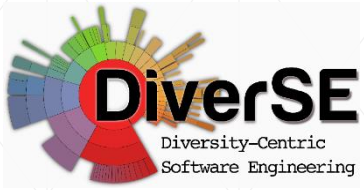
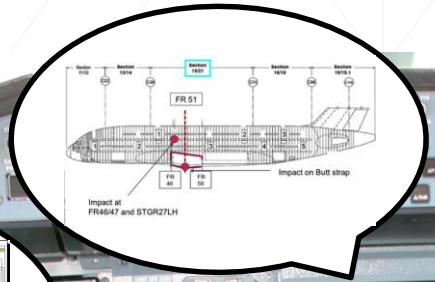


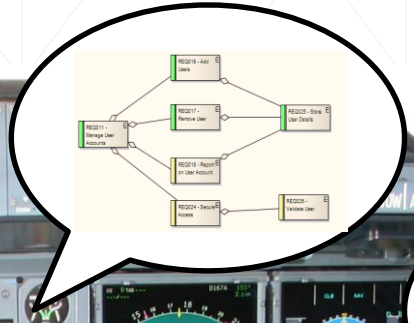
Towards Language Interfaces for DSLs Integration

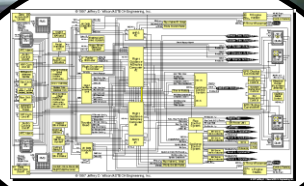
Thomas Degueule – INRIA, France

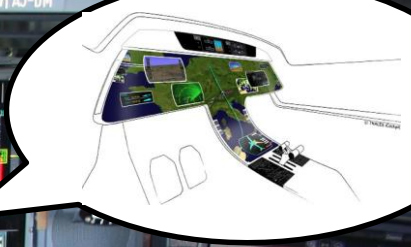

Mechanical Structure
Airlines
Avionics
Human Machine Interaction
Aerodynamics
Multiple Concerns
Environmental Impact
Propulsion System
Safety Regulations
Communications
Navigation
Authorities

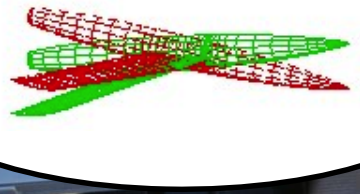
Heterogeneous Modeling

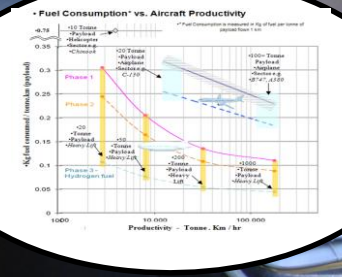


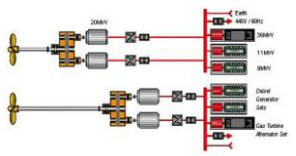


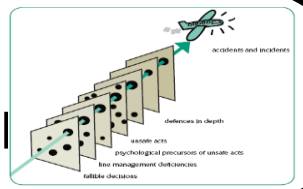


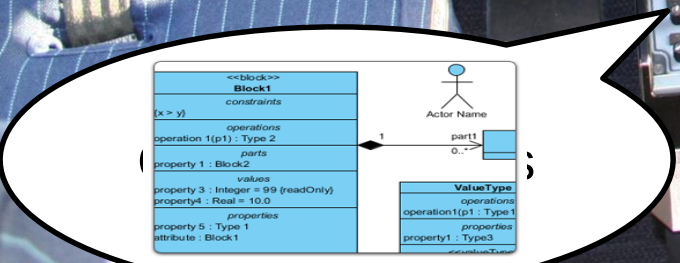


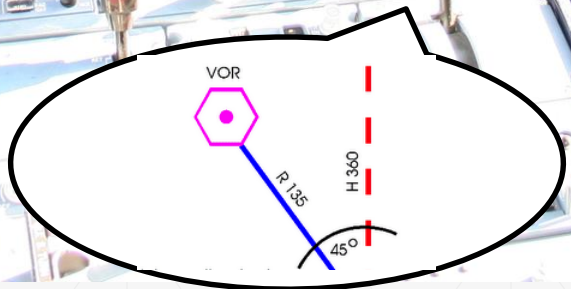






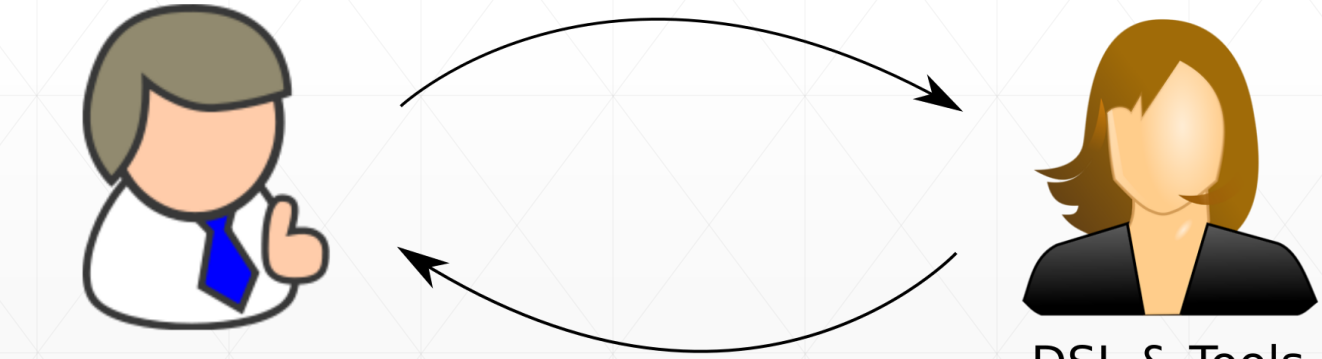




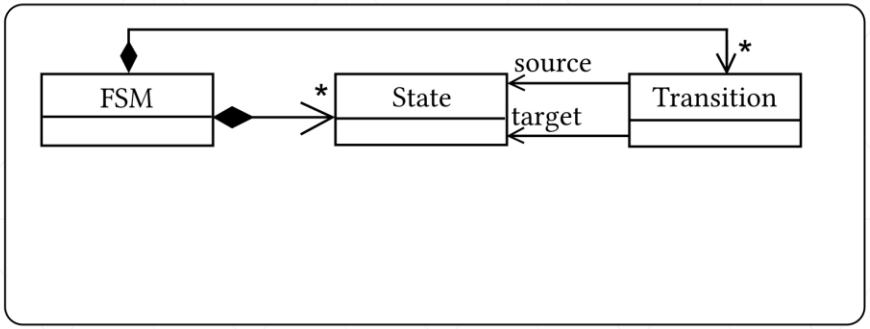


The daily life of DSLs

- Closely evolve with the domain and the experts' understanding of the domain
- Extended, shrunk, customized, replaced with alternatives
- Meant for rapid prototyping, evolution

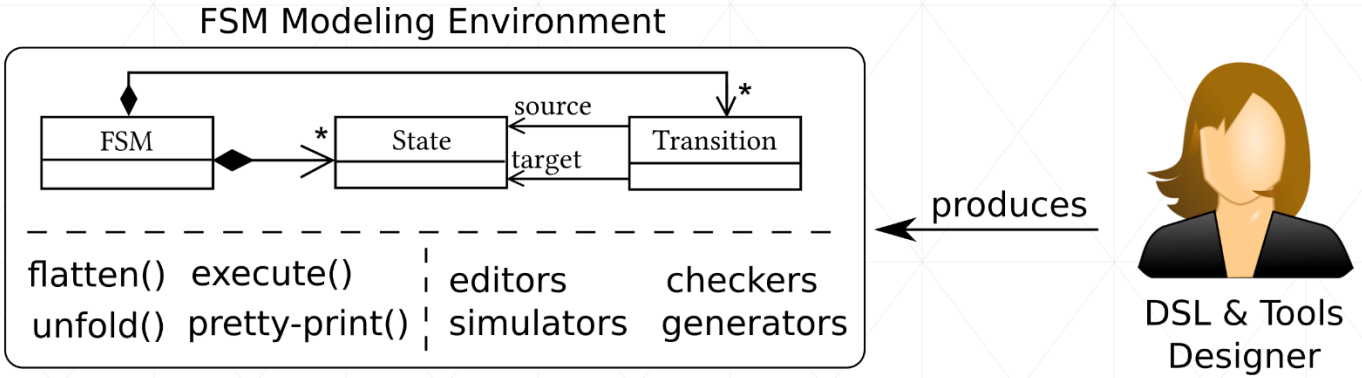


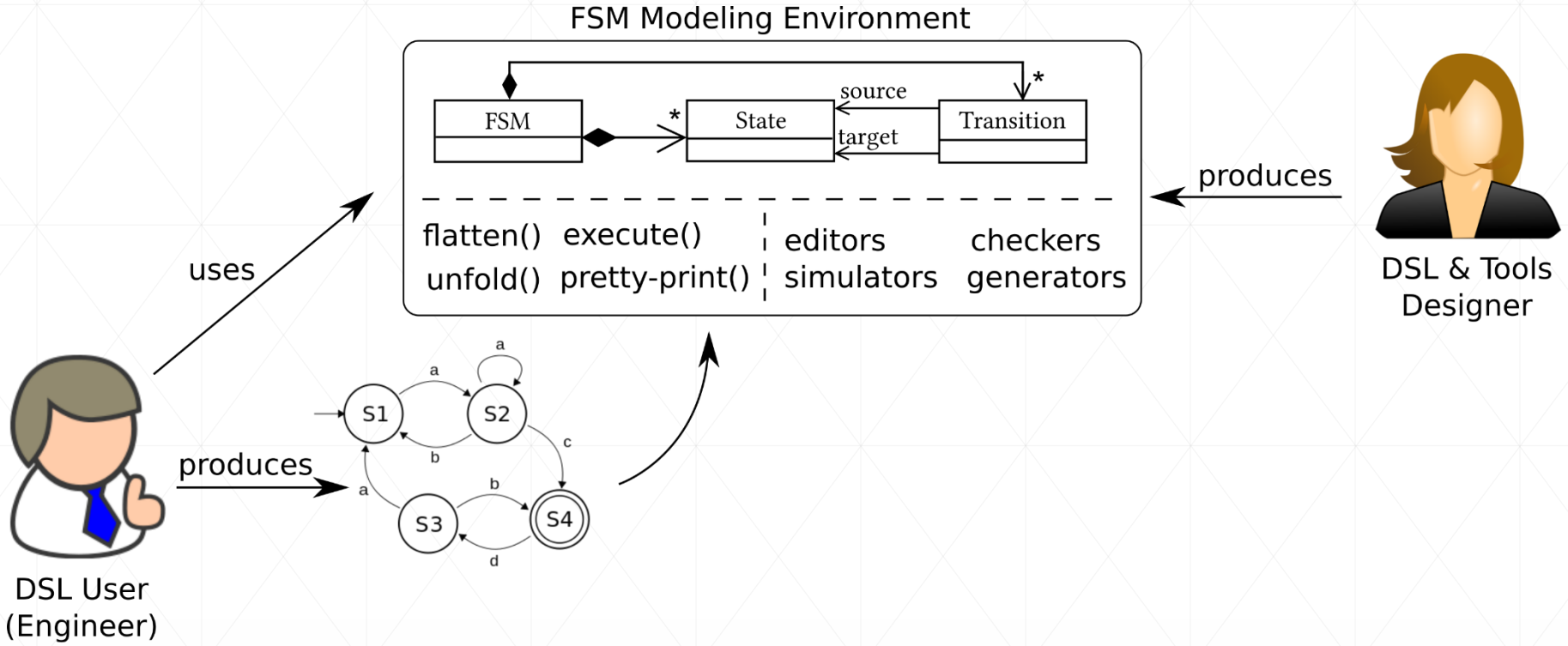
FSM Modeling Environment



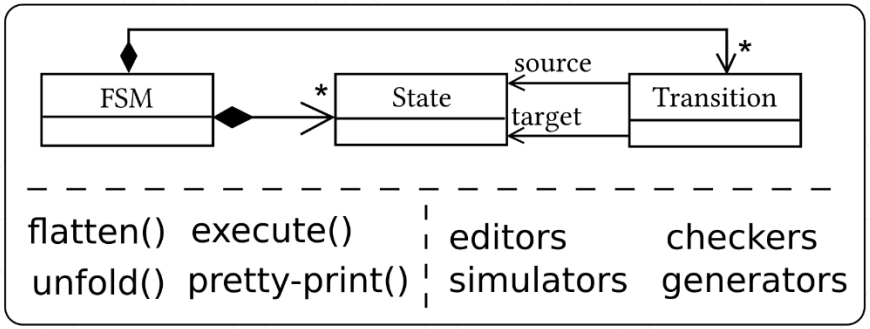
produces

DSL & Tools Designer


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

FSM Modeling Environment



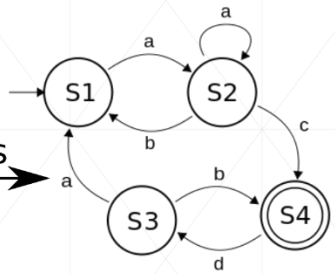
produces

DSL & Tools Designer

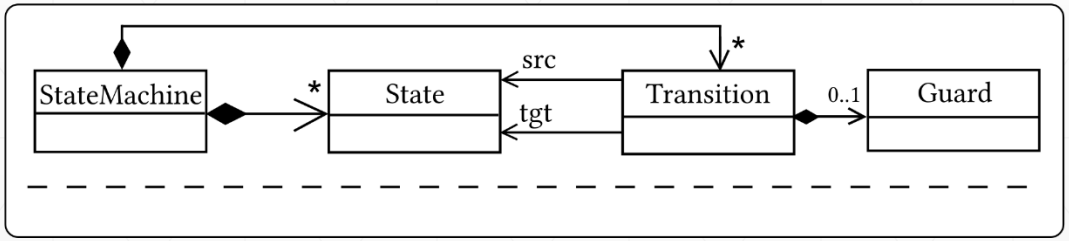
uses

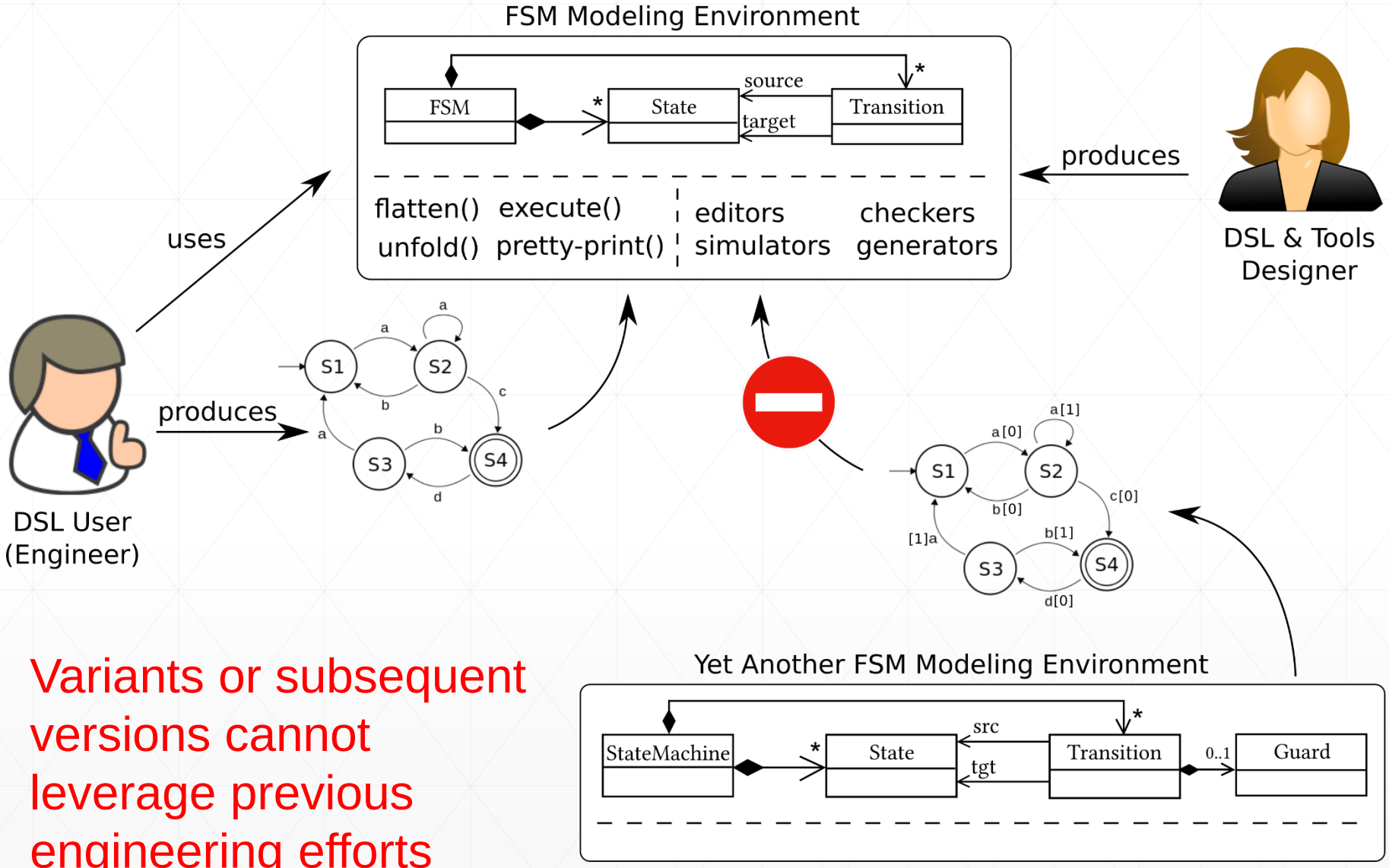
DSL User (Engineer)

produces

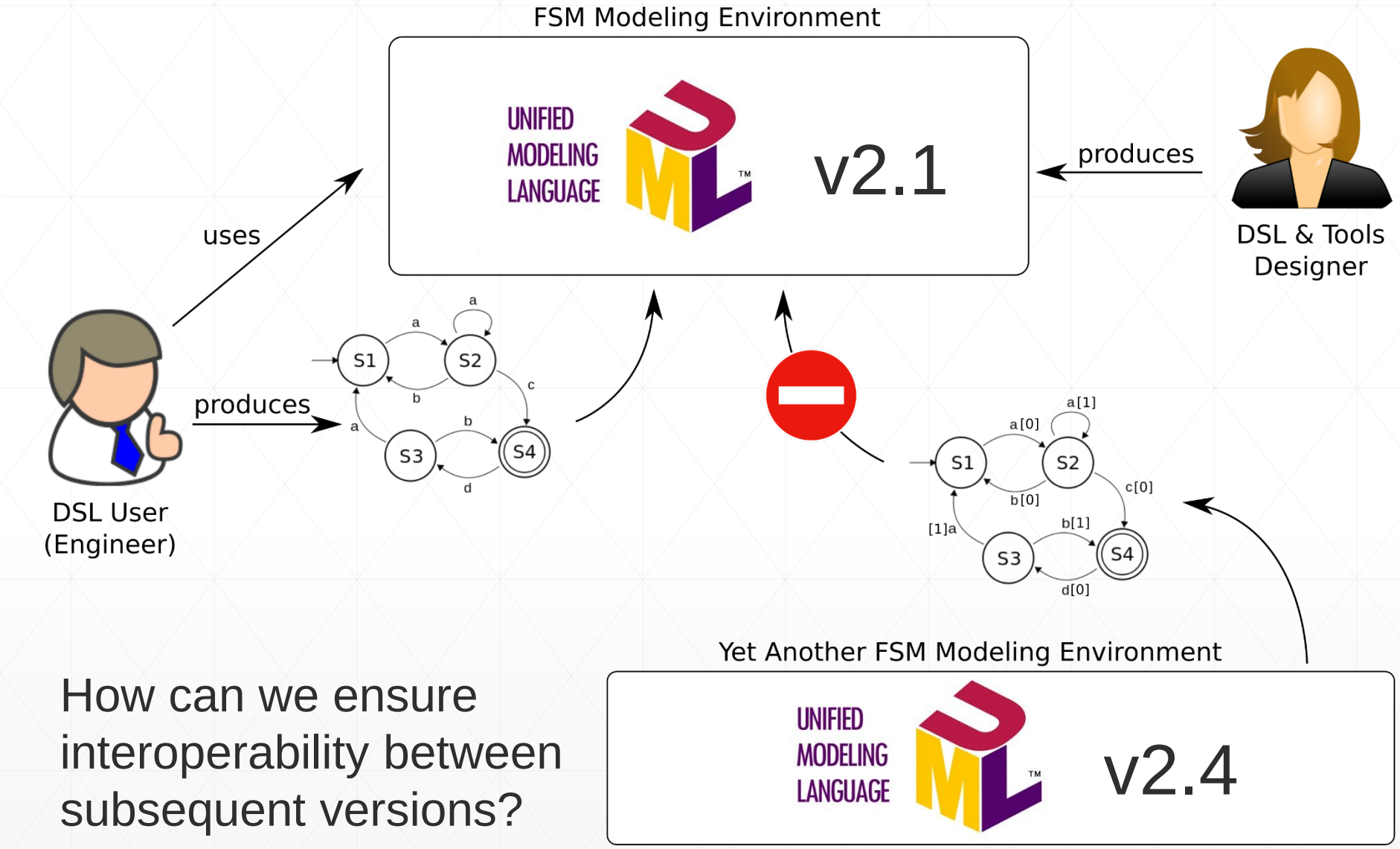


Yet Another FSM Modeling Environment

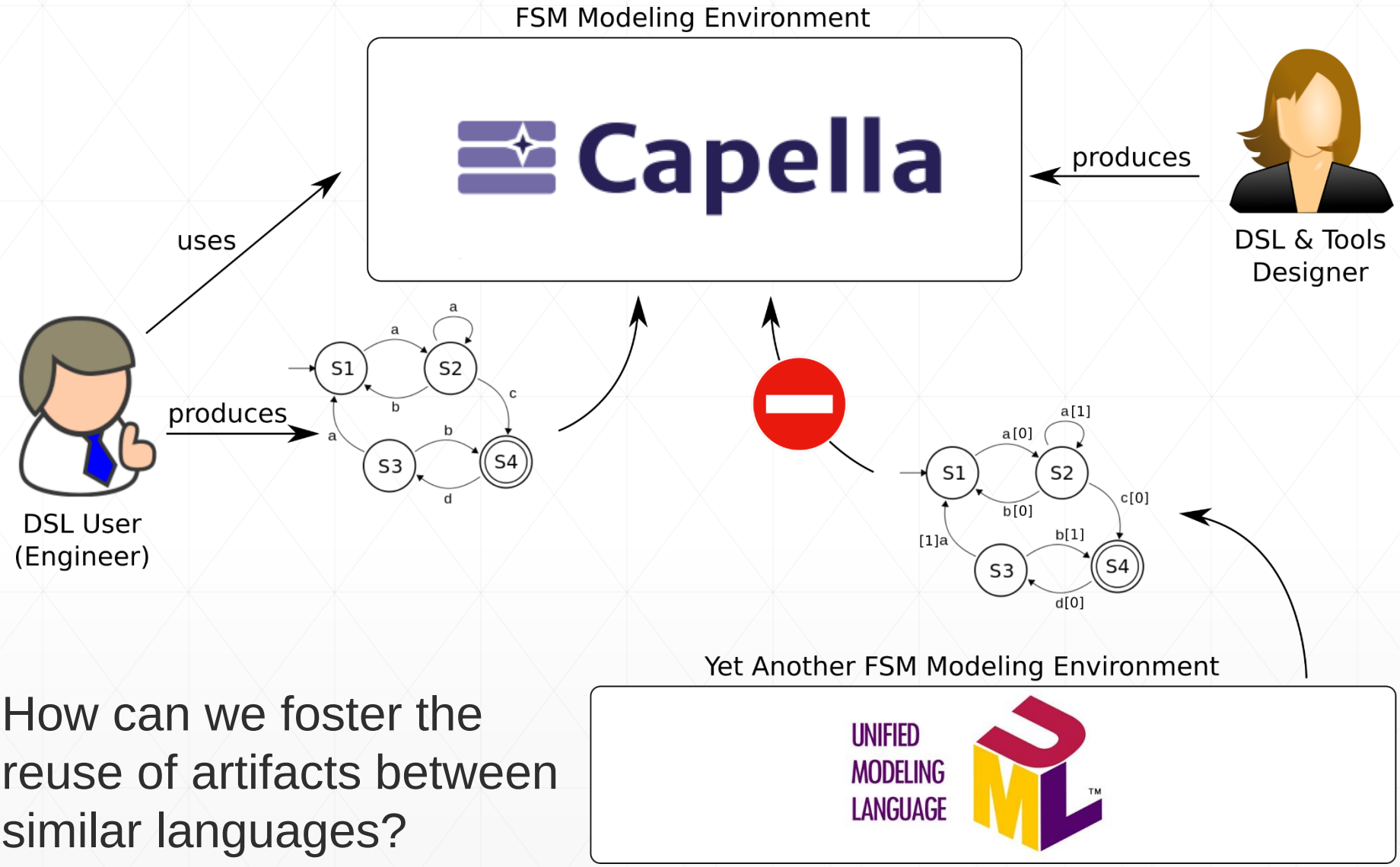




Variants or subsequent versions cannot leverage previous engineering efforts



How can we ensure interoperability between subsequent versions?



How can we foster the reuse of artifacts between similar languages?

Challenges

DSL & Tools Designer

- Manage evolution
- Generic tools & transformations
- Manage syntactical / semantical variation points
- Design families (variants)

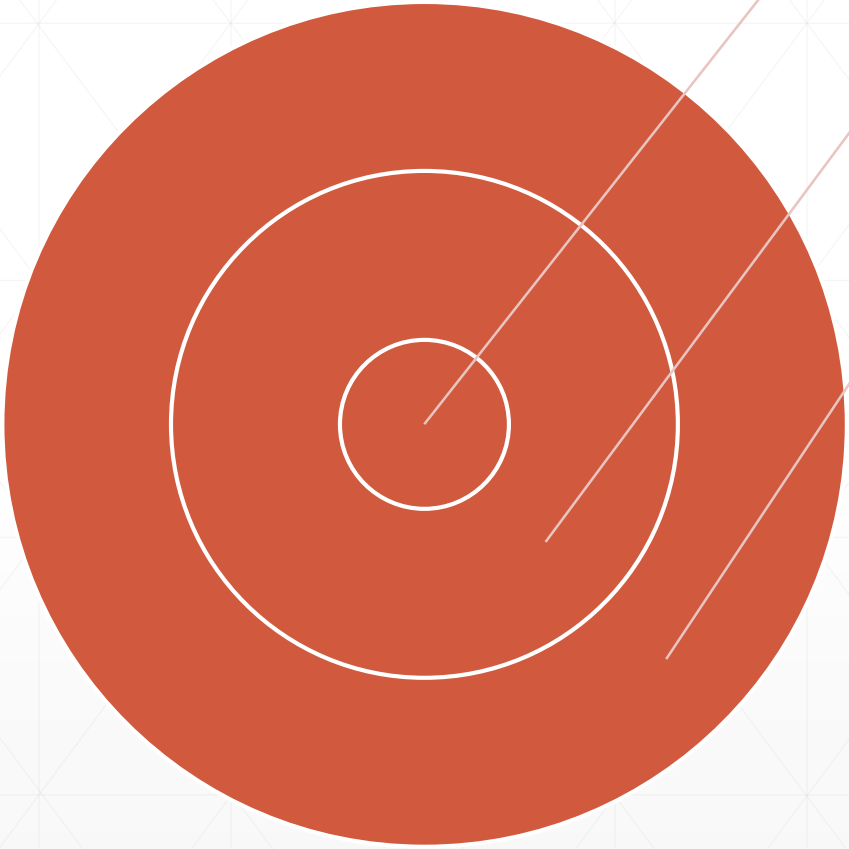
DSL User

- Agile modeling
- Manipulate models in different environments
- Reuse transformations & tools

A unique solution: language interfaces

- Tools, transformations, environments are tightly coupled with the language they were originately defined on
  - → If the language evolves, associated tools break
  - → If a variant exists, tools cannot be reused

- An abstraction layer would reduce the coupling
  - → We realize this abstraction layer with language interfaces



Language Implementation

(abstract syntax, concrete syntaxes, semantics, …)

Language Interface

(meaningful information for a specific purpose)

Language and Model engineering

(transformations, tools, editors, IDEs, …)

→ Multiple DSLs can match the same interface
→ Operators defined on an interface can be reused for all implementing DSLs

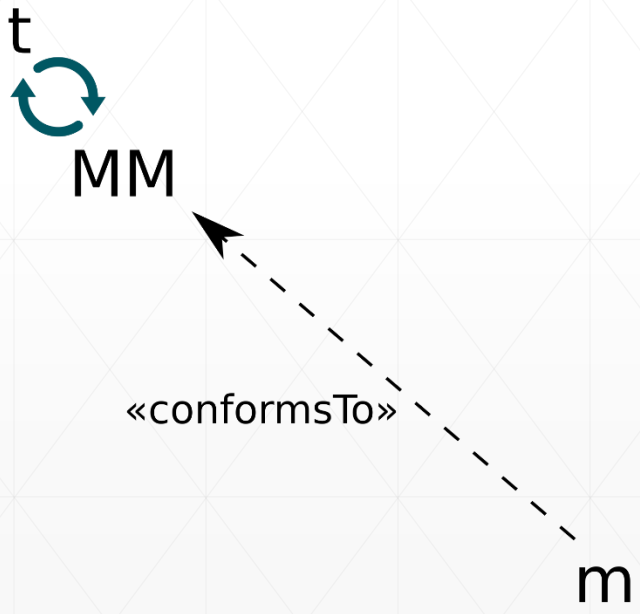
A structural interface: the model type

- Interface over the abstract syntax of a language (a metamodel)
- Focus on the reuse of tools and transformations
- Typing semantics for model manipulation

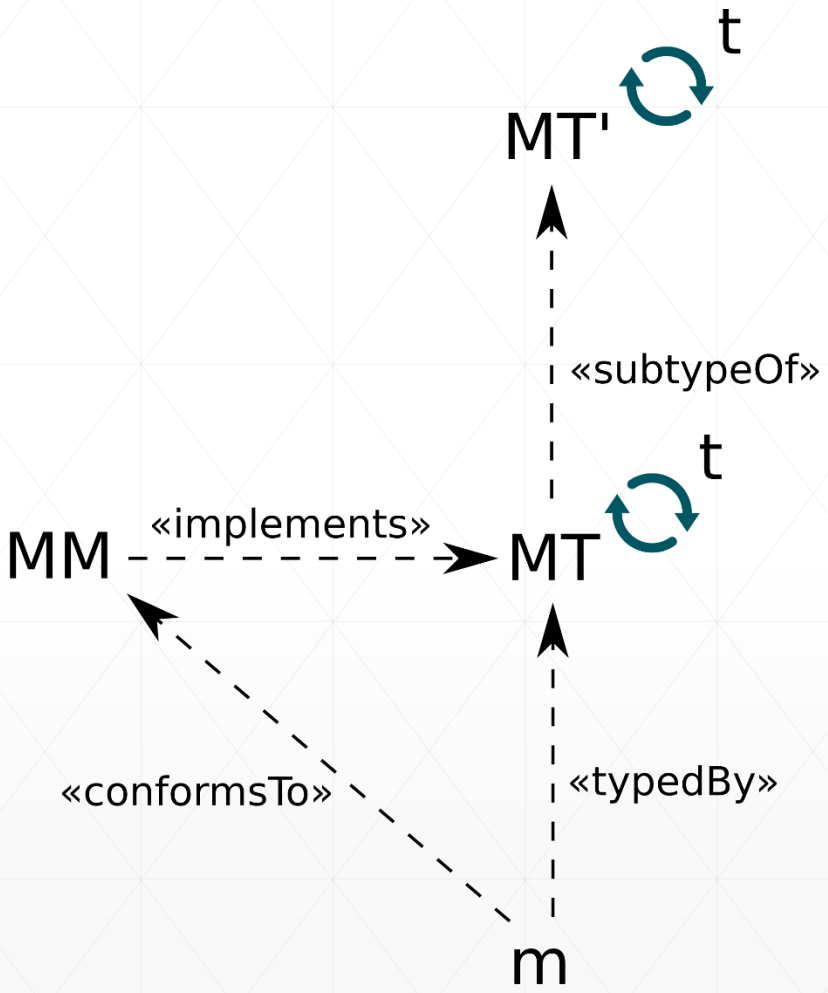
- Supported by a model-oriented type system
  - Models (i.e. graph of objects) as first-class citizens
    - → Type group (family) polymorphism
    - → Structural typing

→ Provides model polymorphism and substitutability

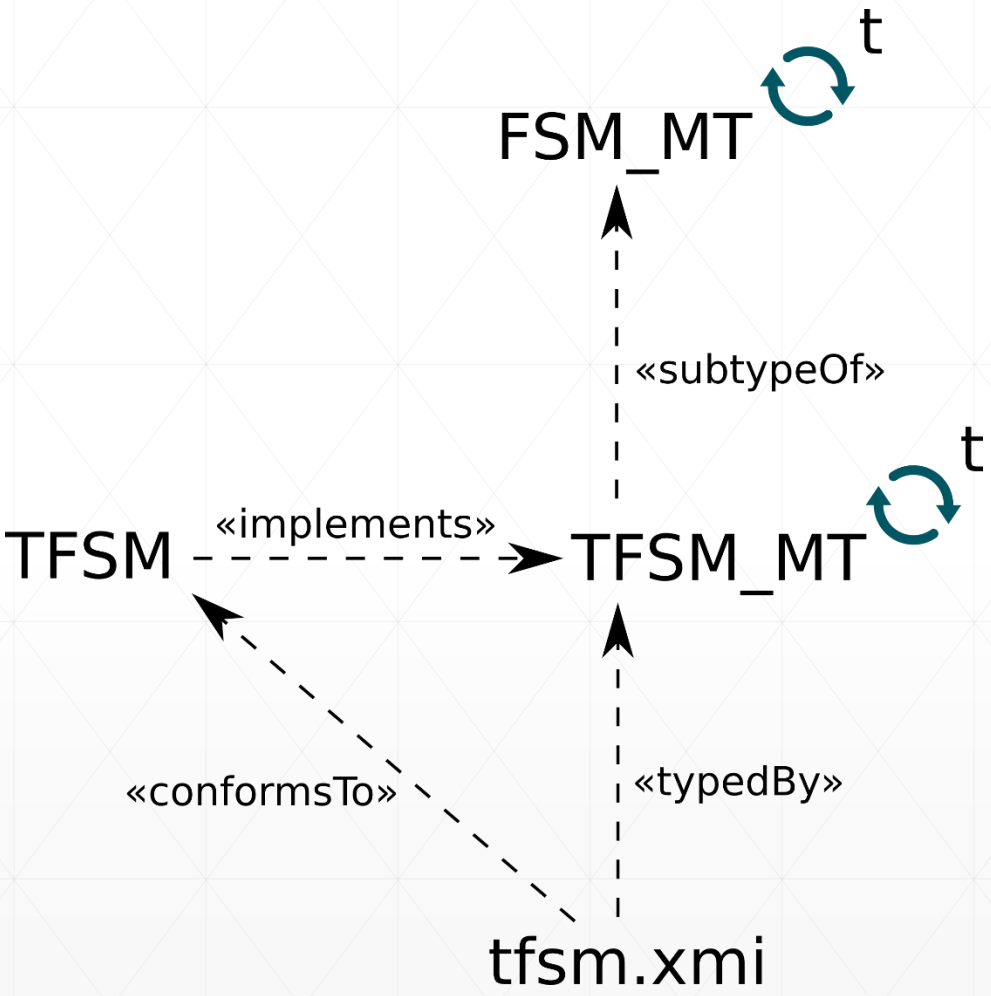
Model typing



Model typing



Model typing



Melange

FsmFamily.melange

```
1  package fsmfamily
2
3  language Fsm {
4      ecore "FSM.ecore"
5      exactType FsmMT
6  }
7
8  language TimedFsm inherits Fsm {
9      exactType TimedFsmMT
10     with timedfsm.TimedTransitionAspect
11 }
12
13 language ExecutableFsm implements FsmMT {
14     ecore "ExecutableFSM.ecore"
15     exactType ExecutableFsmMT
16     with execfsm.ExecutableFSMAspect
17     with execfsm.ExecutableTransitionAspect
18     with execfsm.ExecutableStateAspect
19 }
```

FsmFamily.melange

```
22
23 transformation FsmMT createNewTimedFsm() {
24     val fact = TimedfsmFactory.eINSTANCE
25     return new TimedFsm => [
26         contents += fact.createState => [
27             name = "S1"
28             outgoingTransition += fact.createTransition => [
29                 input = "a"
30             ]
31         ]
32     ]
33 }
34
35 transformation flatten(FsmMT m) { /* ... */ }
36
37 transformation loadFsmModel() {
38     val m1 = Fsm.load("Lights.fsm")
39     val m2 = TimedFsm.load("Temporal.timedfsm")
40     flatten.call(m1)
41     flatten.call(m2)
42     val m3 = m2 as FsmMT
43     flatten.call(m3)
44 }
```

Outline

- fsmfamily
  - Fsm ◁ FsmMT, TimedFsmMT
  - TimedFsm ◀ Fsm ◁ FsmMT, TimedFsmMT
  - ExecutableFsm ◁ FsmMT, TimedFsmMT, Ex
    - ExecutableFSMAspect @ FSM
      - fsm
        - FSM
          - execute
          - currentState
          - State
    - ExecutableTransitionAspect @ Transition
    - ExecutableStateAspect @ State
    - fsm
  - createNewTimedFsm
  - flatten
  - loadFsmModel
  - FsmMT ◁ TimedFsmMT
  - TimedFsmMT ◁ FsmMT

A language-based, model-oriented programming language

Melange: A Language-Based Model-Oriented Programming Language

- A language for defining DSLs
  - Import DSLs implementation
  - Handy operators for SLE: inheritance, merge, etc.
  - Aspect-oriented modeling (e.g. for executable meta-modeling)
  - Generic transformations

DSL & Tools Designer

- A language for manipulating models
  - Models as first-class, typed citizens
  - Model-oriented type system providing model polymorphism
  - Flexible save and load mechanism

DSL User

- Fully interoperable with the EMF ecosystem
- As an external or internal DSL

```
modeltype FsmMT {
    ecore SimpleFSM.ecore
}
```

```
modeltype FsmMT {
    ecore SimpleFSM.ecore
}

language Fsm implements FsmMT {
    ecore FSM.ecore
}
```

```
modeltype FsmMT {
    ecore SimpleFSM.ecore
}

language Fsm implements FsmMT {
    ecore FSM.ecore
}

language ExecFsm {
    ecore FSM.ecore
    with ExecutableSM
    with ExecutableState
    with ExecutableTransition
    exactType ExecFsmMT
}
```

```
modeltype FsmMT {
    ecore SimpleFSM.ecore
}

language Fsm implements FsmMT {
    ecore FSM.ecore
}

language ExecFsm {
    ecore FSM.ecore
    with ExecutableSM
    with ExecutableState
    with ExecutableTransition
    exactType ExecFsmMT
}

language TimedFsm inherits ExecFsm {
    // Variation point
    with TimedTransition
    exactType TimedFsmMT
}
```

```
modeltype FsmMT {
    ecore SimpleFSM.ecore
}

language Fsm implements FsmMT {
    ecore FSM.ecore
}

language ExecFsm {
    ecore FSM.ecore
    with ExecutableSM
    with ExecutableState
    with ExecutableTransition
    exactType ExecFsmMT
}

language TimedFsm inherits ExecFsm {
    // Variation point
    with TimedTransition
    exactType TimedFsmMT
}
```

```
transformation flatten(FsmMT m) {
    m.root.ownedStates.forEach[...]
}
```

```
modeltype FsmMT {
    ecore SimpleFSM.ecore
}

language Fsm implements FsmMT {
    ecore FSM.ecore
}

language ExecFsm {
    ecore FSM.ecore
    with ExecutableSM
    with ExecutableState
    with ExecutableTransition
    exactType ExecFsmMT
}

language TimedFsm inherits ExecFsm {
    // Variation point
    with TimedTransition
    exactType TimedFsmMT
}
```

```
transformation flatten(FsmMT m) {
    m.root.ownedStates.forEach[...]
}

transformation execute(ExecFsmMT m){
    // With dynamic binding
    m.root.execute(«word»)
}
```

```
modeltype FsmMT {
    ecore SimpleFSM.ecore
}

language Fsm implements FsmMT {
    ecore FSM.ecore
}

language ExecFsm {
    ecore FSM.ecore
    with ExecutableSM
    with ExecutableState
    with ExecutableTransition
    exactType ExecFsmMT
}

language TimedFsm inherits ExecFsm {
    // Variation point
    with TimedTransition
    exactType TimedFsmMT
}
```

```
transformation flatten(FsmMT m) {
    m.root.ownedStates.forEach[...]
}

transformation execute(ExecFsmMT m){
    // With dynamic binding
    m.root.execute(«word»)
}

main() {
    val m1 = new Fsm
    val m2 = ExecFsm.load(«Foo.fsm»)
    val m3 = TimedFsm.load(«Foo.tfsm»)

    val m4 = m3 as FsmMT // Viewpoints

    flatten(m1)
    flatten(m2)
    flatten(m3)
    execute(m2)
    execute(m3)
    ~~execute(m1)~~ // Statically forbidden
}
```

Ongoing Experiments

- Families of syntactically and semantically diverse languages
  - Example: FSM
    - Syntaxes: Simple – hierarchical – with time constraints – etc.
    - Semantics: Run-to-completion – concurrent – etc.
    - Generic transformations: flatten – execute – etc.
- Thales' Capella language
  - xCapella: executable extension of Capella
  - Managing the interoperability with UML
- Executable metamodeling within the ANR GEMOC project

Future Work

- Generic meta-programming
- Viewpoints engineering
- Model types as explicit required/provided interfaces of languages units
- Behavioral interfaces (e.g. event structure) for coordinated execution of heterogeneous languages

Wrap-up

DSL & Tools Designer

- DSL engineering
  - High-level operators
    - Inheritance, merge, etc.
    - Aspect-oriented modeling
  - Executable meta-modeling
  - Generic tools definition

DSL User

- Agile modeling
  - Manipulate models in different environments
  - Viewpoints
  - Reuse of tools

Acknowledgments

- Dr. Olivier Barais, University of Rennes, France
- Dr. Arnaud Blouin, INSA Rennes, France
- Dr. Benoit Combemale, INRIA, France
- Prof. Jean-Marc Jezequel, University of Rennes, France
- Prof. Robert France, CSU, USA

http://melange-lang.org
https://github.com/diverse-project/melange