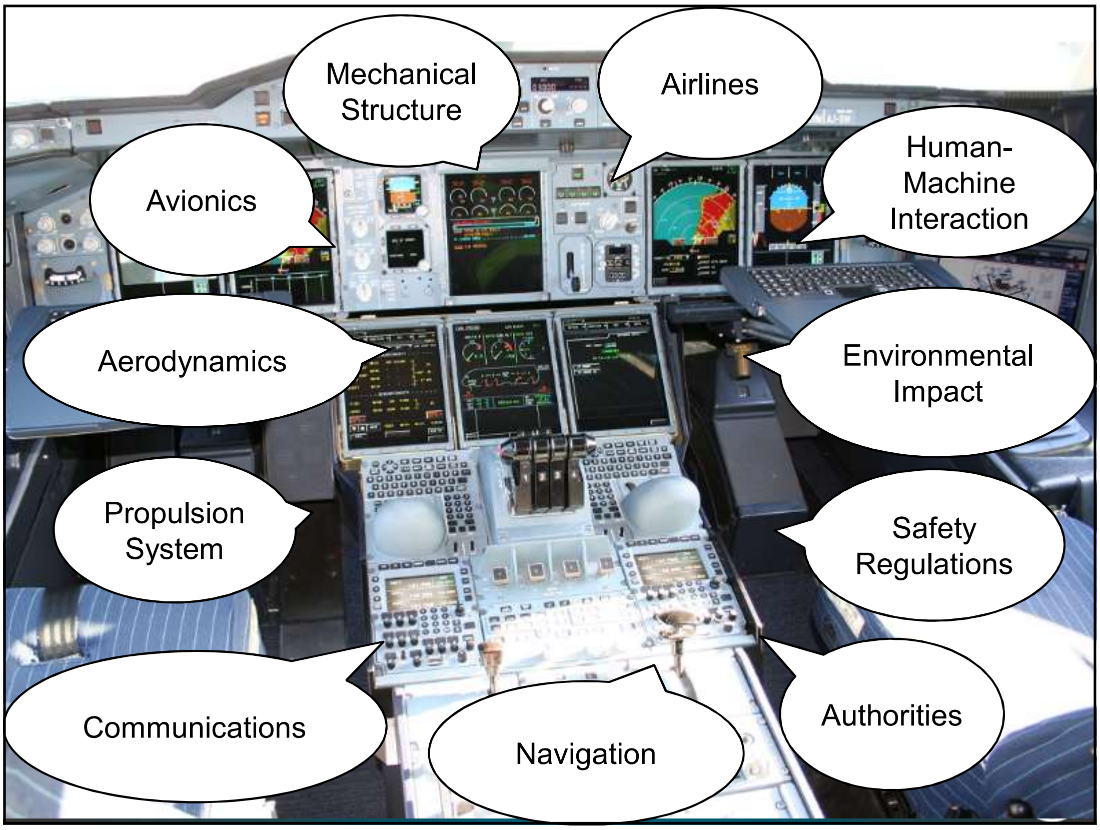
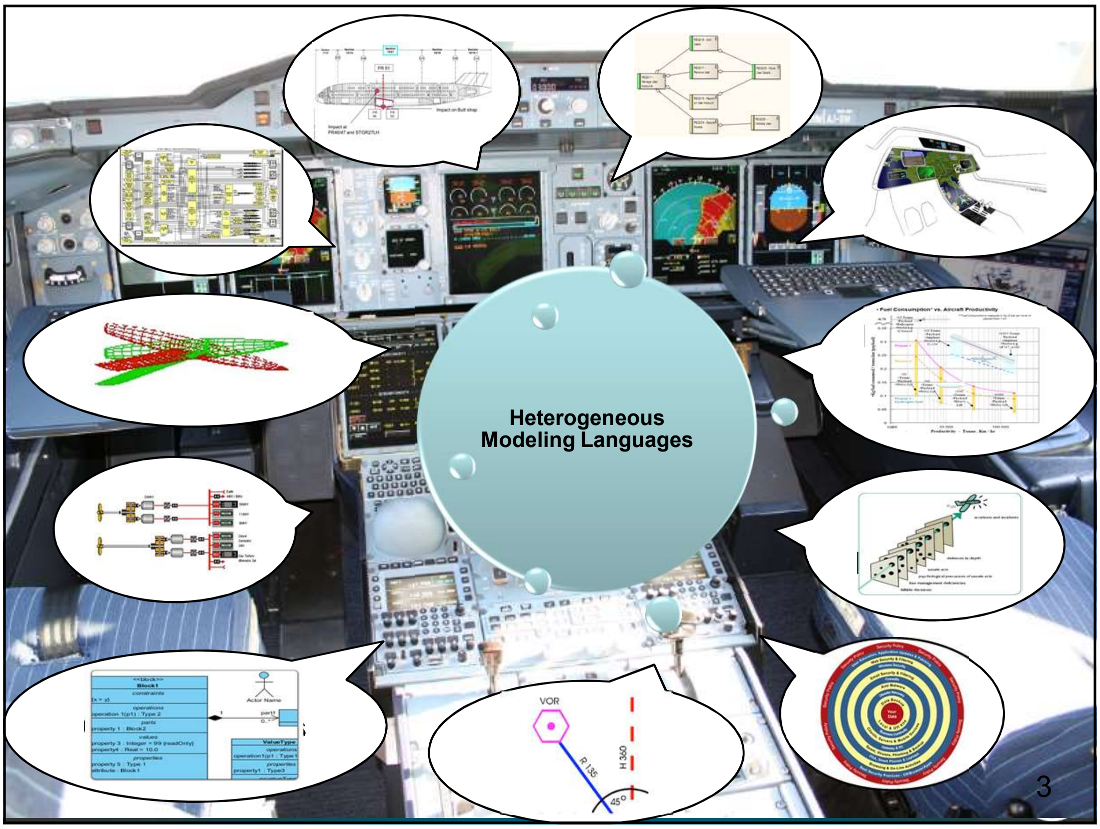
UMR IRISA

DiverSE
Diversity-Centric Software Engineering

Families of Domain Specific Languages

Prof. Jean-Marc Jézéquel
Director of IRISA
jezequel@irisa.fr
http://people.irisa.fr/Jean-Marc.Jezequel



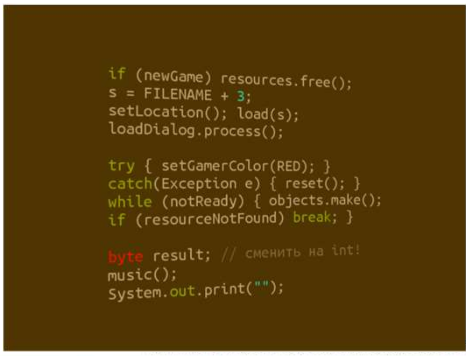


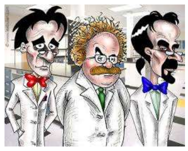
Complex Software Intensive Systems

- Multiple concerns
- Multiple viewpoints & stakeholders
- 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

Limits of General Purpose Languages (1)

- **Abstractions** and **notations** used are not natural/suitable for the stakeholders
  - Even with the best OO languages, impossible to keep all concerns separated down to the implemeation





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 »

aka **Domain-Specific Languages**

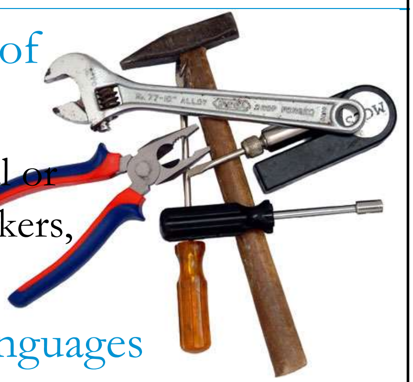
1972

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

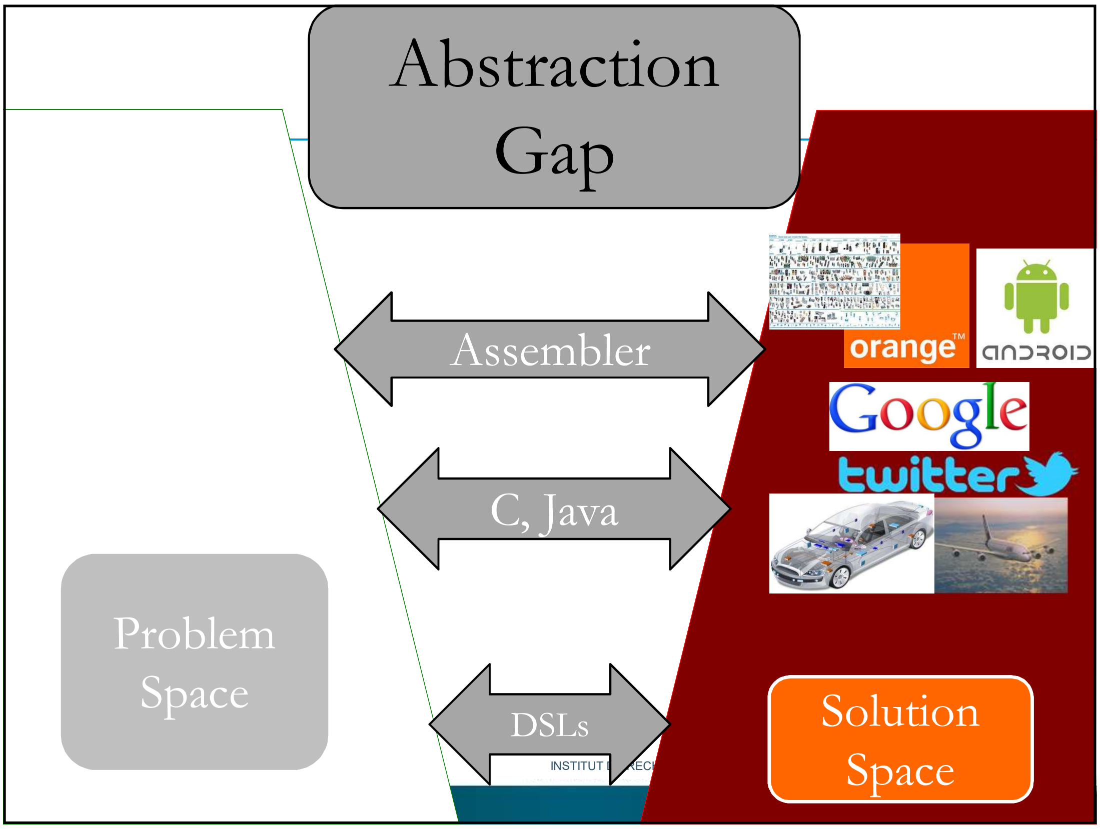
IRISA
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES
15/11/2017
7

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



IRISA
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES
15/11/2017
8



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
- **More and more people are building DSLs**
  - **Ho can we help them?**

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

```
.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

```
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)

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

15/11/2017 13

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

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

15/11/2017 14

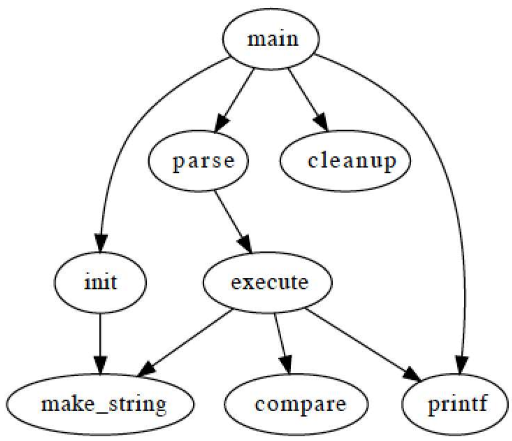
Lighthttpd configuration file

```
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)

PGN (Portable Game Notation)

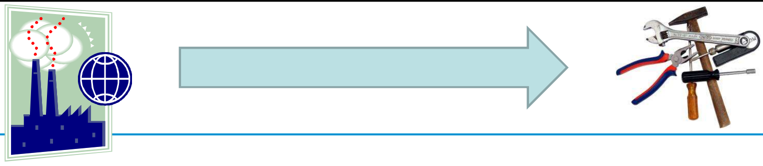
```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia Yugoslavia|JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 {This opening is called the Ruy Lopez.} 3... a6
4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6 8. c3 O-O 9. h3 Nb8  10. d4 Nbd7
11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15. Nb1 h6 16. Bh4 c5 17. dxe5
Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21. Nc4 Nxc4 22. Bxc4 Nb6
23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+ 26. Qxe1 Kxf7 27. Qe3 Qg5 28. Qxg5
hxg5 29. b3 Ke6 30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5
35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5 40. Rd6 Kc5 41. Ra6
Nf2 42. g4 Bd3 43. Re6 1/2-1/2
```

Domain: chess (games)

Regular expression

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

Domain: strings (pattern matching)



Issues of DSL Engineering

- Versions
- Variants
- Separation of concerns / Composition

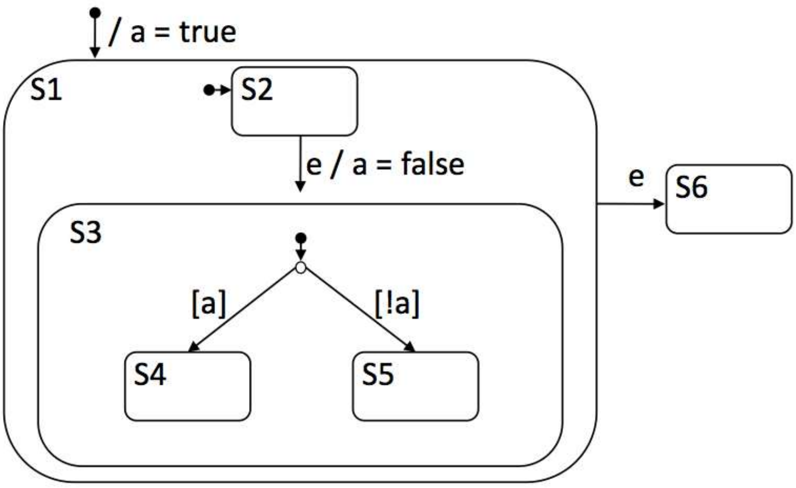
Versions of a DSL: a Typical Lifecycle

- Starts as a simple ‘configuration’ mecanism
  - 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
- 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

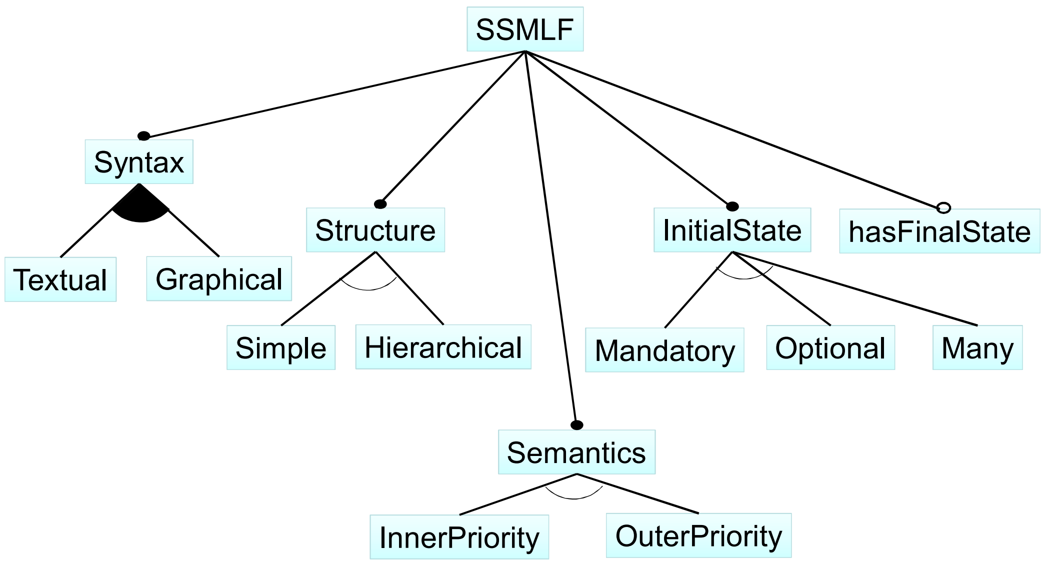
Variants Also at Semantic Level



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



Different shapes for a DSL: External

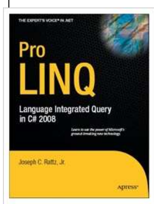
- External DSLs with their own syntax and domain-specific tooling
  - Nice for the non-programmers
  - Good for separation of concerns
  - Bad for integration
- Example: SQL

```
-- 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#

```
// 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);
```



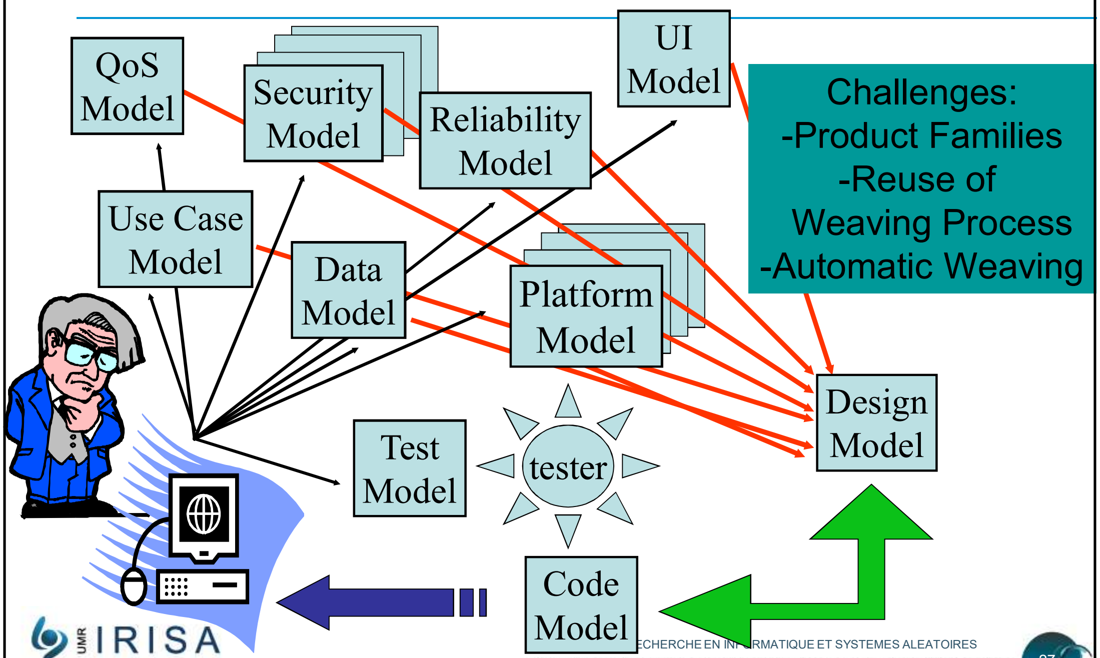
Different shapes for a DSL: Implicit

- 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

```
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("che
   "root");

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

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

```
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();
```

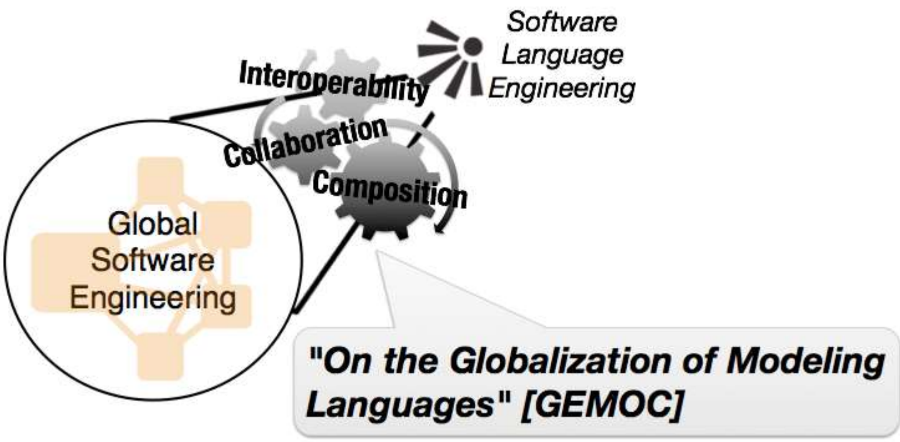
SoC: Modeling and Weaving



Gemoc Initiative

➠Visit http://gemoc.org

Focuses on **SLE tools and methods for** **<u>interoperable</u>**, **<u>collaborative</u>**, **and** **<u>composable</u>** **<u>modeling languages</u>**



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?

My 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

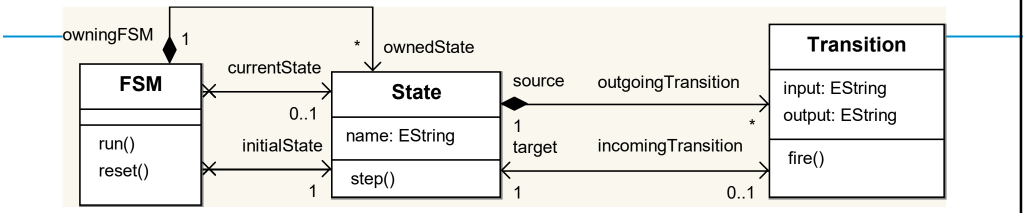
⇒Use abstractions that are familiar to the OO Programmer to define languages

⇒ set of DSL to build DSLs

⇒ Leverage static typing to foster safe reuse

⇒With a appropriate definition of type

Kermeta: Executable Meta-Modeling for the masses



// MyKermetaProgram.kmt
// An E-MOF metamodel is an OO program that does nothing

require "StateMachine.ecore" *// to import it in Kermeta*

// Kermeta lets you weave in **aspects**

// Contracts (OCL WFR)
require "StaticSemantics.ocl"
// Method bodies (Dynamic semantics)
require "DynamicSemantics.xtend"
// Transformations

Context FSM
inv: ownedState->forAll(s1,s2|
s1.name=s2.name implies s1=s2)

```
class FSM {
  public def void reset() {
    currentState = initialState
```

```
class Minimizer {
  public def FSM minimize (source: FSM) {…}
}
```


Tools built with MDE

- 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

Type Systems

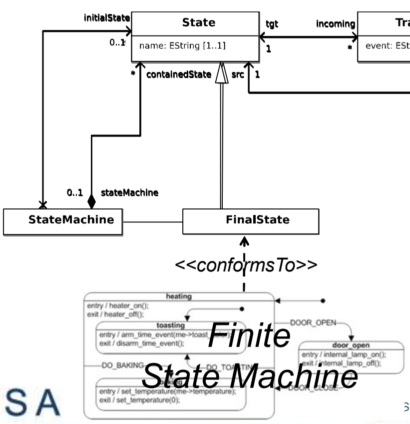
- Type systems provide unified frameworks enabling many **facilities**:
  - Abstraction
  - Reuse and safety
  - Impact analyses
  - Auto-completion
  - …
- What about a model-oriented type system?

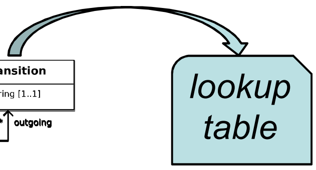
Model Type – motivation

- Motivating example: model transformation [*SoSyM'07*]

takes as input a state machine and produces a lookup table showing the correspondence between the current state, an arriving event, and the resultant state

⇒ side-effect free





➔When can we reuse such a transformation?

Model Type – motivation

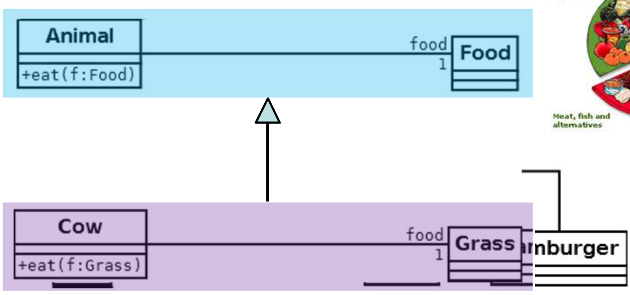
- Issue when considering a model as a set of objects:
  - addition of a property to a class is a common evolution seen in metamodels
  - property = pair of accessor/mutator methods

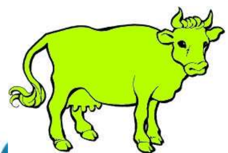
⇒ subtyping for classes requires invariance of property types!!!

⇒ Indeed: adding a property will cause a covariant property type redefinition somewhere in the metamodel.

Class Matching [Bruce et al., ENTCS 1999]

- Substitutability of type groups cannot be achieved through object subtyping





```
Animal a = Animal.new
Food f = Food.new
a.eat(f)
```

Model Type – motivation

- Some (other) differences for objects in MOF:
  - Multiplicities on properties
  - Properties can be combined to form associations: makes checking cyclical
  - Need to check whether properties are reflexive or not
  - Containment (or not) on properties

Model Type – initial implementation

- Bruce has defined the matching relation (<#) between two type groups as a function of the object types which they contain
- Generalizing his definition to the `matching` relation between model type:

Model Type M' <# M iff for each object type C in M there is a corresponding object type with the same name in M' such that every property and operation in $M.C$ also occurs in $M'.C$ with exactly the same signature as in $M.C$.

- `matching` ≅ `subtyping` (by group)

Application to MOF-Class Matching

- C1 matches C2 (C1 <# C2) iff:
  - Same names

Same names

If C1 is abstract, it can only match another abstract class

∀ C2 operation, C1 must have a

∀ C2 property, C1 must have a corresponding property

-With the same name

-With covariant type

-With the same multiplicities

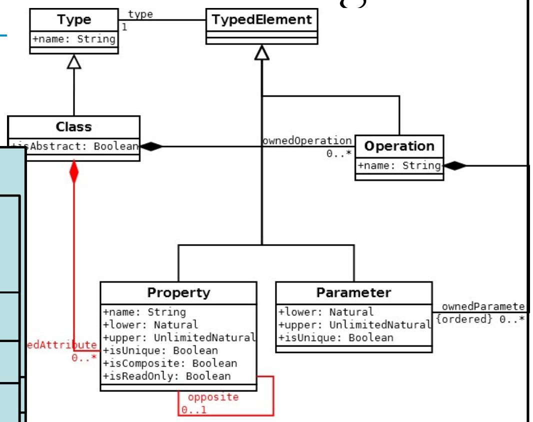
-With the same isUnique

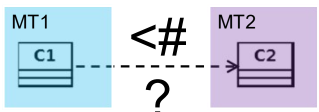
-With the same isComposite

-With an opposite with the same name

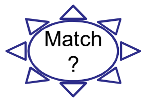
Every mandatory property in C1 must correspond to a C2 property

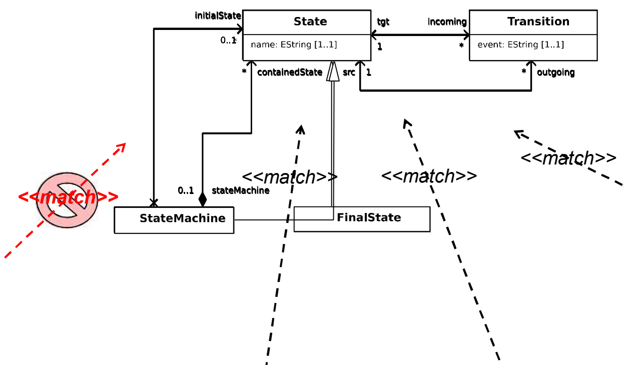
another read only property





Model Type – initial implementation





↱ matches →	Simple	Multiple-Start	Mandatory-Start	Composite	With-Final-States
Simple (Figure 4)	✓	NO	NO	NO	NO
Multiple-Start (Figure 5)	NO	✓	NO	NO	NO
Mandatory-Start (Figure 6)	✓	NO	✓	NO	NO
Composite (Figure 7)	✓	NO	NO	✓	NO
With-Final-States (Figure 8)	✓	NO	NO	NO	✓

Model Type – initial implementation

```
modeltype basic_fsm_type {
        basic_fsm :: FSM ,
        basic_fsm :: State ,
        basic_fsm :: Transition
}
```

Basic FSM Model Type

```
modeltype finalstates_fsm_type {
        finalstates_fsm :: FSM ,
        finalstates_fsm :: State ,
        finalstates_fsm :: Transition ,
        finalstates_fsm :: FinalState
}
```

Final States FSM Model Type

```
class Serializer<MT : basic_fsm_type> {
        operation printFSM(fsm : MT :: FSM) is do
                fsm.ownedState.each{s|
                stdio.writeln("State :" + s.name)
                s.outgoingTransition.each{t|
                                var outputText : String
                        if (t.output != void and t.output != "") then
                                outputText := t.output
                        else
                                outputText := "NC"
                        end
                        stdio.writeln("Transition :" + t.source.name + "-(" +
                        t.input + "/" + outputText + ") ->" + t.target.name)
                        }
                }
        end
}
```

A Basic FSM Operation Applied on a Final States FSM

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

15/11/2017 44

Model Type – initial implementation

- Supports:
  - the addition of new classes (FinalState) [1]
  - the tightening of multiplicity constraints (Mandatory)
  - the addition of new attributes (indirectly with Composite State Charts, via the added inheritance relationship)
  - ⇒ Match-bounded polymorphism
- Does not support: [2]
  - multiple initial states: accessing the `initialState` property in Basic state machine will return a single element typed by `State` while in Multiple state machine it will return a `Collection<State>` => *technical nightmare!*

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

15/11/2017 45

Diapositive 45

2 ne peut-il pas être détecté et générer automatiquement les adapteur ?
Benoit Combemale; 19/09/2011

1 comment inférer si l'addition n'a pas d'impact ?
Par exemple si l'ajout est obligatoire dans un objet instancié par la transformation.
==> exception !
Benoit Combemale; 21/09/2011

Model Type – enhancing matching relation

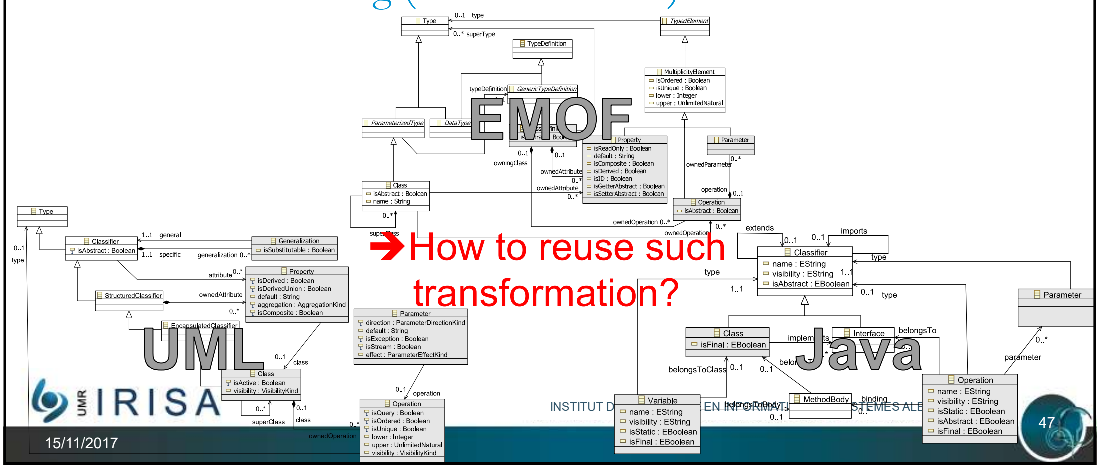
- Issues:
  - metamodel elements (e.g., classes, methods, properties) may have different names.
  - types of elements may be different.
  - additional or missing elements in a metamodel compared to another.
  - opposites may be missing in relationships.
  - the way metamodel classes are linked together may be different from one metamodel to another

Model Type – enhancing matching relation

- Motivating example: model refactoring [*MODELS'09*]

PULL UP METHOD: moving methods to the superclass when methods with identical signatures and results are located in sibling subclasses.

⇒ Model refining (with side-effect)



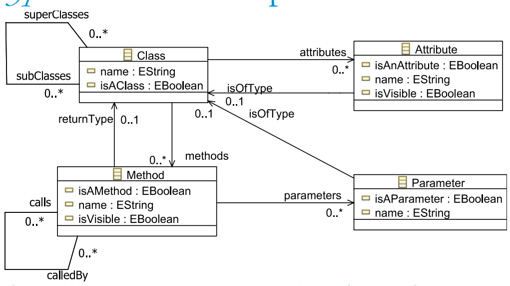
Model Type – enhancing matching relation

Model Type *M’* matches another model type *M* (denoted *M’* <# *M*) iff for each class *C* in *M*, there is one and only one corresponding class or subclass *C’* in *M’* such that every property *p* and operation *op* in *M.C* matches in *M’.C’* respectively with a property *p’* and an operation *op’* with parameters of the same type as in *M.C*.

- In practice to specify generic model refactorings:
  1. specify a lightweight metamodel (or model type) that contains the minimum required elements for refactorings.
  2. specify refactorings based on the lightweight metamodel.
  3. **adapt the target metamodels using Kermeta for weaving aspects adding derived properties and opposites that match with those of the generic metamodel.**
  4. apply the refactoring on the target metamodels

Model Type – enhancing matching relation

1 Generic *Model Type* for the Pull Up Method Refactoring



2 Kermeta Code for the Pull Up Method Refactoring

```
package refactor;

aspect class Refactor<MT : GenericMT> {

  operation pullUpMethod( source  : MT::Class ,
                          target  : MT::Class ,
                          meth    : MT::Method ) : Void

    // Preconditions
  pre sameSignatureInOtherSubclasses is do
      target.subClasses.forAll{ sub |
         sub.methods.exists{ op | haveSameSignature(meth, op) } }
  end

  // Operation body
  is do
      target.methods.add(meth)
      source.methods.remove(meth)
  end
}
```

Model Type – enhancing matching relation

3 Kermeta Code for Adapting the Java Metamodel

```
package java;

require "Java.ecore"

aspect class Classifier {
    reference inv_extends : Classifier[0..*]#extends
    reference extends : Classifier[0..1]#inv_extends
}

aspect class Class {

    property superClasses : Class[0..1]#subClasses
        getter is do
            result:=self.extends
        end

    property subClasses : Class[0..*]#superClasses
        getter is do
            result := OrderedSet<java::Class>.new
            self.inv_extends.each{ subC | result.add(subC) }
        end
}
```

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

15/11/2017

50

Model Type – enhancing matching relation

4 Kermeta Code for Applying the Pull Up Method

```
package refactor;

require "http://www.eclipse.org/uml2/2.1.2/UML"

class Main {
    operation main() : Void is do

        var rep : EMFRepository init EMFRepository.new

        var model : uml::Model
        model ?= rep.getResource("lan_application.uml").one

        var source : uml::Class init getClass("PrintServer")
        var target : uml::Class init getClass("Node")
        var meth   : uml::Operation init getOperation("bill")

        var refactor : refactor::Refactor<uml::UmlMM>
                       init refactor::Refactor<uml::UmlMM>.new

        refactor.pullUpMethod(source, target, meth)
    end
}
```

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

15/11/2017

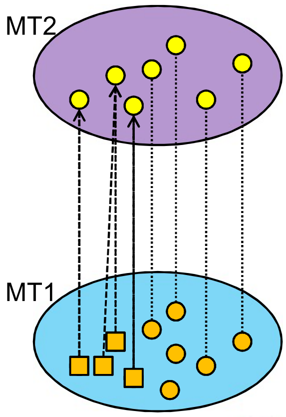
51

Bottom Line: Model Subtyping Relations

- Are models typed by MT1 substitutable to models typed by MT2?
- Two criterions to be considered
  - Structural heterogeneities between the model types
  - Context in which the subtyping relation is used

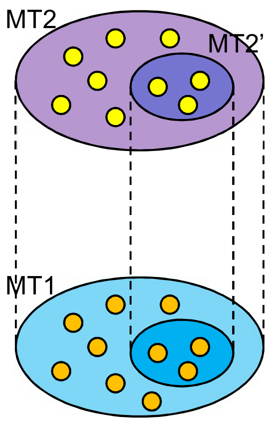
Structural heterogeneities

- Isomorphic
  - MT1 possesses the same structure as MT2
  - Comparison using class matching
- Non-isomorphic
  - Same information can be represented under different forms
  - Model adaptation from MT1 to MT2



Context of use

- Total
  - We can safely use a model typed by MT1 **everywhere** a model typed by MT2 is expected
- Partial
  - We can safely use a model typed by MT1 **in a given context where** a model typed by MT2 is expected
    - I.e., reuse of a given model manipulation *m*
  - MT1 must possess all the information needed for *m*
    - I.e., the **effective model type** of *m* from MT2



4 Model Subtyping Relations

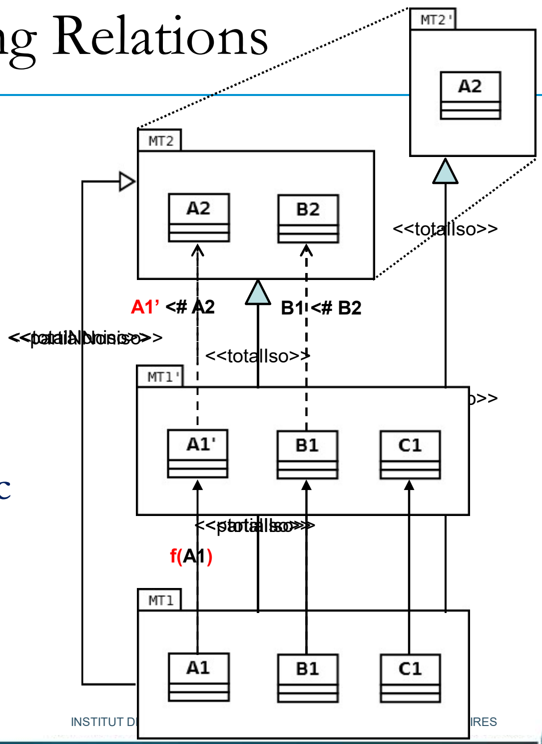
- Total isomorphic

  Matching
- Partial isomorphic

  + Pruning
- Total non-isomorphic

  + Adaptation
- Partial non-isomorphic

  + Pruning + Adaptation



Conclusion on Model Sub-Typing

- Current state in model typing
  - reuse of model transformations between isomorphic graphs
  - deal with structure deviation by weaving derived properties
  - ⇒ *Statically checked in Kermeta!!*

Model Type – *Further Needs in a Model Type System*

- Issues:
  - New DSLs are not created from scratch
    - ⇒ DSLs family (e.g., graph structure)
  - Model transformations cannot yet be specialized
    - ⇒ call to *super* and polymorphism
  - Reuse through model type matching is limited by structural conformance
    - ⇒ use of (metamodel) mapping
  - Chains of model transformations are fixed & hardcoded
    - ⇒ partial order inference of model transformations

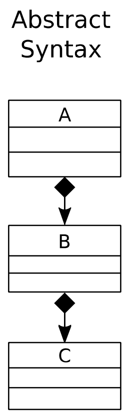
3

Diapositive 57

3 a voir pourquoi ?
Benoit Combemale; 19/09/2011

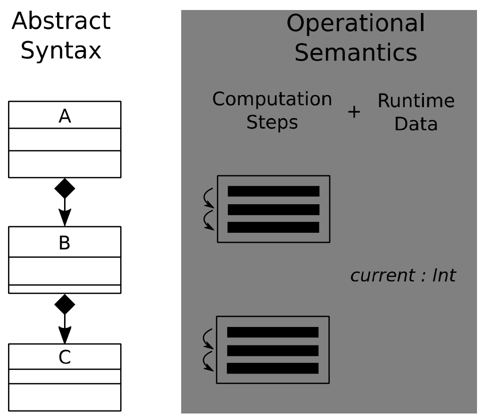
Hypothesis on LANGUAGE DEFINITION

- A metamodel specifies the *AS*



LANGUAGE DEFINITION

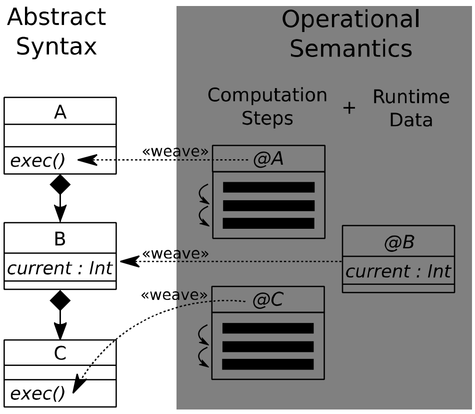
- A metamodel specifies the *AS*
- *Sem* consists of computation steps and



LANGUAGE DEFINITION

Jézéquel et al., *Mashup of metalanguages and its implementation in the kermeta language workbench*, SoSyM, 2013

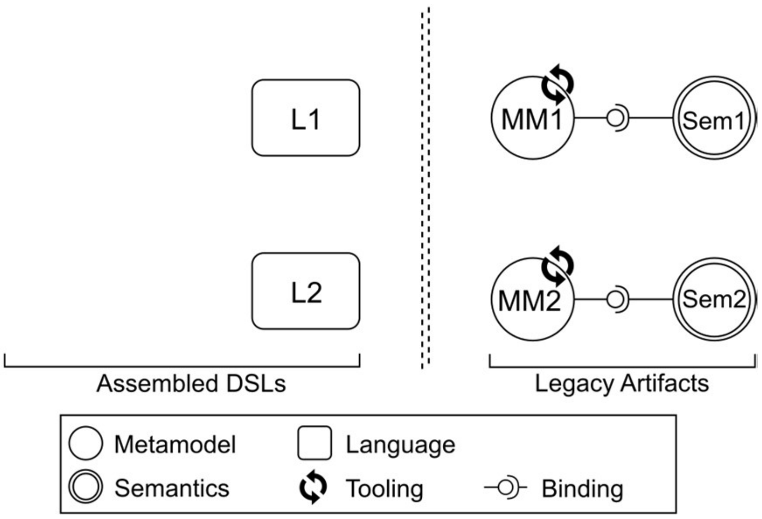
- A metamodel specifies the *AS*
- *Sem* consists of computation steps and



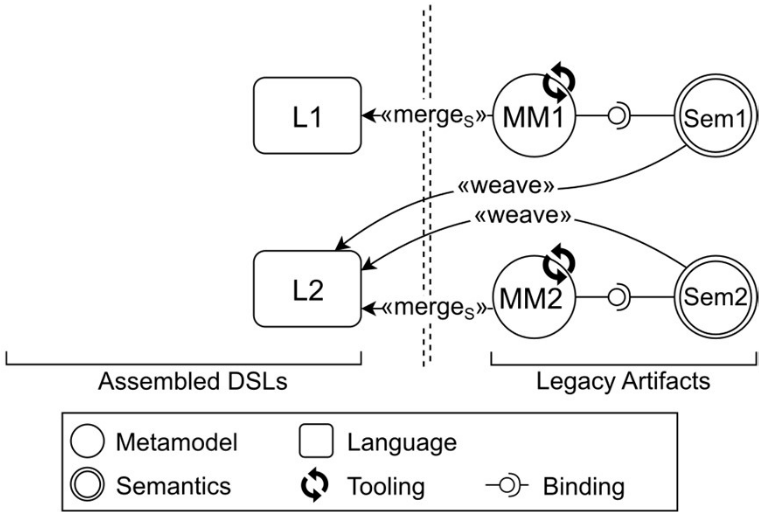
- Melange: a Meta-language for Modular and Reusable Development of DSLs

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

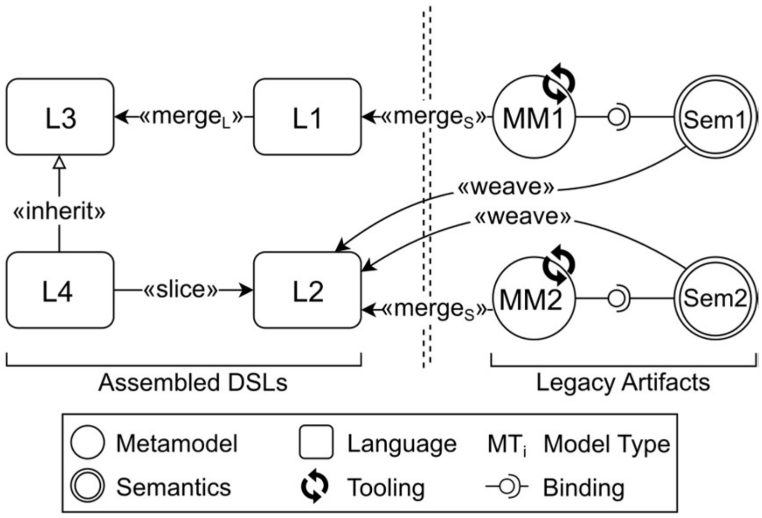
Approach Overview



Approach Overview

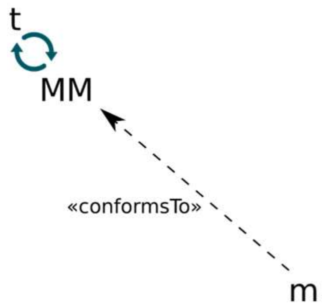


Approach Overview

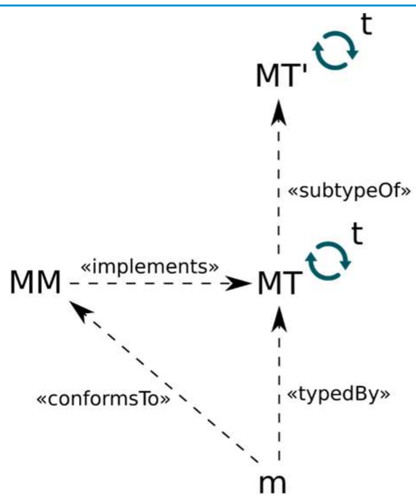


Inspired by *eg*. Erdweg *et al.*, *Language Composition Untangled*, LDTA, 2012

TOOL REUSE THROUGH MODEL TYPING



TOOL REUSE THROUGH MODEL TYPING



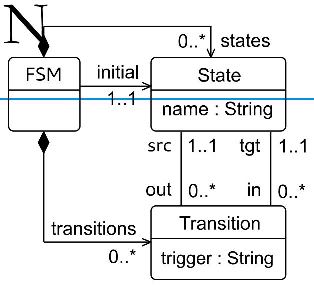
Steel *et al.*, *On Model Typing*, SoSyM, 2007
Guy *et al.*, *On Model Subtyping*, ECMFA, 2012

LANGUAGE DEFINITION

$$
\begin{aligned}
&\mathcal{L} \triangleq \langle AS, Sem, MT\rangle \\
&Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects) \text{ where} \\
&\qquad \forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t \\
&\qquad \forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j \\
&Sem \bullet Sem' \equiv Sem \frown Sem' \\
&sig(Sem) \triangleq \overset{\circ}{\bigcup_{A_i^t \in Sem}} sig(A_i^t) \\
&MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L})) \\
&\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS'\rangle \\
&\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem')\rangle \\
&\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT'\rangle \\
&\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT''\rangle \text{ where} \\
&\qquad MT'' = MT \circ MT' \text{ and} \\
&\qquad MT'' <: MT' \\
&\Lambda_-^+(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2\rangle, \text{ where:} \\
&\qquad AS_2 \triangleq \lambda_-^+(AS_1, c),\ AS_2 \subseteq AS_1, \\
&\qquad Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}, \\
&\qquad MT_1 <: MT_2,
\end{aligned}
$$

$$\mathcal{L} \triangleq \langle AS, Sem, MT\rangle$$

LANGUAGE DEFINITION



$$
\begin{aligned}
&\mathcal{L} \triangleq \langle AS, Sem, MT\rangle \\
&Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects) \text{ where} \\
&\qquad \forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t \\
&\qquad \forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j \\
&Sem \bullet Sem' \equiv Sem \frown Sem' \\
&sig(Sem) \triangleq \overset{\circ}{\bigcup_{A_i^t \in Sem}} sig(A_i^t) \\
&MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L})) \\
&\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS'\rangle \\
&\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem')\rangle \\
&\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT'\rangle \\
&\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT''\rangle \text{ where} \\
&\qquad MT'' = MT \circ MT' \text{ and} \\
&\qquad MT'' <: MT' \\
&\Lambda_-^+(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2\rangle, \text{ where:} \\
&\qquad AS_2 \triangleq \lambda_-^+(AS_1, c),\ AS_2 \subseteq AS_1, \\
&\qquad Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}, \\
&\qquad MT_1 <: MT_2,
\end{aligned}
$$

→

```
language Fsm {
syntax 'FSM.ecore'


}
```

LANGUAGE DEFINITION

$\mathcal{L} \triangleq \langle AS, Sem, MT \rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where

$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t$

$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$$sig(Sem) \triangleq \overset{\circ}{\bigcup_{A_i^t \in Sem}} sig(A_i^t)$$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS' \rangle$

$\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem') \rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT' \rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT'' \rangle$ where

$MT'' = MT \circ MT'$ and

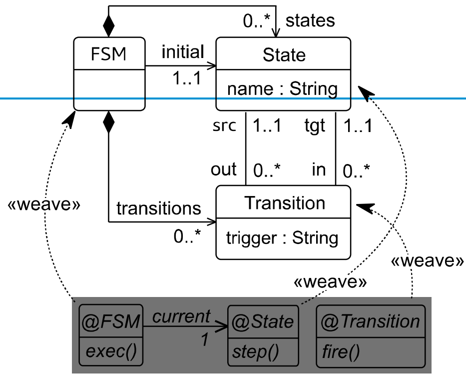
$MT'' <: MT'$

$\Lambda_-^+(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2 \rangle$, where:

$AS_2 \triangleq \lambda_-^+(AS_1, c)$, $AS_2 \subseteq AS_1$,

$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,

$MT_1 <: MT_2$,




```
language Fsm {
    syntax 'FSM.ecore'
    with ExecutableFsm
    with ExecutableState
    with ExecutableTransition

}
```


LANGUAGE DEFINITION

$\mathcal{L} \triangleq \langle AS, Sem, MT \rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where

$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t$

$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$$sig(Sem) \triangleq \overset{\circ}{\bigcup_{A_i^t \in Sem}} sig(A_i^t)$$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS' \rangle$

$\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem') \rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT' \rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT'' \rangle$ where

$MT'' = MT \circ MT'$ and

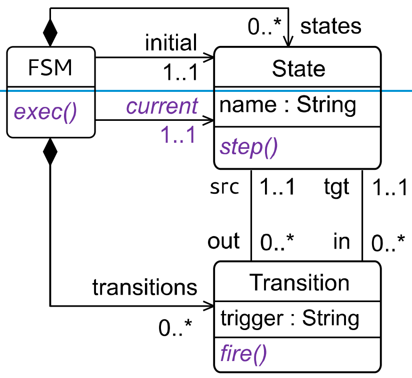
$MT'' <: MT'$

$\Lambda_-^+(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2 \rangle$, where:

$AS_2 \triangleq \lambda_-^+(AS_1, c)$, $AS_2 \subseteq AS_1$,

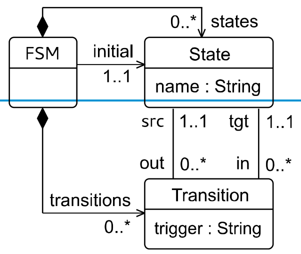
$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,

$MT_1 <: MT_2$,




```
language Fsm {
    syntax 'FSM.ecore'
    with ExecutableFsm
    with ExecutableState
    with ExecutableTransition
    exactType FsmMT
}
```

SYNTAX MERGING



$\mathcal{L} \triangleq \langle AS, Sem, MT\rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where
$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t$
$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$$sig(Sem) \triangleq \bigcup^{\circ}_{A_i^t \in Sem} sig(A_i^t)$$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS'\rangle$

$\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem')\rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT'\rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT''\rangle$ where
$MT'' = MT \circ MT'$ and
$MT'' <: MT'$

$\Lambda^+_-(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2\rangle$, where:
$AS_2 \triangleq \lambda^+_-(AS_1, c)$, $AS_2 \subseteq AS_1$,
$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,
$MT_1 <: MT_2$,

```
language GuardedFsm {
    syntax 'FSM.ecore'

    with ExecutableFsm
    with ExecutableState
    with ExecutableTransition


    exactType GuardedFsmMT
}
```

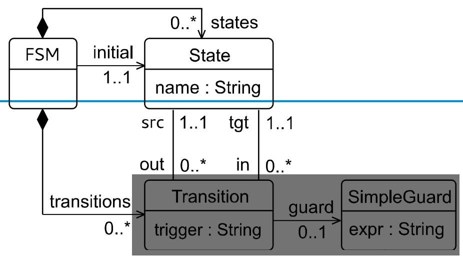
IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

15/11/2017

72

SYNTAX MERGING



$\mathcal{L} \triangleq \langle AS, Sem, MT\rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where
$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t$
$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$$sig(Sem) \triangleq \bigcup^{\circ}_{A_i^t \in Sem} sig(A_i^t)$$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS'\rangle$

$\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem')\rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT'\rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT''\rangle$ where
$MT'' = MT \circ MT'$ and
$MT'' <: MT'$

$\Lambda^+_-(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2\rangle$, where:
$AS_2 \triangleq \lambda^+_-(AS_1, c)$, $AS_2 \subseteq AS_1$,
$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,
$MT_1 <: MT_2$,

```
language GuardedFsm {
    syntax 'FSM.ecore'
    syntax 'Guard.ecore'
->  with ExecutableFsm
    with ExecutableState
    with ExecutableTransition


    exactType GuardedFsmMT
}
```

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

15/11/2017

73

SEMANTICS WEAVING

$\mathcal{L} \triangleq \langle AS, Sem, MT\rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where

$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c$ match t

$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$sig(Sem) \triangleq \bigcup^{\circ}_{A_i^t \in Sem} sig(A_i^t)$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS'\rangle$

$\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem')\rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT'\rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT''\rangle$ where

$MT'' = MT \circ MT'$ and

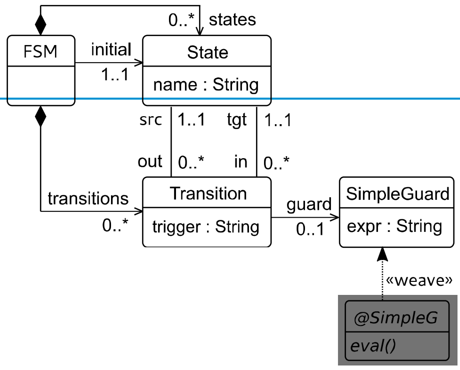
$MT'' <: MT'$

$\Lambda^{\pm}(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2\rangle$, where:

$AS_2 \triangleq \lambda^{\pm}(AS_1, c)$, $AS_2 \subseteq AS_1$,

$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,

$MT_1 <: MT_2$,




```
language GuardedFsm {
    syntax 'FSM.ecore'
    syntax 'Guard.ecore'
    with ExecutableFsm
    with ExecutableState
    with ExecutableTransition
    with EvaluateGuard

    exactType GuardedFsmMT
}
```


SEMANTICS WEAVING

$\mathcal{L} \triangleq \langle AS, Sem, MT\rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where

$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c$ match t

$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$sig(Sem) \triangleq \bigcup^{\circ}_{A_i^t \in Sem} sig(A_i^t)$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS'\rangle$

$\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem')\rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT'\rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT''\rangle$ where

$MT'' = MT \circ MT'$ and

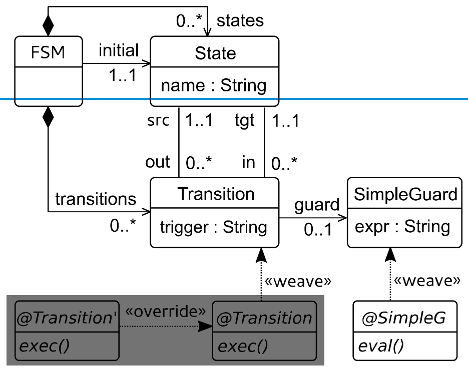
$MT'' <: MT'$

$\Lambda^{\pm}(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2\rangle$, where:

$AS_2 \triangleq \lambda^{\pm}(AS_1, c)$, $AS_2 \subseteq AS_1$,

$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,

$MT_1 <: MT_2$,




```
language GuardedFsm {
    syntax 'FSM.ecore'
    syntax 'Guard.ecore'
    with ExecutableFsm
    with ExecutableState
    with ExecutableTransition
    with EvaluateGuard
    with OverrideTransition
    exactType GuardedFsmMT
}
```

LANGUAGE MERGING

$\mathcal{L} \triangleq \langle AS, Sem, MT \rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where

$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t$

$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$$sig(Sem) \triangleq \overset{\circ}{\bigcup_{A_i^t \in Sem}} sig(A_i^t)$$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS' \rangle$

$\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem') \rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT' \rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT'' \rangle$ where

$MT'' = MT \circ MT'$ and

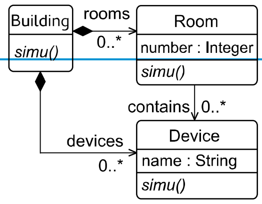
$MT'' <: MT'$

$\Lambda_-^+(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2 \rangle$, where:

$AS_2 \triangleq \lambda_-^+(AS_1, c)$, $AS_2 \subseteq AS_1$,

$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,

$MT_1 <: MT_2$,




```
language Building {
    syntax 'Building.ecore'
    with SimulatorAspect...


    exactType BuildingMT
}
```


LANGUAGE MERGING

$\mathcal{L} \triangleq \langle AS, Sem, MT \rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where

$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t$

$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$$sig(Sem) \triangleq \overset{\circ}{\bigcup_{A_i^t \in Sem}} sig(A_i^t)$$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS' \rangle$

$\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem') \rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT' \rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT'' \rangle$ where

$MT'' = MT \circ MT'$ and

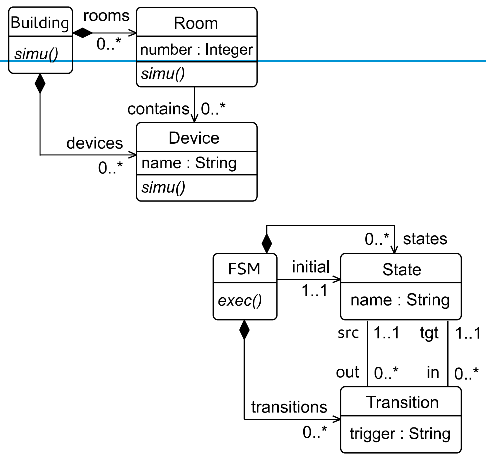
$MT'' <: MT'$

$\Lambda_-^+(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2 \rangle$, where:

$AS_2 \triangleq \lambda_-^+(AS_1, c)$, $AS_2 \subseteq AS_1$,

$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,

$MT_1 <: MT_2$,




```
language Building {
    syntax 'Building.ecore'
    with SimulatorAspect...
    merge Fsm

    exactType BuildingMT
}
```

LANGUAGE MERGING

$\mathcal{L} \triangleq \langle AS, Sem, MT\rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where

$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t$

$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$sig(Sem) \triangleq \bigcup^{\circ}_{A_i^t \in Sem} sig(A_i^t)$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \overset{m}{\leftarrow} AS' = \langle AS \circ AS', Sem, MT \circ AS'\rangle$

$\mathcal{L} \overset{w}{\leftarrow} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem')\rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT'\rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT''\rangle$ where

$MT'' = MT \circ MT'$ and

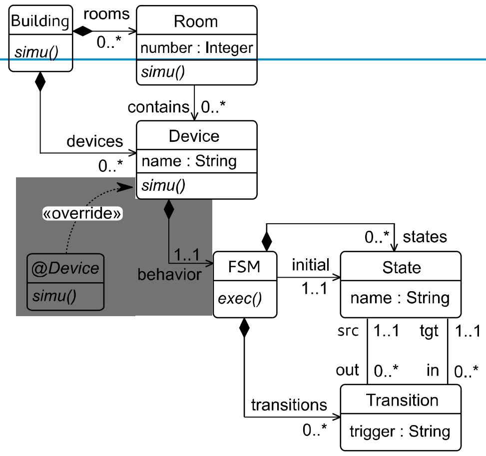
$MT'' <: MT'$

$\Lambda^{+}_{-}(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2\rangle$, where:

$AS_2 \triangleq \lambda^{+}_{-}(AS_1, c)$, $AS_2 \subseteq AS_1$,

$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,

$MT_1 <: MT_2$,




```
language Building {
    syntax 'Building.ecore'
    with SimulatorAspect...
    merge Fsm
    with GlueDeviceToFsm
    exactType BuildingMT
}
```


LANGUAGE INHERITANCE

$\mathcal{L} \triangleq \langle AS, Sem, MT\rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where

$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t$

$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$sig(Sem) \triangleq \bigcup^{\circ}_{A_i^t \in Sem} sig(A_i^t)$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \overset{m}{\leftarrow} AS' = \langle AS \circ AS', Sem, MT \circ AS'\rangle$

$\mathcal{L} \overset{w}{\leftarrow} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem')\rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT'\rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT''\rangle$ where

$MT'' = MT \circ MT'$ and

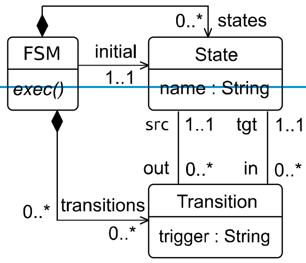
$MT'' <: MT'$

$\Lambda^{+}_{-}(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2\rangle$, where:

$AS_2 \triangleq \lambda^{+}_{-}(AS_1, c)$, $AS_2 \subseteq AS_1$,

$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,

$MT_1 <: MT_2$,




```
language TimedFsm inherits Fsm {


    exactType TimedFsmMT
}
```

LANGUAGE INHERITANCE

$\mathcal{L} \triangleq \langle AS, Sem, MT \rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where

$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t$

$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$sig(Sem) \triangleq \overset{\circ}{\bigcup_{A_i^t \in Sem}} sig(A_i^t)$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS' \rangle$

$\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem') \rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT' \rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT'' \rangle$ where

$MT'' = MT \circ MT'$ and

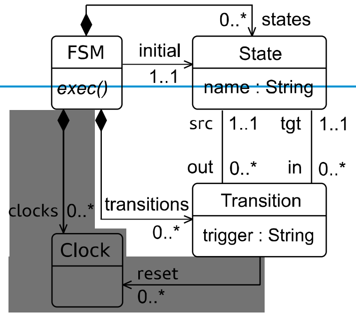
$MT'' <: MT'$

$\Lambda^{+}_{-}(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2 \rangle$, where:

$AS_2 \triangleq \lambda^{+}_{-}(AS_1, c)$, $AS_2 \subseteq AS_1$,

$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,

$MT_1 <: MT_2$,




```
language TimedFsm inherits Fsm {
        syntax 'Clocks.ecore'


        exactType TimedFsmMT
}
```


LANGUAGE INHERITANCE

$\mathcal{L} \triangleq \langle AS, Sem, MT \rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where

$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t$

$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$sig(Sem) \triangleq \overset{\circ}{\bigcup_{A_i^t \in Sem}} sig(A_i^t)$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS' \rangle$

$\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem') \rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT' \rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT'' \rangle$ where

$MT'' = MT \circ MT'$ and

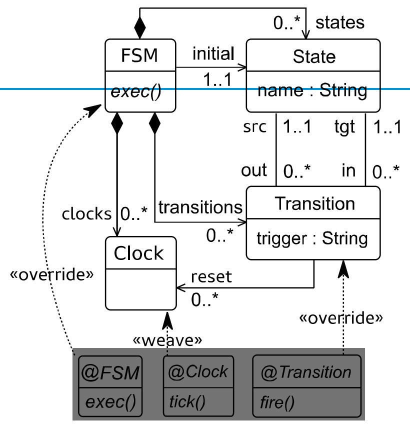
$MT'' <: MT'$

$\Lambda^{+}_{-}(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2 \rangle$, where:

$AS_2 \triangleq \lambda^{+}_{-}(AS_1, c)$, $AS_2 \subseteq AS_1$,

$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,

$MT_1 <: MT_2$,




```
language TimedFsm inherits Fsm {
        syntax 'Clocks.ecore'
           with ClockTick
          with OverrideFsm
        with OverrideTransition
        exactType TimedFsmMT
}
```

LANGUAGE SLICING

$\mathcal{L} \triangleq \langle AS, Sem, MT\rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where

$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c$ match t

$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$$sig(Sem) \triangleq \bigcup^{\circ}_{A_i^t \in Sem} sig(A_i^t)$$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS'\rangle$

$\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem')\rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT'\rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT''\rangle$ where

$MT'' = MT \circ MT'$ and

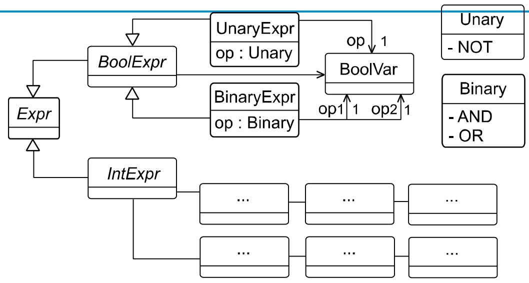
$MT'' <: MT'$

$\Lambda^{+}_{-}(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2\rangle$, where:

$AS_2 \triangleq \lambda^{+}_{-}(AS_1, c)$, $AS_2 \subseteq AS_1$,

$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,

$MT_1 <: MT_2$,




```
language Expressions {
syntax 'Expressions.ecore'
   with EvaluateBoolean
   with EvaluateInteger
  exactType ExpressionsMT
           }
```


LANGUAGE SLICING

$\mathcal{L} \triangleq \langle AS, Sem, MT\rangle$

$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects)$ where

$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c$ match t

$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$

$Sem \bullet Sem' \equiv Sem \frown Sem'$

$$sig(Sem) \triangleq \bigcup^{\circ}_{A_i^t \in Sem} sig(A_i^t)$$

$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$

$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS'\rangle$

$\mathcal{L} \xleftarrow{w} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem')\rangle$

$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT'\rangle$

$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT''\rangle$ where

$MT'' = MT \circ MT'$ and

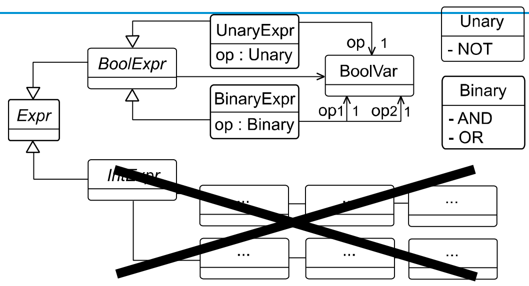
$MT'' <: MT'$

$\Lambda^{+}_{-}(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2\rangle$, where:

$AS_2 \triangleq \lambda^{+}_{-}(AS_1, c)$, $AS_2 \subseteq AS_1$,

$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\}$,

$MT_1 <: MT_2$,



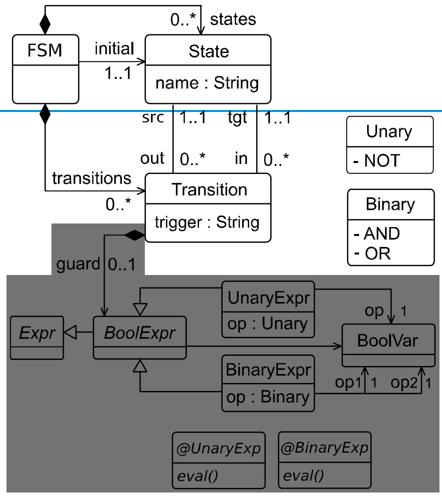

```
language Expressions {
syntax 'Expressions.ecore'
   with EvaluateBoolean
   with EvaluateInteger
  exactType ExpressionsMT
           }
```

→

```
language BooleanExpressions {
     slice Expressions on
       ['BoolExpr']
  exactType BooleanExpressionsMT
           }
```

LANGUAGE SLICING

$$\mathcal{L} \triangleq \langle AS, Sem, MT \rangle$$
$$Sem(\mathcal{L}) \triangleq (A_i^t \in Aspects) \text{ where}$$
$$\forall A_i^t \in Sem(\mathcal{L}), \exists c \in AS(\mathcal{L}) : c \text{ match } t$$
$$\forall A_i^t, A_j^t \in Sem(\mathcal{L}) : A_i^t \lhd A_j^t \implies i > j$$
$$Sem \bullet Sem' \equiv Sem \frown Sem'$$
$$sig(Sem) \triangleq \bigcup_{A_i^t \in Sem}^{\circ} sig(A_i^t)$$
$$MT(\mathcal{L}) \triangleq AS(\mathcal{L}) \circ sig(Sem(\mathcal{L}))$$
$$\mathcal{L} \overset{m}{\leftarrow} AS' = \langle AS \circ AS', Sem, MT \circ AS' \rangle$$
$$\mathcal{L} \overset{w}{\leftarrow} Sem' = \langle AS, Sem \bullet Sem', MT \circ sig(Sem') \rangle$$
$$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT' \rangle$$
$$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT'' \rangle \text{ where}$$
$$MT'' = MT \circ MT' \text{ and}$$
$$MT'' <: MT'$$
$$\Lambda_{-}^{+}(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2 \rangle, \text{ where:}$$
$$AS_2 \triangleq \lambda_{-}^{+}(AS_1, c),\ AS_2 \subseteq AS_1,$$
$$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\},$$
$$MT_1 <: MT_2,$$



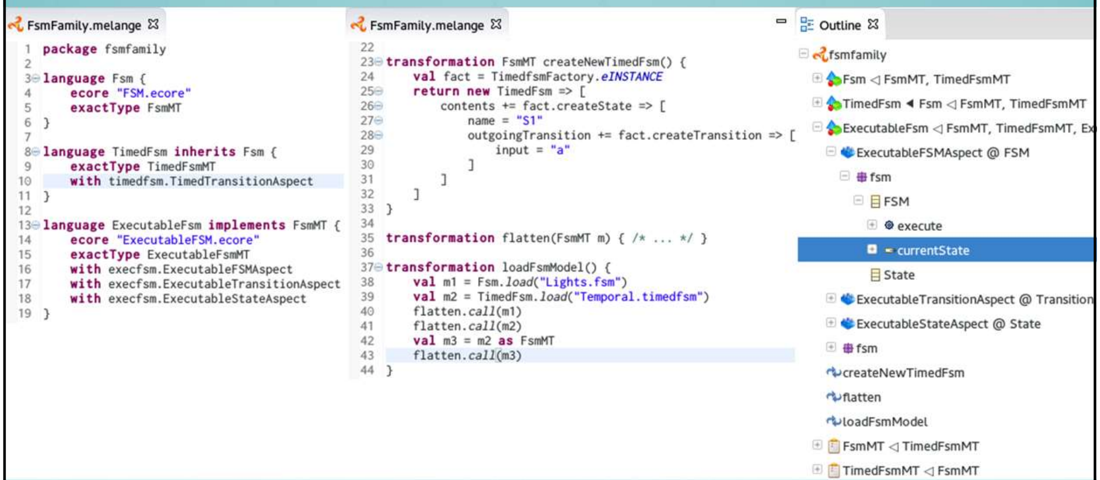

```
language GuardedFsm inherits Fsm {
    with ...
    merge BooleanExpressions
    with AttachGuardToTransition
    exactType GuardedFsmMT
}
```

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALEATOIRES

15/11/2017

84



Melange

A Language Workbench

MELANGE

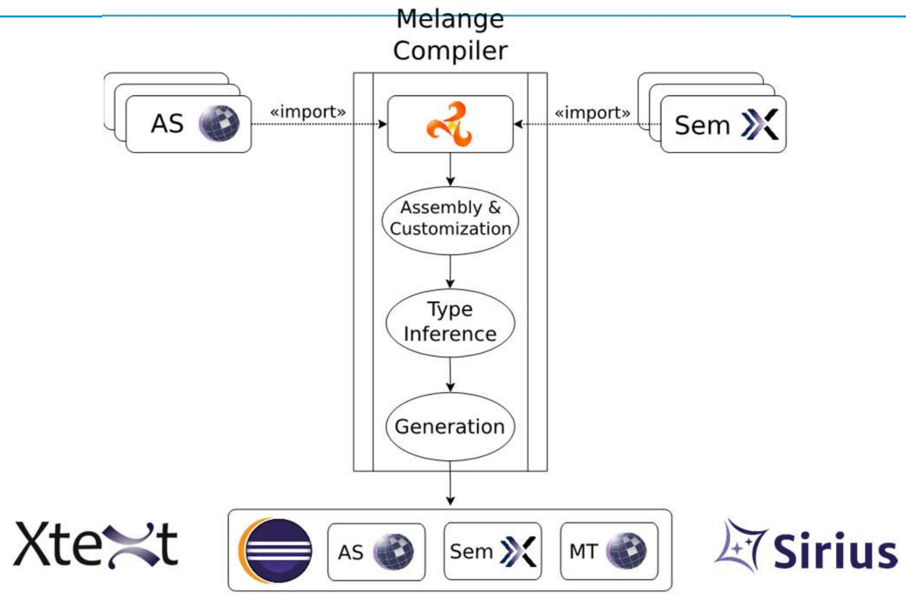
- An open-source (EPL) language workbench
- or… a language-based, model-oriented language for DSL engineering
- An implementation of the algebra
- Supported by a model-oriented type sytem
- Based on Xtext
- Seamlessly integrated with the EMF ecosystem
- Bundled as a set of Eclipse plug-ins

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



Wrap-up: Challenges

- Reuse
  - language constructs, grammars, editors or tool chains (model transformations, compilers…)
- Substitutability
  - replacement of one software artifact (e.g. code, object, module) with another one under certain conditions
- Extension
  - introduction of new constructs, abstractions, or tools

Challenges for DSL Modularity

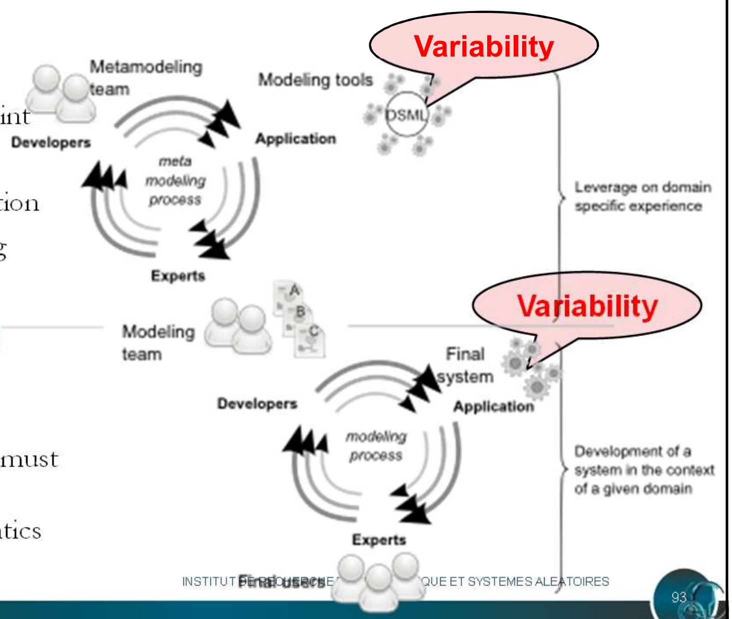
- Modularity and composability
  - structure software applications as sets of interconnected building blocks
- How to breakdown a language?
  - how the language units should be defined so they can be reused in other contexts
    - What is the correct level of granularity?
    - What are the *services* a language unit should offer to be reusable?
    - What is the meaning of a *service* in the context of software languages?
    - What is the meaning of a *services composition* in the context of software languages?

Challenges for DSL Modularity

- How can language units be specified?
  - not only about implementing a subset of the language
  - but also about specifying its boundary
    - the set of services it offers to other language units and the set of services it requires from other language units.
  - classical idea of required and provided interfaces
    - introduced by components-based software engineering approaches.
    - But... What is the meaning of “provided and required services" in the context of software languages?
  - composability & substitutability
    - Extends vs. uses

Big Picture: Variability Everywhere

- Variability in Metamodeling:
  - Semantic variation point
  - DSML Families
  - Knowledge capitalization
  - Language Engineering
- Variability in Modeling:
  - Support positive and negative variability
  - Derivation semantics must take into account the assets language semantics



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