conformsTo — conformance, e.g., in the sense of schema-based validation
- correspondsTo — correspondence, i.e., “systematic” similarity

Relationship types for *composition*

- Artifact partOf Artifact — an artifact being part of another artifact
- Artifact partOf System — an artifact being part of a system
- Technology partOf Technology — a technology being part of another technology
- Language partOf Technology — a language being part of a technology

Relationship types for *languages*

- Artifact ∈ Language — the language of an artifact
- Artifact defines (Language | Function) — languages or functions defined by artifacts
- (Artifact | Technology) implements (Language | Function) — … implemented by technologies
- Function(Artifact) ↦ Artifact — map an artifact to another artifact (“data flow”)
- Language ⊆ Language — subset relationship on languages

More relationship types

- Artifact conformsTo Artifact — conformance, e.g., in the sense of schema-based validation
- Artifact correspondsTo Artifact — correspondence, i.e., “systematic” similarity
- (Artifact | System) refersTo Entity — references to an entity encoded in an artifact
- (Technology | Artifact | System) (uses | facilitates) (Language | Technology | System | Concept)
- (Technology | Artifact | System) implements Concept

http://www.softlang.org/course:techonto

Gliederung

1. Willkommen **09:00-09:10**
2. Befragung der Teilnehmer **09:10-09:20**
3. Überblick über 101, SoLaSoTe und MegaL **09:20-10:15**
4. Klassifizierung von Sprachen, Technologien, u.ä. **10:30-11:15**
5. Beispielhafte Technologiemodelle **11:15-12:15**
6. **Eine Kern-Ontologie zur Technologiemodellierung** **13:00-13:45**
7. Weitere beispielhafte Technologiemodelle **14:00-14:45**
8. Prozess- und Werkzeugunterstützung **14:45-15:30**

Ontologien ...

- ... sind eine formale Spezifikation einer Konzeptionalisierung.
- ... entsprechen Wissensbasen.
- ... entsprechen einer Ansammlung von Fakten zu Entitäten und deren Relationen.
- Man unterscheidet zwischen Fakten zum Schema (T-Box) und Fakten zu Instanzen (A-Box).

Beispiel

https://en.wikipedia.org/wiki/Leaning_Tower_of_Pisa#/media/File:Leaning_tower_of_pisa_2.jpg

Welche Aussagen kann man zu den abgebildeten Gebäude treffen?

https://en.wikipedia.org/wiki/Leaning_Tower_of_Pisa#/media/File:Leaning_tower_of_pisa_2.jpg

- Es ist ein Monument
- Der Name des Monuments ist der `Schiefe Turm von Pisa'
- Der Turm befindet sich in Pisa (Pisa liegt in Italien)
- Wikipedia Artikel beschreibt den Turm.

Beispiel

T-Box

- Jedes Monument ist ein Gebäude.
- Ein Monument hat einen Namen.
- Ein Monument befindet sich an einem Ort.
- Eine Web-Resource beschreibt ein Monument.

A-Box

- T ist ein Monument.
- Der Name des Monuments ist der `Schiefe Turm von Pisa'.
- Der Turm befindet sich in Pisa.
- Wikipedia Artikel beschreibt den Turm.

Arten von Ontologien

- Grundlegende (Foundational) Ontologien erfassen Wissen zu den allgemeinsten Aspekten.
  - Was ist die formale Bedeutung einer Teil-Ganzes Beziehung?
- Kern-Ontologien erfassen Aspekte aus einer Menge von Domänen.
  - Softwaretechnologien sind Entitäten, die über mehrere Domänen hinweg relevant sind.

Arten von Ontologien

- Domänen-Ontologien erfassen spezielles Wissen aus einer Domäne.
  - Es existiert beispielsweise ein spezielles Vokabular im Kontext von Versicherungen.
- Taskorientierte Ontologien beschäftigen sich mit speziellem aufgabenbezogenem Wissen.
  - Beispielsweise lässt sich Wissen zu einem Ausleihprozess sammeln, welches unabhängig von einer Domäne ist.

Allgemeine Problemstellungen

- Was ist der beste Weg, um etwas zu klassifizierten oder zu beschreiben? Subjektivität ist an dieser Stelle ein Problem.
- Es existieren eventuell implizite Informationen in der Namensgebung.
  - Der "schiefe Turm von Pisa".
- Welche Informationen sind essentiell für ein bestimmtes Problem?

Philosophische Problemstellungen

- Realismus : Wie es eigentlich klassifiziert werden sollte, unterscheidet sich eventuell von dem, was wir darüber glauben zu wissen.
- Übereinkunft : Wie lässt sich ein Konsens zwischen den Meinungen mehrerer Experten erreichen?
- Variation : Wie lässt sich das vorhandene Wissen verfeinern?

Methoden zur Bewertung einer Ontologie

- Bewertung durch einen Domänenexperten
- Gold-Standard-basierte Evaluation
  - Entspricht dem Vergleich zu einer Ontologie, die als perfekt angenommen wird.
- Kriterienbasierte Evaluation
  - Bewertung der Ontologie anhand von aufgestellten Kriterien, wie zum Beispiel Metriken oder speziell formulierte Antipattern.

Methoden zur Bewertung einer Ontologie

- Aufgabenbasierte Evaluation
  - Hier ist die Ontologie ein verwendetes Artefakt in einem Programm. Es wird evaluiert wie gut sich die Ontologie für eine Aufgabe einsetzen lässt.
- Datengetriebene Evaluation
  - Meist werden zwei Ontologien mit einem Textcorpus zu einer Domäne verglichen, um zu sehen welche das Vokabular der Domäne besser abdeckt.

Der Technologieraum Onto-Ware

- Welche Möglichkeiten gibt es die T-Box in eine maschinenlesbare Form zu bringen?
- Welche Möglichkeiten gibt es die A-Box in eine maschinenlesbare Form zu bringen?

W3C – Standards : RDF

- Resource Description Format
- Resourcen werden mit Links verbunden.
- Sowohl für Resourcen als auch Links werden URIs verwendet.
- Fakten werden als Tripel erfasst.
- Der Verlinkungsstruktur entspricht einem gerichtetem, gelabelten Graphen.
- Beispielhafte Speicherformate sind RDF/XML oder N-Triple.

W3C Standards - RDF

- <http://example.org/T>
- <http://example.org/locatedIn>
- <http://example.org/Pisa>

W3C Standards - SPARQL

- A query language for RDF.
- A language to formulate queries and updates for a triplestore.

```
PREFIX : <http://myWikiTax.de/>
SELECT ?typename
WHERE {
        ?type :name ?typename .
    ?type :hasSubclassifier+ ?type .
}
```

W3C Standards - RDFS

- Resource Description Framework Schema
- Bietet Vokabular zur Formulierung eines Schemas für RDF Daten.
- Geschrieben in RDF

W3C Standards - RDFS

P rdfs:range C

- P ist Instanz von rdf:Property
- C ist Instanz von rdf:Class
- Resourcen welche als Objekte in Tripeln mit dem Prädikat P vorkommen, sind Instanzen der Klasse C.

W3C Standards - OWL

- W3C Web Ontology Language
- OWL ist eine Semantic Web Sprache
- Die Sprache dient der Formulierung von komplexem, reichhaltigem Wissen zu Entitäten, Gruppen und Relationen zwischen Entitäten.
- OWL ist eine logikbasierte Sprache. Entsprechend ist es für Reasoning besonders geeignet, um zum Beispiel die Konsistenz einer Ontologie zu verifizieren.

W3C Standards - OWL

```
<owl:Class rdf:about="
http://se-on.org/ontologies/domain-spanning/2012/02/code
-flaws.owl#DataClass">
    <rdfs:subClassOf rdf:resource="
    http://se-on.org/ontologies/domain-spanning/2012/02/
    code-flaws.owl#DesignDisharmony"/>
    <rdfs:comment>A data holder class without complex
    functionality that other class rely on.
    </rdfs:comment>
</owl:Class>
```

W3C Standards - OWL

```
<owl:ObjectProperty rdf:about="
http://se-on.org/ontologies/domain-spanning/2012/02/code
-flaws.owl#affectsCodeEntity">
    <rdfs:comment>A design disharmony affects a piece
    of code, such as a class or method.</rdfs:comment>
    <rdfs:domain rdf:resource="
    http://se-on.org/ontologies/domain-spanning/2012/02/
    code-flaws.owl#DesignDisharmony"/>
    <rdfs:range rdf:resource="
    http://se-on.org/ontologies/domain-specific/2012/02/
    code.owl#CodeEntity"/>
</owl:ObjectProperty>
```


Onto-Ware

- Jena und Eclipse RDF4J (früher Sesame) vereinfachen als Java-Technologien die Erstellung und Manipulation von RDF-basierten Datenbanken (Triplestores) in verschiedenen Formaten.

Eine Kern-Ontologie zur Technologiemodellierung

Problemstellung

- Die natürliche Sprache ist meist zu unpräzise, um Entitäten und deren Relationen exakt zu definieren.
- Um eine hohe Präzision zu erreichen, wird eine logikbasierte Sprache verwendet, wie zum Beispiel OWL oder im Folgenden Prädikatenlogik.

Axiomatisierung - Artefakte

- Ein Software-Artefakt ist eine digitale Entität.

$$Artifact(e) \Rightarrow Entity(e).$$
$$Fragment(a) \Rightarrow Artifact(a) \wedge \neg(File(a) \vee Folder(a)).$$
$$File(a) \Rightarrow Artifact(a).$$
$$Folder(a) \Rightarrow Artifact(a).$$
$$WebResource(a) \Rightarrow Artifact(a).$$
$$Transient(a) \Rightarrow Artifact(a).$$

Artefakte : Beispiele

- File(aJavaFile.java).
- File(aJavaByteFile.class).
- File(aJar.jar) and Folder(aJar.jar).
- WebResource(Java8Specification).
- Transient(aJavaObject).

Technologien und Systeme

- Technologien bieten Funktionen, die in verschiedenen Anwendungsszenarien verwendet werden können.
- Ein System entspricht einer Menge von Software-Artefakten, die ein bestimmtes Verwendungsszenario zu einer Technologie bieten.

$$System(e) \Rightarrow Entity(e).$$
$$Technology(e) \Rightarrow Entity(e).$$

Technologien und Systeme

- Technology(JavaPlatform).
- Technology(JavaC), partOf(JavaC,JavaPlatform).
- Technology(JavaDOMAPI).
- System(aJavaProject).
- Outlook : uses(aJavaProject,JavaDOMAPI).

Sprachen und Funktionen

- Function(compile) .
- Function(save).
- Language(Java).
- Language(JavaByteCode).
- Language(JavaObject).

$$Function(e) \Rightarrow Entity(e).$$
$$Language(e) \Rightarrow Entity(e).$$

Implementieren vs. Definieren

- defines(Java8Spec,Java).
  defines(W3CXMLSpec,XML).
- implements(JavaC,compile).
  implements(JavaC,Java).

$$defines(a,e) \Rightarrow Artifact(a) \wedge Entity(e).$$
$$implements(x,y) \Rightarrow (Artifact(x) \vee System(x) \vee Technology(x)) \wedge Function(y) \vee Technology(x) \wedge Language(y).$$
$$Language(l) \Rightarrow (\exists s.Specification(s) \wedge defines(s,l)) \vee (\exists t.Technology(t) \wedge implements(t,l)).$$

Sprachinstanziierung – Teil 1

$$elementOf(x,y) \Rightarrow Artifact(x) \wedge Language(y) \vee Pair(x) \wedge Function(y).$$
$$elementOf(a,l) \Leftarrow \exists s.defines(s,l) \wedge conformsTo(a,s).$$

Konformität zwischen Instanz und Schema

```
<?xml version="1.0"?>
<message>
  <to>Ralf</to>
  <from>Marcel</from>
</message>
```

```
<xs:element name="message">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Konformität zwischen Instanz und Schema

$$conformsTo(a,d) \Rightarrow Artifact(a) \wedge Artifact(d)$$

$$conformsTo(a,a') \Leftarrow (\forall p.partOf(p,a) \wedge \exists p'.partOf(p',a') \wedge conformsTo(p,p')) \vee \exists t.defines(a',t) \wedge elementOf(a,t)$$

Konformität zwischen Instanz und Schema

```
<?xml version="1.0"?>
<message>
  <to>Ralf</to>
  <from>Marcel</from>
</message>
```

```
<xs:element name="message">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```


Ein Artefakt entspricht einem anderen.

$$correspondsTo(x,y) \Rightarrow Artifact(x) \wedge Artifact(y)$$

$$correspondsTo(x,y) \Rightarrow (\forall px.partOf(px,x) \Rightarrow \exists py.partOf(py,y) \wedge correspondsTo(px,py)) \wedge (\forall py.partOf(py,y) \wedge \exists px.partOf(px,x) \wedge correspondsTo(py,px)) \vee (\nexists p.partOf(p,x) \vee partOf(p,y)) \wedge sameAs(x,y)$$

Ein Schema entspricht einem anderen.

```
<!ELEMENT message (to, from)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
```

```
<xs:element name="message">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Funktionen

$$Function(f) \Rightarrow \exists d, r.rangeOf(r,f) \wedge domainOf(d,f)$$

$$domainOf(d,f) \Rightarrow Language(d) \wedge Function(f)$$
$$rangeOf(r,f) \Rightarrow Language(r) \wedge Function(f)$$

Anwendung von Funktionen

$$inputOf(i,p) \Rightarrow Artifact(i) \wedge Pair(p)$$
$$inputOf(i,p) \Leftarrow \exists d, f.Language(d) \wedge Function(f) \wedge domainOf(d,f) \wedge elementOf(p,f) \wedge elementOf(i,d)$$

$$outputOf(o,p) \Rightarrow Artifact(o) \wedge Pair(p)$$
$$outputOf(o,p) \Leftarrow \exists r, f.Language(r) \wedge Function(f) \wedge rangeOf(r,f) \wedge elementOf(p,f) \wedge elementOf(o,r)$$

Sprachinstanziierung – Teil 2

- Artifact(aJavaClass).
  Technology(JavaC).
  Function(compile).
  implements(JavaC,compile).
  domainOf(Java,compile).
  rangeOf(JavaByteCode,compile).
  Pair(io). inputOf(aJavaClass, io).
  elementOf(io,compile).

$$elementOf(a,l) \Leftarrow \exists t.Technology(t) \wedge implements(t,l) \wedge \exists f.Function(f) \wedge implements(t,f) \wedge \exists p.Pair(p) \wedge elementOf(p,f) \wedge inputOf(a,p)$$

Transient

$$Transient(t) \Rightarrow \exists f, p, i.Function(f) \wedge Pair(p) \wedge Artifact(i) \wedge elementOf(p,f) \wedge inputOf(i,p) \wedge outputOf(t,p)$$

Beispiel Transient

- aJavaObject exists for a limited time in the context of a scenario.

```
Technology(JAXB).
Function(save).
implements(JAXB,save).
Pair(saveIO).
elementOf(saveIO,save).
Function(load).
implements(JAXB,load).
Pair(loadIO).
elementOf(loadIO,load).

File(anXMLModel1).
Transient(aJavaObject).
File(anXMLModel2).
domainOf(XML,loadIO).
rangeOf(JVMObject,loadIO).
domainOf(JVMObject,saveIO).
rangeOf(XML,saveIO).
inputOf(anXMLModel1,loadIO).
outputOf(aJavaObject,loadIO).
inputOf(aJavaObject,saveIO).
outputOf(anXMLModel2,saveIO).
```


Verwendung

- Wenn x zusammengesetzt ist (also ein System, oder eine Technologie oder ein Artefakt) und es einen Teil gibt, der die Sprache y instanziiert, dann verwendet x die Sprache y.

$$uses(x,y) \Leftarrow \exists p.partOf(p,x) \wedge elementOf(p,y)$$
$$uses(x,y) \Leftarrow \exists s,p.defines(s,y) \wedge partOf(p,x) \wedge conformsTo(p,s)$$

Vereinfachung

- Eine Technologie kann die Verwendung eines Konzepts vereinfachen. Beispielsweise vereinfacht Django die Verwendung des Model-View-Controller Design Pattern.

$$facilitates(x,y) \Rightarrow Technology(x) \wedge Concept(y)$$
$$facilitates(x,y) \Rightarrow \forall s.System(s) \wedge (uses(s,x) \Rightarrow uses(s,y))$$

Verwendung

- Beispiel für 2. : Die UML Spezifikation definiert die Sprache Java. In der Dokumentation meiner Software x existiert ein Teil, der konform ist zu dieser Spezifikation, also ein UML-Diagramm. Dann verwendet das Dokumentationsartefakt die Sprache UML.

$$uses(x,y) \Leftarrow \exists p.partOf(p,x) \wedge elementOf(p,y)$$
$$uses(x,y) \Leftarrow \exists s,p.defines(s,y) \wedge partOf(p,x) \wedge conformsTo(p,s)$$

Vereinfachung

- Eine Technologie kann die Verwendung eines Konzepts vereinfachen. Beispielsweise vereinfacht Django die Verwendung des Model-View-Controller Design Pattern.

$$facilitates(x,y) \Rightarrow Technology(x) \wedge Concept(y)$$
$$facilitates(x,y) \Rightarrow \forall s.System(s) \wedge (uses(s,x) \Rightarrow uses(s,y))$$

http://www.softlang.org/course:techonto

Gliederung

1. Willkommen **09:00-09:10**
2. Befragung der Teilnehmer **09:10-09:20**
3. Überblick über 101, SoLaSoTe und MegaL **09:20-10:15**
4. Klassifizierung von Sprachen, Technologien, u.ä. **10:30-11:15**
5. Beispielhafte Technologiemodelle **11:15-12:15**
6. Eine Kern-Ontologie zur Technologiemodellierung **13:00-13:45**
7. **Weitere beispielhafte Technologiemodelle** **14:00-14:45**
8. Prozess- und Werkzeugunterstützung **14:45-15:30**

Die Arbeitsgruppe Softwaresprachen entwickelt eine dementsprechende Modellierungssprache ***MegaL***, eine Bibliothek ***MegaLib*** von MegaL-Modellen und ein passendes Validierungstool ***MegaL Checker***.

https://github.com/softlang/megalib/tree/master/models

Wenn es die Zeit zulässt,
sollen an dieser Stelle einige Megamodelle betrachtet werden.

http://www.softlang.org/course:techonto

Gliederung

1. Willkommen **09:00-09:10**
2. Befragung der Teilnehmer **09:10-09:20**
3. Überblick über 101, SoLaSoTe und MegaL **09:20-10:15**
4. Klassifizierung von Sprachen, Technologien, u.ä. **10:30-11:15**
5. Beispielhafte Technologiemodelle **11:15-12:15**
6. Eine Kern-Ontologie zur Technologiemodellierung **13:00-13:45**
7. Weitere beispielhafte Technologiemodelle **14:00-14:45**
8. **Prozess- und Werkzeugunterstützung** **14:45-15:30**

Aspekte der Prozess- und Werkzeugunterstützung

- IDE-, Editier-Unterstützung
- Konsistenzprüfung für Technologiemodelle
- Konsistenzprüfung der Anwendung der Modelle auf Systeme
- Wie gewinnt man ein Modell aus regulärer Dokumentation?
- Wann ist ein Modell ein gutes Modell?
- …

Aufdeckung von Entitäten und Beziehungen

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

Ein MDE-Beispiel

- XML — zur Serialisierung
- EMF — Eclipse Modeling Framework
- ATL — Model transformation language
- Xtext —- Grammar-based language workbench
- API — Generated meta-model-specific Java API

XML

XML is a data exchange format and XSD is a format to describe a schema for XML documents. XSD schemas are written in XML.

```
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
```

EMF

Ecore is another format to define a schema. Here, schemas are called meta-models and documents are called models. A model may be persisted as an XMI file. XMI is an XML language. Meta-models are models, too. The meta-model of meta-models is defined by the meta-meta-model. The meta-model of the meta-meta-model is the meta-meta-model itself.

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


ATL

This is a language for transformations; input and output are supposed to conform to some meta-models. The transformation is applied on a model that conforms to the input meta-model resulting in a model that conforms to the output meta-model.

```
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
```


Xtext

This technology supports the development of a language workbench based on a grammar definition. Xtext integrates with EMF modeling and thereby needs an EMF model. …

```
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
```


API

Interaction with models is supported by a generated meta-model-specific Java API on transient object graphs. Modifications are exchanged between the XMI persistence layer and the JVM representation of models by de-/serialization.

```
Java : Language // Java is a Language
EcoreJava : Language // A Java subset for Ecore APIs
EcoreJava subsetOf Java // An Ecore API is valid Java
EMFGenModel : Language // Language for the generator model
genModel : Artifact // Parameters of the generation
genModel elementOf EMFGenModel
genModel references metaModel // Referencing
EMFGenerator : EMFGenModel → EcoreJava
EMFGenerator(genModel) ↦ api // Application of generator
CustomObjects : Language // Object graphs for Custom
Serialization : CustomObjects → Custom
Deserialization : Custom → CustomObjects
XMI : Language // Format for default persistence for EMF
XMI subsetOf XML // XMI is a subset of XML
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 meta–model
objectGraph : Transient // A runtime artifact
objectGraph elementOf CustomObjects
objectGraph conformsTo javaFiles // Conformance to Java classes
CustomSerialize(objectGraph) ↦ model
CustomDeserialize(model) ↦ objectGraph
```

```
Java : Language // Java is a Language
EcoreJava : Language // A Java subset for Ecore APIs
EcoreJava subsetOf Java // An Ecore API is valid Java
EMFGenModel : Language // Language for the generator model
genModel : Artifact // Parameters of the generation
genModel elementOf EMFGenModel
genModel references metaModel // Referencing
EMFGenerator : EMFGenModel → EcoreJava
EMFGenerator(genModel) ↦ api // Application of generator
CustomObjects : Language // Object graphs for Custom
Serialization : CustomObjects → Custom
Deserialization : Custom → CustomObjects
XMI : Language // Format for default persistence for EMF
XMI subsetOf XML // XMI is a subset of XML
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
```



```
Serialization : CustomObjects → Custom
Deserialization : Custom → CustomObjects
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 meta–model
objectGraph : Transient // A runtime artifact
objectGraph elementOf CustomObjects
objectGraph conformsTo javaFiles // Conformance to Java classes
CustomSerialize(objectGraph) ↦ model
CustomDeserialize(model) ↦ objectGraph
```


Aspekte der Integration von Modell, System und Modellierung

Artifact binding Links to artifacts in a system are part of the model; links can be resolved to the resources; different kinds of destinations are handled by plugins; the URI format may be used.

Semantic annotations Models are annotated in the sense of the Semantic Web; model elements are associated with objects from knowledge resources such as schema.org, DBpedia, or Wikipedia.

Pluggable analyses Relationships between entities are subject to pluggable analyses that give feedback to the user; the functionality may be readily available or require designated effort.

Modularized models Each module corresponds to a context of language and technology usage; reuse corresponds to the formation of larger (compound) contexts.

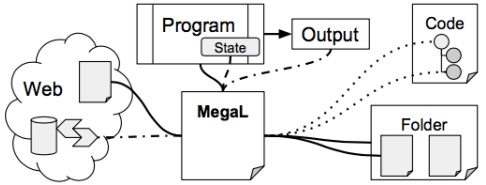
Transient artifacts These artifacts are accessed through some means of interception, as they only manifested temporarily as the result of specific actions performed by or with the system.

Model inference There exist implied model elements, e.g., in the sense of links or parthood; these model elements are inferred so that they do not need to be modeled explicitly.

Explorable connections All 'connections' are explorable by the user; for instance, one may navigate from model elements to system artifacts or relevant plugins for analysis.

Traceability links Relationships between artifacts that consist of parts may be represented as collections of traceability links, i.e., bipartite graphs. These links are inferred and amenable to navigation.

Artifact binding



Interconnection of megamodel and system; solid lines represent direct links; dashed lines indicate introspection; dash-dotted lines show interception; dotted lines depict fragment links.

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

The given URI is handled by three different resolution methods, the first being a GitHub repository resolver, followed by a JAR unpacker and concluded by an XML parser navigating to a certain element.

Semantic annotations

XML	http://dbpedia.org/page/XML
EMF	https://eclipse.org/modeling/emf/

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


Pluggable analyses

file elementOf XML: The analysis is reusable in that the load functionality of an XML API with well-formedness checking may be leveraged to accept or reject the file.

file elementOf EcoreJava: The decision as to whether a given file appears to be an element of the Java subset targeted by the EMF code generator requires a custom analysis where different options may be considered: either the file is searched for a 'watermark' of EMF or the file is analyzed more deeply in terms of structure of the classes and the use of base types.

xmlFile conformsTo xsdSchema: Any sort of analysis for schema- metamodel-like conformance should be reusable because conformance is usually an integral part of a technological (data) space. For instance, we may reuse any XSD schema validator to assert the validity of *file* to conform to *xsdSchema*.

javaFile conformsTo EcoreJavaSpec: We may use a domain-specific specification language for Java subsets so that we can capture the Java subset targeted by the EMF code generator or other Java subsets in a succinct manner. In this manner, we have established a new form of conformance, subject to a custom analysis, i.e., a custom interpreter.

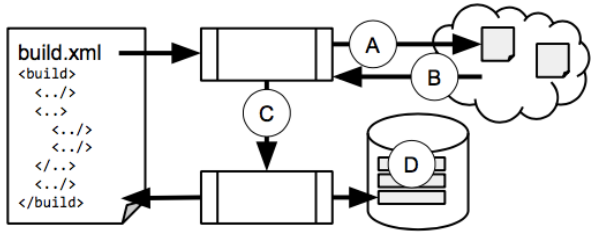
metaModel correspondsTo javaFiles: We could attempt a reusable analysis by means of ...

Modularized models

Entity \ Module	*xml*	*xmltrafo* **imports** *xml*
xmlFile	library.xml	library-in.xml library-out.xml
xsdFiles	libraries.xsd	libraries.xsd

Changes of bindings along reuse of modules.

Transient artifacts



A depiction of data flow and related transient states. A and B represent web request and response respectively, C depicts piping of program output, and D shows transient data in a database (or memory in general).

Model inference

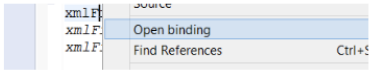
Decomposition into parts: The parthood-related structure of an artifact may be made systematically observable and amenable to part-level relationships; see also Sec. 3.8 on traceability.

Artifact bindings may follow a certain scheme such that they can be inferred. For instance, the names of languages and technologies, as used in the megamodels, may agree with names on knowledge resources such as DBpedia in a schematic manner, e.g., 'Java : Language' versus 'Java_(programming_language)' on DBpedia.

Subset relationships between languages: For instance, Java 7 is a subset of 8. These subset relationships should be interpreted transitively by inference. '*elementOf*' relationships should be inferred so that a statement of a file to be an element of Java 7 implies a statement of the same file to be an element of Java 8. The benefit of the inferred statements is that analyses for the different Java versions would be cross-validated.

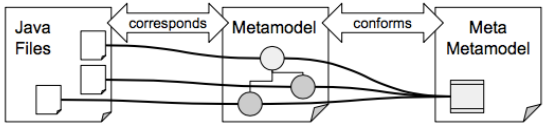
Explorable connections

Navigation from a model element (an entity of type 'Artifact') to the actual (represented) artifact and v.v.
Navigation from a compound artifact to fragments (parts) both on the side of the megamodel element and the actual artifact. (A view on the fragment may be based on an appropriate view of the compound artifact with the fragment being appropriately highlighted or unfolded.)
Navigation along trace links; see Sec. 3.8.
Navigation from a failed relationship or an error message or warning to the relevant analysis (plugin), involved artifacts, or relationship.
Navigation from a megamodel element (an entity or a relationship) to the relevant declaration or to other uses of the same name. (In this manner, one would also navigate, where necessary, to other modules.)



The open binding IDE integration in MegaL/Xtext+IDE, instead of going to the model element's declaration, the editor will navigate to the bound artifact.

Traceability links



Domain	Range
xsdFiles	javaFiles
/xs:schema/xs:complexType	L/org.softlang.megal.examples/src/org/softlang/company/xjc/Employee.java
/xs:schema/xs:element#0	L/org.softlang.megal.examples/src/org/softlang/company/xjc/Company.java
/xs:schema/xs:element#1	L/org.softlang.megal.examples/src/org/softlang/company/xjc/Department.java
xmlFile	objectGraph
/company/department#0	org.softlang.company.xjc.Department@5fd1a6aa
/company/department#0/employee#0	org.softlang.company.xjc.Employee@1a56a6c6
/company/department#0/employee#0/address: Utrecht	Utrecht
/company/department#0/employee#0/name: Erik	Erik
/company/department#0/employee#0/salary: 12345	12345.0
/company/department#0/employee#1	org.softlang.company.xjc.Employee@748e432b

Explorable trace links in MegaL/Xtext+IDE for an extended XML story with involvement of XML-data binding, i.e., Java-class generation from an XML schema. The trace at the top shows similarity of XSD schema versus Java classes. The trace below shows similarity of XML document versus Java object (past deserialization).

Eine weitere Fallstudie zur Technologiemodellierung (Entwicklung von Web-Anwendungen mit Django). Es wird ein weiterer Prozess der Modellerstellung vorgeführt.

We use the Polls app as the running example.

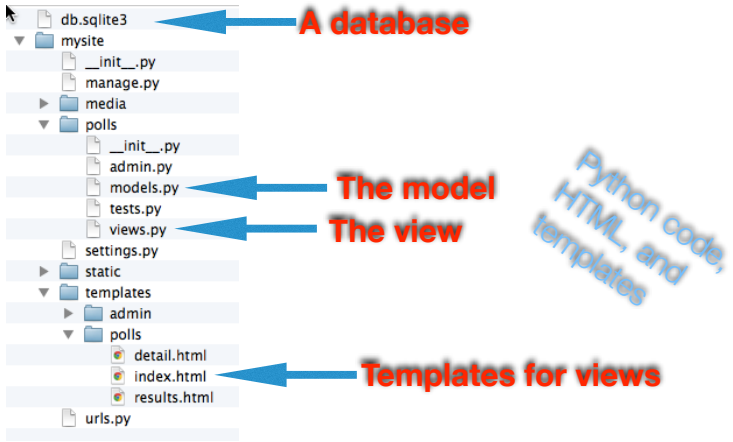
Find the code here:
https://github.com/rlaemmel/mysite

A deployment MAY be available here:
http://rlaemmel.pythonanywhere.com/polls/

Python Web frameworks

- **Django**
- web2py
- Flask
- Bottle

A low-level view on the Polls app



- db_sqlite3 : **File**
- mysite : **Folder**
  - __init__.py : **File**
  - manage.py : **File**
  - media : **Folder**
  - polls : **Folder**
    - __init__.py : **File**
    - admin.py : **File**
    - models.py : **File**
    - tests.py : **File**
    - views.py : **File**
  - settings.py : **File**
  - templates : **Folder**
    - admin : **Folder**
    - polls : **Folder**
      - detail.html : **File**
      - index.html : **File**
      - results.html : **File**
  - urls.py : **File**

We use a example-driven view on Django. We imagine that „any“ Django webapp would have artifacts like the „polls“ app.

Types of artifacts: How useful is that?

- db_sqlite3 : File ∈ **SL3IMG** (a language we made up)
- mysite : Folder
  - __init__.py : File ∈ **Python**
  - manage.py : File ∈ **Python**
  - media : **Folder**
  - polls : Folder
    - __init__.py : File ∈ **Python**
    - admin.py : File ∈ **Python**
    - models.py : File ∈ **Python**
    - tests.py : File ∈ **Python**
    - views.py : File ∈ **Python**
  - settings.py : File ∈ **Python**
  - templates : Folder
    - admin : **Folder**
    - polls : Folder
      - detail.html : File ∈ **HTML**
      - index.html : File ∈ **HTML**
      - results.html : File ∈ **HTML**
  - urls.py : File ∈ **Python**

Languages of artifacts: How useful is that?

Is this HTML, proper?

The beginning of a megamodel for Django

- Web application < System
- Web application framework < Technology
- Interpreter < Technology
- webapp : Web application
- Django : Web application framework
- Python : Language
- Python interpreter : Interpreter
- webapp uses Python
- webapp uses Django
- Django uses Python

We also assume that all „files“ of the app are entities of the technology model.

Issues

- What is the schema underlying the database image?
- What are the roles of the different python scripts?
- How do code and database relate to each other?
- What technologies are used by the app?
- Aren’t the HTML files using non-HTML constructs?

Issues

- **What is the schema underlying the database image?**
- What are the roles of the different python scripts?
- How do code and database relate to each other?
- What technologies are used by the app?
- Aren’t the HTML files using non-HTML constructs?

A command to request the DB schema

```
~ $ pwd
/home/rlaemmel/mysite
~ $ python manage.py sql polls
```

Illustration

Response by Django

```
BEGIN;
CREATE TABLE "polls_poll" (
    "id" integer NOT NULL PRIMARY KEY,
    "question" varchar(200) NOT NULL,
    "pub_date" datetime NOT NULL
)
;
CREATE TABLE "polls_choice" (
    "id" integer NOT NULL PRIMARY KEY,
    "poll_id" integer NOT NULL REFERENCES "polls_poll" ("id"),
    "choice" varchar(200) NOT NULL,
    "votes" integer NOT NULL
)
;
COMMIT;
```

Command line language for Django administration

- **python manage.py sql polls**
- **python manage.py syncdb**
- (There exist more such administrative commands.)

Request schema from database.

To be discussed in a second

Use Django's CLI for administration to retrieve the DB schema

- SL3IMG: Language
- db.sqlite3 : File
- db.sqlite3 ∈ SL3IMG
- Django.AdminCLI : Language
- getSchema : Transient
- getSchema ∈ Django.AdminCLI
- SQL : Language
- schema : Transient
- schema ∈ SQL
- retrieval : Django.AdminCLI × SL3IMG → SQL
- retrieval(getSchema, db.sqlite3) ↦ schema
- Python interpreter defines retrieval

We assume that there is command language as part of Django.

Command and schema are **transients** here, as we assume that they correspond to program input and output when exercising the CLI.

Issues

- What is the schema underlying the database image?
- **What are the roles of the different python scripts?**
- How do code and database relate to each other?
- What technologies are used by the app?
- Aren't the HTML files using non-HTML constructs?

Concepts behind the many Python scripts

- mysite
  - __init__.py **implements *Initialization***
  - manage.py **implements *Administration***
  - polls
    - __init__.py **implements *Initialization***
    - admin.py **implements *View***
    - models.py **implements *Model***
    - tests.py **implements *Testing***
    - views.py **implements *View***
  - settings.py **implements *Configuration***
  - urls.py **implements *Router (Routing)***

We leave out ***declarations*** of the concepts.

Concepts

MVC = Model View Controller

- ***Model***: the data / business logics part MVC
- ***View***: the user interface part of MVC
- ***Router***: a form of controller (part of MVC)
- ***Configuration***: configuration of a component or a system
- ***Initialization***: initialization of a component or a system
- ***Administration***: administration of a system
- ***Testing***: test of an artifact or a system

The *model*

```
from django.db import models
import datetime

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    def __unicode__(self):
        return self.question
    def was_published_today(self):
        return self.pub_date.date() == datetime.date.today()
    was_published_today.short_description = 'Published today?'

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice = models.CharField(max_length=200)
    votes = models.IntegerField()
    def __unicode__(self):
        return self.choice
```

Don't bother about details: these are Python (Django) classes for the business data of the Polls app.

Illustration

The view for „end users“

```
def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    t = loader.get_template('polls/index.html')
    c = Context({
        'latest_poll_list': latest_poll_list,
    })
    return HttpResponse(t.render(c))

def detail(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    return render_to_response('polls/detail.html', {'poll': p},
                              context_instance=RequestContext(request))

def results(request, poll_id):
    p = get_object_or_404(Poll, pk=poll_id)
    return render_to_response('polls/results.html', {'poll': p})
```

Don't bother about details, but a typical view loads or saves data, and renders data as HTML via a template.

Illustration

The view for „admins“ according to Django

```
class ChoiceInline(admin.TabularInline):
    # Another more spacious option
    # class ChoiceInline(admin.StackedInline):
    model = Choice
    extra = 3

class PollAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['question']}),
        ('Date information', {'fields': ['pub_date'], 'classes': ['collapse']}),
    ]
    inlines = [ChoiceInline]
    list_display = ('question', 'pub_date', 'was_published_today')
    list_filter = ['pub_date']
    search_fields = ['question']
    date_hierarchy = 'pub_date'

admin.site.register(Poll, PollAdmin)
```

These views are standardized by Django: they allow us to do basic data management for polls and choices.

Illustration

Routing
(A router maps URLs to views)

```
from django.conf.urls.defaults import patterns, include, url
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    url(r'^polls/$', 'mysite.polls.views.index'),
    url(r'^polls/(?P<poll_id>\d+)/$', 'mysite.polls.views.detail'),
    url(r'^polls/(?P<poll_id>\d+)/results/$', 'mysite.polls.views.results'),
    url(r'^polls/(?P<poll_id>\d+)/vote/$', 'mysite.polls.views.vote'),
    url(r'^admin/', include(admin.site.urls)),
)
```

Regular expression for URLs with parameters

Python function for a particular view

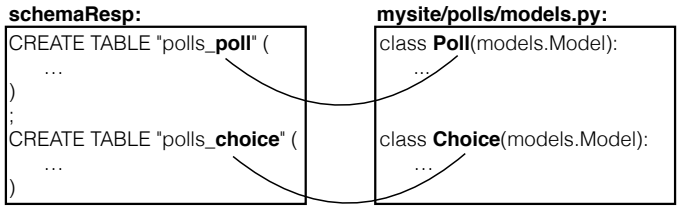
Illustration

Issues

- What is the schema underlying the database image?
- What are the roles of the different python scripts?
- **How do code and database relate to each other?**
- What technologies are used by the app?
- Aren't the HTML files using non-HTML constructs?

How do code (model) and database relate to each other?

schema correspondsTo mysite/polls/models.py



Request of database sync via CLI

```
~ $ pwd
/home/rlaemmel/mysite
~ $ python manage.py syncdb
```

Illustration

Response by Django

```
Creating tables ...
Creating table auth_permission
…
Creating table django_admin_log
Creating table polls_poll
Creating table polls_choice

You just installed Django's auth system, which means you don't have any superusers defined.
Would you like to create one now? (yes/no): yes
Username (Leave blank to use 'rlaemmel'): rlaemmel
…
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
No fixtures found.
```

This is basically just informative text produced by the admin functionality to report on database changes.

Use Django's CLI for administration to sync model code with database

- $db.sqlite3_1$, $db.sqlite3_2$: File
- $db.sqlite3_1$, $db.sqlite3_2 \in$ SL3IMG
- syncdb : Transient
- syncdb ∈ Django.AdminCLI
- syncdb refersTo mysite/polls/models.py
- modification : Django.AdminCLI × SL3IMG → SL3IMG
- modification(syncdb, $db.sqlite3_1$) ↦ $db.sqlite3_2$
- Python interpreter defines modification

This is similar to the CLI use for requesting the database schema, but this time the database is modified.

Issues

- What is the schema underlying the database image?
- What are the roles of the different python scripts?
- How do code and database relate to each other?
- **What technologies are used by the app?**
- Aren't the HTML files using non-HTML constructs?

Referenced python modules

- Runtime < Technology
- Template processor < Technology
- Protocol < Concept
- PythonRuntime : Runtime
- os : Library
- datetime : Library
- Django.db : Library
- Django.test : Library
- Django.template : Template processor
- Django.http : Library
- Database access : Concept
- Testing : Concept
- Template processing : Concept

This naming convention introduces **parts**.

- HTTP : Protocol
- webapp uses os
- webapp uses datetime
- webapp uses Django.db
- webapp uses Django.test
- webapp uses Django.template
- webapp uses Django.http
- datetime partOf PythonRuntime
- os partOf PythonRuntime
- Django.db facilitates Database access
- Django.test facilitates Testing
- Django.http facilitates HTTP

Issues

- What is the schema underlying the database image?
- What are the roles of the different python scripts?
- How do code and database relate to each other?
- What technologies are used by the app?
- **Aren't the HTML files using non-HTML constructs?**

The template for the *index* view

```
{% if latest_poll_list %}
    <ul>
    {% for poll in latest_poll_list %}
        <li><a href="/polls/{{ poll.id }}/">{{ poll.question }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```

Illustration

The template for the *detail* view

```
<h1>{{ poll.question }}</h1>

{% if error_message %}<p><strong>{{ error_message }}</strong></p>{% endif %}

<form action="/polls/{{ poll.id }}/vote/" method="post">
{% csrf_token %}
{% for choice in poll.choice_set.all %}
    <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}" />
    <label for="choice{{ forloop.counter }}">{{ choice.choice }}</label><br />
{% endfor %}
<input type="submit" value="Vote" />
</form>
```

Illustration

The template for the *results* view

```
<h1>{{ poll.question }}</h1>

<ul>
{% for choice in poll.choice_set.all %}
    <li>{{ choice.choice }} -- {{ choice.votes }} vote{{ choice.votes|pluralize }}</li>
{% endfor %}
</ul>

<a href="/polls/{{ poll.id }}/">Vote again?</a>
```

Illustration

A language for *templates*

- We designate a language **Django.Templ** ⊃ **HTML**.
- Django.Templ offers extra constructs like this:
  - Python expressions {{ … }} evaluating to HMTL
  - Loops over Python data to return HTML
- MegaL declarations:
  - **Template language < Language**
  - **Django.Templ : Template language**
  - **HTML ⊆ Django.Templ**
  - **mysite/templates/polls/*.html** ∈ **Django.Templ**

We use this notation as a short cut to refer to many entities in an obvious manner.

Verstehen von Technologien und Sprachen

Danke

für Ihre Aufmerksamkeit und Teilnahme!

http://www.softlang.org/course:techonto

Prof. Dr. R. Lämmel und Marcel Heinz (MSc)
AG Softwaresprachen
Universität Koblenz-Landau
http://www.softlang.org/startseite

Danksagung an weitere Mitarbeiter und Studierende aus der Arbeitsgruppe: Andrei Varanovich, Lukas Härtel, Johannes Härtel.