Domain-Specific Languages

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

1937
Turing Machine

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

\[
\langle \text{State}_{\text{current}}, \text{Symbol}, \text{State}_{\text{next}}, \text{Action} \rangle
\]
Turing Machine
~ kind of state machine
The (Hi)Story of Software Engineering

Computer Science
Programming the Turing Machine

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
Programming the Turing Machine

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

You cannot be serious
Formulas are Turing complete
Formulas are Turing complete

Youtube video https://t.co/RTfJAxXYaX

http://fr.slideshare.net/Felienne/spreadsheets-are-code-online
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?

++++++++++++[>++++++++>++++++++++++]>+++<<-
]>++>.++++++
..+++>.<<++++++++++.>.+++.--.--.--.--.--.--.>+. 
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?
Programming the Turing Machine

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

011110111110

orange™

Google

twitter

Instagram

Android
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
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

```java
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 на int!
music();
System.out.print("\n");
```
Limits of General Purpose Languages (2)

• Not targeted to a particular kind of problem, but to any kinds of software problem.
Domain Specific Languages

• 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!
Hello World

My first Web page.
CSS

.CssMirror {  
    line-height: 1;
    position: relative;
    overflow: hidden;
}
.CssMirror-scroll {  
    /* 30px is the magic margin used to hide the element's real scrollbars */  
    margin-bottom: -30px; margin-right: -30px;
    padding-bottom: 30px; padding-right: 30px;
    height: 100%;  
    outline: none; /* Prevent dragging from highlighting the element */  
    position: relative;
}
.CssMirror-sizer {  
    position: relative;
}

Domain: web (styling)
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)
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"
)

index-file.names = ( "index.html" )

Domain: web server (configuration)
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

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

Domain: strings (pattern matching)
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.
BibTeX
HTML
Graphviz
PGN
Make
Matlab
SQL
Domain-Specific Languages (DSLs)
« 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.

I see a great future for very systematic and very modest programming languages »

1972

ACM Turing Lecture, « The Humble Programmer »
Edsger W. Dijkstra
Domain-specific languages are far more prevalent than anticipated
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. »
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.
What is offered?

- Higher abstractions
- Avoid redundancy
- Separation of concerns
- Use domain concepts

Promises of domain-specific languages
Promises of domain-specific languages

Benefits

- Productivity
- Quality
- V&V
- Domain Expert
- Communication
- No Overhead
- Platform Independent
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.

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

-- 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();
// 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 »

```java
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();
```
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.executeUpdate(query);
    System.out.println("Data inserted in table !");
}
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
  – (deeply) 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

• Both internal and external DSLs have strengths and weaknesses
  – learning curve,
  – cost of building,
  – programmer familiarity,
  – communication with domain experts,
  – mixing in the host language,
  – strong expressiveness boundary

• 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, Sirius
Contract

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

• Foundations and practice of EMF, Xtext, 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
<table>
<thead>
<tr>
<th>Feature</th>
<th>Ensō</th>
<th>Más</th>
<th>MetaEdit+</th>
<th>MPS</th>
<th>Onion</th>
<th>Rascal</th>
<th>Spoonax</th>
<th>SugarJ</th>
<th>Whole</th>
<th>Xtext</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Notation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Textual</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Graphical</td>
<td>●</td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>Tabular</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Symbols</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Semantics</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model2Text</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Model2Model</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Concrete syntax</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Interpretative</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Validation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structural</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Naming</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Types</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Programmatic</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Testing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSL testing</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>DSL debugging</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>DSL prog. debugging</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Composability</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Syntax/views</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Validation</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Semantics</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Editor services</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Editing mode</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Free-form</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Projectional</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Syntactic services</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Highlighting</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Outline</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Folding</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Syntactic completion</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Diff</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Auto formatting</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Semantic services</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference resolution</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Semantic completion</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Refactoring</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Error marking</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Quick fixes</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Origin tracking</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Live translation</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

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

Erdweg et al. SLE’13
Sebastian Erdweg, Tillmann Rendel, Christian Kästner, and Klaus Ostermann. Sugarj: Library-based syntactic language extensibility. OOPSLA'11
Projectional editing

**Parsing**
- Concrete Syntax
  - Abstract Syntax Tree

**Projection**
- Concrete Syntax
  - Abstract Syntax Tree
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 | tp->alt <= 2000 m | tp->alt >= 2000 m
        | tp->speed < 150 mps 0 | 10
        | tp->speed >= 150 mps 5 | 20
    } runnable judger_addTrackpoint
    int16 judger_getResult() <= op judger.getResult {
        return points;
    } runnable judger_getResult
} component Judge
Projectional Editing

```plaintext
exported statemachine FlightAnalyzer initial = beforeFlight {

next(Trackpoint* tp) reset()

<table>
<thead>
<tr>
<th>State</th>
<th>Transition Conditions</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>beforeFlight</td>
<td>[tp-&gt;alt == 0 m] -&gt; airborne</td>
<td></td>
</tr>
<tr>
<td>airborne</td>
<td>[tp-&gt;alt == 0 m &amp;&amp; tp-&gt;speed == 0 mps] -&gt; crashed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[tp-&gt;alt == 0 m &amp;&amp; tp-&gt;speed &gt; 0 mps] -&gt; landing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[tp-&gt;speed &gt; 200 mps &amp;&amp; tp-&gt;alt == 0 m] -&gt; airborne</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[tp-&gt;speed &gt; 100 mps &amp;&amp; tp-&gt;speed &lt;= 200 mps &amp;&amp;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tp-&gt;alt == 0 m] -&gt; airborne</td>
<td></td>
</tr>
<tr>
<td>landing</td>
<td>[tp-&gt;speed == 0 mps] -&gt; landed</td>
<td></td>
</tr>
<tr>
<td>landed</td>
<td>[tp-&gt;speed &gt; 0 mps] -&gt; landing</td>
<td></td>
</tr>
<tr>
<td>crashed</td>
<td>[ ] -&gt; beforeFlight</td>
<td></td>
</tr>
</tbody>
</table>

}
```
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
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](http://www.eclipse.org/modeling)
Eclipse Modeling

Xtext

activity cerm {
  activity LAJUR from 1 jan to 28 feb
  using 1 Tractor and 2 People
}

activity SENLS from 25 mar to 31 apr
  using 1 Tractor and 2 People

activity SIMPLI from 20 jun to 20 aug
  using 1 Tractor and 2 People

activity FERTILITATION from 20 mar to 31 may
  using 1 Tractor and 2 People

activity PRODUCITION from 1 oct to 31 dec
  using 1 Tractor and 2 People

activity RECOLTE from 1 sept to 30 sept
  using 1 Tractor and 2 People
}

activity cerm {
  activity LAJUR from 1 sept to 30 sept
  using 1 Tractor and 2 People
}

activity SENLS from 1 oct to 31 oct
  using 1 Tractor and 2 People

activity FERTILITATION from 1 oct to 31 dec
  using 1 Tractor and 2 People

activity RECOLTE from 1 sept to 30 sept
  using 1 Tractor and 2 People
}

Siriis

Generate Code

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](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 *metametamodel:*
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
Implementation with Java

MOF

The UML metamodel

A UML Model

A Specific phenomenon corresponding to a UML Model

How it works?
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

MOF

The UML metamodel

A UML Model

A Specific phenomenon corresponding to a UML Model

E\textit{\textsubscript{core}}\textit{M} E\textit{\textsubscript{core}}\textit{MM}

<<instanceOf>>

UML M UML MM

<<instanceOf>>

A UML model

.... ?

PROMOTION (EMF Generator)
EMF Toolset from 30.000 Feet

• The EMF Generator do not work on the .ecore
• EMF defines a .genmodel in parallel:
  – New/ Other/ Eclipse Modeling Framework/ EMF Model
  – 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 30.000 Feet

EMF.editor

Holds presentation code

EMF.edit

Holds presentation independent adaptors

EMF.model

Holds the PIM based model code

From "Mastering Eclipse Modeling Framework", V. Bacvanski and P. Graff
EMF Toolset from 30.000 Feet

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.

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

```java
/**
 * @generated NOT
 */

public int order () {
    return this.getEdges().size();
}
```

Do not forget to mark (@generated NOT) to prevent crushing!
Part 1: define a metamodel to help you developing state machines...
Xtext, a popular, easy-to-use model-based tool for developing textual DSLs.

Your textual DSL in 5’ (incl. editors, serializers)
Foundations
(or some course refresh)

\[ M^3 \]

EBNF

\[ M^2 \]

Grammar

\[ M^1 \]

Source Code

Java Grammar

Java Program

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

(Syntaxe concrète)
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}"")
}
Compilation (en français)
DSL? The same!

$M^3$  
EBNF

$M^2$  
Grammar

$M^1$  
Source Code

DSL Grammar

DSL specification/program
$M^3$

$M^2$

$M^1$

EBNF

Grammar

Source Code

Metamodel

Model

Metametamodel
Language and MDE

Grammarware

Grammar

conforms To

Source Code A

Source Code B

Source Code C

conforms To

Metamodel

Model A

Model B

Model C

Modelware

xtext
Give me a **grammar**, I’ll give you (for free)
* a comprehensive editor (auto-completion, syntax highlitening, 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)
Xtext, Grammar, Metamodel

- Grammar
- Metamodel
- Source Code A
- Model A

Xtext conforms to Grammar, Grammar conforms to Metamodel, Source Code A conforms to Model A.
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
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

![Diagram showing DSL Tooling and generation of Poll System application in different platforms (Java, Android, and iOS).]
Motivating Scenario (2)

DSL Tooling

```java
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

```plaintext
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 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 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

```plaintext
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;
```
Grammar Definition

```plaintext
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**
- **Simple assignment**
- **Boolean assignment**
Grammar Definition

```plaintext
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

```plaintext
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

```
PollSystem --> polls
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Poll</td>
<td>questions</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Question</td>
<td>options</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td></td>
</tr>
</tbody>
</table>
```
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' name=ID '{' questions+='Question'+'}'';

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

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

```plaintext
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
;

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

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

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

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

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

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

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

Poll Quality:
- Q1:
  - Value the user experience
  - Options:
    - A: "Bad"
    - B: "Fair"
    - C: "Good"
- 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:
- Q1:
  - Value the time response
  - Options:
    - A: "Bad"
    - B: "Fair"
    - C: "Good"
Xtext, your DSL in 5' (incl. editors and serializers)

Live Demonstration
grammar org.xtext.example.mydsl.Questionnaire with org.eclipse.xtext.common.Terminals

generate questionnaire "http://www.xtext.org/example/mydsl/Questionnaire"

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

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

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

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

It is recommended to use the ANTLR 3 parser generator (BSD licence - http://www.antlr.org/license.html).

Do you agree to download it (size 1MB) from 'http://download.itemis.com/antlr-generator-3.2.0-patch.jar'? (type 'y' or 'n' and hit enter)
y
110353 [main] INFO ipse.emf.mwe.utils.GenModelHelper - Registered GenModel 'http://www.xtext.org/example/mydsl/Questionnaire' from 'platform:/resource/org.xtext.example.questionnaire/model/MyDsl.xtext'
113410 [main] INFO text.generator.junit.Junit4Fragment - generating Junit4 Test support classes
113428 [main] INFO text.generator.junit.Junit4Fragment - generating Compare Framework infrastructure
Create a new file resource.

Enter or select the parent folder:
FooQuestionnaire

File name: foo2.q

Finish
PollSystem {
  Poll p1 {
    Question q1 {
      "What is the best JavaScript framework for testing?"
      options {
        A1: "PhantomJS"
        A2: "Jasmine"
        A3: "Mocha"
        A4: "I prefer to develop my own framework"
      }
    }
  }
  Question q2 {
    "What is the best CSS preprocessor?"
    options {
      A1: "Less.js"
      A2: "Sass"
      A3: "Stylus"
      A4: "I don't care about preprocessing CSS"
    }
  }
}
Poll p2 {
  Question q1 {
    "What is the best Java framework for testing?"
    options {
      A1: "JUnit"
      A2: "Jasmine"
      A3: "I prefer to develop my own framework"
    }
  }
  Question q2 {
    "What is the best Java library for logging?"
    options {
      A1: "Log4J"
      A2: "java.util.logging"
      A3: "I don't care about logging"
    }
  }
}
}
Poll p1 {
  Question q1 {
    "What is the best JavaScript framework for testing?"
    options {
      A1: "PhantomJS"
      A2: "Jasmine"
      A3: "Mocha"
      A4: "I prefer to develop my own framework"
    }
  }
}

Question q2 {
  "What is the best CSS preprocessor?"
  options {
    A1: "Less.js"
    A2: "Sass"
    A3: "Stylus"
    A4: "I don't care about pre-processing CSS"
  }
}

Poll p2 {
  Question q1 {
    "What is the best Java framework for testing?"
    options {
      A1: "JUnit"
      A2: "Jasmine"
      A3: "I prefer to develop my own framework"
    }
  }
}

Question q2 {
  "What is the best Java library for logging?"
  options {
    A1: "Log4J"
    A2: "java.util.logging"
    A3: "I don't care about logging"
  }
}
}
Poll `p1`
  Question `q1` {
    "What is the best JavaScript framework for testing?"
    options {
      A1: "PhantomJS"
      A2: "Jasmine"
      A3: "Mocha"
      A4: "I prefer to develop my own framework"
    }
  }

Question `q2` {
  "What is the best CSS preprocessor?"
  options {
    A1: "Less.js"
    A2: "Sass"
    A3: "Stylus"
    A4: "I don't care about preprocessing CSS"
  }
}

Poll `p2`
  Question `q1` {
    "What is the best Java framework for testing?"
    options {
      A1: "JUnit"
      A2: "Jasmine"
      A3: "I prefer to develop my own framework"
    }
  }

Question `q2` {
  "What is the best Java library for logging?"
  options {
    A1: "Log4J"
    A2: "java.util.logging"
    A3: "I don't care about logging"
  }
}
From Metamodel
To
Grammar (other side)
From Metamodel to Grammar

Grammar

conforms To

Source Code A

xtext

conforms To

MetaModel

conforms To

Model A
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]
Live Demonstration
grammar org.xtext.example.mysdl.Questionnaire2 with org.eclipse.xtext.common.Terminal

import "http://www.xtext.org/example/mysdl/Questionnaire"
import "http://www.eclipse.org/emf/2002/Ecore" as.ecore
deprecated

PollSystem returns PollSystem:
  {PollSystem}
  'PollSystem'
  '{
    ('polls' '{' polls+=Poll ("," polls+=Poll)* '}' )
  };

Poll returns Poll:
  {Poll}
  'Poll'
  name=EString
  '{
    ('questions' '{' questions+=Question ("," questions+=Question)* '}' )
  };

EString returns.ecore::EString:
  STRING | ID;

Question returns Question:
  {Question}
  'Question'
  '{
    ('id' id=EString)?
    ('text' text=EString)?
    ('options' '{' options+=Option ("," options+=Option)* '}' )
  };

Option returns Option:
  {Option}
  'Option'
  '{
    ('id' id=EString)?
    ('text' text=EString)?
  };

/* automatically generated by Xtext */
Part 2: define a textual syntax for your statemachine metamodel...

```plaintext
fsm my_fsm
  state opened entry "open door"
  state [init] closed entry "close door"
  transition open closed --> opened [on]
  transition close opened --> closed [off]
```
DSL,
Model,
Metamodel,
Summary
Abstraction Gap

Problem space
domain-specific
language

Transformation

Solution space
implementation
language
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
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
conforms
To
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)*;

Grammar

MetaModel

conforms To

Source Code/Model

conforms To
Model, Metamodel, Metametamodel, DSML
**Grammar Source Code**

- **M^3**: EBNF
- **M^2**: Grammar
- **M^1**: Source Code

**Metamodel**

- **Metametamodel**: M^3
- **Metamodel**: M^2
- **Model**: M^1
Language and MDE

Grammarware

conforms To

Source Code A

Source Code B

Source Code C

Grammar

conforms To

Model A

Model B

Model C

Modelware

conforms To

Metamodel
MDE, Grammar: there and back again

- Grammar
  - conforms To
  - Source Code B
  - xtext
  - conforms To
  - Model A
  - Metamodell

- MDE, Grammar: there and back again
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?

- UML: 80%
- BPMN: 20%
- Vendor DSL: 20%
- In-house DSL: 40%
- SysML: 10%
- Matlab/Simulink: 5%
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

Structural

Behavioral
Xtext is built using MDE technologies

Xtext (and alternatives) democratize DSL development
My 3 take away 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 productivity 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

Limit Expressiveness

Viewpoints

Evolution

Learn from GPLs

Support

Tooling
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
Questions ?
Engineering Modeling Languages
Turning Domain Knowledge into Tools

Benoit Combemale
Robert B. France
Jean-Marc Jézéquel
Bernhard Rumpe
Jim Steele
Didier Vojtech

 DSL Engineering
Designing, Implementing and Using Domain-Specific Languages

Markus Voelter
with Sebastian Benz, Christian Dietrich, Birgit Engelmann
Mats Helander, Lennart Katz, Reiko Vissier, 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 Moreforskning Molde AS
NO-1757 Halden
Norway
+47 6921 5000
steinar.kristoffersen@hiiof.no

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