On Turning Domain Knowledge into Tools

Prof. Jean-Marc Jézéquel
Director of IRISA
jezequel@irisa.fr
@jmjezequel

Writing Software vs. Creating a Scientific Theory
Scientific Theories & Models

- A scientific theory is an explanation of an aspect of the natural world that can be repeatedly tested, in accordance with the scientific method, using a predefined protocol of observation and experiment
  - "The Structure of Scientific Theories" in The Stanford Encyclopedia of Philosophy

- The scientific method involves
  - the proposal and testing of hypotheses,
    - by deriving predictions from the hypotheses about the results of future experiments
  - then performing those experiments to see whether the predictions are valid

- A Model is an abstraction of an aspect of the world for a specific purpose. Therefore a Scientific Theory is a Model.
  - But a Model is not always a Scientific Theory

Creating a Scientific Theory is (evermore) Writing Software

- Mathematics used to be the language of science... … when science was simple enough
  - Newton’s gravity
    - 2 bodies problem has an analytical solution (Maths)
    - 3+ bodies problem?
  - Solution: model it into software and run it on a computer
    - aka Simulation
    - Idem for nuclear reactions, QCD, meteo, climate …

- Informatics is the language of science of the 21th
  - Of course Math still has a role to play
Conversely writing (useful) Software is like Creating a Scientific Theory

- A Machine is made of
  - A computer C
  - Model M
  - Function f

- Does it do what I want?
  - Test it w.r.t. the World!

The World and the Machine [Jackson]

- A Machine is made of
  - A computer C
  - Model M
  - Function f

- Typical evolution
  1. Model the world
  2. Monitor the world
  3. Control the world
  4. Sometimes becomes the world (but that’s not the point of this talk)

  - Bank accounts, Expedia…
Caveat

- **A Machine is made of**
  - A computer C
    - Nobody knows any longer how modern processors work
  - Model M
    - Abstraction of an aspect of the world
    - it is incomplete, partial and thus **wrong**
  - Function f
    - Users do not really know what they want
    - Many bugs traced to bad requirements

Machine C \( (M \times f) \)

World

Sensors

Actuators

Are you serious? Do you really want to board this plane?

Solution: Abstraction + Separation of Concerns

- **Of course** [Dijkstra]!
  - Abstraction on hardware
  - Models are SoC + abstraction of the world
  - Functions must be understood \( \Rightarrow \) abstracted

- **Are all these abstractions consistent?**
  - Do the thing right [Brooks]: applied maths, eg. Proofs

- **Are they close enough to reality for the purpose?**
  - Do the right thing [Brooks]: Test!
Evolution towards better abstractions

- **The historical approach (50’s->80’s)**
  - Machine = C (M x f) : M x f is « compiled »
    - Typical in Fortran, C, control automation, …
    - Most efficient, but no SoC thus brittle wrt f->f’

- **The object oriented revolution (70’s -> 2000’s)**
  - Machine = C(M) x C(f) : M x f is « interpreted » (M still there)
    - Then it makes it easy to have Machine’ = C(M) x C(f’)
  - Still hard to keep model separated from technical concerns
    - persistency, security, FT, speed…

- **One DSL (Domain Specific Language) per concern (90’s -> ?)**
  - Machine = C(M1) x C(M2) x C(M3) x C(f1) x C(f2) …

**Writing software is (will be) composing abstractions described in different languages**
Modeling and Weaving

Challenges:
- Product Families
- Reuse of Weaving Process
- Automatic Weaving

Multiple viewpoints & stakeholders

Multiple concerns (technicals…)

Multiple domains of expertise

=> Need languages to express them!
  - In a meaningful way for experts
  - With tool support (analysis, code gen., V&V..)
    - Which is still costly to build
  - At some point, all these concerns must be integrated
Example: jHipster

- **JHipster** is a development platform to generate, develop and deploy Spring Boot + Angular Web applications and Spring microservices.
- **Goal** is to generate a complete and modern Web app or microservice architecture, unifying:
  - A high-performance and robust Java stack on the server side with Spring Boot
  - A sleek, modern, mobile-first front-end with Angular and Bootstrap
  - A robust microservice architecture with JHipster Registry, Netflix OSS, ELK stack and Docker
  - A powerful workflow to build your application with Yeoman, Webpack/Gulp and Maven/Gradle
- **Use of 40+ different DSLs!**

Limits of General Purpose Languages (1)

- **Abstractions** and **notations** used are not natural/suitable for the stakeholders
  - Even with the best languages, impossible to keep all concerns separated down to the implementation
Limits of General Purpose Languages (2)

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

General-Purpose Languages

« 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

aka Domain-Specific Languages
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
- Each concern described in its own language => reduce abstraction gap
Domain Specific Languages (DSLs)

• Long history: used for almost as long as computing has been done.

• You’re using DSLs in a daily basis
  ▪ Even if you do not recognize them as DSLs (yet), because they have many different forms [Fowler]

• More and more people are building DSLs
  ▪ How can we help them?

HTML

```xml
<?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

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

.CodeMirror-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;
}
.CodeMirror-sizer {
  position: relative;
}
```

Domain: web (styling)

---

Makefile

```
PACKAGE = package
VERSION = $(date -y)
RELEASE_DIR = .
RELEASE_FILE = $(PACKAGE).$(VERSION)

# Notice that the variable LOGNAME comes from the environment in
# POSIX shells.
# 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:
  grep "^# target: " $(MAKEFILE)

# 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
Regular expression

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

Domain: strings (pattern matching)

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)
**PGN (Portable Game Notation)**

```
[Event "F/S Return Match"]
[Bite "Belgrade, Serbia Yugoslavia"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 (This opening is called the Ruy Lopez.) 3... e6
hxg5 29. b3 Re8 30. a3 Kd6 31. axb4 axb4 32. Rb5 b4 33. d4 cxb4 34. Ba2 Ke5
Kf2 42. g4 Ke3 43. Rxe6 1/2-1/2
```

**Domain: chess (games)**

- **R**: a DSL for statisticians
  - a dynamic, lazy, functional, object-oriented programming language
    - with a rather unusual combination of features
  - designed to
    - ease learning by non-programmers
    - enable rapid development of new statistical methods.
  - base is estimated to be as high as 2 million users.
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)

There are more DSLs than you think…

… because DSLs exist with different shapes
Different shapes for a DSL: External

- External DSLs with their own syntax and domain-specific tooling
  - Nice for the non-programmers
  - Good for separation of concerns
  - Bad for integration
- Example: SQL

```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
```

Different shapes for a DSL: Internal/Embedded

- Internal/Embedded DSLs, blending their syntax and semantics into host language (C++, Scala, C#)
  - Splendid for the gurus
  - Hard for the rest of us
  - Excellent integration
- Example: SQL in LINQ/C#

```csharp
// DataContext takes a connection string
DataContext db = new DataContext("c:\northwind\northwind.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);
```
Different shapes for a DSL: Implicit [Fowler]

- Implicit = from plain-old API to more fluent APIs
  - Good for Joe the Programmer
  - Bad for separation of concerns, V&V
  - Good for integration

- Example: SQL

```java
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("ch",
                                       "root");
  // create statement to execute query
  stmt = con.createStatement();
  // execute insert query
  stmt.execute(query);
  System.out.println("Data inserted in table ",
        System.out.println("Data inserted in table ");
```

C: a DSL for System Programming?

- Hum, wait a minute, is not C a GPL?
  - Yes, but where it is really good is when you write low level stuff

- But what is the difference between a GPL and a DSL then?
  - Turing incompleteness not considered anymore as a criteria
  - Not always black & white, large greyscale
Where does Software go with DSL?

- A DSL program is a 3D abstraction
  - from the domain (cf Newton vs Relativity),
  - from the function (requirements)
  - from the platform
- Embrace uncertainty
  - Smaller abstractions than with GPL
    - We better know and control the unknown
  - Apply rigorous methods to uncertain systems so that we get known uncertainties

DSLs for the masses:

Helping people turn domain knowledge into tools
Issues of DSL Engineering

*DSL is Software!*

- Versions
- Variants
- Separation of concerns / Composition

---

**Versions of a DSL: a Typical Lifecycle**

➢ **Starts as a simple ‘configuration’ mechanism**
  ▪ for a complex framework, e.g.; video processing

➢ **Grows more and more complex over time**
  ▪ `ffmpeg -i input.avi -b:v 64k -bufsize 64k output.avi`
    • Cf [https://www.ffmpeg.org/ffmpeg.html](https://www.ffmpeg.org/ffmpeg.html)

➢ **Evolves into a more complex language**
  ▪ `ffmpeg config file`
    • A preset file contains a sequence of `option=value` pairs, one for each line, specifying a sequence of options. Lines starting with the hash (`#`) character are ignored and are used to provide comments.

➢ **Add macros, if, loops,…**
  ▪ might end up into a Turing-complete language!
Variants of a DSL

- **Abstract syntax variability**
  - functional variability
    - E.g. Support for super states in StateCharts
      - 50+ variants of StateCharts Syntax have been reported!

- **Concrete syntax variability**
  - representation variability
    - E.g. Textual/Graphical/Color…

- **Semantics variability**
  - interpretation variability
    - E.g. Inner vs outer transition priority

Variants Also at Semantic Level

```
Event "e" leads to
S4 (UML), S5 (Rhapsody), or (S6) Stateflow
```

"UML vs. Classical vs. Rhapsody Statecharts: Not All Models are Created Equal ", Michelle Crane, Juergen Dingel
A (Simplified) State Machine Language Family

SSMLF

Syntax
- Textual
- Graphical

Structure
- Simple
- Hierarchical

Semantics
- InnerPriority
- OuterPriority

InitialState
- Mandatory
- Optional
- Many

hasFinalState

Gemoc Initiative

Visit http://gemoc.org

Focuses on SLE tools and methods for interoperable, collaborative, and composable modeling languages.
**DSL: From Craft to Engineering**

- **From supporting a single DSL…**
  - Concrete syntax, abstract syntax, semantics, pragmatics
    - Editors, Parsers, Simulators, Compilers…
    - But also: Checkers, Refactoring tools, Converters…

- **…To supporting Multiple DSLs**
  - Interacting altogether
  - Each DSL with several flavors (variants)
  - And evolving over time (versions)

- **Product Lines of DSLs!**
  - Safe reuse of the tool chains?
  - Backward compatibility, Migration of artifacts?

---

**Our Goal**

- Ease the definition of tool-supported DSL families
  - How to ease and validate the definition of new DSLs/tools?
  - How to correctly reuse existing tools?

⇒ Bring external DSL design abilities to the masses
  ⇒ Use abstractions that are familiar to the OO Programmer to define languages
    ⇒ set of DSL to build DSLs
  ⇒ Leverage static typing to foster safe reuse
    ⇒ With a appropriate definition of type
A tool (aka Model Transformation) is just a program working with specific OO data structures (aka meta-models) representing abstract syntax trees (graphes).

- Kermeta approach: organize the program along the OO structure of the meta-model
- Any software engineer can now build a DSL toolset!
  - No longer just for genius...

Product Lines of DSLs = SPL of OO programs

- Safe reuse of the tool chains -> Static typing
- Backward compatibility, Migration of artifacts -> Adaption
Melange: a Meta-language for Modular and Reusable Development of DSLs

with Thomas Degueule, Benoit Combemale, Arnaud Blouin, Olivier Barais

Approach Overview
Inspired by eg. Erdweg et al., Language Composition Untangled, LDTA, 2012
TOOL REUSE THROUGH MODEL TYPING

Steel et al., On Model Typing, SoSyM, 2007
Guy et al., On Model Subtyping, ECMFA, 2012
LANGUAGE DEFINITION

\[ \mathcal{L} \triangleq \langle \text{AS}, \text{Sem}, \text{MT} \rangle \]

\[ \text{Sem}(L) \triangleq (A'_i \in \text{Aspects}) \text{ where } \]
\[ \forall A_i \in \text{Sem}(L), \exists c \in \text{AS}(L) : c \text{ match } t \]
\[ \forall A_i, A_j \in \text{Sem}(L) : A'_i \cap A'_j \Rightarrow i \neq j \]

\[ \text{Sem} \bullet \text{Sem}' \triangleq \text{Sem} \cup \text{Sem}' \]

\[ \text{sig}(\text{Sem}) \triangleq \bigcup_{A_i \in \text{Sem}} \text{sig}(A'_i) \]

\[ \text{MT}(L) \triangleq \text{AS}(L) \circ \text{sig}(\text{Sem}(L)) \]

\[ \mathcal{L} \leftarrow \text{AS}' = \langle \text{AS} \circ \text{AS}', \text{Sem}, \text{MT} \circ \text{AS}' \rangle \]

\[ \mathcal{L} \leftarrow \text{Sem}' = \langle \text{AS}, \text{Sem} \bullet \text{Sem}', \text{MT} \circ \text{sig}(\text{Sem}') \rangle \]

\[ \mathcal{L} \leftarrow \mathcal{L}' = \langle \text{AS} \circ \text{AS}', \text{Sem} \bullet \text{Sem}', \text{MT} \circ \text{MT}' \rangle \]

\[ \mathcal{L} \leftarrow \mathcal{L}' = \langle \text{AS} \circ \text{AS}', \text{Sem} \bullet \text{Sem}', \text{MT} \circ \text{MT}' \rangle \]

\[\]

\[ \mathcal{L} \triangleq \langle \text{AS}, \text{Sem}, \text{MT} \rangle \]

\[ \text{linguage} \text{Fsm} \{ \]
\[ \text{syntax} \ '\text{FSM.ecore}' \]
\[ \}

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]
LANGUAGE DEFINITION

\[ L \triangleq \{ AS, Sem, MT \} \]

\[ Sem(L) \triangleq (A_i \in \text{Aspects}) \text{ where} \]
\[ \forall A_i \in \text{Sem}(L), \exists c \in AS(L) : c \text{ match } t \]
\[ \forall A_i, A_j \in \text{Sem}(L) : A_i \neq A_j \implies i > j \]

\[ \text{Sem} \cdot \text{Sem'} \equiv \text{Sem} - \text{Sem'} \]

\[ \text{sig(Sem)} \triangleq \bigcup_{A_i \in \text{Sem}} \text{sig}(A_i) \]

\[ MT(L) \triangleq \text{AS}(L) \circ \text{sig(Sem}(L)) \]

\[ L \xleftarrow{\text{exec}} AS' = (AS \circ AS', Sem, MT \circ AS') \]

\[ L \xleftarrow{\text{exec}} \text{Sem'} = (AS, \text{Sem} \cdot \text{Sem'}, MT \circ \text{sig(Sem')}) \]

\[ L \circ L' = (AS \circ AS', \text{Sem} \cdot \text{Sem'}, MT \circ MT') \text{ where} \]
\[ MT'' = MT \circ MT' \text{ and} \]
\[ MT' \subseteq MT'' \]

\[ A^L_{\text{exec}}(c) = (AS_2, \text{Sem}_2, MT_2), \text{ where:} \]
\[ AS_2 \triangleq \lambda c. (AS_1(c), c, AS_2 \subseteq AS_1) \]
\[ \text{Sem}_2 \triangleq \{ A_i \in \text{Sem}_1, fp(A_i' \cdot AS_1) \subseteq AS_1 \} \]
\[ MT_2 \subseteq MT' \]

\[ L \xleftarrow{\text{exec}} \text{Fsm} \{ \]
\[ \text{syntax } 'FSM.ecore' \]
\[ \text{with } \text{ExecutableFsm} \]
\[ \text{with } \text{ExecutableState} \]
\[ \text{with } \text{ExecutableTransition} \]
\[ \}

\[ \text{exec}() \]

\[ \text{current} \]

\[ \text{step}() \]

\[ \text{fire}() \]

language Fsm {

\[ \text{syntax } 'FSM.ecore' \]

\[ \text{with } \text{ExecutableFsm} \]

\[ \text{with } \text{ExecutableState} \]

\[ \text{with } \text{ExecutableTransition} \]

\[ \text{exactType } \text{FsmMT} \]
SYNTAX MERGING

\[ \mathcal{L} \triangleq (\mathcal{AS}, \mathcal{Sem}, \mathcal{MT}) \]

\[ \mathcal{Sem}(\mathcal{L}) \triangleq (\mathcal{A}_i^r \in \mathcal{Aspects}) \text{ where} \]
\[ \forall \mathcal{A}_i \in \mathcal{Sem}(\mathcal{L}), \exists e \in \mathcal{AS}(\mathcal{L}) : e \text{ match } t \]
\[ \forall \mathcal{A}_i, \mathcal{A}_j \in \mathcal{Sem}(\mathcal{L}) : \mathcal{A}_i^r \odot \mathcal{A}_j^r \implies i > j \]

\[ \mathcal{Sem} \bullet \mathcal{Sem}' \equiv \mathcal{Sem} \cup \mathcal{Sem}' \]

\[ \sigma(\mathcal{Sem}) \triangleq \bigcup_{\mathcal{A}_i \in \mathcal{Sem}} \sigma(\mathcal{A}_i^r) \]

\[ MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \odot \sigma(\mathcal{Sem}(\mathcal{L})) \]

\[ \mathcal{L} \overset{\text{m}}{\rightarrow} \mathcal{AS'} = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem}, \mathcal{MT} \odot \mathcal{MT'}) \]

\[ \mathcal{L} \odot \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \odot \mathcal{MT}') \]

\[ \mathcal{L} \triangleright \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \odot \mathcal{MT'}) \]

\[ \mathcal{L} \rightharpoondown \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \rightharpoondown \mathcal{MT'}) \]

\[ \mathcal{L} \circ \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \odot \mathcal{MT'}) \]

\[ \mathcal{L} \triangleright \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \triangleright \mathcal{MT'}) \]

\[ \mathcal{L} \rightharpoondown \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \rightharpoondown \mathcal{MT'}) \]

\[ \mathcal{L} \rightharpoondown \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \rightharpoondown \mathcal{MT'}) \]

\[ \mathcal{L} \rightharpoondown \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \rightharpoondown \mathcal{MT'}) \]

\[ \mathcal{L} \rightharpoondown \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \rightharpoondown \mathcal{MT'}) \]

\[ \mathcal{L} \rightharpoondown \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \rightharpoondown \mathcal{MT'}) \]

\[ \mathcal{L} \rightharpoondown \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \rightharpoondown \mathcal{MT'}) \]

\[ \mathcal{L} \rightharpoondown \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \rightharpoondown \mathcal{MT'}) \]

\[ \mathcal{L} \rightharpoondown \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \rightharpoondown \mathcal{MT'}) \]

\[ \mathcal{L} \rightharpoondown \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \rightharpoondown \mathcal{MT'}) \]

\[ \mathcal{L} \rightharpoondown \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \rightharpoondown \mathcal{MT'}) \]

\[ \mathcal{L} \rightharpoondown \mathcal{L}' = (\mathcal{AS} \odot \mathcal{AS'}, \mathcal{Sem} \circ \mathcal{Sem}', \mathcal{MT} \rightharpoondown \mathcal{MT'}) \]
```
language GuardedFsm {
  syntax 'FSM.ecore'
  syntax 'Guard.ecore'
  with ExecutableFsm
  with ExecutableState
  with ExecutableTransition
  with EvaluateGuard

  exactType GuardedFsmMT
}
```
LANGUAGE MERGING

\[ \mathcal{L} \triangleq \{ \text{AS, Sem, MT} \} \]

Sem(C) \triangleq \{ A_i \mid i \in \text{Aspects} \}

\[ \forall A_i \in \text{Sem}(C), \exists c \in \text{As}(C) : c \text{ match } t \]

\[ \forall A_i, A_j \in \text{Sem}(C) : A_i \neq A_j \implies i > j \]

\[ \text{Sem} \cdot \text{Sem}' = \text{Sem} \cdot \text{Sem}' \]

\[ \text{sig}(\text{Sem}) \triangleq \bigcup_{A_i \in \text{Sem}} \text{sig}(A_i) \]

\[ \text{MT}(\mathcal{L}) \triangleq \text{AS}(\mathcal{L}) \cdot \text{sig}(\text{Sem}(\mathcal{L})) \]

\[ \mathcal{L} \overset{\rightarrow}{\sim} \mathcal{L}' = (\text{AS} \cdot \text{AS}', \text{Sem}, \text{MT} \cdot \text{AS}') \]

\[ \mathcal{L} \overset{\sim}{\sim} \text{Sem}' = (\text{AS} \cdot \text{Sem} \cdot \text{Sem}', \text{MT} \cdot \text{AS}') \]

\[ \mathcal{L} \overset{\to}{\sim} \mathcal{L}' = (\text{AS} \cdot \text{AS}', \text{Sem} \cdot \text{Sem}', \text{MT} \cdot \text{MT}') \]

\[ \text{MT}'' \triangleq \text{MT} \cdot \text{MT}' \]

\[ \Lambda^n_{\text{L}}(\text{L}, c) = \{ A_2, \text{Sem}_2, \text{MT}_2 \}, \text{where:} \]

\[ \text{AS}_2 \triangleq \Lambda^n_{\text{A}}(\text{AS}_1, c), \text{AS}_2 \subseteq \text{AS}_1, \]

\[ \text{Sem}_2 \triangleq \{ A_i \in \text{Sem}_1, f_p(A_i, \text{AS}_1) \subseteq \text{AS}_2 \}, \text{MT}_2 \triangleq \text{MT}_1 \]

\[ \text{language} \text{ Building } \{
\text{syntax} \text{ `'Building.ecore' with SimulatorAspect...}
\text{exactType} \text{ BuildingMT}
\}
\]
**LANGUAGE MERGING**

\[ \mathcal{L} \triangleq (\text{AS}, \text{Sem}, \text{MT}) \]

\[ \text{Sem}(\mathcal{L}) \triangleq (A'_0 \in \text{Aspects}) \text{ where} \]

\[ \forall A'_i \in \text{Sem}(\mathcal{L}), \exists e \in \text{AS}(\mathcal{L}) : e \text{ match } t \]

\[ \forall A'_i, A'_j \in \text{Sem}(\mathcal{L}) : A'_i \neq A'_j \implies i > j \]

\[ \text{Sem} \sqcup \text{Sem}' = \text{Sem} \sqcup \text{Sem}' \]

\[ \operatorname{sig}(\text{Sem}) \triangleq \bigcup_{A'_i \in \text{Sem}} \operatorname{sig}(A'_i) \]

\[ \mathcal{MT}(\mathcal{L}) \triangleq \text{AS}(\mathcal{L}) \circ \text{sig}(\text{Sem}(\mathcal{L})) \]

\[ \mathcal{L} \leftarrow \mathcal{AS}' = (\text{AS} \circ \text{AS}', \text{Sem}, \text{MT} \circ \text{AS}') \]

\[ \mathcal{L} \leftarrow \mathcal{Sem}' = (\text{AS}, \text{Sem} \circ \text{Sem}', \text{MT} \circ \text{sig}(\text{Sem}')) \]

\[ \mathcal{L} \sqcup \mathcal{L}' = (\text{AS} \circ \text{AS}', \text{Sem} \circ \text{Sem}, \text{MT} \circ \text{MT}') \]

\[ \mathcal{L} \sqcup \mathcal{L}' = (\text{AS} \circ \text{AS}', \text{Sem} \circ \text{Sem}, \text{MT} \circ \text{MT}') \]  

\[ \chi^o(L_1, c) = (\text{AS}_2, \text{Sem}_2, \text{MT}_2) \text{, where:} \]

\[ \text{AS}_2 \triangleq \chi^o(\text{AS}_1, c), \text{AS}_2 \subseteq \text{AS}_1, \]

\[ \text{Sem}_2 \triangleq \{ A'_i \in \text{Sem}_1, fp(A'_i, \text{AS}_1) \subseteq \text{AS}_2 \}, \]

\[ \text{MT}_2 \subset \text{MT}_1 \]

**LANGUAGE INHERITANCE**

\[ \mathcal{L} \triangleq (\text{AS}, \text{Sem}, \text{MT}) \]

\[ \text{Sem}(\mathcal{L}) \triangleq (A'_0 \in \text{Aspects}) \text{ where} \]

\[ \forall A'_i \in \text{Sem}(\mathcal{L}), \exists e \in \text{AS}(\mathcal{L}) : e \text{ match } t \]

\[ \forall A'_i, A'_j \in \text{Sem}(\mathcal{L}) : A'_i \neq A'_j \implies i > j \]

\[ \text{Sem} \sqcup \text{Sem}' = \text{Sem} \sqcup \text{Sem}' \]

\[ \operatorname{sig}(\text{Sem}) \triangleq \bigcup_{A'_i \in \text{Sem}} \operatorname{sig}(A'_i) \]

\[ \mathcal{MT}(\mathcal{L}) \triangleq \text{AS}(\mathcal{L}) \circ \text{sig}(\text{Sem}(\mathcal{L})) \]

\[ \mathcal{L} \leftarrow \mathcal{AS}' = (\text{AS} \circ \text{AS}', \text{Sem}, \text{MT} \circ \text{AS}') \]

\[ \mathcal{L} \leftarrow \mathcal{Sem}' = (\text{AS}, \text{Sem} \circ \text{Sem}', \text{MT} \circ \text{sig}(\text{Sem}')) \]

\[ \mathcal{L} \sqcup \mathcal{L}' = (\text{AS} \circ \text{AS}', \text{Sem} \circ \text{Sem}, \text{MT} \circ \text{MT}') \]

\[ \mathcal{L} \sqcup \mathcal{L}' = (\text{AS} \circ \text{AS}', \text{Sem} \circ \text{Sem}, \text{MT} \circ \text{MT}') \]  

\[ \chi^o(L_1, c) = (\text{AS}_2, \text{Sem}_2, \text{MT}_2) \text{, where:} \]

\[ \text{AS}_2 \triangleq \chi^o(\text{AS}_1, c), \text{AS}_2 \subseteq \text{AS}_1, \]

\[ \text{Sem}_2 \triangleq \{ A'_i \in \text{Sem}_1, fp(A'_i, \text{AS}_1) \subseteq \text{AS}_2 \}, \]

\[ \text{MT}_2 \subset \text{MT}_1 \]
**LANGUAGE INHERITANCE**

\[ \mathcal{L} \triangleq (\mathcal{A}, \mathcal{S}, \mathcal{M}, \mathcal{T}) \]

\[ \text{Sem}(\mathcal{L}) \triangleq (\mathcal{A}', \mathcal{S}', \mathcal{M}, \mathcal{T}) \text{ where} \]

\[ \forall \mathcal{A} \in \text{Sem}(\mathcal{L}), \exists \mathcal{A} \in \mathcal{A} : \mathcal{A} \to \mathcal{A} \]

\[ \mathcal{S} \cdot \mathcal{S}' = \mathcal{S} \cdot \mathcal{S}' \]

\[ \text{Sem} \cdot \mathcal{S}' = \text{Sem} \cdot \mathcal{S}' \]

\[ \text{sig}(\text{Sem}) \triangleq \bigcup_{\mathcal{A} \in \text{Sem}} \text{sig}(\mathcal{A}) \]

\[ \mathcal{M}(\mathcal{L}) \triangleq \mathcal{A}(\mathcal{L}) \circ \text{sig}(\text{Sem}(\mathcal{L})) \]

\[ \mathcal{L} \leftarrow \mathcal{A} ' = (\mathcal{A} \circ \mathcal{A}', \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S}' = (\mathcal{A} \circ \mathcal{S}', \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S} = (\mathcal{A} \circ \mathcal{S}, \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S} = (\mathcal{A} \circ \mathcal{S}, \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \text{MT}(\mathcal{L}) = \mathcal{M}(\mathcal{L}) \circ \text{sig}(\text{Sem}(\mathcal{L})) \]

\[ \mathcal{L} \leftarrow \mathcal{A} ' = (\mathcal{A} \circ \mathcal{A}', \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S}' = (\mathcal{A} \circ \mathcal{S}', \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S} = (\mathcal{A} \circ \mathcal{S}, \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S} = (\mathcal{A} \circ \mathcal{S}, \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{A} ' = (\mathcal{A} \circ \mathcal{A}', \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S}' = (\mathcal{A} \circ \mathcal{S}', \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S} = (\mathcal{A} \circ \mathcal{S}, \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S} = (\mathcal{A} \circ \mathcal{S}, \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{A} ' = (\mathcal{A} \circ \mathcal{A}', \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S}' = (\mathcal{A} \circ \mathcal{S}', \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S} = (\mathcal{A} \circ \mathcal{S}, \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S} = (\mathcal{A} \circ \mathcal{S}, \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{A} ' = (\mathcal{A} \circ \mathcal{A}', \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S}' = (\mathcal{A} \circ \mathcal{S}', \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]

\[ \mathcal{L} \leftarrow \mathcal{S} = (\mathcal{A} \circ \mathcal{S}, \mathcal{S}, \mathcal{M}, \mathcal{T} \circ \mathcal{T'}) \]
**LANGUAGE SLICING**

\[ \mathcal{L} \triangleq (\text{AS, Sem, MT}) \]

\[ \text{Sem}({\mathcal{L}}) \triangleq (A'_i \in \text{Aspects}) \text{ where } \]

\[ \forall A_i \in \text{Sem}({\mathcal{L}}), \exists A_i \in \text{AS}({\mathcal{L}}): a \text{ match } t \]

\[ \forall A_i, A'_i \in \text{Sem}({\mathcal{L}}): A'_i \neq A_i \implies i > j \]

\[ \text{Sem} \circ \text{sem} \triangleq \text{Sem} \circ \text{Sem} \]

\[ \text{sig}(\text{Sem}) \triangleq \bigcup_{A_i \in \text{Sem}} \text{sig}(A_i) \]

\[ \text{MT}({\mathcal{L}}) \triangleq \text{AS}({\mathcal{L}}) \circ \text{sig}(\text{Sem}({\mathcal{L}})) \]

\[ \mathcal{L} \subseteq \text{AS}' = (\text{AS} \circ \text{AS}', \text{Sem}, \text{MT} \circ \text{AS}') \]

\[ \mathcal{L} \subseteq \text{Sem}' = (\text{AS} \circ \text{Sem} \circ \text{sem}', \text{MT} \circ \text{sig}(\text{Sem}') \]

\[ \mathcal{L} \subseteq \mathcal{L}' = (\text{AS} \circ \text{AS}', \text{Sem} \circ \text{sem}', \text{MT} \circ \text{MT}') \]

\[ \text{MT}' = \text{MT} \circ \text{MT}' \]

\[ \text{MT}' \subseteq \text{MT} \]

\[ \mathcal{L}'(\mathcal{L}, c) = (\text{AS}_2, \text{Sem}_2, \text{MT}_2), \text{where:} \]

\[ \text{AS}_2 \triangleq \lambda_i \text{AS}_1' \text{, } \text{AS}_2 \subseteq \text{AS}_1 \]

\[ \text{Sem}_2 \triangleq \{ A_i \in \text{sem}_1 : \text{fp}(A_i', \text{AS}_1) \subseteq \text{AS}_2 \} \]

\[ \text{MT}_1 \subseteq \text{MT}_2 \]

**Melange: a Meta-language for Modular and Reusable Development of DSLs**

**LANGUAGE SLICING**

\[ \mathcal{L} \triangleq (\text{AS, Sem, MT}) \]

\[ \text{Sem}({\mathcal{L}}) \triangleq (A'_i \in \text{Aspects}) \text{ where } \]

\[ \forall A_i \in \text{Sem}({\mathcal{L}}), \exists A_i \in \text{AS}({\mathcal{L}}): a \text{ match } t \]

\[ \forall A_i, A'_i \in \text{Sem}({\mathcal{L}}): A'_i \neq A_i \implies i > j \]

\[ \text{Sem} \circ \text{sem} \triangleq \text{Sem} \circ \text{Sem} \]

\[ \text{sig}(\text{Sem}) \triangleq \bigcup_{A_i \in \text{Sem}} \text{sig}(A_i) \]

\[ \text{MT}({\mathcal{L}}) \triangleq \text{AS}({\mathcal{L}}) \circ \text{sig}(\text{Sem}({\mathcal{L}})) \]

\[ \mathcal{L} \subseteq \text{AS}' = (\text{AS} \circ \text{AS}', \text{Sem}, \text{MT} \circ \text{AS}') \]

\[ \mathcal{L} \subseteq \text{Sem}' = (\text{AS} \circ \text{Sem} \circ \text{sem}', \text{MT} \circ \text{sig}(\text{Sem}') \]

\[ \mathcal{L} \subseteq \mathcal{L}' = (\text{AS} \circ \text{AS}', \text{Sem} \circ \text{sem}', \text{MT} \circ \text{MT}') \]

\[ \text{MT}' = \text{MT} \circ \text{MT}' \]

\[ \text{MT}' \subseteq \text{MT} \]

\[ \mathcal{L}'(\mathcal{L}, c) = (\text{AS}_2, \text{Sem}_2, \text{MT}_2), \text{where:} \]

\[ \text{AS}_2 \triangleq \lambda_i \text{AS}_1' \text{, } \text{AS}_2 \subseteq \text{AS}_1 \]

\[ \text{Sem}_2 \triangleq \{ A_i \in \text{sem}_1 : \text{fp}(A_i', \text{AS}_1) \subseteq \text{AS}_2 \} \]

\[ \text{MT}_1 \subseteq \text{MT}_2 \]
**LANGUAGE SLICING**

\[ \mathcal{L} \models (AS, Sem, MT) \]

\[ Sem(\mathcal{L}) \models (A'_i \in \text{Aspects}) \text{ where} \]
\[ \forall A_i \in \text{Sem}(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t \]
\[ \forall A_i, A'_j \in \text{Sem}(\mathcal{L}) : A'_i \neq A'_j \implies i > j \]

\[ Sem \circ Sem' \equiv \text{Sem}^{-} \circ \text{Sem}' \]

\[ \text{sig}(\text{Sem}) \models \bigcup_{A_i \in \text{Sem}} \text{sig}(A'_i) \]

\[ MT(\mathcal{L}) \models AS(\mathcal{L}) \circ \text{sig}(\text{Sem}(\mathcal{L})) \]

\[ \mathcal{L} = AS' = (AS \circ AS', Sem, MT \circ AS') \]

\[ \text{guard} \left( \frac{\text{Expr}}{\text{UnaryExp} \circ \text{Unary}} \right) \]

**Melange**

A Language Workbench
MELANGE

• An open-source (EPL) language workbench
• or… a language-based, model-oriented language for DSL engineering
• An implementation of the algebra
• Supported by a model-oriented type system
• Based on Xtext
• Seamlessly integrated with the EMF ecosystem
• Bundled as a set of Eclipse plug-ins

Implementation Choices

• Abstract syntax: Ecore (EMOF)
• Merging: Customized UML PackageMerge\(^1\)
  ▪ Trading UML specificities with EMOF specificities
  ▪ Support for renaming
• Slicing: Kompren\(^2\)
• Operational semantics: K3 (Xtend on steroids)

\(^1\)Dingel et al., Understanding and Improving UML PackageMerge, SoSyM, 2008
\(^2\)Blouin et al., Kompren: Modeling and generating model slicers, SoSyM, 2012
Compilation Scheme

OK, but does it work in practice?

*Talk is cheap, show me the code!*

[Linus Torvalds]
Farming System Modeling (/ Software Defined Farming)

https://github.com/gemoc/farmingmodeling

Water flood prediction
High Performance Computing

**Nablab Environment**
Technologies: Eclipse EMF, Xtext, Sirius
Contributors: MPO, BL, BC

**Nabla Language**
Technologies: Flex, Bison, C++
Contributors: JSC, BL
Open Challenges

- Diversity/complexity of DSL relationships
  - Far beyond structural/behavioral alignment, refinement, decomposition
  - Separation of concerns vs. Zoom-in/Zoom-out
- Live and collaborative (meta)modeling
  - Minimize the round trip between the DSL specification, the model, and its application (interpretation/compilation)
  - Model experiencing environments (MEEs): what-if/for scenarios, trade-off analysis, design-space exploration
- Integration of analysis and predictive models into DSL semantics
  - Towards unpredictable languages
    - Specify the correctness envelope to avoid over-specification
    - Identify plastic computation zones
    - Vary the execution flow of the program

Conclusion

- From supporting a single DSL…
  - Concrete syntax, abstract syntax, semantics, pragmatics
    - Editors, Parsers, Simulators, Compilers…
    - But also: Checkers, Refactoring tools, Converters…
- …To supporting Multiple DSLs
  - Interacting altogether
  - Each DSL with several flavors: families of DSLs
  - And evolving over time
- Product Lines of DSLs
  - Share and reuse assets: metamodels and transformations
Acknowledgement

- All these ideas have been developed with my colleagues of the DiverSE team at IRISA/Inria

DiverSE
Diversity-Centric Software Engineering

Formely known as Triskell