Composition and Interoperability for External Domain-Specific Language Engineering

Thomas Degueule

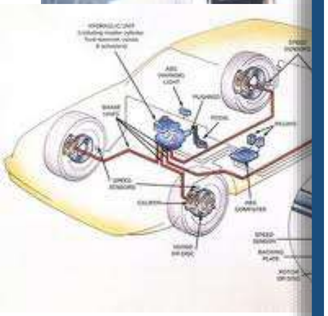
PhD Defense

December 12, 2016

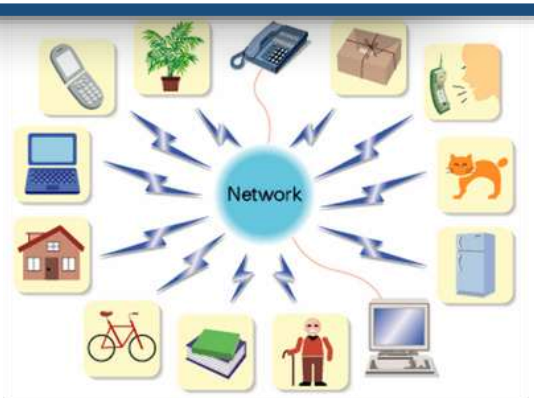
Pr. Mark van den Brand, Eindhoven University of Technology	*Reviewer*
Pr. Richard Paige, University of York	*Reviewer*
Pr. Sandrine Blazy, University of Rennes	*Examiner*
Pr. Ralf Lämmel, University of Koblenz-Landau	*Examiner*
Pr. Bernhard Rumpe, RWTH Aachen University	*Examiner*
Pr. Olivier Barais, University of Rennes	*Advisor*
Dr. Arnaud Blouin, INSA Rennes	*Advisor*
Dr. Benoit Combemale, University of Rennes	*Advisor*

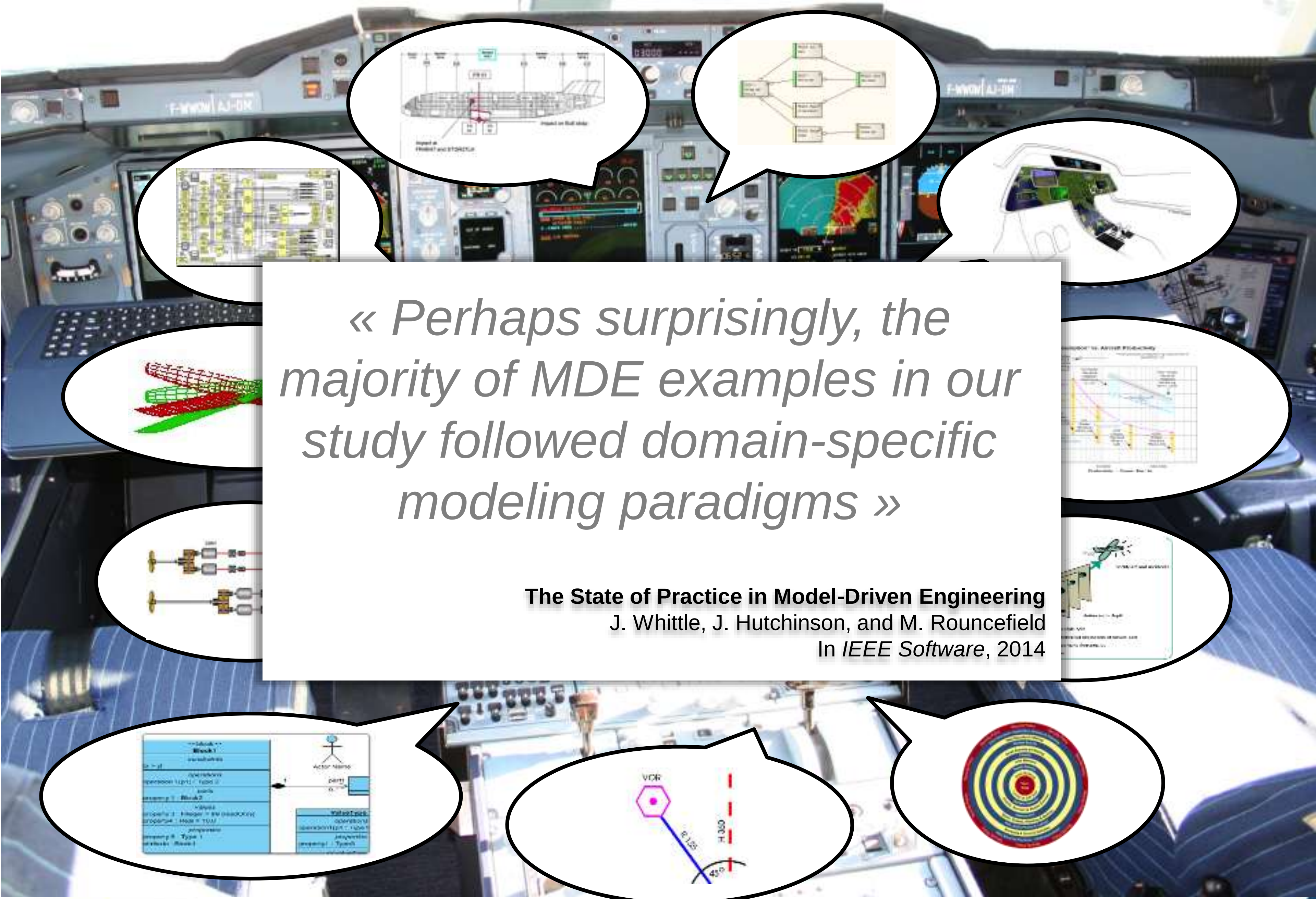
DiverSE · Inria · UMR IRISA · Université de Rennes 1 · MERGE Safety & Security

Complex Software-Intensive Systems



- Multiple concerns & stakeholders
- Multi-engineering approach
- Software as an integration layer





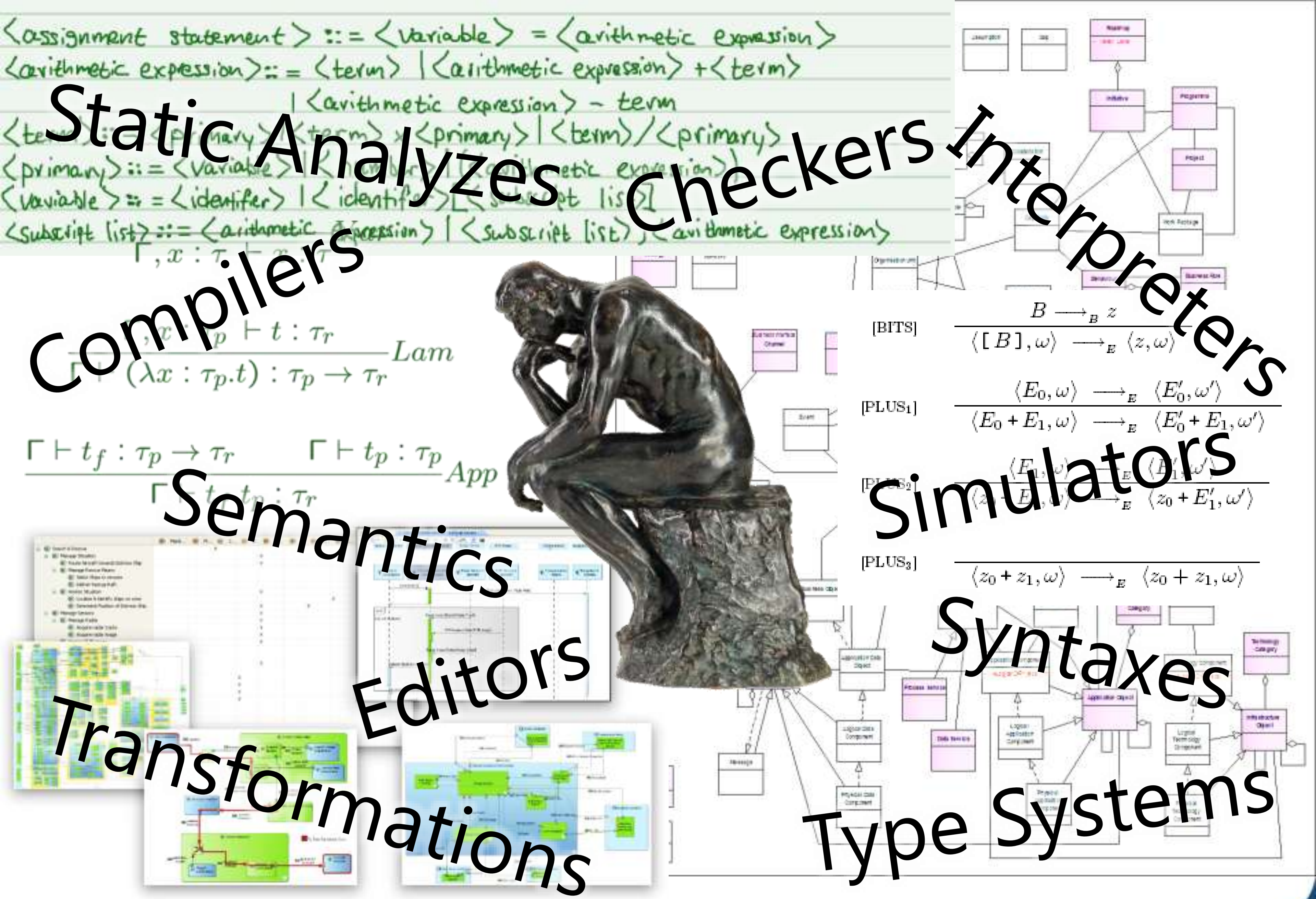
« *Perhaps surprisingly, the majority of MDE examples in our study followed domain-specific modeling paradigms* »

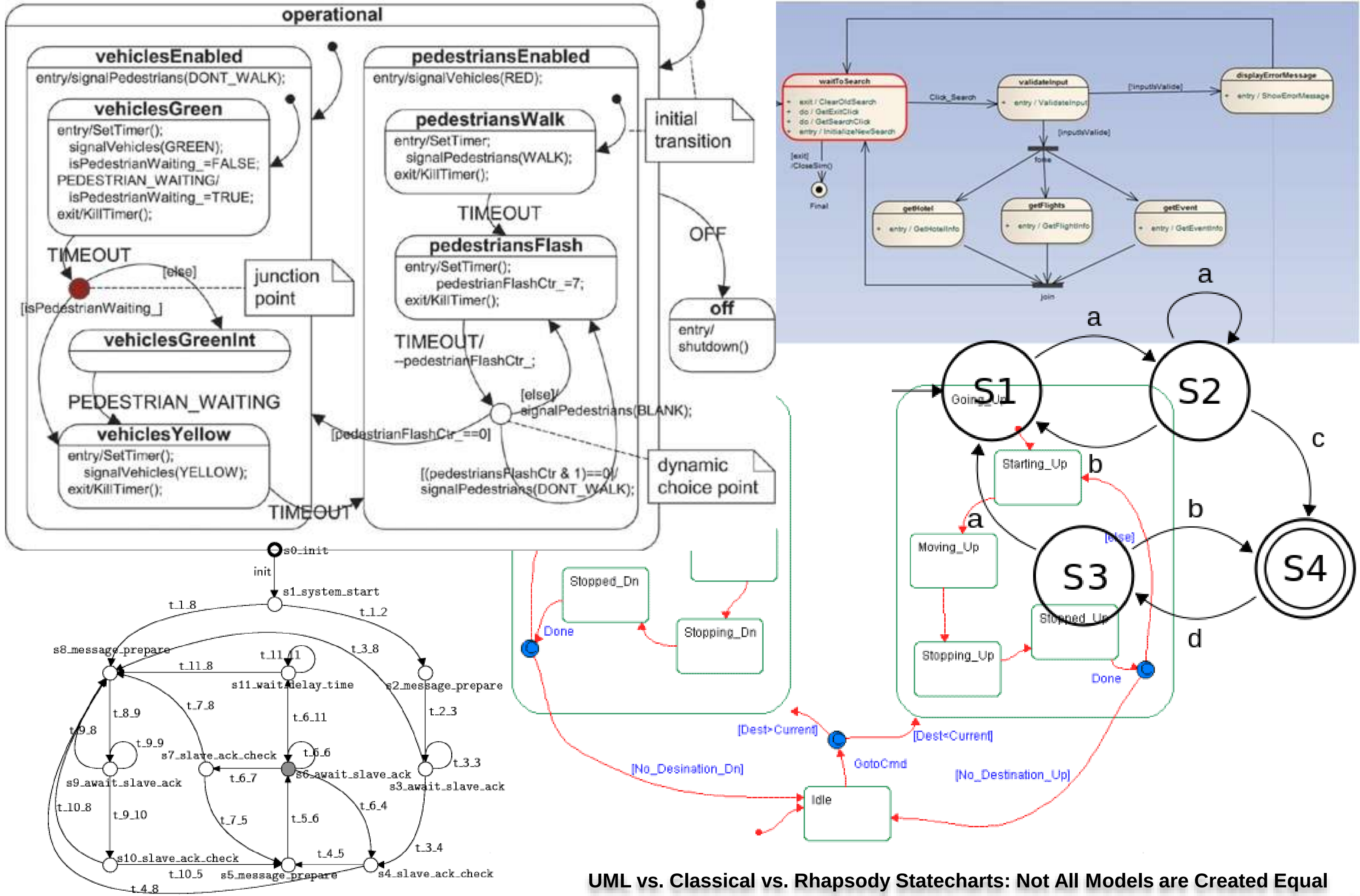
The State of Practice in Model-Driven Engineering
J. Whittle, J. Hutchinson, and M. Rouncefield
In *IEEE Software*, 2014

Domain-Specific Languages

VS

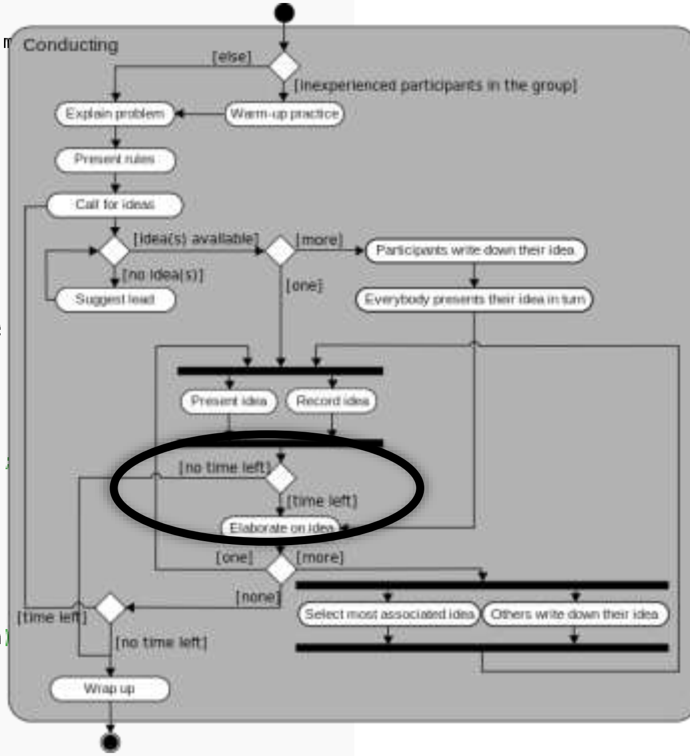
- Abstractions, notations, and tools specifically tailored to the domain
- Easier to understand, reason about, and maintain
- *External DSLs*
  - Carry their own syntax, representation, semantics, environment

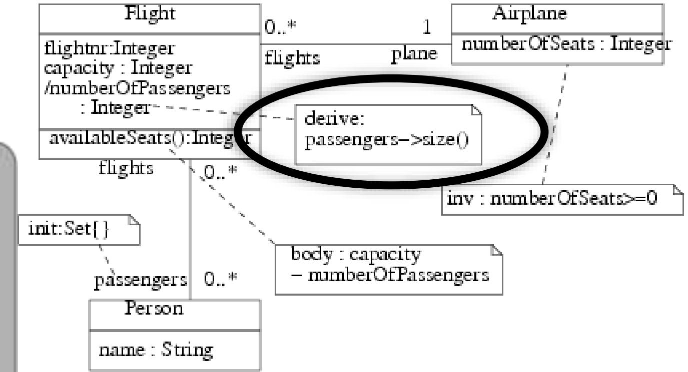

Static Analyzes
Checkers
Interpreters
Compilers
Simulators
Semantics
Syntaxes
Editors
Transformations
Type Systems



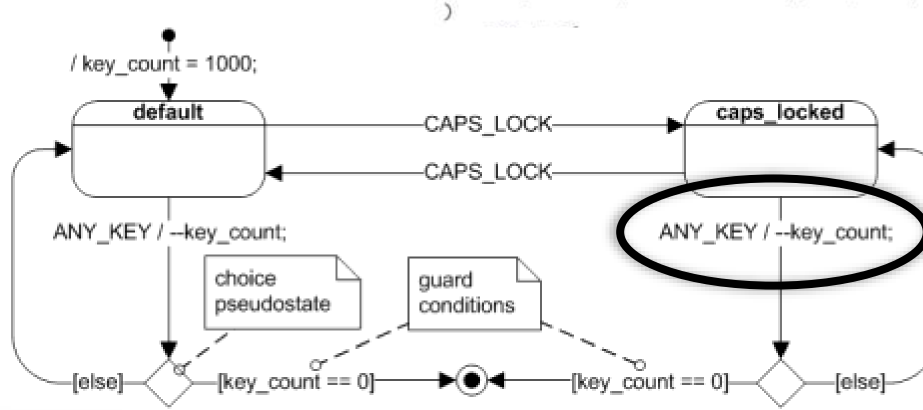
UML vs. Classical vs. Rhapsody Statecharts: Not All Models are Created Equal
Michelle L. Crane, Jürgen Dingel
In *Software & Systems Modeling* (SoSyM), 2007

```
private Table interpStm(Stm s, Table table) {
  %match(s) {
    AssignStm(name,e1) -> {
      Pair p = interpExp(`e1,table);
      Table newTable = `Table(name,p.getValue(),p.getTable());
      return newTable;
    }

    SeqStm(s1,s2*) -> {
      Table newTable = `interpStm(s2,interpStm
      return newTable;
    }

    PrintStm(l) -> {
      Table t = interpPrint(`l,table);
      System.out.println();
      return t;
    }
  }
  return `EmptyTable();
}

private Table interpPrint(ExpList list, Table
  %match(list) {
    ExpList(e1,tail*) -> {
      Pair p = `interpExp(e1,table);
      System.out.print(p.getValue());
      System.out.print(" ");
      return interpPrint(`tail*,p.getTable());
    }
  }
  return table;
}

private Pair interpExp(Exp e, Table table) {
  %match(e) {
    IdExp(n) -> { return `Pair(lookup(table,n)
    NumExp(v) -> { return `Pair(v,table); }
    OpExp(e1,op,e2) -> {
      Pair p1 = `interpExp(e1,table);
      Pair p2 = `interpExp(e2,p1.getTable());
      %match(p1,op,p2) {
        Pair(i1,t1), Plus(), Pair(i2,t2) -> { return `Pair(i1 + i2,t2); }
        Pair(i1,t1), Minus(), Pair(i2,t2) -> { return `Pair(i1 - i2,t2); }
        Pair(i1,t1), Times(), Pair(i2,t2) -> { return `Pair(i1 * i2,t2); }
        Pair(i1,t1), Div(), Pair(i2,t2) -> { return `Pair(i1 / i2,t2); }
      }
    }

    SeqExp(s1,e1) -> {
      Table t = `interpStm(s1,table);
      Pair p = `interpExp(e1,t);
      return p;
    }

  }
  System.out.println("should not be there: " + e);
  return null;
}
```






```
def List<Aspect> allSemantics(Language l) {
    val tmp = newArrayList

    tmp += l.superLanguages.map[allSemantics].flatten
    tmp +=
        l.operators.map[op |
            if (op instanceof Slice)
                op.targetLanguage.allSemantics
            else if (op instanceof Merge)
                op.targetLanguage.allSemantics
            else
                newArrayList
        ].flatten
    tmp += l.semantics

    val res = newArrayList
    tmp.forEach[a1 |
        if (!res.exists[Aspect a2 | a2.aspectTypeRef.i
            && (!a1.hasAspectAnnotation || l.syntax.pkg
        )
```



SLE Challenges

DSL & Tools Designer

- Reduce development costs
- Avoid engineering DSLs from scratch
- Reuse & customize existing DSLs

Language Composition

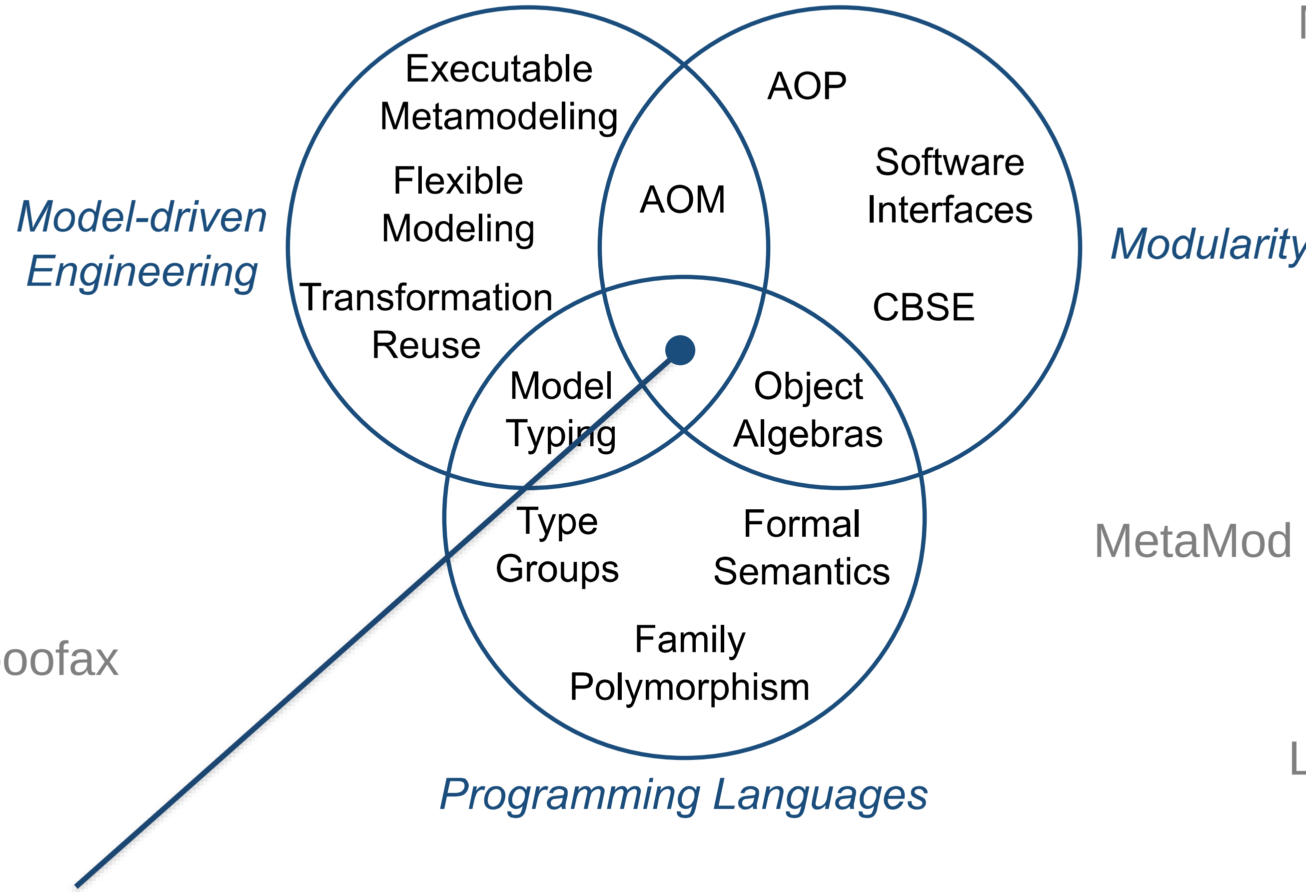
DSL User

- Foster model sharing and collaboration
- Manipulate models in different environments
- Reuse tools and services

Language Interoperability

State of the Art

MontiCore



DeLara's concepts

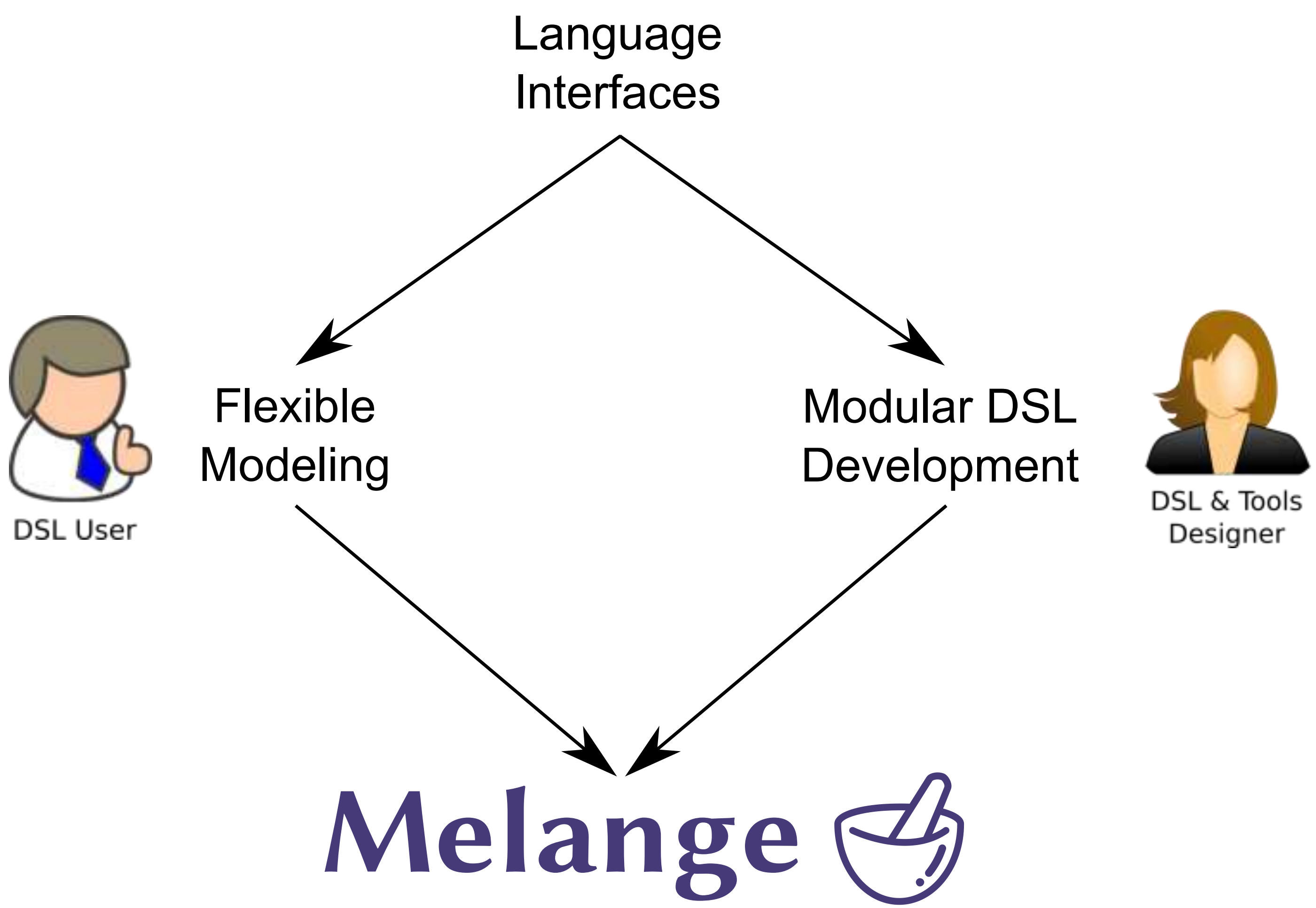
Spoofax

MetaMod

LISA

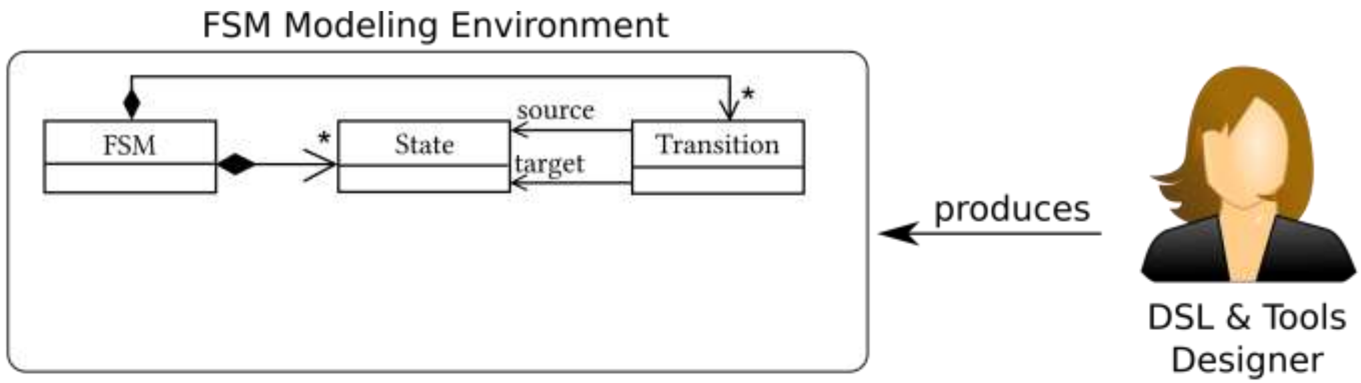
a non-intrusive and tool-supported approach to composition and interoperability for SLE applicable to legacy DSLs

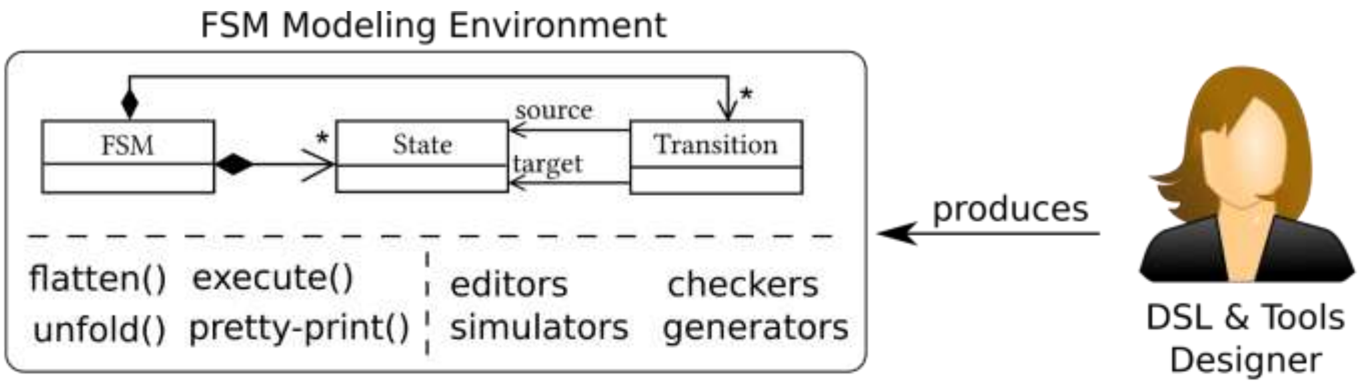
Outline of the Contributions

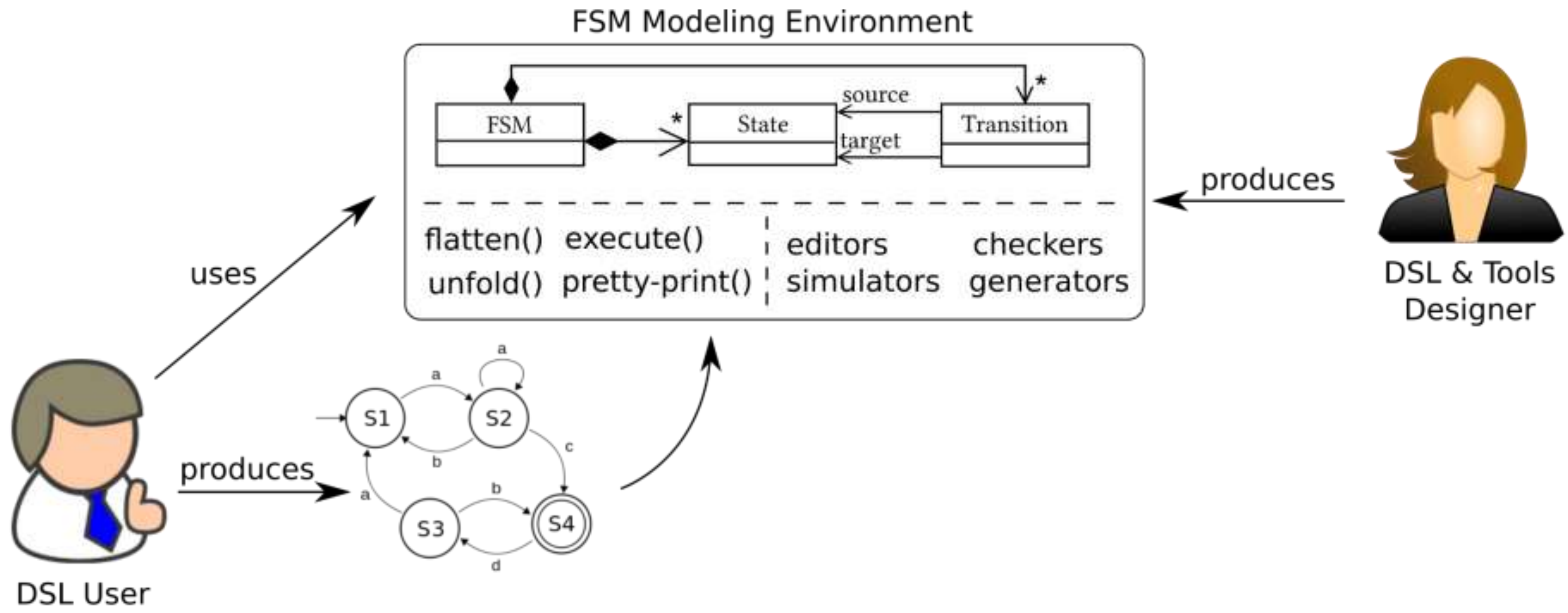


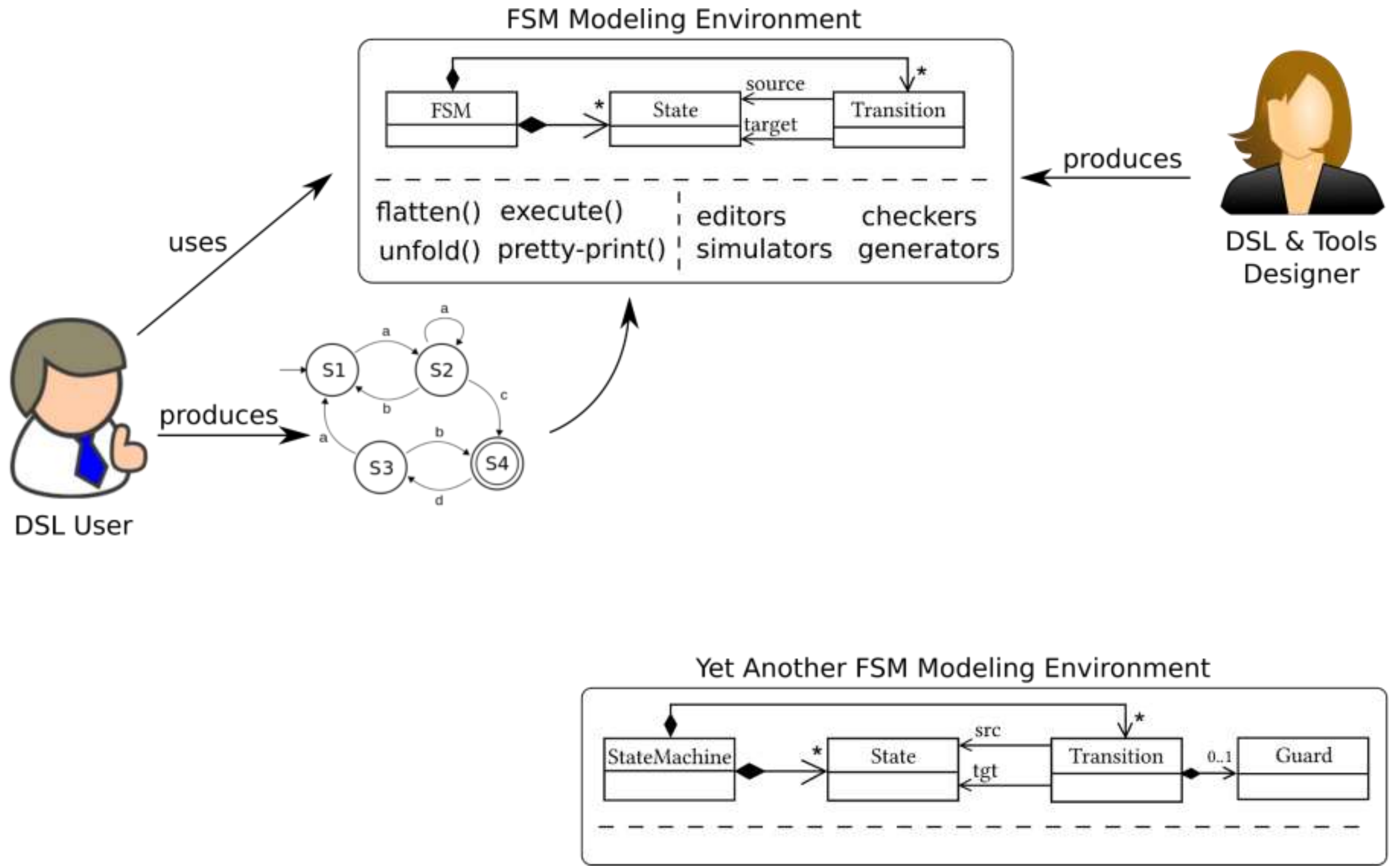
On Language Interfaces

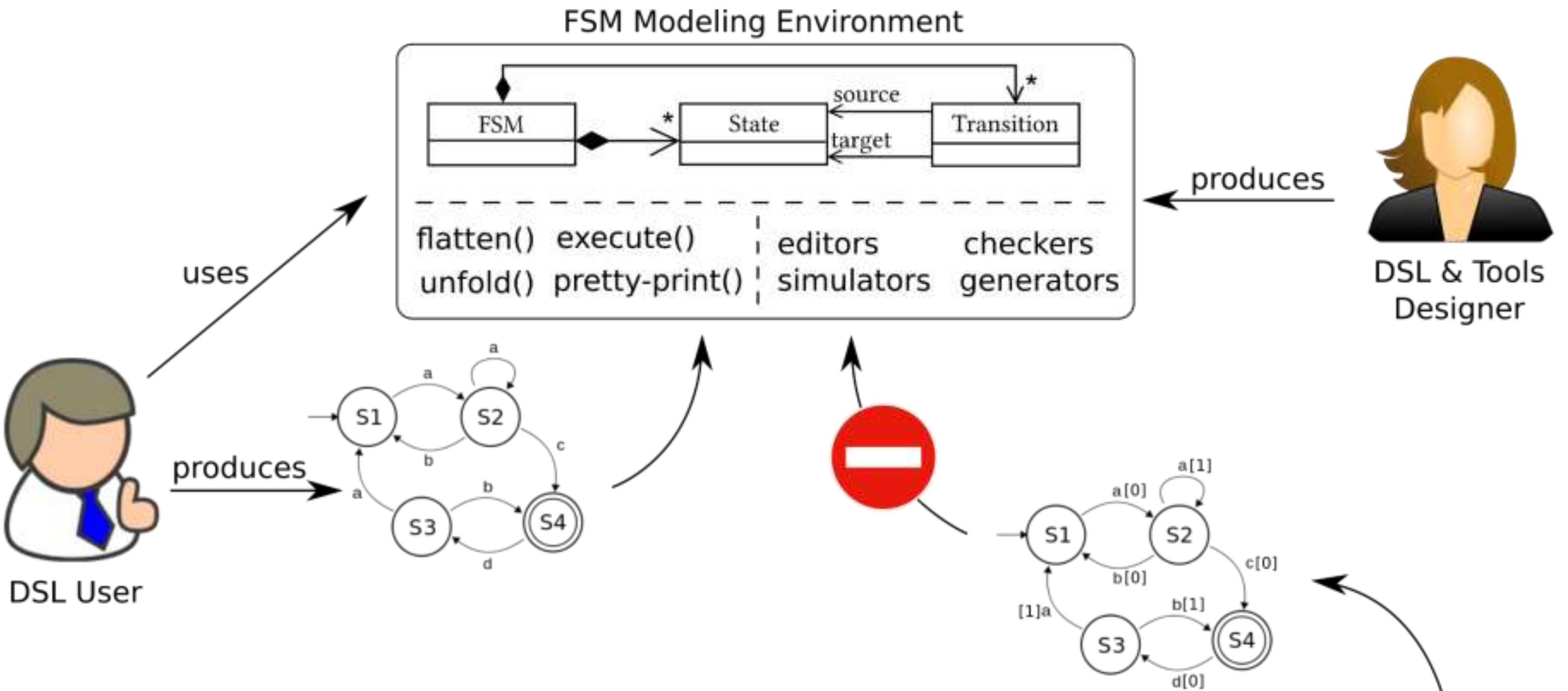
On Language Interfaces
Thomas Degueule, Benoit Combemale and Jean-Marc Jézéquel
In *PAUSE: Present And Ulterior Software Engineering, 2017*
Ed. Bertrand Meyer and Manuel Mazzara


FSM Modeling Environment
FSM
State
Transition
source
target
*
*
produces
DSL & Tools
Designer

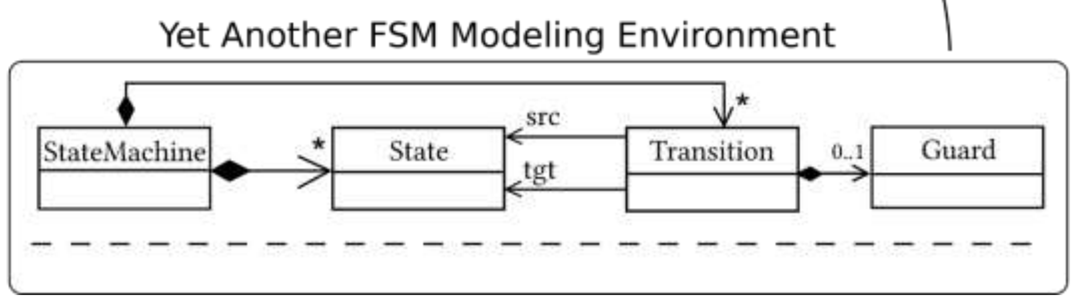

FSM Modeling Environment
FSM
State
Transition
source
target
*
*
flatten()
execute()
unfold()
pretty-print()
editors
checkers
simulators
generators
produces
DSL & Tools Designer

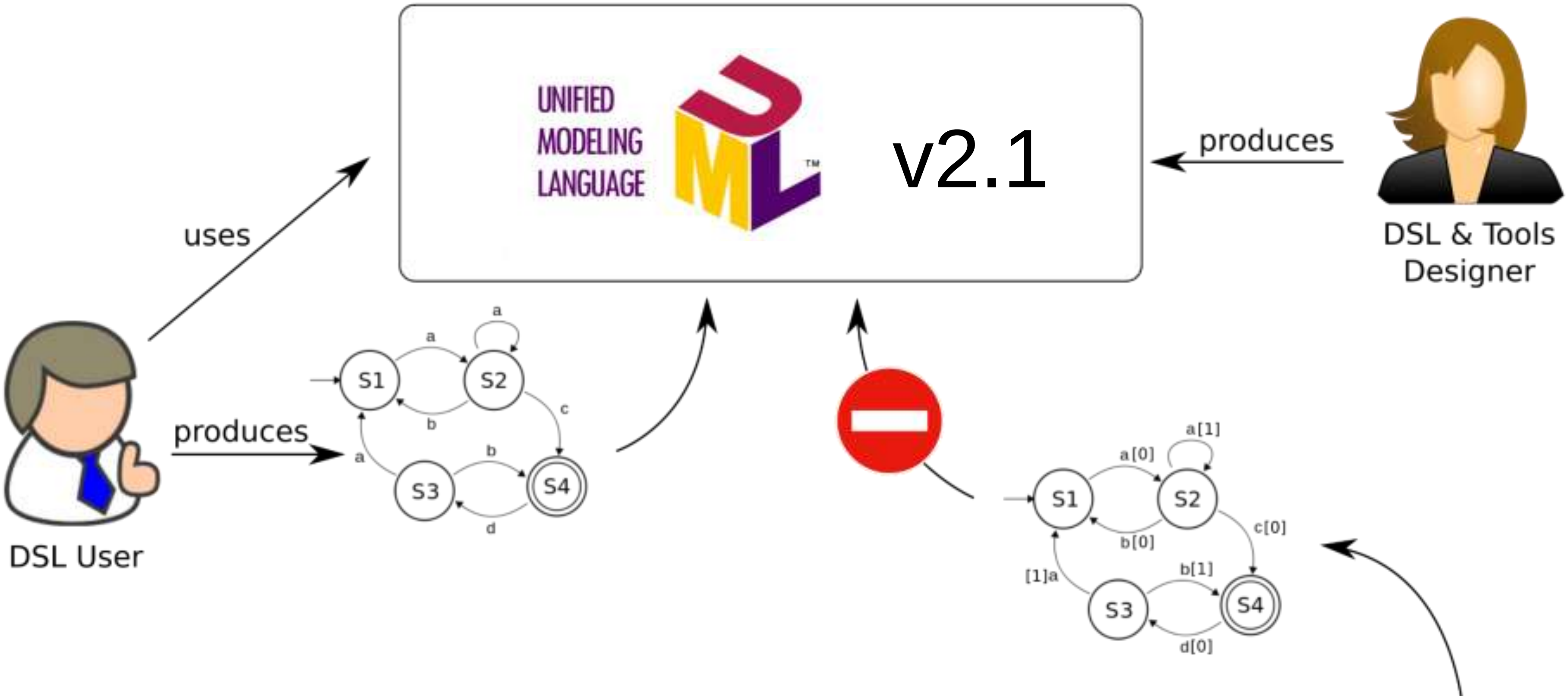

FSM Modeling Environment
FSM
State
Transition
source
target
*
*
flatten() execute()
unfold() pretty-print()
editors checkers
simulators generators
produces
DSL & Tools Designer
uses
produces
DSL User
S1
S2
S3
S4
a
b
c
d


FSM Modeling Environment
FSM
State
Transition
source
target
*
*
flatten() execute()
unfold() pretty-print()
editors checkers
simulators generators
produces
DSL & Tools Designer
uses
produces
DSL User
S1
S2
S3
S4
a
b
c
d
Yet Another FSM Modeling Environment
StateMachine
State
Transition
Guard
src
tgt
0..1
*
*

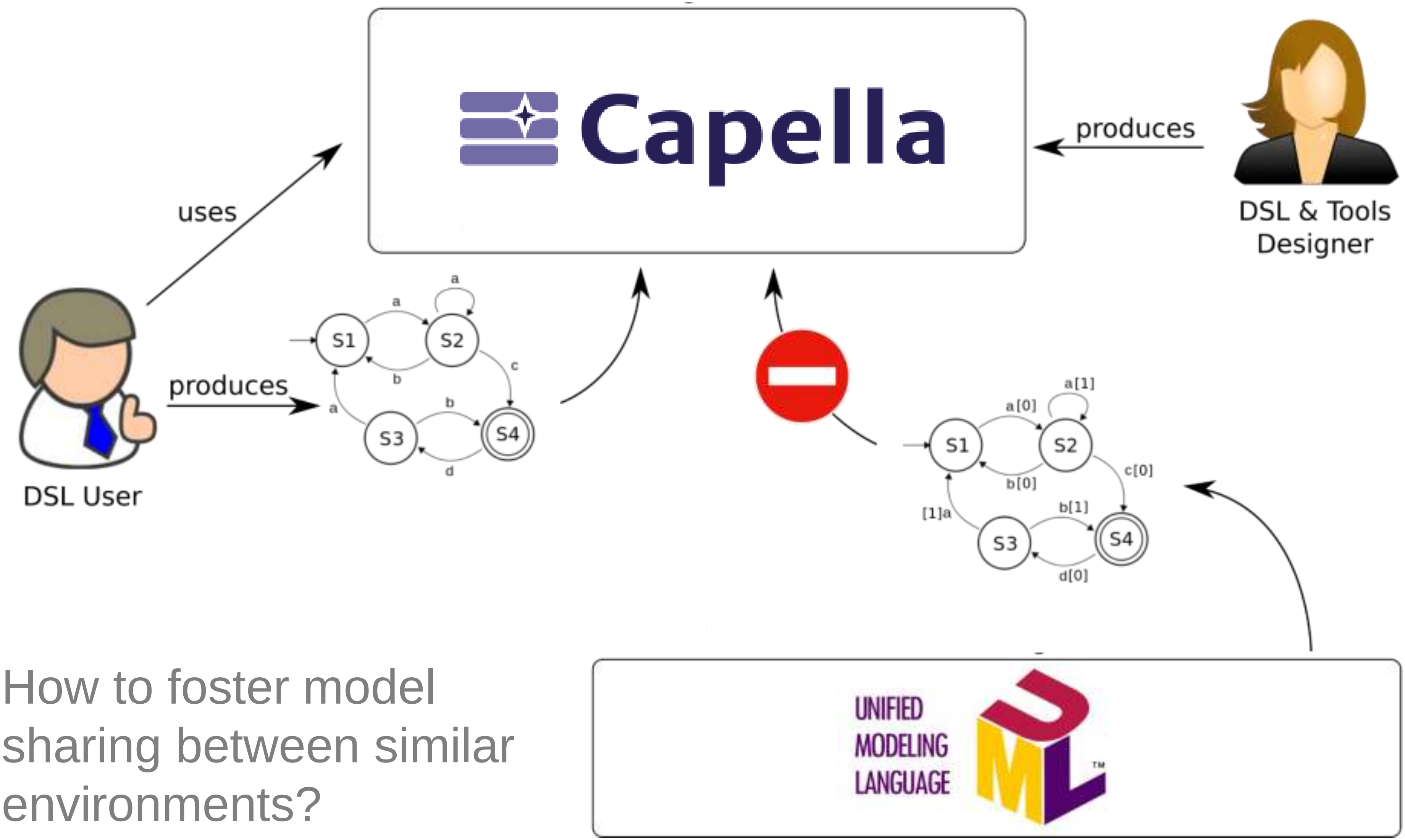


Variants or subsequent versions cannot leverage previous engineering efforts





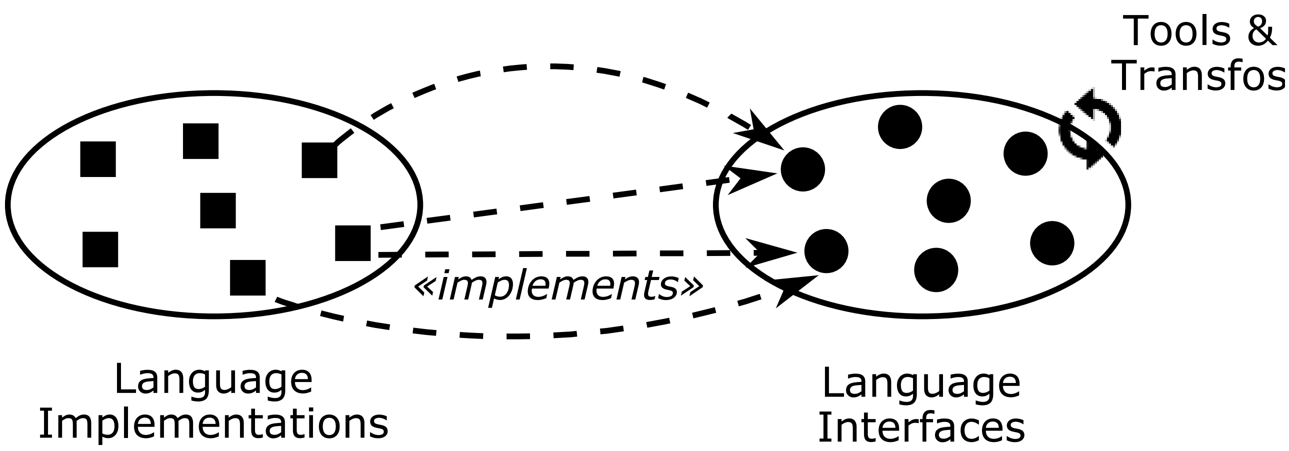
How to ensure interoperability between subsequent versions?



How to foster model sharing between similar environments?

Software Language Interfaces

- Abstract over the intrinsic complexity of language implementations
- Expose meaningful information
  - Concerning an aspect of a language (e.g. abstract syntax)
  - For a given purpose (e.g. composition, coordination, analysis)
  - In an appropriate formalism (e.g. a metamodel, a control-flow graph)
- Provide a reasoning layer atop language implementations

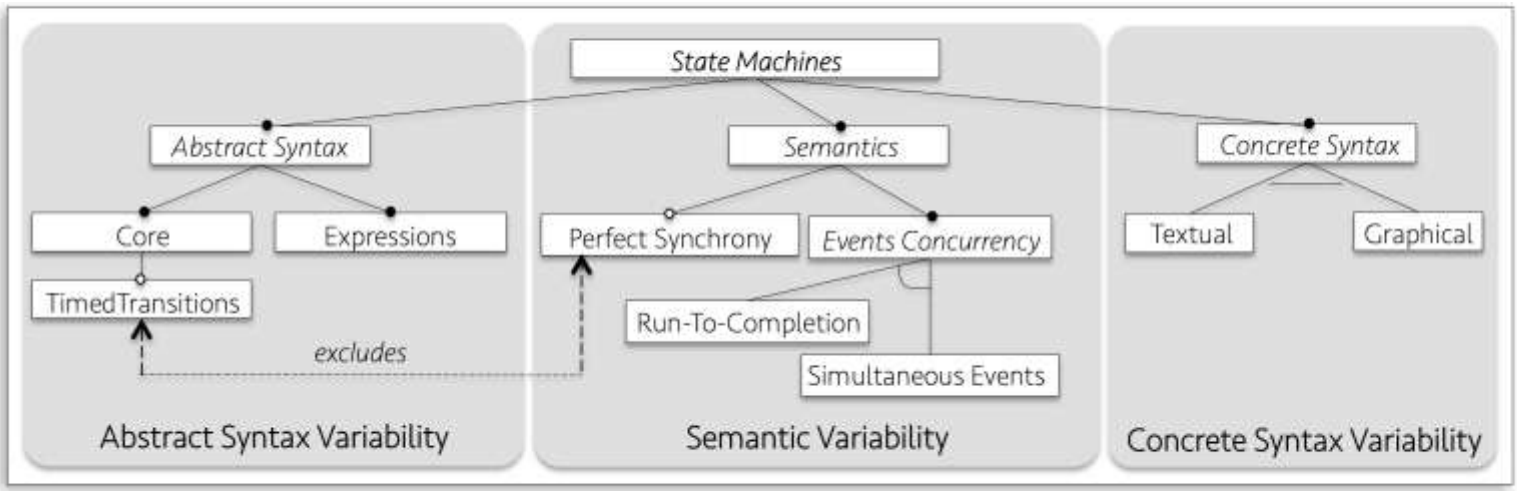


Language interfaces in the wild: micro-grammars (Brown et al.), concepts (De Lara et al.), Microsoft LSP, etc.

Software Language Interfaces

1. Ease the definition and reuse of services
2. Enable language coordination
3. Enable language composition

- A concrete application: *language families*



Leveraging Software Product Lines Engineering in the Development of External DSLs: A Systematic Literature Review
David Méndez-Acuña, José A. Galindo, Thomas Degueule, Benoit Combemale and Benoit Baudry
In *Computer Languages, Systems and Structures* (COMLAN), 2016

Safe Model Polymorphism for Flexible Modeling

Safe Model Polymorphism for Flexible Modeling

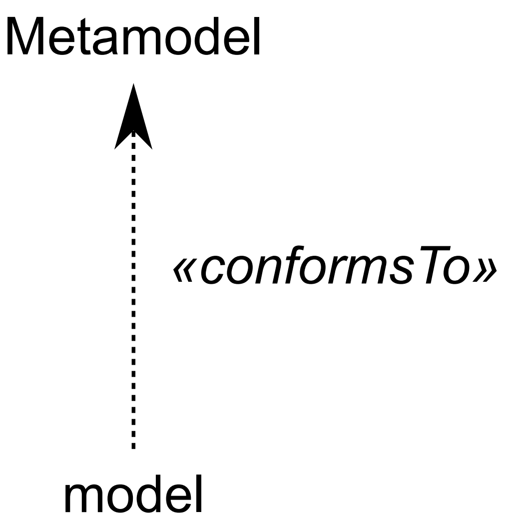
Thomas Degueule, Benoit Combemale, Arnaud Blouin, Olivier Barais and Jean-Marc Jézéquel

In *Computer Languages, Systems and Structures* (COMLAN), 2016

Limits of the Conformance Relation

- In MDE, a metamodel is the cornerstone artifact defining a DSL
- The conformance relation states
  - *Which* models are valid instances of a given DSL
  - *How* these models must be manipulated wrt. this DSL

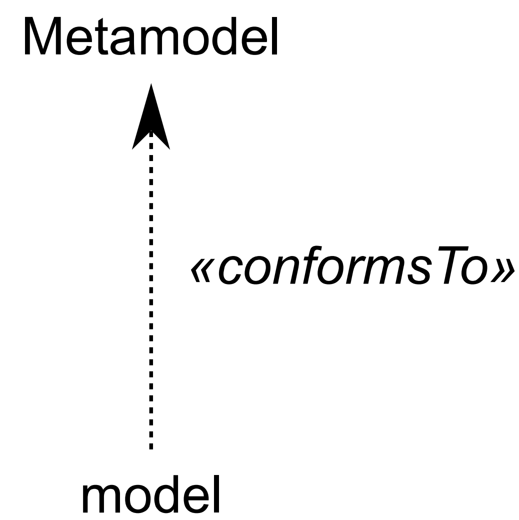
- Theoretical limitations (literature review)
  1. Conformance is based on *instantiation*
  2. Conformance is *nominal*
  3. *A model conforms to one and only one metamodel*



Limits of the Conformance Relation

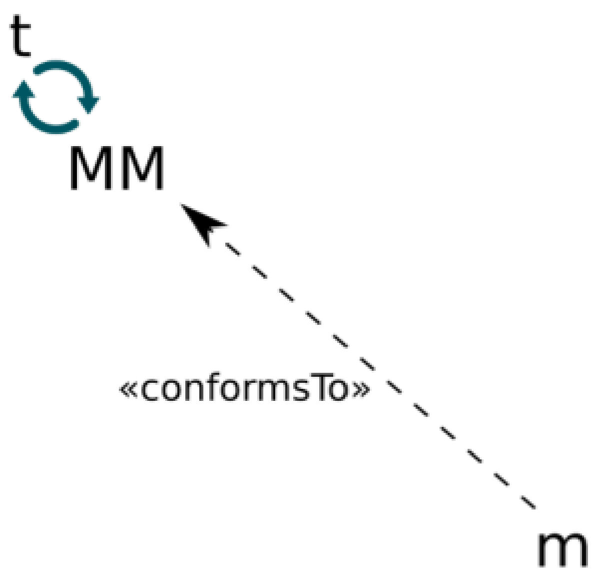
- In MDE, a metamodel is the cornerstone artifact defining a DSL
- The conformance relation states
  - *Which* models are valid instances of a given DSL
  - *How* these models must be manipulated wrt. this DSL

- Analyze UML models publicly available on Github
- Conforming to the UML implementation of Eclipse
- 1651 models – UML2.2 to UML2.5
- Force to bypass the conformance check
- Key findings
  - **7%** of the models are valid wrt. only one version of UML
  - **83%** of the models are valid wrt. every version of UML



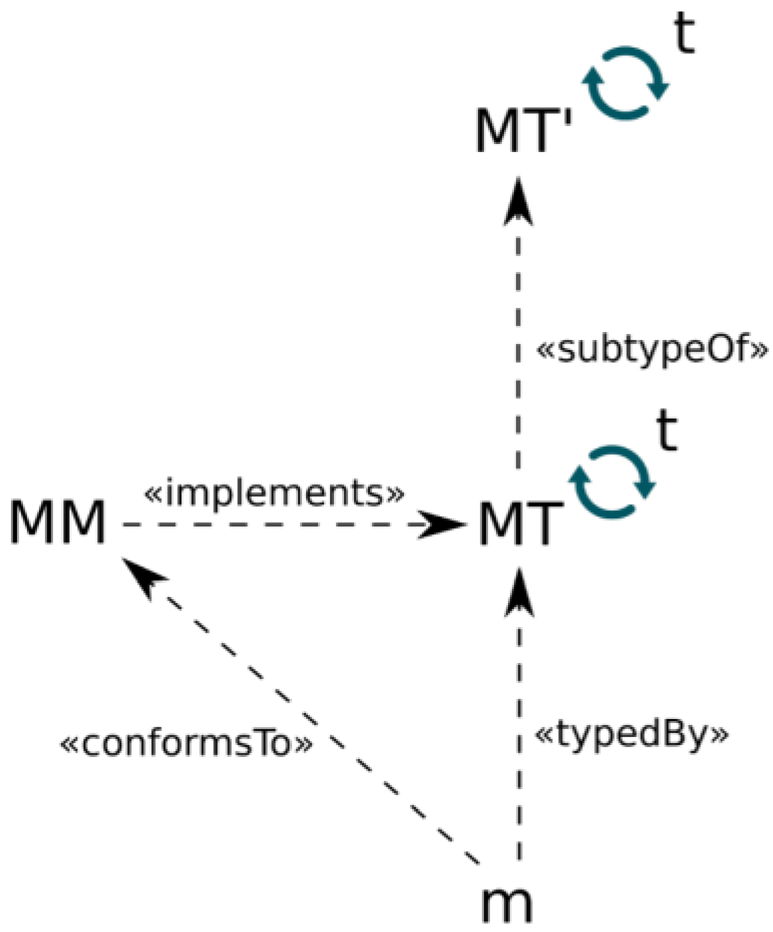
Flexible Modeling beyond the Conformance Relation

- Conformance relation ensures safe manipulation regardless of context
- But it hinders flexibility



Flexible Modeling beyond the Conformance Relation

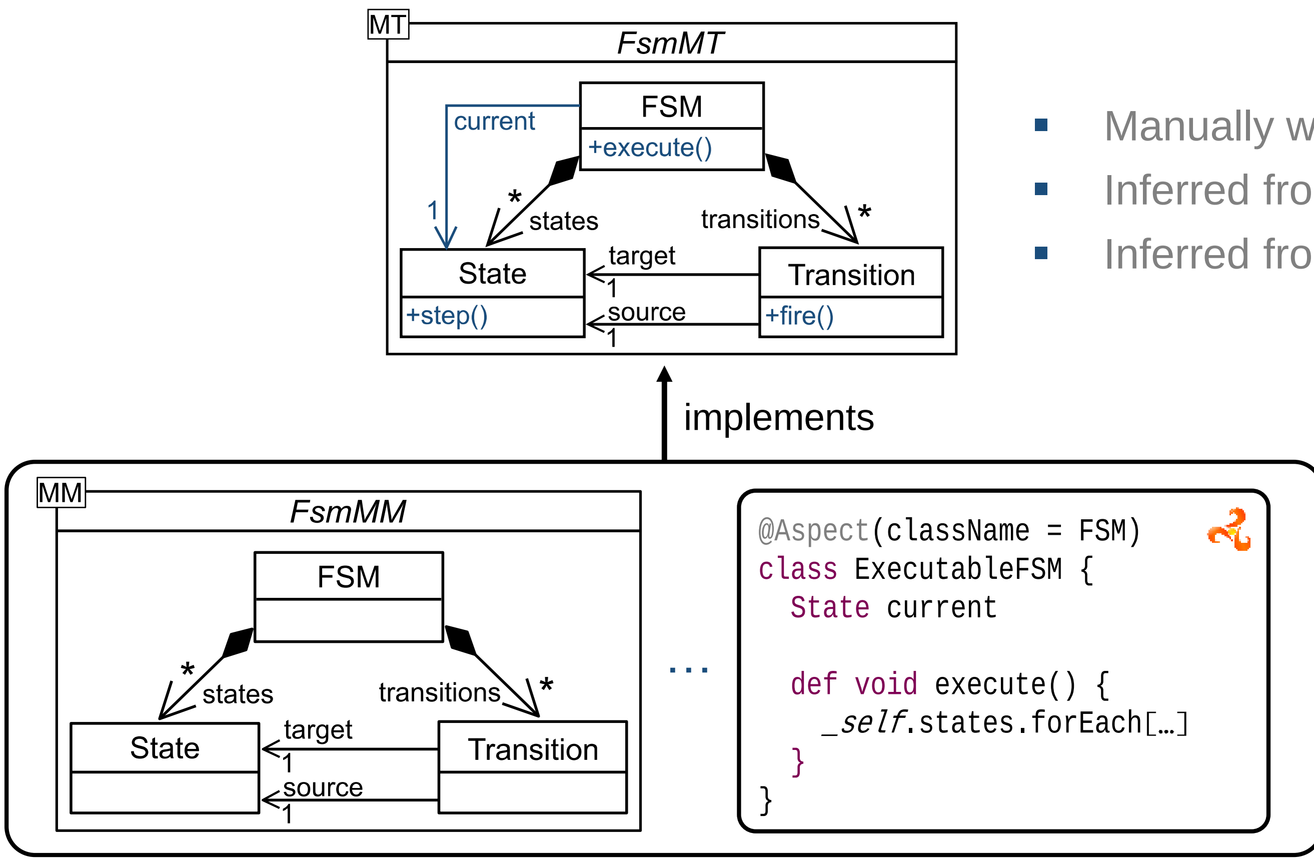
- Conformance relation ensures safe manipulation regardless of context
- But it hinders flexibility



Model Polymorphism

Flexible Modeling beyond the Conformance Relation

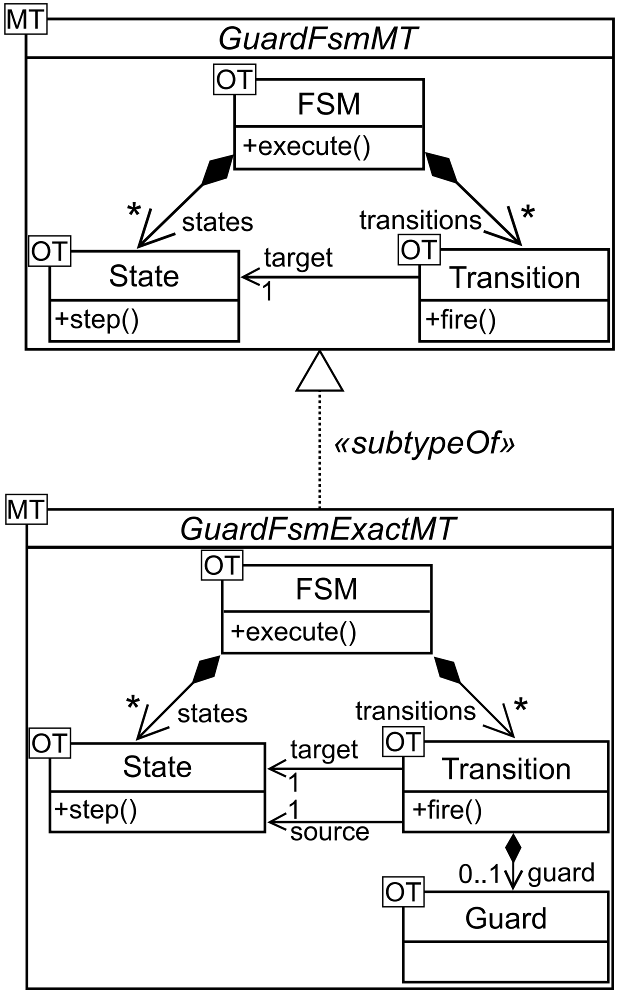
- Model types as structural interfaces atop language implementations



- Manually written
- Inferred from a language
- Inferred from a footprint

Model Subtyping

- States whether models typed by a given MT can be substituted to models typed by another MT $MT \times MT \rightarrow Boolean$
- Different subtyping relations [1]
  - Total isomorphic
  - Partial isomorphic
  - Total non-isomorphic
  - Partial non-isomorphic
- Up to behavioral substitutability [2]
- The choice of a subtyping relation vary with particular needs



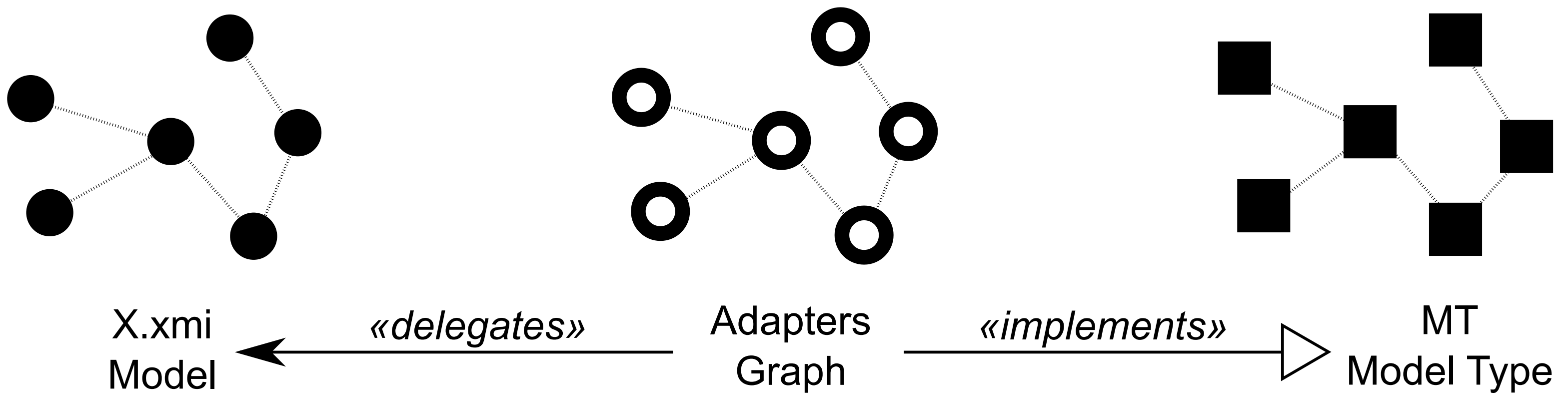
[1] **On model subyping**, Guy et al., *ECMFA*, 2012
[2] **Using model types to support contract-aware model substitutability**, Sun et al., *ECMFA*, 2012

Languages and Model Types in Melange

- Model types defined explicitly or inferred from implementations
- Implementations relations defined explicitly or automatically inferred
  - Based on structural typing
  - Using the total isomorphic subtyping relation
- Simple renaming operator to align structurally dissimilar languages

```
// Explicit model type definition
// e.g. a footprint that captures
// the contract of a transformation
modeltype FsmMT {
  syntax ‘FsmMT.ecore’
}

// Language definition
language GuardFsm {
  syntax    ‘GuardFsm.ecore’
  with      ExecutableFsm
  with      ExecutableState
  with      ExecutableTransition
  exactType GuardFsmMT
}

// Explicit implementation
language OtherFsm implements FsmMT {
  […]
  renaming ‘otherfsm’ to ‘fsm’
}

transformation p-print(FsmMT m) {
  val root = m.contents.head
  m.states.forEach[s | print(s)]
}
```

Seamless Model Polymorphism



"How to fit type groups semantics, structural typing, and family polymorphism in a language (Java) and framework (EMF) that do not support any of them"

Experiment: a Family of FSM Languages

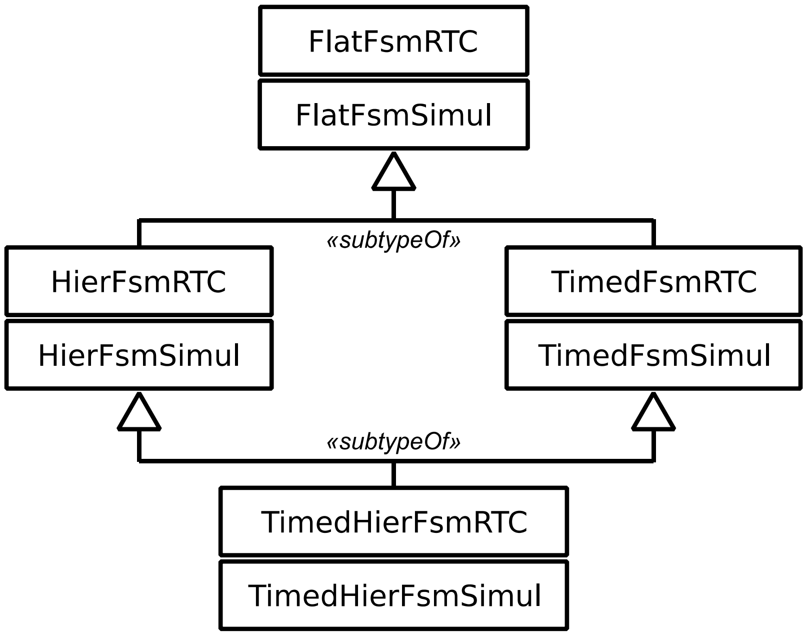
- 4 syntactic variations
  - Simple
  - Hierarchical
  - Timed
  - Timed-Hierarchical
- 2 semantic variations
  - Run-to-completion
  - Simultaneous processing
- 8 FSM language variants

```
// Flat state machine complying to the
// run-to-completion policy, e.g. UML/Rhapsody
language FlatFsmRtc {
  syntax    'metamodels/FlatFsm.ecore'
  with      rtc.ExecutableStateMachine
  with      rtc.ExecutableState
  exactType FlatFsmRtcMT
}

// Hierarchical state machine complying
// to the simultaneous events processing
// policy, e.g. Classical statecharts
language HierarchicalFsmSimultaneous {
  syntax    'metamodels/HierarchicalFsm.ecore'
  with      simultaneous.ExecutableStateMachine
  with      simultaneous.ExecutableState
  with      simultaneous.ExecutableTransition
  exactType HierarchicalFsmSimultaneousMT
}

[6 more omitted]
```

UML vs Classical vs Rhapsody Statecharts: Not all Models are Created Equal
Michelle L. Crane and Jürgen Dingel
In *Software & Systems Modeling (SoSyM), 2007*

Experiment: a Family of FSM Languages

- 4 syntactic variations
  - Simple
  - Hierarchical
  - Timed
  - Timed-Hierarchical
- 2 semantic variations
  - Run-to-completion
  - Simultaneous processing
- 8 FSM language variants



Subtyping relations amongst variants

UML vs Classical vs Rhapsody Statecharts: Not all Models are Created Equal
Michelle L. Crane and Jürgen Dingel
In *Software & Systems Modeling (SoSyM), 2007*

Experiment: a Family of FSM Languages

```
module FlattenFsm;
create OUT : FlatFsm from
       IN  : CompositeFsmMT;
-- Creates a new FlatFsm
rule SM2SM {
  from sm1 : CompositeFsmMT!StateMachine
  to   sm2 : FlatFsm!StateMachine
}
-- Initial states of composite
-- states become regular states
rule Initial2State {
  from is1 : CompositeFsmMT!InitialState
     (not is1.parentState.oclIsUndefined())
  to is2 : FlatFsm!State(
    stateMachine <- is1.stateMachine,
    name <- is1.name)
}
-- Resolves a transition originating from
-- a composite state
rule T2TB {
  from t1: CompositeFsmMT!Transition,
     src : CompositeFsmMT!CompositeState,
     trg : CompositeFsmMT!State,
     c   : CompositeFsmMT!State (
       t1.source = src and
       t1.target = trg and
       c.parentState = src and
       not trg.oclIsTypeOf(
         CompositeFsmMT!CompositeState))
  to t2 : FlatFsm!Transition (
       name <- t1.name,
       stateMachine <- t1.stateMachine,
       source <- c,
       target <- trg )
}
```

TransfoFsm.qvto

```
modeltype FsmMT uses “http://fsmmt/”;
modeltype Fsm   uses “http://fsm/”;

transformation dummyInvert(
in inFsm : FsmMT, out outFsm : Fsm);

main() {
  inFsm.rootObjects()[FsmMT::FSM] ->
  map mapFSM();
}
mapping FsmMT::FSM::mapFSM():Fsm::FSM {
  ownedState := self.ownedState -> map
mapState();
  initialState := self.finalState ->
first().map mapState();
  finalState := self.initialState.map
mapState();
}
mapping FsmMT::State::mapState() :
Fsm::State {
  name := self.name;
  outgoingTransition :=
    self.incomingTransition -> map
    mapTransition();
}
mapping FsmMT::Transition::mapTransition() :
Fsm::Transition {
  input  := self.input;
  output := self.output;
  target := self.source.map mapState();
}
```

ExecuteFsm.java

```
// Delegate the execution of the state
// machine "fsm" to the "execute" method
// of its operational semantics.
public void execute(
  StateMachine fsm, String input) {
  // Dynamically dispatched on the actual
  // language implementation of execute()
  root.execute(input);
}

List<String> models = new ArrayList<>();
models.add(
 "melange:/m1.flat?mt=FlatFsmRtcMT");
models.add(
 "melange:/m2.timed?mt=FlatFsmRtcMT");
models.add(
 "melange:/m3.hier?mt=FlatFsmRtcMT");
models.add(
 "melange:/m4.timedhier?mt=FlatFsmRtcMT");
ResourceSet rs = new ResourceSetImpl();

// Load the model pointed by the given
// URI, retrieve its root StateMachine,
// and execute it
for (String uri : models) {
  Resource res = rs.getResource(uri,true);
  StateMachine root = (StateMachine)
    res.getContents().get(0);
  execute(res, "{x;y;z;o;p;q}");
}
```

Modular & Reusable Development of DSLs

Melange: A Meta-language for Modular and Reusable Development of DSLs
Thomas Degueule, Benoit Combemale, Arnaud Blouin, Olivier Barais and Jean-Marc Jézéquel
In *Proceedings of the 8th International Conference on Software Language Engineering (SLE'15)*, 2015

Overview

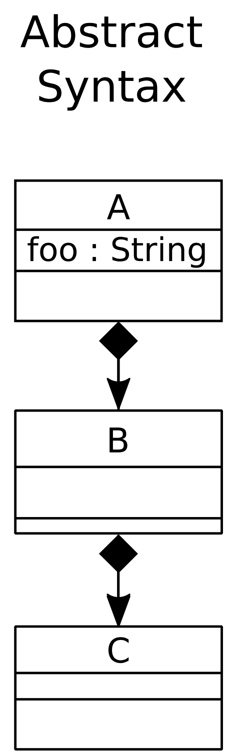
- Existing DSLs can be reused when developing new ones
  - Reuse syntax, semantics, tools & services
  - *Reuse is not enough, context matters!*

- Finely tune the resulting DSLs
  - To comply with new requirements
  - Or the specificities of a new domain of application
  - e.g. restricting or extending expressiveness, specializing semantics...

An algebra of operators for assembling legacy DSLs and customizing them at a fine-grained level, while ensuring type groups consistency and tool reuse

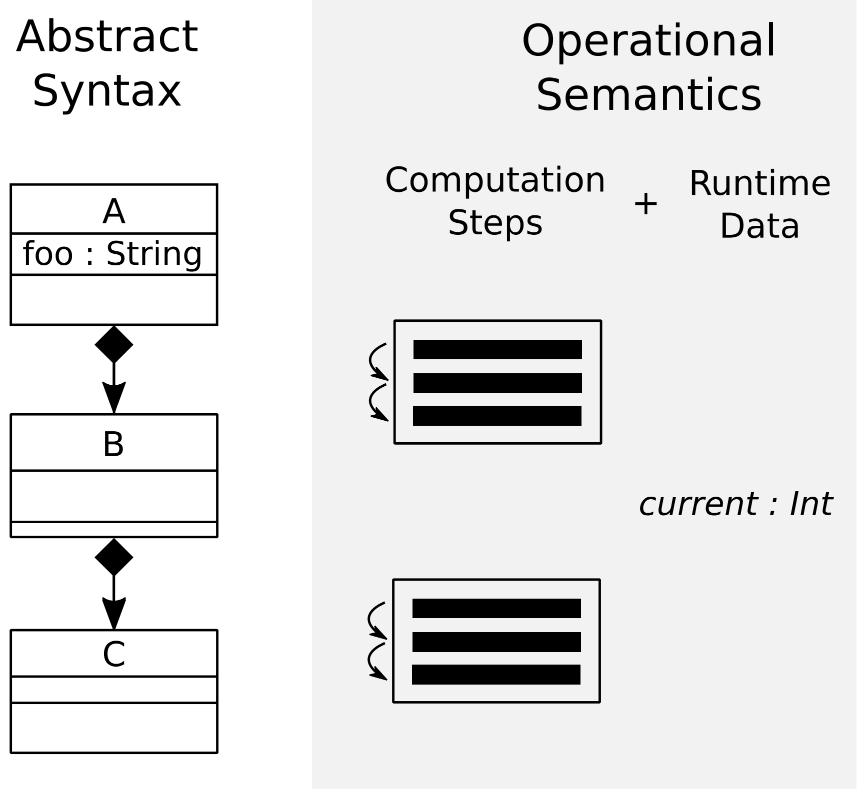
Hypothesis on Language Definition

- A metamodel defines the *AS*



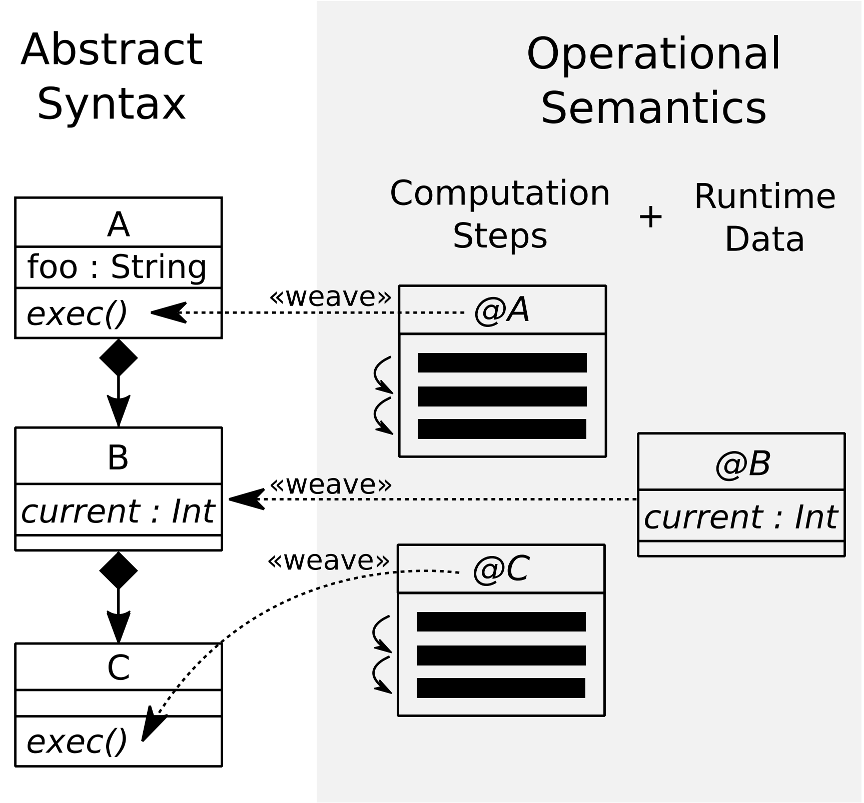
Hypothesis on Language Definition

- A metamodel defines the *AS*
- *Sem* consists of computation steps and runtime data



Hypothesis on Language Definition

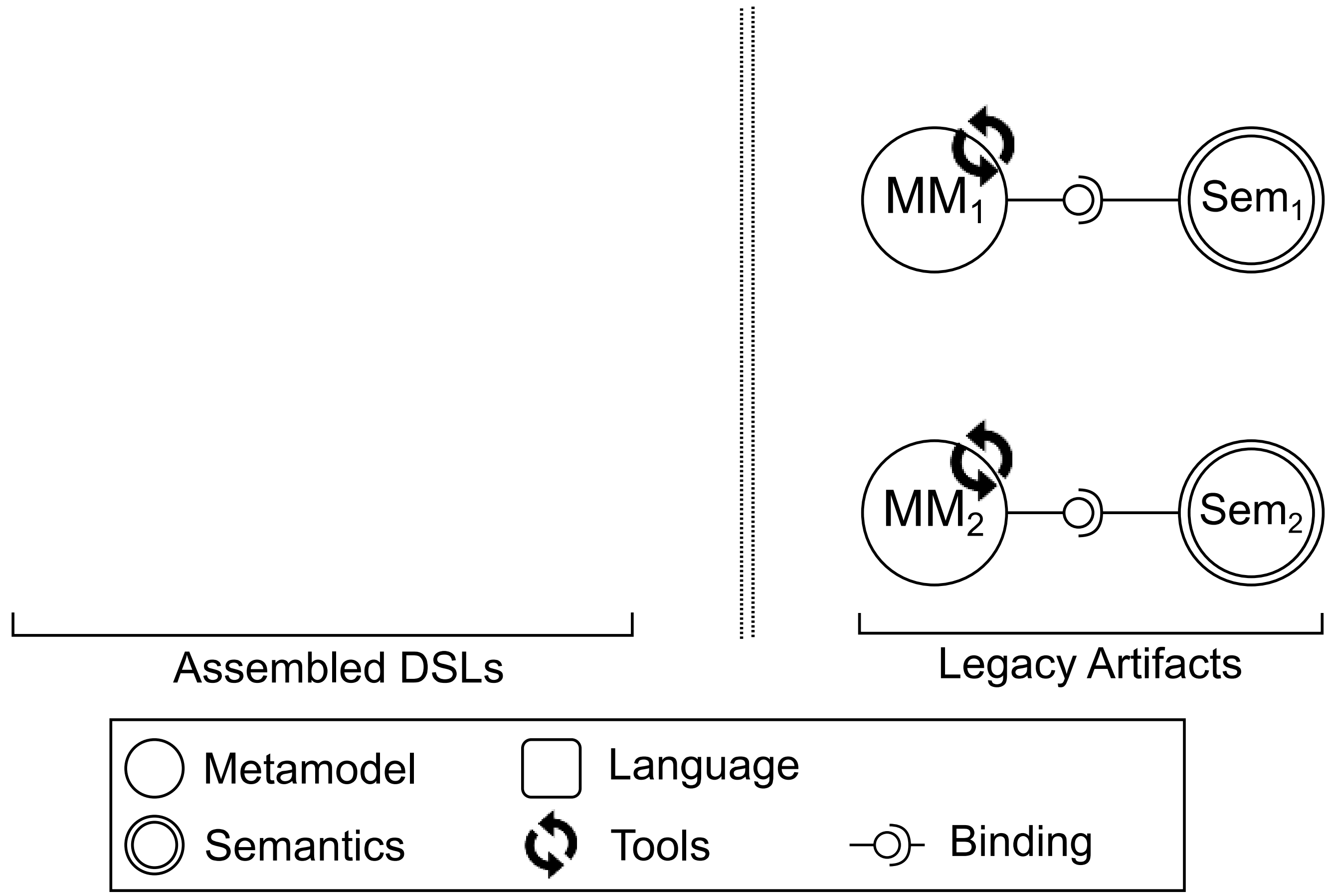
```
@Aspect(className = B)
class AspectB {
  int current

  def int exec() {
    _self.myCs.forEach[…]
  }
}
```



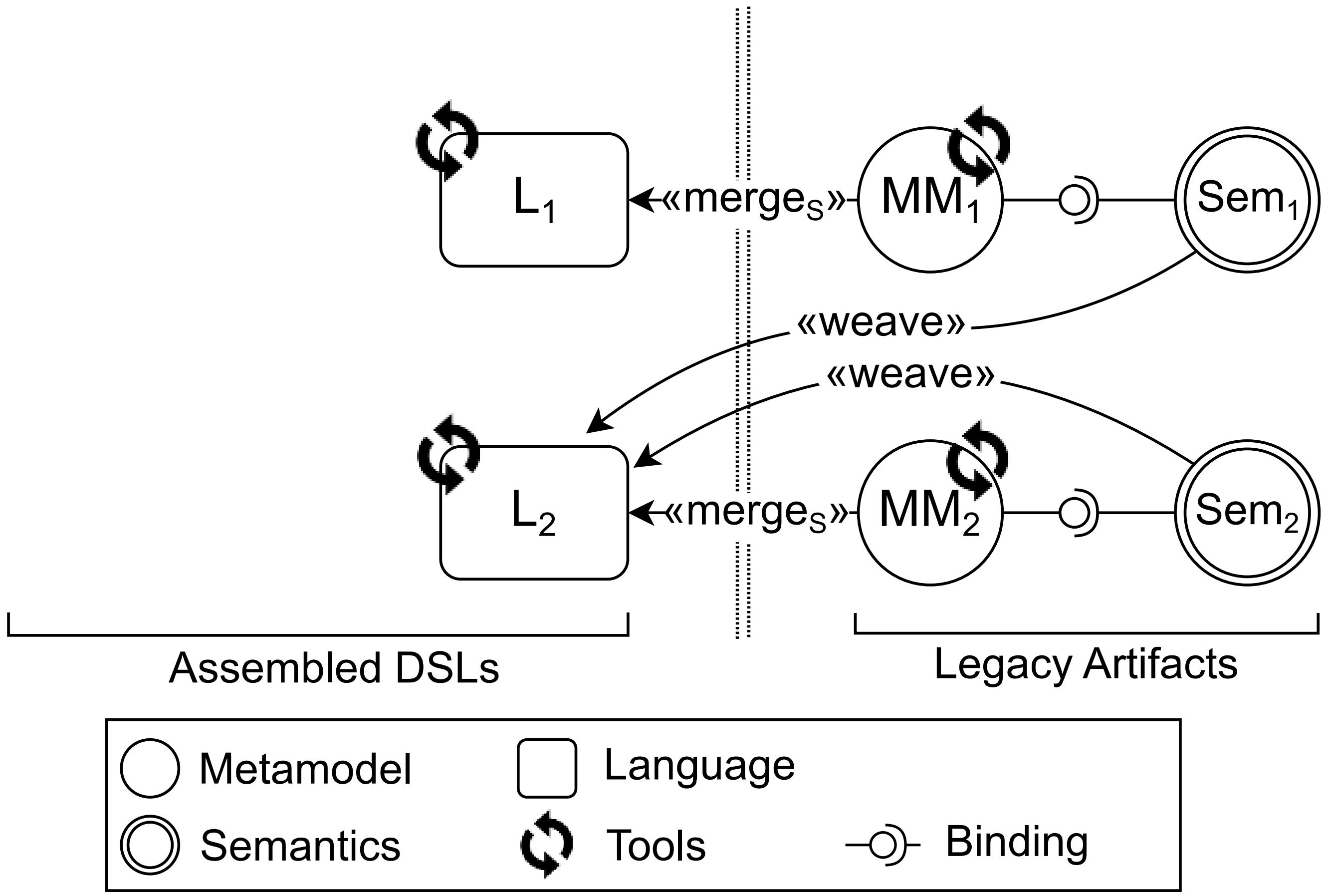
- Aspect-oriented modeling:
  Sem is woven directly in the *AS*
- Interpreter/visitor pattern

Mashup of Meta-languages and its Implementation in the Kermeta Language Workbench
Jean-Marc Jézéquel, Benoit Combemale, Olivier Barais, Martin Monperrus and François Fouquet
In *Software & Systems Modeling (SoSyM), 2015*

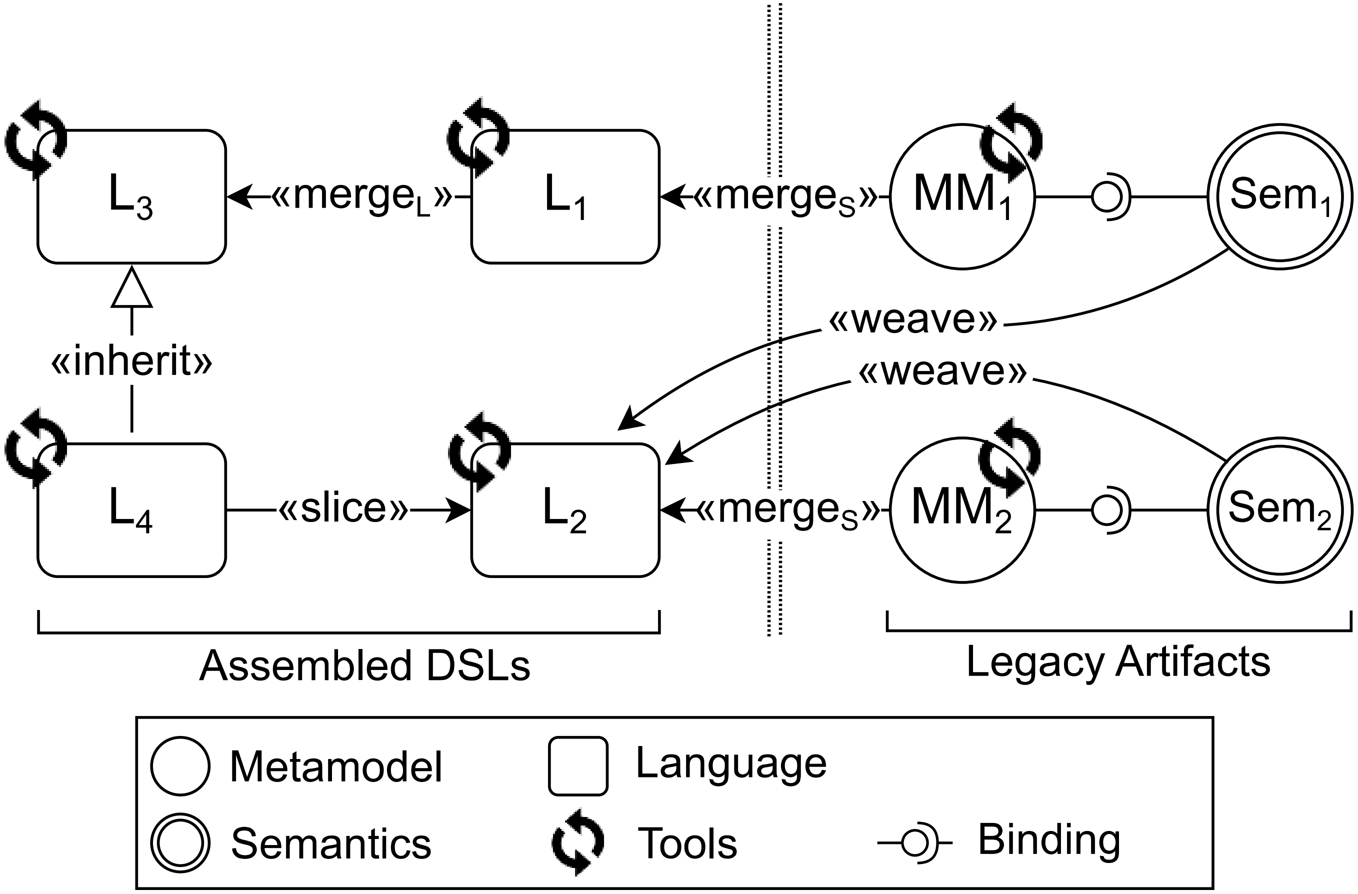
Approach Overview



Approach Overview

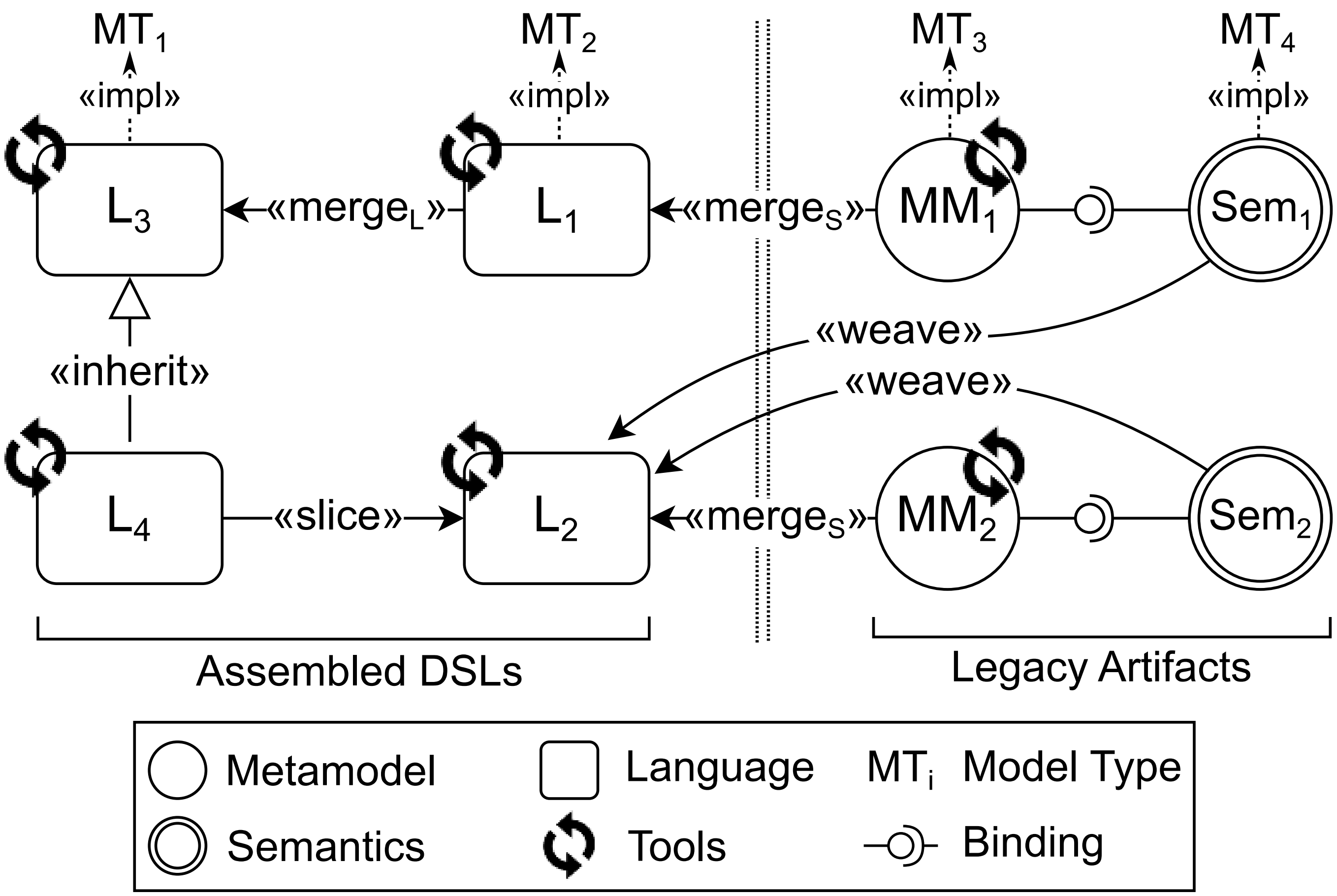


Approach Overview



merge, *slice*, and *inherits* inspired from language composition taxonomies, e.g.
Language Composition Untangled
Sebastian Erdweg, Paolo G. Giarrusso, Tillmann Rendel
In *LDTA, 2012*

Approach Overview



Language Definition

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

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

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

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

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

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

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

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

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

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

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

$$MT'' = MT \circ MT' \text{ and}$$

$$MT'' <: MT'$$

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

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

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

$$MT_1 <: MT_2,$$

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

Language Definition

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

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

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

$$sig(Sem) \triangleq \bigcup^{\circ}_{A^t_i \in Sem} sig(A^t_i)$$

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

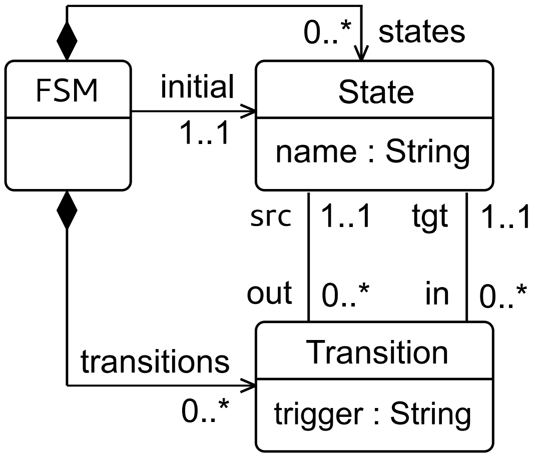
$$\mathcal{L} \overset{m}{\longleftarrow} AS' = \langle AS \circ AS', Sem, MT \circ AS' \rangle$$

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

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

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

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




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


}
```

Language Definition

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

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

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

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

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

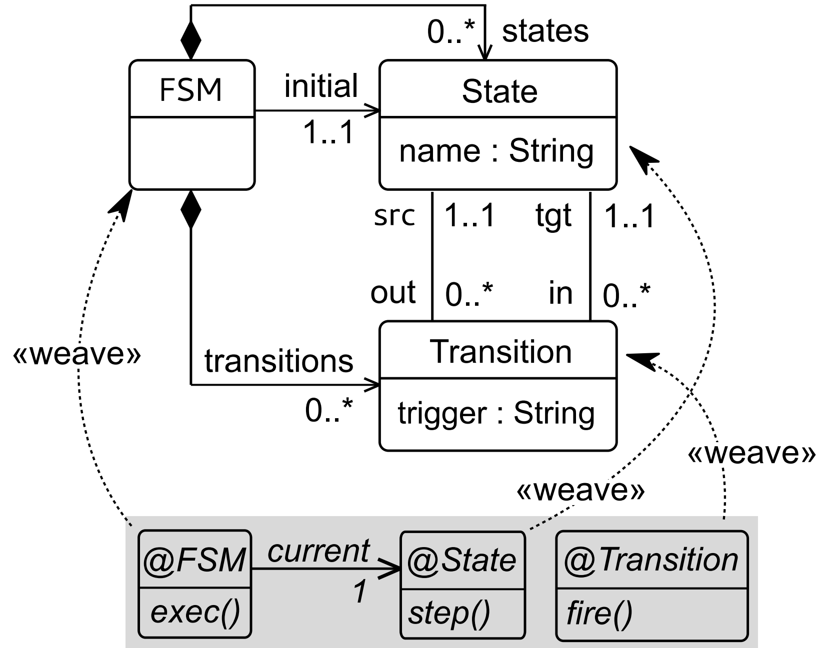
$$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS' \rangle$$

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

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

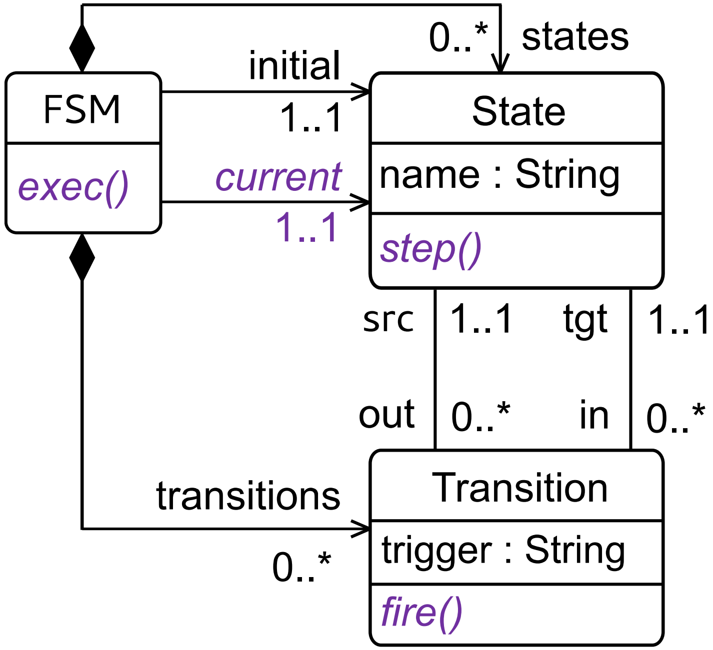
$$\mathcal{L} \oplus \mathcal{L}' = \langle AS \circ AS', Sem' \bullet Sem, MT'' \rangle \text{ where}$$
$$MT'' = MT \circ MT' \text{ and}$$
$$MT'' <: MT'$$

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




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

}
```

Model Types



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

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

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

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

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

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

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

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

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

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

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

$$MT'' = MT \circ MT' \text{ and}$$

$$MT'' <: MT'$$

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

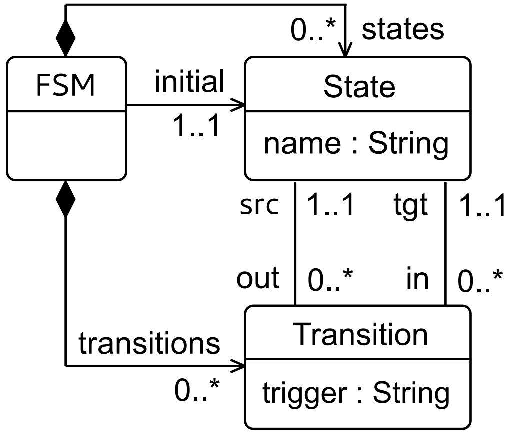
$$AS_2 \triangleq \lambda_-^+(AS_1, c),\ AS_2 \subseteq AS_1,$$

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

$$MT_1 <: MT_2,$$

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

Syntax Merging



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

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

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

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

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

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

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

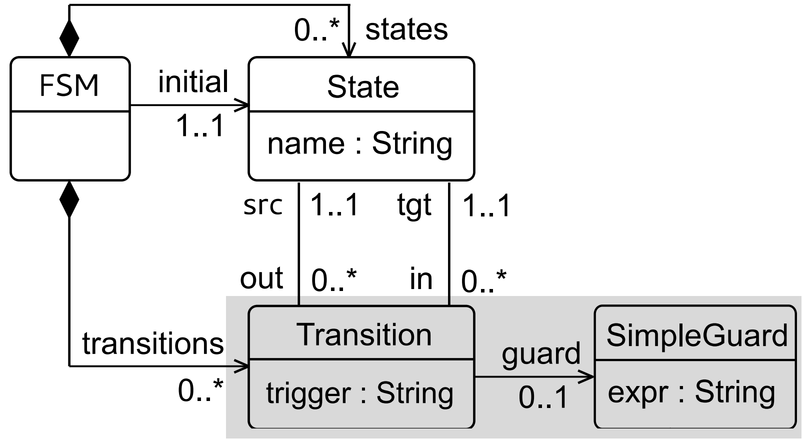
$$\mathcal{L} \uplus \mathcal{L}' = \langle AS \circ AS', Sem \bullet Sem', MT \circ MT' \rangle$$

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

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

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

    with ExecutableFsm
    with ExecutableState
    with ExecutableTransition


    exactType GuardedFsmMT
}
```

Syntax Merging



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

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

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

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

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

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

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

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

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

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

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

$$MT'' = MT \circ MT' \text{ and}$$

$$MT'' <: MT'$$

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

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

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

$$MT_1 <: MT_2,$$

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


    exactType GuardedFsmMT
}
```

Semantics Weaving

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

$$Sem(\mathcal{L}) \triangleq (A^t_i \in Aspects) \text{ where}$$

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

$$\forall A^t_i, A^t_j \in Sem(\mathcal{L}) : A^t_i \lhd A^t_j \implies i > j$$

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

$$sig(Sem) \triangleq \overset{\circ}{\bigcup_{A^t_i \in Sem}} sig(A^t_i)$$

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

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

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

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

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

$$MT'' = MT \circ MT' \text{ and}$$

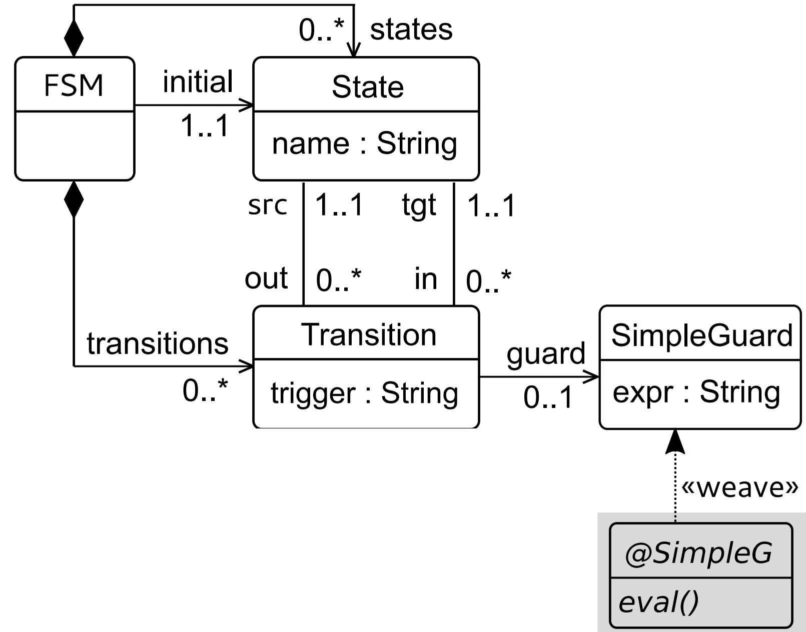
$$MT'' <: MT'$$

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

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

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

$$MT_1 <: MT_2,$$




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

    exactType GuardedFsmMT
}
```

Semantics Weaving

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

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

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

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

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

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

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

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

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

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

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

$$MT'' = MT \circ MT' \text{ and}$$

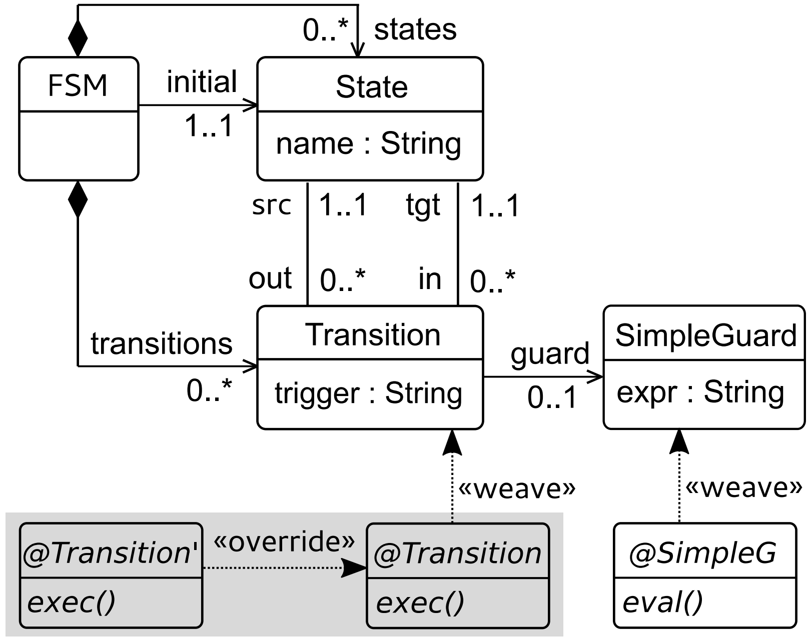
$$MT'' <: MT'$$

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

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

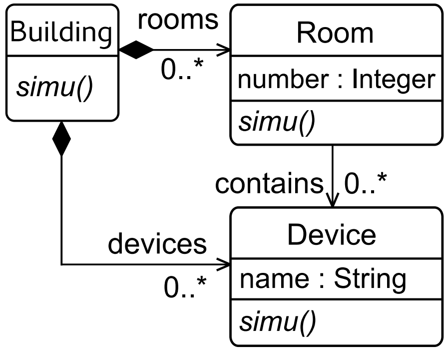
$$Sem_2 \triangleq \{A_i^t \in Sem_1, fp(A_i^t, AS_1) \subseteq AS_2\},$$

$$MT_1 <: MT_2,$$




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

Language Merging



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

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

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

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

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

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

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

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

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

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

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

$$MT'' = MT \circ MT' \text{ and}$$

$$MT'' <: MT'$$

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

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

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

$$MT_1 <: MT_2,$$

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


    exactType BuildingMT
}
```

Language Merging

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

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

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

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

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

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

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

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

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

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

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

$$MT'' = MT \circ MT' \text{ and}$$

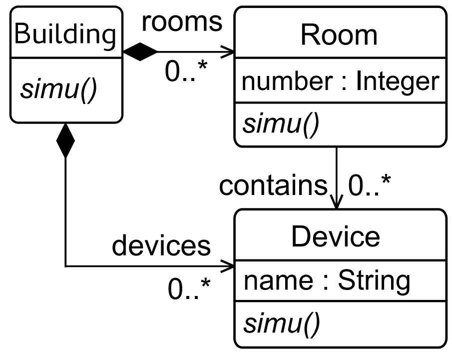
$$MT'' <: MT'$$

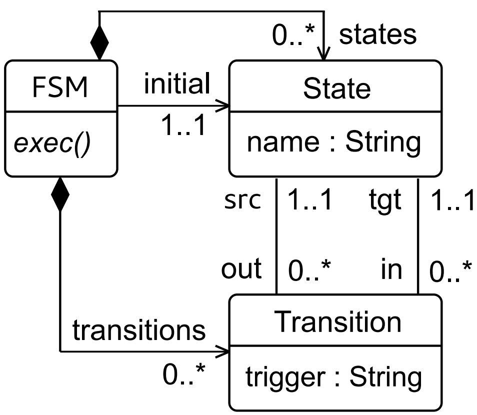
$$\Lambda_{-}^{+}(\mathcal{L}_1, c) = \langle AS_2, Sem_2, MT_2 \rangle, \text{ where:}$$

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

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

$$MT_1 <: MT_2,$$






```
language Building {
    syntax ‘Building.ecore’
    with SimulatorAspect...
    merge Fsm

    exactType BuildingMT
}
```

Language Merging

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

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

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

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

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

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

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

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

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

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

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

$$MT'' = MT \circ MT' \text{ and}$$

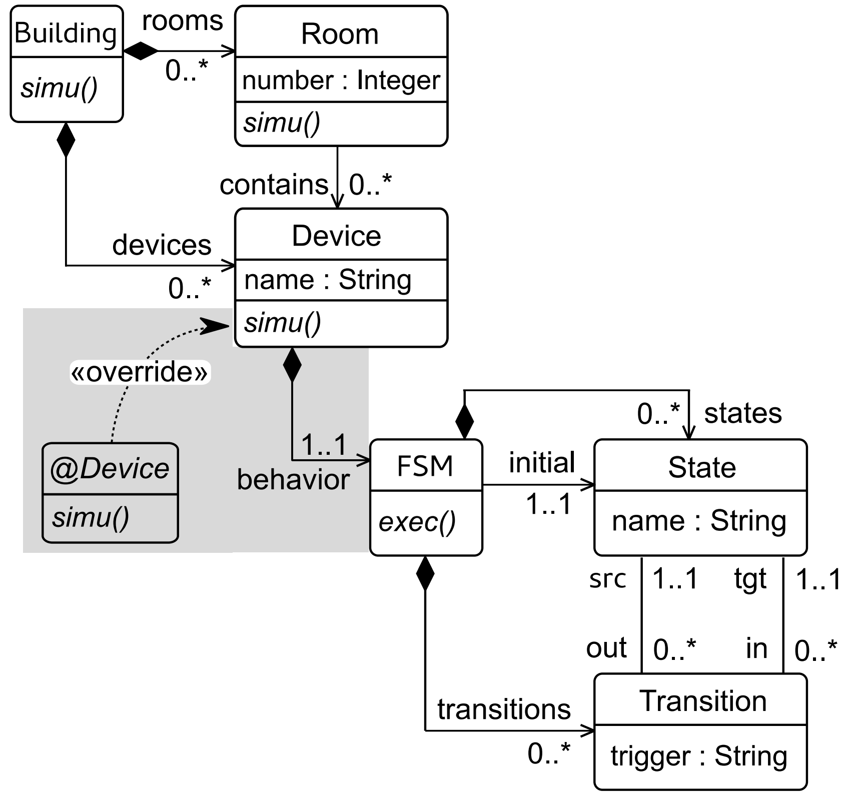
$$MT'' <: MT'$$

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

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

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

$$MT_1 <: MT_2,$$




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

Language Inheritance

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

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

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

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

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

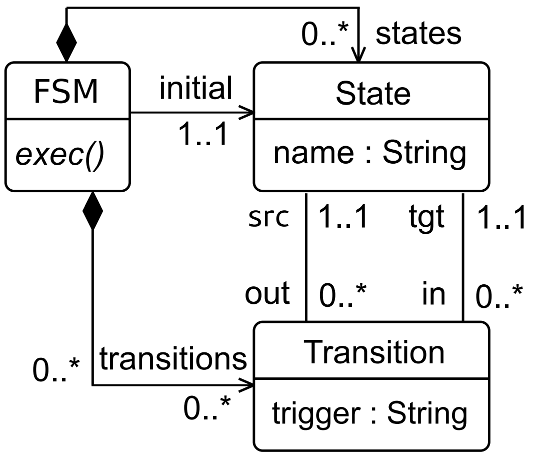
$$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS' \rangle$$

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

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

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

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




```
language TimedFsm inherits Fsm {


    exactType TimedFsmMT
}
```

Language Inheritance

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

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

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

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

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

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

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

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

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

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

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

$$MT'' = MT \circ MT' \text{ and}$$

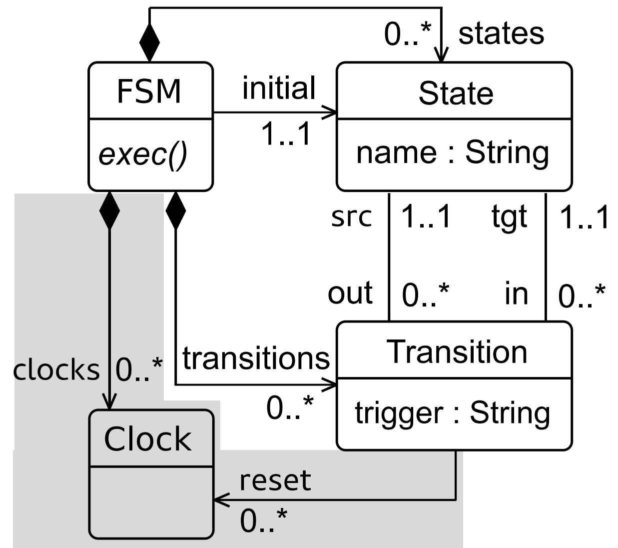
$$MT'' <: MT'$$

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

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

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

$$MT_1 <: MT_2,$$




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


  exactType TimedFsmMT
}
```

Language Inheritance

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

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

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

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

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

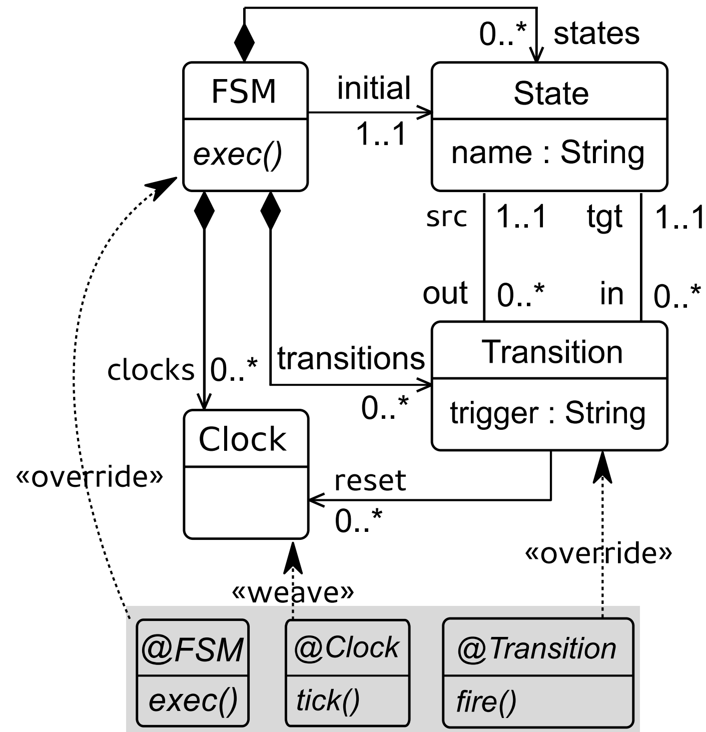
$$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS' \rangle$$

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

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

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

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




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

Language Slicing

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

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

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

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

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

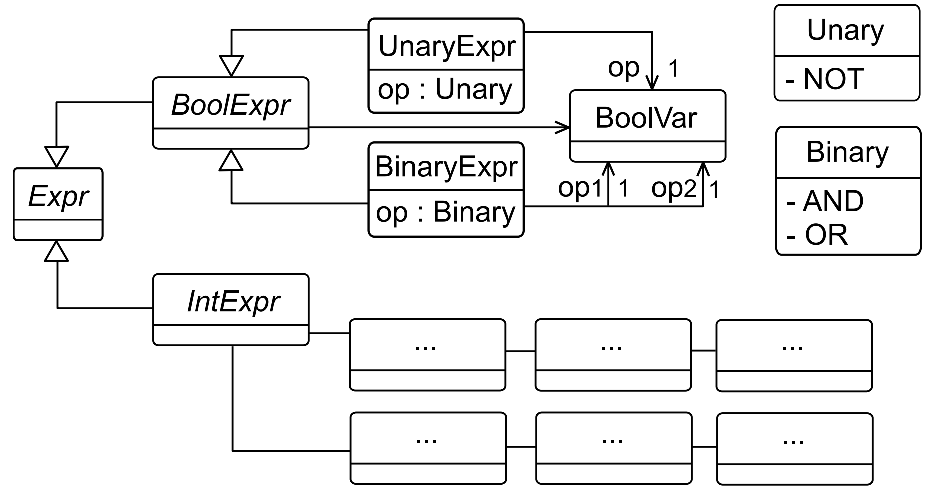
$$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS' \rangle$$

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

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

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

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




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

Kompren: Modeling and Generating Model Slicers
Arnaud Blouin, Benoit Combemale, Benoit Baudry and Olivier Beaudoux
In *Software & Systems Modeling (SoSyM), 2015*

Language Slicing

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

$$Sem(\mathcal{L}) \triangleq (A^t_i \in Aspects) \text{ where}$$

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

$$\forall A^t_i, A^t_j \in Sem(\mathcal{L}) : A^t_i \lhd A^t_j \implies i > j$$

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

$$sig(Sem) \triangleq \bigcup^{\circ}_{A^t_i \in Sem} sig(A^t_i)$$

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

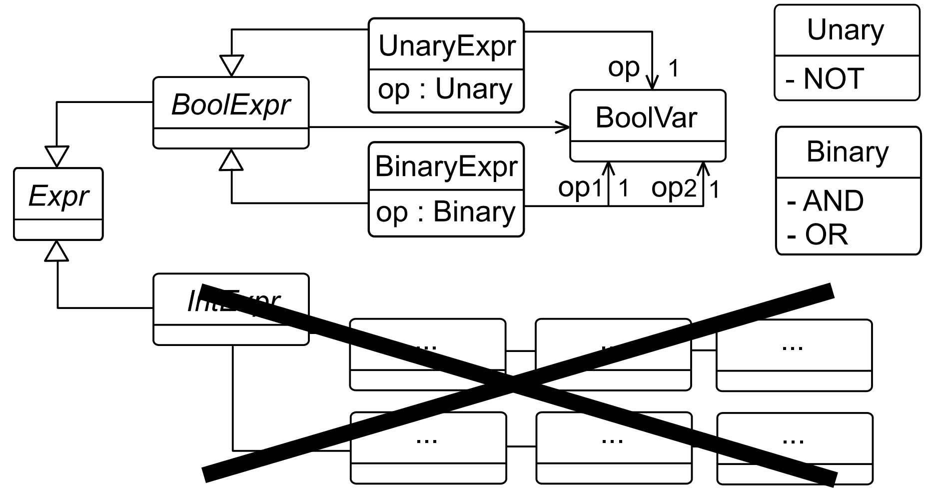
$$\mathcal{L} \xleftarrow{m} AS' = \langle AS \circ AS', Sem, MT \circ AS' \rangle$$

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

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

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

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

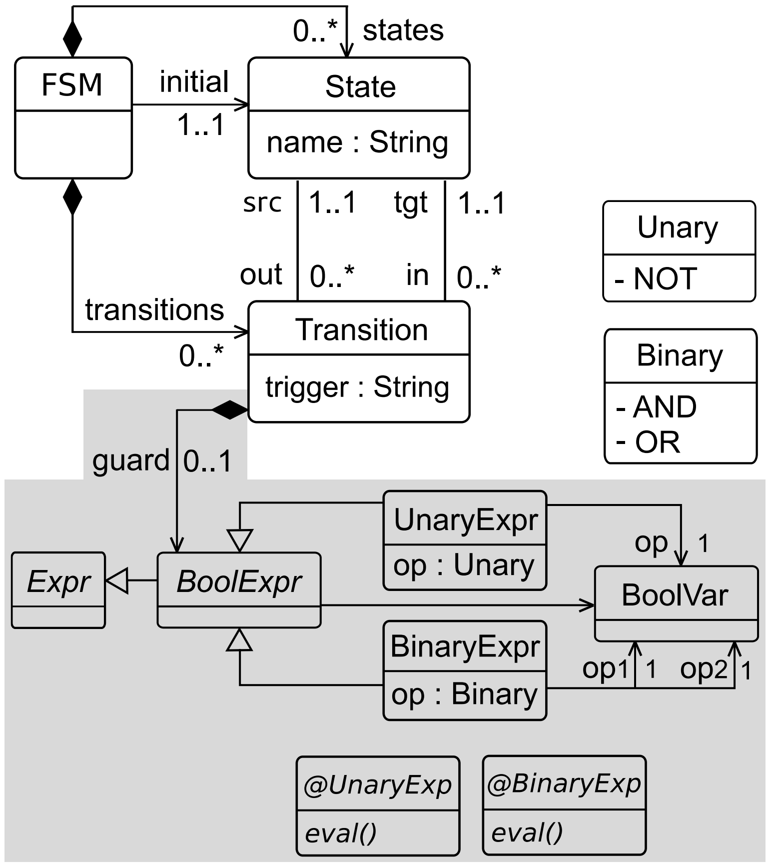



```
language Expressions {
    syntax 'Expressions.ecore'
    with EvaluateBoolean
    with EvaluateInteger
    exactType ExpressionsMT
}

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

Kompren: Modeling and Generating Model Slicers
Arnaud Blouin, Benoit Combemale, Benoit Baudry and Olivier Beaudoux
In *Software & Systems Modeling (SoSyM), 2015*

Wrap-up

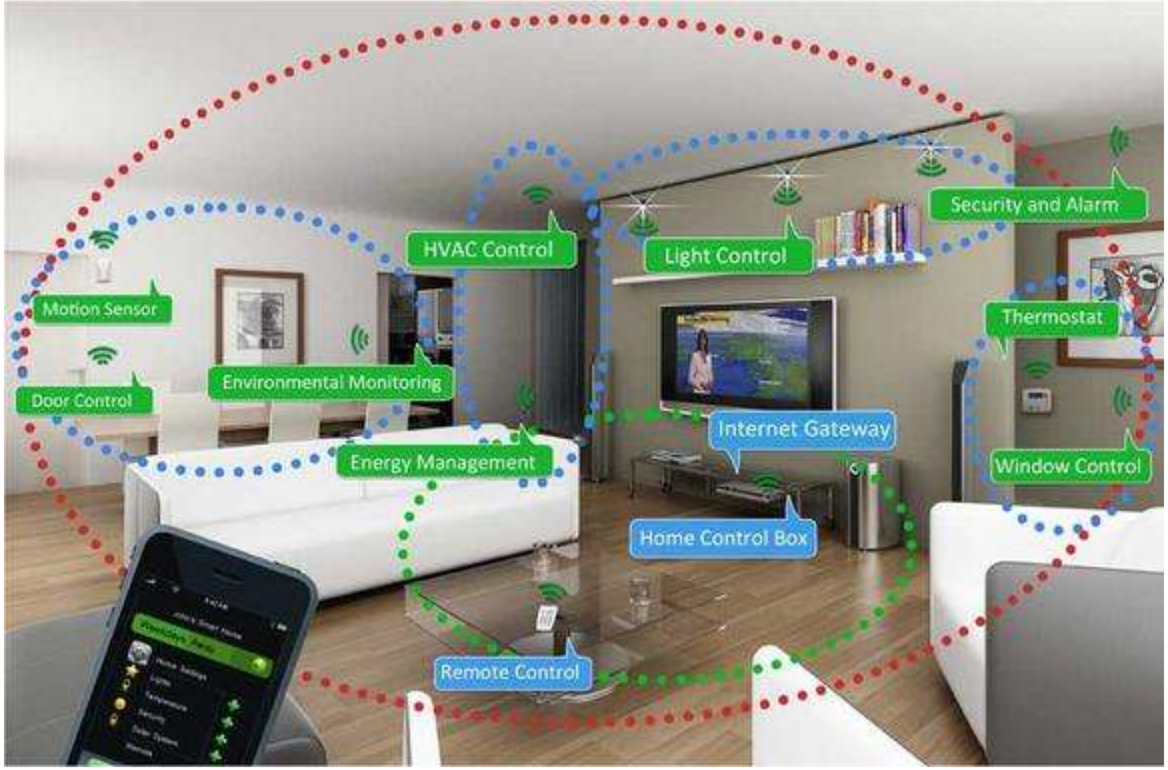
- Language extension with *language inheritance*
- Language unification with *language merging*
- Language restriction with *language slicing*
- *Syntax merging* and *aspect weaving* as fine-grained customization mechanisms



Language Composition Untangled
Sebastian Erdweg, Paolo G. Giarrusso, Tillmann Rendel
In *LDTA, 2012*

Experiment: A Modeling Language for IoT

- Enable modeling the behavior of communicating sensors built on top of resource-constrained devices (e.g. Arduino, Raspberry Pi)
- Provide appropriate simulators to experiment different scenarios
- Objective: reuse existing DSLs whenever possible



Requirements for the IoT Language

1. Model sensors' interface
   - OMG Interface Description Language

From Github

2. Model sensors' control flow (sketch)
   - UML's Activity Diagram

From TTC'15

3. Express sensors' actions
   - Lua programming language

From Github

Companion webpage: http://melange-lang.org/sle15/

```
language IDL {
  syntax 'IDL.ecore'
}
```

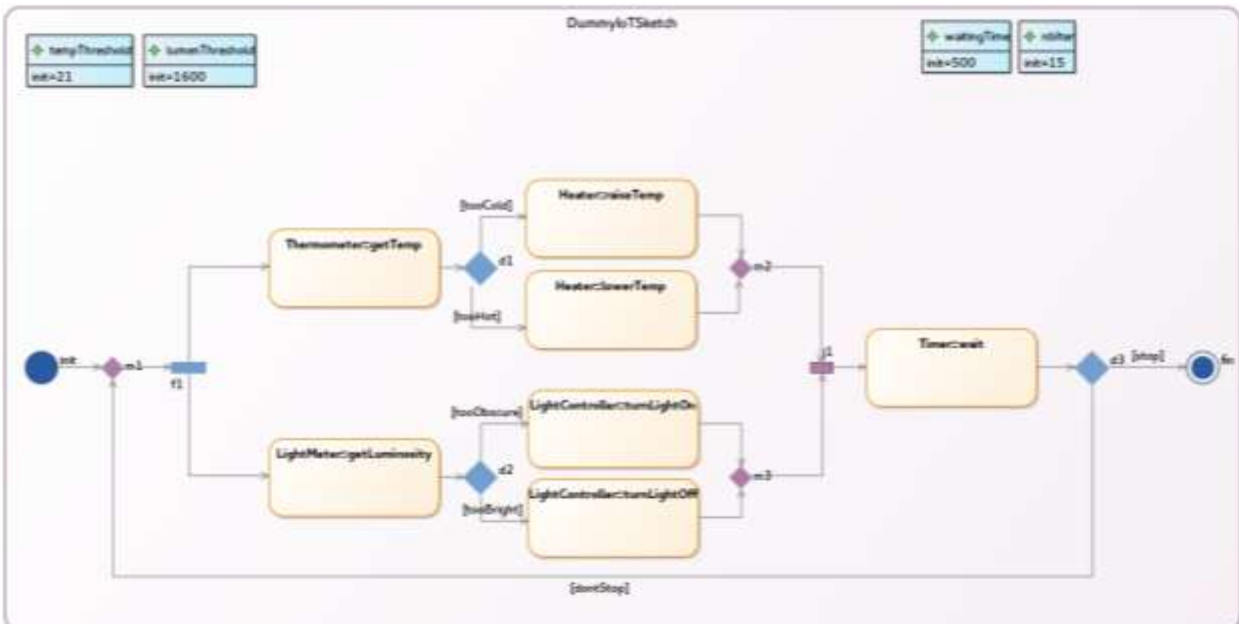
```
language ActivityDiagram {
  syntax 'ActivityDiagram.ecore'
  syntax 'RuntimeModel.ecore'
  with ad.semantics.*
}
```

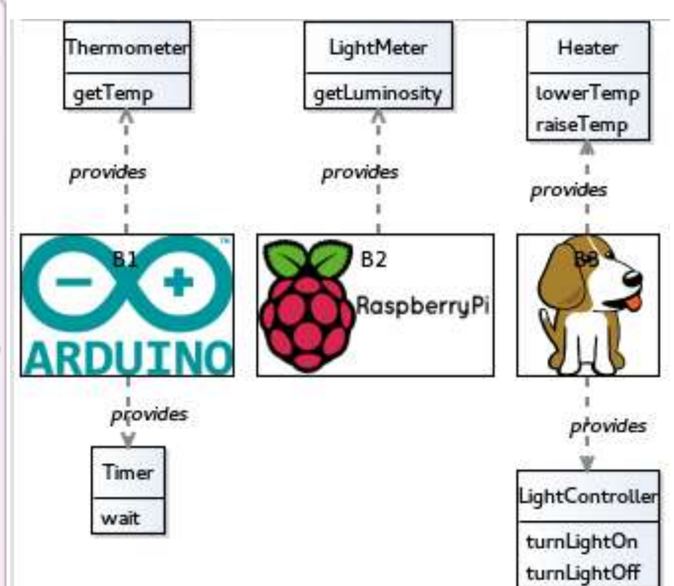
```
language Lua {
  syntax 'Lua.ecore'
  with lua.semantics.*
}
```

```
language IoT {
  syntax 'IoT.ecore'
  slice Idl on ['OperationDef', 'PrimitiveDef']
    renaming { 'idlmm' to 'iot' }
  merge Lua
    renaming { 'lua' to 'iot' }
  merge ActivityDiagram
    renaming { 'activitydiagram' to 'iot' }
  with iot.glue.OpaqueActionGlue
  with iot.glue.OperationDefGlue
}
```

The IoT Language in Melange

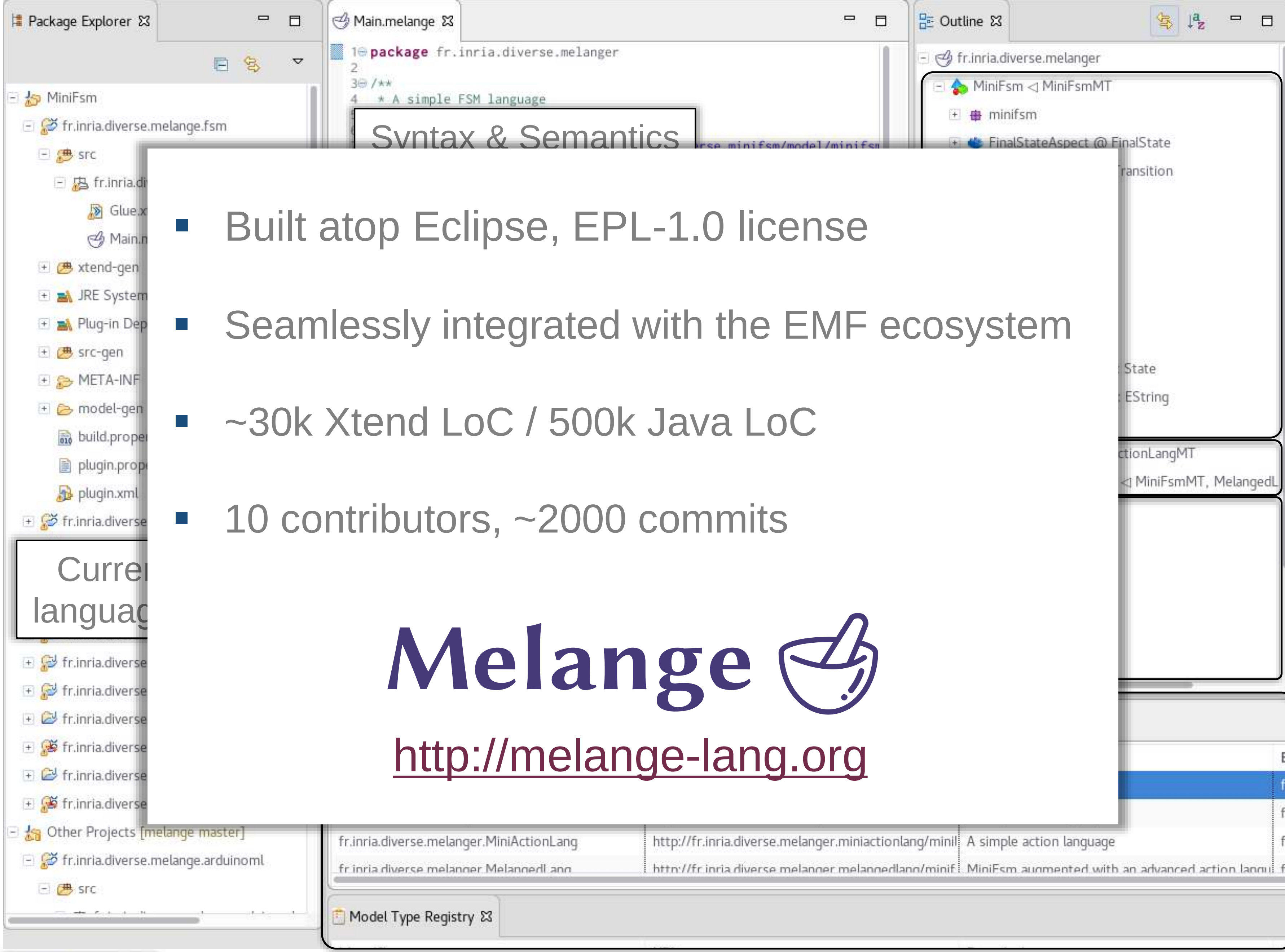
- Full EMF compliance (e.g. integrated for free within the GEMOC studio)
- Reuse of tools & services between the base languages and the IoT lang
- Glue: ~30 LoC (mainly Lua – ActivityDiagram context translation)





Reusing Legacy DSLs with Melange
Thomas Degueule, Benoit Combemale, Arnaud Blouin, Olivier Barais
In *Proceedings of the 15th workshop on Domain-Specific Modeling (DSM'15), 2015*

The Melange Language Workbench



- Built atop Eclipse, EPL-1.0 license
- Seamlessly integrated with the EMF ecosystem
- ~30k Xtend LoC / 500k Java LoC
- 10 contributors, ~2000 commits

Melange

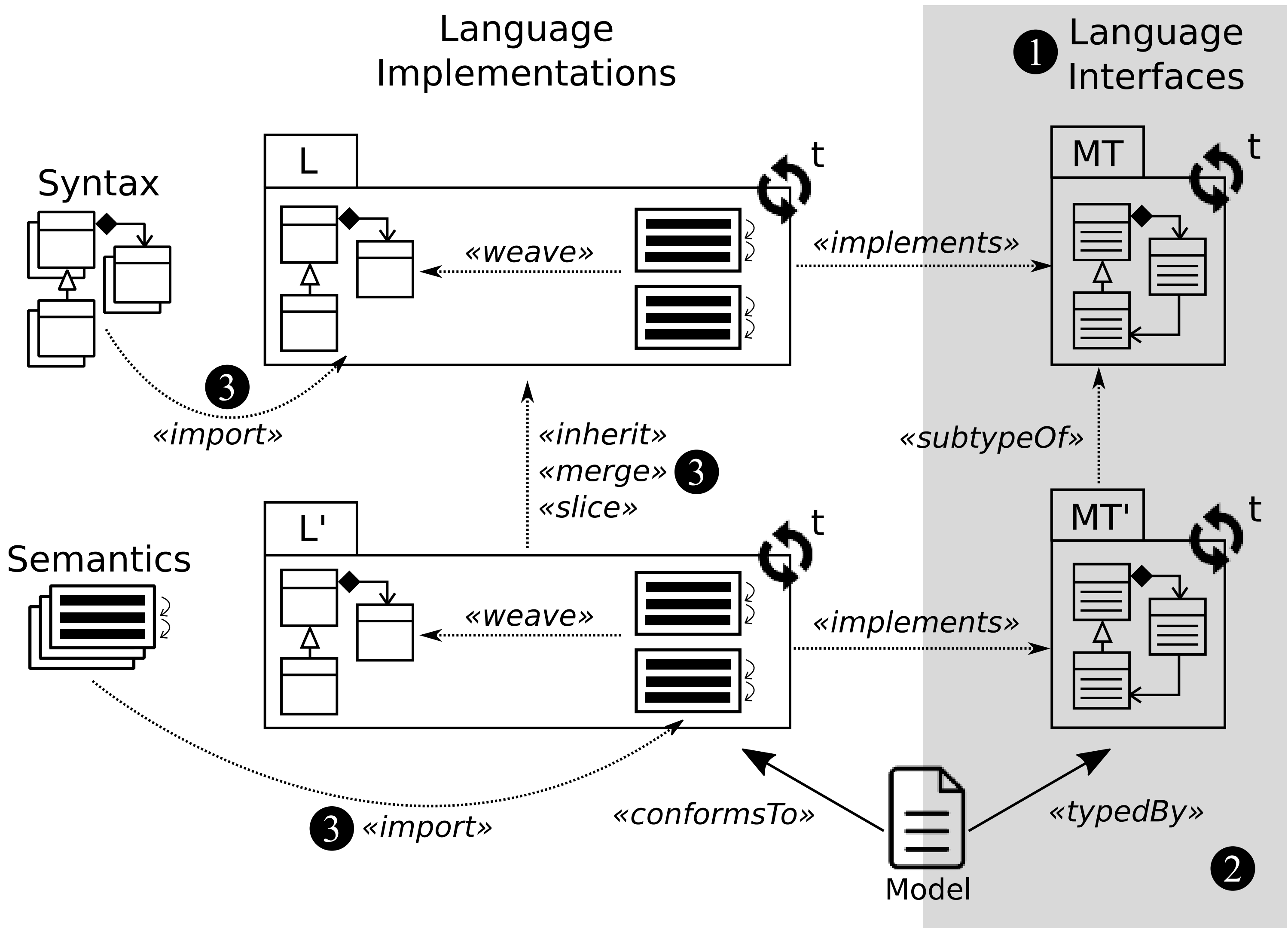
http://melange-lang.org

Melange in Collaborative Projects

- ANR INS GEMOC [GEMOC Studio]
  - Assemble xDSMLs syntaxes and semantics
  - Provide a unified structural interface for tools
  - Examples: TFSM, RobotML, ArduinoML, SigPML, etc.
- LEOC Clarity [Capella Studio]
  - Viewpoints engineering on Capella
  - Current solution: KitAlpha
  - Melange as a lightweight metamodel extension mechanism ensuring type groups consistency and tool reuse

Conclusion & Perspectives

Wrap-up


Language
Implementations
1 Language
Interfaces
Syntax
L
t
«weave»
«implements»
MT
t
3
«import»
«inherit»
«merge» 3
«slice»
«subtypeOf»
Semantics
L'
t
«weave»
«implements»
MT'
t
3 «import»
«conformsTo»
Model
«typedBy»
2

Future Work

- Component-based software *language* engineering

Language Components

Towards Software Language Engineering for the masses

On Language Interfaces
Thomas Degueule, Benoit Combemale and Jean-Marc Jézéquel
In *PAUSE: Present And Ulterior Software Engineering, 2017*

Leveraging Software Product Lines Engineering in the Development of External DSLs: A Systematic Literature Review
David Méndez-Acuña, José A. Galindo, Thomas Degueule,
Benoit Combemale and Benoit Baudry
In *Computer Languages, Systems and Structures* (COMLAN), 2016

Safe Model Polymorphism for Flexible Modeling
Thomas Degueule, Benoit Combemale, Arnaud Blouin, Olivier Barais and Jean-Marc Jézéquel
In *Computer Languages, Systems and Structures* (COMLAN), 2016

Execution Framework of the GEMOC Studio (Tool Demo)
Erwan Bousse, Thomas Degueule, Didier Vojtisek, Tanja Mayerhofer,
Julien Deantoni and Benoit Combemale
In *Proceedings of SLE*, 2016

Interoperability and Composition of DSLs with Melange
Thomas Degueule
ACM Student Research Competition Grand Finals, 2016

Towards an Automation of the Mutation Analysis Dedicated to Model Transformation
Vincent Aranega, Jean-Marie Mottu, Anne Etien, Thomas Degueule,
Benoit Baudry and Jean-Luc Dekeyser
In *Software Testing, Verification and Reliability* (STVR), 2015

Reusing Legacy DSLs with Melange
Thomas Degueule, Benoit Combemale, Arnaud Blouin and Olivier Barais
In *Proceedings of DSM*, 2015

Melange

A Solution to the TTC'15 Model Execution Case Using the GEMOC Studio
Benoit Combemale, Julien DeAntoni, Olivier Barais, Cédric Brun,
Arnaud Blouin, Thomas Degueule, Erwan Bousse and Didier Vojtisek
In *Proceedings of TTC@STAF*, 2015

Melange: A Meta-language for Modular and Reusable Development of DSLs
Thomas Degueule, Benoit Combemale, Arnaud Blouin, Olivier Barais
and Jean-Marc Jézéquel
In *Proceedings of SLE*, 2015

Tooling Support for Variability and Architectural Patterns in Systems Engineering (Tool demo)
Thomas Degueule, João Bosco Ferreira Filho, Olivier Barais et al.
In *Proceedings of SPLC*, 2015

Motivating Use Cases for the Globalization of DSLs
Betty H. C. Cheng, Thomas Degueule, Colin Atkinson, Siobhán Clarke,
Ulrich Frank, Pieter J. Mosterman and Janos Sztipanovits
In *Globalizing Domain-Specific Languages*, 2014

When Systems Engineering Meets Software Language Engineering
Jean-Marc Jézéquel, David Mendez-Acuna, Thomas Degueule,
Benoit Combemale and Olivier Barais
In *Proceedings of CSDM*, 2014

Variability and Patterns in Safety/Security Systems Engineering: An Overview
Thomas Degueule, João Bosco Ferreira Filho, Jérôme Le Noir, Olivier Barais,
Mathieu Acher, Grégory Gailliard, Godefroy Burlot, Olivier Constant et al.
In *Journées Lignes de Produits* (JLDP), 2014

Using Meta-model Coverage to Qualify Test Oracles
Olivier Finot, Jean-Marie Mottu, Gerson Sunyé and Thomas Degueule
In *Proceedings of AMT*, 2013

https://melange-lang.org

EOF