TOPCASED

Toolkit in Open-Source for Critical Application & Systems Development

# Using model transformation technologies to implement a state machine simulator

**Stéphane Duprat**
**Raphaël Faudou**
**David Ribeiro Campelo**
**Ludi Akue**

Atos Origin

# Agenda

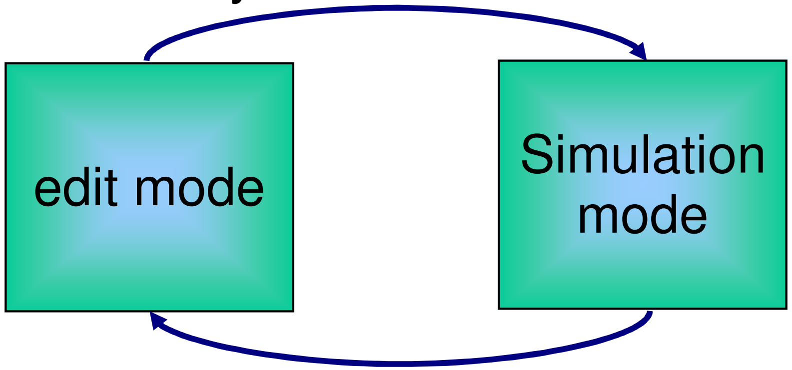
- Topcased Model Simulation Keypoints
- Expected benefits
- Tool design
- Dynamic Model & Dynamic metamodel
- Dynamic metamodel of UML
- Using M2M ?
- Example of a SmartQVT Implementation
- Conclusion

# Topcased model simulation keypoints

**What is simulated ?**
State machine defined behavioural specifications :
- UML StateMachine
- SAM Automaton
- potentially other DSM

**GUI features**
- same aspect as in edition mode
- added dynamic informations
- control console

**Close to the model**
simulation by "model-execution"




**Control features**
- INPUT:
  - interactive Mode
  - scenario in input
- OUTPUT:
  - model trace
  - scenario

# Expected benefits of the simulation

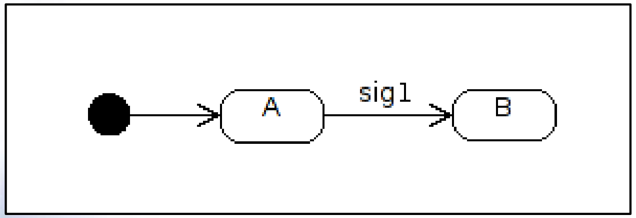
- Enable model debugging
  - Early error detection (bad connection or trigger, wrong OCL expression)
- Help designer and architect:
  - Enable them to check interactively semantic of the model
- Dynamic graphical demonstrator
  - For presentation purpose, visualization capabilities
- For verification at the specification/design stage
  - Verification of the model against its specification (with scenarii form the use-cases)
  - Non-regression scenarii (by replaying reference scenarii)

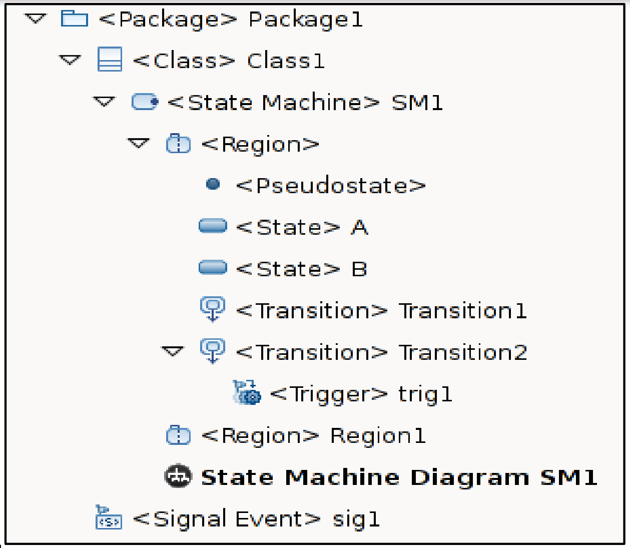
# Design of the tools

- Main principles
  - Reuse of the model editor
    - Preserve initial disposition
  - Genericity
    - Topcased simulation targetq UML/SysML and DSM
      - Generic components
      - Specific components
  - “Full model" approach
    - Model execution
    - Dynamic model / metamodel
    - Scenario model / metamodel
    - M2M transformations

# The Dynamic Model & the Dynamic Metamodel

- The Static definition of a StateMachine and its associated model




- Dynamic "informations" to add for a dynamic execution
  - current state
  - fireables transitions


•the current state is "A"
•"sig1" is the only fireable transition

•the current state is "B"
•there is no fireable transition

# The Dynamic Model & Dynamic Metamodel

- The dynamic informations
  - current state
  - fireables transitions

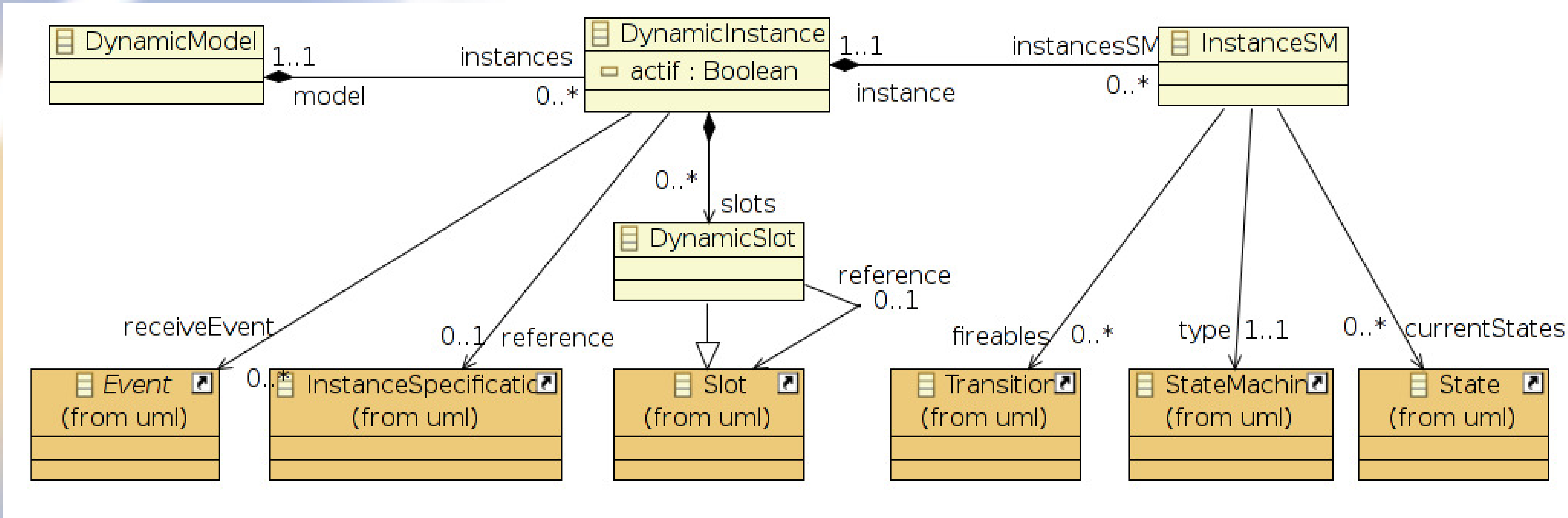
- How to process them ?

  - in case of code generation : the state of the generated program
    - not our case

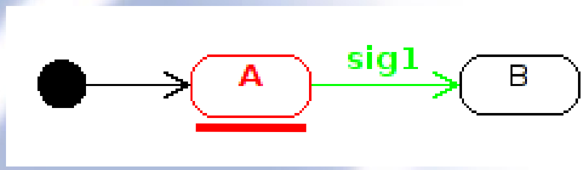
  - or in java variables of the tool
    - desynchronized with the spec. model

  - in a model
    - that refers the spec. model
    - to allow M2M techniques

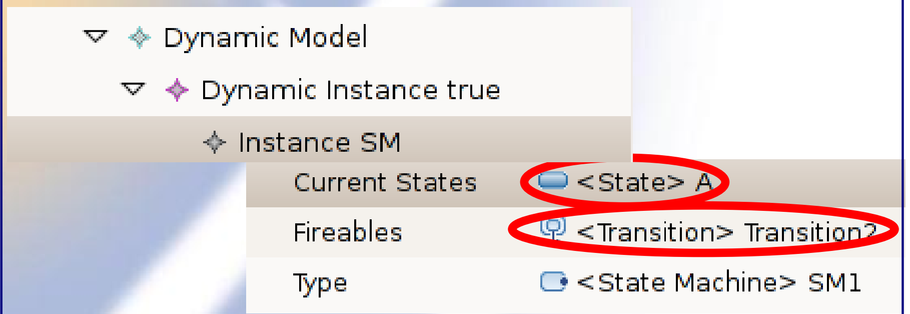
# The UML Dynamic Metamodel



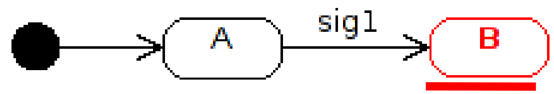
# The dynamic model in action



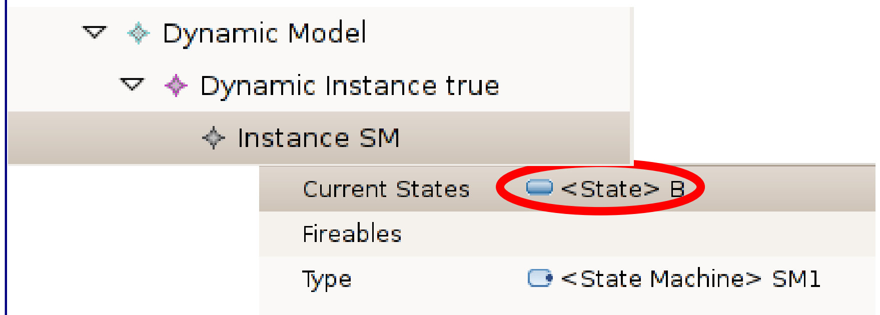

- *the current state is "A"*
- *"sig1" is the only fireable transition*



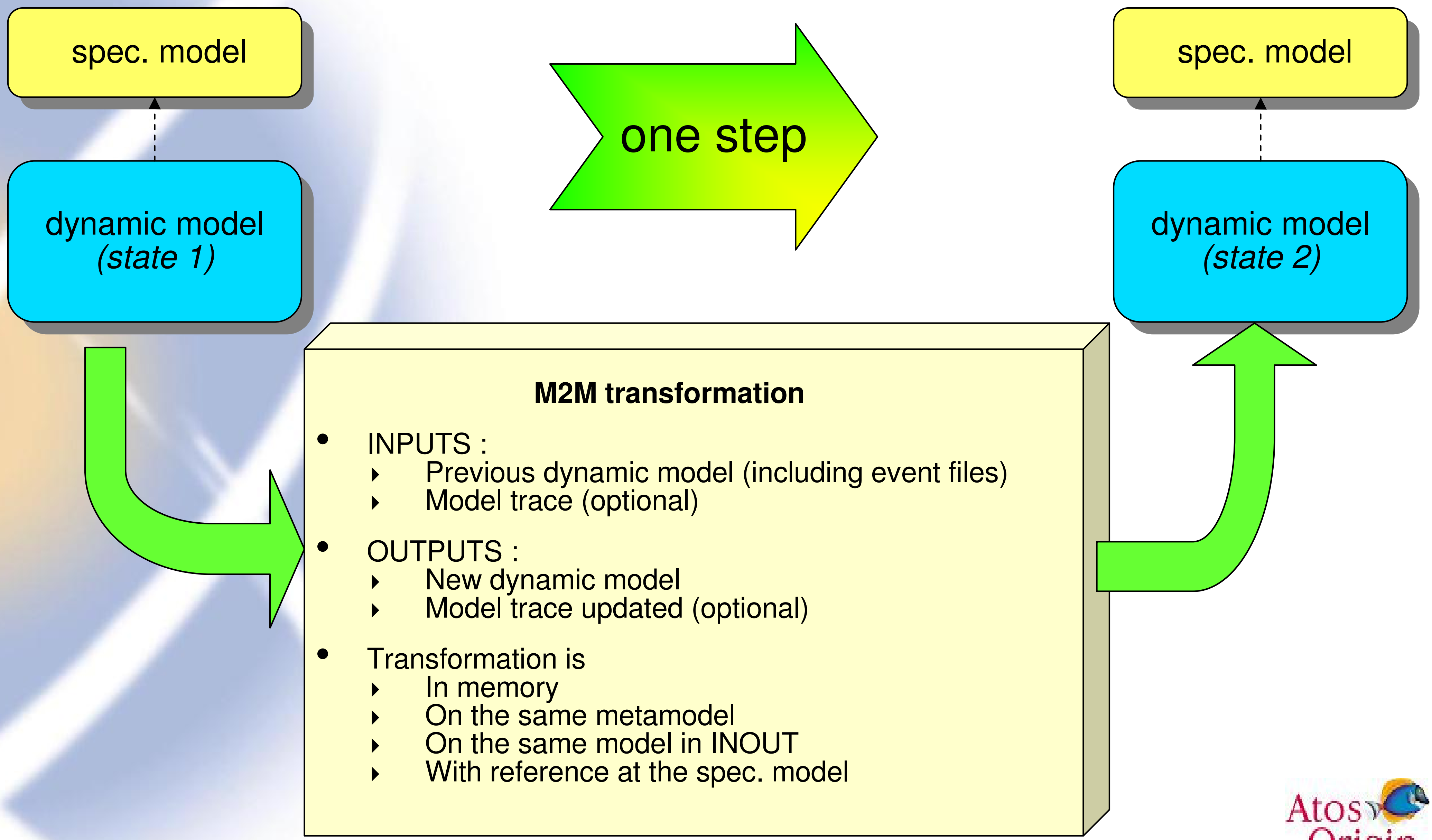

sig1 →




- *the current state is "B"*
- *there is no fireable transition*



# Using M2M transformations



# Example of a smartQVT implementation

```
--mapping on InstanceSM : actualisation of
currentStates and deduction of fireables
transitions
mapping inout
umldynamic::InstanceSM::InstanceSM2InstanceS
M(in pInjectSimEvent:InjectSimEvent) {

var lEvent := pInjectSimEvent.event;

var lFutureTransition :Sequence(Transition)
  :=currentState->collect(c |c.outgoing);

lFutureTransition:= lFutureTransition
  ->select(t | (self->collect(i |
  i.instance))->possibleTransitionGuard(t)
  ->first());

var lnewcurrentState : Sequence(State)
  :=( lFutureTransition
  ->select(t | t.trigger->exists(tri|
  tri.event.name=lEvent.name)))
  ->collect(t2 | t2.target)[State];
.../...
```

```
--loldCurrentState : olds currents states
var loldCurrentState : Set(State) :=
  currentState
->select(s | s.outgoing
  ->exists(t | lnewcurrentState
    ->exists(ns |ns = t.target)))[State];
.../...

lnewcurrentState += currentState->select(s |
  not(loldCurrentState.include(s)));

--affect the news currents states and calculate the
fireables of them
currentStates := lnewcurrentState;

fireables := currentStates->collect(s |
  s.outgoing);

fireables +=((currentStates->collect(s1 |
  s1.container))->collect(r | r.state))
  ->collect(s2 |s2.outgoing);
}
```



Atos Origin

# Conclusion

- Technical
  - Effective and fast integration with other components
  - QVT is a high level language, with easier maintenance
  - Performances are the same as Java
- Methodology
  - Good separation with other modules
  - But QVT spec. should be reread for semantic definitions
  - M2M best solution for tools dealing with models

# Links

- UML
  http://www.uml.org/
- SysML
  http://www.omgsysml.org/
- smartQVT
  http://smartqvt.elibel.tm.fr/
- Topcased
  http://www.topcased.org/
- Topcased Simulation Project
  http://gforge.enseeiht.fr/projects/topcased-ms/



Atos Origin

# Contacts

- Stephane Duprat (stephane.duprat@atosorigin.com)
- Raphaël Faudou (raphael.faudou@atosorigin.com)
- David Ribeiro (david.ribeirocampelo@atosorigin.com)
- Ludi Akue (ludi.akue@atosorigin.com)