Outline

- Motivating Software Documentation
- Classic Software Documentation
- Megamodel-based Documentation
- Introducing MegaL
  - JAXB Megamodel
  - ANTLR Megamodel
  - OpenGL Megamodel (Draft)
Why do I need to document?

Johnny, the programmer

© 2017 Software Languages Team
Problems

- Missing Expertise
- High costs for introducing new technologies
- Exhaustion
- Job Security?
- Vendor lock in (dependency on a software vendor)
- ...

© 2017 Software Languages Team
How should I document?

Johnny, the programmer
A viewpoint describes the types of entities and relationships that are used to model a system. Different viewpoints are necessary for distinct stakeholders. A viewpoint is described in a metamodel.
Classic Software Documentation - Structure Viewpoint

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

© 2017 Software Languages Team
Classic Software Documentation - Structure + Usage Viewpoint

Von Guaer aus der deutschsprachigen Wikipedia, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=4725029

© 2017 Software Languages Team
Classic Software Documentation
- Process Viewpoint


By Stkl [CC BY-SA 3.0](http://creativecommons.org/licenses/by-sa/3.0), via Wikimedia Commons

© 2017 Software Languages Team
Classic Software Documentation - Architecture Descriptions

Logical view → Development view

Process view ← System & environment → Physical view

https://commons.wikimedia.org/wiki/File%3A4%2B1_Architectural_View_Model.svg
By mpan [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

© 2017 Software Languages Team
Interface Stream\(<T>\>

Type Parameters:
\(T\) - the type of the stream elements

All Superinterfaces:
AutoCloseable, BaseStream\(<T,Stream\(<T>\)\>)

public interface Stream\(<T>\>
extends BaseStream\(<T,Stream\(<T>\)\>)

A sequence of elements supporting sequential and parallel aggregate operations. The following example illustrates an aggregate operation using Stream and IntStream:

```java
int sum = widgets.stream()
    .filter(w -> w.getColor() == RED)
    .mapToInt(w -> w.getWeight())
    .sum();
```

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html
Creating a project

If this is your first time using Django, you’ll have to take care of some initial setup. Namely, you’ll need to auto-generate some code that establishes a Django project – a collection of settings for an instance of Django, including database configuration, Django-specific options and application-specific settings.

From the command line, `cd` into a directory where you’d like to store your code, then run the following command:

```
$ django-admin startproject mysite
```

This will create a `mysite` directory in your current directory. If it didn’t work, see Problems running `django-admin`.

**Note**

You’ll need to avoid naming projects after built-in Python or Django components. In particular, this means you should avoid using names like `django` (which will conflict with Django itself) or `test` (which conflicts with a built-in Python package).
Classic Software Documentation
- Literate Programming

\section{Hello world}

Today I awoke and decided to write some code, so I started to write Hello World in \texttt{C}.

\begin{verbatim}
<<hello.c>>=
/*
 /**
  <<license>>
 */
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Hello World!\n");
    return 0;
}
@\noindent\ldots then I did the same in PHP.
\end{verbatim}

https://en.wikipedia.org/wiki/Noweb

© 2017 Software Languages Team
We have a problem!
Johnny, the programmer

How can I use <Technology> in a proper way?

How does <Technology> work?

What is <Technology> used for?
Softlang Team's Approach

SoLaSoTe ontology

Linguistic architecture  Social coding

Software chrestomathy 101

Code  Doc  Wiki

Megamodeling with MegaL

EMF  Java  Django

© 2017 Software Languages Team
Softlang Team's Approach

- Illustrative Software Applications implementing a feature model. (e.g., 101companies)
- **Software Technology Models (MegaL)**
- Software Language Software Technology Ontology (SoLaSoTe)
Prescriptive vs Descriptive

- Prescriptive models describe the properties holding for software that exists in the future.
  - Specification purpose
  - Target of code generation

- Descriptive models describe properties of existing software.
  - Documentation purpose
MegaL

- MegaL is short for 'Megamodeling Language', where a megamodel describes models and their relationships from a conceptual perspective.
A Technology Model

- XSD
  - subsetOf
  - elementOf: schema.xsd

- schema.xsd
  - conformsTo: doc.xml
  - correspondsTo: object-model.java

- doc.xml
  - correspondsTo: object-graph

- object-model.java
  - conformsTo: object-graph
  - elementOf: Java

- object-graph
  - elementOf: JVMObject Language
  - out: deserialization
  - in: XML

- XML
  - elementOf: XSD
  - elementOf: JVMObject Language

© 2017 Software Languages Team
MegaL

- Multiple implementations exist.
- **MegaL-Text is trimmed towards simplicity and well-formedness checking.**
- MegaL/Xtext aims at validating facts in an actual system.
MegaL-Text

- A knowledge representation language using textual syntax (a DSL)
- Facilitates modularization
- Facilitates literate programming
- Evolved Vocabulary compared to previous papers.
- Usable Well-formedness Checker to validate MegaL models.
MegaL-Text
- Prelude

• The Prelude module contains all types of entities and types of relationships.

• It is imported, when processing a new model.

• *(For the course, we use a subset of the kinds of entities and relationships)*
A Technology Model

XSD ⊆ XML

schema.xsd ⊆ schema.xsd

doc.xml ⊆ xml

object-model.java ⊆ Java

object-graph ⊆ object-model

JVMObject Language ⊆ object-model.java

deserialization in out

© 2017 Software Languages Team
Prelude - Language

• A language is a set of syntactic entities.
  – Language < Entity
• A language can be further classified by its purpose.
  – ProgrammingLanguage < Language
• Examples:
  – Java : ProgrammingLanguage
  – XML : DataExchangeLanguage
Prelude - Language Subsets

• One language can be a subset of another.
  – `subsetOf < Language # Language`.

• Example:
  – `XML : DataExchangeLanguage`
  – `XSD : SchemaLanguage`
  – `XSD subsetOf XML`
Prelude - Artifact

- An artifact is a digital entity.
  - Artifact < Entity
- An artifact is primarily classified by its manifestation.
  - File < Artifact
  - File+ < Artifact
  - Other subtypes are Transient, Transient+, Folder, Fragment.
- Examples:
  - objectModel : File+
  - doc : File
  - objectGraph : Transient
Prelude - Artifact

• Any artifact can be classified by its language.
  – `elementOf : Artifact # Language`
  – e.g., `objectmodel elementOf Java`

• Any artifact can be classified by a conceptual role that it plays in a system.
  – `Role < Concept`  //and we have `Concept < Entity`
  – `hasRole < Artifact # Role`
Prelude - Artifact

- Artifacts can correspond to each other.
  - correspondsTo: Artifact # Artifact

- One artifact can conform to another.
  - conformsTo: Artifact # Artifact
Prelude - Functions

• A function defines a mapping from domain language(s) to range language(s).
  – Function < Entity
  – deserialization : XML -> JVMObjectLanguage

• Functions may be applied to concrete artifacts that are element of the specified languages.
  – deserialization(doc) |-> object-graph

© 2017 Software Languages Team
Johnny, the programmer

Why do we speak of DSLs when using XML?
Prelude – Language Definition

- Languages cannot just be any set of syntactic entities.
- Many times, languages are defined by a grammar (see ANTLR) or a schema (e.g., XSD grammars).
  - defines < Artifact # Language
- Other times, the grammar exists only implicitly as a parser implements the language’s syntax.
  - implements < Technology # Language
XML Language Triangle

- XML
- CompanyXML
- company.xsd
- company.xml

relations:
- defines
- conformsTo
- subsetOf
- elementOf
Now, where are the technologies?
/* A technology is reusable 'third party' Software. See Wiki */

System < Entity;
    = "wikipedia::Software_system".

Technology < System;
    = "wikipedia::Information_technology";
    = "elsevierJournal::information-and-software-technology/".

Library < Technology;
    = "wikipedia::Library_(computing)".

API < Technology;
    = "wikipedia::Application_programming_interface".

Framework < Technology;
    = "wikipedia::Software_framework".

Plugin < Technology;
    = "wikipedia::Plug-in_(computing)".

Application < Technology;
    = "wikipedia::Application_software".

SoftwareSuite < Technology;
    = "wikipedia::Software_suite".

Service < Technology;
    = "wikipedia::Service_(systems_architecture)".

Platform < Technology;
    = "wikipedia::Computing_platform".
Prelude - System

/* Implementation relationships */
implments < System # Function;
    = "wikipedia::Implementation".
implments < System # Language.

/* A technology or a language can facilitate the usage or realization of a conceptual description. */
facilitates < Technology # Concept.

/* A software system may help in a problem context */
supports < System # ProgrammingDomain;
    = "wikipedia::Software_feature". //needs better link

/* A language is used in the sense that some part is a member of the language. */
uses < System # Language;
    = "wikipedia::Computer_programming".

/* Something uses a concept, if its implementation or structure is informally conform to the concept's definition. */
uses < System # Concept.

/* Systems, technologies and artifacts can be reused. There is always a reference from the using entity to the used entity. */
reuses < System # System;
    = "wikipedia::Code_reuse".
Prelude - Parthood

/* Part-of relationships */
partOf < Artifact # Artifact;
    = "oxfordDict::part".
partOf < Artifact # System.
partOf < System # System.
MegaL-Text - Linking

- Descriptive Link with URL or Filepath for further Reading:
  
  MegaL = "http://softlang.org/megal".
  MegaL = "file://../docs/languageDescription.pdf".
MegaL-Text - Binding

- Binding artifacts to where the modelled artifacts actually live:

```java
javaFile1 ~= "http://softlang.org/JavaFile1.java".
javaFile1 ~= "file:///..//org/softlang/JavaFile1.java".
```
MegaL-Text - Namespaces

/* Encyclopedia and Dictionary namespaces */
101wiki :: "https://101wiki.softlang.org".
wikipedia :: "https://en.wikipedia.org/wiki".
oxfordDict :: "https://en.oxforddictionaries.com/definition".

/* Tutorial pages */
vogella :: "http://www.vogella.com/tutorials/".

/* Organization namespaces */
w3school :: "http://www.w3schools.com".
w3c :: "https://www.w3.org".
oracleDocs :: "https://docs.oracle.com/".

/* Research namespaces */
researchgate :: "https://www.researchgate.net/publication".
elsevierJournal :: "https://www.journals.elsevier.com".
springerLink :: "http://link.springer.com/article".
ieeeXplore :: "http://ieeexplore.ieee.org".
acm :: "http://dl.acm.org".

© 2017 Software Languages Team
Johnny, the programmer

Explain to me how JAXB works!
JAXB

/*
 * @Description:
 * JAXB is used for mapping Java objects to XML.
 *
 * @Resources:
 * See headline of https://101wiki.softlang.org/Technology:JAXB
 */

JAXB : Framework;
    = "101wiki::Technology:JAXB";
    uses Java;
    facilitates Mapping;
    facilitates XMLDataBinding.
JAXB

/*
@Description:
   As a part of JAXB, XJC deals with schema compilation.

@Resources:
   See description of https://101wiki.softlang.org/Technology:JAXB
*/

XJC: Compiler;
   = "101wiki::Technology:xjc";
   = "JAXBRI::xjc/src/main/java/com/sun/tools/xjc"
partOf JAXB;
uses Java;
uses Annotation;
facilitates Compilation.
/*
@Description:
The compiler can produce Java code, but not any Java code. It specifically produces syntactic fragments that belong to a subset of the full Java language.

@Resources:
See the AST here:

*/

```
XJCJJava : ProgrammingLanguage;
    = "JAXBRI::codemodel/codemodel/src/main/java/com/sun/codemodel";
    subsetOf Java;
    ^implements XJC.
generateJava : XSD -> XJCJJava.
XJC implements generateJava.
```
/*
@Description:
    jaxbChoice is an implementation of the 101companies feature model.

@Rationales:
    See https://101wiki.softlang.org/Contribution:jaxbChoice
*/

JAXBCHOICE ::= "file://../code/jaxbChoice".
101JAXB : Application;
    = "101wiki::Contribution:jaxbChoice";
    reuses JAXB;
    uses Mapping;
    uses XMLDataBinding;
    reuses XJC;
    uses Compilation.
/ *
@Description
 local deployment of 101JAXB
@Rationales
 See line 13-19 in "jaxbChoice/build.gradle"
 */

companyXSD : File;
  ~= ".../code/jaxbChoice/inputs/Company.xsd";
 partOf 101JAXB;
 elementOf XSD;
 hasRole Schema.

companyModel : File+;
  ~= ".../code/jaxbChoice/src/main/java/org/softlang/company/xjc";
 partOf 101JAXB;
 elementOf XJCJava;
 hasRole POJO.

generateJava(companyXSD) |-> companyModel.
/*
@Description
Next, we pose object instances and serialization.

@Rationales
See description of https://101wiki.softlang.org/Technology:JAXB
*/

JAXB.Binding : Framework;
    = "JAXBRI::runtime/jaxb1/src/main/java/com/sun/xml/bind";
    uses Java;
    facilitates Serialization;
    facilitates Mapping;
    facilitates XMLDataBinding.
marshal : JVMObjectLanguage -> XML .
unmarshal : XML -> JVMObjectLanguage .
JAXB.Binding implements marshal;
    implements unmarshal.
*/
@Description
   More precise languages

@Rationale
   Any XSD schema creates a new subset of XML
   Same for Javaclasses and JVMObjectLanguage
*/
CompanyXML : DataExchangeLanguage;
   = "JAXBCHOICE::inputs/Company.xsd";
subsetOf XML;
^defines companyXSD.
CompanyObjectLanguage : ValueLanguage;
   = "JAXBCHOICE::src/main/java/org/softlang/company/xjc";
subsetOf JVMObjectLanguage;
^defines companyModel.
JAXB

/*
 * Description
 * marshal is applied to a company object
 *
 * Rationales
 * See jaxbChoice/src/main/java/org/softlang/company/features/Serialization.java
 */

companyObject : Transient;
~ = "JAXBCHOICE::src/test/java/org/softlang/company/tests/SerializationTest.java#L18";
~ = "JAXBCHOICE::src/main/java/org/softlang/company/features/Serialization.java#L38";

companyFile : File;
~ = "JAXBCHOICE::outputs/sampleCompany.tmp";

elemetOf CompanyObjectLanguage.

marshal(companyObject) |-> companyFile.
unmarshal(companyFile) |-> companyObject.
/*
@Description:
   Conformance and Correspondence Relationships

@Rationale:
   See the definition of mapping
*/

companyFile correspondsTo companyObject.
companyObject correspondsTo companyFile.

companyXSD correspondsTo companyModel.
companyModel correspondsTo companyXSD.

companyFile conformsTo companyXSD.
companyObject conformsTo companyModel.
Assignment 6

- http://www.softlang.org/course:ptt17-assignment6
Okay, how about explaining something from the DSL Lecture.
ANTLR
Johnny, the programmer

What does this have to do with ANTLR?
ANTLR

/*
@Description:
 ANTLR is a "parser-generator". Thus, it is a framework used to generate parsers.

@Rationale: See https://101wiki.softlang.org/Technology:ANTLR4 */
ANTLR : ParserGenerator;
   = "https://github.com/antlr/antlr4";
facilitates ParserGeneration;
uses Java.
ANTLR

/*
@Description:
ANTLR implements a grammar language
and is able to produce some Java subset.

@Rationale:
See https://github.com/antlr/antlr4
and below.
*/

G4 : GrammarLanguage;
"ANTLRGITHUB::doc/grammars.md";
"ANTLRGITHUB::doc/lexer-rules.md";
^implements ANTLR.
ANTLRJava : ProgrammingLanguage;
"ANTLRGITHUB::tool/resources/org/antlr/v4/tool/templates/codegen/Java/Java.stg";
subsetOf Java;
^implements ANTLR. //Simplification.
ANTLR

/*
@Description:
   Lexer and Parser are generated.

@Rationale:
   See  https://github.com/antlr/antlr4  */
generateLexer : G4 -> ANTLRJava .
generateParser : G4 -> ANTLRJava .
*/
@Description:
    Introducing the FSML implementation

@Rationale:
    See YASANTRL */
FSML : Language;
    = "101wiki::Language:FSML".
FSMLImpl : Application;
    = "YASANTRL::";
uses Java;
    implements FSML.
ANTLR

/*
@Description:
 Lexer and Parser are generated for FSML.

@Rationale:
See YASANTLR::Makefile. */

fsmlGrammar : File;
  ~= "YASANTLR::Fsml.g4";
  hasRole Grammar;
  defines FSML.

fsmlLexerFile : File;
  ~= "YASANTLR::org/softlang/fsml/FsmlLexer.java";
  hasRole Lexer.

fsmlParserFile : File;
  ~= "YASANTLR::org/softlang/fsml/FsmlParser.java";
  hasRole Parser.

generateLexer(fsmlGrammar) |-> fsmlLexerFiles.

generateParser(fsmlGrammar) |-> fsmlParserFiles.
/*
@Description:
  ANTLR parser is preceded by a lexical analysis

@Rationale:
  See YASANTLR::org/softlang/fsml/tests/FsmlParserTest.java
 */
TokenStreamLanguage : ValueLanguage;
  =
    "ANTLRGITHUB::runtime/Java/src/org/antlr/v4/runtime/CommonTokenStream.java";
  subsetOf JVMObject;
  ^implements ANTLR.
lex : FSML -> TokenStreamLanguage.
FSMLImpl implements lex;
  uses LexicalAnalysis.
/*
@Description:
  Applying lexical analysis

@Rationale:
  See YASANTLR::org/softlang/fsml/tests/FsmlParserTest.java */
turnstile : File;
  ~= "YAS::languages/FSML/sample.fsml"
  elementOf FSML;
  conformsTo fsmlGrammar.
tokenstream : Transient;
  ~= "YASANTLR::org/softlang/fsml/tests/FsmlParserTest.java#L25";
  elementOf TokenStreamLanguage.
lex(turnstile) |-> tokenstream.
/*
@Description:
   Parsing to CST

@Rationale:
   See YASANTR::org/softlang/fsml/tests/FsmlParserTest.java
   for boilerplate code */
ANTLR::ValueLanguage;
= "ANTLRGITHUB::runtime/Java/src/org/antlr/v4/runtime/tree/ParseTree.java";
   subsetOf JVMObject.
parseToCST: TokenStreamLanguage -> AntlrCSTLanguage.
FSMLImpl implements parseToCST;
   uses Parsing.
/*
@Description:
Applying parsing to CST

@Rationale:
See YASANTLR::org/softlang/fsml/tests/FsmlParserTest.java */

fsmlCST : Transient;
~~ "YASANTLR::org/softlang/fsml/tests/FsmlParserTest.java#L24";
  elementOf AntlrCSTLanguage;
  hasRole ConcreteSyntaxTree.
parseToCST(tokenstream) |-> fsmlCST.
/*
@Description:
    Abstract CST to AST

@Rationale:
    See YASANTLR:org/softlang/fsml/tests/FsmlParserTest.java */
FsmlASTLanguage : ValueLanguage;
    = "YASANTLR:org/softlang/fsml/Fsm.java";
    subsetOf JVMObjectLanguage;
^implements FSMLImpl.
parseToAST : AntlrCSTLanguage -> FsmlASTLanguage .
FSMLImpl implements parseToAST;
    uses Abstraction.
fsmlAST : Transient;
    =~ "YASANTLR:org/softlang/fsml/tests/FsmlParserTest.java#L24";
    elementOf FsmlASTLanguage;
    hasRole AbstractSyntaxTree;
parseToAST(fsmlCST) |-> fsmlAST.
Johnny, the programmer

What about modularization of models?
On Modularization

• For the sake of simplicity we did not use complex MegaL concepts.
Project Megalib
Summary

• There exist classic approaches towards documenting technology.
• Megamodelling raises the link to source code.
• MegaL facilitates Megamodelling :-)
• An essential research question is what information is necessary to understand a technology.

© 2017 Software Languages Team
Questions?
Naming Conventions

If you want to author a technology model with MegaL we advise you to use the following naming conventions.

- Module names start with a capital letter.
- You shouldn't use spaces or underscores in an entity's name and use camel-case instead.
- The symbol '.' must only be used when you want to emphasize a parthood relationship as follows: `<composite>.<part>`, e.g., 'EMF.CodeGenerator'.
- The name of instances of Language, System, Role and Concept should start with a capital letter unless there exists some original spelling.
Naming Conventions

• The name of instances of Function and Artifact starts with a lower case letter.

• An artifact's name starts by relating to a language, if this helps with distinguishing multiple artifacts, e.g., '?ecoreJavaModel' or '?ecoreXMIModel'.
Naming Conventions

• An artifact's name should not start with 'a' (as the indefinite article) or 'my'.

• Any function's name may relate to language, if it helps with distinguishing functions, such as 'parseXML'.

• If a language does not have any name you can relate to the implementing technology and the language's superset `<technology><superset>`, e.g., 'ANTLRJava'. Or you can relate to any existing superset `<name><superset>`, e.g., 'FSMLXML', or provide an original name.