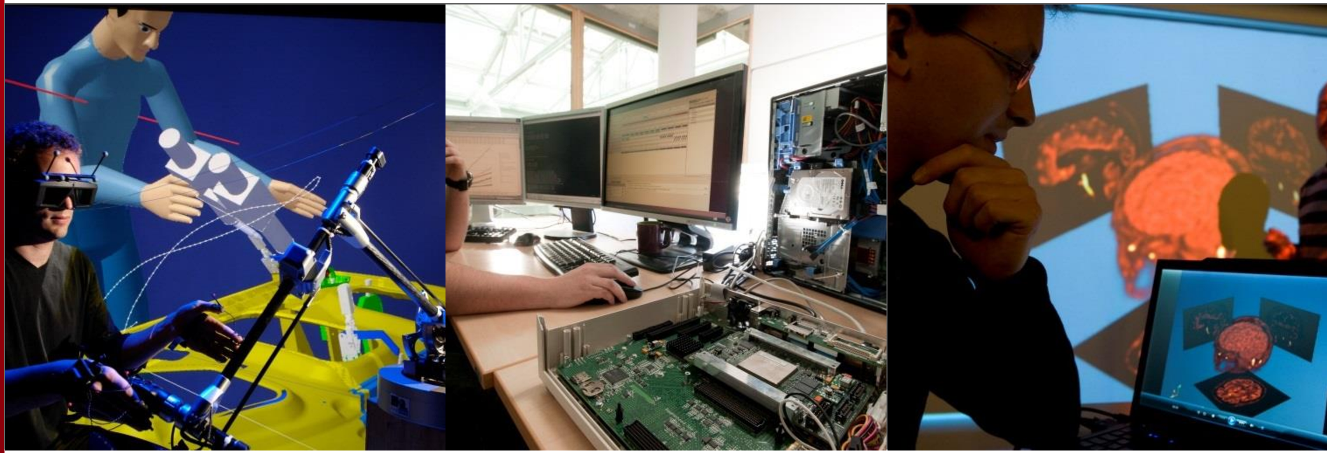
[PSSM] – WEBEX NOVEMBER 6 TH

Jérémie TATIBOUET (CEA LIST)

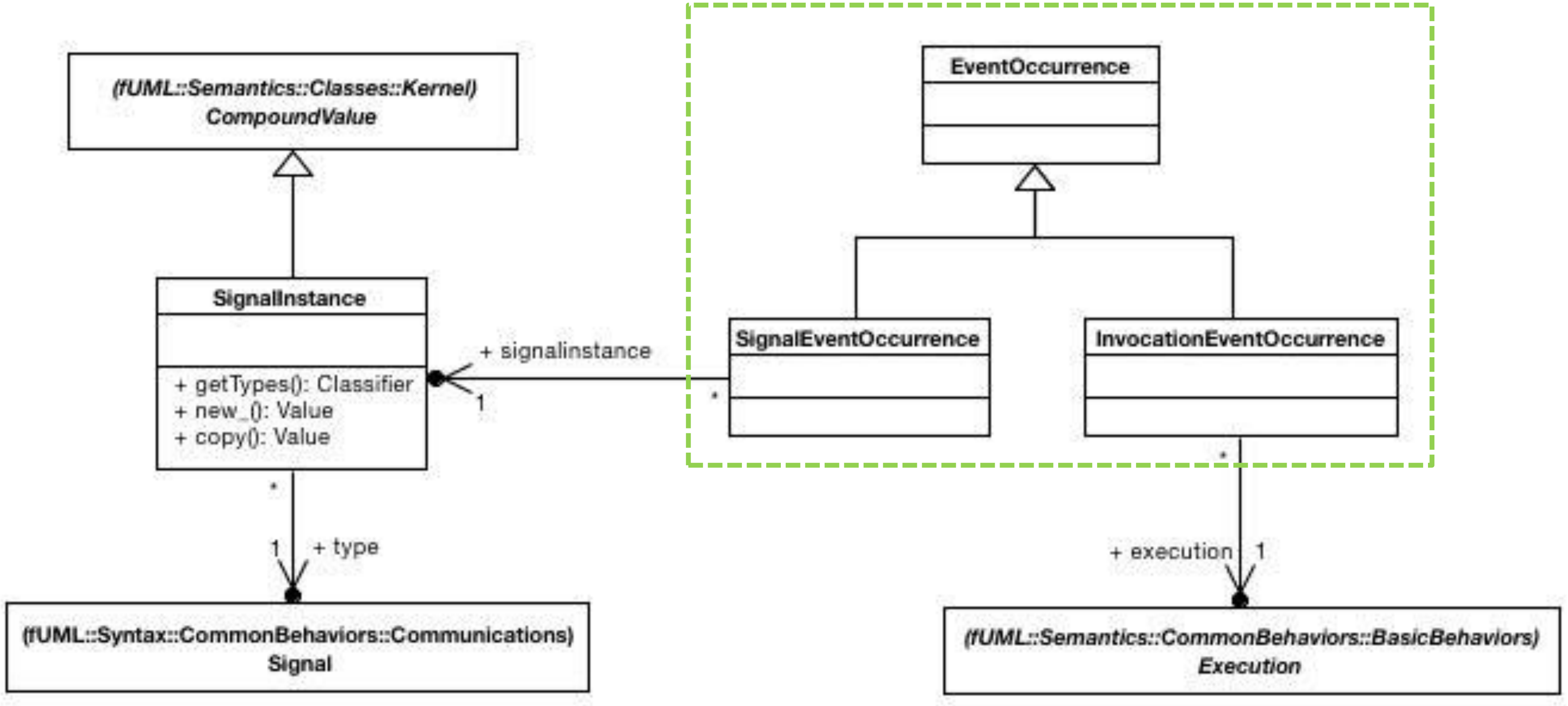
Arnaud CUCCURU (CEA LIST)



MAJOR CHANGE INTRODUCED BY FUML 1.2

Event occurrence

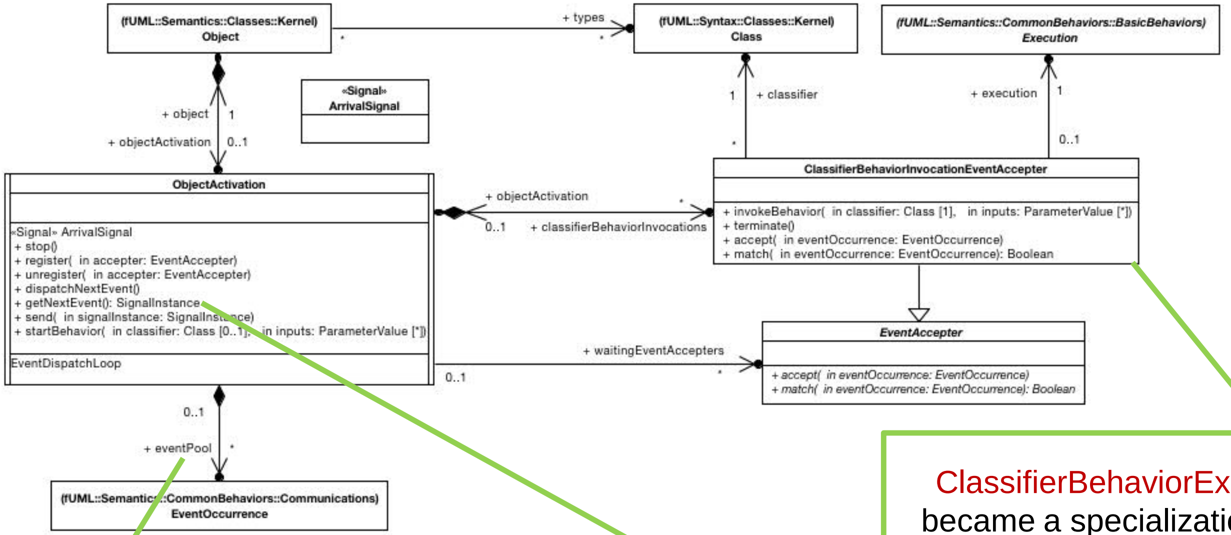
- Separation between the event itself and the data that may be shipped
  - SignalEventOccurrence (i.e. reference the signal instance)
- The pool now stores event occurrence
  - First step to make CalEvent integration simpler



MAJOR CHANGE INTRODUCED BY FUML 1.2

Run to completion step

- Each classifier behavior starts on a run to completion step
  - An event accepter is registered for the classifier behavior
  - An “InvocationEventOccurrence” is placed in the pool
  - The consumption of the event makes the classifier behavior to start



(fUML::Semantics::Classes::Kernel)
Object
+ types
*
*
(fUML::Syntax::Classes::Kernel)
Class
(fUML::Semantics::CommonBehaviors::BasicBehaviors)
Execution
«Signal»
ArrivalSignal
+ object
1
+ objectActivation
0..1
1
+ classifier
*
+ execution
1
0..1
ObjectActivation
«Signal» ArrivalSignal
+ stop()
+ register(in accepter: EventAccepter)
+ unregister(in accepter: EventAccepter)
+ dispatchNextEvent()
+ getNextEvent(): SignalInstance
+ send(in signalInstance: SignalInstance)
+ startBehavior(in classifier: Class [0..1],   in inputs: ParameterValue [*])
EventDispatchLoop
+ objectActivation
0..1
+ classifierBehaviorInvocations
*
ClassifierBehaviorInvocationEventAccepter
+ invokeBehavior(in classifier: Class [1],   in inputs: ParameterValue [*])
+ terminate()
+ accept(in eventOccurrence: EventOccurrence)
+ match(in eventOccurrence: EventOccurrence): Boolean
EventAccepter
+ accept(in eventOccurrence: EventOccurrence)
+ match(in eventOccurrence: EventOccurrence): Boolean
+ waitingEventAccepters
*
0..1
0..1
+ eventPool
*
(fUML::Semantics::CommonBehaviors::Communications)
EventOccurrence

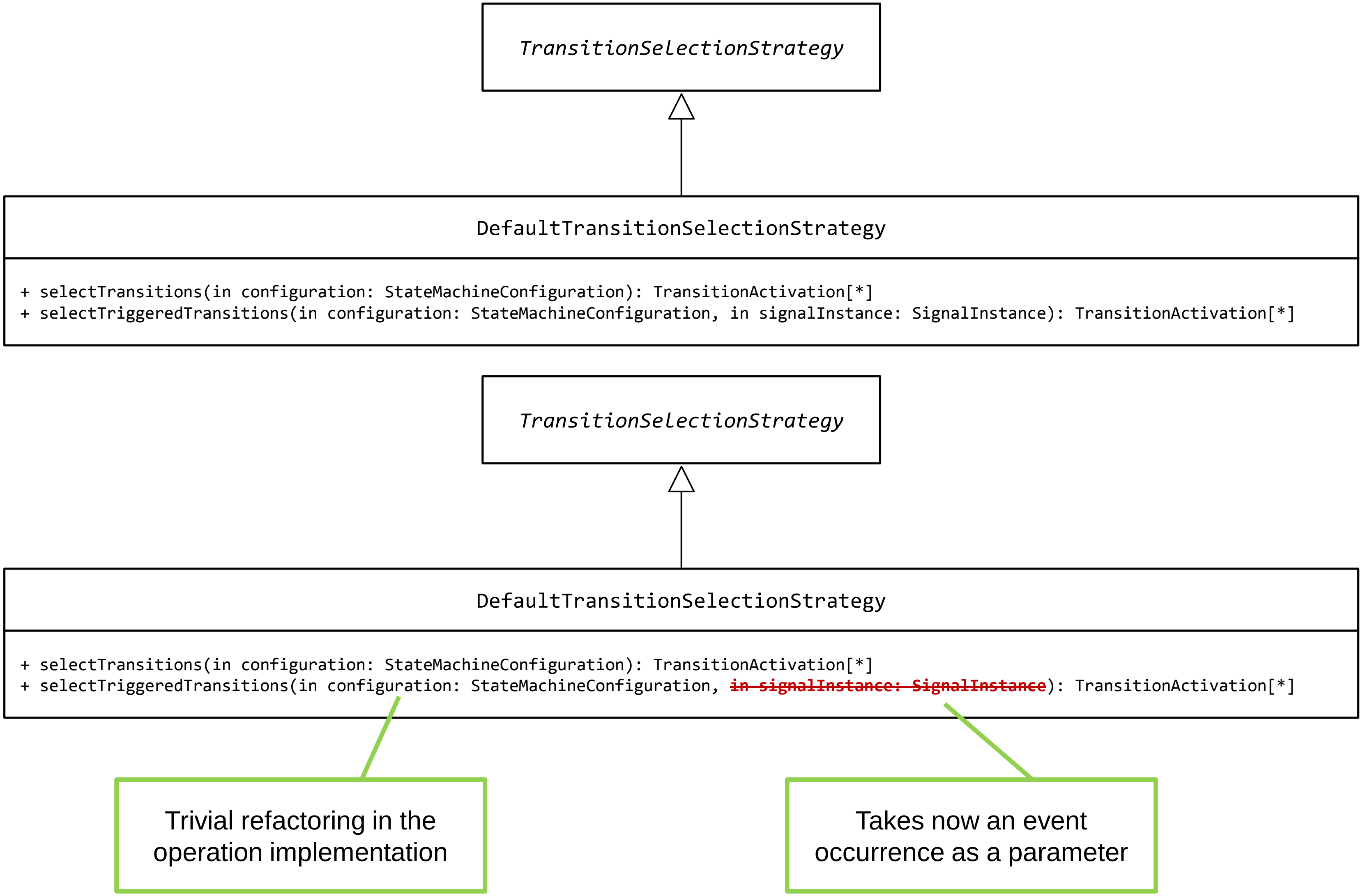
ClassifierBehaviorExecution class became a specialization of an event accepter. The classifier behavior starts when accept is executed

“getNextEvent” now provides EventOccurrence

The pool is now a list of EventOccurrence

PSSM ALIGNEMENT WITH FUML 1.2

VERY SMALL IMPACTS



TransitionSelectionStrategy

DefaultTransitionSelectionStrategy

```
+ selectTransitions(in configuration: StateMachineConfiguration): TransitionActivation[*]
+ selectTriggeredTransitions(in configuration: StateMachineConfiguration, in signalInstance: SignalInstance): TransitionActivation[*]
```

TransitionSelectionStrategy

DefaultTransitionSelectionStrategy

+ selectTransitions(in configuration: StateMachineConfiguration): TransitionActivation[*]

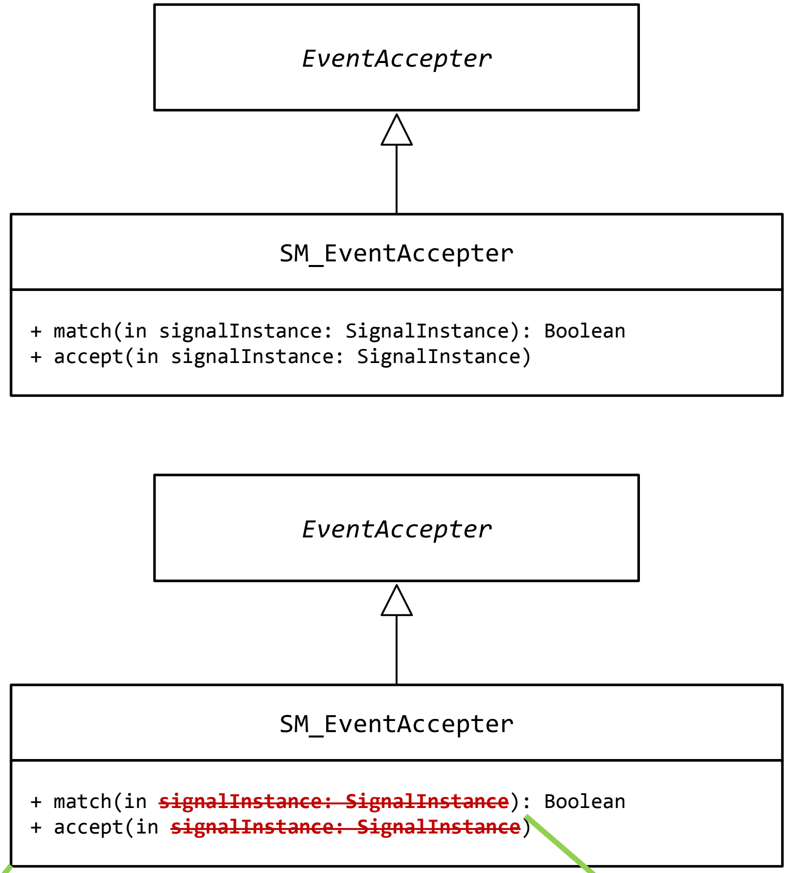
+ selectTriggeredTransitions(in configuration: StateMachineConfiguration, ~~in signalInstance: SignalInstance~~): TransitionActivation[*]

Trivial refactoring in the operation implementation

Takes now an event occurrence as a parameter

PSSM ALIGNEMENT WITH FUML 1.2

VERY SMALL IMPACTS



EventAccepter

SM_EventAccepter

```
+ match(in signalInstance: SignalInstance): Boolean
+ accept(in signalInstance: SignalInstance)
```

EventAccepter

SM_EventAccepter

+ match(in ~~signalInstance: SignalInstance~~): Boolean
+ accept(in ~~signalInstance: SignalInstance~~)

Trivial refactoring in the operations implementations

Take now an event occurrence as a parameter

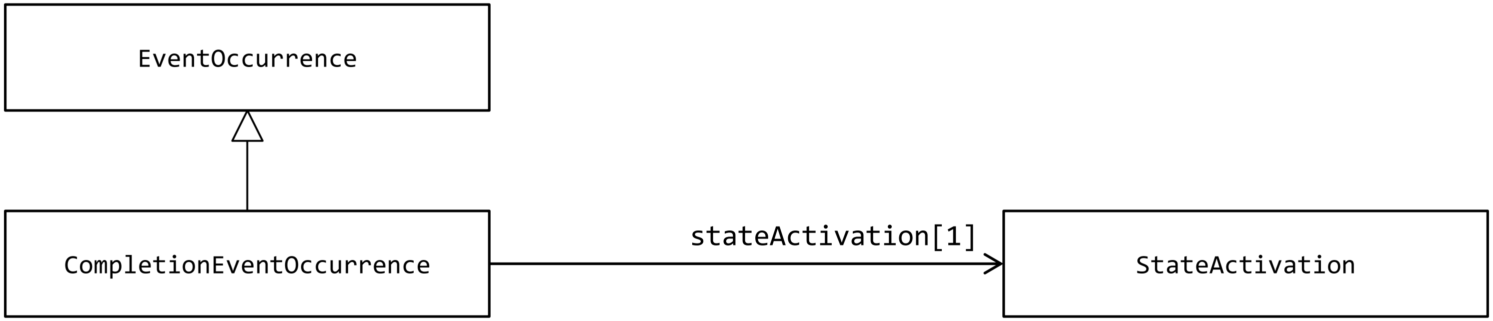
PSSM PROGRESSES

INTRODUCTION OF COMPLETION EVENT

Transition_015	In case of simple States, a completion event is generated when the associated entry and doActivity Behaviors have completed executing. (p.328)
Transition_016	If no such Behaviors are defined, the completion event is generated upon entry into the State. (p.328)
Transition_017	For composite States, a completion event is generated under the following circumstances: All internal activities (e.g., entry and doActivity Behaviors) have completed execution, and all its orthogonal Regions have reached a FinalState. (p.328 - 329)
Transition_019	Completion events have dispatching priority. That is, they are dispatched ahead of any pending Event occurrences in the event pool. (p.329)

PSSM PROGRESSES

CHANGES IN THE SEMANTIC MODEL



PSSM PROGRESSES

CHANGES IN THE SEMANTIC MODEL

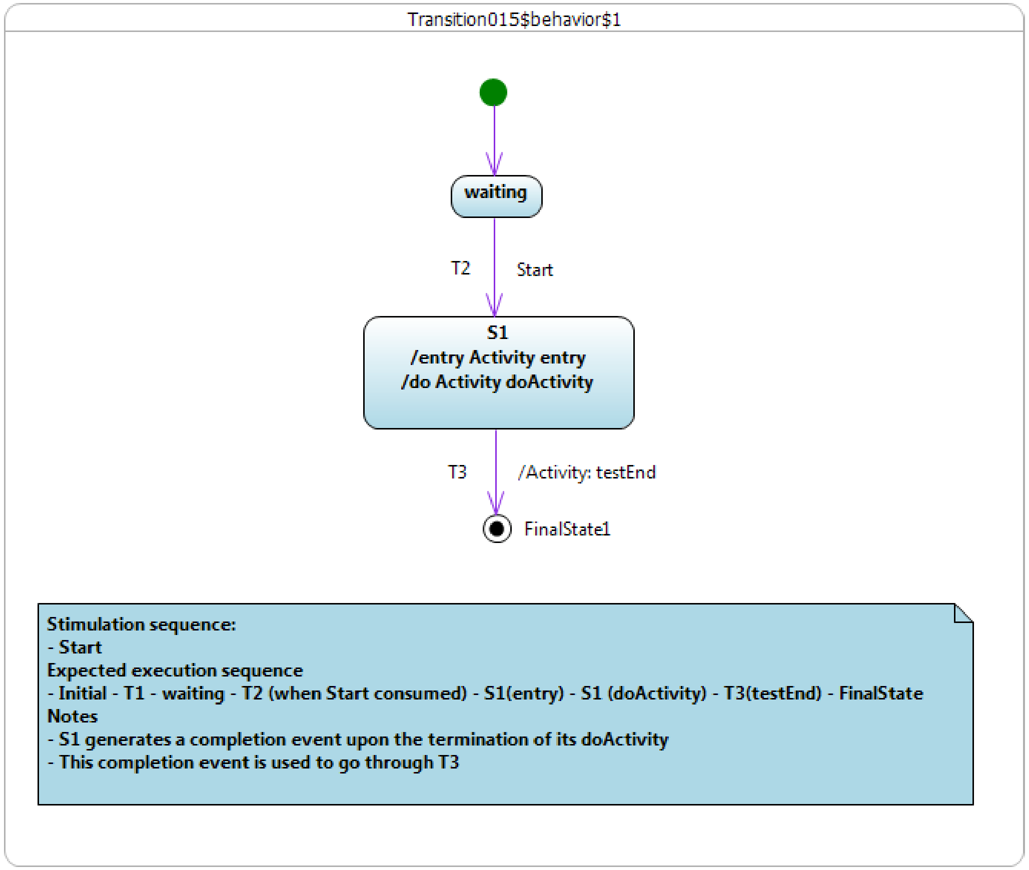
Completion events

- StateActivation
  - Boolean flags to know if the different prerequisites to generate the completion event are fulfilled
  - “notifyCompletion” operation was added
  - “hasCompleted” operation was added
- RegionActivation
  - Boolean flag added to know if the region finished through its final state
- FinalState
  - Completion test added (force the state to evaluate its completion when one of its region has reached its final state)
- StateMachineExecution
  - Additional loop in the “execution” operation was removed
- TransitionSelectionStrategy
  - Now only a single single operation “selectTransition” remains

PSSM TEST SUITE PROGRESSES

TEST CASE 1

Transition_015	In case of simple States, a completion event is generated when the associated entry and doActivity Behaviors have completed executing. (p.328)



Stimulation sequence:
- Start

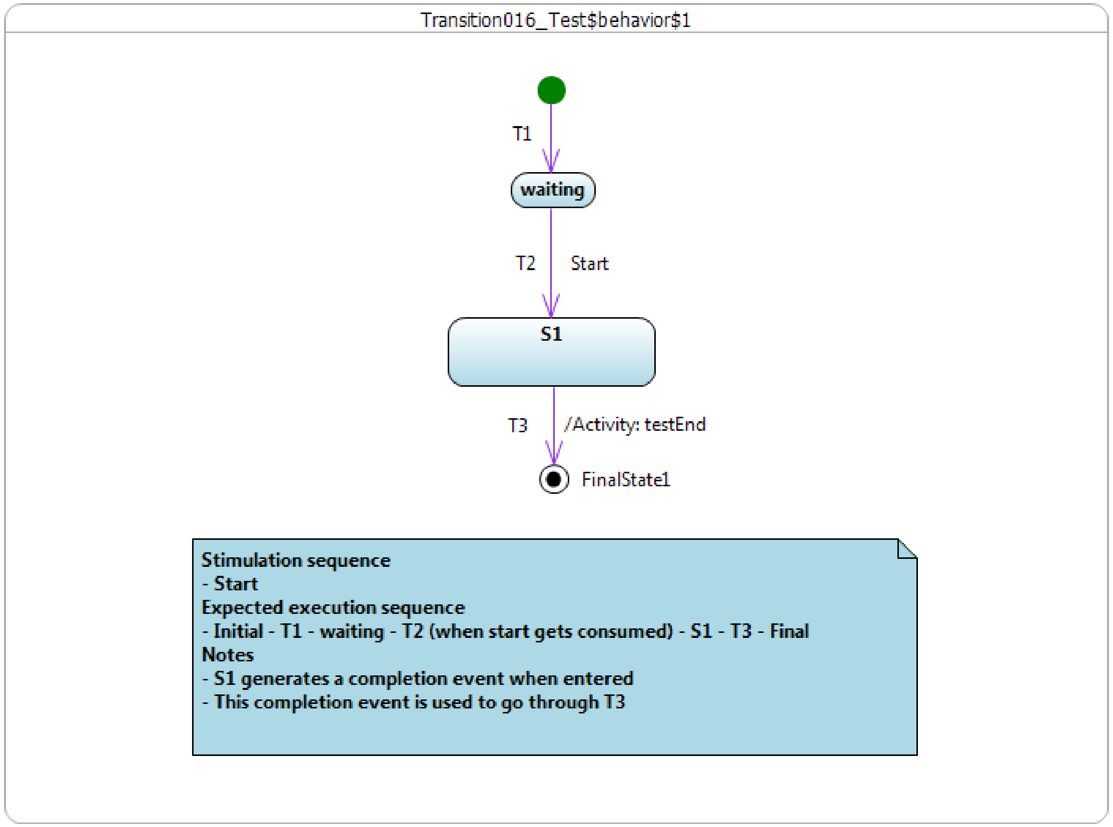
Expected execution sequence
- Initial - T1 - waiting - T2 (when Start consumed) - S1(entry) - S1 (doActivity) - T3(testEnd) - FinalState

Notes
- S1 generates a completion event upon the termination of its doActivity
- This completion event is used to go through T3

PSSM TEST SUITE PROGRESSES

TEST CASE 2

Transition_016	If no such Behaviors are defined, the completion event is generated upon entry into the State. (p.328)



Stimulation sequence
- Start

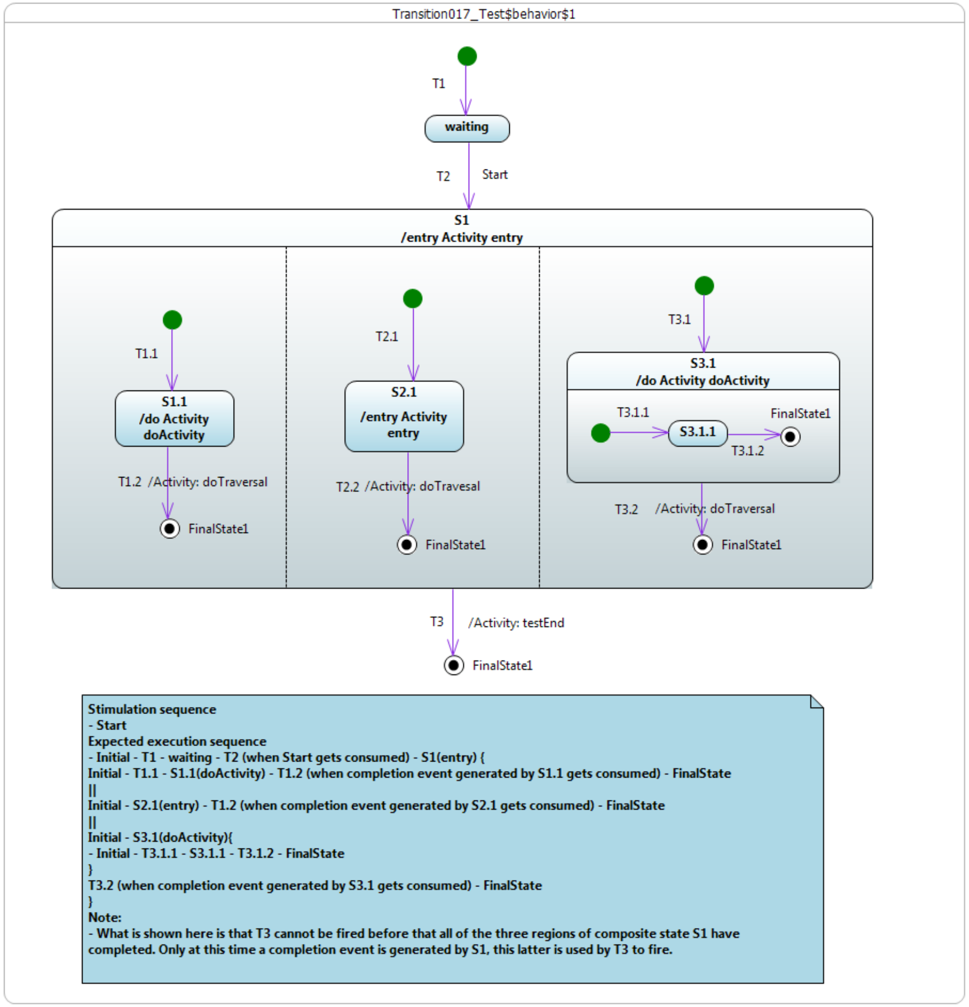
Expected execution sequence
- Initial - T1 - waiting - T2 (when start gets consumed) - S1 - T3 - Final

Notes
- S1 generates a completion event when entered
- This completion event is used to go through T3

PSSM TEST SUITE PROGRESSES

TEST CASE 3

Transition_017	For composite States, a completion event is generated under the following circumstances: All internal activities (e.g., entry and doActivity Behaviors) have completed execution, and all its orthogonal Regions have reached a FinalState. (p.328 - 329)



Stimulation sequence
- Start

Expected execution sequence
- Initial - T1 - waiting - T2 (when Start gets consumed) - S1(entry) {
Initial - T1.1 - S1.1(doActivity) - T1.2 (when completion event generated by S1.1 gets consumed) - FinalState
||
Initial - S2.1(entry) - T1.2 (when completion event generated by S2.1 gets consumed) - FinalState
||
Initial - S3.1(doActivity){
- Initial - T3.1.1 - S3.1.1 - T3.1.2 - FinalState
}
T3.2 (when completion event generated by S3.1 gets consumed) - FinalState
}

Note:
- What is shown here is that T3 cannot be fired before that all of the three regions of composite state S1 have completed. Only at this time a completion event is generated by S1, this latter is used by T3 to fire.