Domain-Specific Languages

The Art Of Domain-Specific Languages
Let's Hack Our Own Languages!

Slides adapted from materials originally created by Prof. Benoit Combemale. Used with permission.

Plan

- Domain-Specific Languages (DSLs)
  - Languages and abstraction gap
  - Examples and rationale
  - DSLs vs General purpose languages, taxonomy
- External DSLs
  - Grammar and parsing
  - EMF, Xtext, Langium, *Sirius*

Plan

- Domain-Specific Languages (DSLs)
  - Languages and abstraction gap
  - Examples and rationale
  - DSLs vs General purpose languages, taxonomy
- External DSLs
  - Grammar and parsing
  - EMF, Xtext, Langium, *Sirius*

Contract

- Better understanding/source of inspiration of software languages and DSLs
  - Revisit of history and existing languages

- Foundations and practice of Xtext
  - State-of-the-art language workbench (mature and used in a variety of industries)

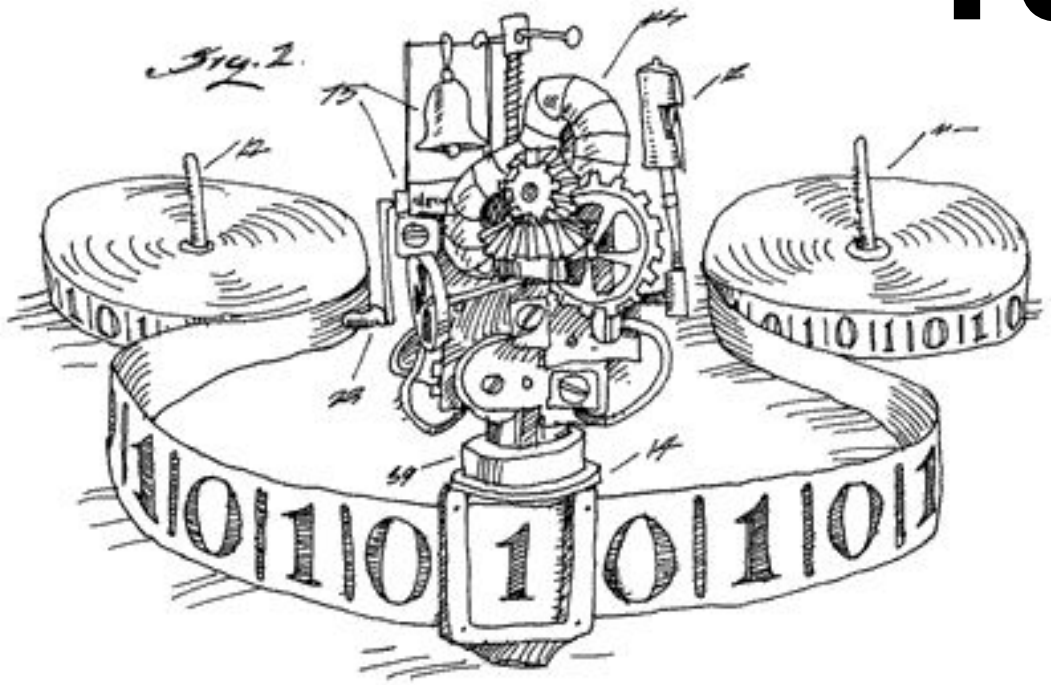
What are DSLs?

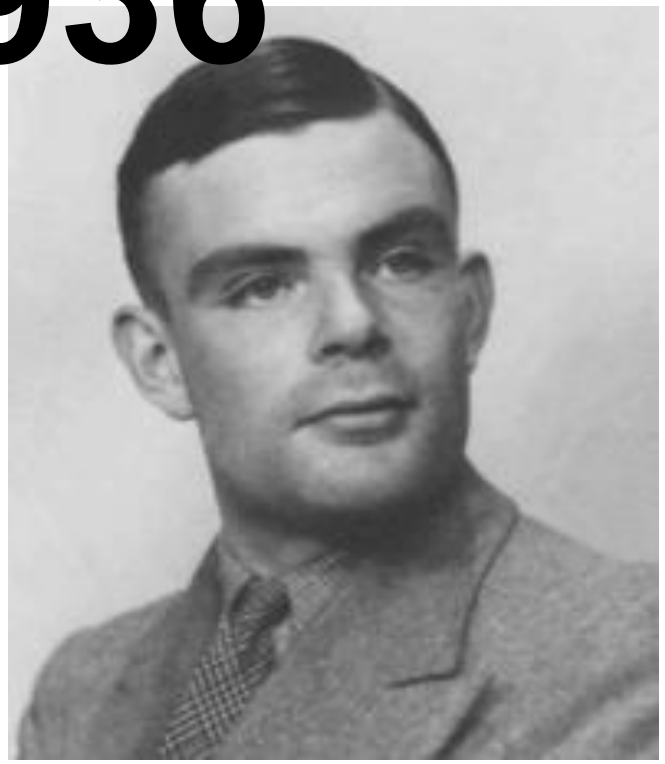
Where are DSLs?

Why DSLs (will) matter?

The (Hi)Story of Software Engineering / Computer Science

1936



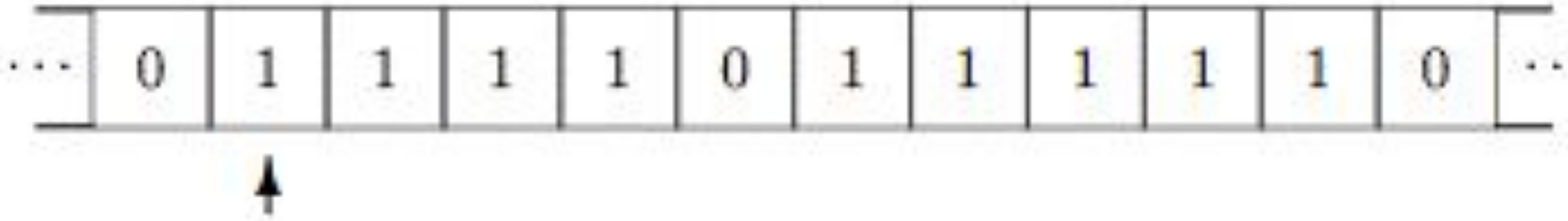


Turing Machine

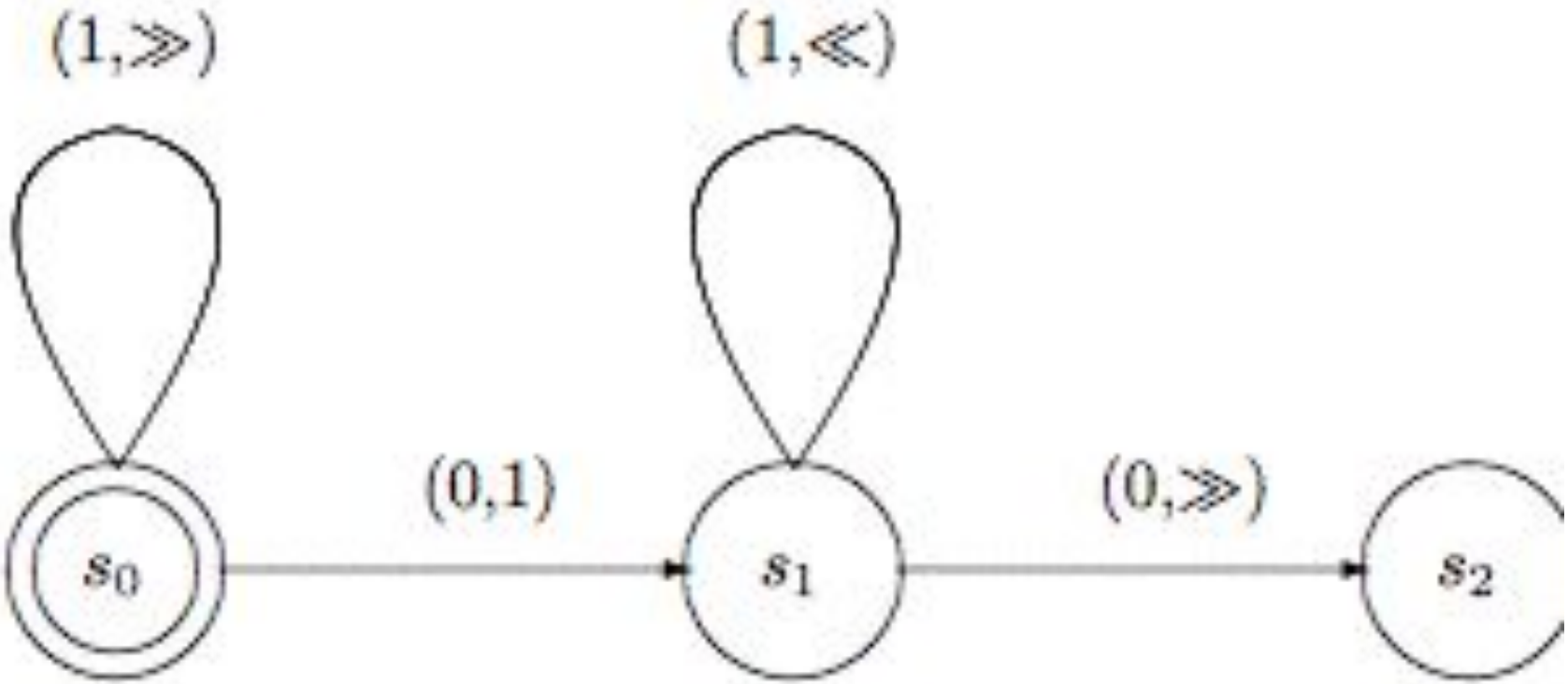
- Infinite tape divided into Cells (0 or 1)
- Read-Write Head
- Transition rules

Write a symbol
or move to left (>>) or right (<<)

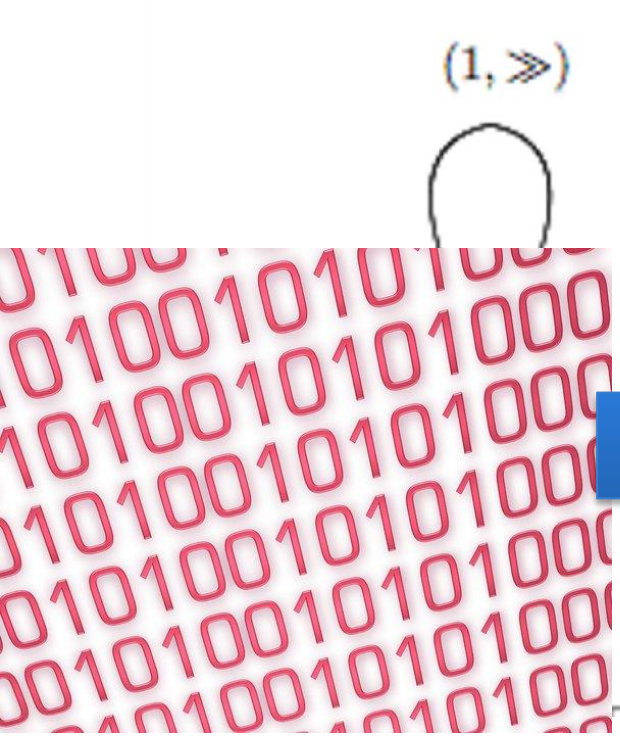
$$\langle\ State_{\text{current}}, Symbol, State_{\text{next}}, Action\ \rangle$$

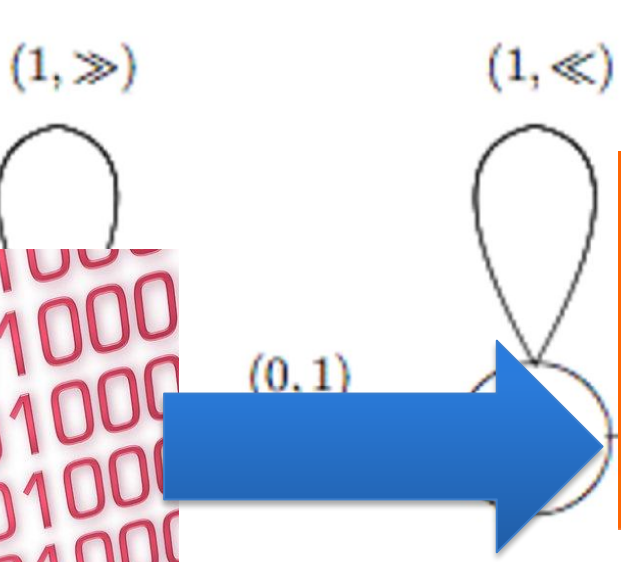


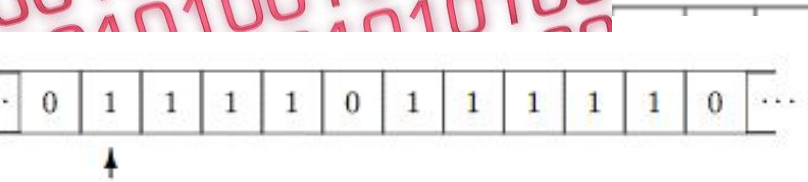
Turing Machine
~ kind of state machine

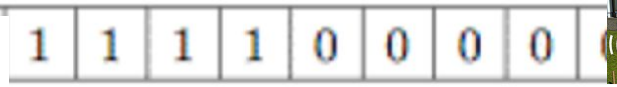


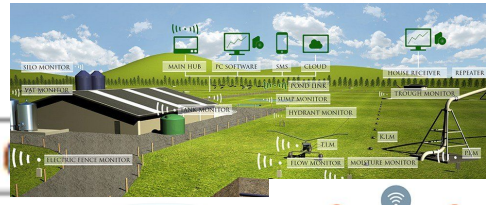
The (Hi)Story of Software Engineering / Computer Science





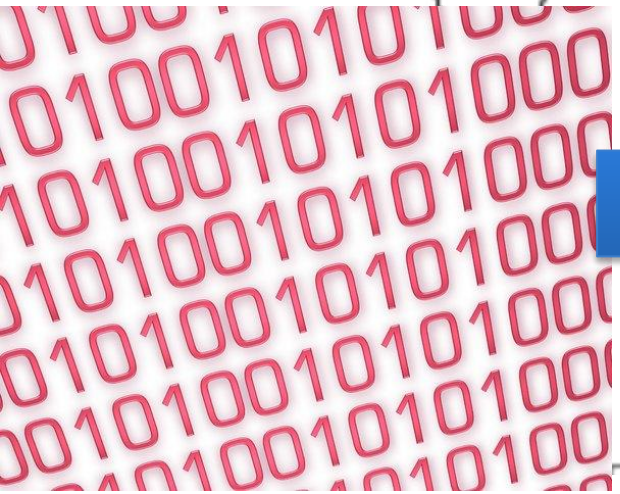


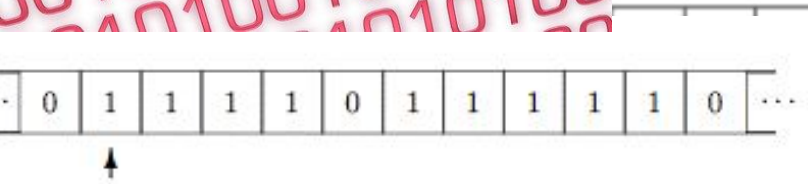


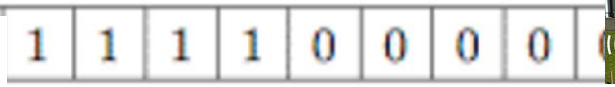


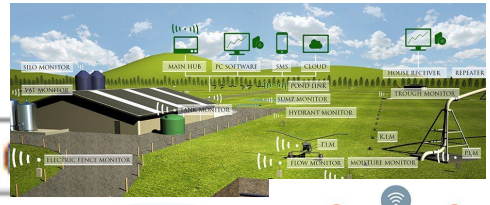
The (Hi)Story of Software Engineering / Computer Science

Software Languages



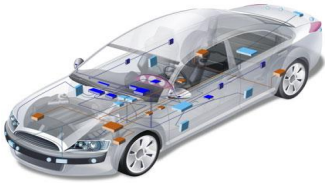






Why aren't we using tapes, states and transitions after all ?

Complex Systems



Distributed systems

Thousands of engineers/expertise

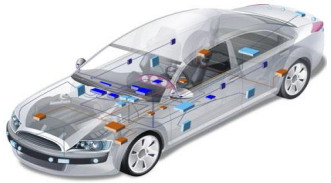
Web dev.

Large-scale systems

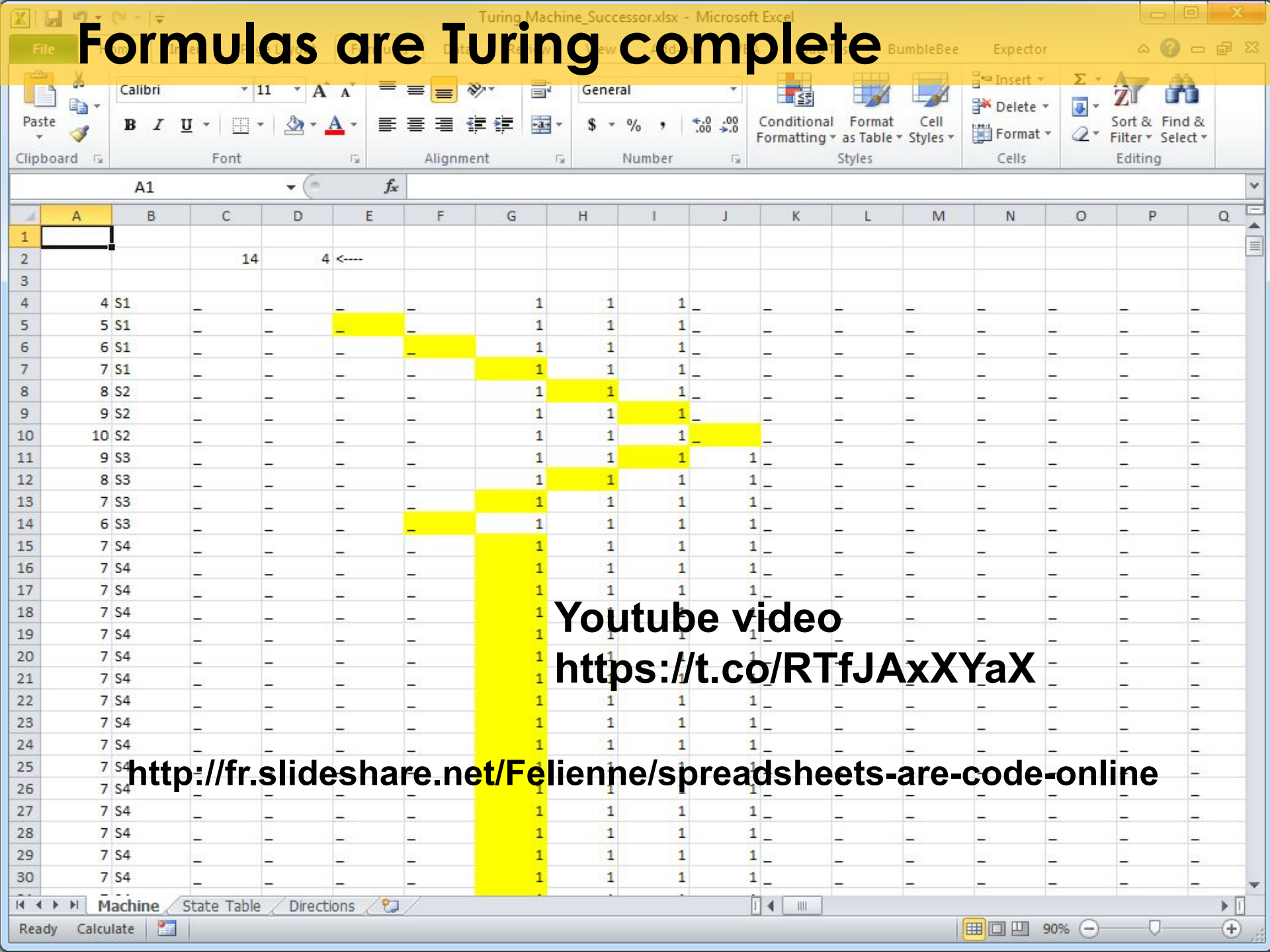
Critical Systems

Why aren’t we using tapes, states and transitions after all ?

You cannot be serious



Formulas are Turing complete



Esoteric programming languages

- Designed to test the boundaries of computer programming language design, as a proof of concept, as software art, or as a joke.
  - extreme paradigms and design decisions
  - Eg https://esolangs.org/wiki/Brainfuck
- Usually, an esolang's creators do not intend the language to be used for mainstream programming.

(brainfuck)
What does it compute?

```
++++++++++[>++++++++>++++++++++>+++<<<-]>++.>+.+++
++++
 ..+++.>++.<<++++++++++++++++.>.+++.------.--------.>+.
```

The same but in Ook Ook

Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook.
Ook! Ook. Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook?
Ook! Ook! Ook? Ook! Ook? Ook. Ook. Ook. Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook. Ook! Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook! Ook. Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook. Ook! Ook.
Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook. Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook.
Ook? Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook! Ook. Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook.
Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
Ook! Ook. Ook. Ook? Ook. Ook? Ook. Ook. Ook! Ook.

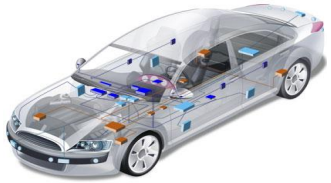
Quizz Time

- Why assembly language is not the mainstream language?
- Why spreadsheets are not used for building Google?
- Why esoteric languages are not used for mainstream programming?

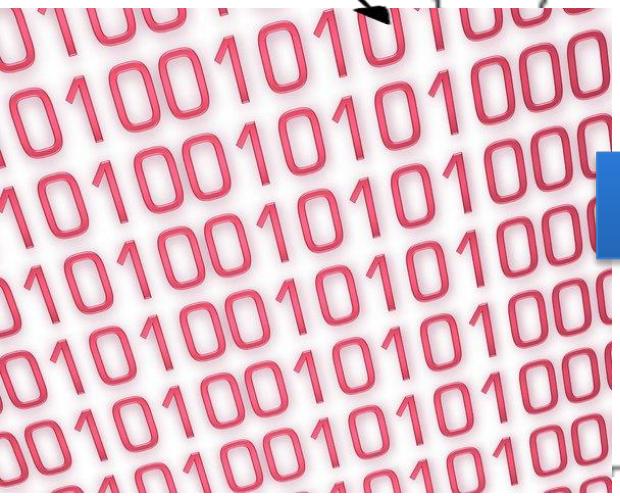
Why aren't we using tapes, states and transitions after all ?

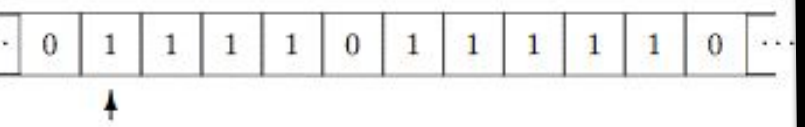
Software Languages

Not fun.
Over complicated.
Hard to write and understand.
No abstractions.
Poor language constructs.
Tooling Support?

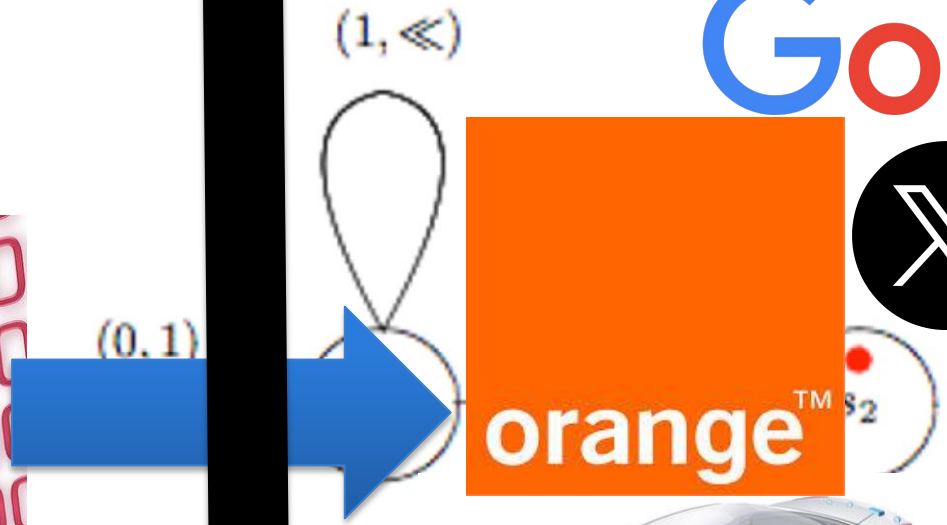


Languages



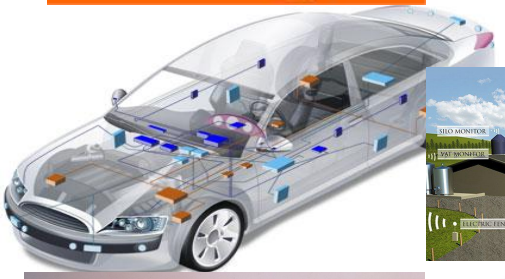


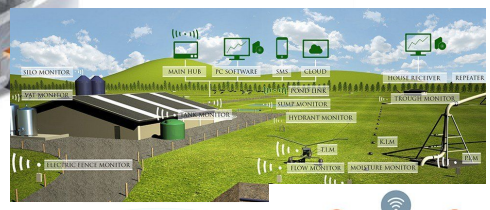
Complex Systems



Google

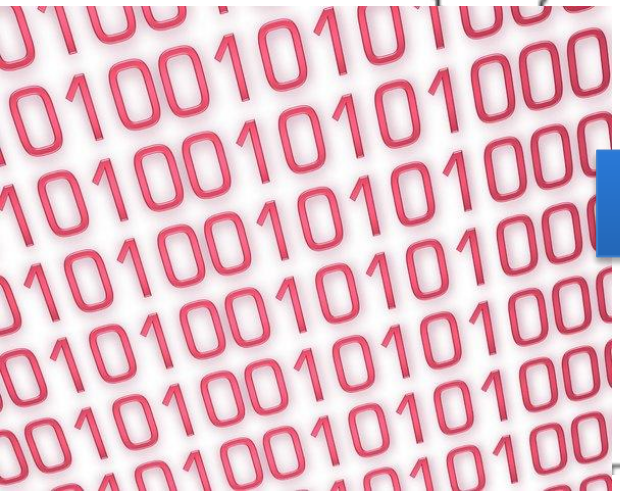
orange™

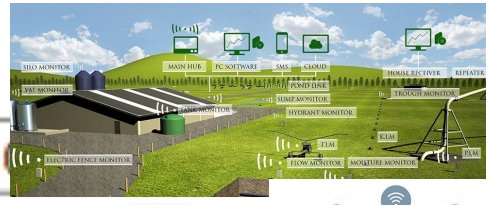




Languages quality impact

Software Languages





COGNITIVE PSYCHOLOGY

How Language Shapes Thought

The languages we speak affect our perceptions of the world

By Lera Boroditsky

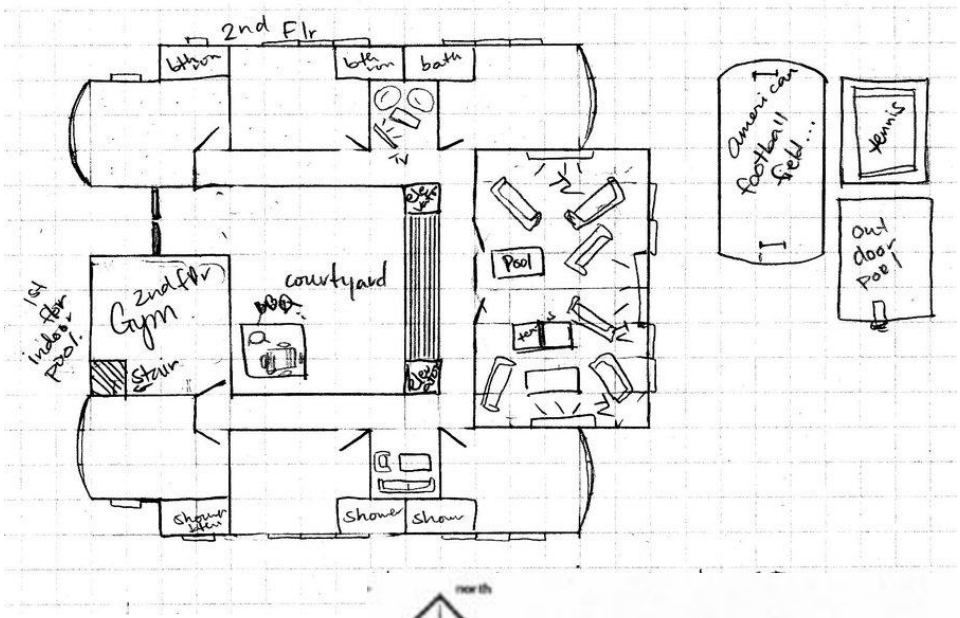
"Even variations in grammar can profoundly affect how we see the world."

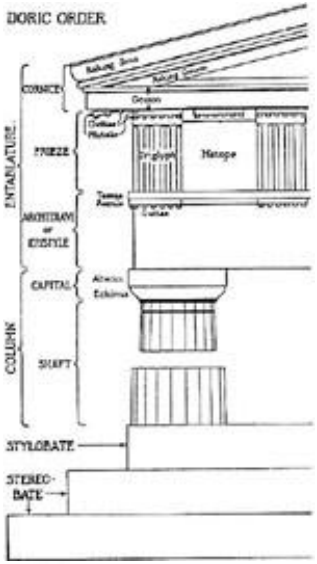
She's talking about real languages; **what about synthetic, programming languages?**

What is a language?

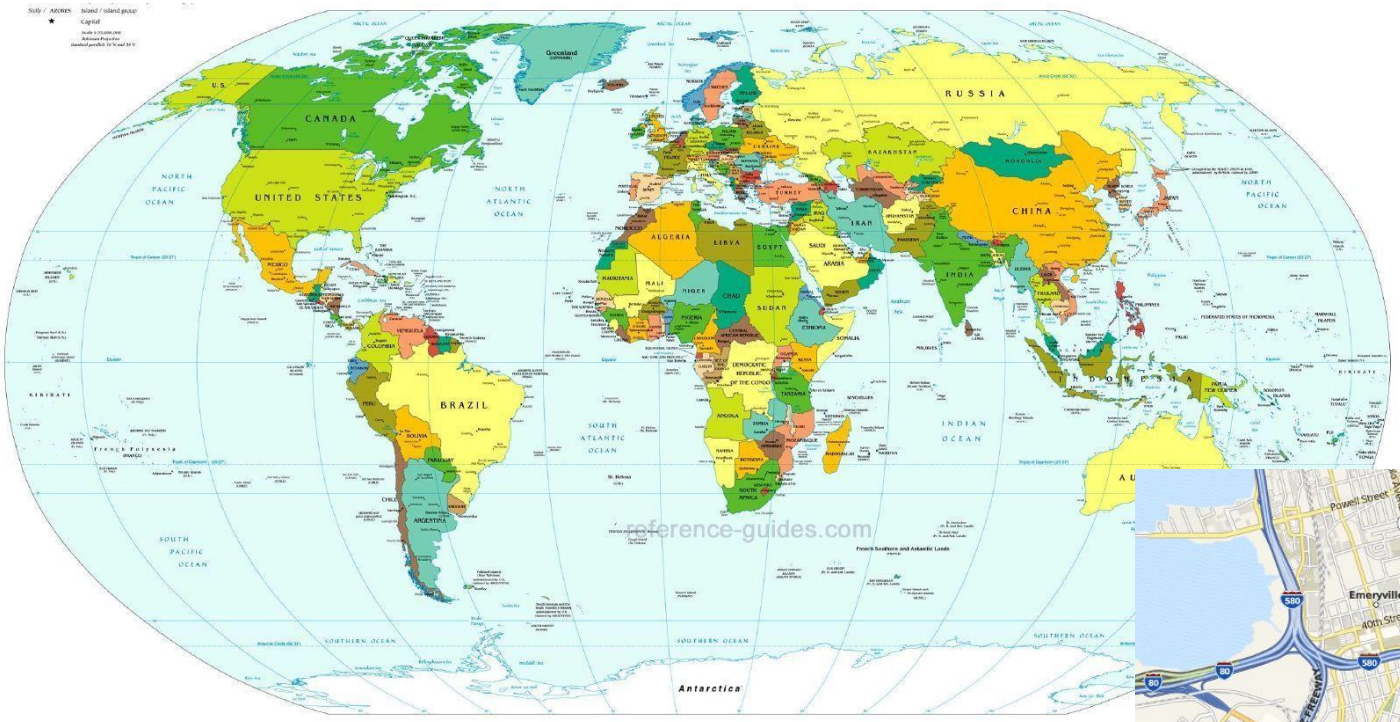
- « A system of signs, symbols, gestures, or rules used in **communicating** »
- « The **special** vocabulary and usages of a scientific, professional, or other group »
- « A system of symbols and rules used for communication with or between computers. »

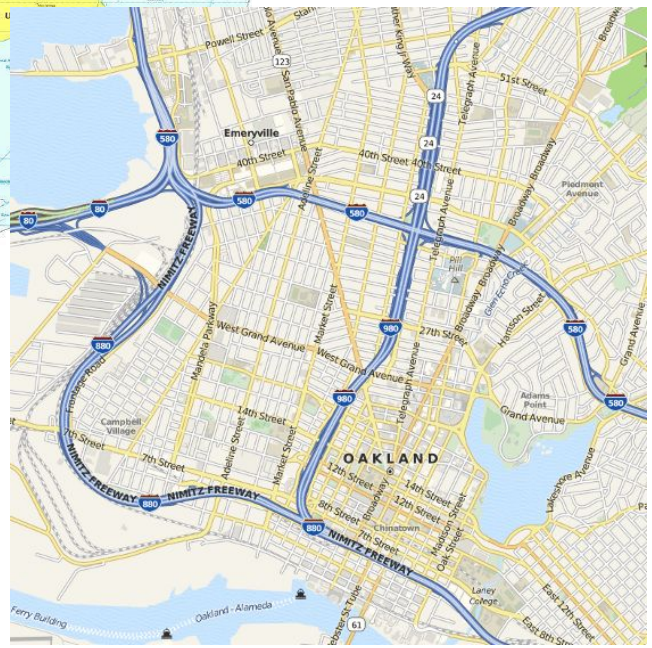
Architecture



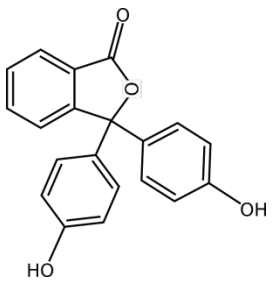


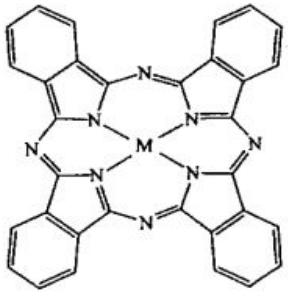
Cartography





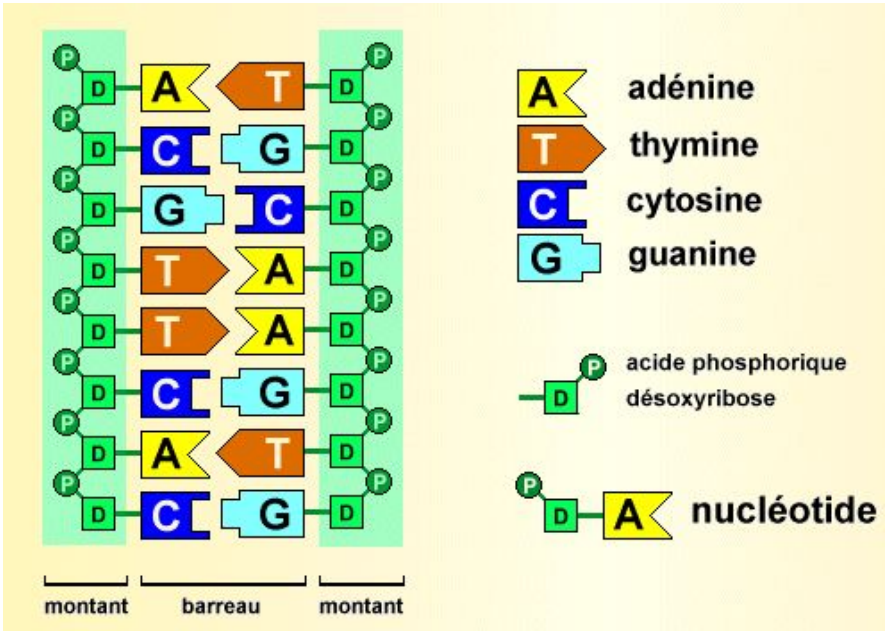
Biology



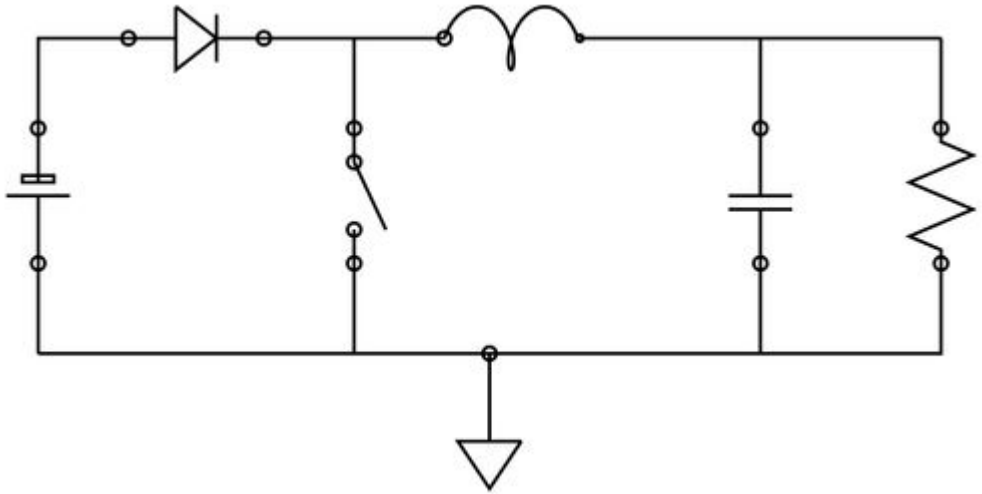


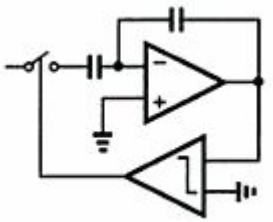
phthalocyanine

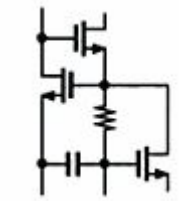
```
        60          70          80          90         100
AGACCCCCAG  CAACCCCCGG  GGGCGTGCGG  CGTCGGTCGT  GTCGTGTGAT
       160         170         180         190         200
AGACCCCGCG  TACGAATGCC  GGTCCACCAA  CAACCCGTGG  GCTTCGCAGC
       260         270         280         290         300
CTGCCGGGCA  TGTACAGTCC  TTGTCGGCAG  TTCTTCCACA  AGGAAGACAT
       360         370         380         390         400
GGCTTGCTGG  GGCCCCCGCC  ACCAGCACTA  CAGACCTCCA  GTACGTCGTG
       460         470         480         490         500
GGCCTATCCC  ACGCTCGCCG  CCAGCCACAG  AGTTATGCTT  GCCGAGTACA
       560         570         580         590         600
GAAGAGGTGG  CGCCGATGAA  GAGACTATTA  AAGCTCGGAA  ACAAGGTGGT
       660         670         680         690         700
ATAGTGGTTA  ACTTCACCTC  CAGACTCTTC  GCTGATGAAC  TGGCCGCCCT
       760         770         780         790         800
AAAATATACA  GGCATTGGGC  CTGGGGTGCG  TATGCTCACG  TGAGACATCT
       860         870         880         890         900
CCTGGAGGAG  GTTCGCCCGG  ACAGCCTGCG  CCTAACGCGG  ATGGATCCCT
       960         970         980         990        1000
AGCAACACCC  AGCTAGCAGT  GCTACCCCCA  TTTTTTAGCC  GAAAGGATTC
      1060        1070                Pvu II site 1090    1100
TGCCGCAGCA  ACTGGGGCAC  GCTATTCTGC  AGCAGCTGTT  GGTGTACCAC
      1160        1170        1180        1190        1200
ACTTGATCTA  TATACCACCA  ATGTGTCATT  TATGGGGCGC  ACATATCGTC
      1260        1270        1280        1290        1300
CTGTCCATGT  ACCTTTGTAT  CCTATCAGCC  TTGGTTCCCA  GGGGGTGTCT
      1360        1370        1380        1390        1400
TGTTTGAGGG  GGTGGTGCCA  GATGAGGTGA  CCAGGATAGA  TCTCGACCAG
      1460        1470        1480        1490        1500
TCAGAGTCTC  AGTTCTATAT  TTAATCTTGG  CCCCAGACTG  CACGTGTATG
      1560        1570        1580        1590        1600
CGATTTGAAG  CGGGGGGGGT  ATGGCGTCAT  CTGATATTCT  GTCGGTTGCA
      1660        1670        1680        1690        1700
AAAAACTACC  GTCTACCTGC  CGGACACTGA  ACCCTGGGTG  GTAGAGACCG
      1760        1770        1780        1790        1800
AAGCTTCATC  GTGGTGCCCT  GCCCTCAAAT  TCTCACAACG  GCTTGAGGAT

CTG.
```

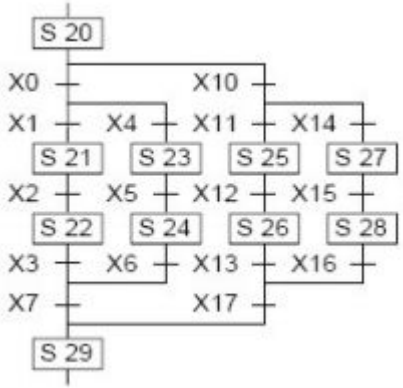


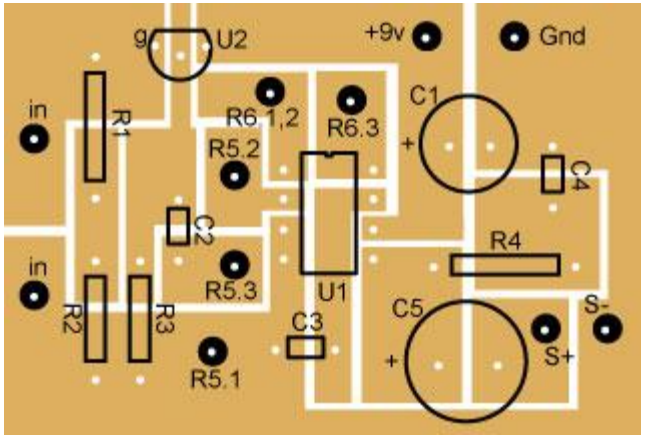
Electronics











In Software Engineering

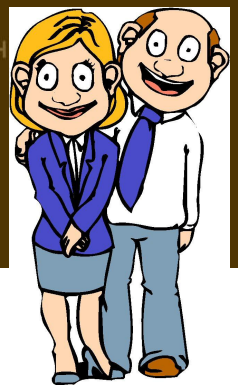
« Languages are the primary way in which system developers communicate, design and implement software systems »

General Purpose Languages

Limits of General Purpose Languages (1)

- **Abstractions** and **notations** used are not natural/suitable for the stakeholders


```
if (newGame) resources.free();
s = FILENAME + 3;
setLocation(); load(s);
loadDialog.process();

try { setGamerColor(RED); }
catch(Exception e) { reset(); }
while (notReady) { objects.make();
if (resourceNotFound) break; }

byte result; // сменить н
music();
System.out.print("");
```



Limits of General Purpose Languages (2)

- Not targeted to a **particular** kind of problem, but to any kinds of software problem.

Domain Specific Languages (DSLs)

- Targeted to a **particular** kind of problem, with dedicated notations (textual or graphical), support (editor, checkers, etc.)
- Promises: more « efficient » languages for resolving a set of specific problems in a domain

Domain Specific Languages (DSLs)

- Long history: used for almost as long as computing has been done.
- You're using DSLs in a daily basis
- You've learnt many DSLs in your curriculum
- Examples to come!

HTML

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml1-transitional.dtd">
<html xml:lang="en" lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
      <title>Hello World</title>
  </head>
  <body>
      <p>My first Web page.</p>
  </body>
</html>
```

Domain: web (markup)

CSS

```css
.CodeMirror {
  line-height: 1;
  position: relative;
  overflow: hidden;
}

.CodeMirror-scroll {
  /* 30px is the magic margin used to hide the element's real scrollbars */
  /* See overflow: hidden in .CodeMirror, and the paddings in .CodeMirror-sizer */
  margin-bottom: -30px; margin-right: -30px;
  padding-bottom: 30px; padding-right: 30px;
  height: 100%;
  outline: none; /* Prevent dragging from highlighting the element */
  position: relative;
}
.CodeMirror-sizer {
  position: relative;
}
```

Domain: web (styling)

SQL

```sql
SELECT Book.title AS Title,
       COUNT(*) AS Authors
 FROM  Book
 JOIN  Book_author
   ON  Book.isbn = Book_author.isbn
 GROUP BY Book.title;

 INSERT INTO example
 (field1, field2, field3)
 VALUES
 ('test', 'N', NULL);
```

Domain: database (query)

Makefile

```
PACKAGE      = package
    VERSION      = ` date "+%Y.%m%d%" `
    RELEASE_DIR  = ..
    RELEASE_FILE = $(PACKAGE)-$(VERSION)

    # Notice that the variable LOGNAME comes from the environment in
    # POSIX shells.
    #
    # target: all - Default target. Does nothing.
    all:
            echo "Hello $(LOGNAME), nothing to do by default"
            # sometimes: echo "Hello ${LOGNAME}, nothing to do by default"
            echo "Try 'make help'"

    # target: help - Display callable targets.
    help:
            egrep "^# target:" [Mm]akefile

    # target: list - List source files
    list:
            # Won't work. Each command is in separate shell
            cd src
            ls

            # Correct, continuation of the same shell
            cd src; \
            ls
```

Domain: software building

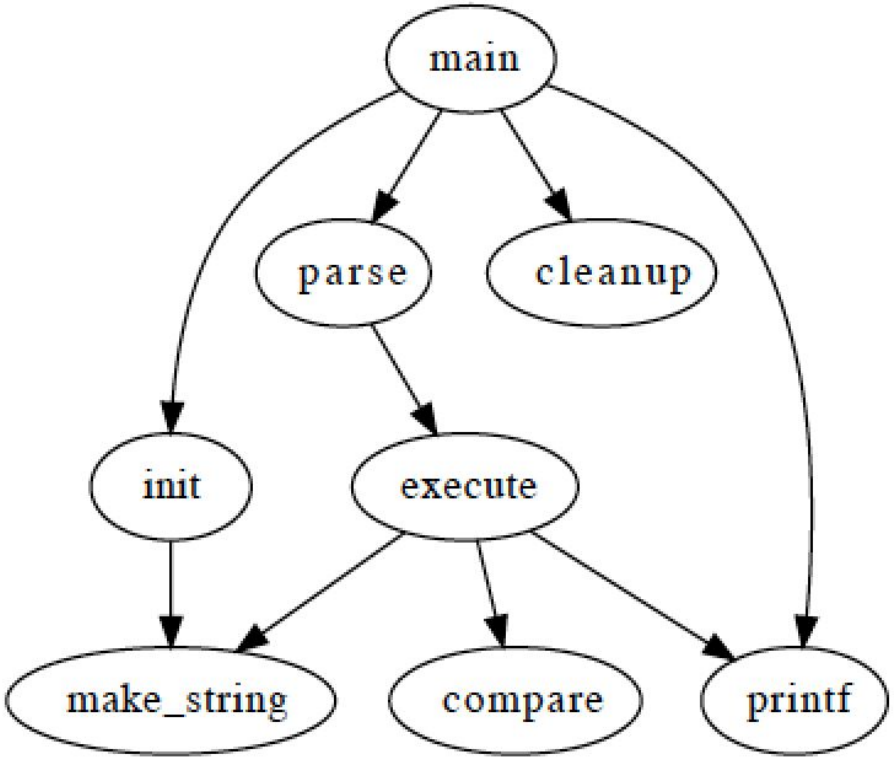
Lighthttpd configuration file

```
server.document-root = "/var/www/servers/www.example.org/pages/"

server.port = 80

server.username = "www"
server.groupname = "www"

mimetype.assign = (
  ".html" => "text/html",
  ".txt" => "text/plain",
  ".jpg" => "image/jpeg",
  ".png" => "image/png"
)

static-file.exclude-extensions = ( ".fcgi", ".php", ".rb", "~", ".inc" )
index-file.names = ( "index.html" )
```

Domain: web server (configuration)

Graphviz

```
digraph G {
main -> parse -> execute;
main -> init;
main -> cleanup;
execute -> make_string;
execute -> printf
init -> make_string;
main -> printf;
execute -> compare;
 }
```



Domain: graph (drawing)

Regular expression

Domain: strings (pattern matching)

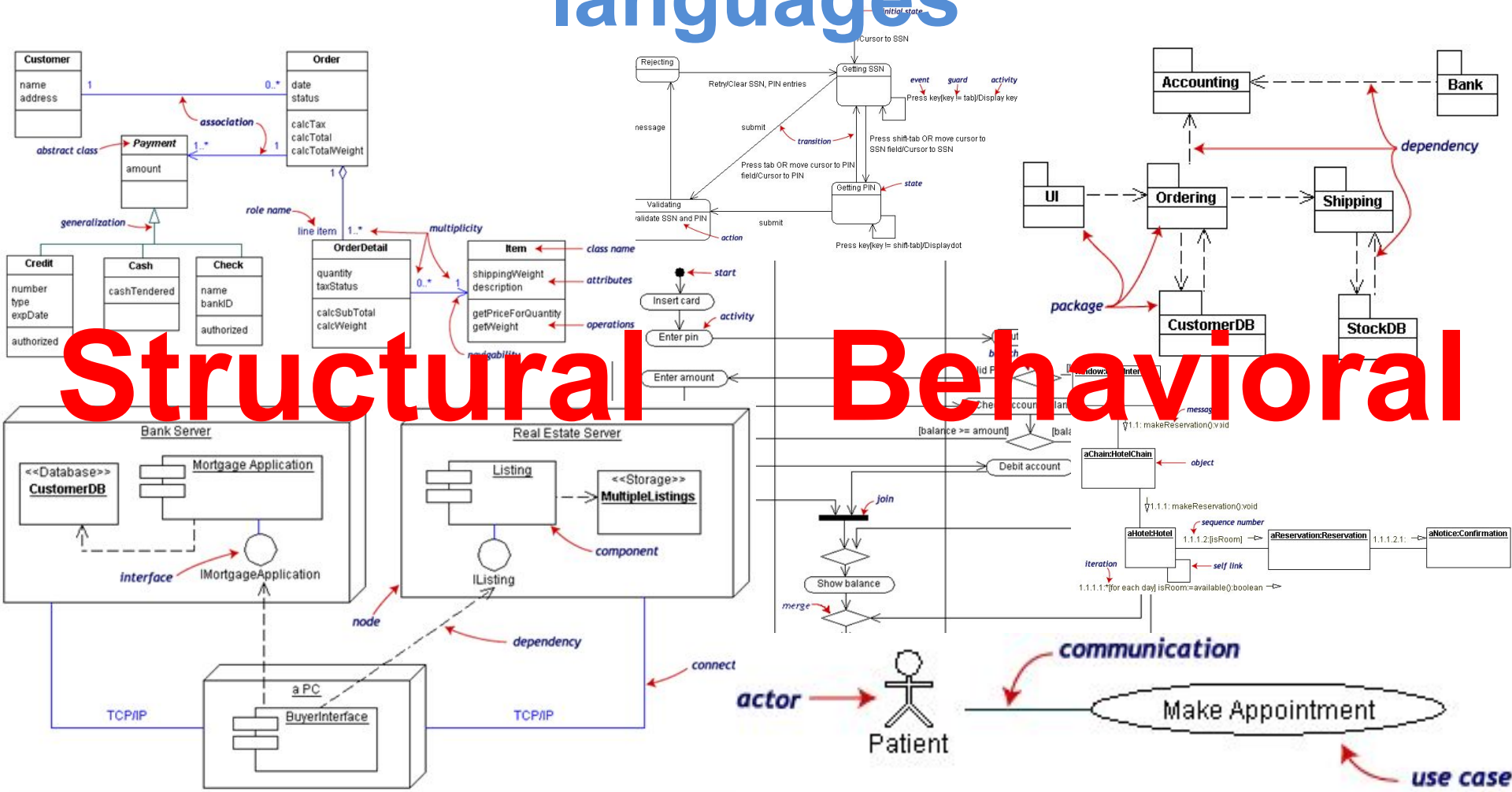
```
<TAG\b[^>]*>(.*?)</TAG>
```

OCL

Domain: model management

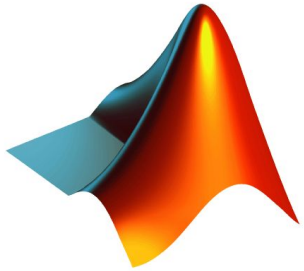
```
self.questions->size
self.employer->size
self.employee->select (v | v.wages>10000 )->size
Student.allInstances
  ->forAll( p1, p2 |
            p1 <> p2 implies p1.name <> p2.name )
```

UML can be seen as a collection of domain-specific modeling languages

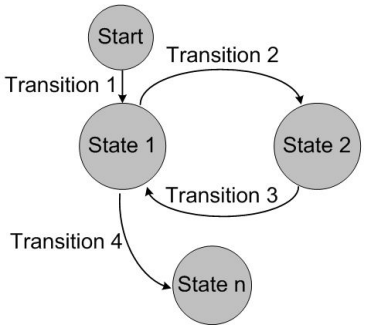


Domain-Specific Languages (DSLs)

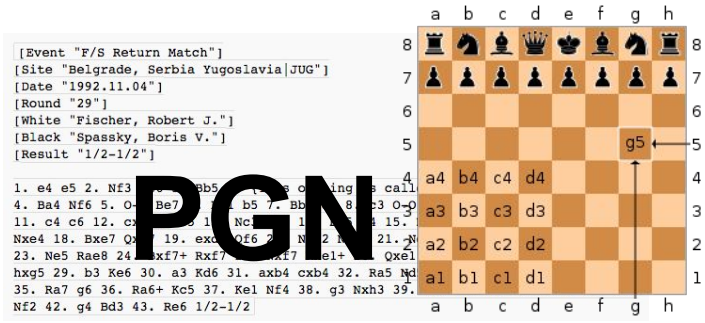
LaTeX



Matlab



Finite State Machine



PGN

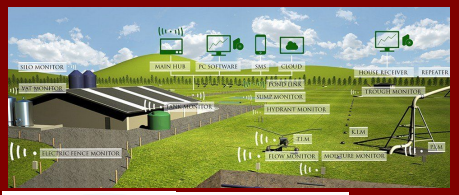
Abstraction Gap

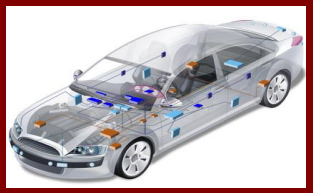
Problem Space

Assembler

C, Java

DSLs





Solution Space

ACM Turing Lecture, « The Humble Programmer » Edsger W. Dijkstra, 1972

« Another lesson we should have learned from the recent past is that the development of 'richer' or 'more powerful' programming languages was a mistake in the sense that these baroque monstrosities, these conglomerations of idiosyncrasies, are really unmanageable, both mechanically and mentally » aka **General-Purpose Languages**

« I see a great future for very systematic and very modest programming languages » aka **Domain-Specific Languages**

Empirical Assessment of MDE in Industry

John Hutchinson, Jon Whittle, Mark Rouncefield
School of Computing and Communications
Lancaster University, UK
+44 1524 510492
{j.hutchinson, j.n.whittle, m.rouncefield}@lancaster.ac.uk

Steinar Kristoffersen
Østfold University College and Møreforskning Molde AS
NO-1757 Halden
Norway
+47 6921 5000
steinar.kristoffersen@hiof.no

Model-Driven Engineering Practices in Industry

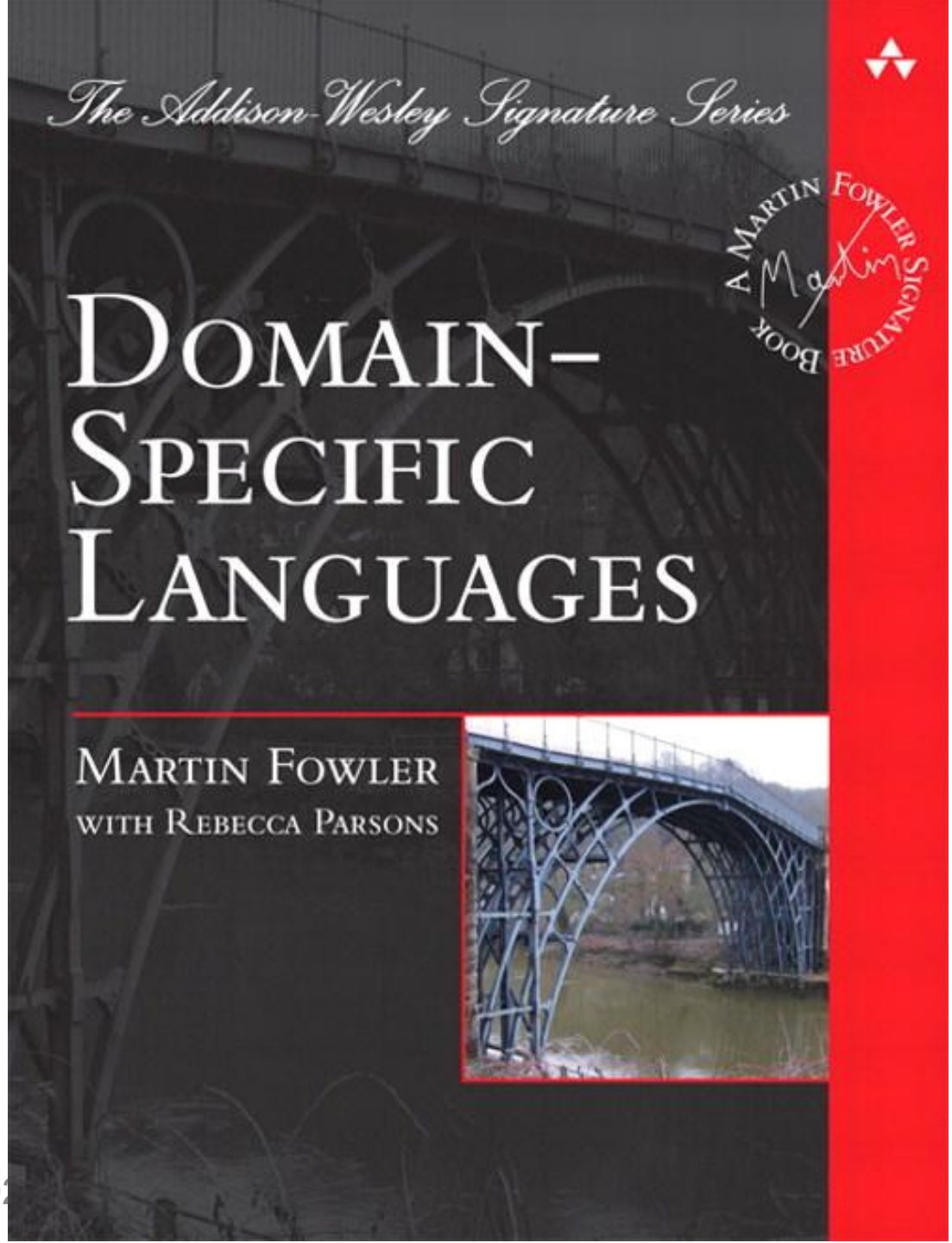
John Hutchinson
School of Computing and Communications
Lancaster University, UK
+44 1524 510492
{j.hutchinson@lancaster.ac.uk}

Mark Rouncefield
School of Computing and Communications
Lancaster University, UK
+44 1524 510492
{m.rouncefield@lancaster.ac.uk}

Jon Whittle
School of Computing and Communications
Lancaster University, UK
+44 1524 510492
{j.n.whittle@lancaster.ac.uk}

2011

« Domain-specific languages are far more prevalent than anticipated »


The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Domain-Specific Languages
Martin Fowler
with Rebecca Parsons

2011

What is a domain-specific language ?

- « Language **specially** designed to perform a task in a **certain domain** »
- « A formal processable language targeting at a **specific viewpoint or aspect** of a software system. Its **semantics and notation** is designed in order to support working with that viewpoint as good as possible »
- « A computer language that's targeted to a particular kind of problem, **rather than a general purpose language** that's aimed at any kind of software problem. »

GPL (General Purpose Language)

A GPL provides notations that are used to describe a computation in a human-readable form that can be translated into a machine-readable representation.

A GPL is a formal notation that can be used to describe problem solutions in a precise manner.

A GPL is a notation that can be used to write programs.

A GPL is a notation for expressing computation.

A GPL is a standardized communication technique for expressing instructions to a computer. It is a set of syntactic and semantic rules used to define computer programs.

Promises of domain-specific languages

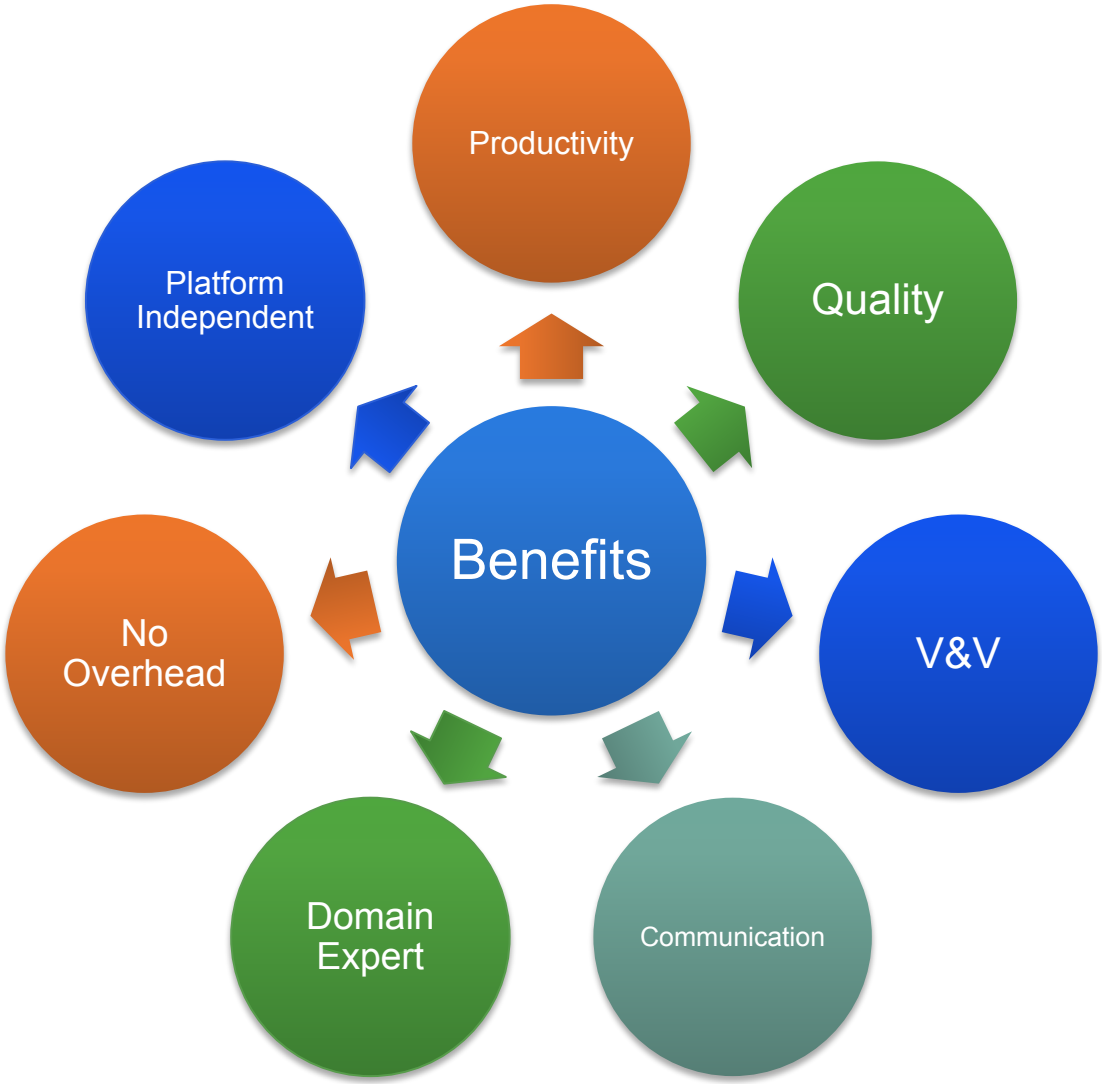
Higher abstractions

Avoid redundancy

Separation of concerns

Use domain concepts

Promises of domain-specific languages



GeneralPL vs DomainSL

The boundary isn't as clear as it could be. Domain-specificity is not black-and-white, but instead gradual: a language is more or less domain specific

	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

External DSLs vs Internal DSLs

- An **external** DSL is a completely separate language and has its own custom syntax/tooling support (e.g., editor)

- An internal DSL is more or less a set of APIs written on top of a host language (e.g., Java).
  - Fluent interfaces

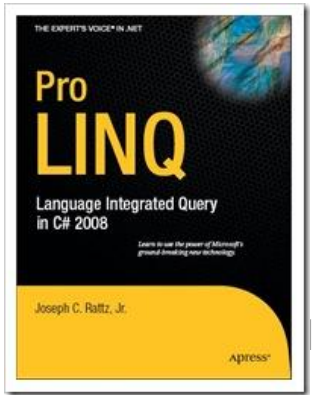
External vs Internal DSL (SQL example)

```sql
-- Select all books by authors born after 1920,
-- named "Paulo" from a catalogue:
SELECT *
  FROM t_author a
  JOIN t_book b ON a.id = b.author_id
 WHERE a.year_of_birth > 1920
   AND a.first_name = 'Paulo'
 ORDER BY b.title
```

```
Result<Record> result =
create.select()
      .from(T_AUTHOR.as("a"))
      .join(T_BOOK.as("b")).on(a.ID.equal(b.AUTHOR_ID))
      .where(a.YEAR_OF_BIRTH.greaterThan(1920)
      .and(a.FIRST_NAME.equal("Paulo")))
      .orderBy(b.TITLE)
      .fetch();
```

Internal DSL (LINQ/C# example)

```
// DataContext takes a connection string
DataContext db = new   DataContext("c:\\northwind\\northwnd.mdf");
// Get a typed table to run queries
Table<Customer> Customers = db.GetTable<Customer>();
// Query for customers from London
var q =
   from c in Customers
   where c.City == "London"
   select c;
foreach (var cust in q)
   Console.WriteLine("id = {0}, City = {1}", cust.CustomerID, cust.City);
```



Internal DSL

- « Using a host language (e.g., Java) to give the host language the feel of a particular language. »
- **Fluent** Interfaces
  - « The more the use of the API has that language like flow, the more fluent it is »

```
Result<Record> result =
create.select()
      .from(T_AUTHOR.as("a"))
      .join(T_BOOK.as("b")).on(a.ID.equal(b.AUTHOR_ID))
      .where(a.YEAR_OF_BIRTH.greaterThan(1920)
      .and(a.FIRST_NAME.equal("Paulo")))
      .orderBy(b.TITLE)
      .fetch();
```

```
-- Select all books by authors born after 1920,
-- named "Paulo" from a catalogue:
SELECT *
  FROM t_author a
  JOIN t_book b ON a.id = b.author_id
 WHERE a.year_of_birth > 1920
   AND a.first_name = 'Paulo'
 ORDER BY b.title
```

SQL in… Java

DSL in GPL

```
Connection con = null;

// create sql insert query
String query = "insert into user values(" + student.getId() + ",'"
  + student.getFirstName() + "','" + student.getLastName()
  + "','" + student.getEmail() + "','" + student.getPhone()
  + "')";
try {
 // get connection to db
 con = new CreateConnection().getConnection("checkjdbc", "root",
   "root");

 // get a statement to execute query
 stmt = con.createStatement();

 // executed insert query
 stmt.execute(query);
 System.out.println("Data inserted in table !");
```

Regular expression in… Java

DSL in GPL

```
public class RegexTestStrings {
  public static final String EXAMPLE_TEST = "This is my small example "
      + "string which I'm going to " + "use for pattern matching.";

  public static void main(String[] args) {
    System.out.println(EXAMPLE_TEST.matches("\\w.*"));
    String[] splitString = (EXAMPLE_TEST.split("\\s+"));
    System.out.println(splitString.length);// Should be 14
    for (String string : splitString) {
      System.out.println(string);
    }
    // Replace all whitespace with tabs
    System.out.println(EXAMPLE_TEST.replaceAll("\\s+", "\t"));
  }
}
```

Terminology

- Traditional dichotomy between internal DSL and external DSL (Fowler et al., 2010)
  - Fluent APIs
  - Internal DSLs (shallow / deep embedded DSLs)
  - External DSLs
- Boundary between DSL and GPL is not that clear (Voelter et al., 2013)
  - What is and what is not a DSL is still a debate

Internal DSLs vs External DSL

Criterion	Internal DSL	External DSL
Learning curve	Low for developers	Better suited for domain experts
Development cost	Low (no need for custom parsing)	Higher (custom syntax, parser, tools required)
Developer familiarity	High (same as host language)	Requires learning a new language
Communication with domain	Less natural for non-technical users	Custom syntax, closer to the domain
Integration with host code	Easy (embedded in the host language)	Requires bridging (e.g., transformation, code gen)
Expressiveness	Limited by host language syntax	Very flexible (custom grammar and structure)
Tooling	Uses existing tooling of the host language	Needs dedicated tooling (editor, parser, etc.)

Focus of the course

- **external DSL:** a completely separate language with its own custom syntax and tooling support (e.g., editor)

Plan

- Domain-Specific Languages (DSLs)
  - Languages and abstraction gap
  - Examples and rationale
  - DSLs vs General purpose languages, taxonomy
- External DSLs
  - Grammar and parsing
  - EMF, Xtext, Langium, *Sirius*

Contract

- Better understanding/source of inspiration of software languages and DSLs
  - Revisit of history and existing languages

- Foundations and practice of EMF, Xtext, Langium *(and Sirius)*
  - State-of-the-art language workbench (mature and used in a variety of industries)

DSL = Syntax + Services

Specialized notation:

Textual or Graphical
Specific Vocabulary
Idiomatic constructs

Specialized tools/IDE:

Editor with auto-completion, syntax highlighting, etc.
Compiler
Interpreter
Debugger
Profiler
Syntax/Type Checker
…

Language workbenches

- Tools for reducing the gap between the design and implementation of (external) domain-specific languages

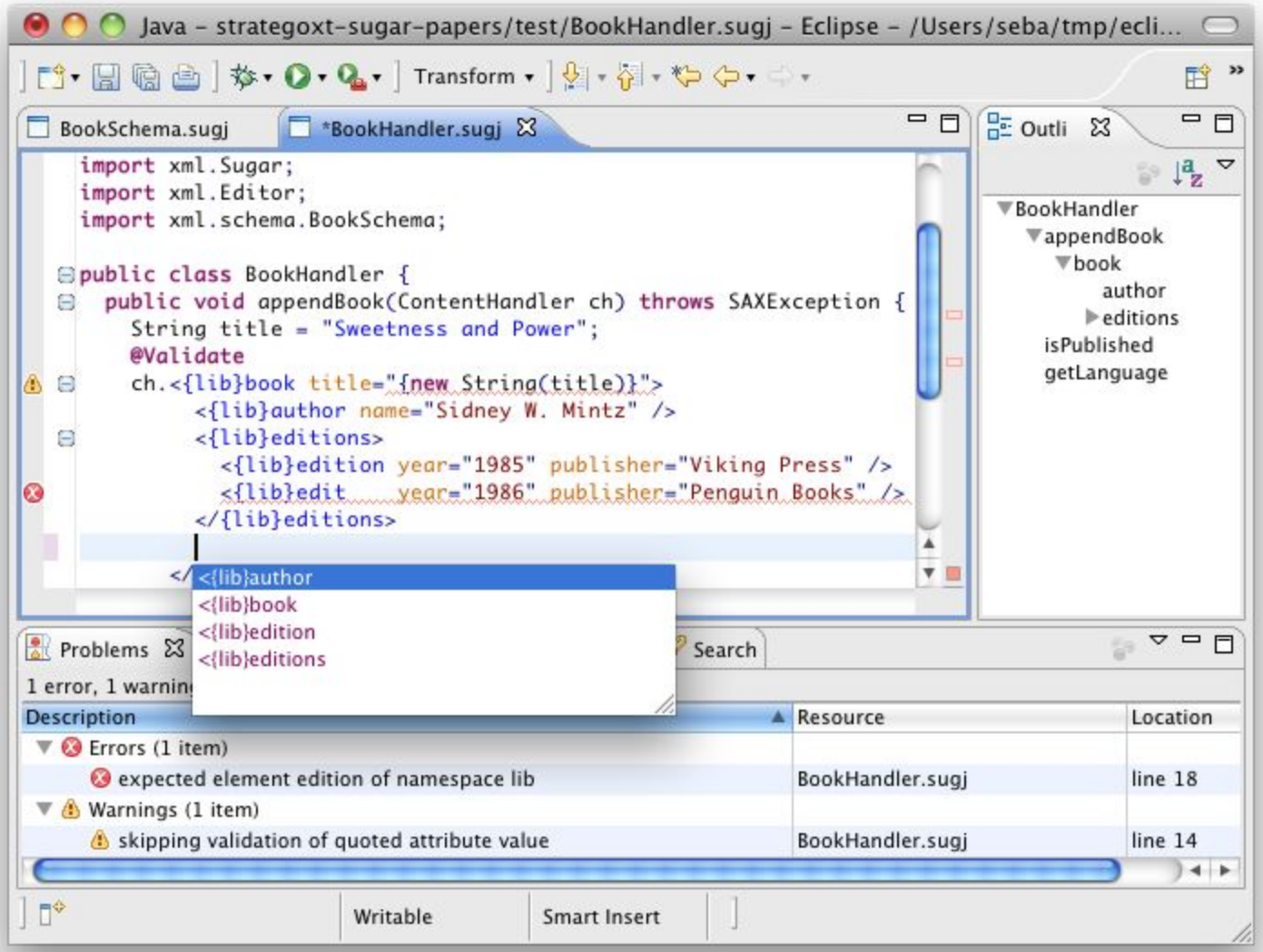
- The Killer App for DSLs? http://www.martinfowler.com/articles/languageWorkbench.html

Language Workbenches

Erdweg et al. SLE'13

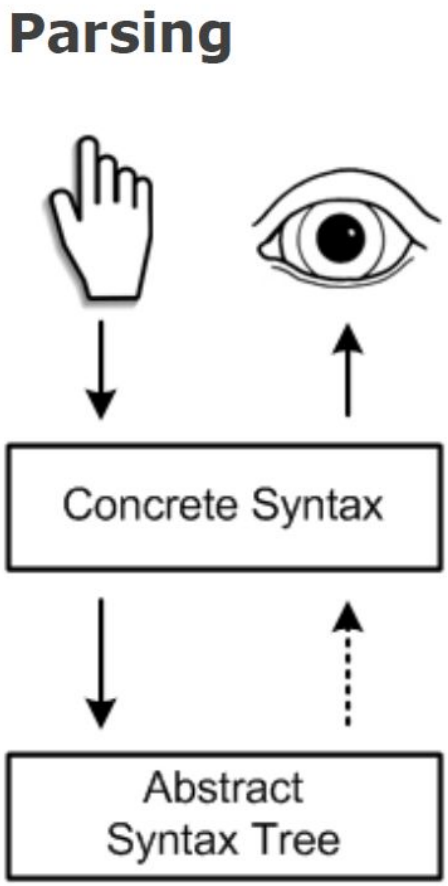
		Ensō	Más	MetaEdit+	MPS	Onion	Rascal	Spoofax	SugarJ	Whole	Xtext
Notation	Textual	●	●		●	●	●	●	●	●	●
	Graphical	●	◒	●			◒			●	
	Tabular		●	●	●					●	
	Symbols			●	●					●	
Semantics	Model2Text		●	●	●	●	●	●	●	●	●
	Model2Model			●	●	●	●	●	●	●	●
	Concrete syntax			●	●	●	●	●	●		
	Interpretative	●		●	●		◒	●		●	●
Validation	Structural	●	●	●	●	●	●	●	●	●	●
	Naming	◒	●	●	●	●		●		●	◒
	Types				●				●		●
	Programmatic	●			●	●	●	●	●		●
Testing	DSL testing				●		◒	●		●	●
	DSL debugging	●		●	●		●			●	●
	DSL prog. debugging	●			●					●	●
Composability	Syntax/views	●		●	●	●	●	●	●	●	◒
	Validation			●	●	●	●	●	●	●	●
	Semantics	●		●	●	●	●	●	●		●
	Editor services			●	●	●	●	●	●		●
Editing mode	Free-form	●		●		●	●	●	●		●
	Projectional		●		●	●				●	
Syntactic services	Highlighting		◒	●	●	●	●	●	●	●	●
	Outline			●	●	●	●	●	●	●	●
	Folding		●	●	●	●	●	●	●	●	●
	Syntactic completion			●	●	●		●	●		●
	Diff	●		●	●	●	●	●	●		●
	Auto formatting	●	●	●	●	●	●	●		●	●
Semantic services	Reference resolution		●	●	●	●	●	●	●		●
	Semantic completion		●	●	●	●	●	●	●	●	●
	Refactoring		◒	●	●		●	●		●	
	Error marking		●	●	●	●	●	●	●	●	●
	Quick fixes				●						●
	Origin tracking	●		●	●		●	●	●		●
	Live translation			●		●	◒	●		●	●

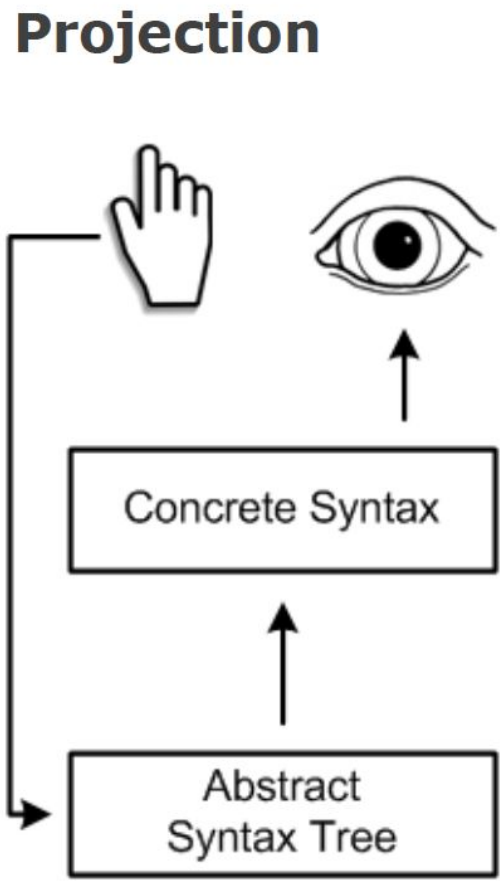
Table 1: Language Workbench Features (● = full support, ◒ = partial/limited support)



Sebastian Erdweg, Tillmann Rendel, Christian Kästner, and Klaus Ostermann. Sugarj: Library-based syntactic language extensibility. OOPSLA'11

Projectional editing

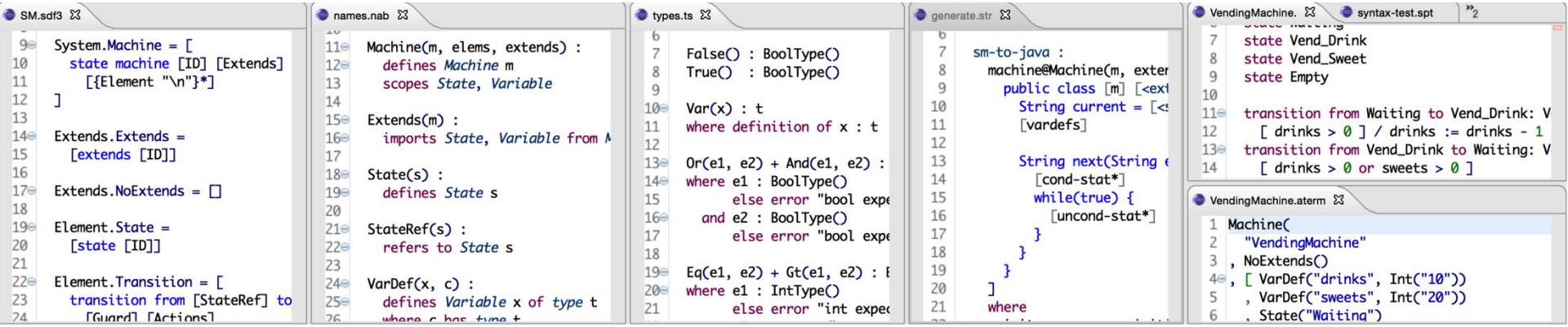




Projectional editing

```
exported component Judge extends nothing {
  provides FlightJudger judger
  int16 points = 0;
  void judger_reset() <= op judger.reset {
    points = 0;
  } runnable judger_reset
  void judger_addTrackpoint(Trackpoint* tp) <= op judger.addTrackpoint {
    points += 0
  } runnable judger_addTrackpoint
  int16 judger_getResult() <= op judger.getResult {
    return points;
  } runnable judger_getResult
} component Judger
```

	tp->alt <= 2000 m	tp->alt >= 2000 m
tp->speed < 150 mps	0	10
tp->speed >= 150 mps	5	20



The Spoofax Language Workbench

Spoofax is a platform for developing textual domain-specific languages with full-featured Eclipse editor plugins.

With the Spoofax language workbench, you can write the grammar of your language using the high-level SDF grammar formalism. Based on this grammar, basic editor services such as syntax highlighting and code folding are automatically provided. Using high-level descriptor languages, these services can be customized. More sophisticated services such as error marking and content completion can be specified using rewrite rules in the Stratego language.

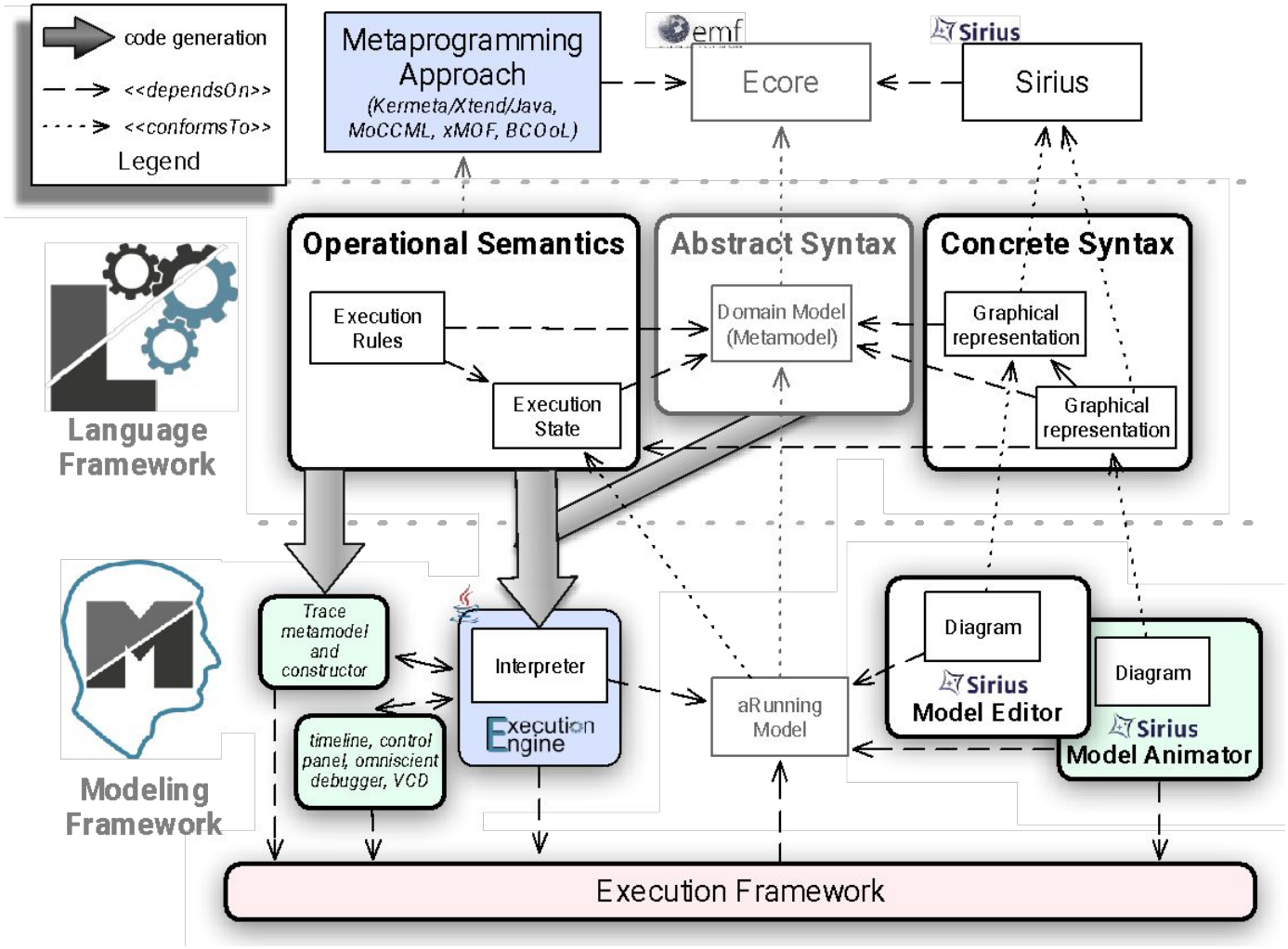
Meta Languages

Language definitions in Spoofax are constructed using the following meta-languages:

- The SDF3 syntax definition formalism
- The NaBL name binding language
- The TS type specification language
- The Stratego transformation language

http://metaborg.org/spoofax/#meta-languages

GEMOC Studio



EMF, a popular, open source, easy-to-use modeling framework for developing DSLs

Your domain model in 5’

Eclipse Modeling: Overview

- Eclipse Modeling is the umbrella project for **all things about modeling** that happen on the Eclipse platform:

  The Eclipse Modeling Project (EMP) focuses on the evolution and promotion of model-based development technologies within the Eclipse community by providing a unified set of modeling frameworks, tooling, and standards implementations.

- Eclipse Modeling is **not formally related to OMG**, but implements several of their standards.

- It is fair to say that **many leading edge modeling** tools are hosted/developed at Eclipse Modeling.

- Everything **Open Source** under the Eclipse Public License

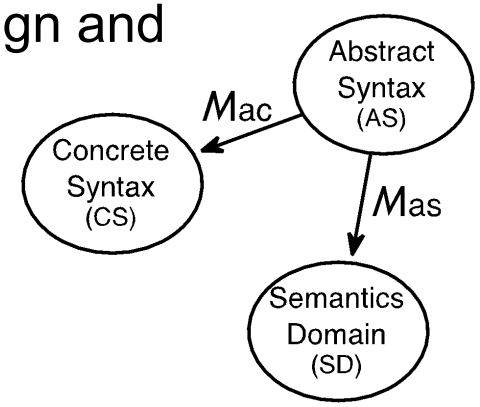
Eclipse Modeling: Overview

The answer to "What is Eclipse Modeling?" depends on who you ask!

A set of Eclipse projects dedicated to…

- … **Modeling**: modeling tools
  - Model Development Tools (UML2, OCL, SysML, MARTE, BPMN2, etc.)

- … **Metamodeling**: workbench for language design and implementation
  - Abstract Syntax Development (EMF)
  - Concrete Syntax Development (GMP, TMF)
  - Model Transformation (M2M, M2T)



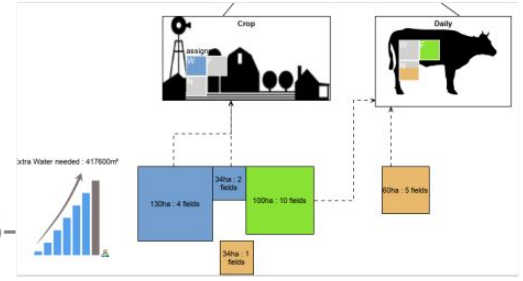
- See http://www.eclipse.org/modeling

Eclipse Modeling

Xtext

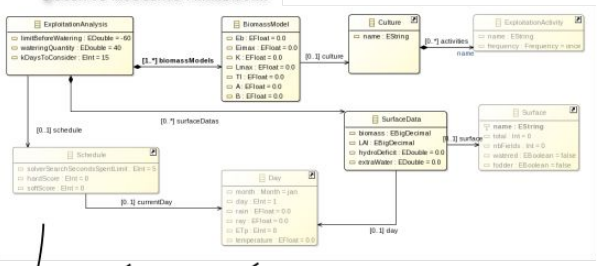
```
culture corn {

    activity LABOUR from 1 jan to 28 feb
    using 1 Tractor and 1 People

    activity SEMIS from 15 mar to 15 apr [
        after LABOUR && no rain since 3 days && temperature > 10 °C
    ] using 1 Tractor and 2 People

    activity IRRIGATION weekly from 15 jun to 15 aug [
        after SEMIS
    ] using 1 Tractor and 1 People

    activity FERTILISATION from 15 mar to 15 jun [
        after SEMIS is done since 30 days &&
        no rain since 1 days
    ] using 1 Tractor and 1 People

    activity RECOLTE from 1 sept to 30 sept  [
        grain is "mature"
    ] using 1 Tractor and 2 People

}

culture weat {

    activity LABOUR from 1 sept to 30 sept  [
        no rain since 3 days
    ] using 1 Tractor and 1 People

    activity SEMIS from 1 oct to 31 oct  [
        after LABOUR &&
        no rain since 3 days &&
        temperature > 5°C
    ] using 1 Tractor and 1 People

    activity FERTILISATION from 1 feb to 28 feb [
        after SEMIS is done since 30 days &&
        no rain since 1 days
     ] using 1 Tractor and 1 People

    activity RECOLTE from 1 jun to 30 jun  [
        grain is "mature"
    ] using 1 Tractor and 1 People

}
```

Sirius



...

emf
ECLIPSE MODELING FRAMEWORK



...

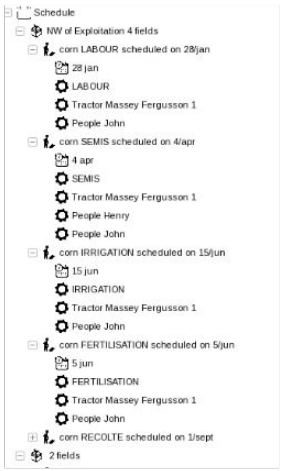
Acceleo
Generate Code

EMFCompare
Diff & Merge
SCM integration

Sirius Animator
Model Debugging
Animation

Domain Specific API

Java Logic
Business rules



EMF: Overview

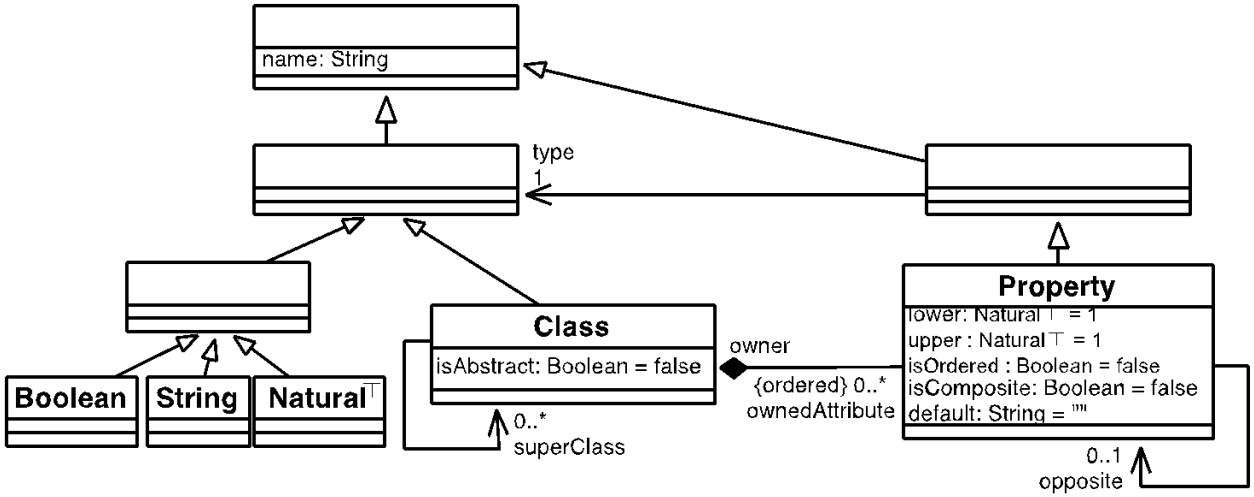
- What is it?
  - **Meta**Modeling (think of UML/OCL)
  - Interoperability (think of XMI)
  - Editing tool support (think Eclipse)
  - Code generation (think of MDA)

- EMF serves as the foundation: It provides the Ecore meta-metamodel, and frameworks and tools around it for tasks such as
  - Editing
  - Transactions
  - Validation
  - Query
  - Distribution/Persistence (CDO, Net4j, Teneo)

- See http://www.eclipse.org/modeling/emf

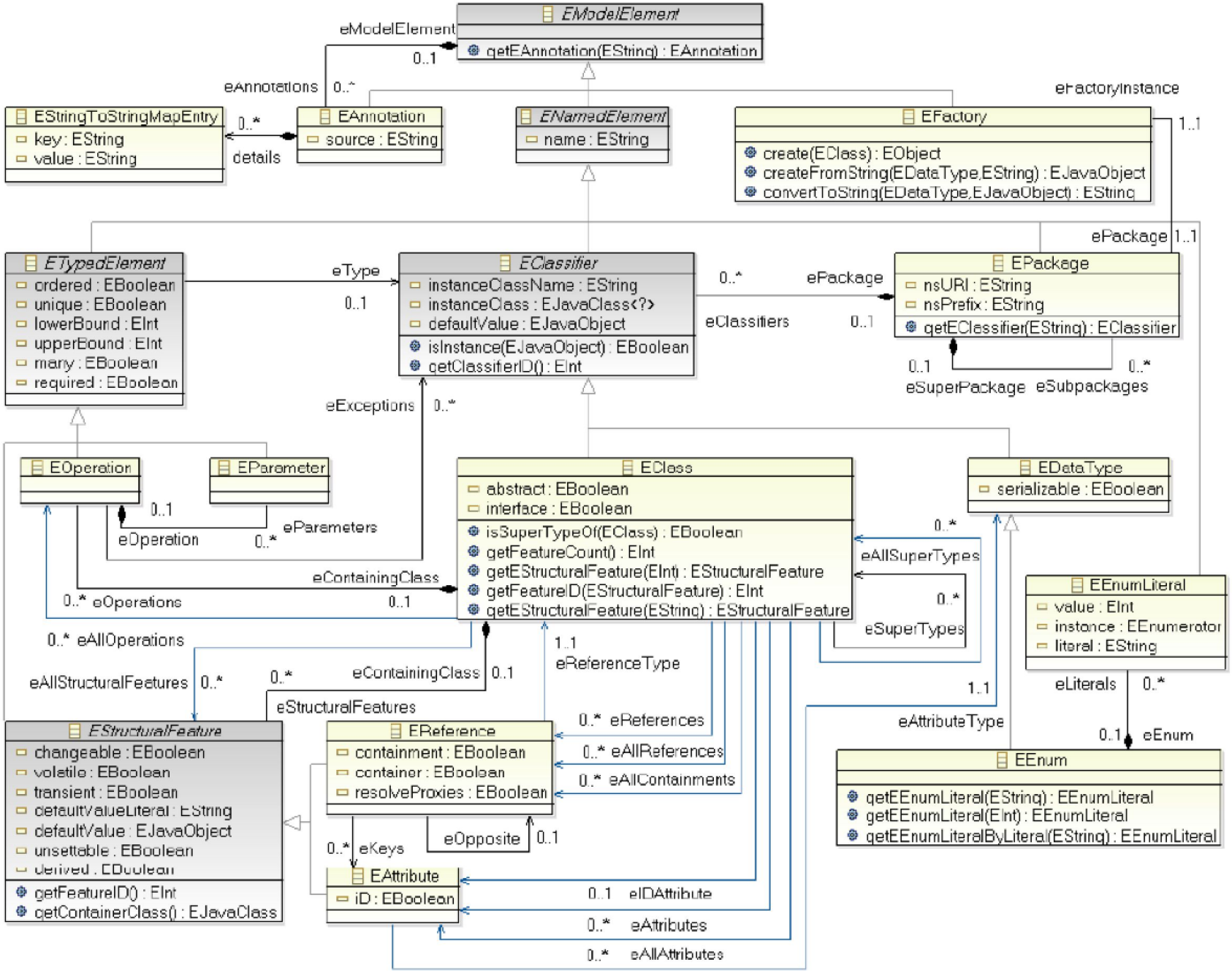
OMG (Essential) MOF

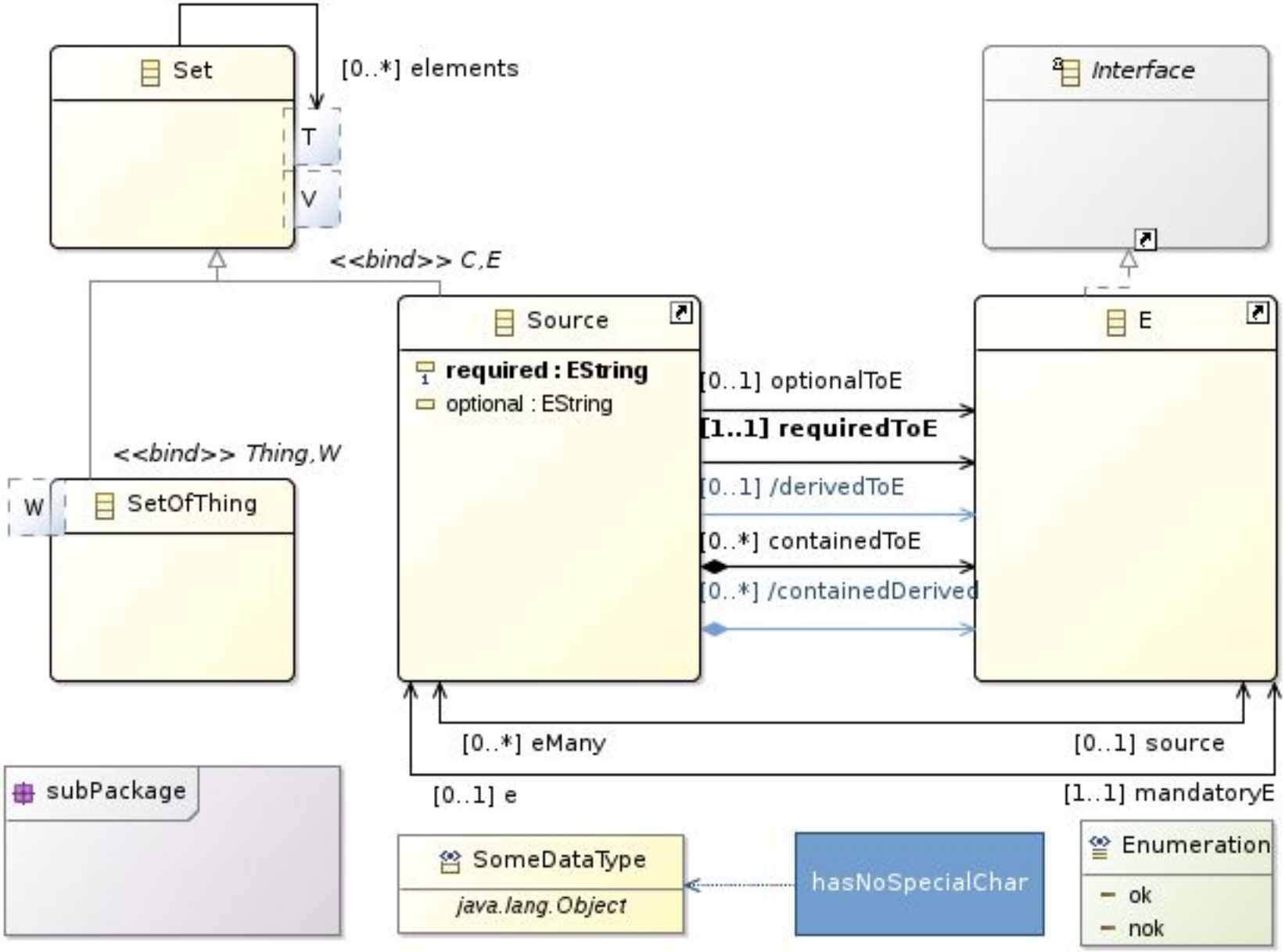
- Provides language constructs for specifying a DSL metamodel
  - mainly based on Object-Oriented constructs: *package*, *classes*, *properties* (*attribute* and *reference*), and (multiple) *inheritance*.
  - specificities: composition, opposite…
- Defined as a model, called *meta*metamodel:



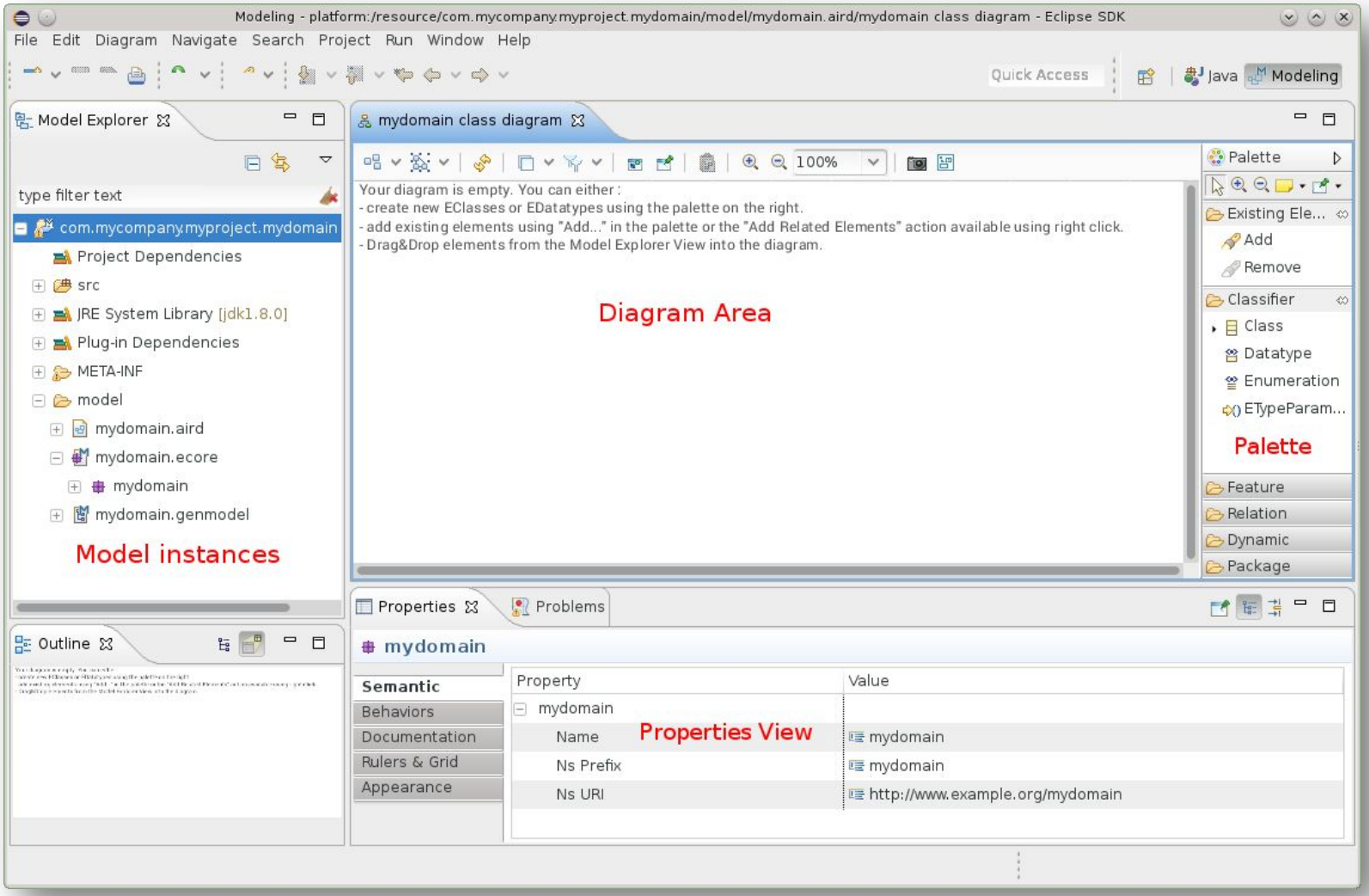
Ecore: a metamodel for metamodels

- Ecore is an implementation proposed by EMF, and aligned to EMOF
- Provides a language to build languages
- A metamodel is a model; and its metamodel is Ecore.
  - So a metamodel is an Ecore model!
- Ecore has concepts like:
  - Class – inheritance, have properties
  - Property – name, multiplicity, type
- Essentially this is a simplified version of class modeling in UML

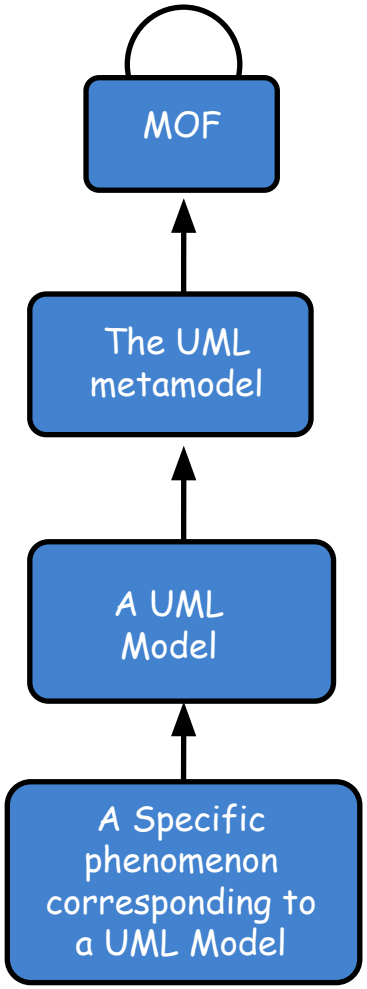

EModelElement
getEAnnotation(EString) : EAnnotation
eModelElement
0..1
eAnnotations
0..*
EStringToStringMapEntry
key : EString
value : EString
details
EAnnotation
source : EString
ENamedElement
name : EString
EFactory
create(EClass) : EObject
createFromString(EDataType,EString) : EJavaObject
convertToString(EDataType,EJavaObject) : EString
eFactoryInstance
1..1
ePackage
ETypedElement
ordered : EBoolean
unique : EBoolean
lowerBound : EInt
upperBound : EInt
many : EBoolean
required : EBoolean
eType
EClassifier
instanceClassName : EString
instanceClass : EJavaClass<?>
defaultValue : EJavaObject
isInstance(EJavaObject) : EBoolean
getClassifierID() : EInt
eClassifiers
EPackage
nsURI : EString
nsPrefix : EString
getEClassifier(EString) : EClassifier
eSuperPackage
eSubpackages
eExceptions
EOperation
EParameter
eOperation
eParameters
EClass
abstract : EBoolean
interface : EBoolean
isSuperTypeOf(EClass) : EBoolean
getFeatureCount() : EInt
getEStructuralFeature(EInt) : EStructuralFeature
getFeatureID(EStructuralFeature) : EInt
getEStructuralFeature(EString) : EStructuralFeature
EDataType
serializable : EBoolean
eAllSuperTypes
eSuperTypes
EEnumLiteral
value : EInt
instance : EEnumerator
literal : EString
eLiterals
eEnum
eContainingClass
eOperations
eAllOperations
eAllStructuralFeatures
eStructuralFeatures
eReferenceType
eAttributeType
EStructuralFeature
changeable : EBoolean
volatile : EBoolean
transient : EBoolean
defaultValueLiteral : EString
defaultValue : EJavaObject
unsettable : EBoolean
derived : EBoolean
getFeatureID() : EInt
getContainerClass() : EJavaClass
EReference
containment : EBoolean
container : EBoolean
resolveProxies : EBoolean
eReferences
eAllReferences
eAllContainments
eOpposite
eKeys
EAttribute
iD : EBoolean
eIDAttribute
eAttributes
eAllAttributes
EEnum
getEEnumLiteral(EString) : EEnumLiteral
getEEnumLiteral(EInt) : EEnumLiteral
getEEnumLiteralByLiteral(EString) : EEnumLiteral

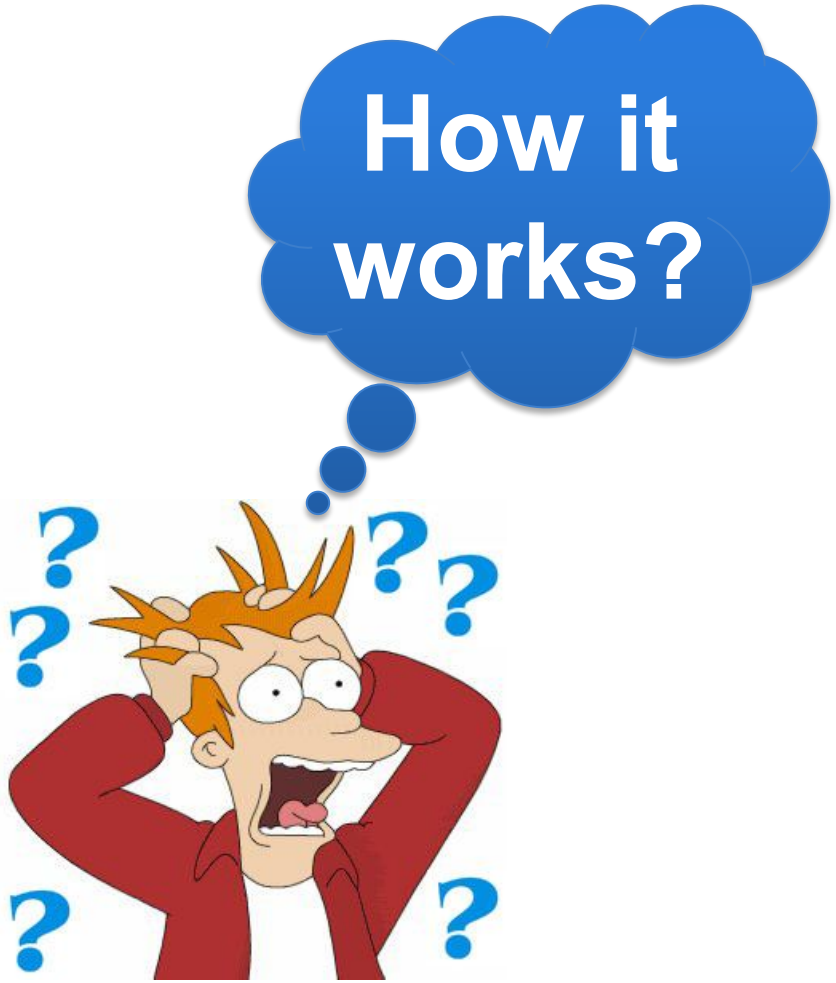

Set
[0..*] elements
T
V
Interface
<<bind>> C,E
Source
required : EString
optional : EString
E
[0..1] optionalToE
[1..1] requiredToE
[0..1] /derivedToE
[0..*] containedToE
[0..*] /containedDerived
<<bind>> Thing,W
W
SetOfThing
[0..*] eMany
[0..1] source
[0..1] e
[1..1] mandatoryE
subPackage
SomeDataType
java.lang.Object
hasNoSpecialChar
Enumeration
ok
nok

Ecore Tools


Modeling - platform:/resource/com.mycompany.myproject.mydomain/model/mydomain.aird/mydomain class diagram - Eclipse SDK
File Edit Diagram Navigate Search Project Run Window Help
Quick Access
Java
Modeling
Model Explorer
type filter text
com.mycompany.myproject.mydomain
Project Dependencies
src
JRE System Library [jdk1.8.0]
Plug-in Dependencies
META-INF
model
mydomain.aird
mydomain.ecore
mydomain
mydomain.genmodel
Model instances
mydomain class diagram
100%
Your diagram is empty. You can either :
- create new EClasses or EDatatypes using the palette on the right.
- add existing elements using "Add..." in the palette or the "Add Related Elements" action available using right click.
- Drag&Drop elements from the Model Explorer View into the diagram.
Diagram Area
Palette
Existing Ele...
Add
Remove
Classifier
Class
Datatype
Enumeration
ETypeParam...
Palette
Feature
Relation
Dynamic
Package
Outline
Properties
Problems
mydomain
Semantic
Behaviors
Documentation
Rulers & Grid
Appearance
Property
Value
mydomain
Name
mydomain
Ns Prefix
mydomain
Ns URI
http://www.example.org/mydomain
Properties View

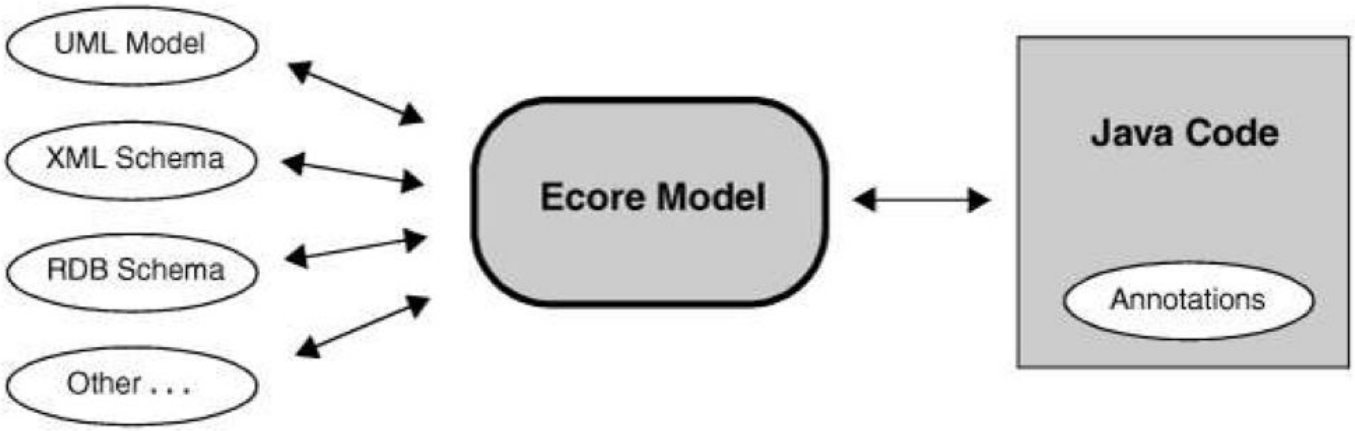
Implementation with Java





EMF

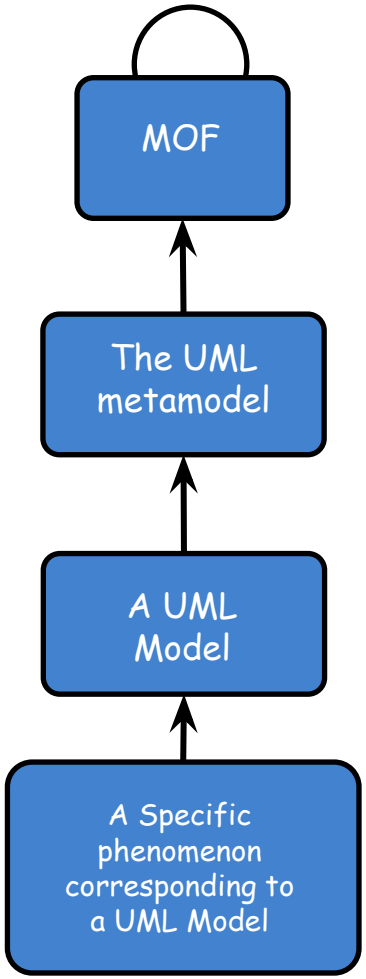
An Ecore model and its sources
(from *EMF: Eclipse Modeling Framework 2nd*)

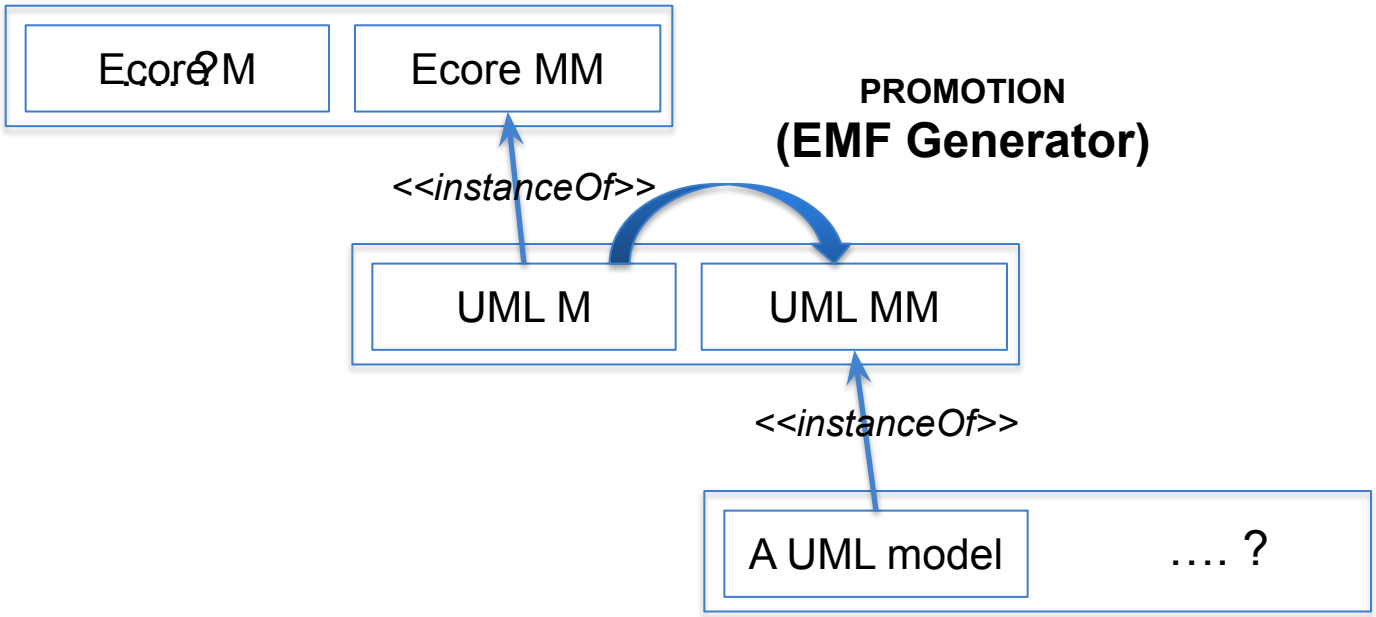


Implementation with Java

- EMF is a software (E)framework
- Model driven…, but implemented using a programming language!
- Reification MDE → Java:
  - Metamodels are represented with EClasses
  - Models are represented with EObjects

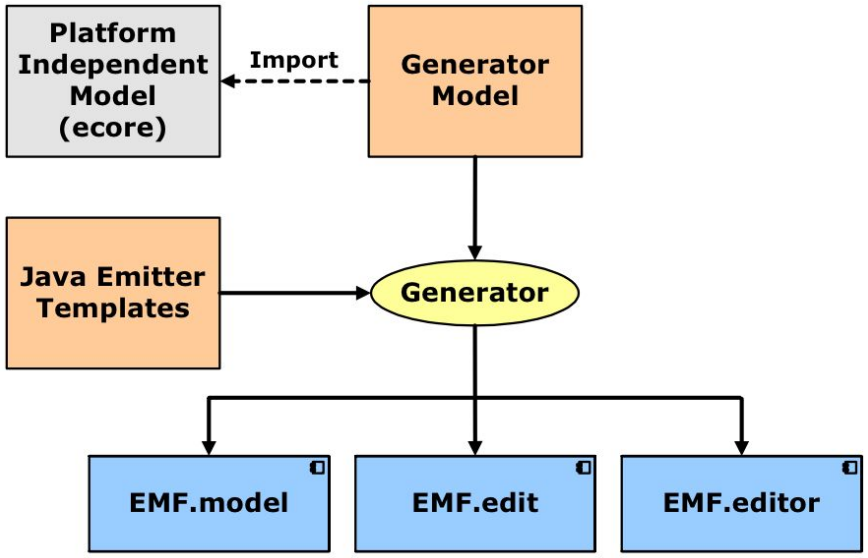
Implementation with Java





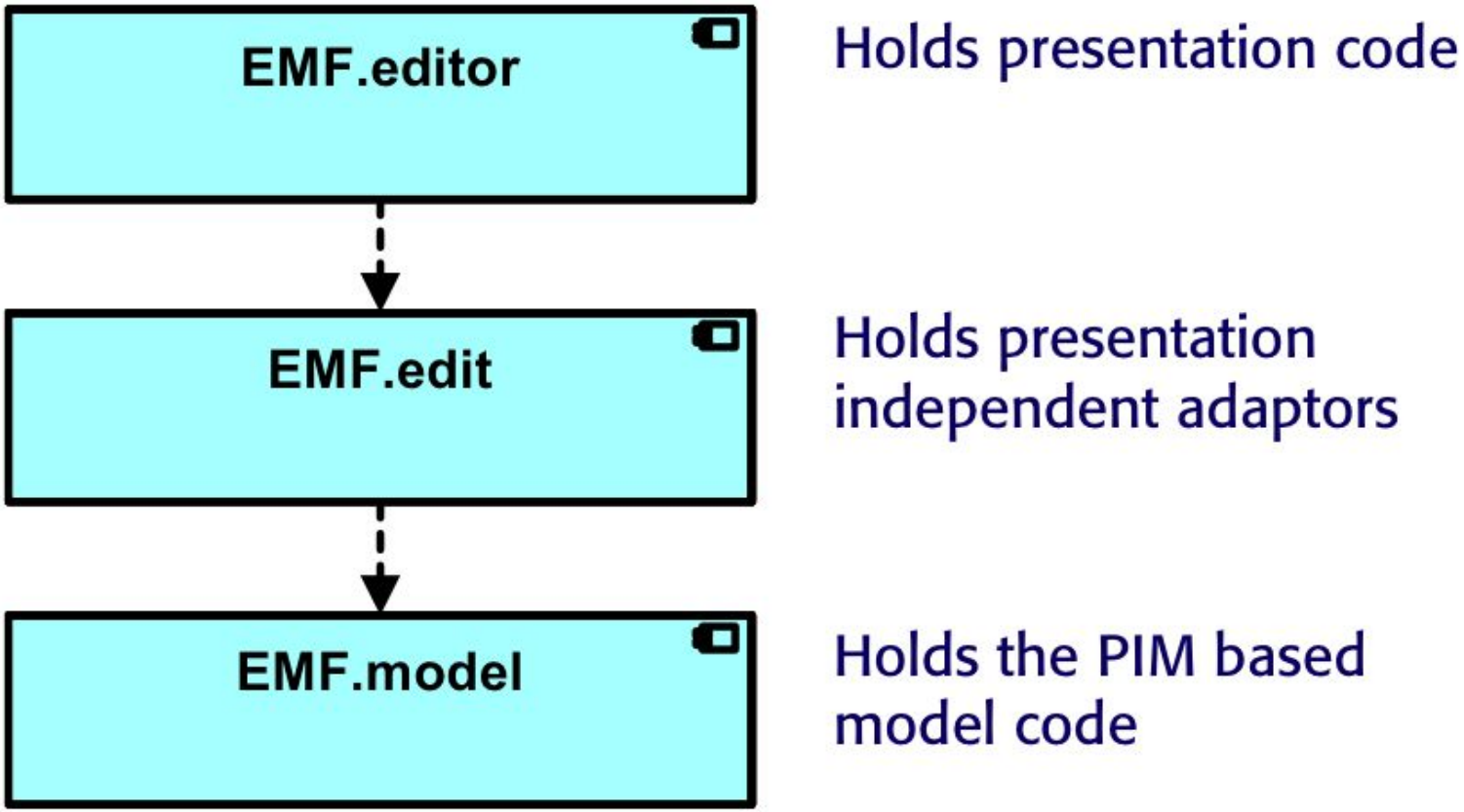
EMF Toolset

- The EMF Generator do not work on the .ecore
- EMF defines a .genmodel in parallel:
  - We can customize the code generator!
  - The IDE takes care of maintaining the consistency (or not!)



From "Mastering Eclipse Modeling Framework", V. Bacvanski and P. Graff

EMF Toolset



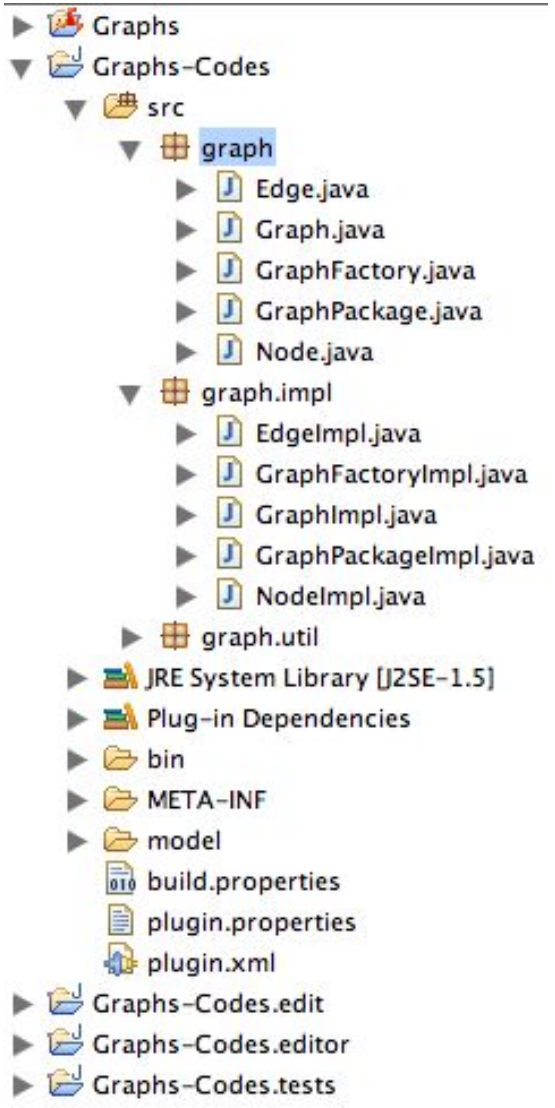
From "Mastering Eclipse Modeling Framework", V. Bacvanski and P. Graff

EMF Toolset

Actions available on the metamodel:

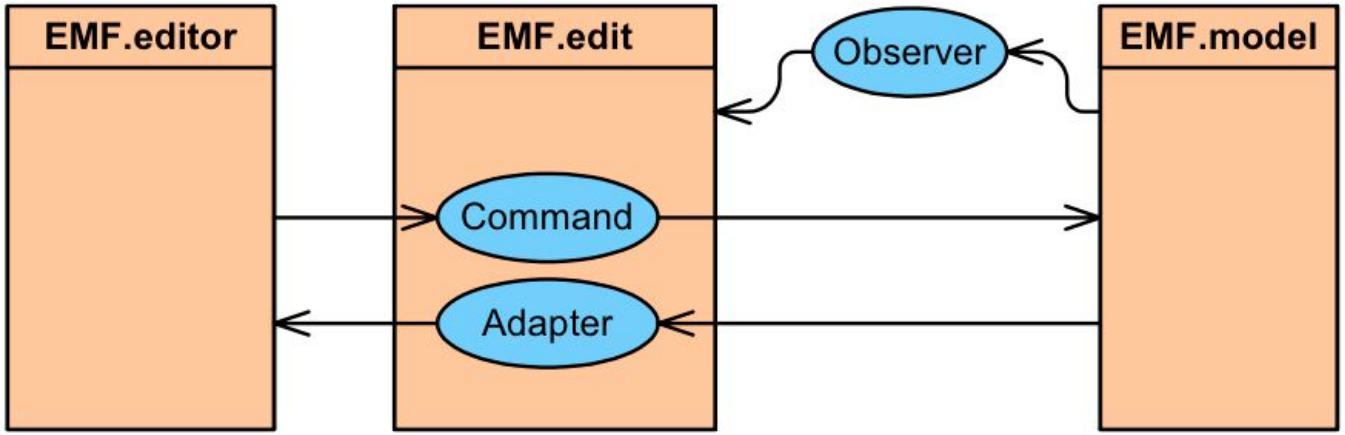
1. *Generate Model Code*: Java Classes corresponding to the metamodel
2. *Generate Edit Code*: Plugin supporting the edition
3. *Generate Editor Code*: Plugin for a tree based model editor
4. *Generate Test Code*: Plugin for unit testing

Actions available from the .genmodel, and into an EMF Project.



EMF: open the box

- The EMF.edit separates the GUI from the business model
- To understand the EMF.edit plug-in, it is essential to understand three basic design patterns
  - Observer pattern
  - Command pattern
  - Adapter pattern



From "Mastering Eclipse Modeling Framework", V. Bacvanski and P. Graff

EOperation Implementation

Localization of the methods in the generated code

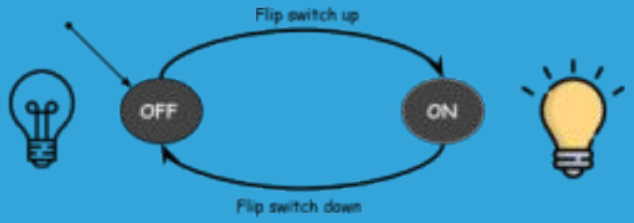
1. In the subpackage `graph.impl`
2. In the class `GraphImpl`
3. Scattered in the code automatically generated by EMF…

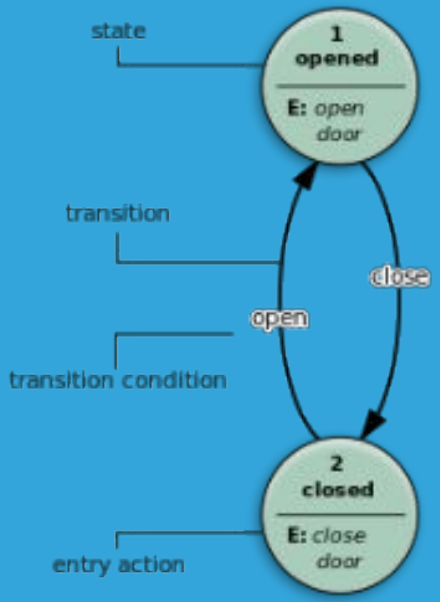
```
/**
 * @generated NOT
 */
public int order () {
 return this.getEdges().size();
}
```

Do not forget to mark (`@generated NOT`) to prevent crushing!

COURSE ORGANIZ

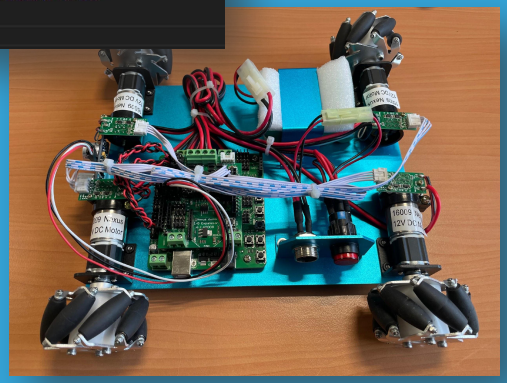
Part 1: define a metamodel to help you developing state machines...





Part 1: define a metamodel for robots…

```
let bool entry () {
    setSpeed(30)
    var int count = 0
    var int eval = 1
    loop count < 5
    {
        count = count + 1
        square()
    }
}

let bool square(){
    Forward 200
    Clock 90
    Forward 200
    Clock 90
    Forward 200
    Clock 90
    Forward 200
    Clock 90
    return true
}

```



KEEP CALM AND DO IT YOURSELF

Xtext, a popular, easy-to-use model-based tool for developing textual DSLs

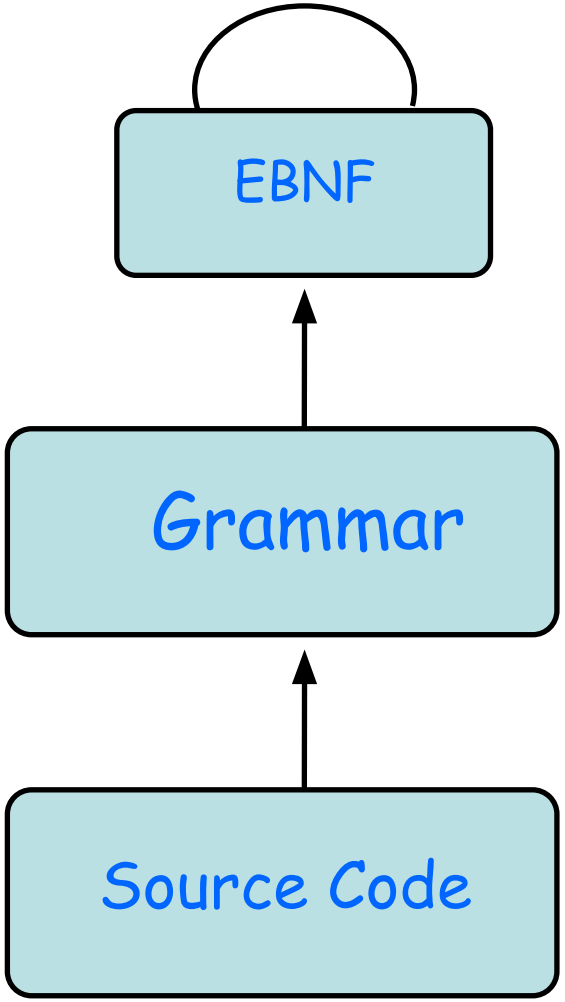
Your textual DSL in 5’ (incl. editors, serializers)

Foundations

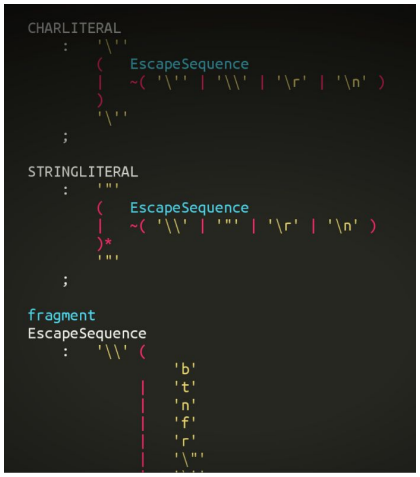
M^3

M^2

M^1



Java Grammar

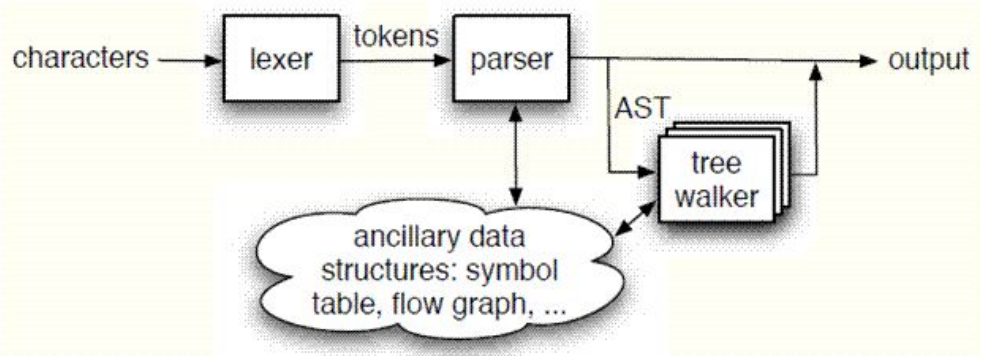


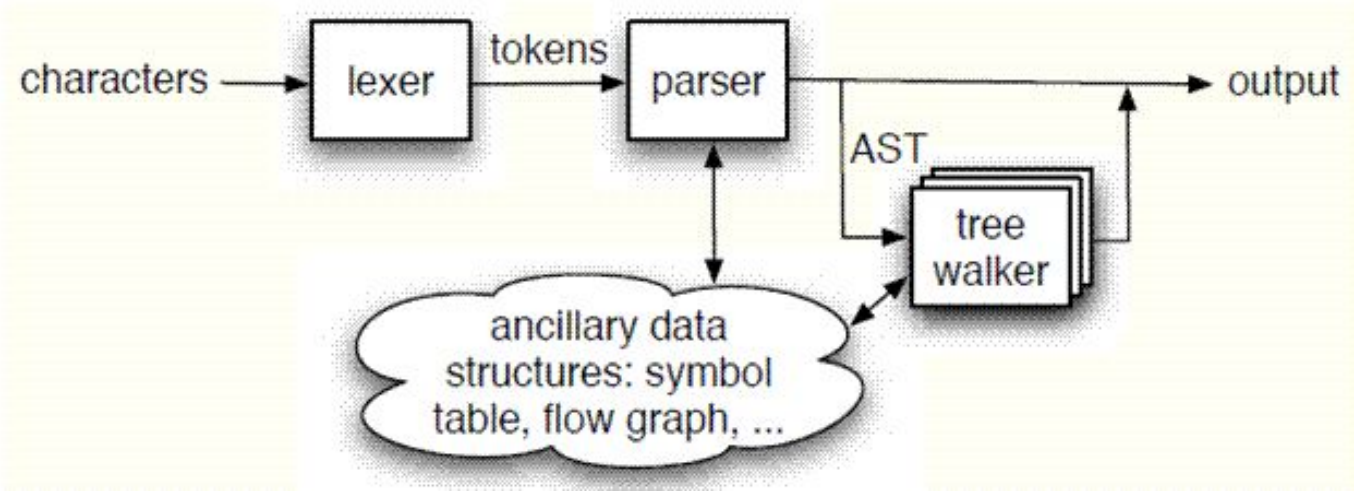
Java Program

```java
/*********************************/

public class HelloWorld {

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

}
```

Compilation Process

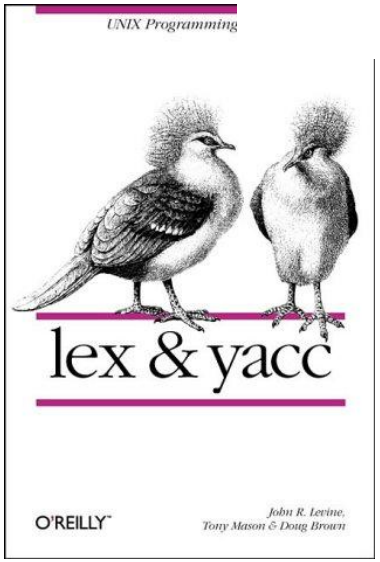
- Source code
  - Concrete syntax used for specifying a program
  - Conformant to a grammar
- Lexical analysis
  - Converting a sequence of characters into a sequence of **tokens**
- Parsing (Syntactical analysis)
  - Abstract Syntax Tree (AST)






```
CHARLITERAL
    :   '\''
        (   EscapeSequence
        |   ~( '\'' | '\\' | '\r' | '\n' )
        )
        '\''
    ;

STRINGLITERAL
    :   '"'
        (   EscapeSequence
        |   ~( '\\' | '"' | '\r' | '\n' )
        )*
        '"'
    ;

fragment
EscapeSequence
    :   '\\' (
                'b'
            |   't'
            |   'n'
            |   'f'
            |   'r'
            |   '\"'
            |   '\''
```

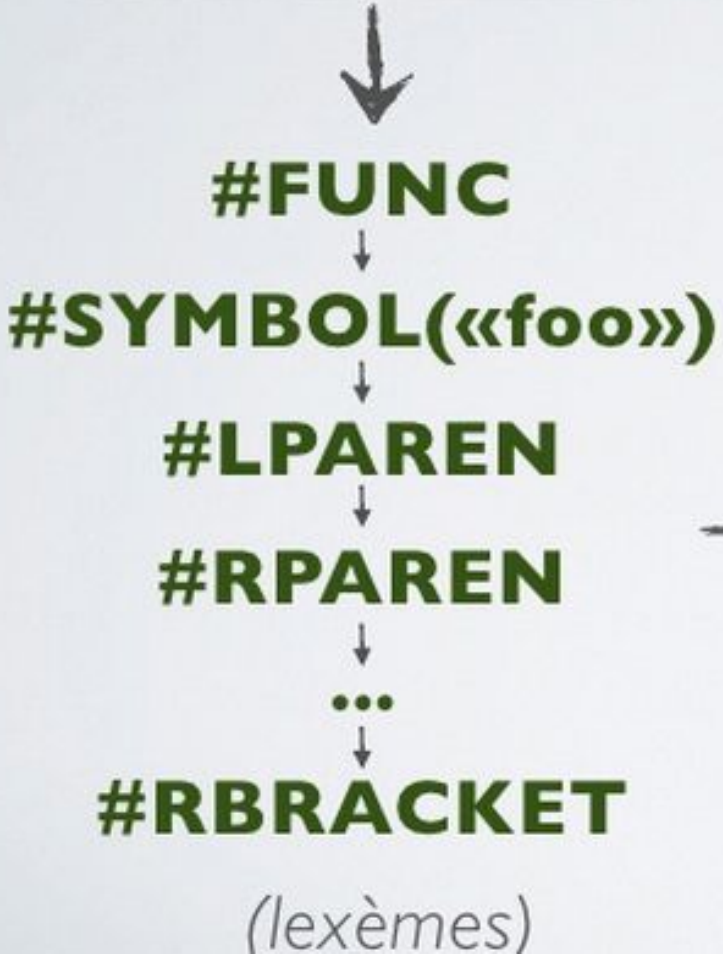
```
classOrInterfaceDeclaration
    :   classDeclaration
    |   interfaceDeclaration
    ;


modifiers
    :
    (   annotation
    |   PUBLIC
    |   PROTECTED
    |   PRIVATE
    |   STATIC
    |   ABSTRACT
    |   FINAL
    |   NATIVE
    |   SYNCHRONIZED
    |   TRANSIENT
    |   VOLATILE
    |   STRICTFP
    )*
    ;


variableModifiers
    :   (   FINAL
        |   annotation
        )*
    ;


classDeclaration
    :   normalClassDeclaration
    |   enumDeclaration
```

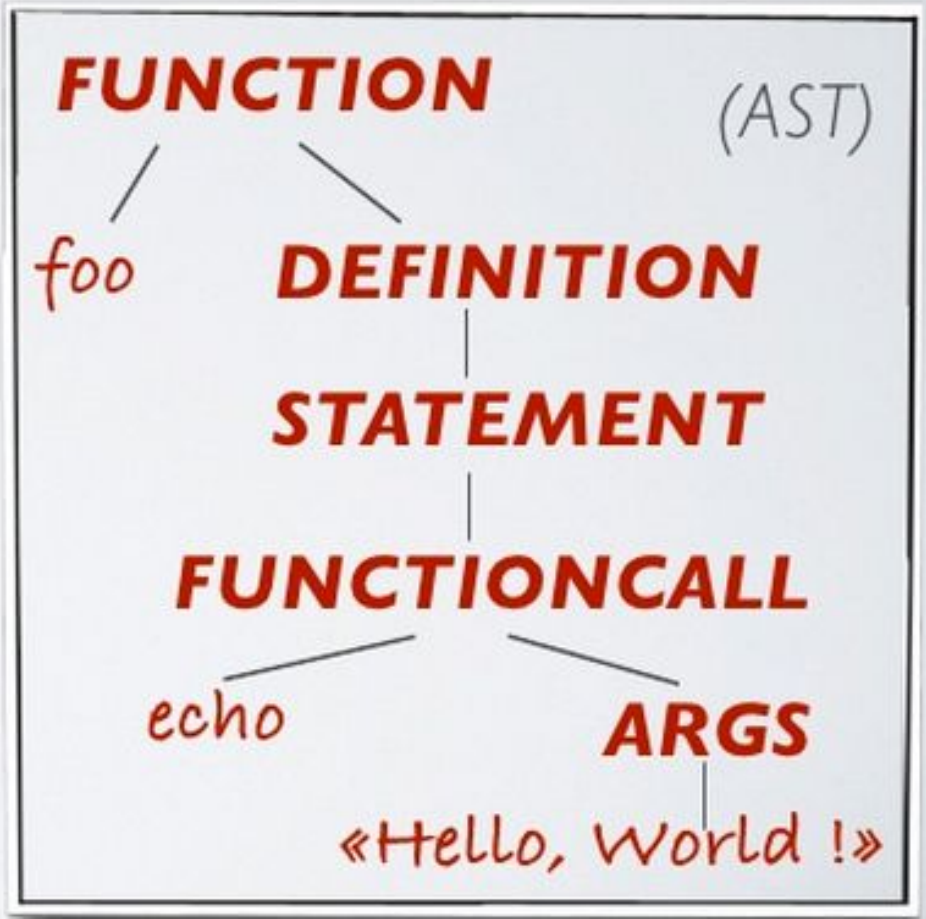


EXEMPLE

```
function foo() {
  echo «Hello, World !»;
}
```

(Syntaxe concrète)



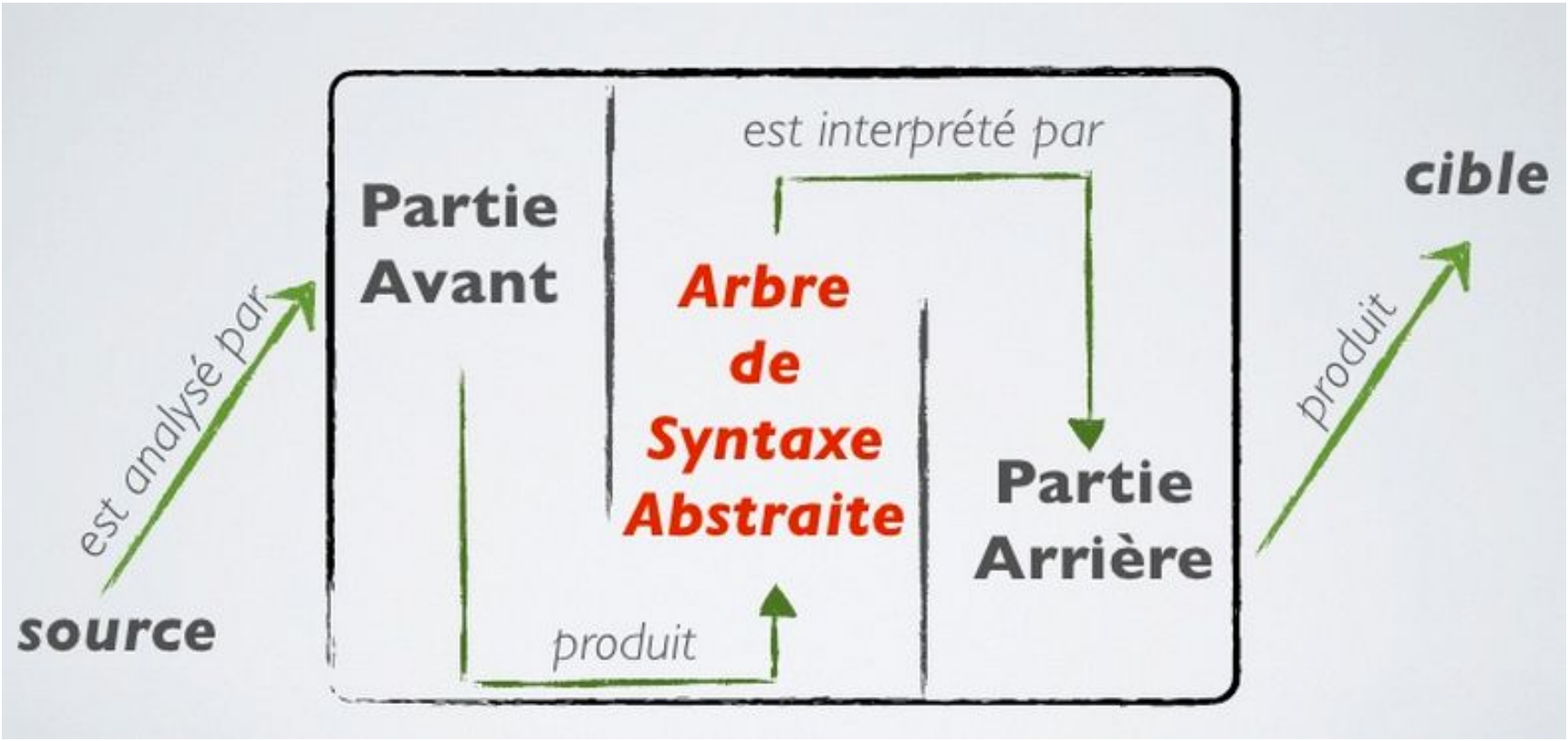


```scala
class StringInterp {
  val int = 42
  val dbl = Math.PI
  val str = "My hovercraft is full of eels"

  println(s"String: $str Double: $dbl Int: $int Int Expr: ${int * 1.0}")
}
```

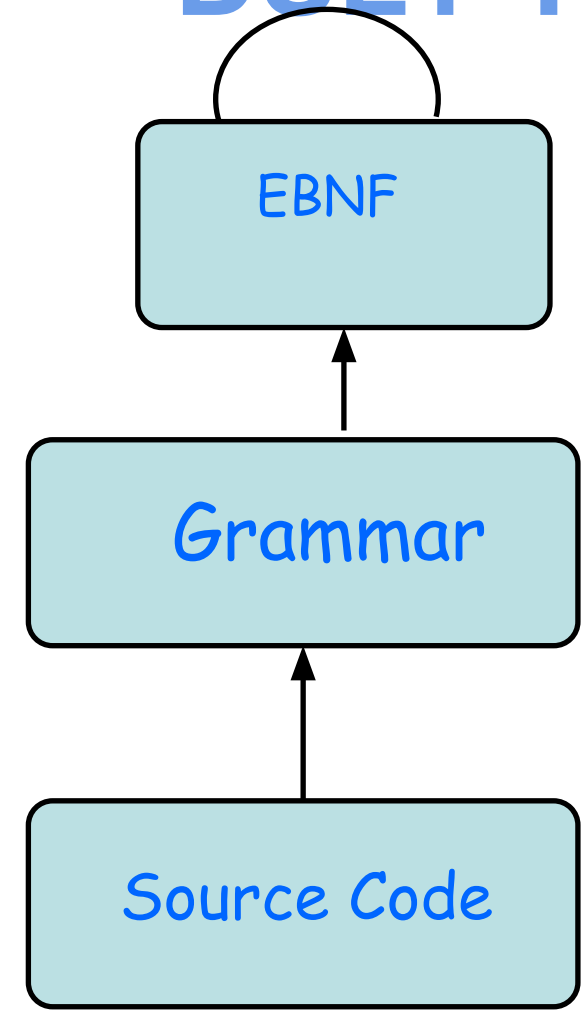
Scala AST (example)

```
Block(
  List(
    ClassDef(Modifiers(), TypeName("StringInterp"), List(), Template(
      List(Ident(TypeName("AnyRef"))), noSelfType, List(DefDef(Modifiers(), termNames.CONSTRUCTOR,
        List(),
        List(List()),
        TypeTree(), Block(List(Apply(Select(Super(This(typeNames.EMPTY), typeNames.EMPTY),
        termNames.CONSTRUCTOR), List())), Literal(Constant(())))), ValDef(Modifiers(), TermName("int"),
        TypeTree(), Literal(Constant(42))), ValDef(Modifiers(), TermName("dbl"), TypeTree(),
        Literal(Constant(3.141592653589793))), ValDef(Modifiers(), TermName("str"), TypeTree(),
        Literal(Constant("My hovercraft is full of eels"))), Apply(Select(Ident(scala.Predef),
        TermName("println")), List(Apply(Select(Apply(Select(Ident(scala.StringContext), TermName("apply")),
        List(Literal(Constant("String: ")), Literal(Constant(" Double: ")), Literal(Constant(" Int: ")),
          Literal(Constant(" Int Expr: ")), Literal(Constant("")))), TermName("s")),
        List(Select(This(TypeName("StringInterp")), TermName("str")), Select(This(TypeName("StringInterp")),
          TermName("dbl")), Select(This(TypeName("StringInterp")), TermName("int")),
          Apply(Select(Select(This(TypeName("StringInterp")), TermName("int")), TermName("$times")),
            List(Literal(Constant(1.0))))))))
    ))), Literal(Constant(()))))
```

Compilation (en français)

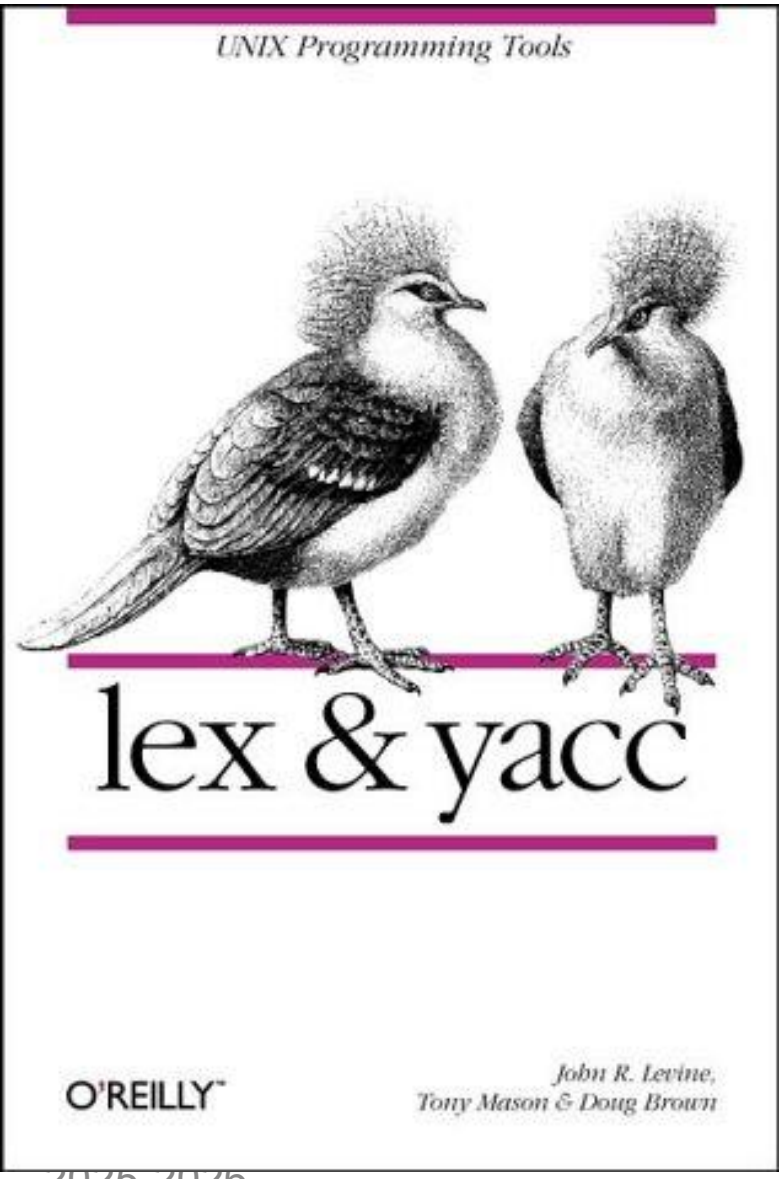


DSL? The same!

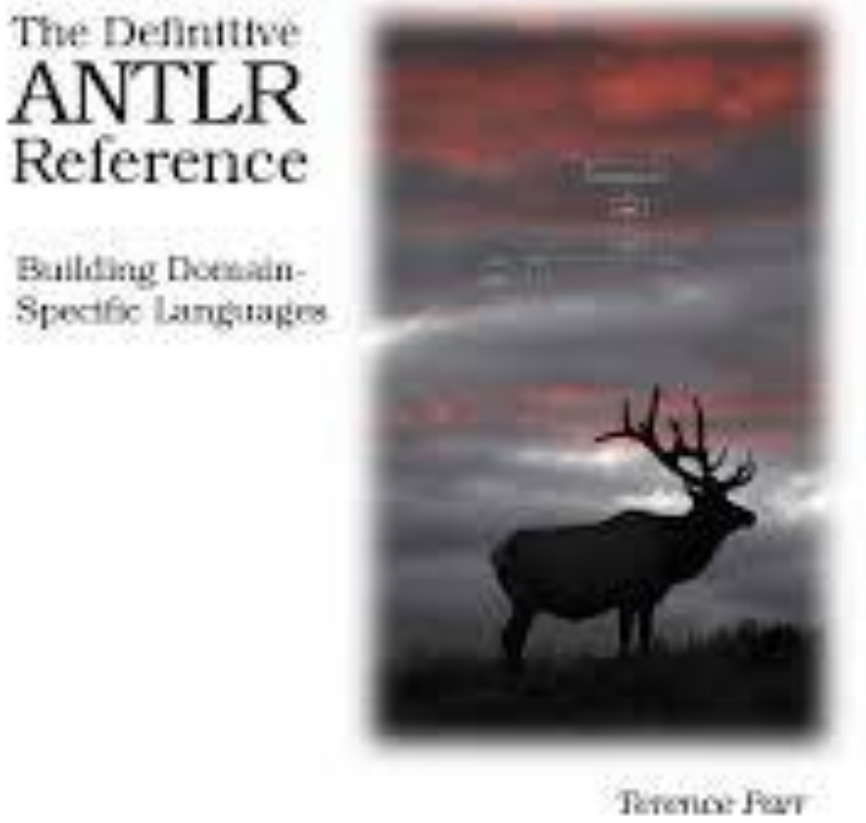


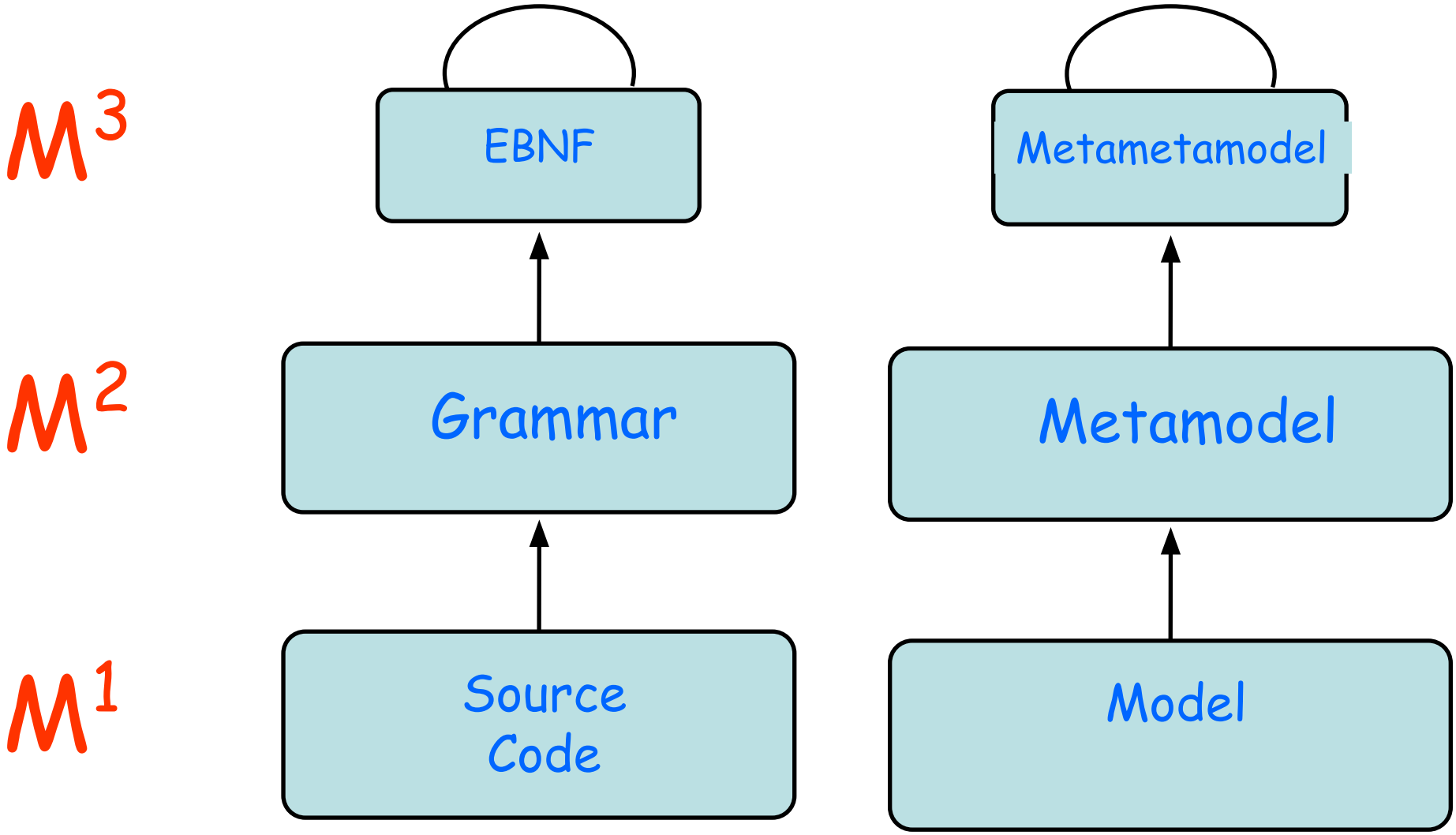
DSL
Grammar

DSL
specification/program

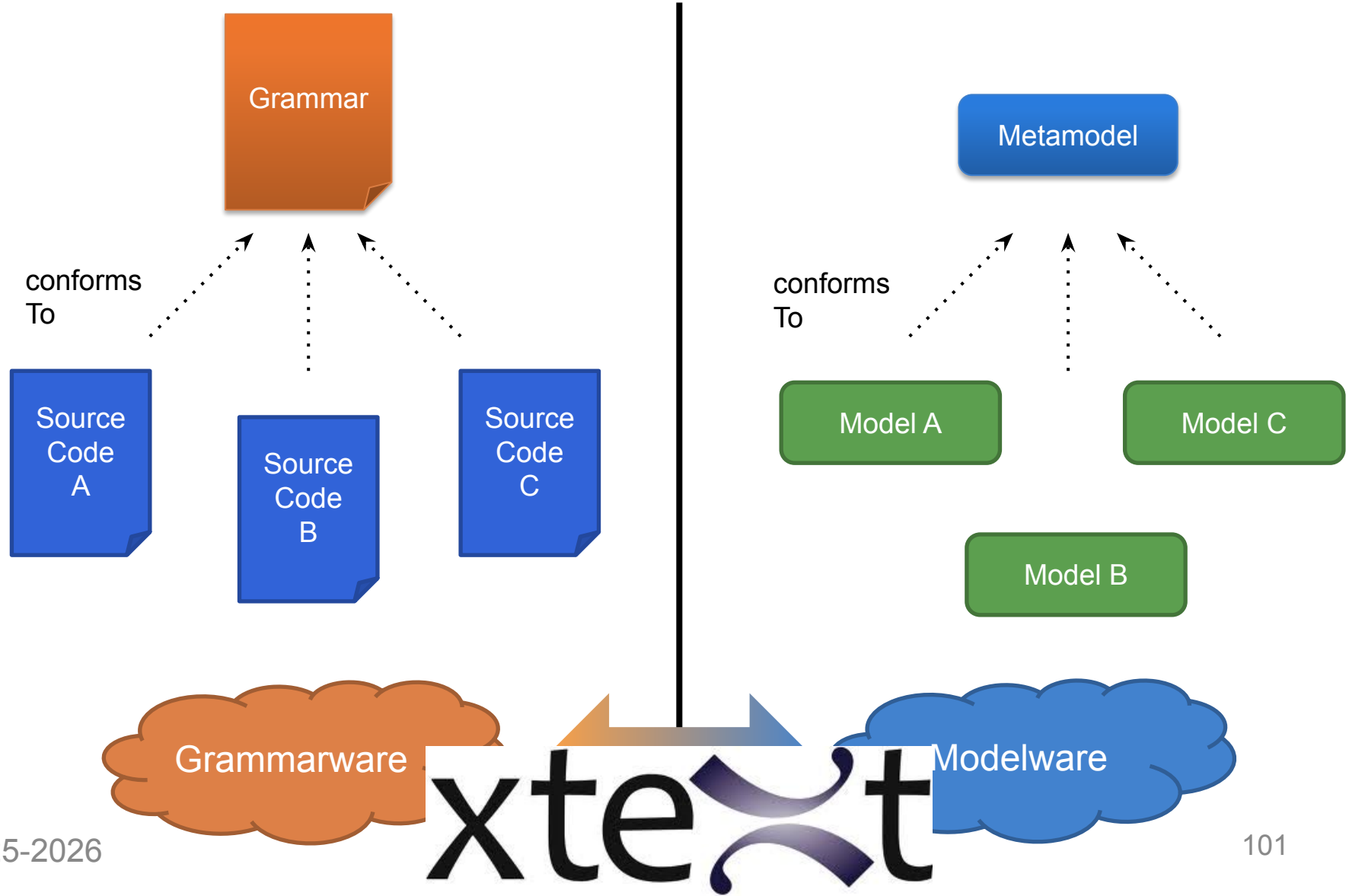

UNIX Programming Tools
lex & yacc
O'REILLY
John R. Levine,
Tony Mason & Doug Brown


The Pragmatic Programmers


The Definitive
ANTLR
Reference
Building Domain-
Specific Languages
Terence Parr


M3
M2
M1
EBNF
Grammar
Source Code
Metametamodel
Metamodel
Model

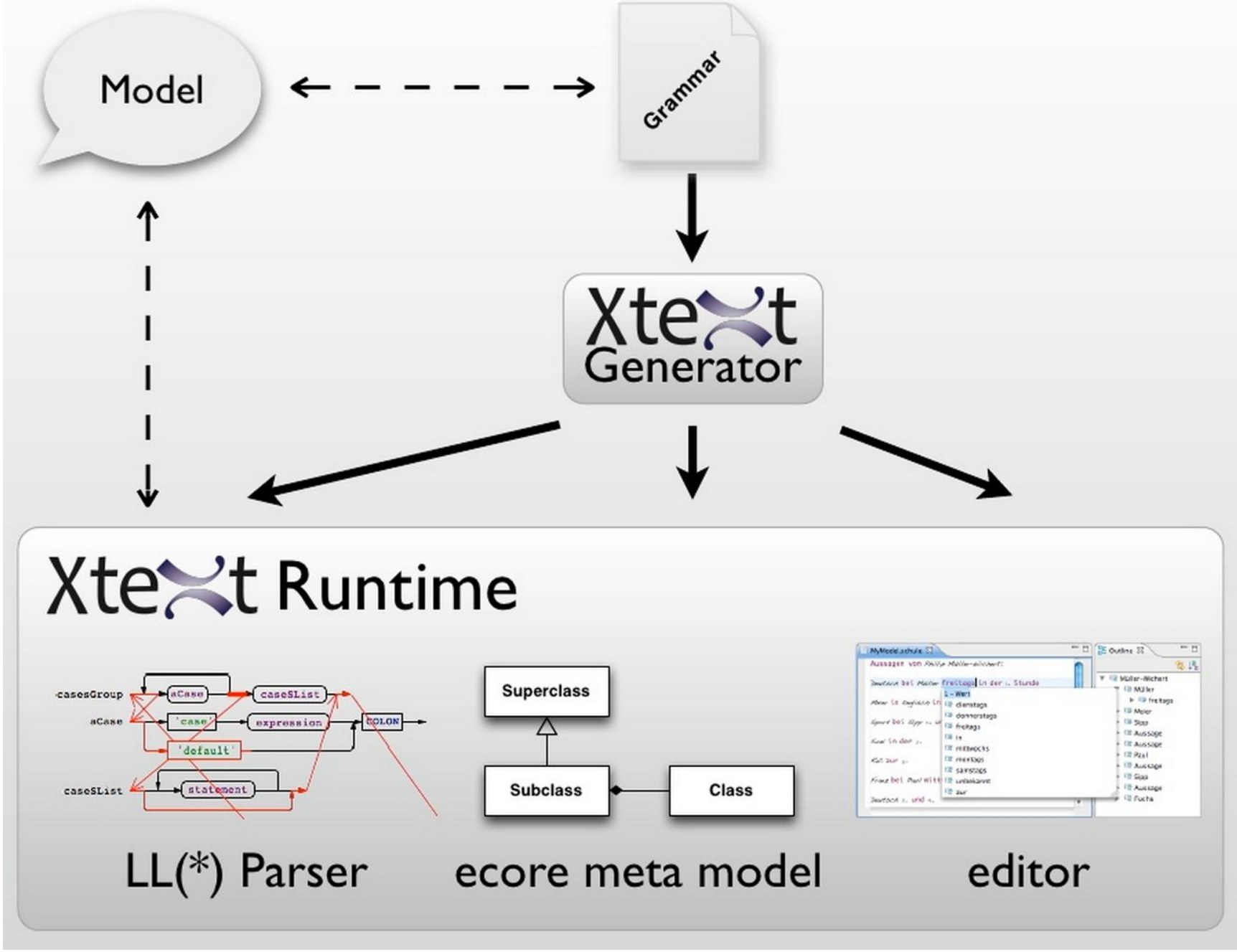
Language and MDE



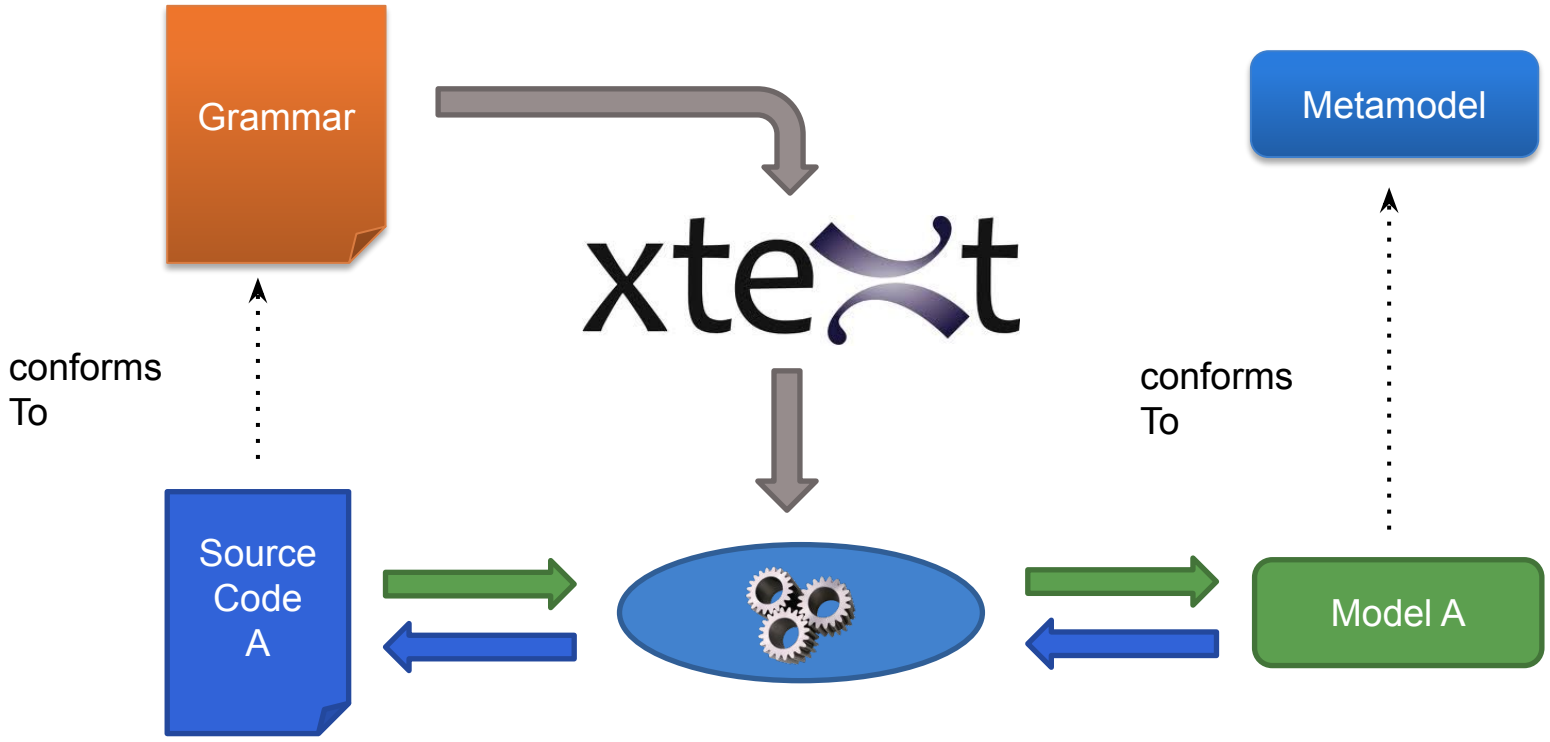
Give me a **grammar**,

I'll give you (for free)

* a comprehensive editor (auto-completion, syntax highlighting, etc.) in Eclipse
* an Ecore metamodel and facilities to load/serialize/visit conformant models (Java ecosystem)
* extension to override/extend « default » facilities (e.g., checker)


Model
Grammar
Xtext Generator
Xtext Runtime
casesGroup
aCase
caseSList
'case'
expression
COLON
'default'
statement
Superclass
Subclass
Class
LL(*) Parser
ecore meta model
editor

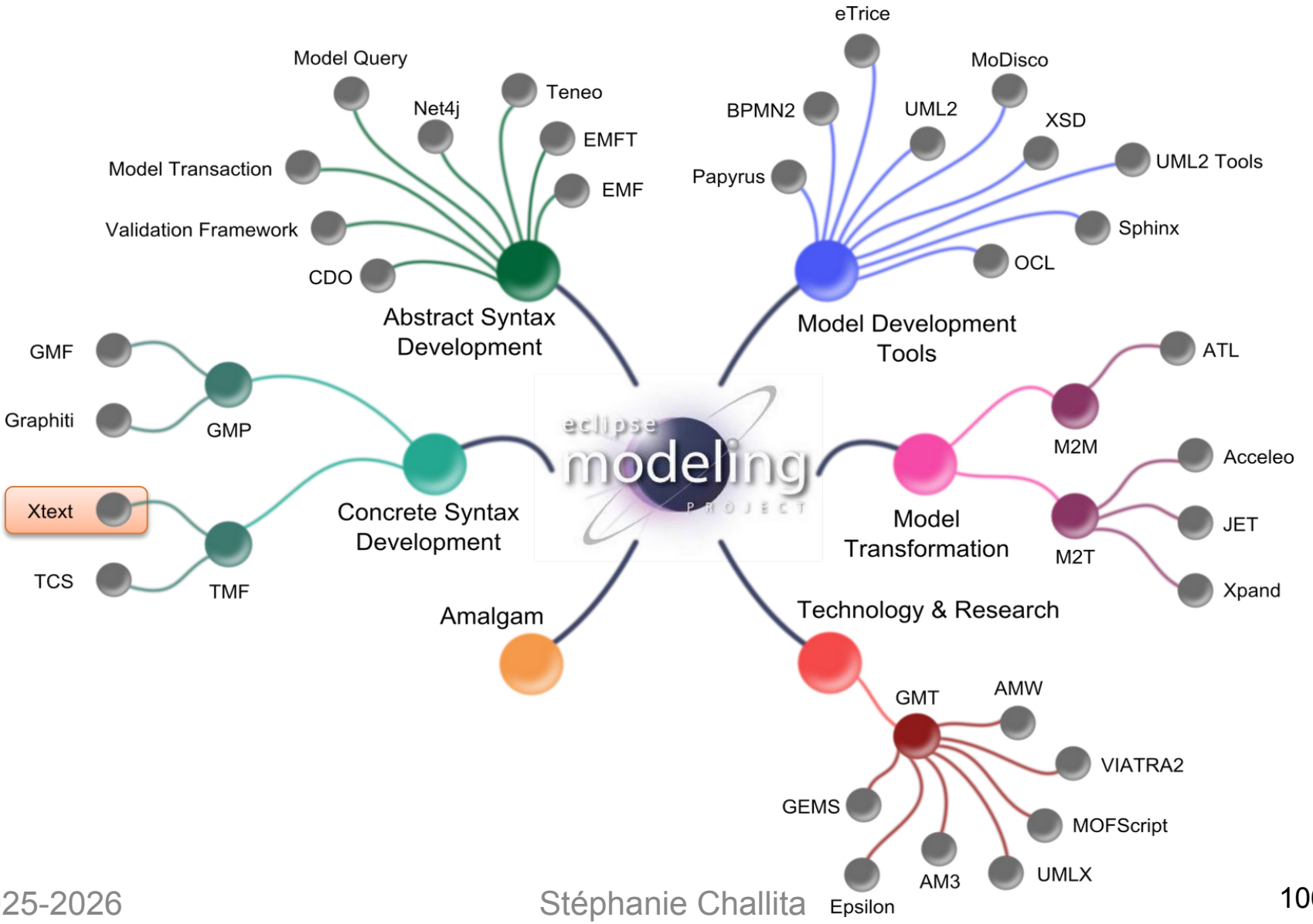
Xtext, Grammar, Metamodel



Xtext Project

- Eclipse Project
  - Part of Eclipse Modeling
  - Part of Open Architecture Ware
- Model-driven development of Textual DSLs
- Part of a family of languages
  - **Xtext**
  - Xtend
  - Xbase
  - Xpand
  - Xcore

Eclipse Modeling Project



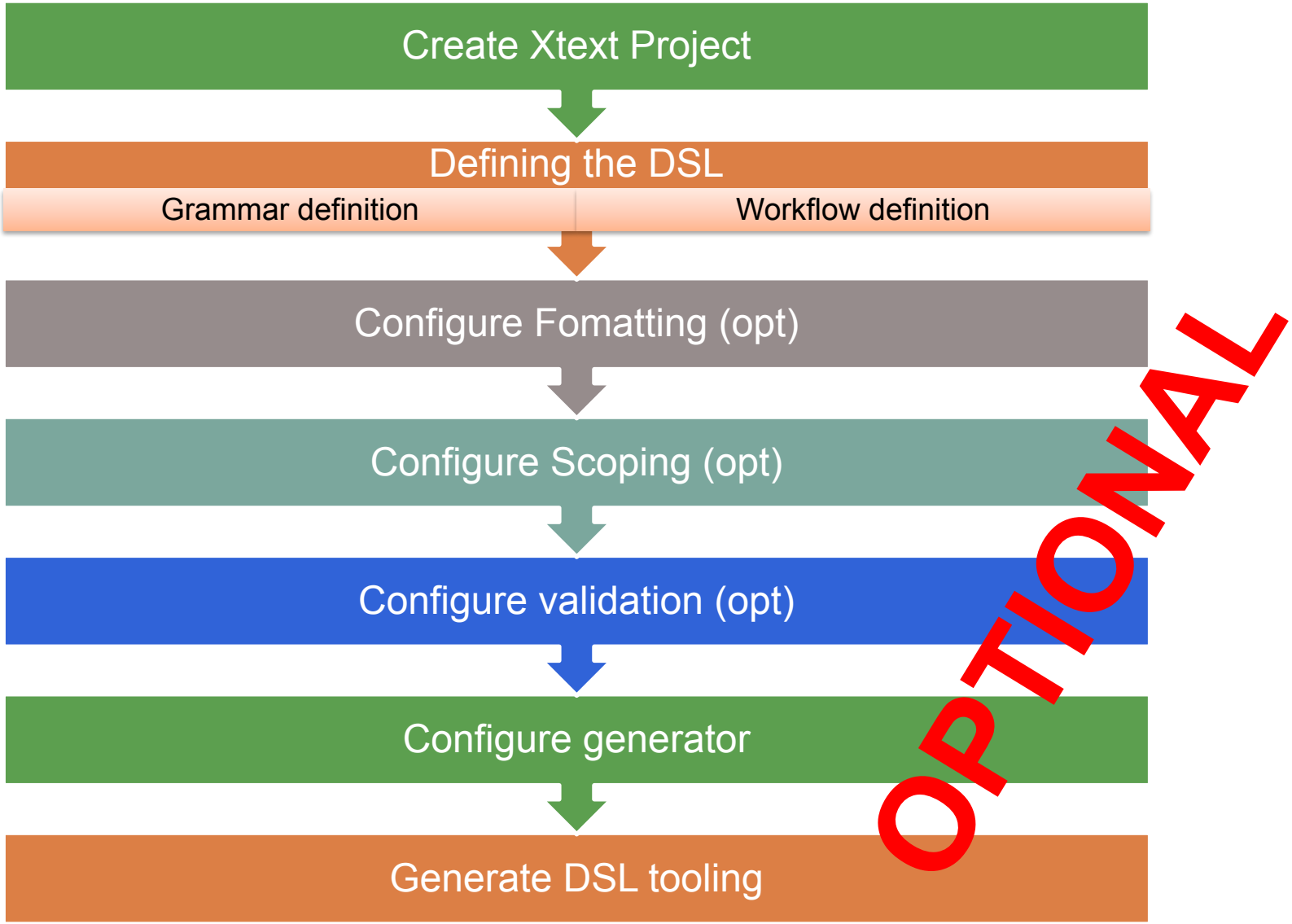
The Grammar Language of Xtext

- Corner-stone of Xtext
- A… DSL to define textual languages
  - Describe the concrete syntax
  - Specify the mapping between concrete syntax and domain model
- From the grammar, it is generated:
  - The domain model
  - The parser
  - The tooling

Main Advantages

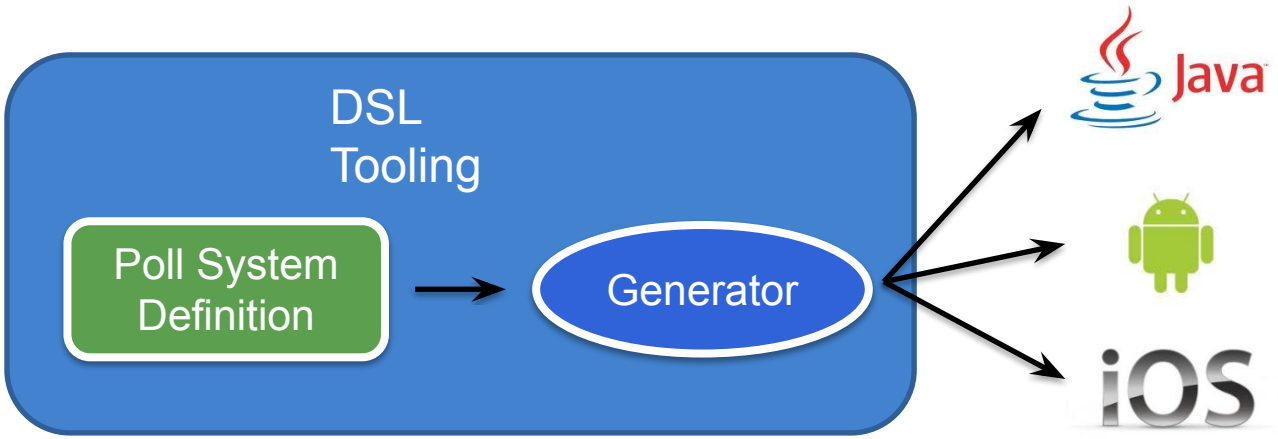
- Consistent look and feel
- Textual DSLs are a resource in Eclipse
- Open editors can be extended
- Complete framework to develop DSLs
- Easy to connect to any Java-based language

Development Process



Motivating Scenario

- Poll System application
  - Define a Poll with the corresponding questions
  - Each question has a text and a set of options
  - Each option has a text
- Generate the application in different platforms



Motivating Scenario (2)

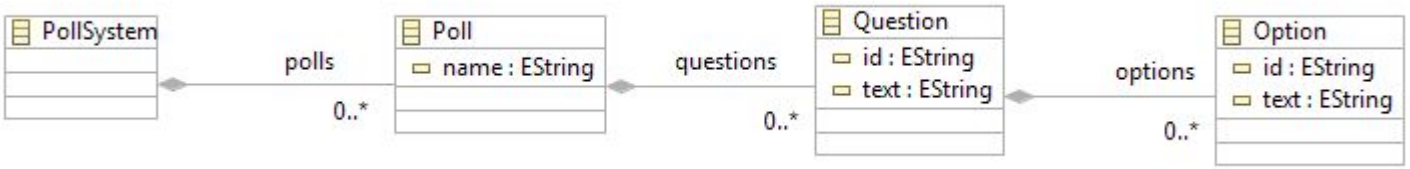
DSL
Tooling

```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```

Grammar Definition

Grammar definition →

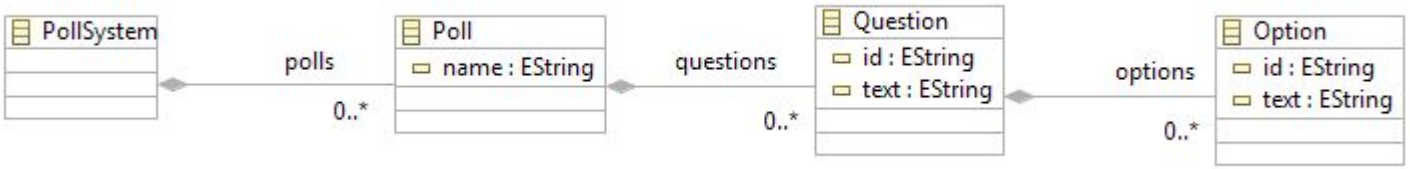
```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```



Grammar Definition

Grammar reuse

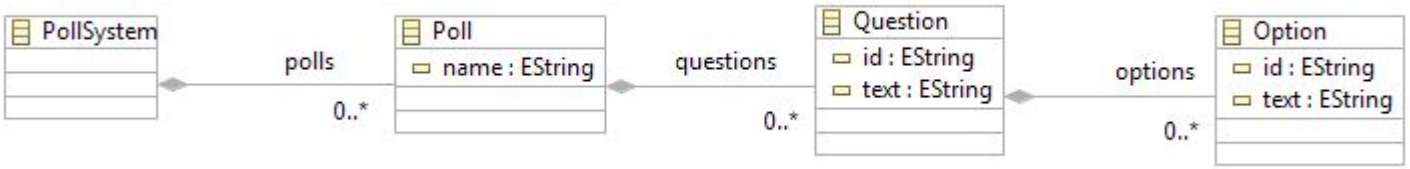
```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```



Grammar Definition

Derived metamodel →

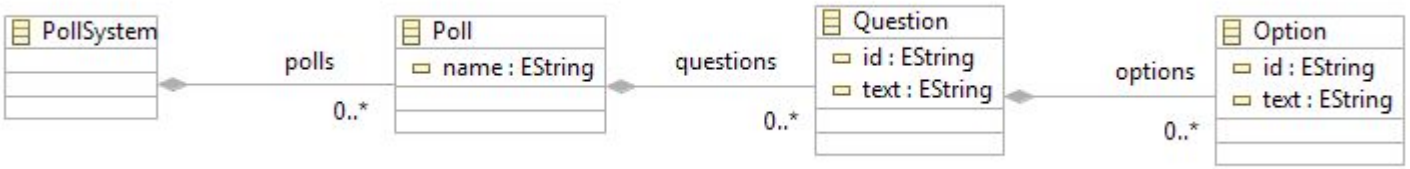
```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```



Grammar Definition

Parser Rules

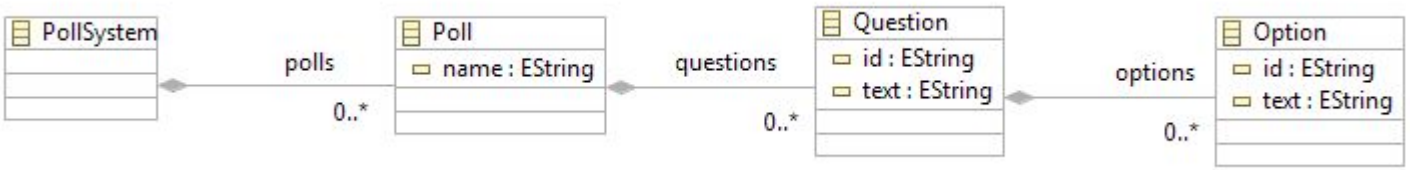
```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```

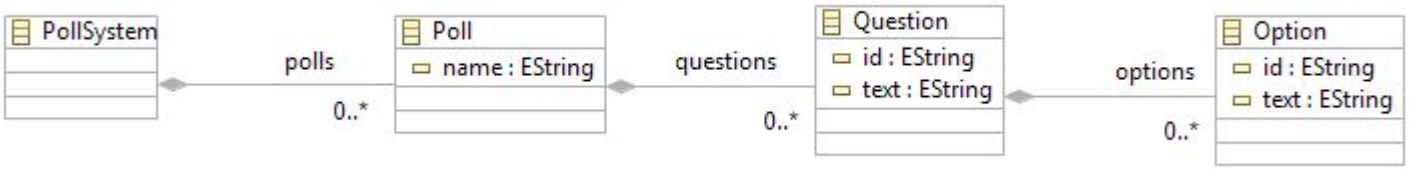
Keywords



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```

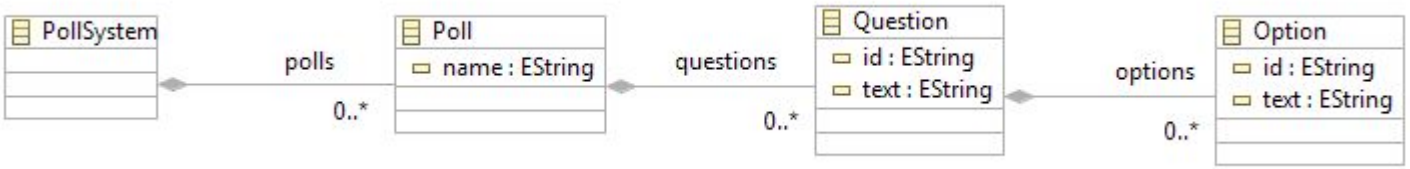
Simple assignment



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```

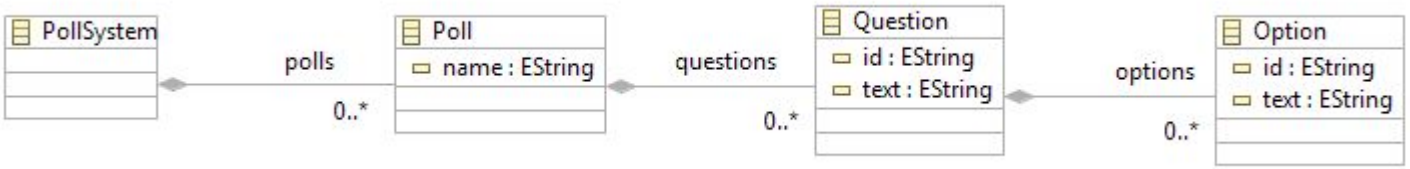
Multivalue assignment



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```

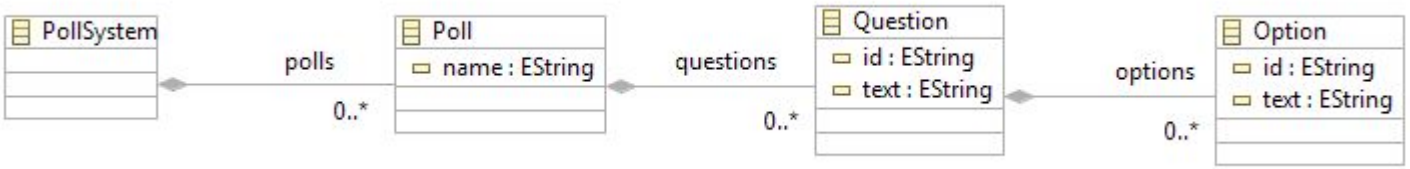
?= Boolean assignment



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```

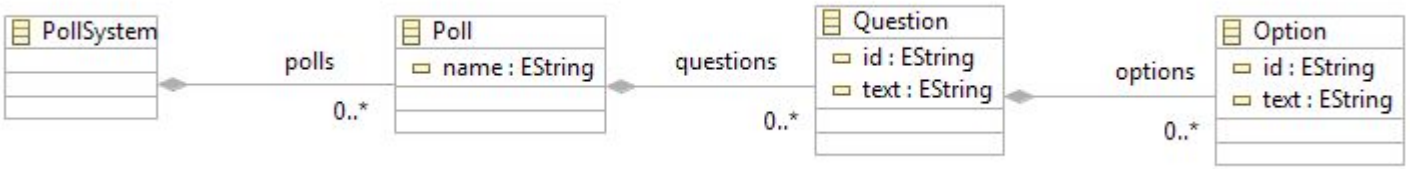
Cardinality (others: * ?)



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```

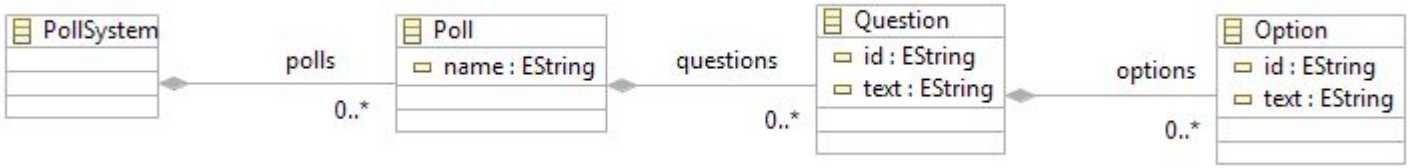
Containment



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```

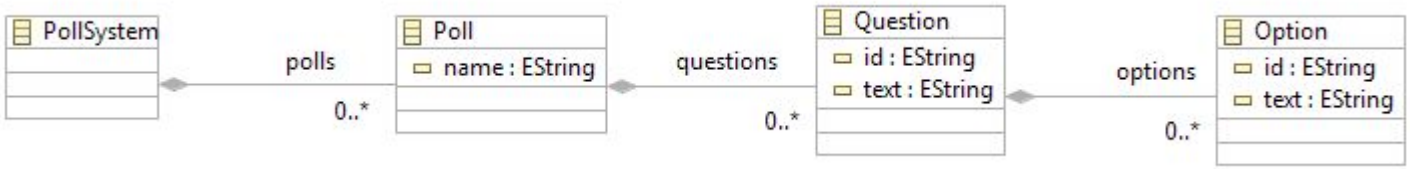
```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```

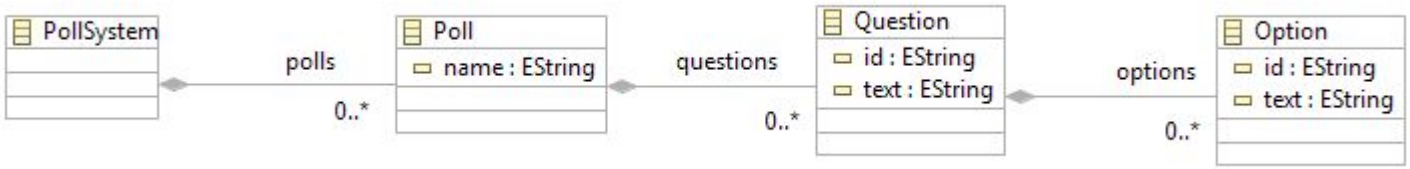
```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```

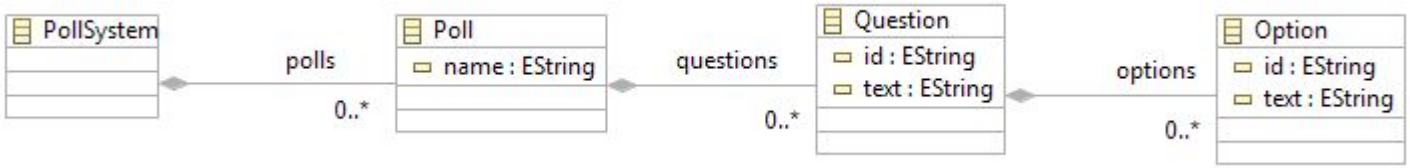
```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```



Grammar Definition

```
grammar fr.miage.xtext.Poll with org.eclipse.xtext.common.Terminals

generate poll "http://www.miage.fr/xtext/Poll"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}' ;

Poll:
    'Poll' name=ID '{' questions+=Question+'}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;
```

```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```

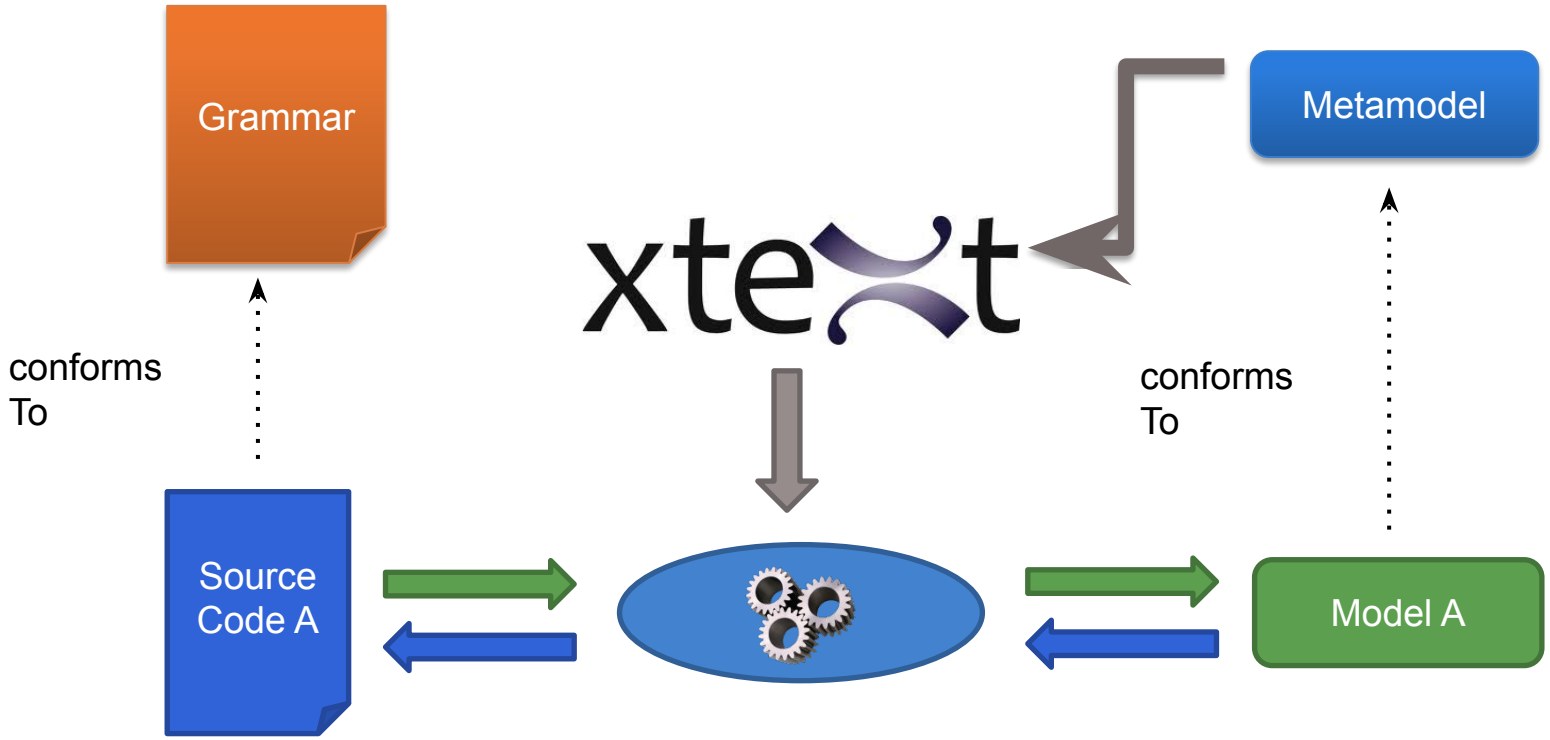


From Metamodel

To

Grammar (other way)

From Metamodel to Grammar



Give me a **metamodel**,

I'll give you (for free)
 * a comprehensive editor (auto-completion, syntax highlitening, etc.) in Eclipse
 * a grammar and facilities to load/serialize/visit conformant models (Java ecosystem)
 * extension to override/extend « default » facilities (e.g., checker)

Give me a **metamodel**,

The grammar can be « weird » (i.e., not as concise and as comprehensible than if you made it manually)

[Same observation actually applies to the other side: generated metamodels (from grammar) can be weird as well, but you have at least some control in Xtext-based grammar]
[We will experiment in the lab sessions]

Part 2: define a textual syntax (with Xtext) for your statemachine metamodel…

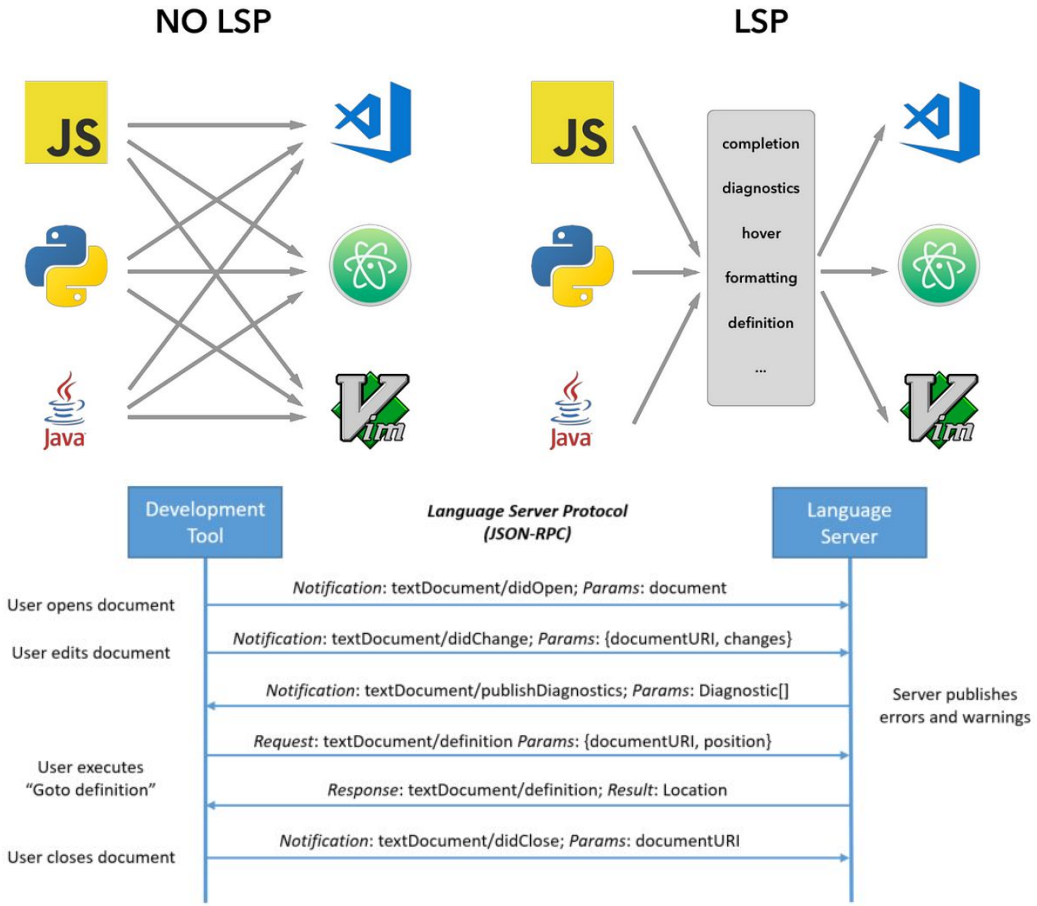
```
fsm door
        state opened entry "open
        door"
        state init closed entry "close
        door"
        transition open closed ->
        opened [on]
        transition close opened ->
        closed [off]
```

Langium, a new brand, web-based, LSP-centric, easy-to-use TS tool for developing textual DSLs

Your textual DSL in 5’ (incl. editors, serializers)

The Language Server Protocol (LSP)

*"A **Language Server** is meant to provide the language-specific smarts and communicate with development tools over a protocol that enables inter-process communication"*



https://microsoft.github.io/language-server-protocol/

The Language Server Protocol (LSP)

- Defines a bidirectional protocol between a tool (Client) and a “language smartness provider” (Language Server)
- Open de-facto Standard from Microsoft (Team around Erich Gamma)
  - https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/

Now complemented with the ***Language Server Index Format (LSIF)*** to define a standard format for language servers or other programming tools to dump their knowledge about a workspace

https://microsoft.github.io/language-server-protocol/

The Language Server Protocol (LSP)

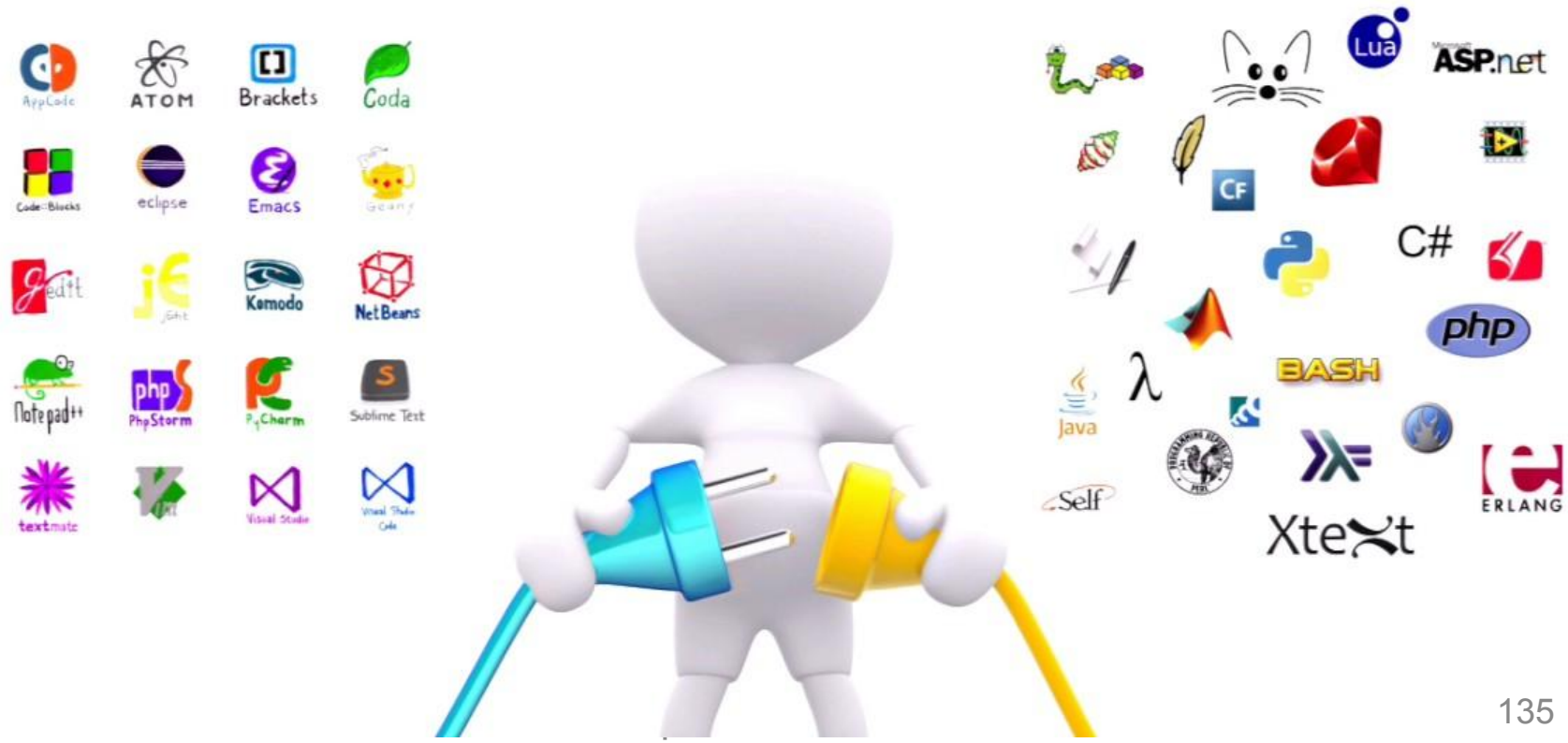
- Generic integration of language features: Content Assist, Go to definition, Validation, Find references, etc.
- JSON-RPC used to communicate Requests, Responses and Notifications
- Server provides semantics of a program in a certain language
- Client sends messages and requests on interactions: open document, change text…

https://microsoft.github.io/language-server-protocol/

The Language Server Protocol (LSP)

Editor developers focus on the editor part

Language developers focus on the language support



Language Servers Implementations

1C Enterprise
ABAP
ActionScript 2.0
Ada/SPARK
AML
Ansible
Angular
Antlr
API Elements
APL
Apache Camel
Apex
IBM High Level **Assembler**
IBM High Level **Assembler**
ASN.1
AsyncAPI
AWK
B/ProB
Ballerina
Bash
Bicep
BrightScript/BrighterScript
C#
C#
C++
C++/clang
C/C++/Objective-C
C/C++/Objective-C
CSS/LESS/SASS
Ceylon
Clarity
Clojure

CMake
Coq
IBM Enterprise **COBOL** for z/OS
IBM Enterprise **COBOL** for z/OS
IBM Enterprise **RPG ILE** for IBM i
IBM Enterprise **CL ILE** for IBM i
CodeQL
CoffeeScript
CWL
Crystal
Crystal
Cucumber/Gherkin
D
D
Dart
Data Pack
Delphi
DenizenScript
Deno (TypeScript/JavaScript)
Dockerfiles
DreamMaker
Erlang
Erlang
Erlang
Elixir
Elm
Ember
Ember
F#
F#
Fortran
Fortran

Fuzion
GLSL
GLSL for Minecraft
Gauge
GDScript
Gleam
Glimmer templates
Gluon
Go
Go
GraphQL
GraphQL
Graphviz/DOT
Grain
Groovy
Groovy
Groovy
HTML
Haskell
Haxe
HLSL
ink!
Isabelle
Idris2
Java (Eclipse)
Java
JavaScript
JavaScript Flow
JavaScript Flow
JavaScript-Typescript
JSON

Jsonnet
Julia
Kerboscript (kOS)
KerML
Kotlin
Language Server Robot
LanguageTool
LanguageTool
LaTeX
Lox
Lua
Lua
Lua
Liquid
IBM **LALR** Parser Generator language
Markdown
Markdown
MATLAB
Motorola 68000 Assembly
MSBuild
Nginx
Nim
OCaml/Reason
OCaml/Reason
OpenAPI
openVALIDATION
Papyrus
Perl
Perl
Perl
Pest

Crane **PHP**
PHP
PHP
PHP
PHP
PHP
PHPUnit
IBM Enterprise **PL/I** for z/OS
Polymer
PowerPC Assembly
PowerShell
PromQL
PureScript
Puppet
Python
Python
Python
Python
Pony
Q#
QML
R
Racket
Raku
RAML
RAML
ReasonML
Red
REL
ReScript
IBM TSO/E **REXX**
Robot Framework
Robot Framework
Robot Framework
Ruby
Ruby
Ruby
Ruby
Ruby
Ruby
Rust
Rust

Scala
Scala
Scheme
Shader
Slint
Smalltalk/Pharo
Snyk
SPARQL
SQL
Standard ML
Stylable
Svelte
Swift
SysML v2
Sysl
Systemtap
SystemVerilog
SystemVerilog
T-SQL
Tads3
Terraform
Terraform
Thrift
Tibbo Basic
Trino SQL
TTCN-3
TTCN-3
Turtle
Twig
TypeCobol
TypeScript
Typst
V
Vala
VDM-SL, VDM++, VDM-RT
Veryl
VHDL
VHDL
VHDL
Viml
Visualforce

Vue
WebAssembly
WebGPU Shading Language
Wolfram Language (Mathematica)
Wolfram Language
WXML
XML
XML
Mini**YAML**
YAML (with JSON schemas)
YAML
YARA
YANG
Zig

Tools supporting the LSP

Acme	Emacs	MS Monaco Editor	Theia
Atom	Emacs	MS Paint IDE	vim8 and neovim
BBEdit	Emacs	Multiple editors	vim8 and neovim
Brackets	ecode	Neovim	vim8 and neovim
Coginiti Pro	GNOME Builder	Nova	vim8 and neovim
Coginiti Premium	Helix Editor	Oni	vim8 and neovim
Cloud Studio	JCIDE	OpenSumi	vim8 and neovim
CodeLite	JupyterLab	Qt Creator	vim8 and neovim
CodeMirror	Kakoune	RAD Studio (Delphi and C++Builder)	vim9
CudaText	Kate	RJ TextEd	Visual Studio
Eclipse Che	Lite XL	Spyder	Visual Studio
Eclipse IDE	Moonshine IDE	Sublime Text	Visual Studio Code

https://microsoft.github.io/language-server-protocol/

Langium https://langium.org/

- State-of-the-art language workbench, successor of Xtext
- Issues of Xtext:
  - bounded to Java and Eclipse ecosystems
    - Also to Ecore meta-meta-model
  - not maintained anymore (Langium!)
  - Python Xtext https://pypi.org/project/textX/
  - LSP support (unstable)
- Langium is Web oriented:
  - TypeScript: for writing your parser/interpreter or integrating your DSL to a Web app
  - LSP (language server protocol) to target any editor (Eclipse, IntelliJ, VSCode, or even Web editors like Monaco)

Langium https://langium.org/

- Same principle as Xtext, but mainly
  - Grammar and concrete syntax first!
  - But the methodology and core ideas remain the same

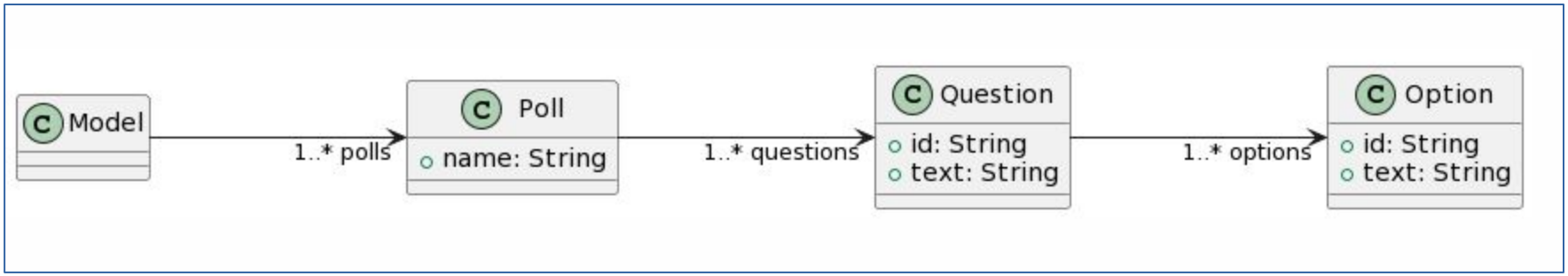
- Langium grammar to define the concrete syntax and drive the metamodel as well as tooling facilities (parser, validator, editor, etc.)

The Grammar Language of Langium

- Corner-stone of Langium
- A… DSL to define textual languages
  - Describe the concrete syntax
  - Specify the mapping between concrete syntax and domain model
- From the grammar, it is generated:
  - The domain model (aka semantic model or metamodel)
  - The language server
  - Additionally: syntax highlighting, etc.

Langium Grammar

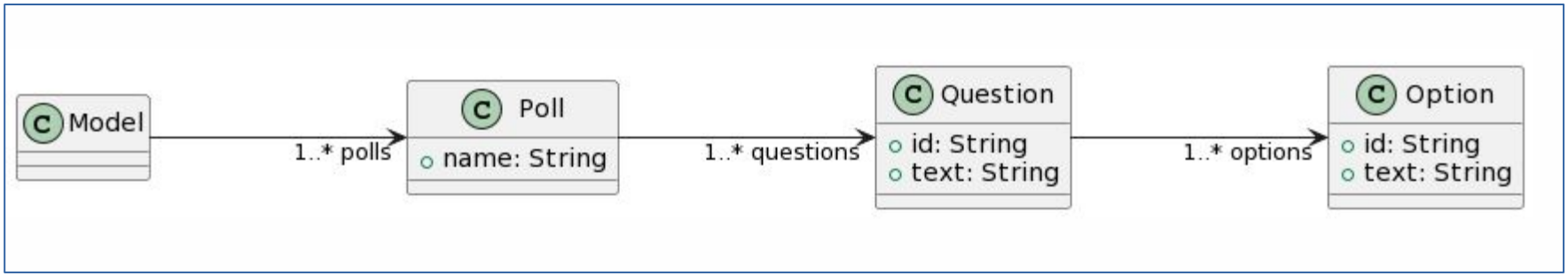
```
grammar PollSystem

entry Model:
    'PollSystem' '{' polls+=Poll+ '}';

Poll:
    'Poll' name=ID '{' questions+=Question+ '}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;

hidden terminal WS: /\s+/;
terminal ID: /[_a-zA-Z][\w_]*/;
terminal STRING: /"([^"]*)"/;

hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```



Parser rules: (1) valid sequence of tokens; (2) type of objects to be created by the parser and result in the creation of the AST.

Parser rules

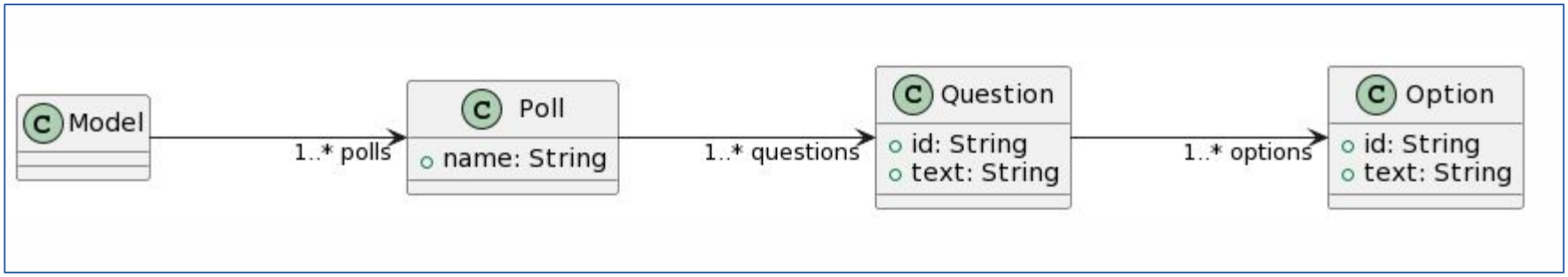
```
1   grammar PollSystem
2
3   entry Model:
4       'PollSystem' '{' polls+=Poll+ '}';
5
6   Poll:
7       'Poll' name=ID '{' questions+=Question+ '}';
8
9   Question:
10      'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';
11
12  Option:
13      id=ID ':' text=STRING;
14
15  hidden terminal WS: /\s+/;
16  terminal ID: /[_a-zA-Z][\w_]*/;
17  terminal STRING: /"([^"]*)"/;
18
19  hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
20  hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```



Keywords: inline terminals to match a character sequence surrounded by single or double quotes. Technical remark: must not be empty and must not contain white space.

```
1   grammar PollSystem
2
3   entry Model:
4       'PollSystem' '{' polls+=Poll+ '}';
5
6   Poll:
7       'Poll' name=ID '{' questions+=Question+ '}';
8
9   Question:
10      'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';
11
12  Option:
13      id=ID ':' text=STRING;
14
15  hidden terminal WS: /\s+/;
16  terminal ID: /[_a-zA-Z][\w_]*/;
17  terminal STRING: /"([^"]*)"/;
18
19  hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
20  hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```

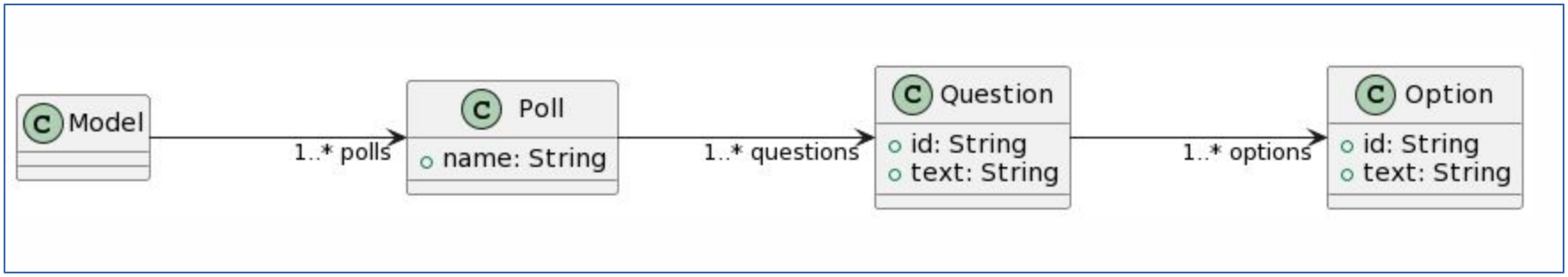
Keywords



Terminals: match a stream of characters and transforms into a stream of tokens; based on Javascript Regular Expressions

Terminals

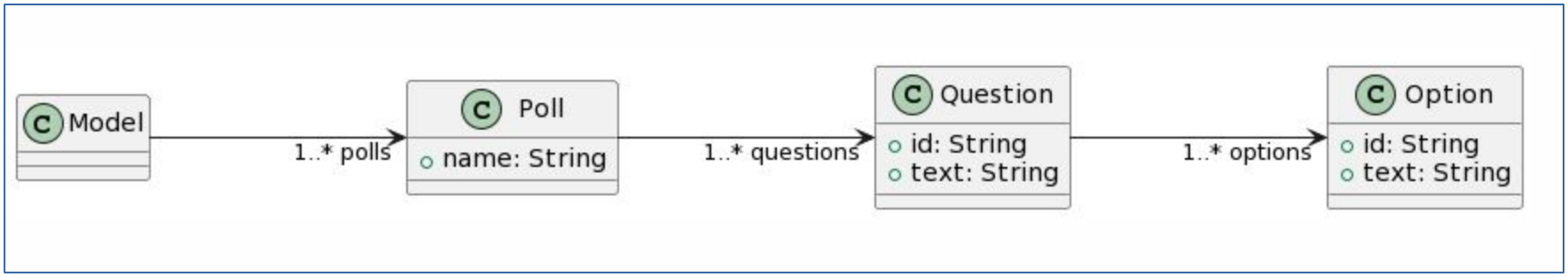
```
1   grammar PollSystem
2
3   entry Model:
4       'PollSystem' '{' polls+=Poll+ '}';
5
6   Poll:
7       'Poll' name=ID '{' questions+=Question+ '}';
8
9   Question:
10      'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';
11
12  Option:
13      id=ID ':' text=STRING;
14
15  hidden terminal WS: /\s+/;
16  terminal ID: /[_a-zA-Z][\w_]*/;
17  terminal STRING: /"([^"]*)"/;
18
19  hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
20  hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```

Hidden Terminal Rules: specify which characters or sequence of characters need to be ignored during lexing and parsing.
Eg ignoring whitespaces and comments.

Hidden Terminals

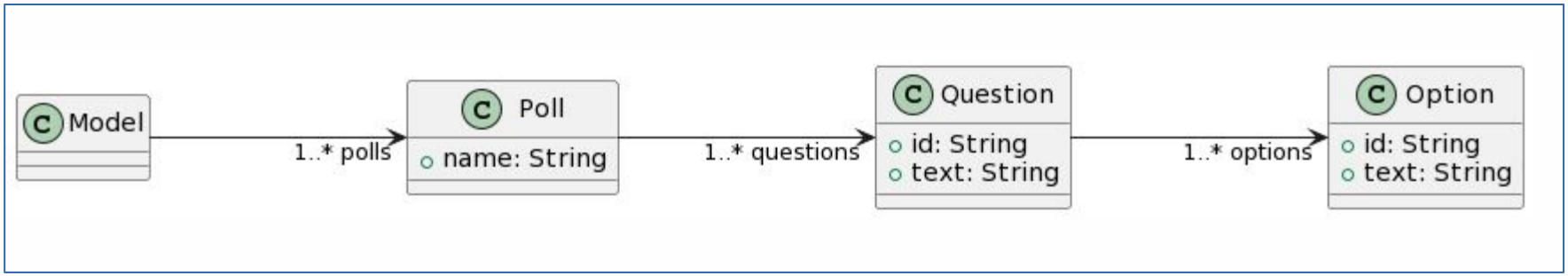
```
1   grammar PollSystem
2
3   entry Model:
4       'PollSystem' '{' polls+=Poll+ '}';
5
6   Poll:
7       'Poll' name=ID '{' questions+=Question+ '}';
8
9   Question:
10      'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';
11
12  Option:
13      id=ID ':' text=STRING;
14
15  hidden terminal WS: /\s+/;
16  terminal ID: /[_a-zA-Z][\w_]*/;
17  terminal STRING: /"([^"]*)"/;
18
19  hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
20  hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```

Multiple or single value assignment

= is used to assign single values to an object property.

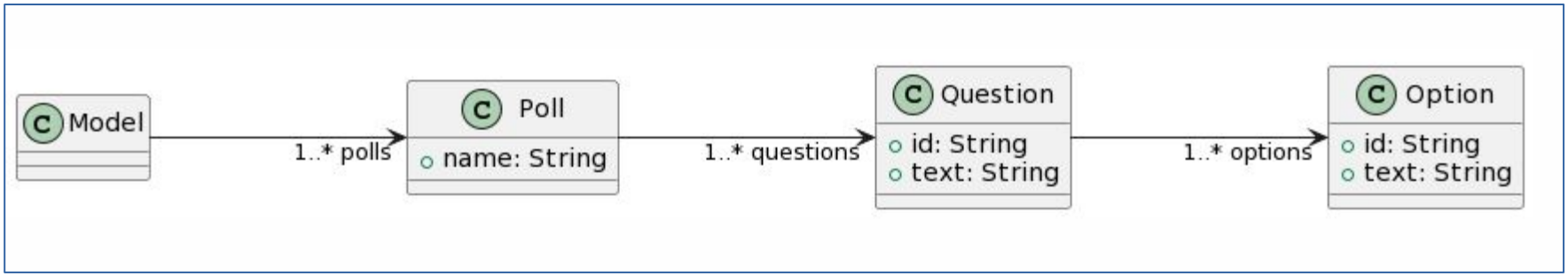
```
1   grammar PollSystem
2
3   entry Model:
4       'PollSystem' '{' polls+=Poll+ '}';
5
6   Poll:
7       'Poll' name=ID '{' questions+=Question+ '}';
8
9   Question:
10      'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';
11
12  Option:
13      id=ID ':' text=STRING;
14
15  hidden terminal WS: /\s+/;
16  terminal ID: /[_a-zA-Z][\w_]*/;
17  terminal STRING: /"([^"]*)"/;
18
19  hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
20  hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```



Multiple or single value assignment

+= is used to assign multiple values to an array property.

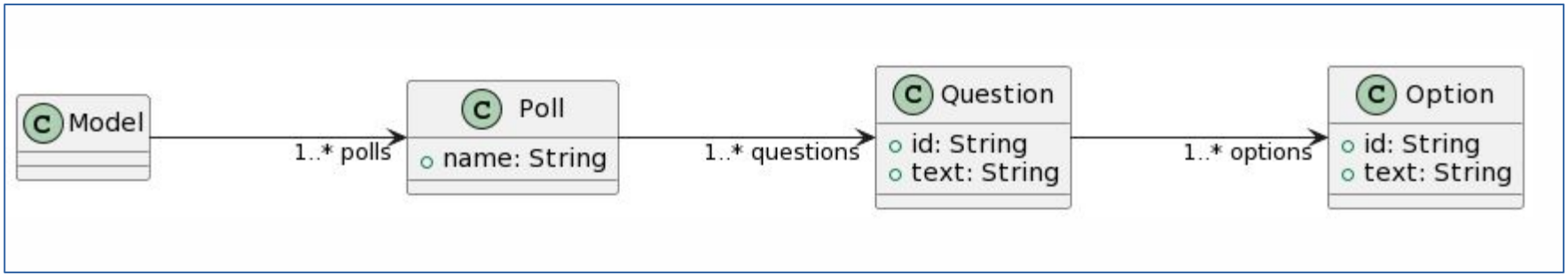
```
grammar PollSystem

entry Model:
    'PollSystem' '{' polls+=Poll+ '}';

Poll:
    'Poll' name=ID '{' questions+=Question+ '}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;

hidden terminal WS: /\s+/;
terminal ID: /[_a-zA-Z][\w_]*/;
terminal STRING: /"([^"]*)"/;

hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```



Cardinality

- exactly one/mandatory (**no** operator)
- zero or one (operator **?**)
- zero or many (operator ***)
- one or many (operator **+**)

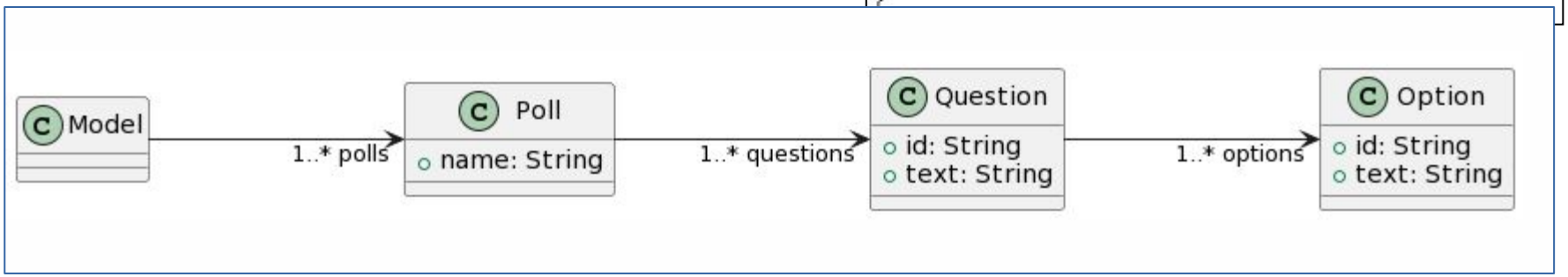
```
grammar PollSystem

entry Model:
    'PollSystem' '{' polls+=Poll+ '}';

Poll:
    'Poll' name=ID '{' questions+=Question+ '}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;

hidden terminal WS: /\s+/;
terminal ID: /[_a-zA-Z][\w_]*/;
terminal STRING: /"([^"]*)"/;

hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```



Grammar and Programs/Specifications/Models

```
grammar PollSystem

entry Model:
    'PollSystem' '{' polls+=Poll+ '}';

Poll:
    'Poll' name=ID '{' questions+=Question+ '}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;

hidden terminal WS: /\s+/;
terminal ID: /[_a-zA-Z][\w_]*/;
terminal STRING: /"([^"]*)"/;

hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```

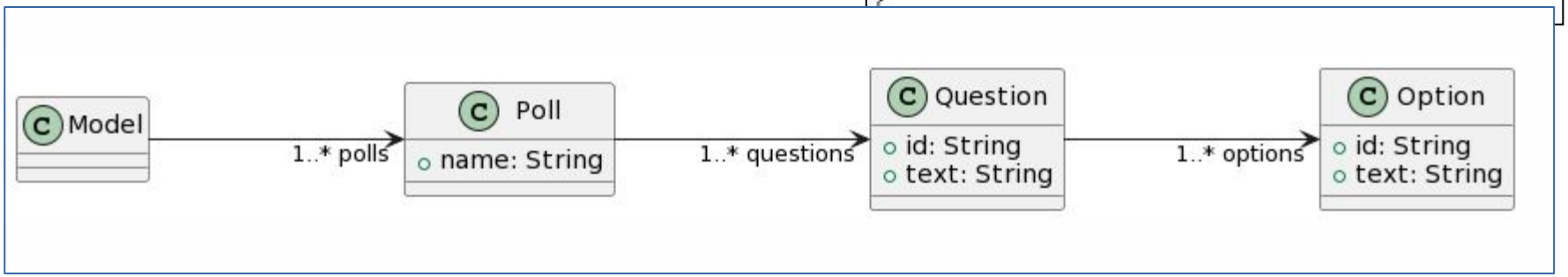
```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```



Grammar and Programs/Specifications/Models

```
grammar PollSystem

entry Model:
    'PollSystem' '{' polls+=Poll+ '}';

Poll:
    'Poll' name=ID '{' questions+=Question+ '}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;

hidden terminal WS: /\s+/;
terminal ID: /[_a-zA-Z][\w_]*/;
terminal STRING: /"([^"]*)"/;

hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```

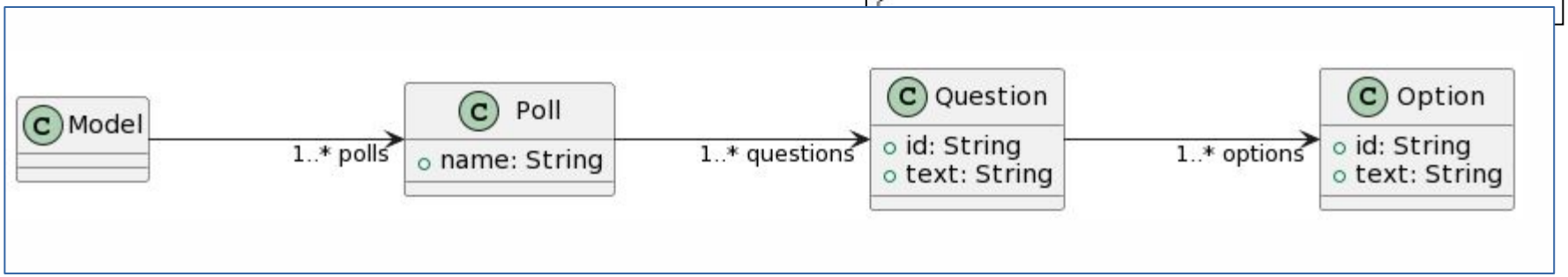
```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```



Grammar and Programs/Specifications/Models

```
grammar PollSystem

entry Model:
    'PollSystem' '{' polls+=Poll+ '}';

Poll:
    'Poll' name=ID '{' questions+=Question+ '}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;

hidden terminal WS: /\s+/;
terminal ID: /[_a-zA-Z][\w_]*/;
terminal STRING: /"([^"]*)"/;

hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```

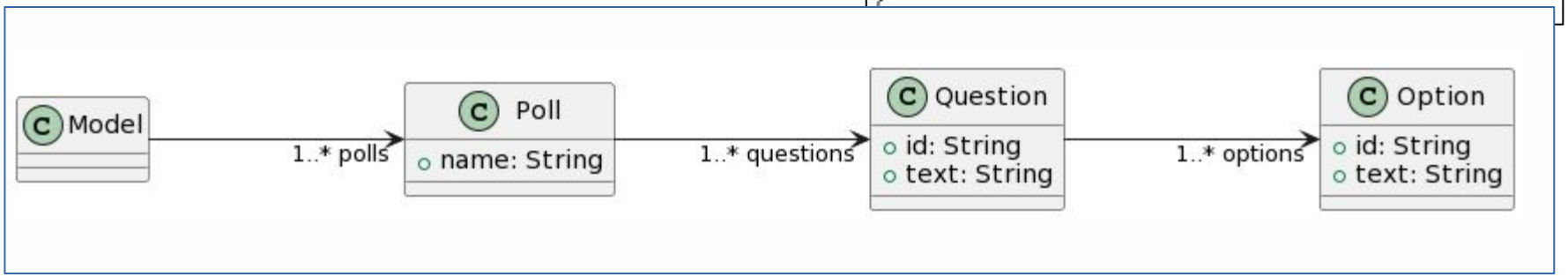
```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```



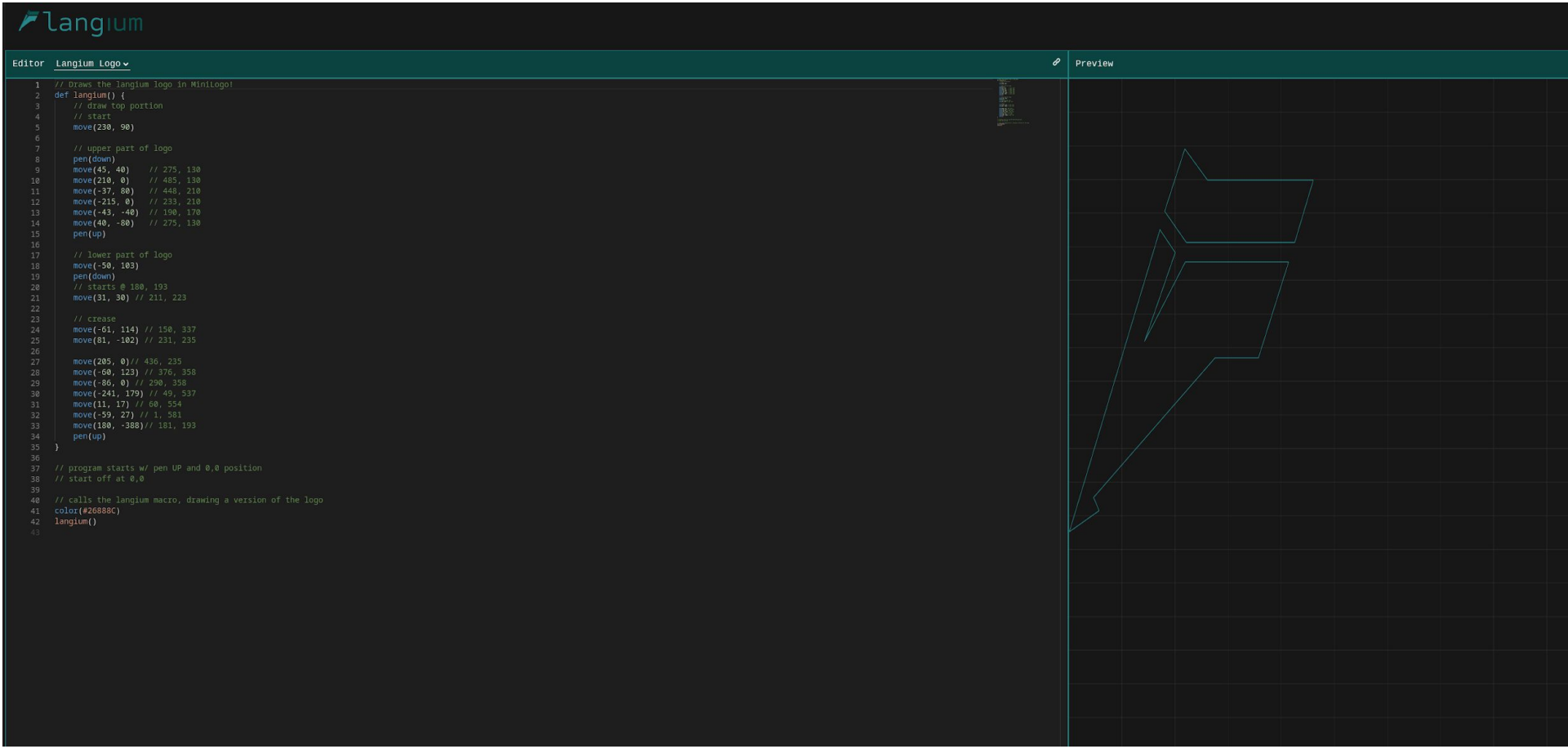
Grammar and Programs/Specifications/Models

```
grammar PollSystem

entry Model:
    'PollSystem' '{' polls+=Poll+ '}';

Poll:
    'Poll' name=ID '{' questions+=Question+ '}';

Question:
    'Question' id=ID '{' text=STRING 'options' '{' options+=Option+ '}' '}';

Option:
    id=ID ':' text=STRING;

hidden terminal WS: /\s+/;
terminal ID: /[_a-zA-Z][\w_]*/;
terminal STRING: /"([^"]*)"/;

hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
```

```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```



Langium (short demonstration)
https://langium.org/showcase/



Part 2: define a textual syntax with Langium for your statemachine metamodel…

langium

```
events
    switchCapacity
    next

initialState PowerOff

state PowerOff
    switchCapacity => RedLight
end

state RedLight
    switchCapacity => PowerOff
    next => GreenLight
end

state YellowLight
    switchCapacity => PowerOff
    next => RedLight
end

state GreenLight
    switchCapacity => PowerOff
    next => YellowLight
end
```

Part 2: define a textual syntax for your robot modeling language...

```
// RoboML is running in the web!

let bool entry () {
    setSpeed(30)
    var int count = 0
    var int eval = 1
    loop count < 5
    {
        count = count + 1
        s
    }
}

let bool
    Forwa
    Clock
    Forwa
    Clock
    Forwa
    Clock
    Forwa
    Clock
    retur
}
```

```
let bool entry () {
    setSpeed(30)
    var int count = 0
    var int eval = 1
    loop count < 5
    {
        count = count + 1
        square()
    }
}

let bool square(){
    Forward 200
    Clock 90
    Forward 200
    Clock 90
    Forward 200
    Clock 90
    Forward 200
    Clock 90
    return true
}
```

KEEP CALM AND DO IT YOURSELF

Plan

- Domain-Specific Languages (DSLs)
  - Languages and abstraction gap
  - Examples and rationale
  - DSLs vs General purpose languages, taxonomy
- External DSLs
  - Grammar and parsing
  - Xtext
- DSLs, DSMLs, and (meta-)modeling

Contract

- Better understanding/source of inspiration of software languages and DSLs
  - Revisit of history and existing languages
- Foundations and practice of Xtext
  - State-of-the-art language workbench (Most Innovative Eclipse Project in 2010, mature and used in a variety of industries)
- Models and Languages
  - Perhaps a more concrete way to see models, metamodels and MDE (IDM in french)

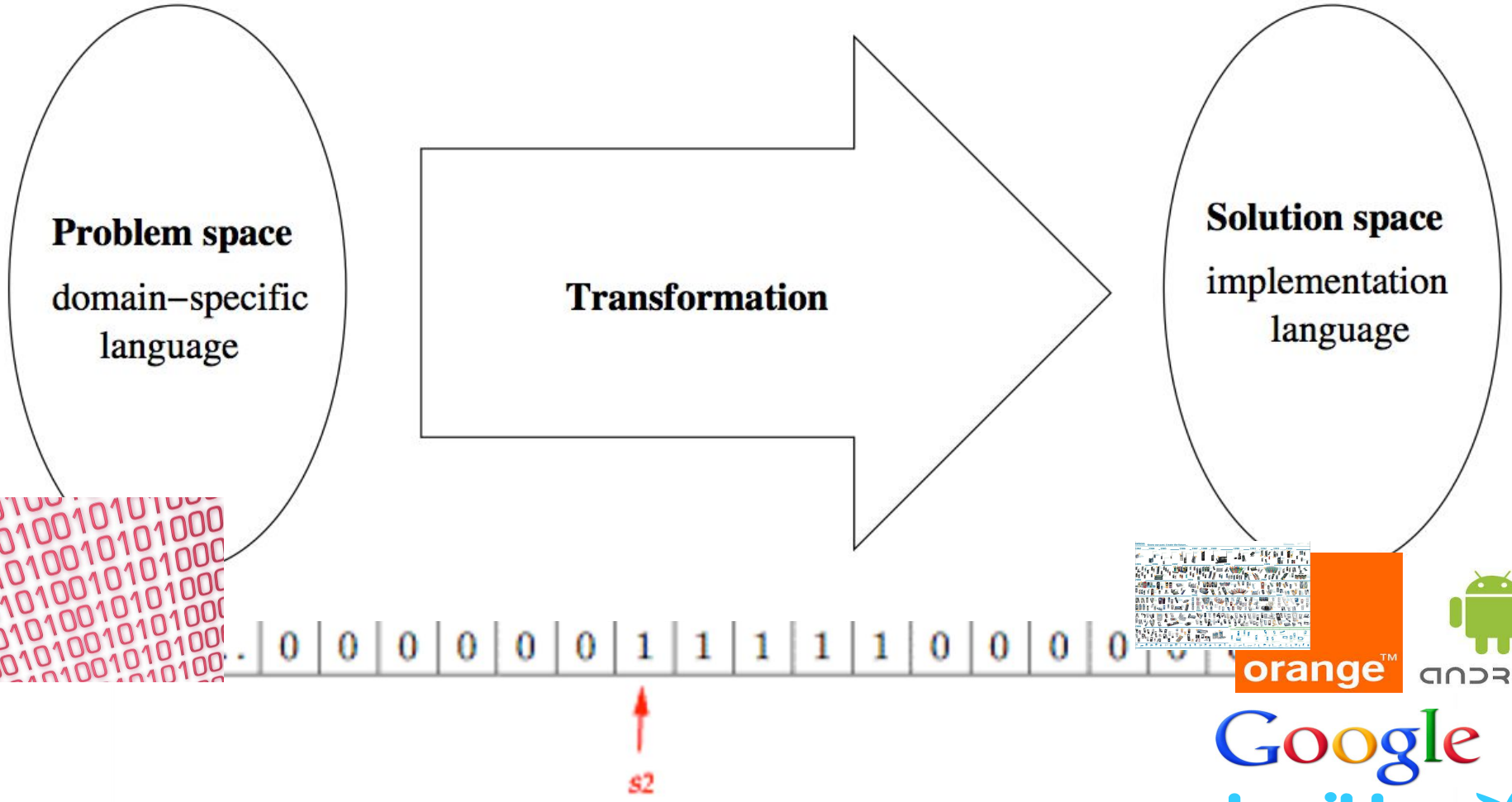
DSL,

Model,

Metamodel,

Summary

Abstraction Gap



Generative approach

- Programming the generation of programs
  - Very old practice
  - Metaprogramming: generative language and target language are the same
    - Reflection capabilities

- Generalization of this idea:
  - from a specification written in one or more textual or graphical domain-specific languages
  - you generate customized variants

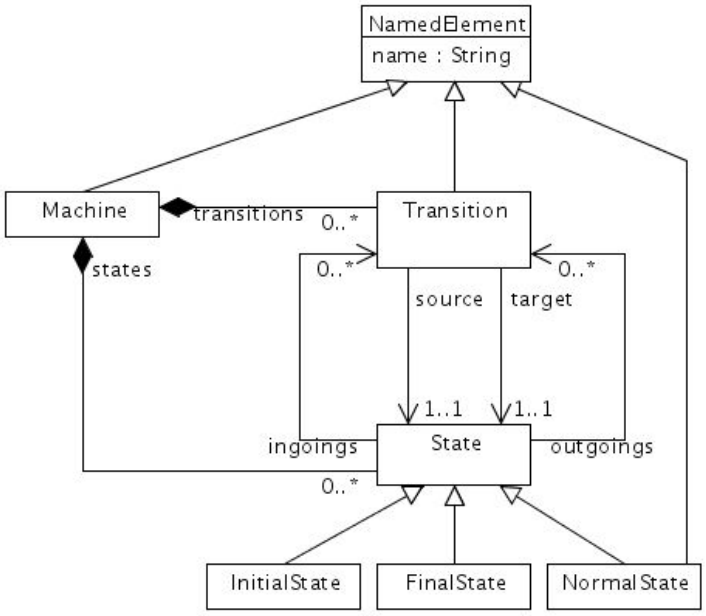
Models/MDE

- In essence, a model is an **abstraction** of some aspect of a system under study.
- Some details are hidden or removed to **simplify** and focus attention.
- A model is an abstraction since **general** concepts can be formulated by abstracting common properties of instances or by extracting common features from specific examples
- **(Domain-specific) Languages** enable the specification or execution of models

Grammar

```
machineDefinition:
  MACHINE OPEN_SEP stateList
  transitionList CLOSE_SEP;

stateList:
  state (COMMA state)*;

state:
  ID_STATE;

transitionList:
  transition (COMMA transition)*;

transition:
  ID_TRANSITION OPEN_SEP
  state state CLOSE_SEP;

MACHINE: 'machine';
OPEN_SEP: '{';
CLOSE_SEP: '{';
COMMA: ',';
ID_STATE: 'S' ID;
ID_TRANSITION: 'T' (0..9)+;
ID: (a..zA..Z_) (a..zA..Z0..9)*;
```

Metamodel



conforms to

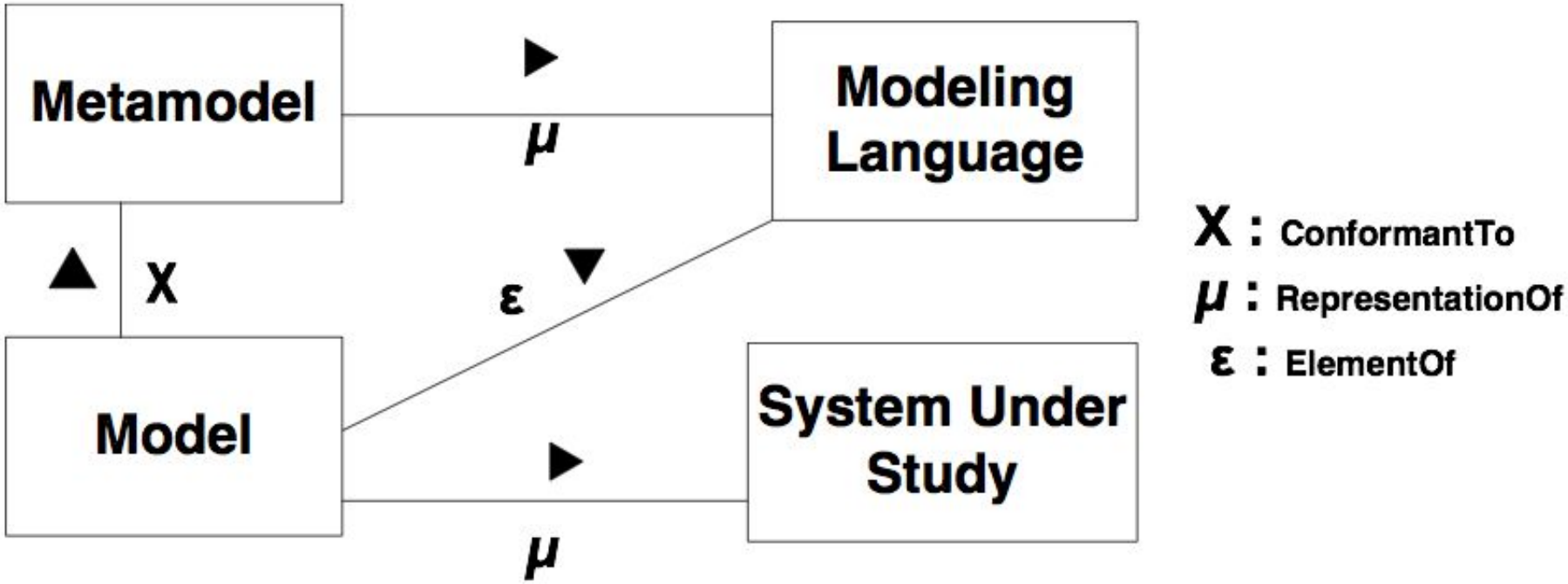
conforms

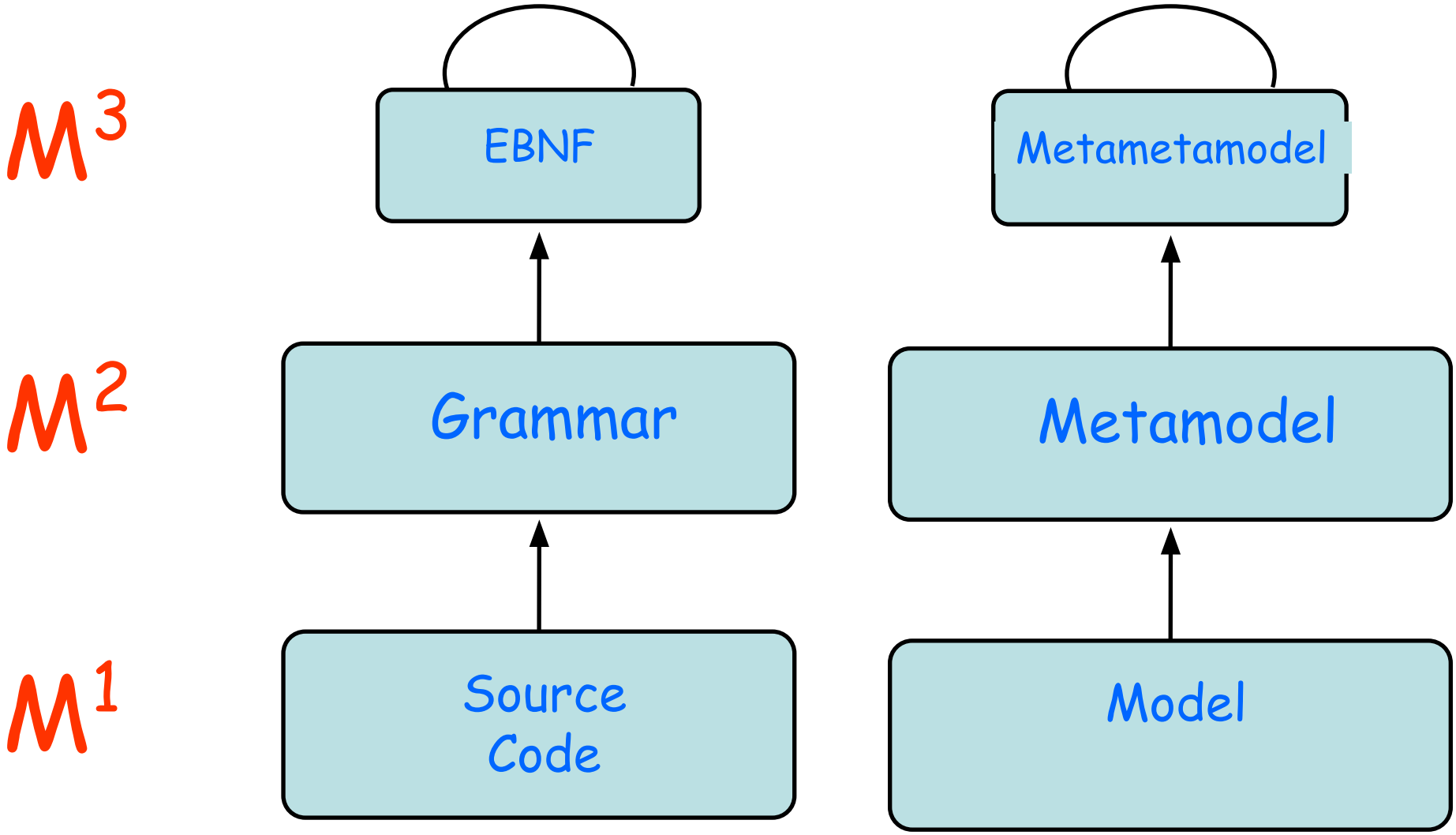
To

```
machine {
  SOne STwo
  T1 { SOne STwo }
}
```

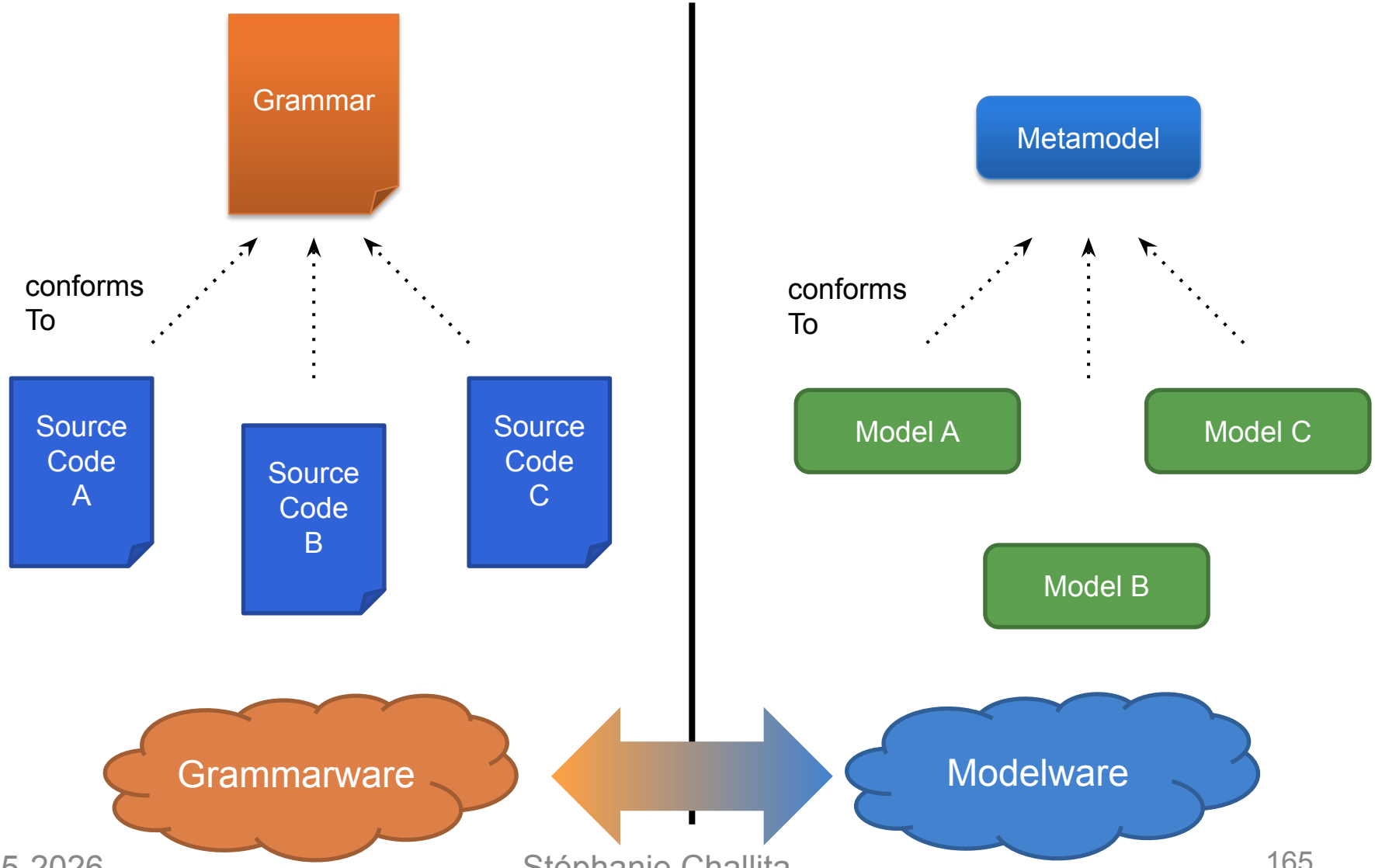
Source Code/Model

Model, Metamodel, Metametamodel, DSML

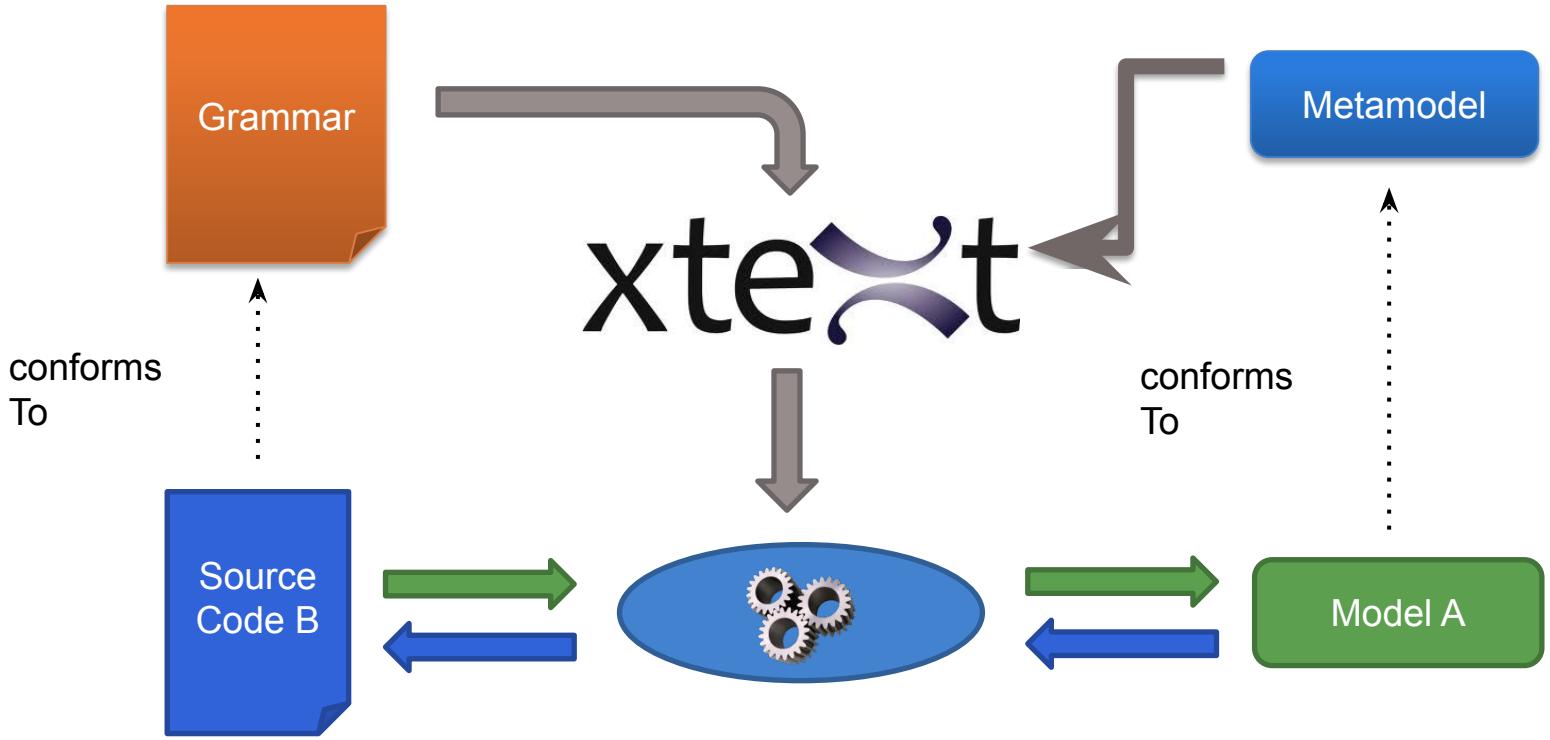



M3
M2
M1
EBNF
Grammar
Source Code
Metametamodel
Metamodel
Model

Language and MDE



MDE, Grammar: there and back again



Empirical Assessment of MDE in Industry

John Hutchinson, Jon Whittle, Mark Rouncefield
School of Computing and Communications
Lancaster University, UK
+44 1524 510492
{j.hutchinson, j.n.whittle, m.rouncefield}@lancaster.ac.uk

Steinar Kristoffersen
Østfold University College and Møreforskning Molde AS
NO-1757 Halden
Norway
+47 6921 5000
steinar.kristoffersen@hiof.no

Model-Driven Engineering Practices in Industry

John Hutchinson
School of Computing and Communications
Lancaster University, UK
+44 1524 510492
{j.hutchinson@lancaster.ac.uk}

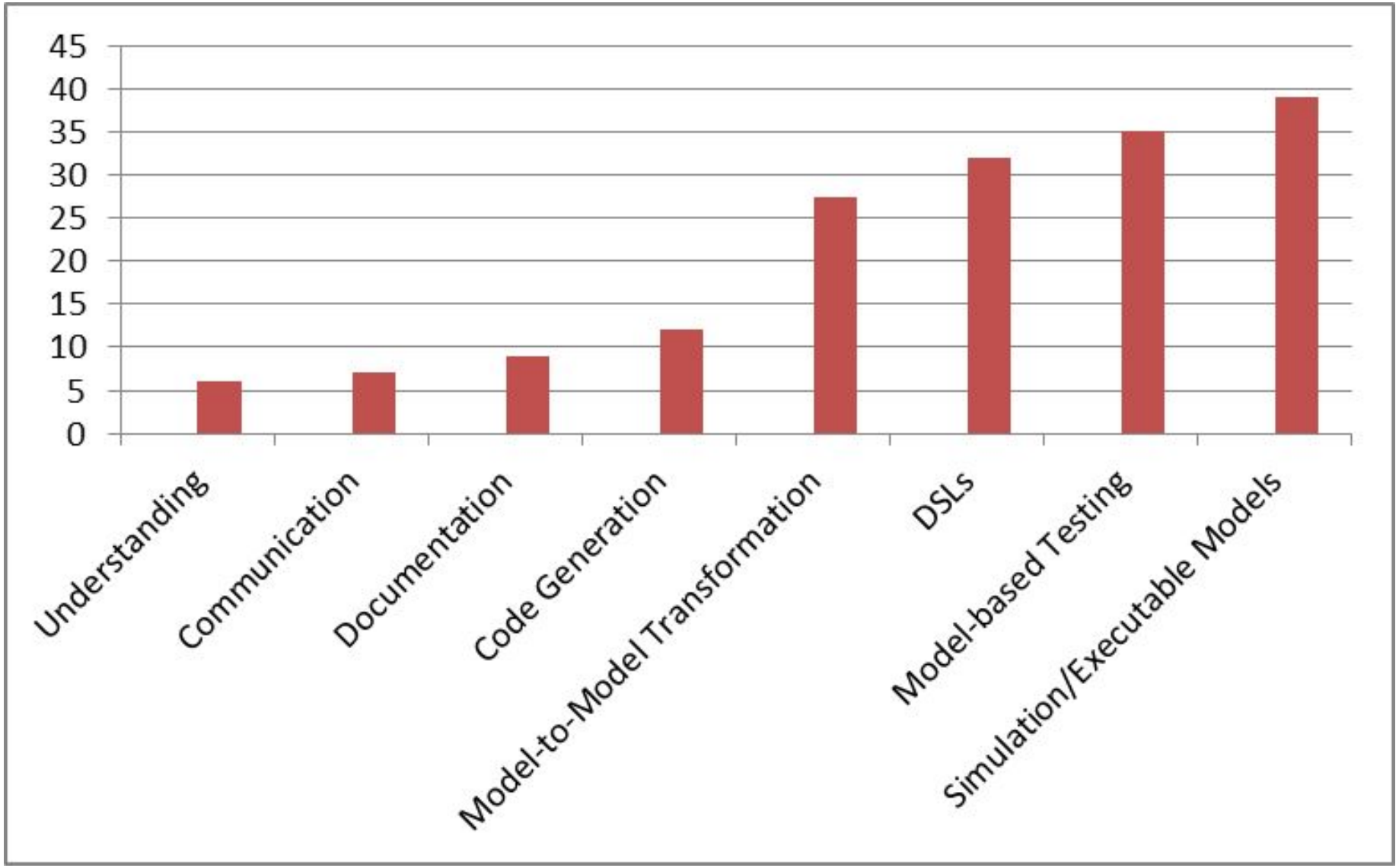
Mark Rouncefield
School of Computing and Communications
Lancaster University, UK
+44 1524 510492
{m.rouncefield@lancaster.ac.uk}

Jon Whittle
School of Computing and Communications
Lancaster University, UK
+44 1524 510492
{j.n.whittle@lancaster.ac.uk}

2011

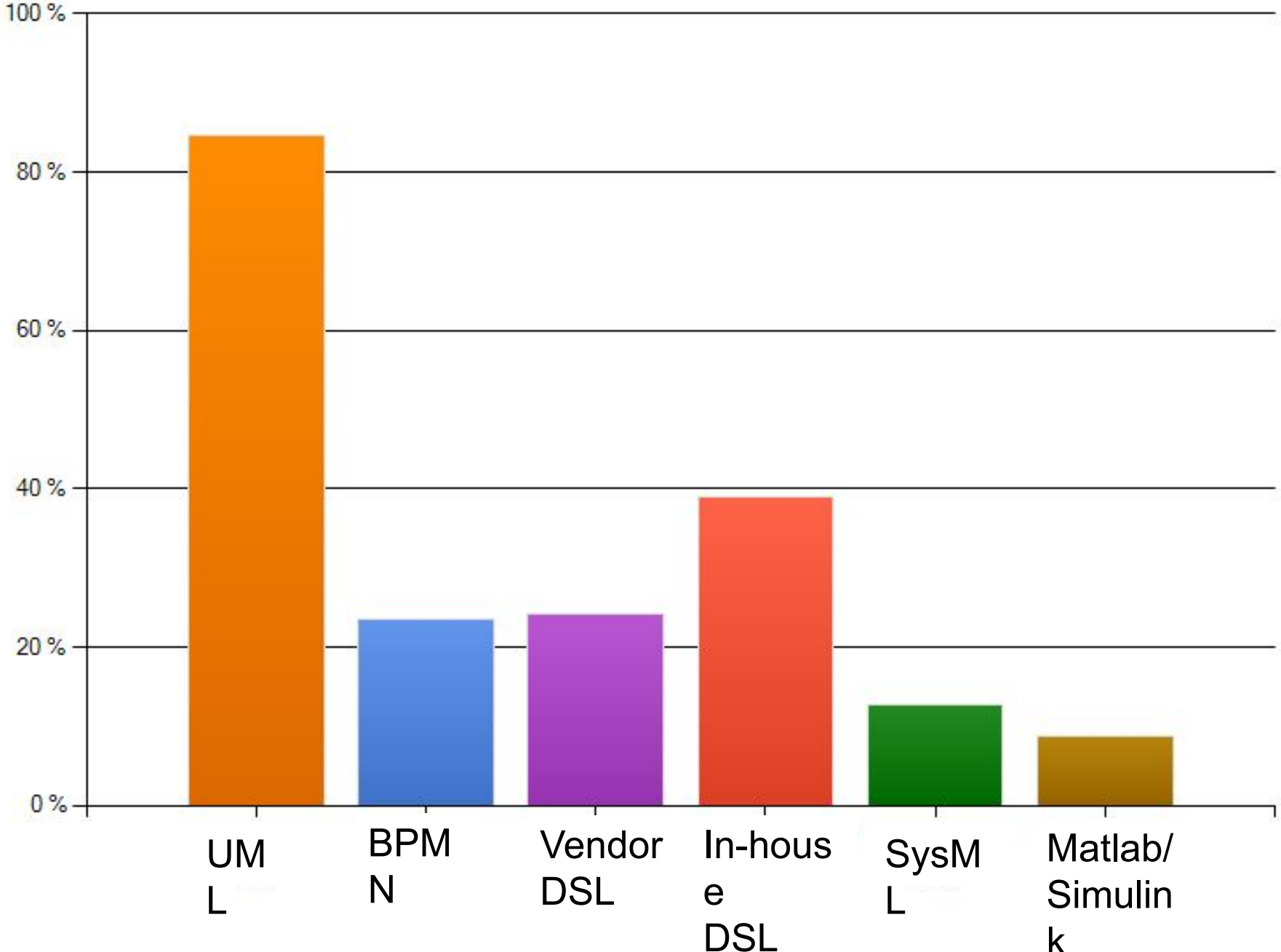
« Domain-specific languages are far more prevalent than anticipated »

What are models used for?

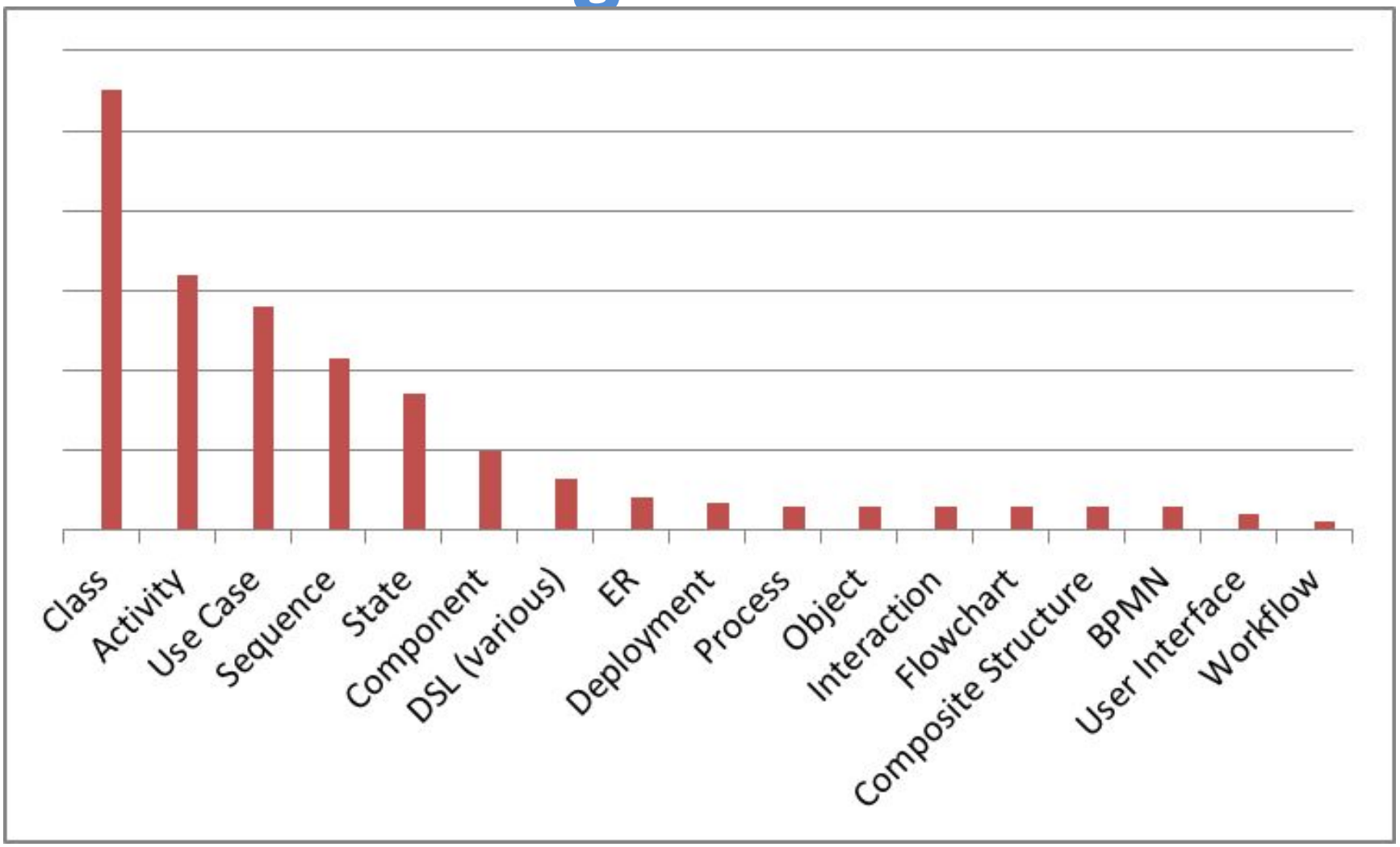


"Do not use" percentages for MDE activities

Which modeling languages do you use?



Which diagrams are used?

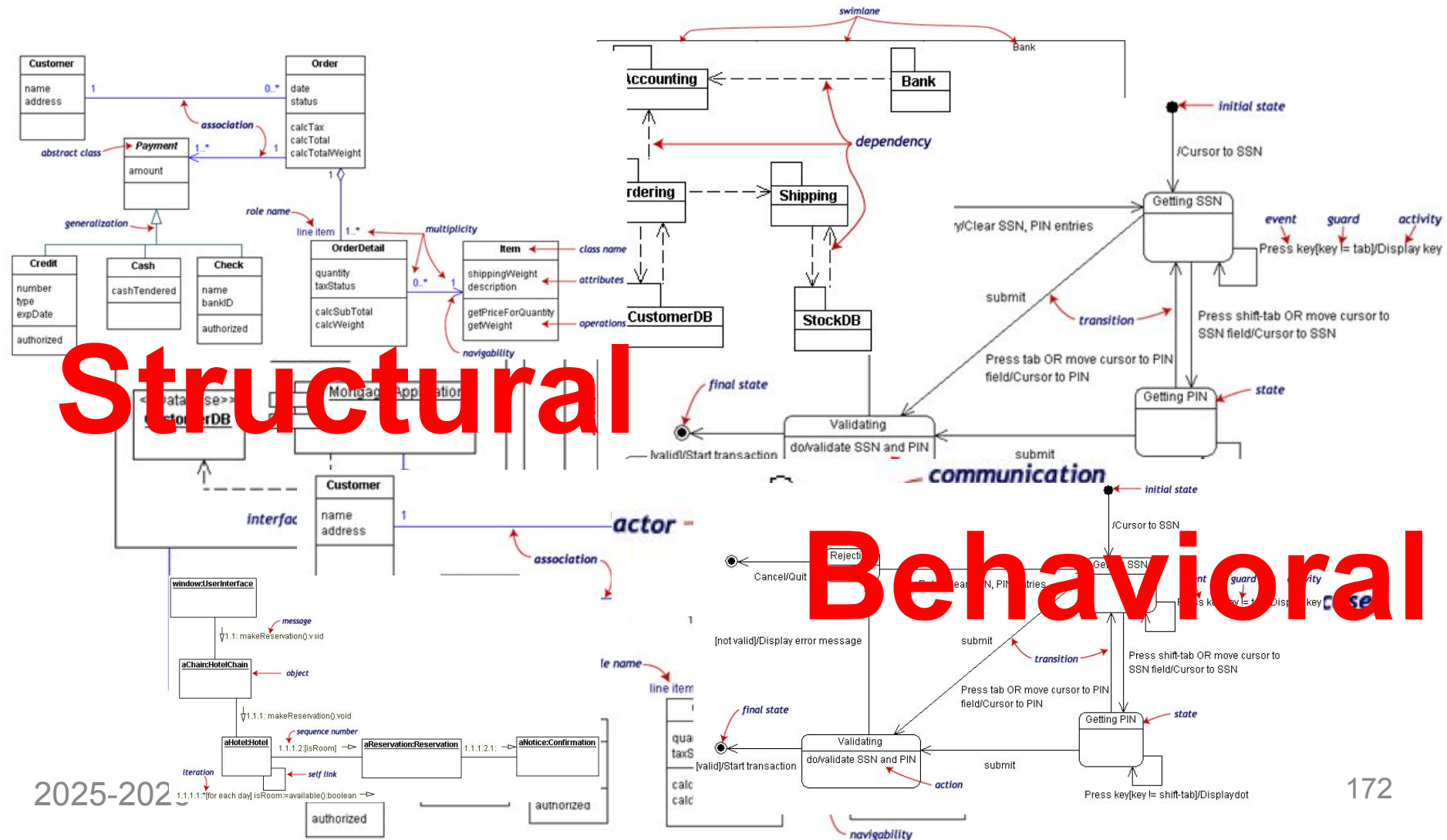


19 different diagram types are used regularly

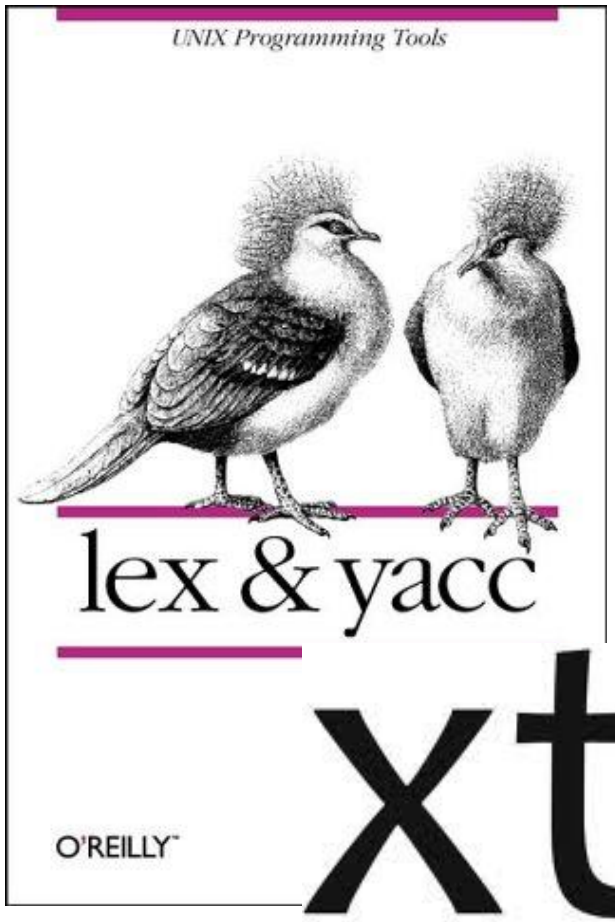
Use of multiple languages (DSLs)

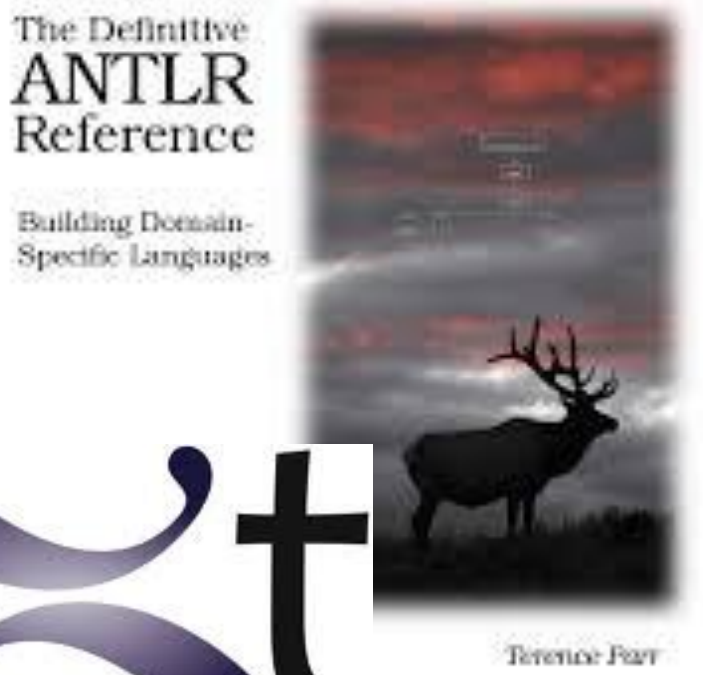
- 62% of those using custom DSLs also use UML
- Almost all users of SysML and BPMN also use UML
- UML is the most popular ‘single use’ language
  - 38% of all respondents
- UML used in combination with just about every combination of modeling languages
  - 14% of UML users combine with vendor DSL
  - 6% with both custom and vendor DSL

UML can be seen as a collection of domain-specific modeling languages



Xtext is built using MDE technologies





Xtext (and alternatives) democratize DSL development

My 3 takeaway messages

1. DSLs are important (as intuited for a long time – it will become more and more apparent)
2. DSL technology is here (no excuse)
3. MDE meets language engineering

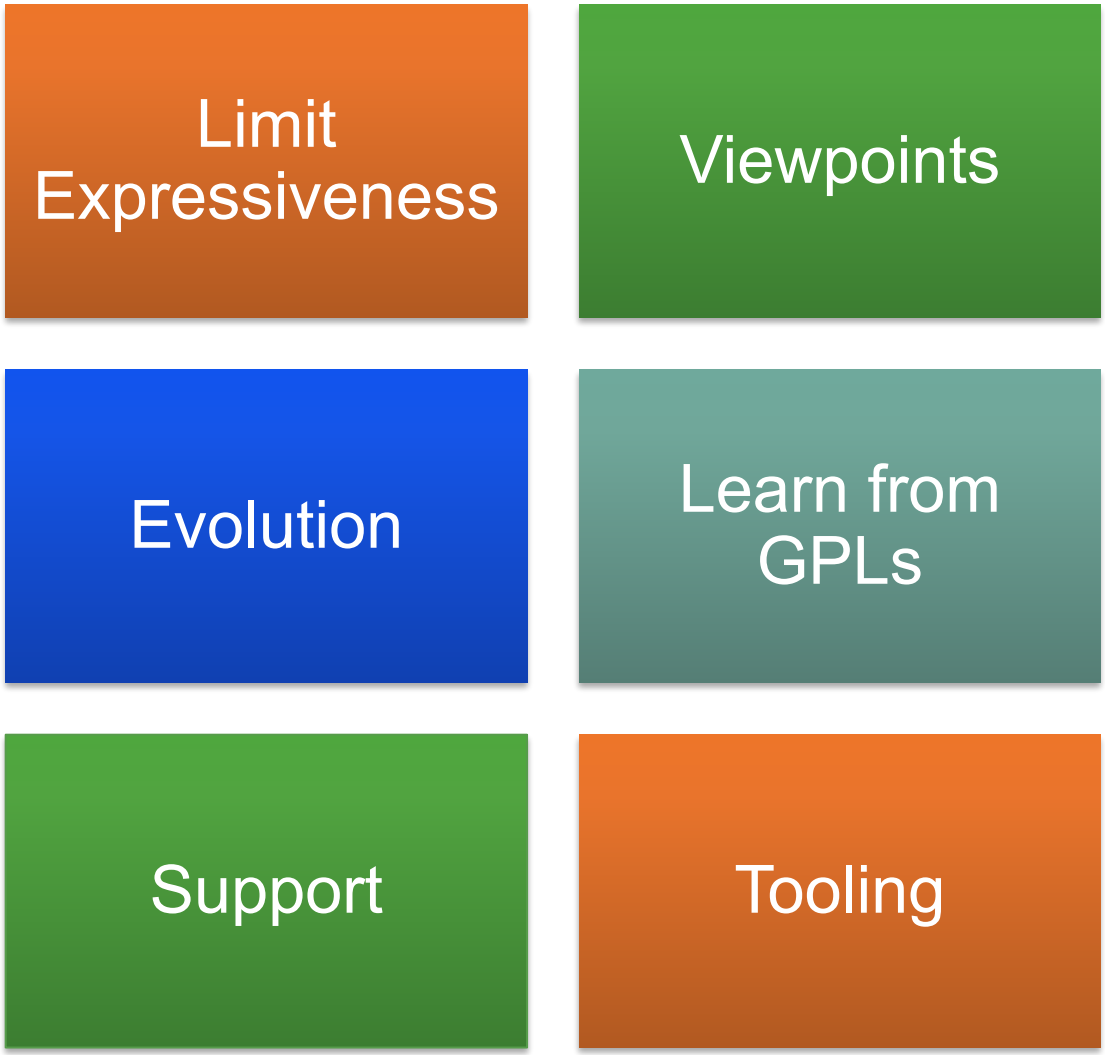
But my take away message is NOT

That DSLs should be used systematically, in every situations

When Developing DSLs?

- Tradeoff cost/time of development versus producivity gained for solving problems
  - If you use your DSL for resolving one problem, just one time, hum…
  - DSL: reusable, systematic means to resolve a specific task in a given domain
- DSL development can pay off quickly
  - 5’ you can get a DSL
- But DSL development can be time-consuming and numerous worst practices exists

Best Practices



Worst Practices

- Initial conditions
  - Only Gurus allowed
    - Believe that only gurus can build languages or that "I'm smart and don't need help"
  - Lack of Domain Understanding
    - Insufficiently understanding the problem domain or the solution domain
  - Analysis paralysis
    - Wanting the language to be theoretically complete, with its implementation assured

Worst Practices

- The source for Language Concepts
  - UML: New Wine in Old Wineskins
    - Extending a large, general-purpose modeling language
  - 3GL Visual Programming
    - Duplicating the concepts and semantics of traditional programming languages
  - Code: The Library is the Language
    - Focusing the language on the current code's technical details
  - Tool: if you have a hammer
    - Letting the tool's technical limitations dictate language development

Worst Practices

- The resulting language
  - Too Generic / Too Specific
    - Creating a language with a few generic concepts or too many specific concepts, or a language that can create only a few models
  - Misplaced Emphasis
    - Too strongly emphasizing a particular domain feature
  - Sacred at Birth
    - Viewing the initial language version as unalterable

Worst Practices

- Language Notation
  - Predetermined Paradigm
    - Choosing the wrong representational paradigm or the basis of a blinkered view
  - Simplistic Symbols
    - Using symbols that are too simple or similar or downright ugly

Worst Practices

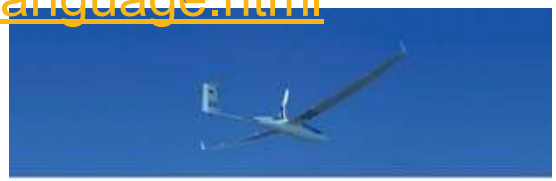
- Language Use
  - Ignoring the use process
    - Failing to consider the language's real-life usage
  - No training
    - Assuming everyone understands the language like its creator
  - Pre-adoption Stagnation
    - Letting the language stagnate after successful adoption

Engineering Modeling Languages

Turning Domain Knowledge into Tools

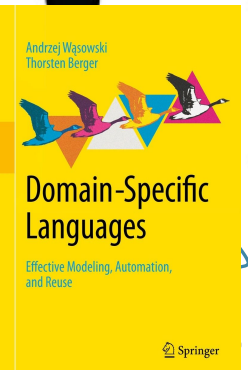
http://martinfowler.com/bliki/DomainSpecificLanguage.html

Benoit Combemale
Robert B. France
Jean-Marc Jézéquel
Bernhard Rumpe
Jim Steel
Didier Vojtisek



DSL Engineering

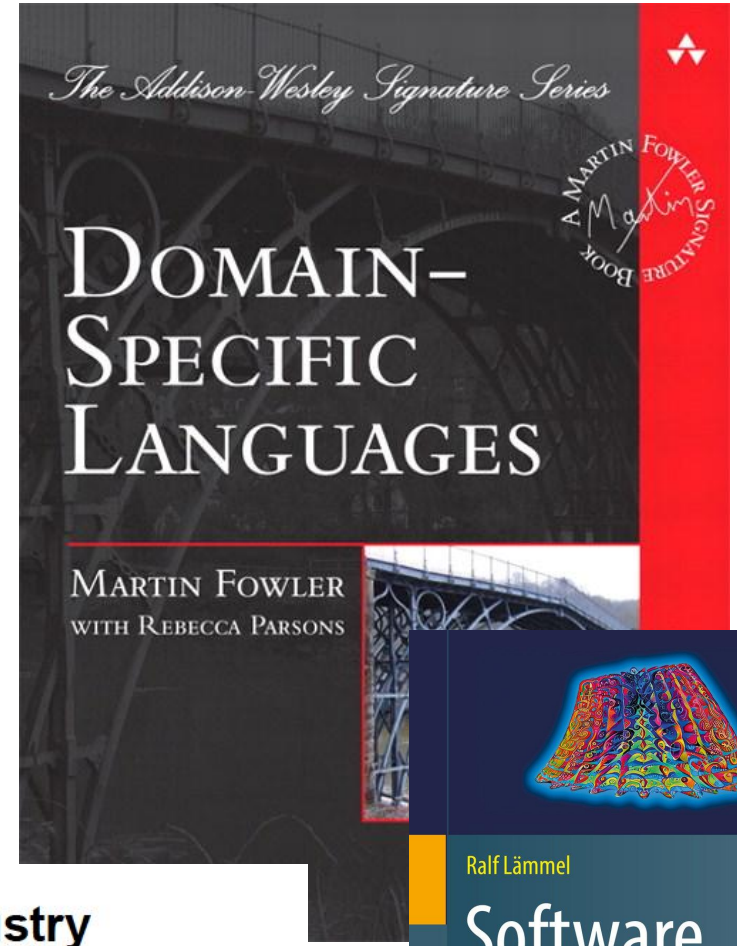
Designing, Implementing and Using Domain-Specific Languages

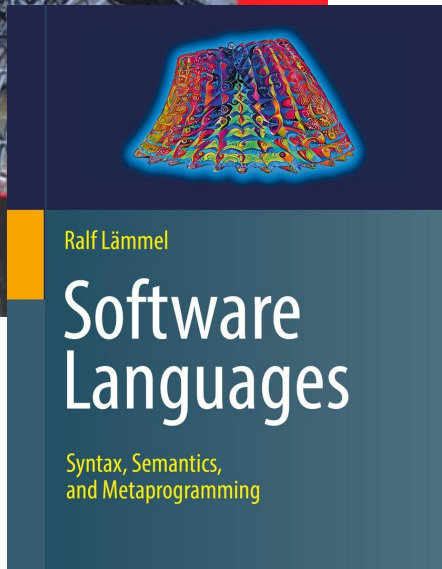


{S} spoofax

Markus Voelter

with Sebastian Benz, Christian Dietrich, Birgit Engelmann
Mats Helander, Lennart Kats, Eelco Visser, Guido Wachsmuth





Empirical Assessment of MDE in Industry

John Hutchinson, Jon Whittle, Mark Rouncefield
School of Computing and Communications
Lancaster University, UK
+44 1524 510492
{j.hutchinson, j.n.whittle, m.rouncefield}@lancaster.ac.uk

Steinar Kristoffersen
Østfold University College and Møreforskning Molde AS
NO-1757 Halden
Norway
+47 6921 5000
steinar.kristoffersen@hiof.no