

Apprivoiser et synthétiser la diversité des langages logiciels

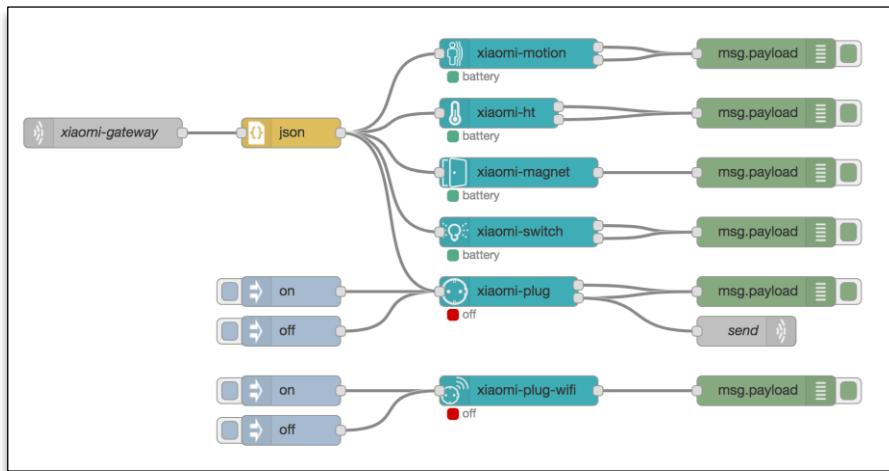
Thomas Degueule

Software Analysis and Transformation Group

Centrum Wiskunde & Informatica

<https://tdegueul.github.io>

Software languages



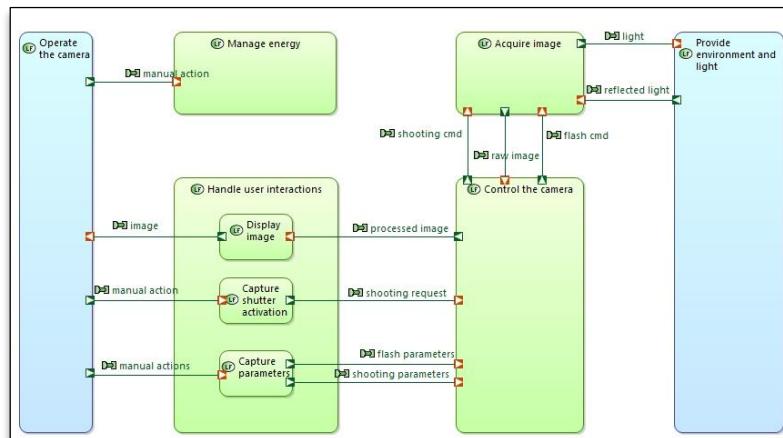
Node-RED
Internet of Things Language

```
library IEEE;
use IEEE.std_logic_1164.all;

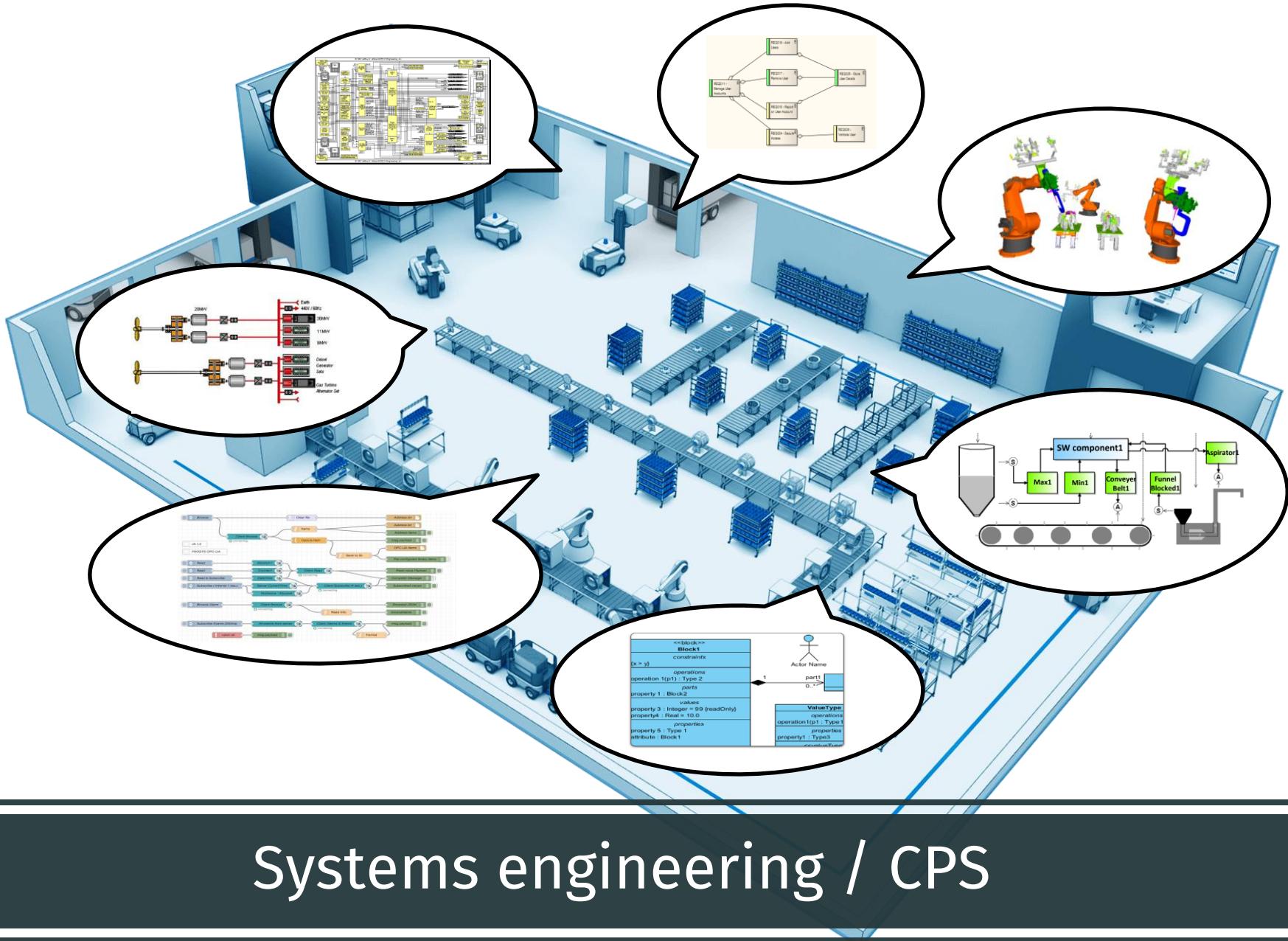
-- this is the entity
entity ANDGATE is
port (
    I1 : in std_logic;
    I2 : in std_logic;
    O : out std_logic);
end entity ANDGATE;

-- this is the architecture
architecture RTL of ANDGATE is
begin
    O <= I1 and I2;
end architecture RTL;
```

VHDL
Hardware Description Language



Capella
Systems Engineering Language



Systems engineering / CPS

Model Explorer Logical Steps

cultures.activities

```

culture corn {
    activity LABOUR from 1 jan to 28 feb
    using 1 Tractor and 1 People

    activity SEMIS from 15 mar to 15 apr [
        after LABOUR & no rain since 3 days & temp > 10°C
    ] using 1 Tractor and 2 People

    activity IRRIGATION weekly from 15 jun to 15 aug
    after SEMIS
    using 1 Tractor and 1 People

    activity FERTILISATION from 15 mar to 15 jun [
        after SEMIS is done since 30 days &&
        no rain since 1 days
    ] using 1 Tractor and 1 People

    activity RECOLTE from 1 sept to 30 sept [
        grain is "mature" &&
        after SEMIS
    ] using 1 Tractor and 2 People
}

```

***exploitation description**

***Schedule**

- NW of Exploitation 4 fields
 - corn LABOUR scheduled on 13/jan
 - corn SEMIS scheduled on 31/mar
 - corn IRRIGATION scheduled on 4/aug
 - corn FERTILISATION scheduled on 5/may
 - corn RECOLTE scheduled on 1/sept
- 2 fields
 - corn LABOUR scheduled on 1/jan
 - corn SEMIS scheduled on 15/mar
 - corn IRRIGATION scheduled on 15/jun
 - corn FERTILISATION scheduled on 27/may
 - corn RECOLTE scheduled on 21/sept

***Hydro Analysis**

	Extra Water	Rain	Hyd	Biomass	LAI
John's Exp...	0.0	0.0	57.0		
Surface...	0.0	0.0			
31 mar	0.0	0.0			
1 apr	0.0	0.0			
2 apr	0.0	0.0			
3 apr	0.0	0.0			
4 apr	0.0	0.0			
5 apr	0.0	0.0			
6 apr	0.0	0.0			
7 apr	0.0	0.0			
8 apr	0.0	0.0	11.5	0.04652190...	0.000
9 apr	0.0	0.0	11.5	0.04548258...	0.000
10 apr	0.0	0.0	11.5	0.04743018...	0.000
11 apr	0.0	0.5	0.0	0.05144848...	0.000
12 apr	0.0	2.5	-0.5	0.05425001...	0.000
13 apr	0.0	0.5	-3.0	0.05683363...	0.000
14 apr	0.0	0.0	10.4	0.05862190...	0.000
15 apr	0.0	0.0	11.0	0.05862190...	0.000
16 apr	0.0	0.0	11.4	0.05862190...	0.000
17 apr	0.0	0.5	9.9	0.05862190...	0.000
18 apr	0.0	2.5	10.7	0.05862190...	0.000
19 apr	0.0	0.5	9.7	0.05862190...	0.000
20 apr	0.0	0.0	815.0	0.05862190...	0.000

<https://github.com/gemoc/farmingmodeling>

Scientific modeling and simulation



Programming education

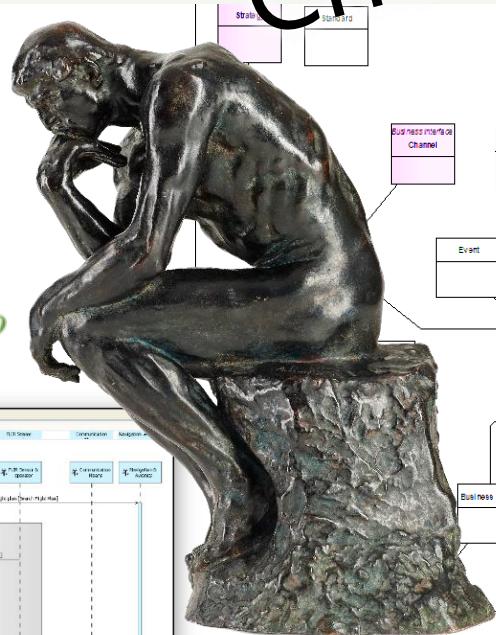
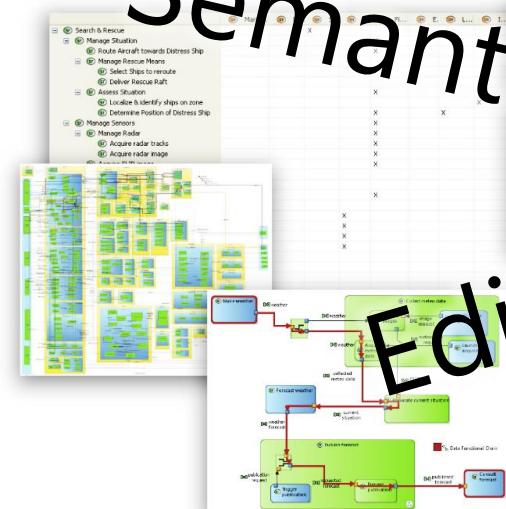
Credits: Christopher R Blais (@blaistech)

Software Language Engineering (SLE)

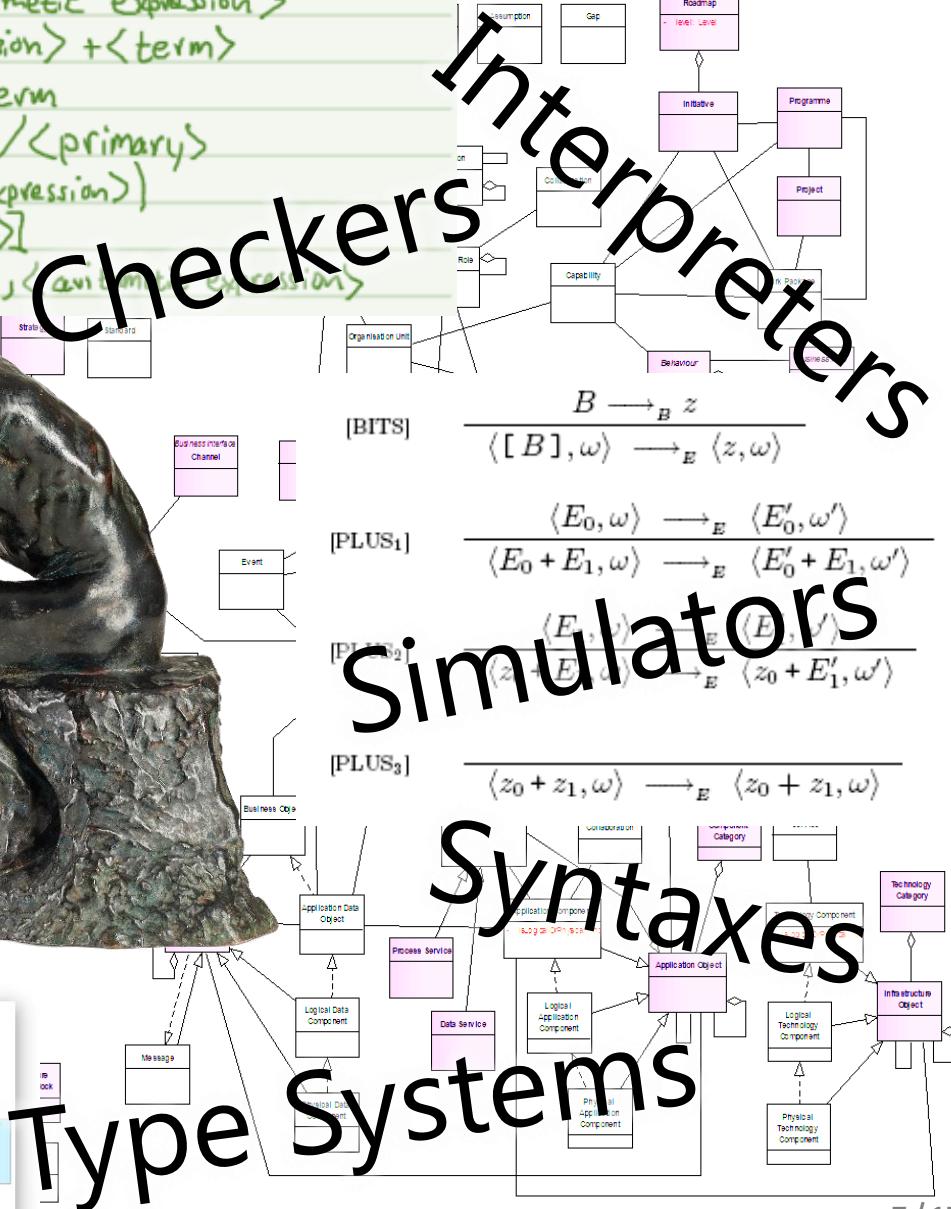
$\langle \text{assignment statement} \rangle ::= \langle \text{variable} \rangle = \langle \text{arithmetic expression} \rangle$
 $\langle \text{arithmetic expression} \rangle ::= \langle \text{term} \rangle \mid \langle \text{arithmetic expression} \rangle + \langle \text{term} \rangle$
 $\quad \quad \quad \mid \langle \text{arithmetic expression} \rangle - \langle \text{term} \rangle$
 $\langle \text{term} \rangle ::= \langle \text{primary} \rangle \mid \langle \text{term} \rangle \cdot \langle \text{primary} \rangle \mid \langle \text{primary} \rangle / \langle \text{primary} \rangle$
 $\langle \text{primary} \rangle ::= \langle \text{variable} \rangle \mid \langle \text{number} \rangle \mid \langle \text{arithmetic expression} \rangle$
 $\langle \text{variable} \rangle ::= \langle \text{identifier} \rangle \mid \langle \text{identifier} \rangle [\langle \text{subscript list} \rangle]$
 $\langle \text{subscript list} \rangle ::= \langle \text{arithmetic expression} \rangle \mid \langle \text{subscript list} \rangle , \langle \text{arithmetic expression} \rangle$

$\frac{}{\Gamma, x : \tau_p \vdash t : \tau_r}$
 $\frac{\Gamma, x : \tau_p \vdash t : \tau_r}{(\lambda x : \tau_p. t) : \tau_p \rightarrow \tau_r} \text{ Lam}$

$\frac{\Gamma \vdash t_f : \tau_p \rightarrow \tau_r \quad \Gamma \vdash t_p : \tau_p}{\Gamma \vdash t_f t_p : \tau_r} \text{ App}$



Editors



Type Systems

Research Activity

- Scientific contributions

- Modularity, reuse, composition
- Flexible modeling, live modeling
- Empirical software engineering

(e.g. [SLE'15], [COMLAN'16], [MODELS'17])

(e.g. [COMLAN'17], [SLE'18₁], [SLE'18₂])

(e.g. [MSR'18], [ICSE'19])

- Technological contributions



<http://melange.inria.fr>



<http://www.kermeta.org>



<https://eclipse.org/gemoc>



<http://eclipse.org/scava/>

- Industrial collaborations



- Experimental validation

- Programming languages
- Systems engineering
- Internet of Things
- Open-source software

Modular Language Extension

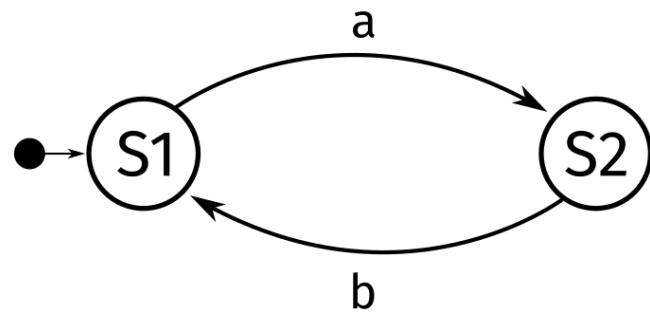
Revisiting Visitors for Modular Extension of Executable DSMLs

Manuel Leduc, Thomas Degueule, Benoit Combemale, Tijs van der Storm, Olivier Barais

20th International Conference on Model Driven Engineering Languages and Systems (MODELS'17)

Modular language extension

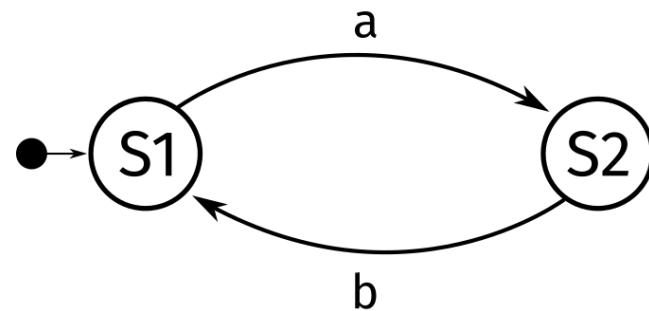
FSM



Syntax

Modular language extension

FSM



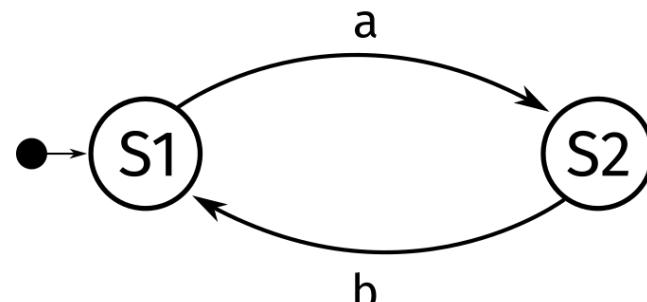
Syntax

```
eval(fsm, 'ab'):  
    s1 -> s2  
    s2 -> s1
```

Interpreter

Modular language extension

FSM

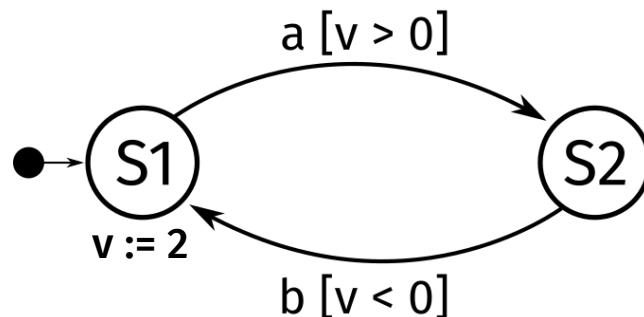


Syntax

```
eval(fsm, 'ab'):  
  s1 -> s2  
  s2 -> s1
```

Interpreter

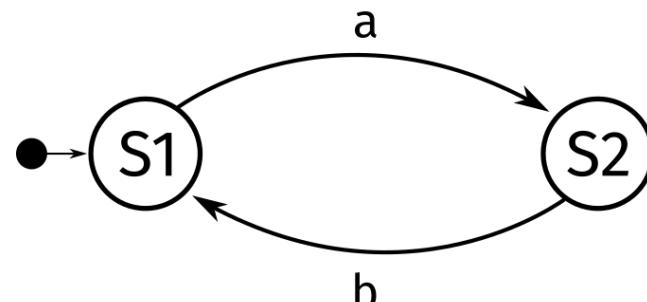
GuardedFSM



Syntax'

Modular language extension

FSM

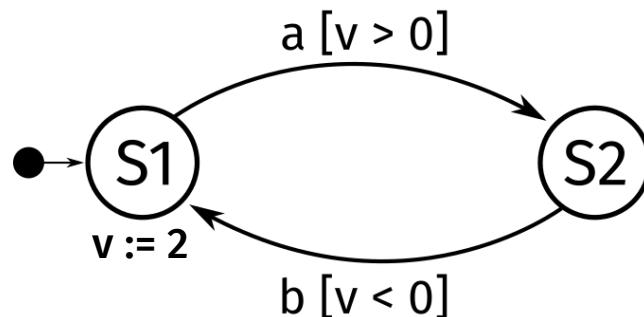


Syntax

```
eval(fsm, 'ab'):  
  s1 -> s2  
  s2 -> s1
```

Interpreter

GuardedFSM



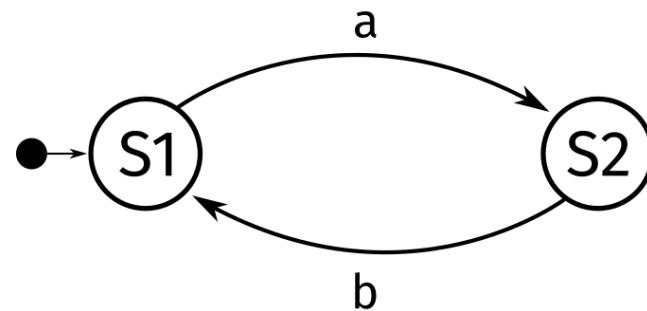
Syntax'

```
eval(gfsm, 'ab'):  
  s1 -> s2  
  s2 -> s1
```

Interpreter'

Modular language extension

FSM

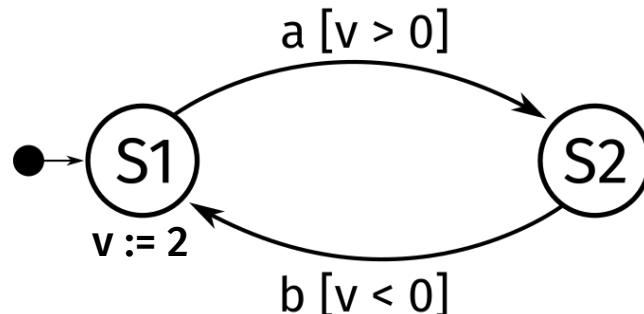


Syntax

```
eval(fsm, 'ab'):  
  s1 -> s2  
  s2 -> s1
```

Interpreter

GuardedFSM



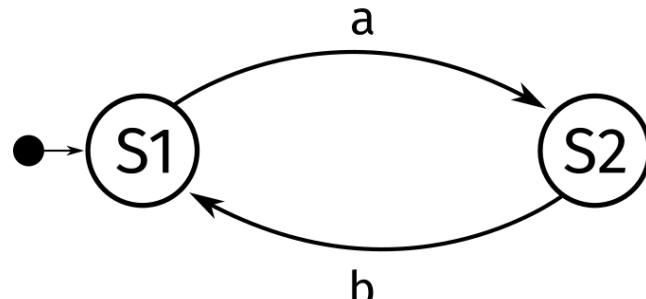
Syntax'

```
eval(gfsm, 'ab'):  
  s1 -> s2  
<deadlock>
```

Interpreter'

Modular language extension

FSM



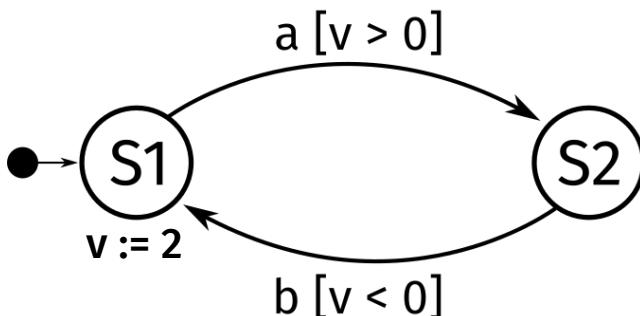
Syntax

```
eval(fsm, 'ab'): s1 -> s2  
s2 -> s1
```

Interpreter

```
pretty-print(fsm):  
  machine  
  initial S1  
  a to S2  
  state S2  
  b to S1  
Pretty-printer
```

GuardedFSM



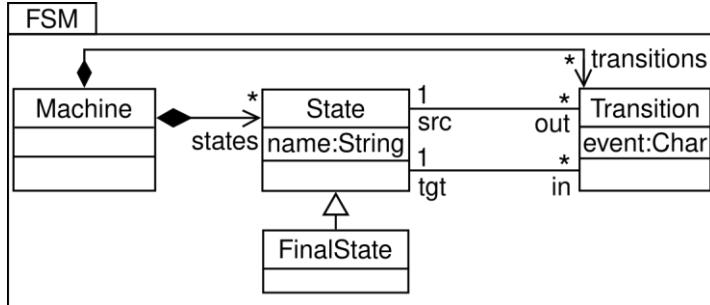
Syntax'

```
eval(gfsm, 'ab'): s1 -> s2  
<deadlock>
```

Interpreter'

```
pretty-print(gfsm):  
  machine  
  initial S1  
  a to S2 if v > 0  
  state S2  
  b to S1 if v < 0  
Pretty-printer'
```

Modular language extension

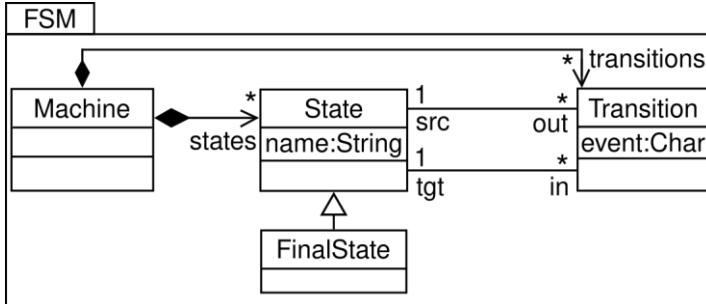


Abstract Syntax

```
interpret(State s, String evt) {  
    // Find a transition for the input event  
    State next = s.out.findFirst[event == evt]  
    // Fire it  
    next.fire()  
}
```

Execution Semantics

Modular language extension



Abstract Syntax

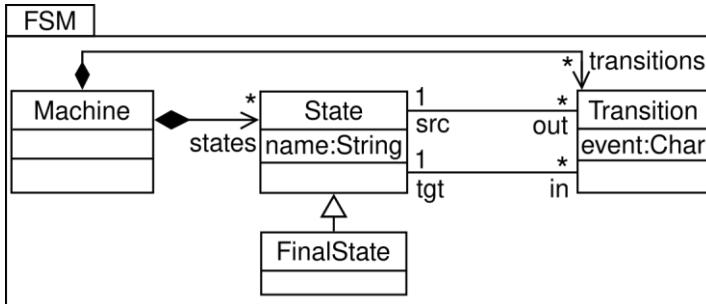
```
class Machine { List<State> states; [...] }
class State   { String name; [...] }
class Trans   { char event; [...] }
class FS extends State { [...] }
```

Abstract Syntax Classes

```
interpret(State s, String evt) {
    // Find a transition for the input event
    State next = s.out.findFirst[event == evt]
    // Fire it
    next.fire()
}
```

Execution Semantics

Modular language extension



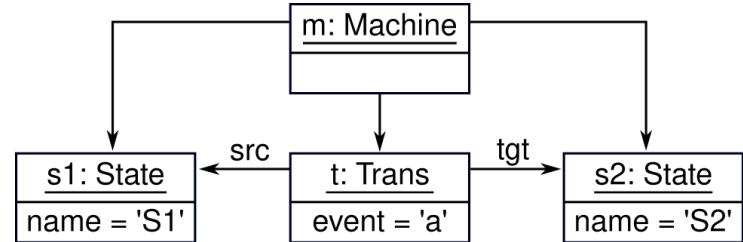
Abstract Syntax

```
class Machine { List<State> states; [...] }
class State { String name; [...] }
class Trans { char event; [...] }
class FS extends State { [...] }
```

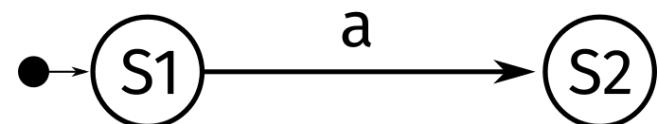
Abstract Syntax Classes

```
interpret(State s, String evt) {
    // Find a transition for the input event
    State next = s.out.findFirst[event == evt]
    // Fire it
    next.fire()
}
```

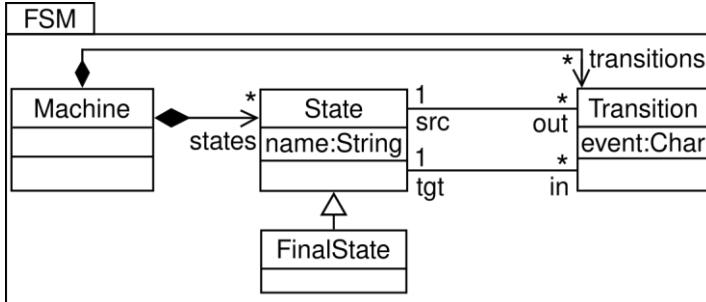
Execution Semantics



Abstract Syntax Graph



Modular language extension



Abstract Syntax

```
interpret(State s, String evt) {
    // Find a transition for the input event
    State next = s.out.findFirst[event == evt]
    // Fire it
    next.fire()
}
```

Execution Semantics

```
class Machine { List<State> states; [...] }
class State { String name; [...] }
class Trans { char event; [...] }
class FS extends State { [...] }
```

Abstract Syntax Classes

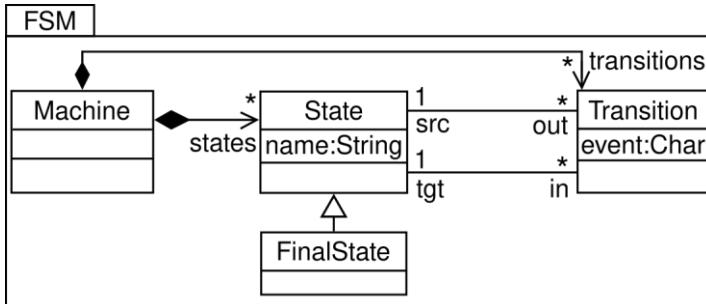
```
interface Interpret { void interpret(); }
class Machine implements Interpret { [...] }
class State implements Interpret { [...] }
class Trans implements Interpret { [...] }
```

Interpreter Pattern

```
class Machine { void accept(Visitor v); }
class State { void accept(Visitor v); }
class Trans { void accept(Visitor v); }
interface Visitor {
    void visit(Machine m);
    void visit(State s);
    void visit(Transition t);
}
class ExecMachine implements Visitor {
    void visit(Machine m) { [...] }
    void visit(State s) { [...] }
    void visit(Transition t) { [...] }
}
```

Visitor Pattern

Modular language extension



Abstract Syntax

```
interpret(State s, String evt) {
    // Find a transition for the input event
    State next = s.out.findFirst[event == evt]
    // Fire it
    next.fire()
}
```

Execution Semantics

```
pretty-print(State s) {
    print("State " + s.name)
    print("Transitions:")
    for (t in s.transitions)
        pretty-print(t)
}
```

Printing Semantics

```
class Machine { List<State> states; [...] }
class State { String name; [...] }
class Trans { char event; [...] }
class FS extends State { [...] }
```

Abstract Syntax Classes

```
interface Interpret { void interpret(); }
interface Print { void print(); }
class Machine implements Interpret { [...] }
class State implements Print { [...] }
class Trans implements Interpret { [...] }
```

Interpreter Pattern

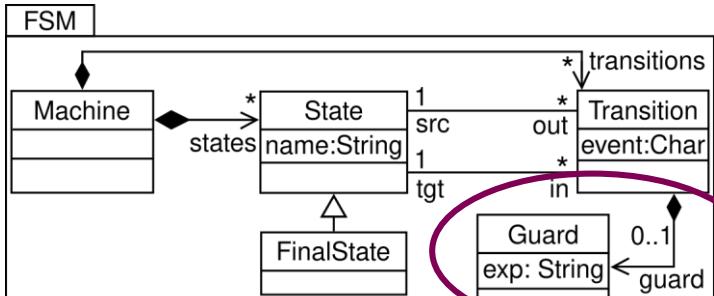
```
interface Visitor { [...] }
class ExecMachine implements Visitor {
    void visit(Machine m) { [...] }
    void visit(State s) { [...] }
    void visit(Transition t) { [...] }
}
```

```
class PrintMachine implements Visitor {
    void visit(Machine m) { [...] }
    void visit(State s) { [...] }
    void visit(Transition t) { [...] }
}
```

Visitor Pattern



Modular language extension



Abstract Syntax

```
class Machine { List<State> states; [...] }
class State { String name; [...] }
class Trans { char event; Guard guard; }
class FS extends State { [...] }
class Guard { String exp; }
```

Abstract Syntax Classes

```
interpret(State s, String evt) {
    // Find a transition for the input event
    State next = s.out.findFirst[event == evt]
    // Fire it
    next.fire()
}
```

Execution Semantics



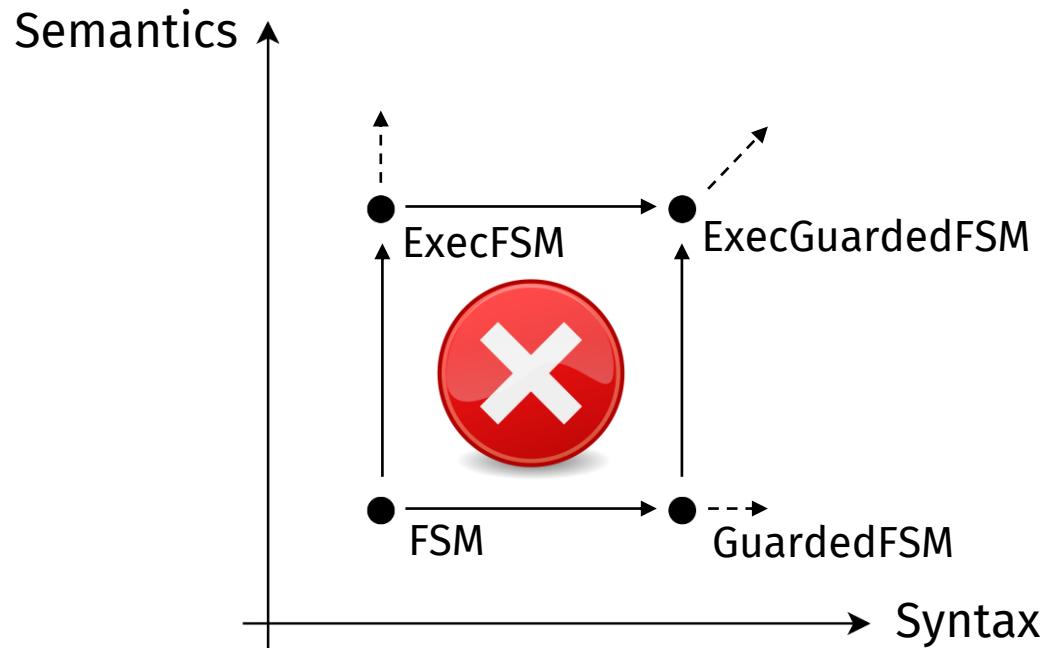
```
interface Interpret { void interpret(); }
class Machine implements Interpret { [...] }
class State implements Interpret { [...] }
class Trans implements Interpret { [...] }
class Guard implements Interpret { [...] }
```

Interpreter Pattern

```
interface Visitor {
    void visit(Machine m);
    void visit(State s);
    void visit(Transition t);
    void visit(Guard g); ✗
}
class ExecMachine implements Visitor {
    void visit(Machine m) { [...] }
    void visit(State s) { [...] }
    void visit(Transition t) { [...] }
    void visit(Guard g) { [...] } ✗
}
```

Visitor Pattern

The *Expression Problem*

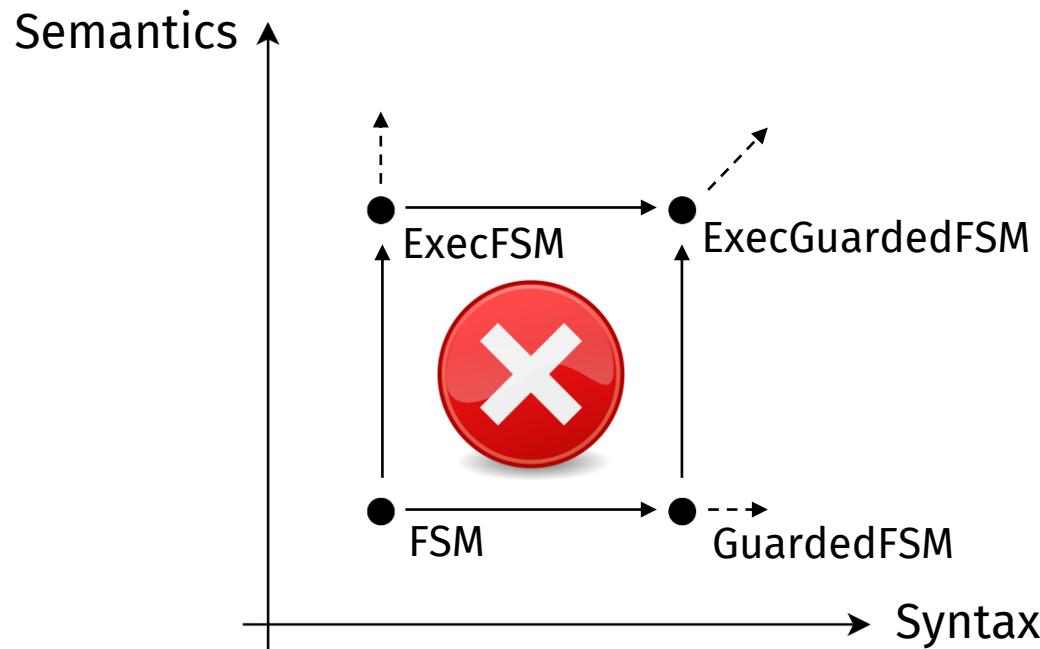


"The expression problem is a new name for an old problem. The goal is to define a datatype by cases, where one can add new cases to the datatype and new functions over the datatype, without recompiling existing code, and while retaining static type safety."

Philip Wadler, 1998

- Non-linear and independent extension
- Without anticipation
- Without modification or duplication
- With incremental compilation
- While ensuring static type safety

The *Expression Problem*

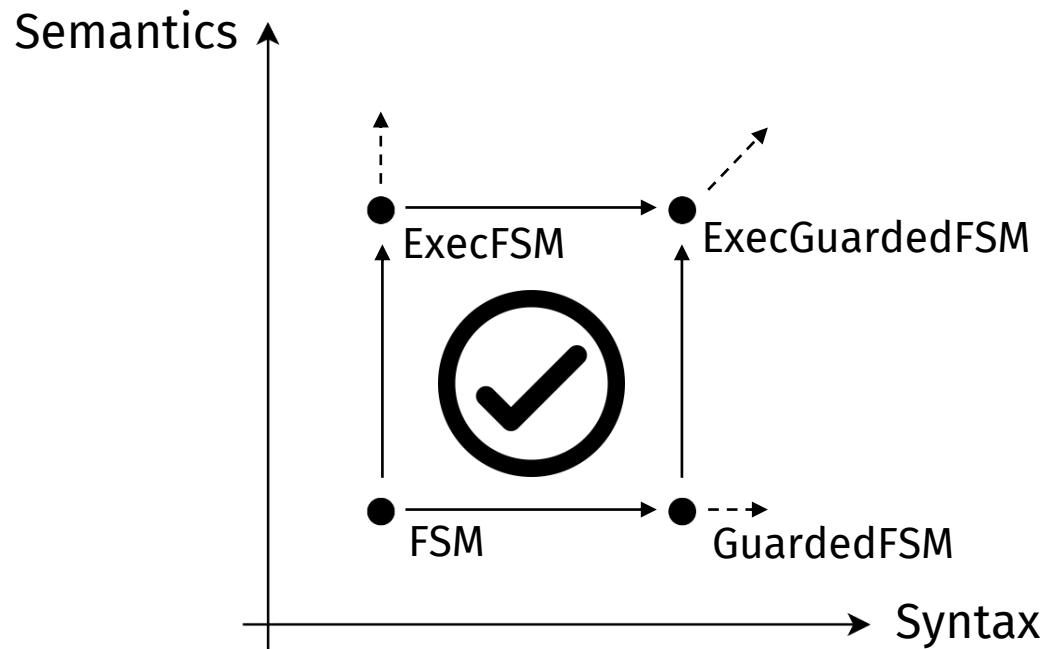


*"The expression problem is a new name for an old problem. The goal is to define a datatype by cases, where one can add **new cases** to the datatype and **new functions** over the datatype, **without recompiling existing code**, and while retaining **static type safety**."*

Philip Wadler, 1998

- Multi-methods
- Open classes
- Virtual classes
- ...

The *Expression Problem*



Extensibility for the Masses

Practical Extensibility with Object Algebras

Bruno C. d. S. Oliveira¹ and William R. Cook²

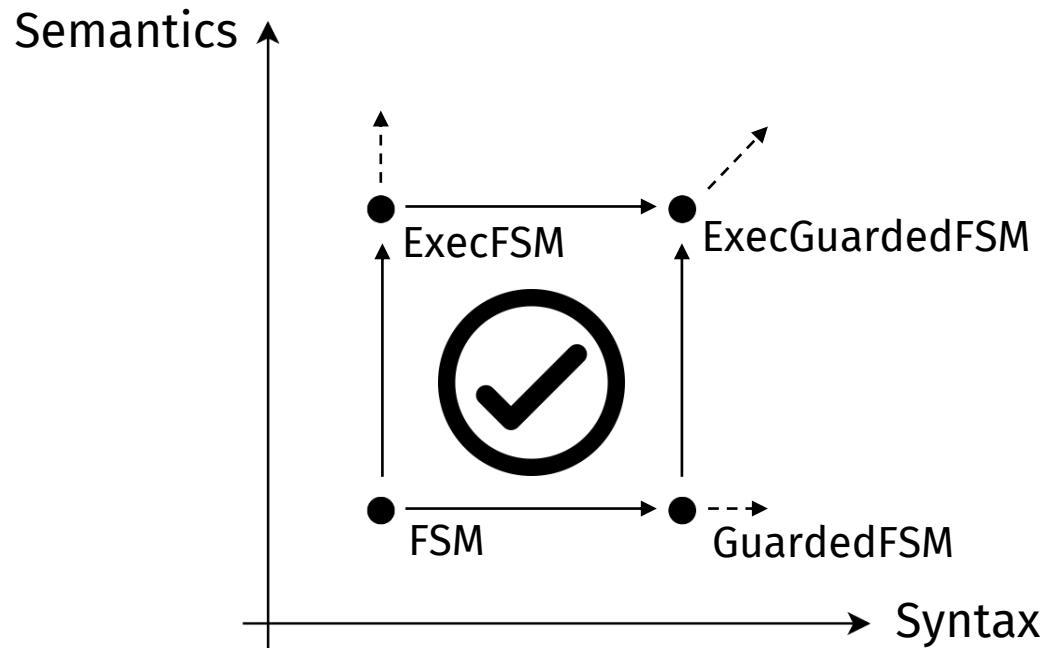
¹National University of Singapore
bruno@ropas.snu.ac.kr

²University of Texas, Austin
wcook@cs.utexas.edu

- An encoding of algebraic signatures in the object-oriented paradigm

Incompatible with metamodels!

The *Expression Problem*



Revisiting Visitors for Modular Extension of Executable DSMLs



Manuel Leduc
University of Rennes 1
France
manuel.leduc@irisa.fr

Thomas Degueule
CWI
The Netherlands
thomas.degueule@cwi.nl

Benoit Combemale
University of Rennes 1
France
benoit.combemale@irisa.fr

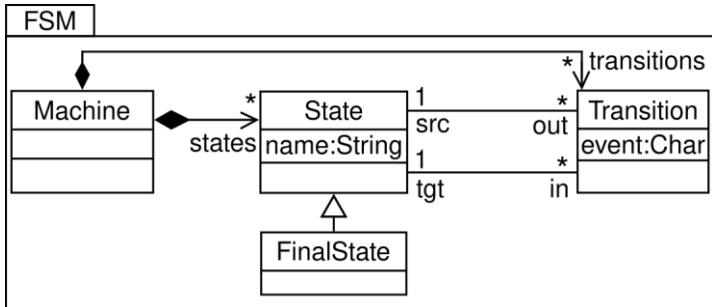
Tijs van der Storm
CWI & U of Groningen
The Netherlands
tijs.van.der.storm@cwi.nl

Olivier Barais
University of Rennes 1
France
olivier.baraais@irisa.fr

- A language implementation pattern that solves the Expression Problem
- Reconcile the structural extensibility of the object-oriented style with the behavioral extensibility of the functional style

The REVISITOR Pattern

The Revisitor Pattern



Abstract Syntax

```
interpret(State s, String evt) {  
    // Find a transition for the input event  
    State next = s.out.findFirst[event == evt]  
    // Fire it  
    next.fire()  
}
```

Execution Semantics

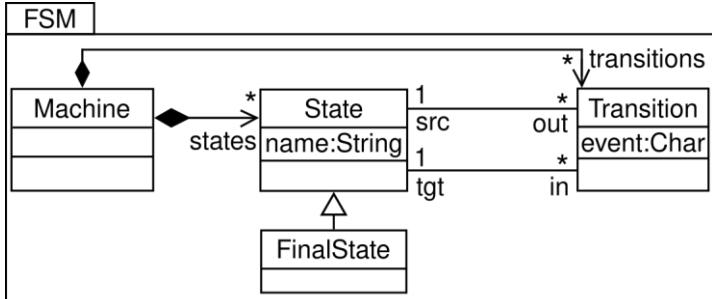
```
class Machine { List<State> states; [...] }  
class State { String name; [...] }  
class Trans { char event; [...] }  
class FS extends State { [...] }
```

Abstract Syntax Classes

```
interface FsmRev<M, S, F extends S, T> {  
    M machine(Machine it);  
    T trans (Trans it);  
    default M $(Machine it) { return machine(it); }  
    default T $(Trans it) { return trans(it); }  
    [...]  
}
```

REVISITOR Interface

The Revisitor Pattern



Abstract Syntax

```
interpret(State s, String evt) {
    // Find a transition for the input event
    State next = s.out.findFirst[event == evt]
    // Fire it
    next.fire()
}
```

Execution Semantics

```
class Machine { List<State> states; [...] }
class State { String name; [...] }
class Trans { char event; [...] }
class FS extends State { [...] }
```

Abstract Syntax Classes

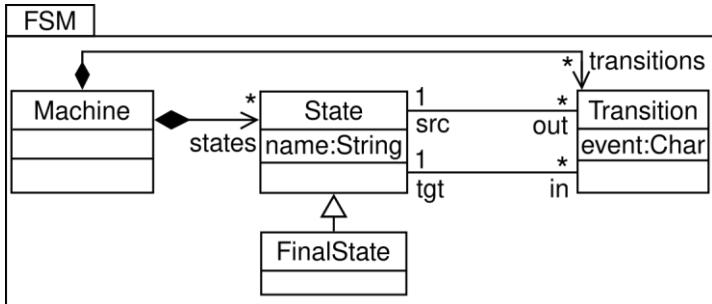
```
interface FsmRev<M, S, F extends S, T> {
    M machine(Machine it);
    T trans (Trans it);
    default M $(Machine it) { return machine(it); }
    default T $(Trans it) { return trans(it); }
    ...
}
```

REVISITOR Interface

```
interface EvalFsm extends FsmRev<EM,ES,EFS,ET> {
    default ES state(State it) {
        return (evt) -> {
            State next = it.out.findFirst[event==evt];
            $(next).fire();
        };
    }
    ...
}
```

REVISITOR Implementation

The Revisitor Pattern



Abstract Syntax

```
interpret(State s, String evt) {  
    // Find a transition for the input event  
    State next = s.out.findFirst[event == evt]  
    // Fire it  
    next.fire()  
}
```

Execution Semantics

```
pretty-print(State s) {  
    print("State " + s.name)  
    print("Transitions:")  
    for (t in s.transitions)  
        pretty-print(t)  
}
```

Printing Semantics

```
class Machine { List<State> states; [...] }  
class State { String name; [...] }  
class Trans { char event; [...] }  
class FS extends State { [...] }
```

Abstract Syntax Classes

```
interface FsmRev<M, S, F extends S, T> {  
    M machine(Machine it);  
    T trans (Trans it);  
    default M $(Machine it) { return machine(it); }  
    default T $(Trans it) { return trans(it); }  
    [...]  
}
```

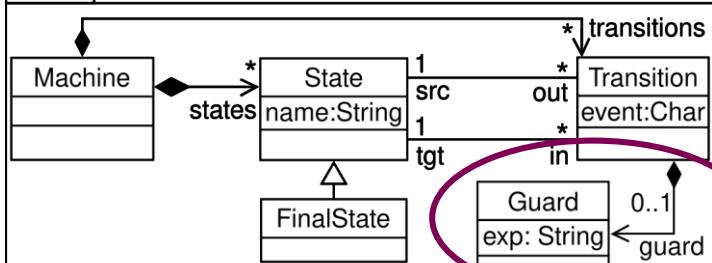
REVISITOR Interface

```
interface PrintFsm extends FsmRev<Pr,Pr,Pr,Pr> {  
    default Pr state(State it) {  
        return () -> "State" + it.name + [...] +  
            "Transitions:" +  
            it.trans.map[t | $(t).print()];  
    }  
    [...]  
}
```

REVISITOR Implementation

The Revisitor Pattern

FSM



Abstract Syntax

```
interpret(State s, String evt) {  
    // Find a transition for the input event  
    State next = s.out.findFirst[event == evt]  
    // Fire it  
    next.fire()  
}
```

Execution Semantics

```
pretty-print(State s) {  
    print("State " + s.name)  
    print("Transitions:")  
    for (t in s.transitions)  
        pretty-print(t)  
}
```

Printing Semantics

```
class Machine { List<State> states; [...] }  
class State { String name; [...] }  
class Trans { char event; Guard guard; [...] }  
class FS extends State { [...] }  
class Guard { String exp; }
```

Abstract Syntax Classes

```
interface GuardFsmRev<M, S, F extends S, T, G>  
    extends FsmRev<M, S, F, T> {  
    G guard(Guard it);  
    default G $(Guard it) { return guard(it); }  
}
```

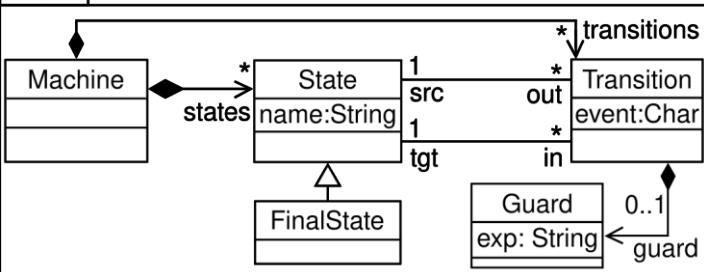
REVISITOR Interface

```
interface PrintGuardFsm  
    extends PrintFsm,  
        GuardFsmRev<Pr, Pr, Pr, Pr, Pr> {  
    default Pr guard(Guard it) {  
        return () -> it.exp;  
    }  
    @Override  
    default Pr trans(Trans it) {  
        return () -> super.print() +$(it.guard).print();  
    }  
}
```

REVISITOR Implementation

The ALE Language

FSM



Abstract Syntax

```
open class Machine {  
    def String print() {  
        return "Machine " + self.name + "\n" +  
            self.states.map[s | $[s].print()];  
    }  
}  
  
open class State {  
    def String print() {  
        return "State " + self.name + "\n" +  
            self.outgoing.map[t | $[t].print()];  
    }  
}  
  
open class Guard {  
    def String print() {  
        return self.exp;  
    }  
}
```

ALE: a concise and intuitive language
for defining the semantics of
metamodel-based languages

```
class Machine { List<State> states; [...] }  
class State { String name; [...] }  
class Trans { char event; Guard guard; [...] }  
class FS extends State { [...] }  
class Guard { String exp; }
```

Abstract Syntax Classes

```
interface GuardFsmRev<M, S, F extends S, T, G>  
    extends FsmRev<M, S, F, T> {  
    G guard(Guard it);  
    default G $(Guard it) { return guard(it); }  
}
```

REVISITOR Interface

```
interface PrintGuardFsm  
    extends PrintFsm,  
        GuardFsmRev<Pr, Pr, Pr, Pr, Pr> {  
    default Pr guard(Guard it) {  
        return () -> it.exp;  
    }  
    @Override  
    default Pr trans(Trans it) {  
        return () -> super.print() + $(it.guard).print();  
    }  
}
```

REVISITOR Implementation

Summary

- **Scientific contributions**

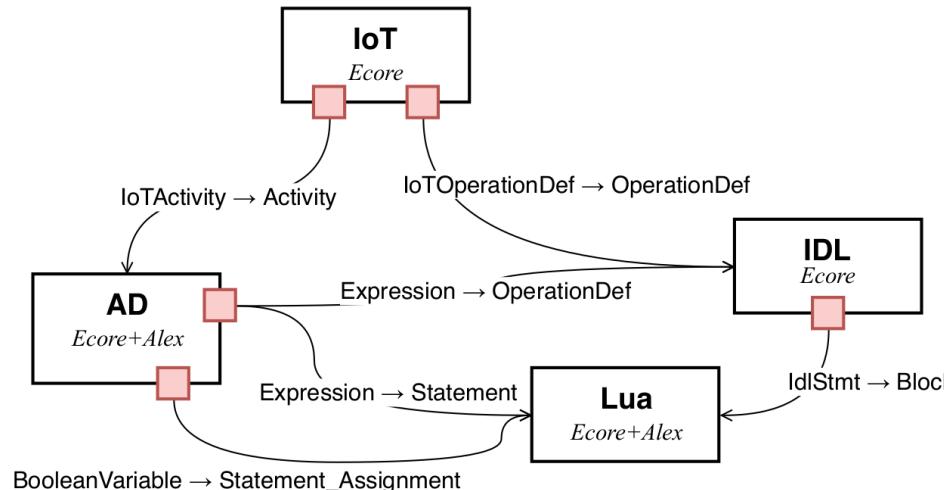
- First broadly-applicable solution to the *Expression Problem*
 - Applicable in C++, Java, C#, Scala, the *Eclipse Modeling Framework*, etc.
- Strong theoretical foundations
 - Object algebras, Visitors, etc.
 - Object-oriented structural extensibility ◦ functional behavioral extensibility
- Later extended to modular language *composition*

Modular Language Composition for the Masses

M. Leduc, T. Degueule, B. Combemale

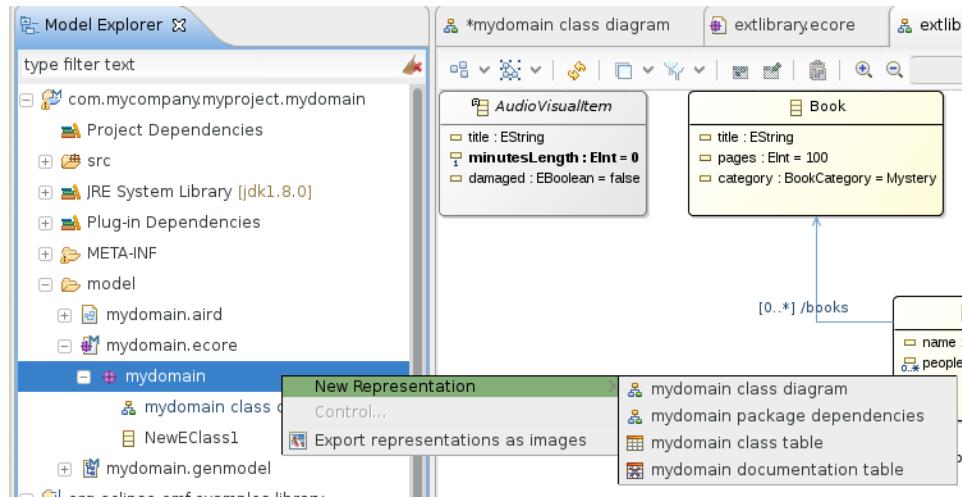
11th International Conference on Software Language Engineering (SLE'18)

Best Artifact Award



Summary

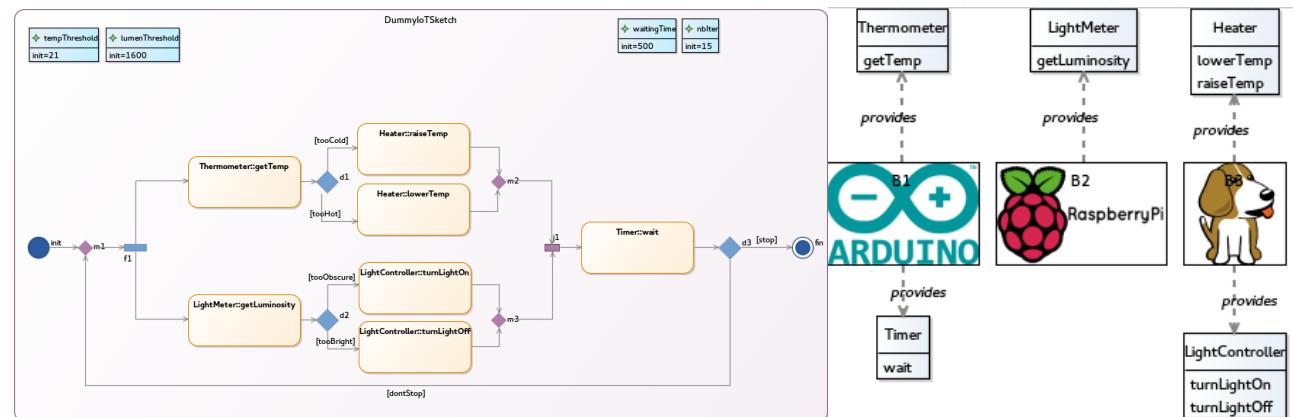
- Technological contributions



[Talk@EclipseCon'17] EcoreTools Next:
Executable DSL made (more) accessible

- Experimental validation

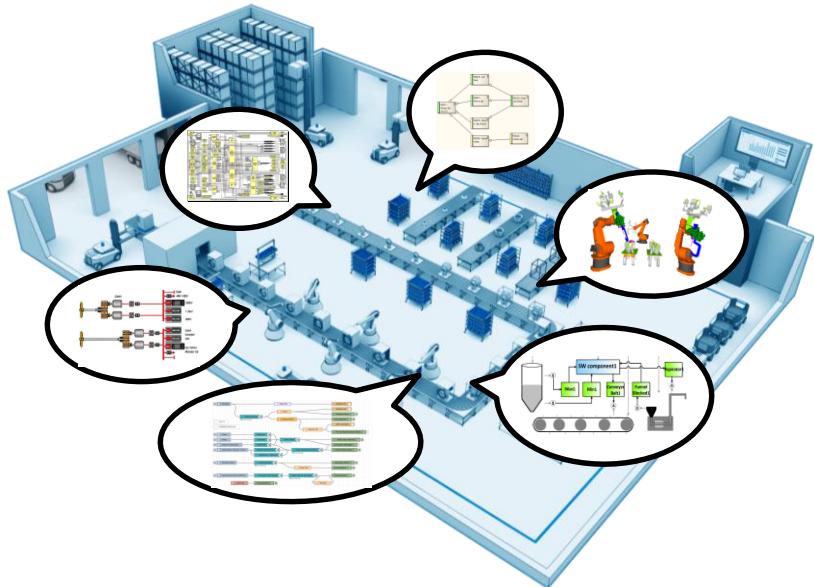
- UML
- State machines
- Internet of Things
- etc.



Research Project

*Software language diversity
Linguistic analysis of APIs*

Software language diversity

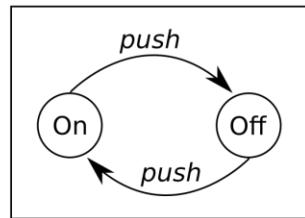


Domain diversity

API

```
FSM fsm =  
    new FSMBuilders("Button")  
    .init("On")  
    .to("Off").on("push")  
    .state("Off")  
    .to("On").on("push")  
    .build();
```

Graphical



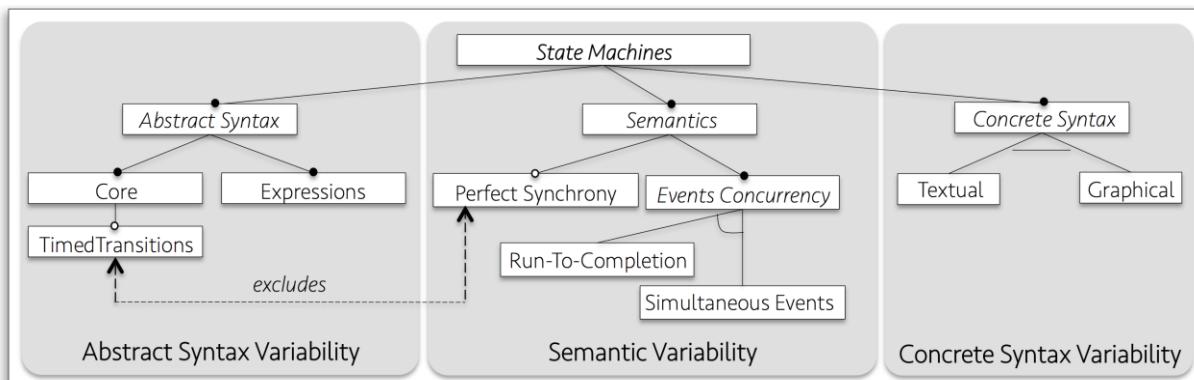
Textual

```
machine Button  
state On  
    on push => Off  
end  
state Off  
    on push => On  
end
```

Annotations

```
class Button {  
    @State static String ON = "On";  
    @State static String OFF = "Off";  
    @Trans(on="push", in=ON, out=OFF)  
    public void doPush() {  
    }  
}
```

Shape diversity

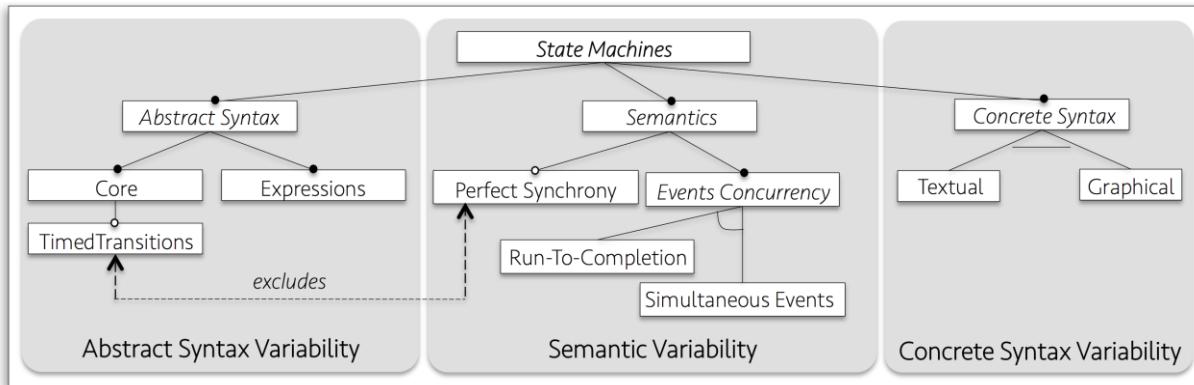


Abstraction diversity

Language Families

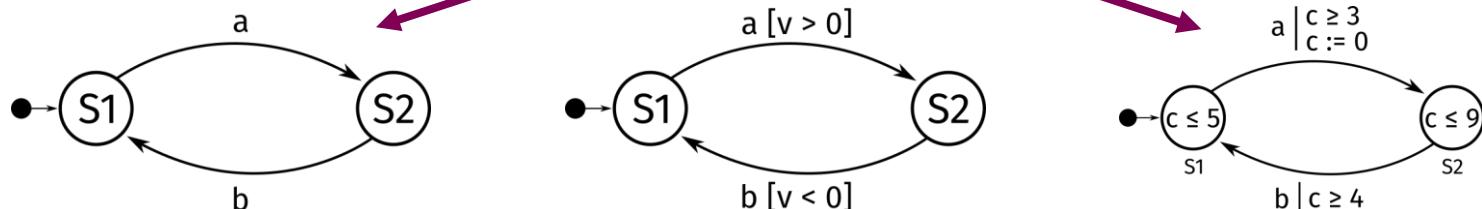


configure



compose

Language Components



Linguistic Analysis of APIs

- APIs are brittle, hard to learn, constantly evolving

```
MongoClient client = new MongoClient("mongodb://db.server");
DB db = client.getDB("MyDB");
DBCollection col = db.getCollection("MyCollection");
col.insert(new BasicDBObject(...));
DBCursor cursor = col.find(new BasicDBObject(...));
DBObject obj = cursor.one();
String name = obj.get("name");
```

- Intuition: reify APIs as languages

```
connectTo "mongodb://db.server"
col = get MyCollection from MyDB
insert [...] in col
cursor = find [...] in col
name = cursor[name]
```

- Apply linguistic methods to the analysis of APIs
 - Co-evolution of client code \leftrightarrow API \Leftrightarrow Co-evolution of model \leftrightarrow language

Apprivoiser et synthétiser la diversité des langages logiciels

Thomas Degueule

Software Analysis and Transformation Group

Centrum Wiskunde & Informatica

<https://tdegueul.github.io>

Intégration

UMR 9189 – CRYStAL – Lille

UMR 5505 – IRIT – Toulouse

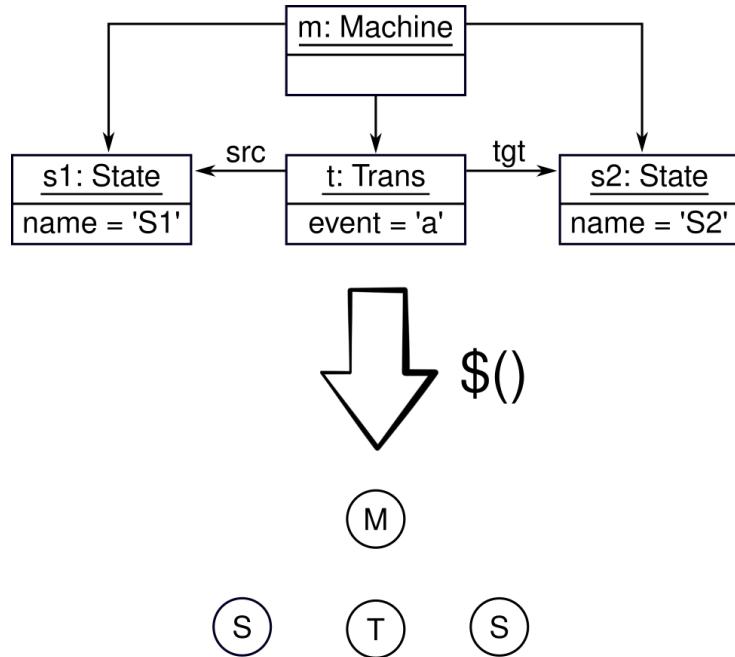


<Backup>

Revisitor Interface

- Extensible mapping between syntactic objects and abstract semantic types

```
interface FsmRev<M, S, F extends S, T> {  
    M machine(Machine it);  
    S state (State it);  
    F fState (FState it);  
    T trans (Trans it);  
  
    default M $(Machine it) { return machine(it); }  
    default F $(FState it) { return fState(it); }  
    default T $(Trans it) { return trans(it); }  
    default S $(State it) {  
        if (it instanceof FState)  
            return fState(it);  
        return state(it);  
    }  
}
```



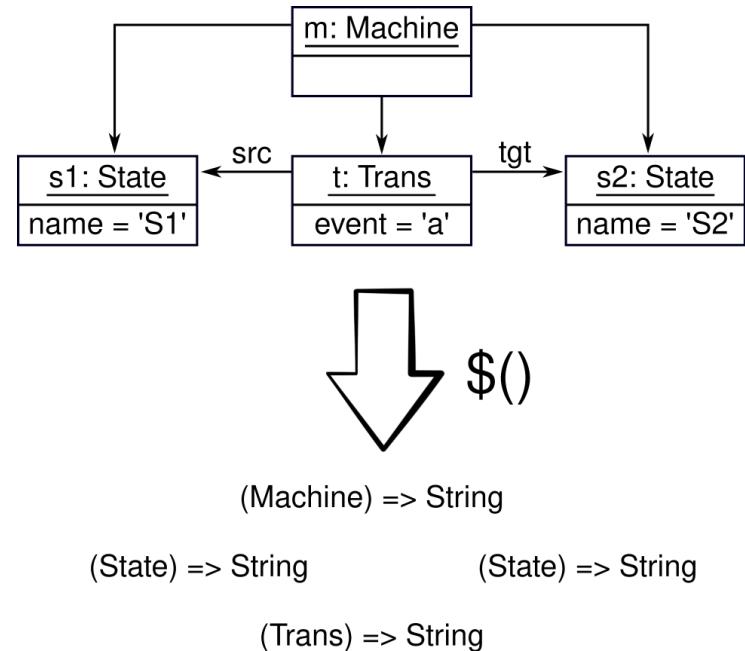
Revisitor Implementation

- Assign concrete types to the semantic objects
- Retroactive implementation of the semantics

```
interface Pr { String print(); }

interface PrintFsm extends FsmRev<Pr, Pr, Pr, Pr> {
}

}
```

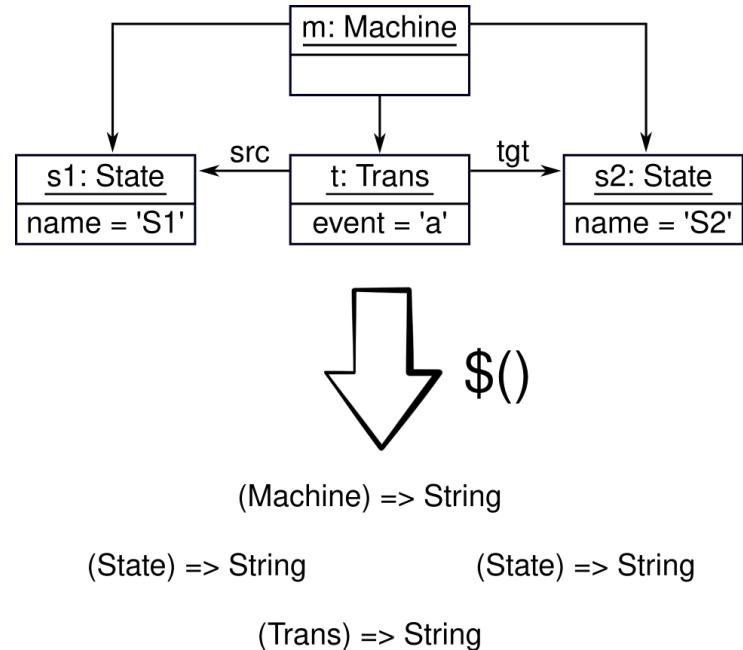


Revisitor Implementation

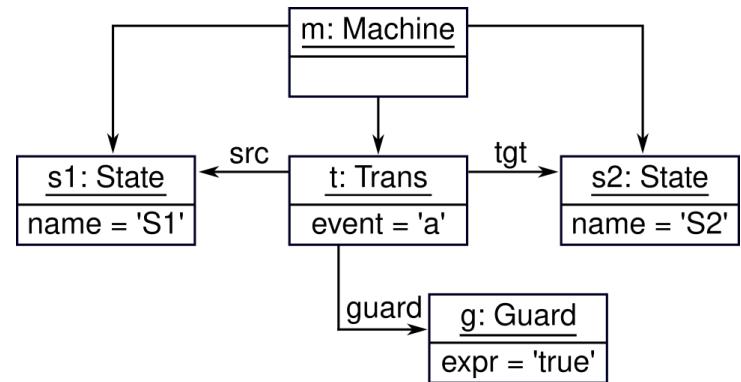
- Assign concrete types to the semantic objects
- Retroactive implementation of the semantics

```
interface Pr { String print(); }

interface PrintFsm extends FsmRev<Pr, Pr, Pr, Pr> {
    default Pr machine(Machine it) {
        return () -> it.name + ":\n" +
            it.states.map(s -> ${s}.print());
    }
    default Pr state(State it) {
        return () -> /*      print a state      */;
    }
    default Pr fState(FState it) {
        return () -> /* print a final state */;
    }
    default Pr trans(Trans it) {
        return () -> /* print a transition */;
    }
}
```



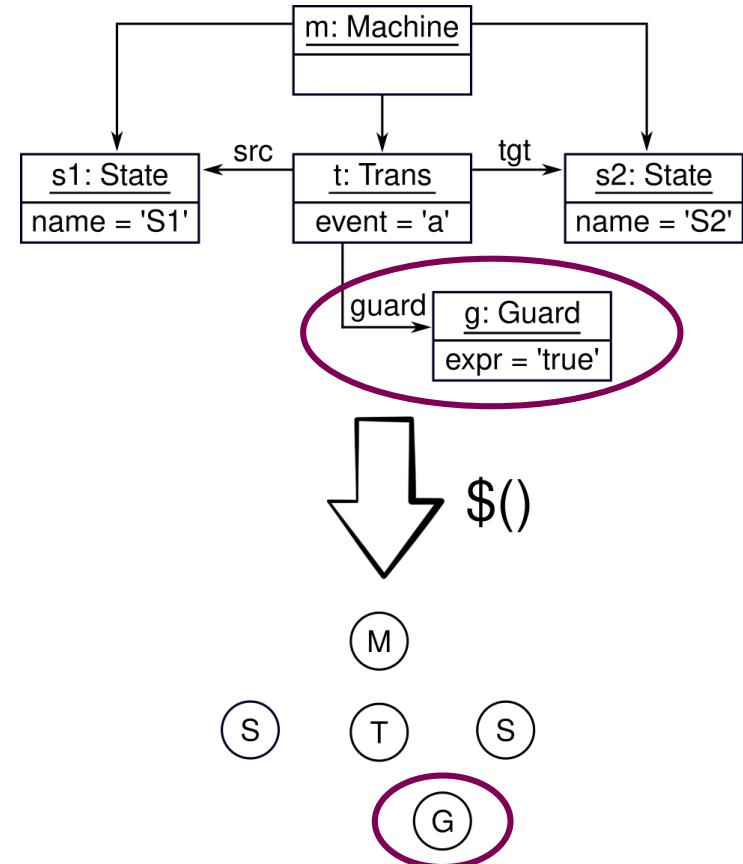
Modular Extension with Revisitors



Modular Extension with Revisitors

```
interface GuardFsmRev<M, S, F extends S, T, G>
  extends FsmRev<M, S, F, T> {
  G guard(Guard it);
  default G $(Guard it) { return guard(it); }
}
```

Modularly extend the mapping



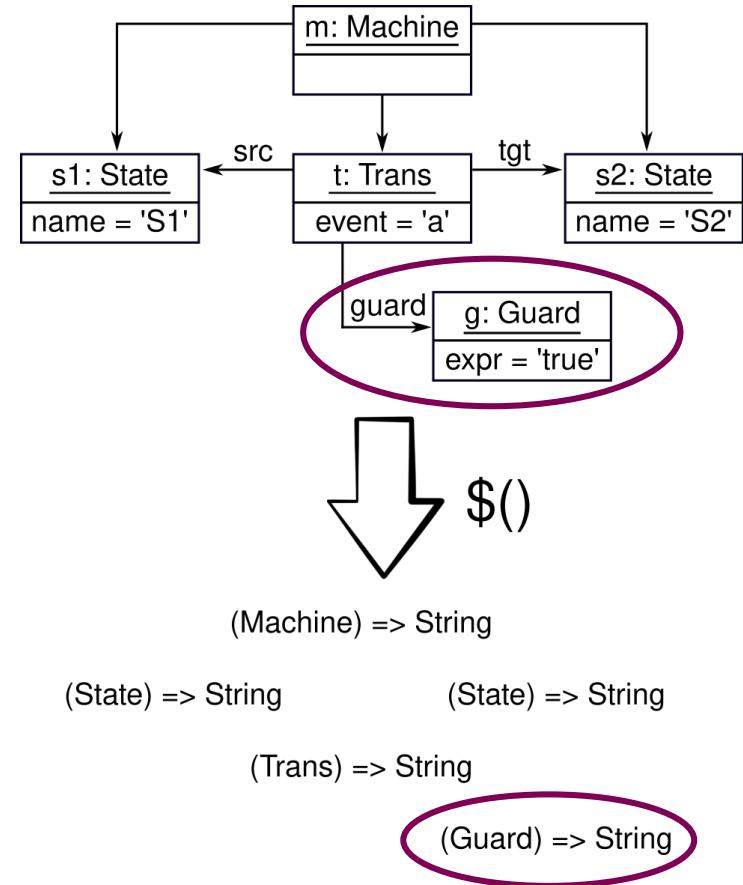
Modular Extension with Revisitors

```
interface GuardFsmRev<M, S, F extends S, T, G>
  extends FsmRev<M, S, F, T> {
  G guard(Guard it);
  default G ${Guard it) { return guard(it); }
}
```

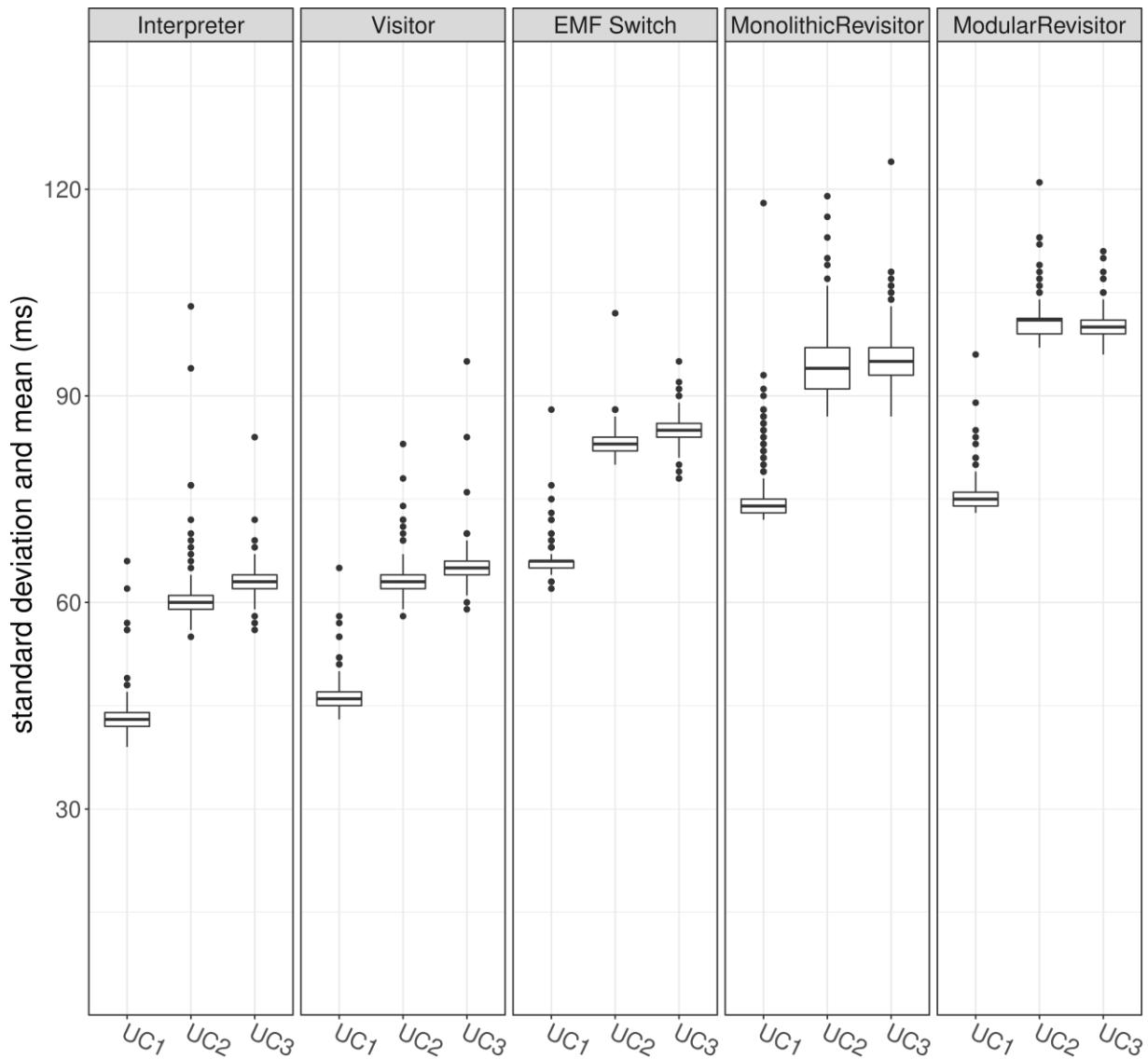
Modularly extend the mapping

```
interface PrintGuardFsm
  extends PrintFsm,
         GuardFsmRev<Pr, Pr, Pr, Pr, Pr> {
  default Pr guard(Guard it) {
    return () -> it.expr;
  }
  @Override
  default Pr trans(Trans it) {
    return () -> super.print() + ${it.guard).print();
  }
}
```

Modularly extend the semantics



Revisitors Benchmark



Specify Equality Condition

To specify equality conditions, use the `com.mongodb.client.model.Filters.eq_` method to create the query filter document:

```
and(eq( <field1>, <value1>), eq( <field2>, <value2>) ...)
```

copy

The following example selects from the `inventory` collection all documents where the `status` equals "D":

```
findIterable = collection.find(eq("status", "D"));
```

copy

This operation corresponds to the following SQL statement:

```
SELECT * FROM inventory WHERE status = "D"
```

copy

Official Documentation

Repositories 0

Code 38K+

Commits 68

Issues 130

Marketplace 0

Topics 0

Wikis 46

Users 0

Languages

Java X

Text 4,924

Showing 38,397 available code results [?](#)

DBCursor	<code>find(DBObject query)</code> Select documents in collection and get a cursor to the selected documents.
DBCursor	<code>find(DBObject query, DBCollectionFindOptions options)</code> Select documents in collection and get a cursor to the selected documents.
DBCursor	<code>find(DBObject query, DBObject projection)</code> Select documents in collection and get a cursor to the selected documents.
DBCursor	<code>find(DBObject query, DBObject projection, int numToSkip, int batchSize)</code> Deprecated. use <code>DBCursor.skip(int)</code> and <code>DBCursor.batchSize(int)</code> on the <code>DBCursor</code> returned from <code>find(DBObject, DBObject)</code>
DBCursor	<code>find(DBObject query, DBObject projection, int numToSkip, int batchSize, int options)</code> Deprecated. use <code>DBCursor.skip(int)</code> , <code>DBCursor.batchSize(int)</code> and <code>DBCursor.setOptions(int)</code> on the <code>DBCursor</code> returned from <code>find(DBObject, DBObject)</code>

API Documentation

Get ID of last inserted document in a mongoDB w/ Java driver

Is there an easy way to get the ID (ObjectID) of the last inserted document of a mongoDB instance using the Java driver?

92

java mongodb

share improve this question

19

add a comment

asked Jul 26 '10 at 21:03
 Matt W
4,370 2 25 39

7 Answers

active oldest votes

Hate to answer my own question, but I just realized you can do this:

177

```
BasicDBObject doc = new BasicDBObject( "name", "Matt" );
collection.insert( doc );
ObjectId id = (ObjectId)doc.get( "_id" );
```

✓

share improve this answer

answered Jul 26 '10 at 21:41
 Matt W
4,370 2 25 39

5 answering your own question is [encouraged](#). don't worry about it. – [Eliran Malka](#) Apr 13 '14 at 10:16

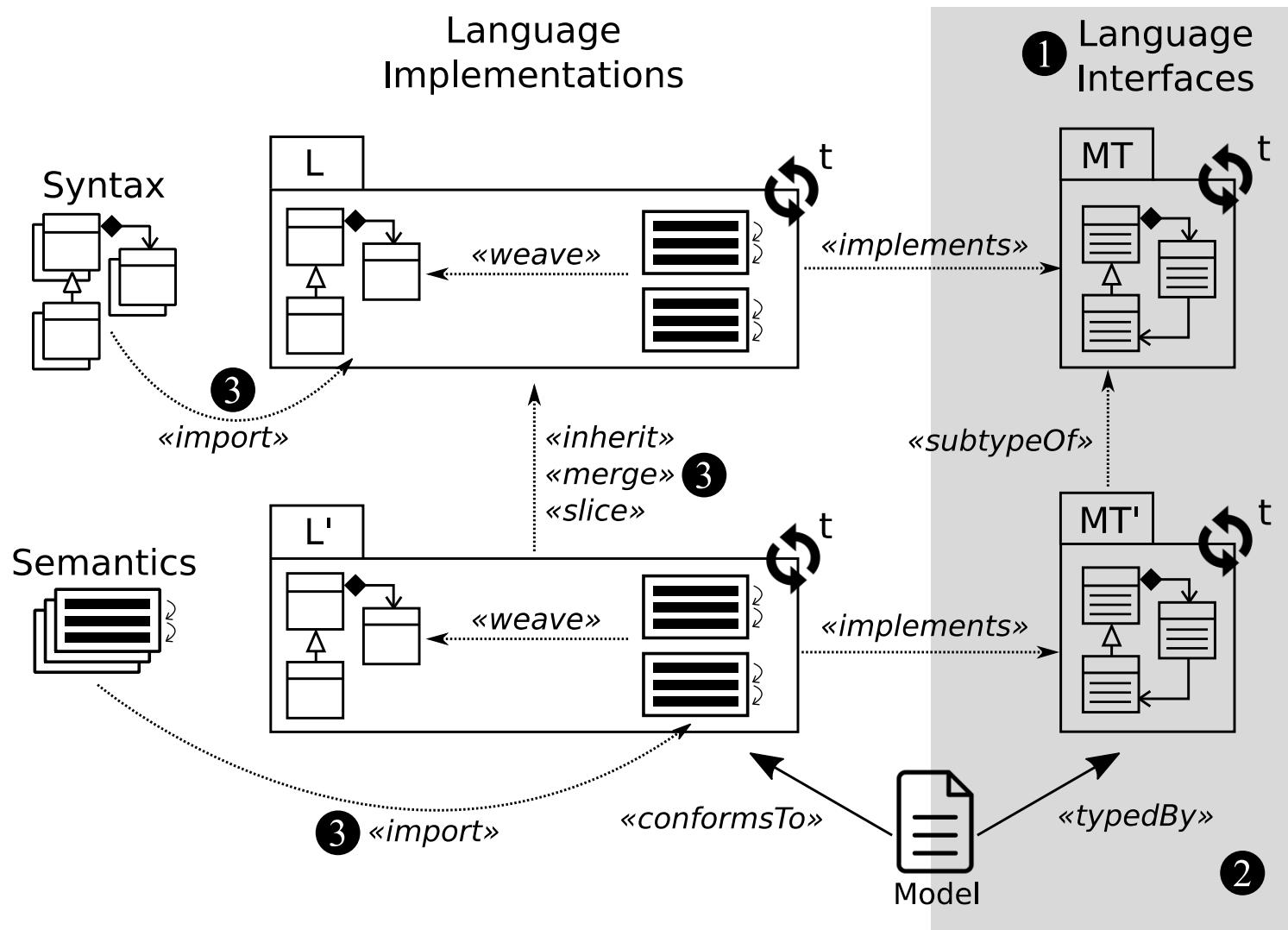
Thanks for this . Do u know how to do the same using spring data mongodb ? – [Mohammed shebin](#) Nov 2 '15 at 7:09

add a comment

Q&A Websites

Similar Projects

Thèse (2013 – 2016)



Post-doctorat (2017 – 2020)

- Groupe Software Analysis and Transformation au CWI, Amsterdam

- CWI—Inria ALE (*Agile Language Engineering*)

- Implémentation modulaire de langages logiciels
- Modélisation flexible

 <http://eclipse.org/scava/>

[MODELS'17][SLE'18₁][COMLAN'18]
[SLE'18₂][SLE'18₃]

- H2020 CROSSMINER (*Developer-centric Knowledge Mining from OSS Repositories*)

- Génie logiciel empirique, rétro-ingénierie
- Analyse d'APIs, systèmes de recommandations

[MSR'18]

[ICSE'19][ICSME'19 – en rédaction]



Lina Ochoa
ICSE'19 (accepté)
MSR'18 (publié)



Manuel Leduc
MODELS'17 (publié)
COMLAN'18 (publié)
SLE'18₁ (publié)



Fabien Coulon
SLE'18₃ (publié)