FROM RESEARCH TO INDUSTRY

cea tech

[PSSM] – ORLANDO OMG MEETING

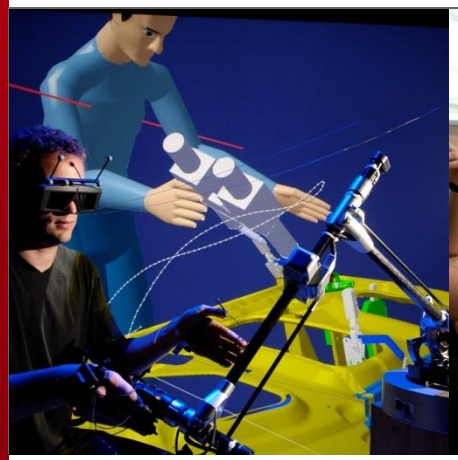
JUNE 23RD, 2016

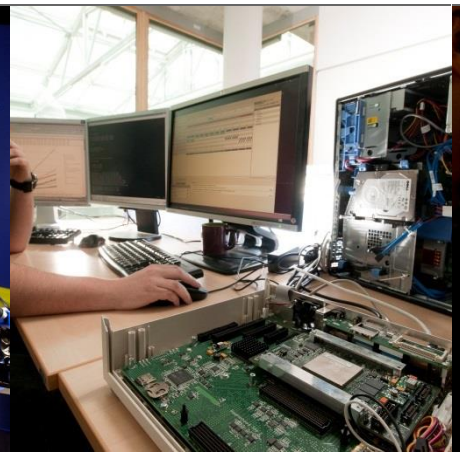
Jérémie TATIBOUET (CEA LIST)

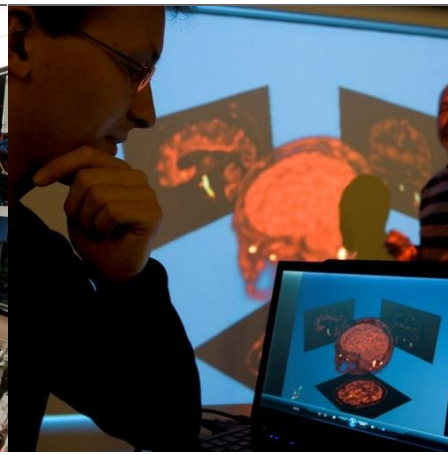
Arnaud CUCCURU (CEA LIST)

Sébastien GERARD (CEA LIST)

list







INSTITUT CARNOT CEA LIST digiteo

OUTLINE

1. Deferred events semantics
   - Support other dispatching strategies than FIFO
2. State machine configuration analysis
   - Transition priority (i.e., level of nesting) shall not affect other regions
3. Test suite refactoring
   - Define alternative execution traces for each test
4. Call event semantics
   - Integrate in the semantic model a support for call events
5. Event data passing
   - Data shipped by event occurrences must be available to behaviors
6. DoActivity
   - Integrate in the semantic model the resolution discussed in Reston
7. State machine redefinition
   - Integrate in the semantic model a support for redefined state-machines

DEFERRED EVENT SEMANTICS AND DISPATCHING STRATEGIES

DEFERRED EVENT SEMANTICS

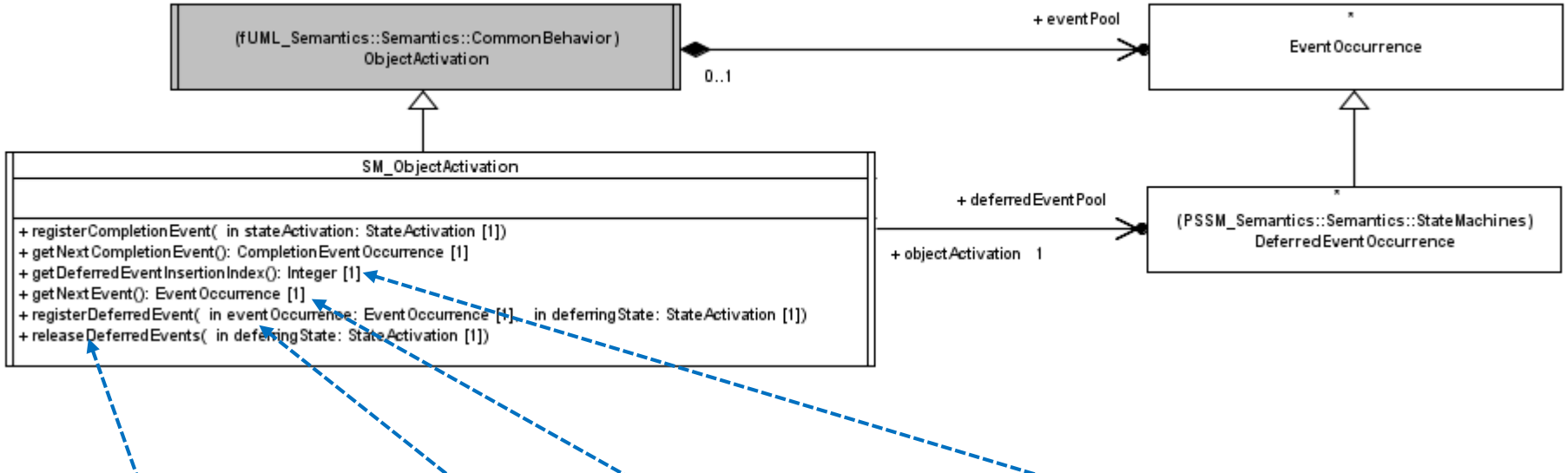
ALTERNATIVE EVENT DISPATCHING STRATEGY [1]

Problem

- Possibility to support alternative event dispatching strategy
  - e.g. LIFO, priority-based

Resolution

- Update to `SM_ObjectActivation`



Put back to the regular event pool all event occurrences deferred by the specified *StateActivation*

Add the specified event occurrence to the deferred event pool

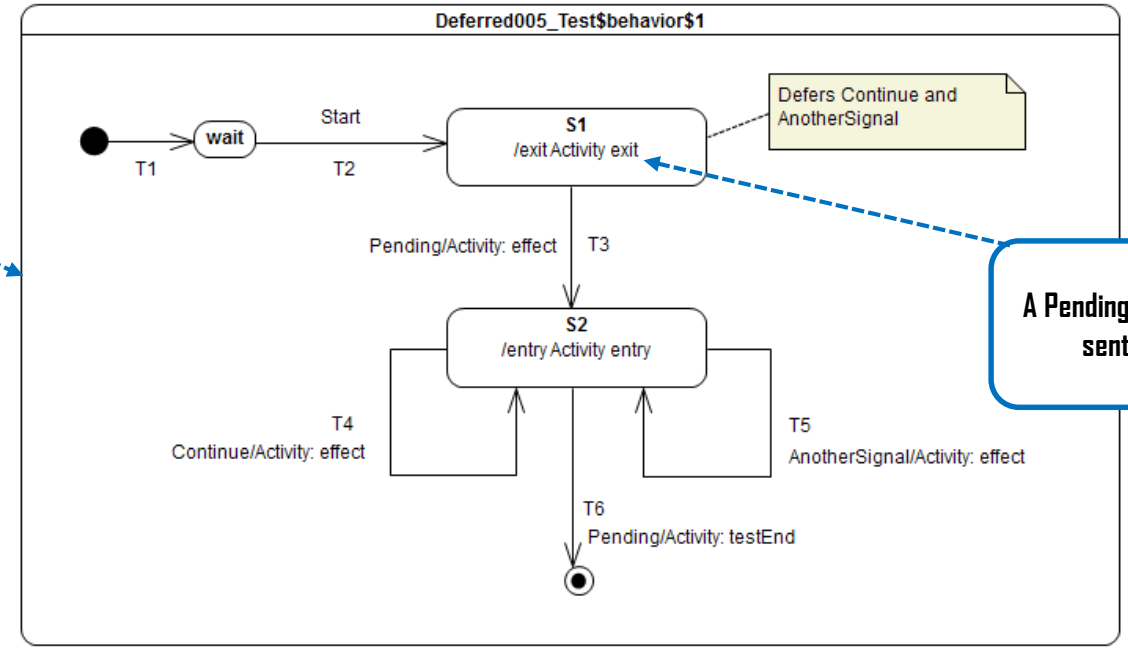
All completion events are dispatched first. If there is no completion event then regular events are dispatched following the chosen strategy. If the dispatched event is a deferred event then it is unwrapped and dispatched

Compute insertion index of released deferred events. Calculation based on presence of completion events

DEFERRED EVENT SEMANTICS

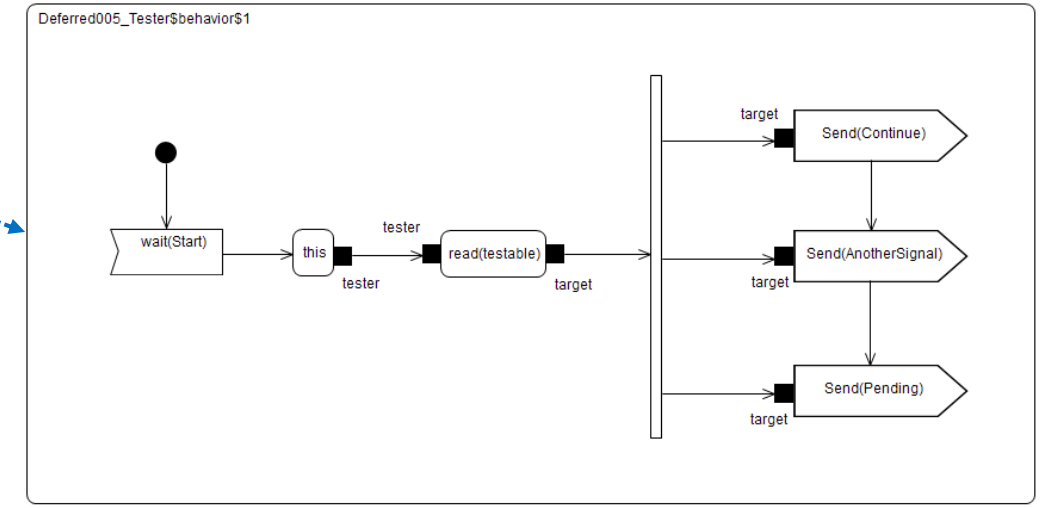
TEST EXTRACTED FROM THE TEST SUITE

Tested state-machine



A Pending signal event occurrence is sent to the context object.

Stimulation sequence



DEFERRED EVENT SEMANTICS

EXECUTION USING THE FIFO STRATEGY

	Event pool	Deferred event	Configuration	Transitions
1	[]	[]	[] – Initial step	[T1]
2	[Start, **CE(wait)**]	[]	[wait]	[]
3	[**Start**]	[]	[wait]	[T2]
4	[Continue, **CE(S1)**]	[]	[S1]	[]
5	[AnotherSignal, **Continue**]	[]	[S1]	[]
6	[**AnotherSignal**]	[Continue]	[S1]	[]
7	[**Pending**]	[Continue, AnotherSignal]	[S1]	[T3]
8	[Pending, AnotherSignal, Continue, **CE(S2)**]	[]	[S2]	[]
9	[Pending, AnotherSignal, **Continue**]		[S2]	[T4]
10	[Pending, AnotherSignal, **CE(S2)**]		[S2]	[]
11	[Pending, **AnotherSignal**]	[]	[S2]	[T5]
12	[Pending, **CE(S2)**]	[]	[S2]	[]
13	[**Pending**]	[]	[S2]	[T6]

Deferred events are placed back in the regular event pool

DEFERRED EVENT SEMANTICS

EXECUTION USING THE LIFO STRATEGY

	Event pool	Deferred event	Configuration	Transitions
1	[]	[]	[] – Initial step	[T1]
2	[Start, **CE(wait)**]	[]	[wait]	[]
3	[**Start**]	[]	[wait]	[T2]
4	[Continue, **CE(S1)**]	[]	[S1]	[]
5	[**AnotherSignal**, Continue]	[]	[S1]	[]
6	[**Continue**]	[AnotherSignal]	[S1]	[]
7	[**Pending**]	[AnotherSignal, Continue]	[S1]	[T3]
8	[Pending, AnotherSignal, Continue, **CE(S2)**]	[]	[S2]	[]
9	[**Pending**, AnotherSignal, Continue]	[]	[S2]	[T6]

Completion events still have priority

Second Pending signal event occurrence emitted during the execution of the S1 exit behavior is dispatched first due to the FIFO strategy. Therefore T6 is fired.

STATE MACHINE CONFIGURATION ANALYSIS

STATE MACHINE CONFIGURATION ANALYSIS

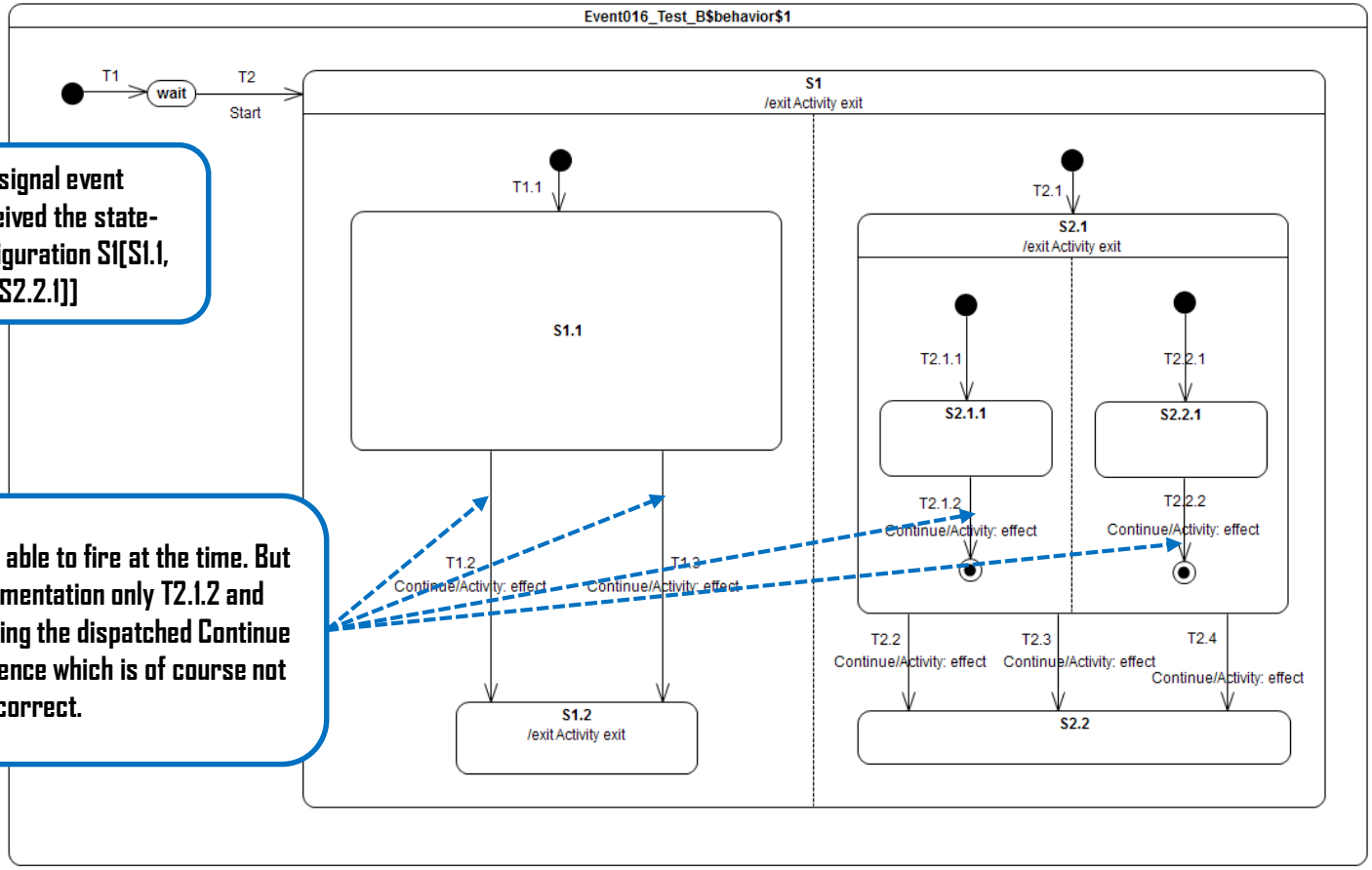
INCORRECT HANDLING OF TRANSITION PRIORITY

Problem

- In the presence of orthogonal regions:
  - If multiple transitions located in different regions and having different levels of nesting are able to fire using the same event occurrence, then only those with the deeper level of nesting are fired.
- Test Event 016 - B

When Continue signal event occurrence is received the state-machine is the configuration S1[S1.1, S2.1[S2.1.1, S2.2.1]]

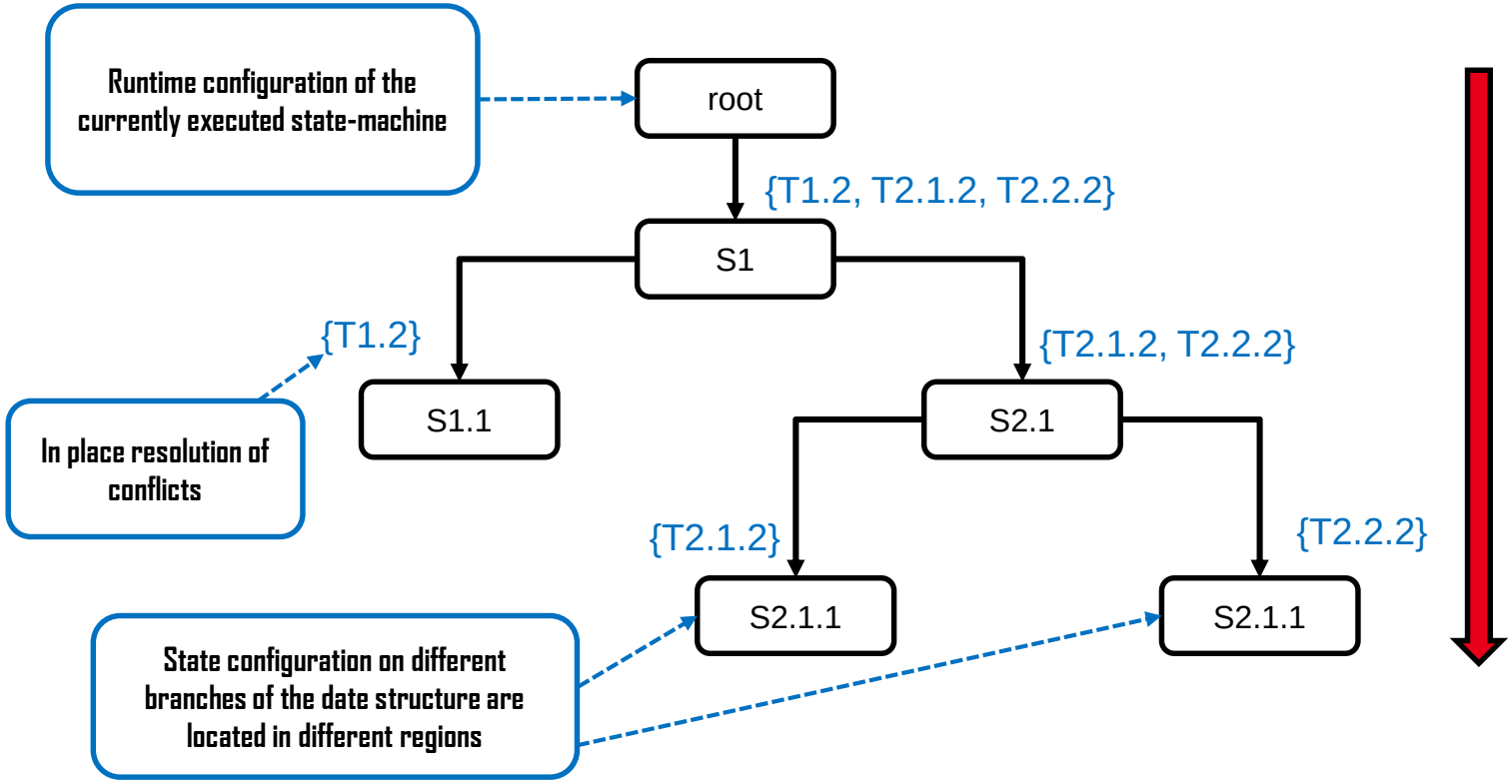
Four transitions are able to fire at the time. But in the initial implementation only T2.1.2 and T2.2.2 were fired using the dispatched Continue signal event occurrence which is of course not correct.



Resolution

- StateMachineConfiguration
  - Does not maintain any cartography making easier configuration analysis
- StateMachineEventAccepter
  - Move to a fully recursive algorithm to analyze the configuration.

- State machine configuration of Event016-B after 2nd RTC step



STATE MACHINE CONFIGURATION ANALYSIS

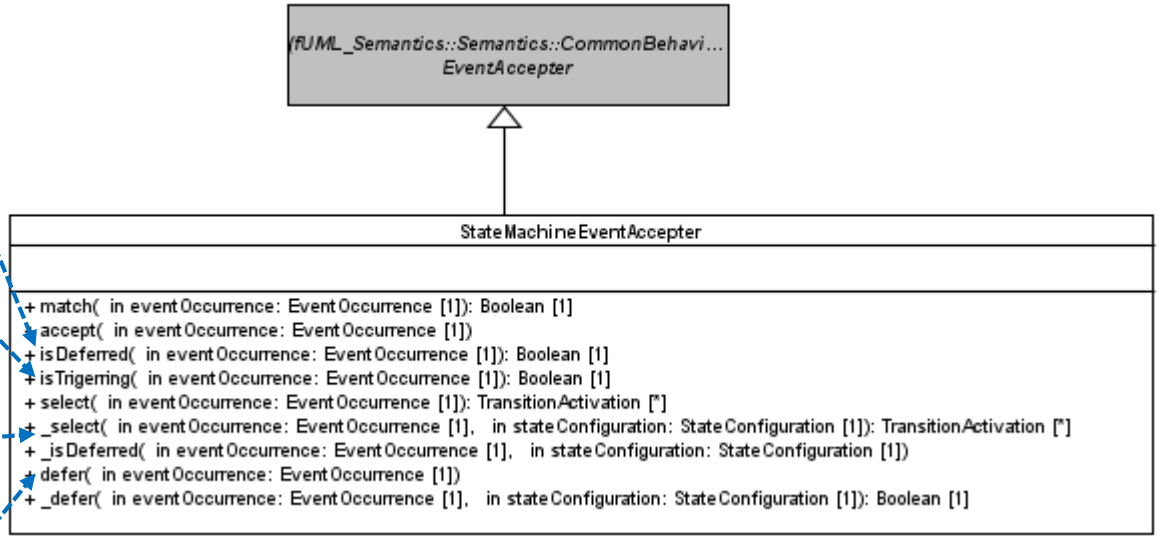
IMPACT ON THE PSSM SEMANTIC MODEL

Evaluates if in the context of the current state-machine configuration the event occurrence will be deferred

Evaluates if in the context of the current state-machine configuration the event occurrence triggers at least a single transition

Select all transitions available to fire using the give event occurrence

Realize the deferral of the given occurrence



Other changes

- StateMachineConfiguration
  - Attribute "`cartography`" was removed and all statements updating this attributes have been removed.
- DefaultTransitionSelectionStrategy
  - Merged into StateMachineEventAccepter

STATE MACHINE CONFIGURATION ANALYSIS

RTC STEPS NOW REALIZED WHEN EXECUTING EVENT 016-B

	Event pool	Deferred event	Configuration	Transitions
1	[]	[]	[] – Initial step	[T1]
2	[Start, **CE(wait)**]	[]	[wait]	[]
3	[**Start**]	[]	[wait]	[T2(T1.1, T2.1(T2.1.1, T2.2.1))]
4	[CE(S2.2.1), CE(S2.1.1), **CE(S1.1)**]	[]	[S1[S1.1, S2.1[S2.1.1, S2.2.1]]]	[]
5	[Continue, CE(S2.2.1), **CE(S2.1.1)**]	[]	[S1[S1.1, S2.1[S2.1.1, S2.2.1]]]	[]
6	[Continue, **CE(S2.2.1)**]	[]	[S1[S1.1, S2.1[S2.1.1, S2.2.1]]]	[]
7	[**Continue**]	[]	[S1[S1.1, S2.1[S2.1.1, S2.2.1]]]	[T1.3, T2.1.2, T2.2.2]
8	[Continue, CE(S1.2), **CE(S2.1)**]	[]	[S1[S1.2, S2.1]]	[]
9	[Continue, **CE(S1.2)**]	[]	[S1[S1.2, S2.1]]	[]
10	[**Continue**]	[]	[S1[S1.2, S2.1]]	[T2.3]
11	[Continue, **CE(S2.2)**]	[]	[S1[S1.2, S2.2]]	[]
12	[**Continue**]	[]	[S1[S1.2, S2.2]]	[T3]

Transition set firing for the same event occurrence is now correct

TEST SUITE REFINEMENT

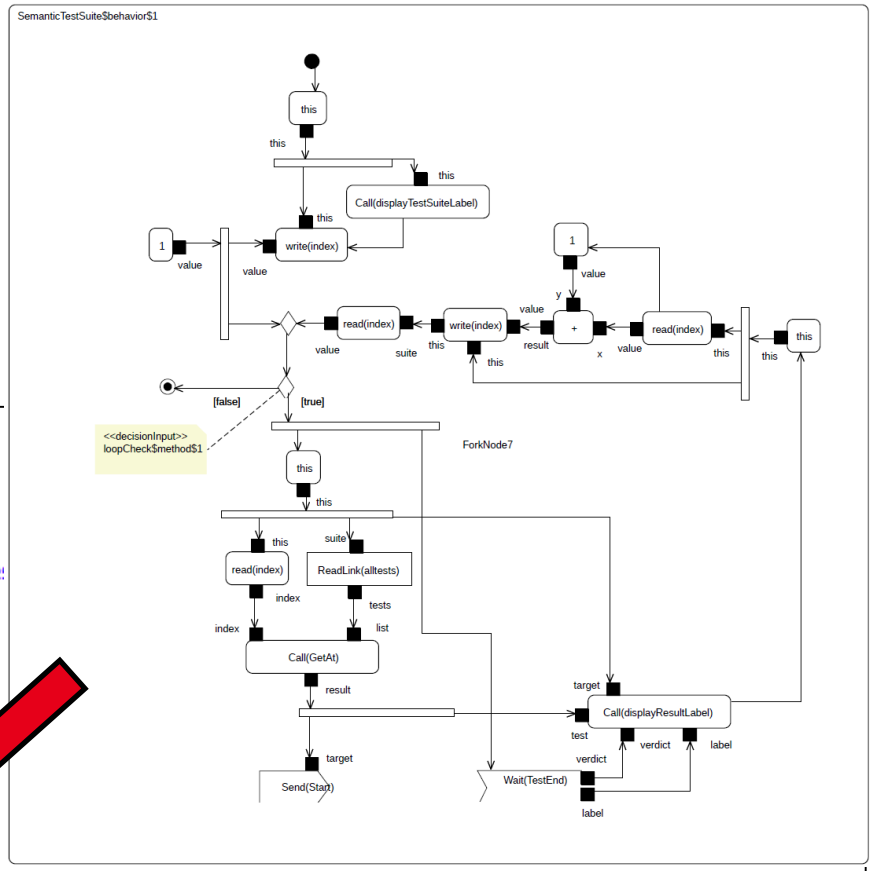
SPECIFY ALTERNATIVE EXECUTION TRACES FOR THE REQUIRED TEST CASES

TEST SUITE

SEMANTIC TEST SUITE

The semantic test suite is now formalized using ALF which makes it a lot more readable.

1. It is notified to Start by the semantic test
2. It iteratively start the different test registered and compute the global verdict.
3. Results are displayed on the chosen output channel

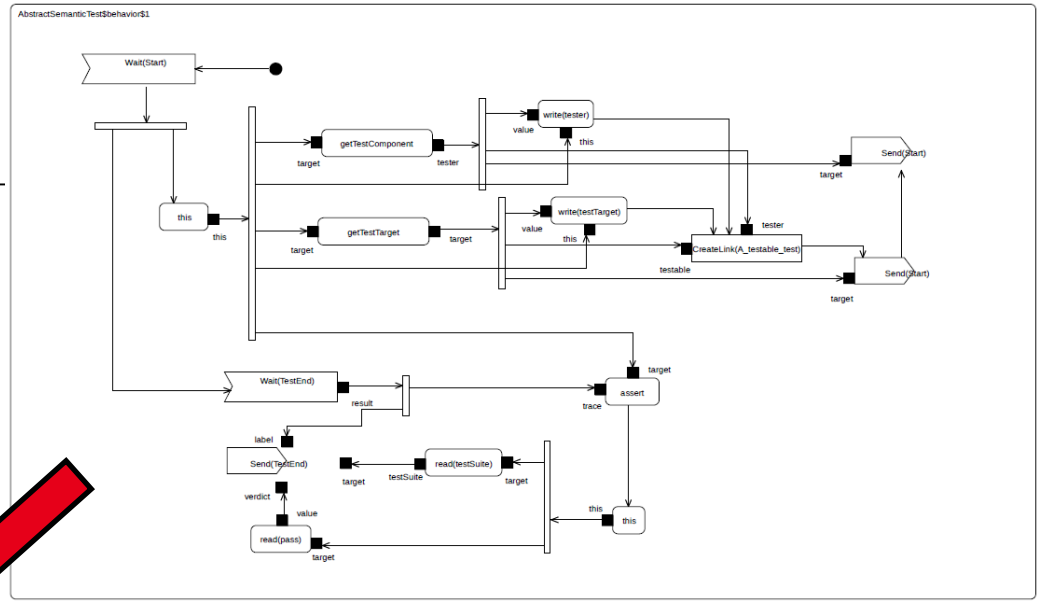



```
private import Util::Protocol::Messages::TestEnd;
private import Util::Protocol::Messages::Start;

activity 'SemanticTestSuite$behavior$1'() {
  accept(Start);
  WriteLine("\n[TEST SUITE ("+IntegerFunctions::ToString(this.tests->size())+" te
  // Execute all semantic tests registered in that test suite
  Integer failures = 0;
  for(i in 1..this.tests->size(){
    // Test gets the authorization to start
    this.tests->at(i).Start();
    // Test suite waits for the results
    accept(testResult : TestEnd){
      if(testResult.verdict == false){
        failures++;
      }
    }
  }
  // There is at least one failure then the test suite is considered as failed
  if(failures > 0){
    WriteLine("\n[TEST SUITE] - "+this.name + " FAILURE ("+IntegerFunctions::ToString(failures)+" / "+IntegerFunctions::ToString(this.tests->size())+" failed)\n");
  }
}
```

TEST SUITE

SEMANTIC TEST BEHAVIOR IN ALF


```
private import Util::Protocol::Messages::Start;
private import Util::Protocol::Messages::End;

activity 'AbstractSemanticTest$behavior$1'() {
  // Wait for a start signal
  accept(Start);
  // Tester and target are created and started
  target = this.getTestTarget();
  tester = this.getTestComponent();
  // Create link connection between the two
  target.tester = tester;
  tester.testable = target;
  this.target = target;
  this.tester = tester;
  // Both are started
  target.Start();
  tester.Start();
  // Wait for the arrival of the result emitted by the target
  accept(executionResult : End);
  this.pass = this.matches(executionResult.trace);
  // Result analysis
  if(!this.pass){
      WriteLine("[TEST] "+ this.name + " ** FAIL ** - {"+executionResult.trace + "} does not match any of the follo
      for(i in 1..this.expectedTraces->size()){
        WriteLine("-----> [" + IntegerFunctions::ToString(i)+"] - "+ this.expectedTraces->at(i));
      }
  }else{
      WriteLine("[TEST] " + this.name + " - PASS -");
  }
  // Provides the result back to the test suite governing this semantic te
  this.testSuite.TestEnd(this.pass);
}
```




```
abstract active class SemanticTest {

  public pass: Boolean;
  public name: String;
  public expectedTraces: String[1..*];
  public abstract getTestComponent(): Tester;
  public abstract getTestTarget(): Target;
  @Create
  public SemanticTest();
  @Destroy
  public destroy();
  public matches(in trace: String): Boolean;
  public register(in possibleTrace: String);
  public receive Util::Protocol::Messages::End;
  public receive Util::Protocol::Messages::Start;

} do 'AbstractSemanticTest$behavior$1'
```

Allow a semantic test to specify a set of expected traces.

TESTS REFINED WITH ALTERNATIVE TRACES

Test suite

- Transition
  - Transition 017 – (test simplified)
  - Transition 019
- Event
  - Event 016
- Entering
  - Entering 011
  - Entering 010
- Exiting
  - Exiting 001
  - Exiting 003
- Exit
  - Exit 002
- Entry
  - Entry 002 – A
  - Entry 002 – B
- Standalone
  - Standalone 001

TESTS REFINED WITH ALTERNATIVE TRACES

Test suite

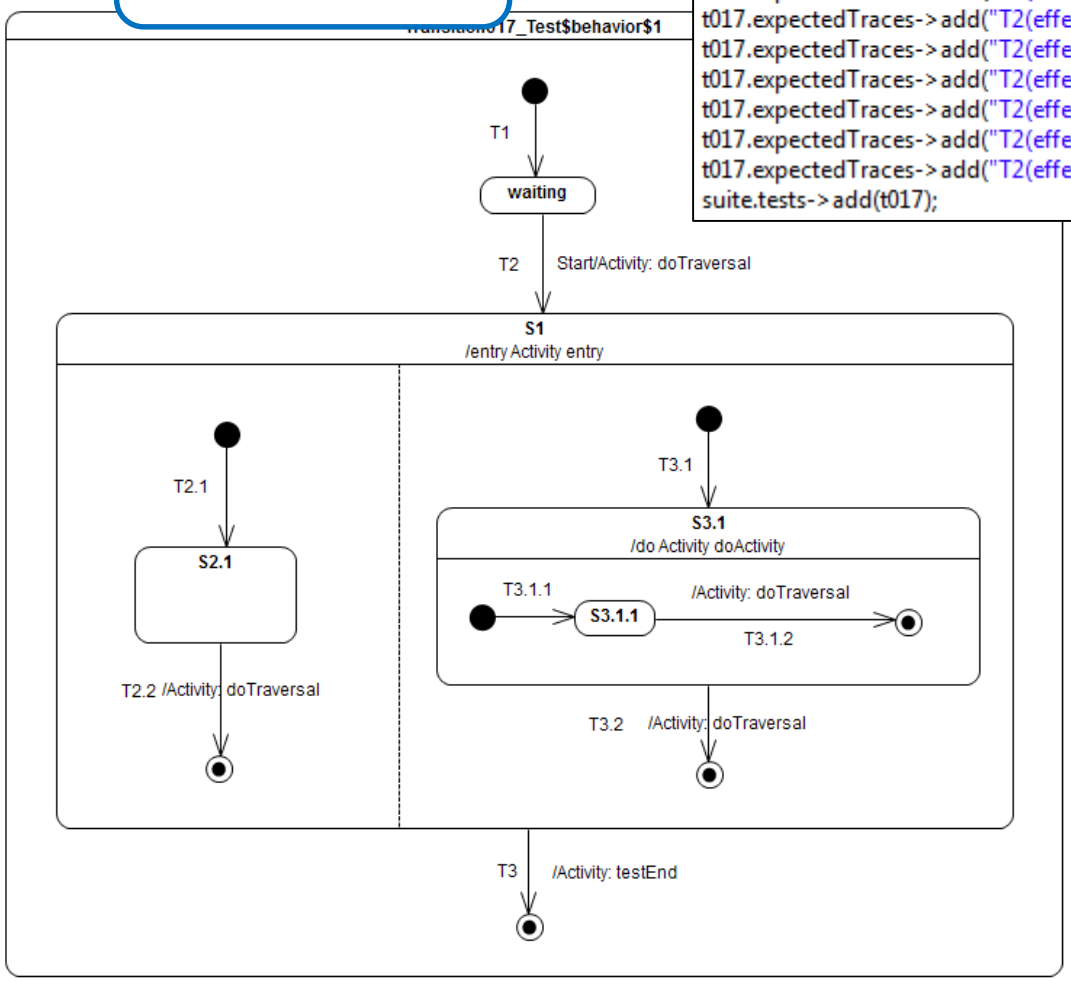
- Fork
  - Fork 001
  - Fork 002
- Join
  - Join 001
  - Join 002
- Terminate
  - Terminate 001
  - Terminate 002

TRANSITION 017

EXCERPT FROM THE TEST SUITE

Set of all valid traces that can be produced with respect to UML semantics

```
/*Transition017*/
t017 = new 'Transition017_SemanticTest'();
t017.name = "Transition 017";
t017.expectedTraces->add("T2(effect)::S1(entry)::T2.2(effect)::S3.1(doActivity)::T3.1.2(effect)::T3.2(effect)");
t017.expectedTraces->add("T2(effect)::S1(entry)::T2.2(effect)::T3.1.2(effect)::S3.1(doActivity)::T3.2(effect)");
t017.expectedTraces->add("T2(effect)::S1(entry)::T2.2(effect)::T3.2(effect)::S3.1(doActivity)::T3.1.2(effect)");
t017.expectedTraces->add("T2(effect)::S1(entry)::T2.2(effect)::T3.2(effect)::T3.1.2(effect)::S3.1(doActivity)");
t017.expectedTraces->add("T2(effect)::S1(entry)::S3.1(doActivity)::T2.2(effect)::T3.1.2(effect)::T3.2(effect)");
t017.expectedTraces->add("T2(effect)::S1(entry)::S3.1(doActivity)::T3.1.2(effect)::T2.2(effect)::T3.2(effect)");
t017.expectedTraces->add("T2(effect)::S1(entry)::T3.1.2(effect)::T2.2(effect)::S3.1(doActivity)::T3.2(effect)");
t017.expectedTraces->add("T2(effect)::S1(entry)::T3.1.2(effect)::S3.1(doActivity)::T2.2(effect)::T3.2(effect)");
suite.tests->add(t017);
```



T3.1.2 can fire first if S3.1.1 registered its completion first.

The doActivity invoked from S3.1 can complete before the end of the second RTC step.

T2.2 can fire first if S2.1 registers its completion event first.

BUML CONFORMANCE

CHANGES REQUIRED IN THE IMPLEMENTATION TO CONFORM TO BUML

BUML CONFORMANCE

REQUIRED IMPLEMENTATION CHANGES

Goal

- Make sure our execution model is implemented using the java textual surface notation for activities.
  - The execution model is an fUML model which can be executed using fUML semantics.

Implementation analysis

- Excel file recording most (all) of the changes that need to apply to have a implementation conforming to bUML.
- Typical changes
  - Usage of `for(<var> : <collections>)` instead of for loop using an index (see clause A.4.12 of fUML 1.2.1)
  - Usage of iterative for loop (see clause A.4.12 of fUML 1.2.1) where a parallel loop (using iterator) must be used (see clause A.5.6 of fUML 1.2.1).
  - Usage of while loop with an index starting from 0 instead of 1.
  - Usage of while loop with postfix increment to the index. This shall be replaced by the pattern `<index> = <index> + 1`.
  - Usage of constructors with parameters (see clause A.5.5)

CALL EVENT SEMANTICS

STATE MACHINE CONFIGURATION ANALYSIS

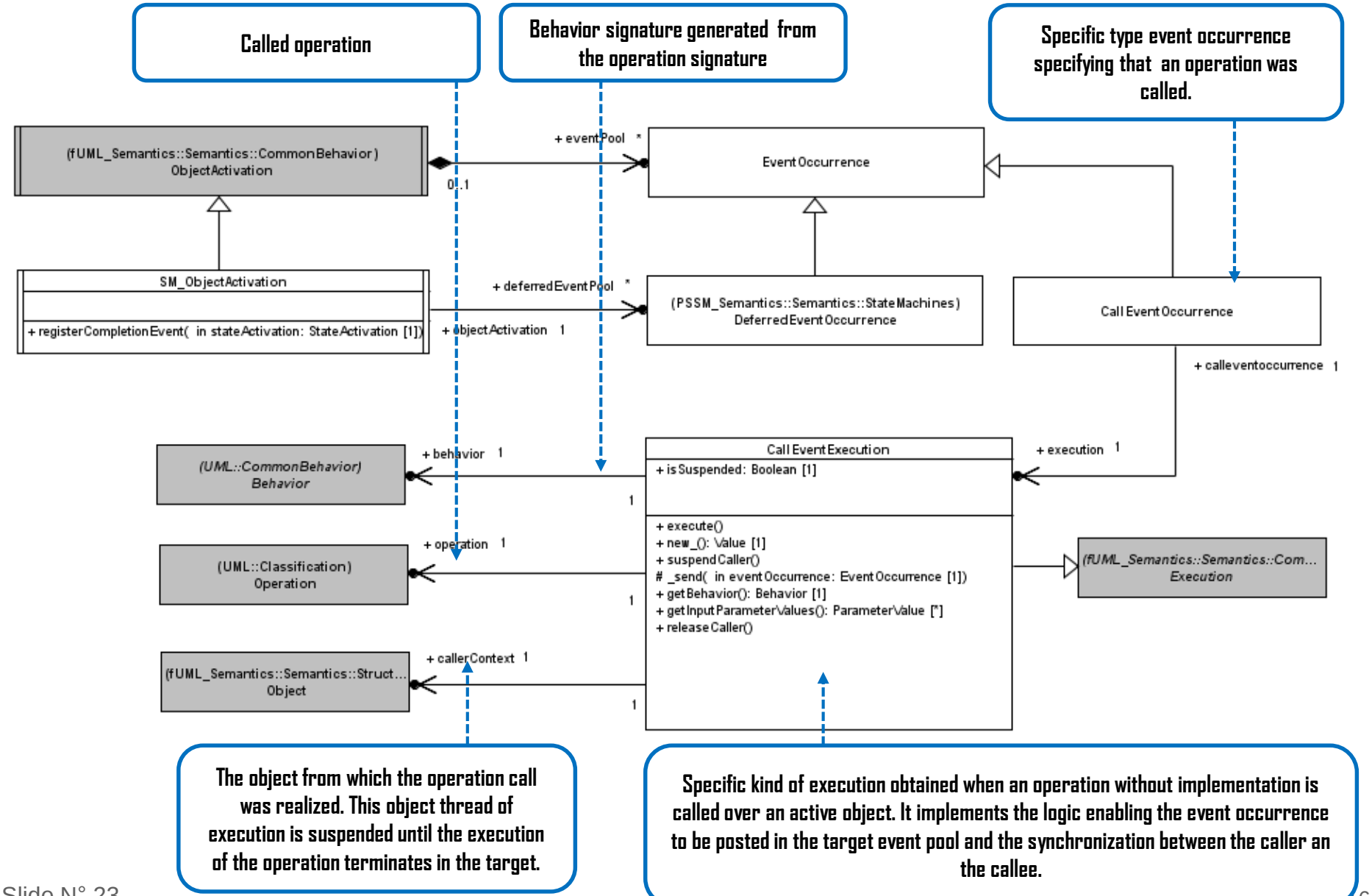
WHAT SAYS THE RFP

Precise semantics

- Transition (except for redefinition), including completion transitions (with no triggers) and transitions with triggers for the following kinds of events:
  - CallEvent (for ***synchronous*** calls) ✗
  - SignalEvent OK
- Agenda : April 2016

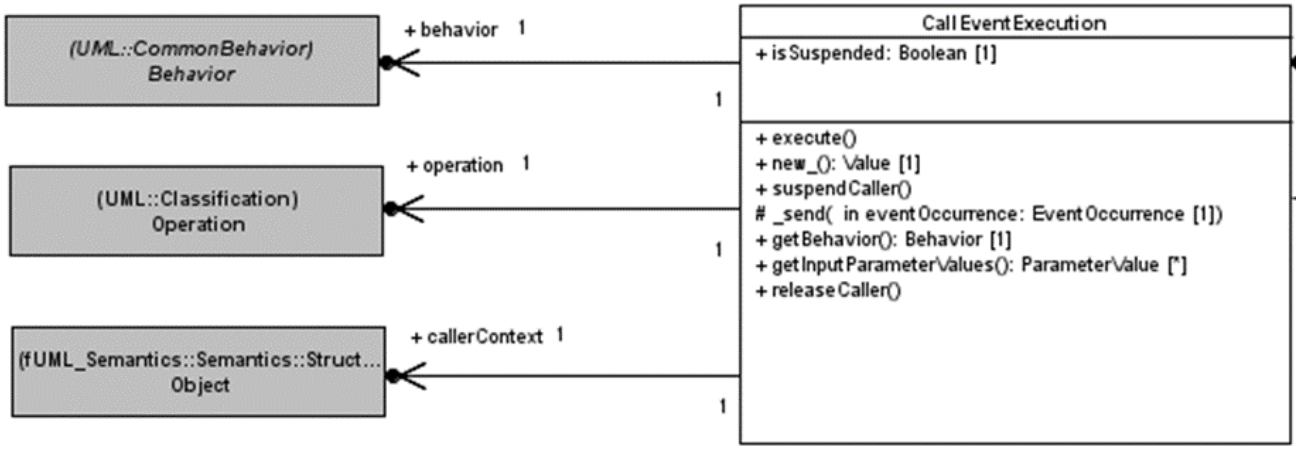
SEMANTIC MODEL CHANGES

CALL EVENT EXECUTION AND CALL EVENT OCCURRENCE



SEMANTIC MODEL CHANGES

CALL EVENT EXECUTION

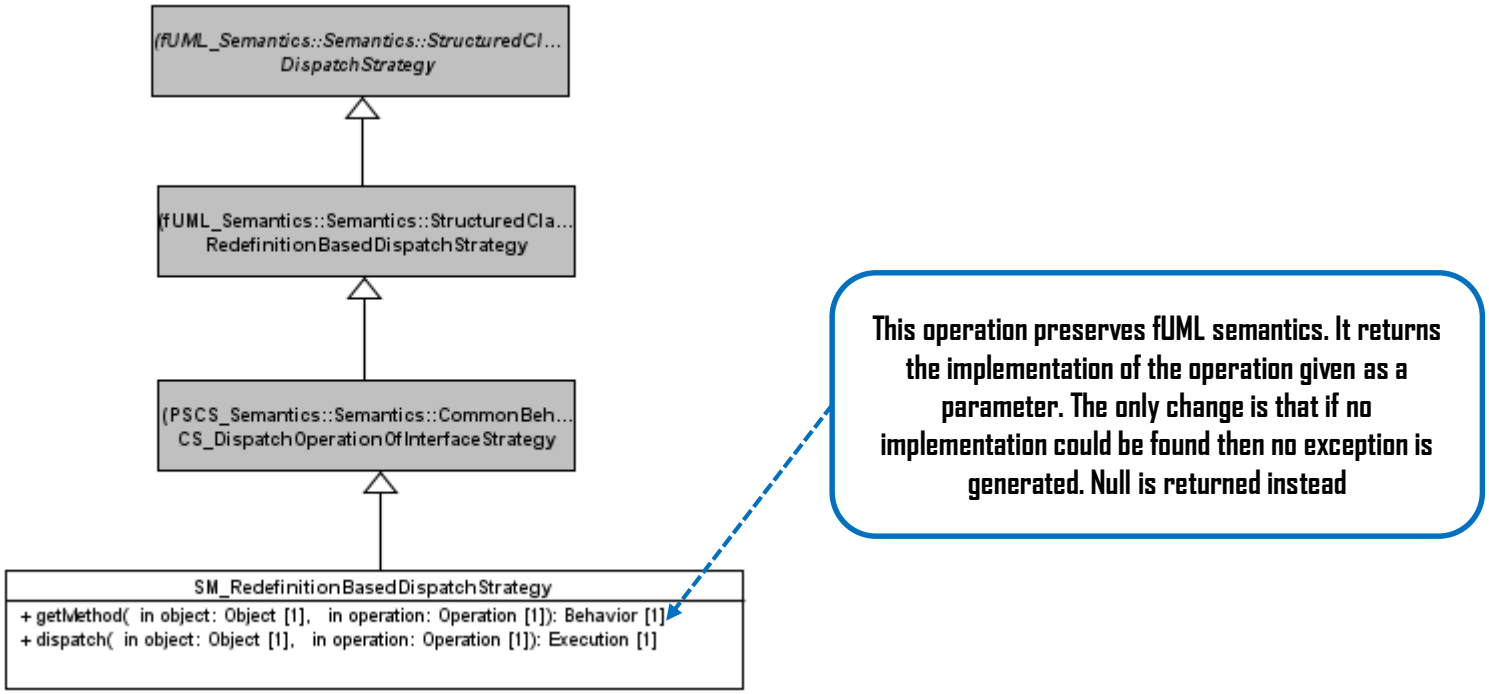


Other changes

- `execute()`
  - Send the call event occurrence to the target object
  - Suspend the caller until the target notified the end of the call
- `suspendCaller()`
  - The object from which the event was sent is suspended thanks to a spin loop (needs to be improve – usage of mutex)
- `releaseCaller()`
  - The object from which the event was sent can resume its execution

SEMANTIC MODEL CHANGES

OPERATION DISPATCHING

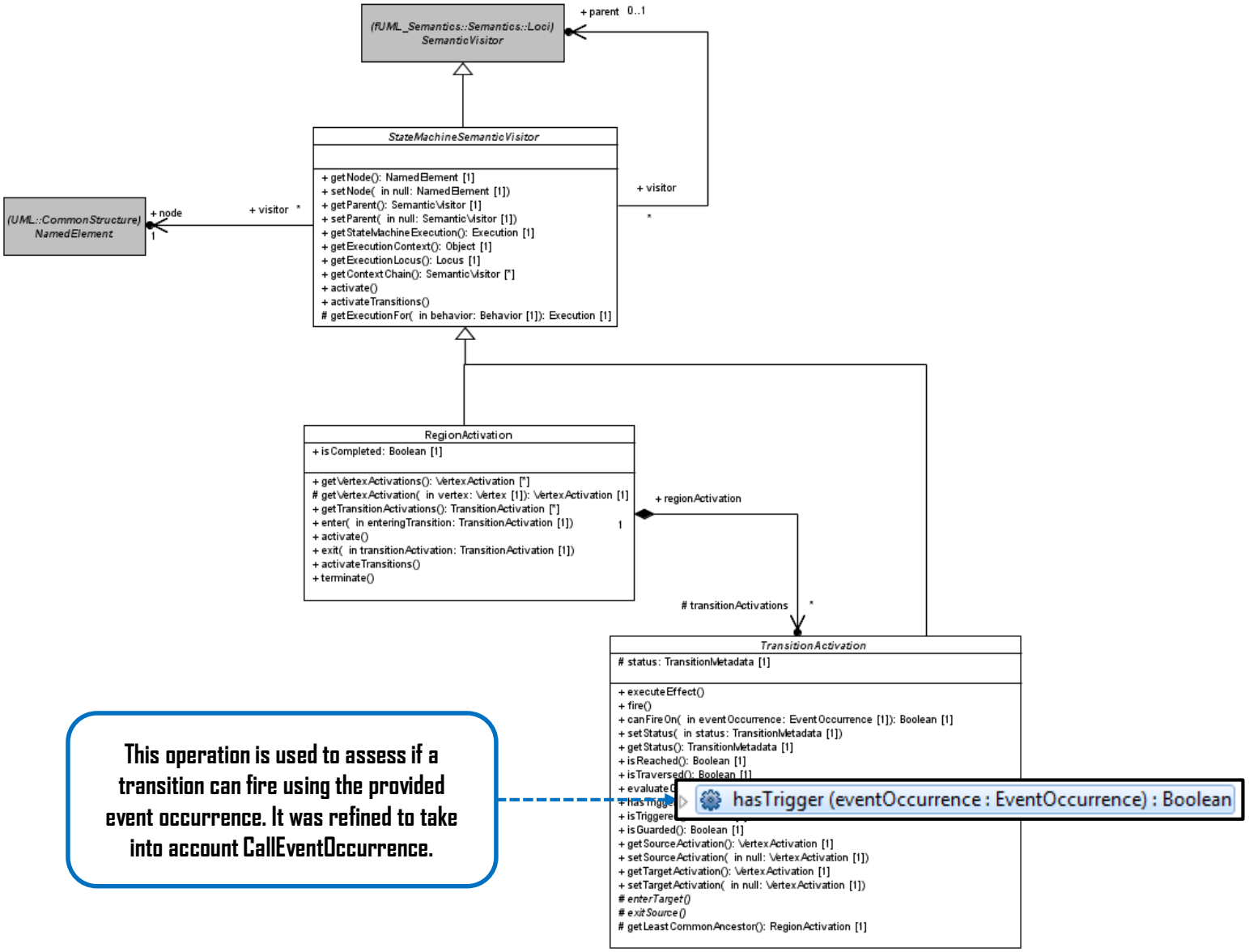


SM_RedefinitionBasedDispatchStrategy

- `dispatch(…): Execution`
  - If an implementation (i.e. a behavior) could be found then the appropriate execution is instantiated.
  - If no implementation could be found then a `CallEventExecution` is created.

SEMANTIC MODEL CHANGES

ENABLE ACCEPTANCE OF CALL EVENT



This operation is used to assess if a transition can fire using the provided event occurrence. It was refined to take into account CallEventOccurrence.

SEMANTIC MODEL CHANGES

ENABLE ACCEPTANCE OF CALL EVENT

`hasTrigger(in EventOccurrence): Boolean`

- Does it exist a trigger for this transition which matches the provided event occurrence.
- Matching rules:
  - If the trigger is for a Signal then the provided event occurrence must be a `SignalEventOccurrence` referencing a `SignalInstance` whose type is the signal referenced by the trigger.
  - If the trigger is for a Operation call then the provided event occurrence must be a `CallEventOccurrence` referencing an operation which must be the operation referenced by the trigger.
  - If the trigger specifies the `onPort` property then the provided event occurrence can only be a `SignalEventOccurrence`. This limitation is introduced by the fact that PSCS does not provide a `CS_EventOccurrence` related to a particular `CS_InteractionPoint`.

SEMANTIC MODEL CHANGES

DETECT END OF AN RTC STEP

A. Problem

- At some point of the execution the caller (i.e. the object from which the call event was emitted) needs to be resumed.

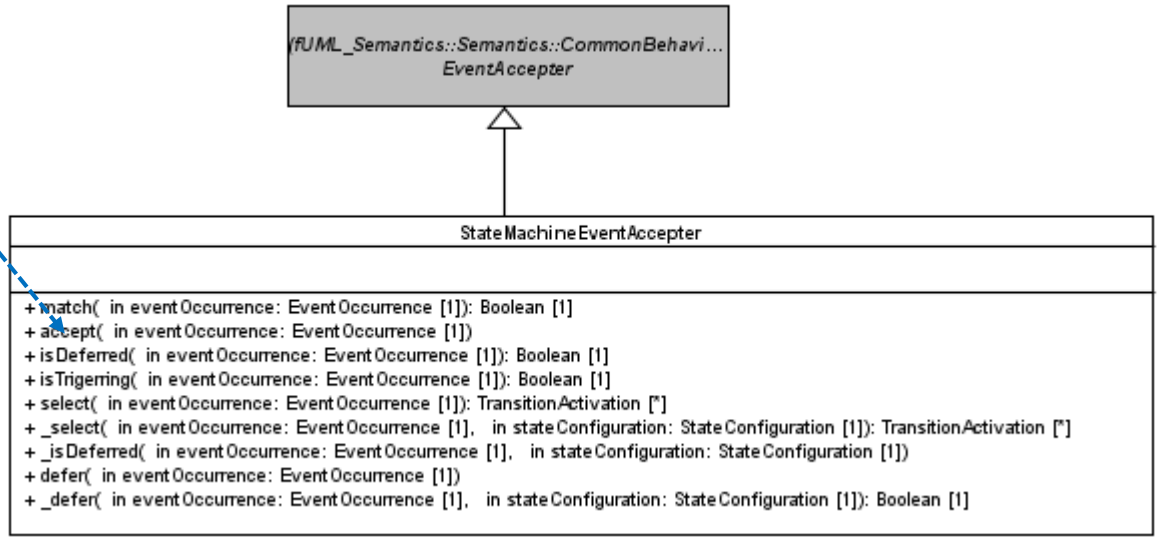
B. Proposal

- The caller must be resumed when the RTC step in which the CallEventOccurrence is dispatched terminates.

C. Implementation

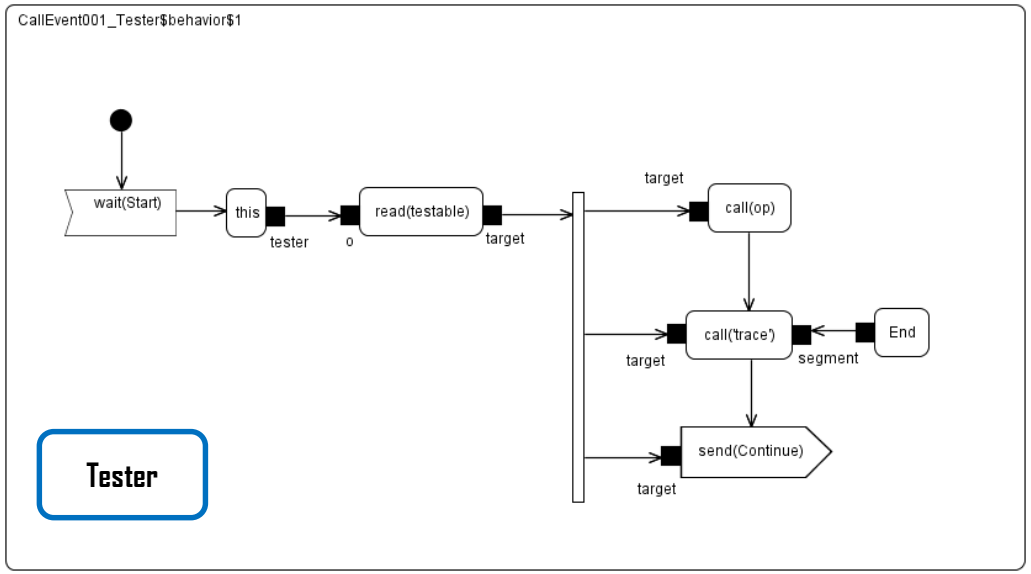
- Update `accept` operation of the state-machine event accepter

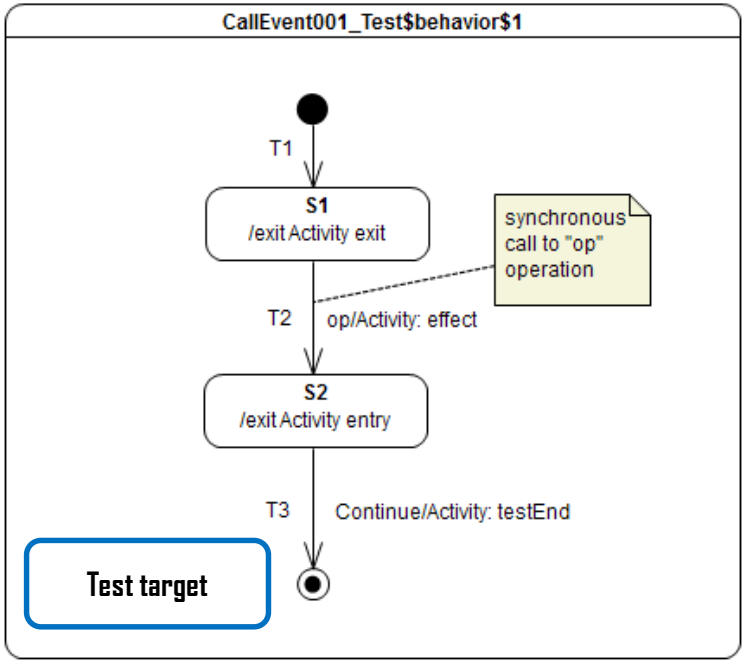
After the concurrent firing of the transitions selected for this step, if the dispatched event occurrence is a CallEventOccurrence then the CallEventExecution referenced by this event occurrence is used to notify the caller that it can resume its execution.

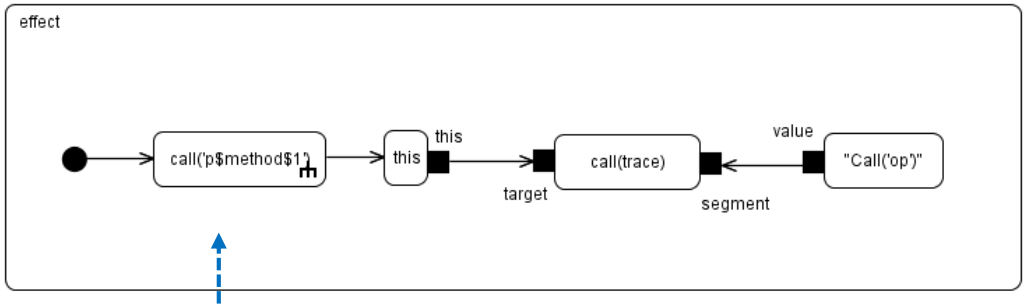


TESTING – CALL EVENT 001

EXCERPT FROM THE TEST SUITE








```
/*Test call event 001 */
ce001 = new 'CallEvent001_SemanticTest'();
ce001.name = "CallEvent 001";
ce001.expectedTraces->add("S1(exit)::Call(op)::End::S2(entry)");
suite.tests->add(ce001);
```

Implementation of the effect of transition T2. It call the behavior "p$method$1" which implements the operation behavior.

Expected execution trace.

STATE MACHINE CONFIGURATION ANALYSIS

RTC STEPS NOW REALIZED WHEN EXECUTING EVENT 016-B

	Event pool	Trace evolution	Configuration	Transitions
1	[]	[]	[] – Initial step	[T1]
2	[**CE(S1)**]	[]	[S1]	[]
3	[**Call(op)**]	[S1(exit)::call(op)]]	[S1]	[T2]
4	[**CE(S2)**]	[S1(exit)::call(op)::End]	[S2]	[]
5	[**Continue**]	[S1(exit)::call(op)::End::S2(entry)]	[S2]	[T3]

EVENT DATA PASSING

EVENT DATA PASSING

WHAT SAYS THE RFP

Precise semantics

- Proposals shall define how data associated with event occurrences shall be accessed by transition guards and passed to transition effect behaviors and state behaviors during the process of event dispatching and transition triggering. ✖
- Agenda: May 2016

Semantics

- Signal Event Occurrence
  - If the behavior has no parameter, the signal event occurrence is not passed to this behavior.
  - If the behavior has one parameter, then the signal instance embedded by the signal event occurrence is passed to the behavior
- Call Event Occurrence
  - If a Behavior has parameters, then the values of the input Parameters of for the call are passed into the Behavior Execution as the values of the corresponding input Parameters of the Behavior.
  - If an effect, entry or exit Behavior is not just *input-conforming*, then the values of its output Parameters are passed out of its Behavior Execution on its completion as *potential* values for the output Parameters of the called Operation.

Refactoring

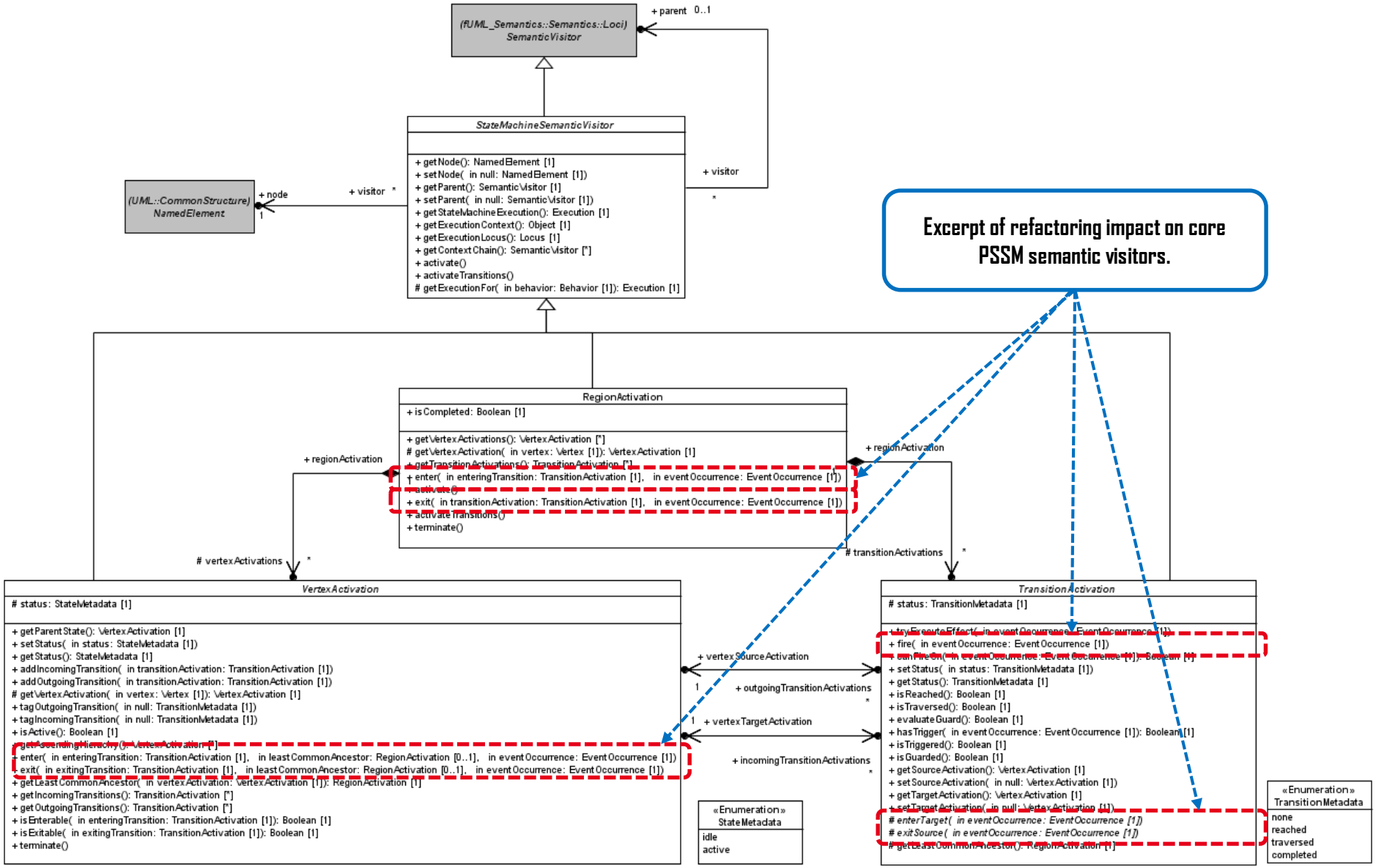
- Proposal
  - Enable the dispatched event occurrence to be accessible by the different visitors involved in a RTC step.

Impact on the semantic model

- VertexActivation
  - enter(in TransitionActivation, in RegionActivation, in EventOccurrence)
  - exit(in TransitionActivation, in RegionActivation, in EventOccurrence)
  - Applies to all specializations of VertexActivation and recursively
- StateActivation
  - tryExecuteEntry(in eventOccurrence)
  - tryInvokeDoActivity(in eventOccurrence)
  - tryExecuteExit(in eventOccurrence)
- TransitionActivation
  - fire(in EventOccurrence)
  - tryExecuteEffect(in eventOccurrence)
  - enterTarget(in eventOccurrence)
  - exitSource(in eventOccurrence)
- RegionActivation
  - enter(in TransitionActivation, in eventOccurrence)
  - exit(in TransitionActivation, in eventOccurrence)

PROPOSAL

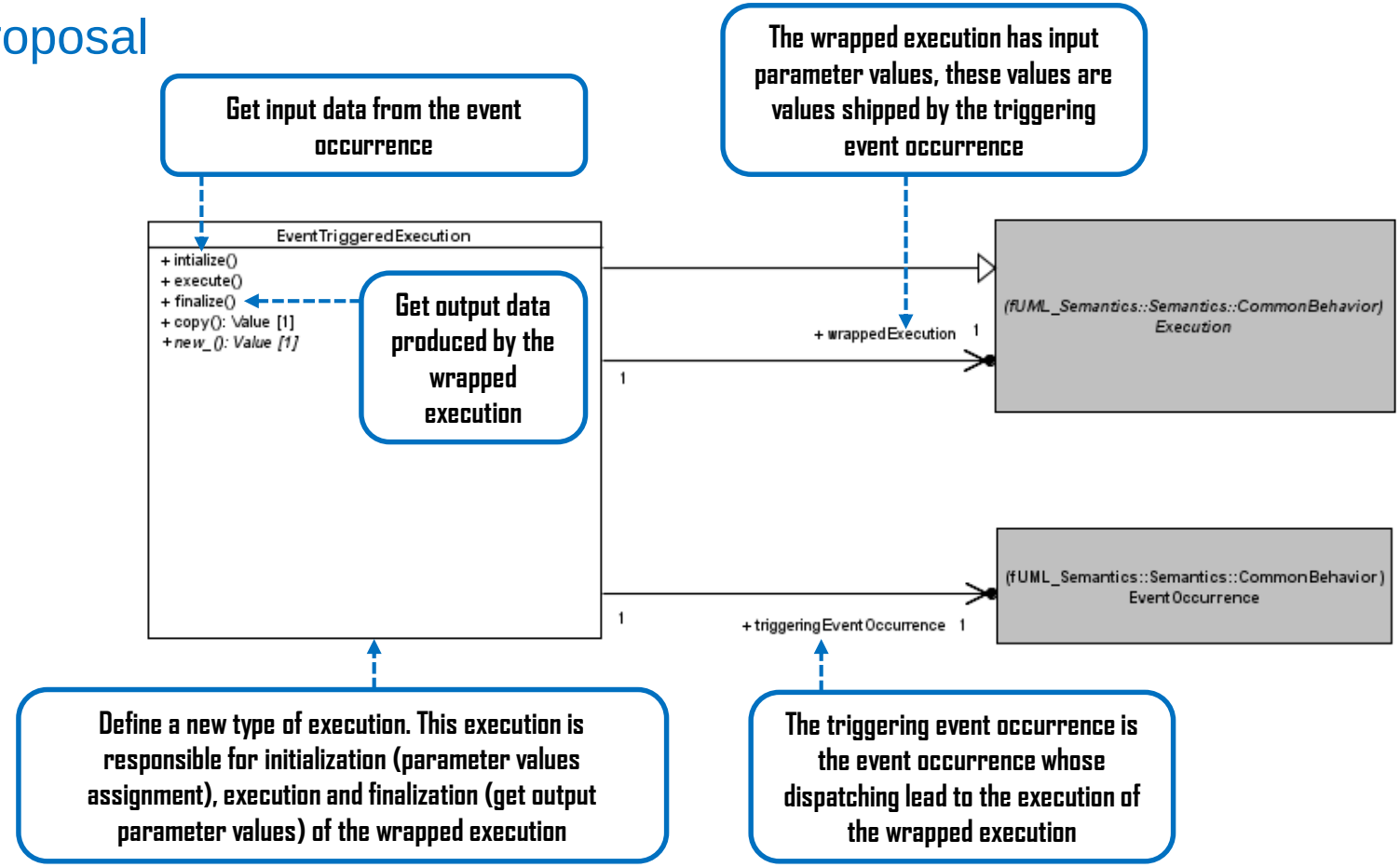
GENERAL REFACTORING TO VERTEX ACTIVATION AND ITS SPECIALIZATIONS



Objective

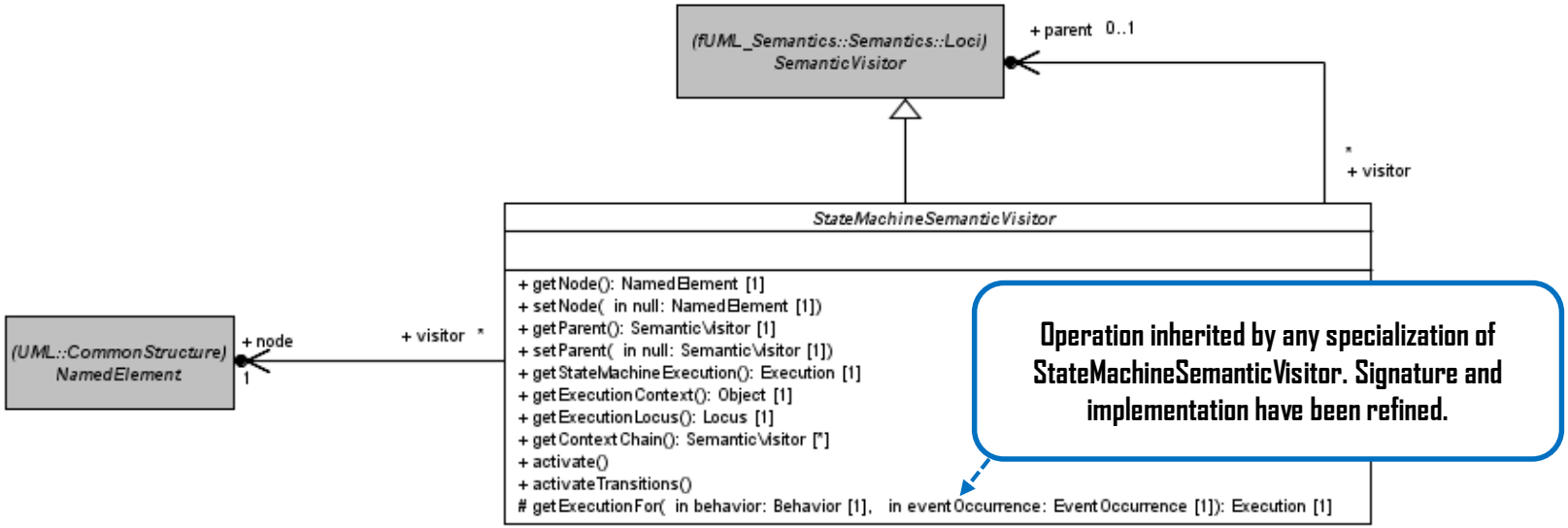
- Exploit data available in event occurrence and limit the impact of this feature implementation over the semantic model.

Proposal



PROPOSAL

HOW THIS NEW EXECUTION GET INSTANTIATED



StateMachineSemanticVisitor

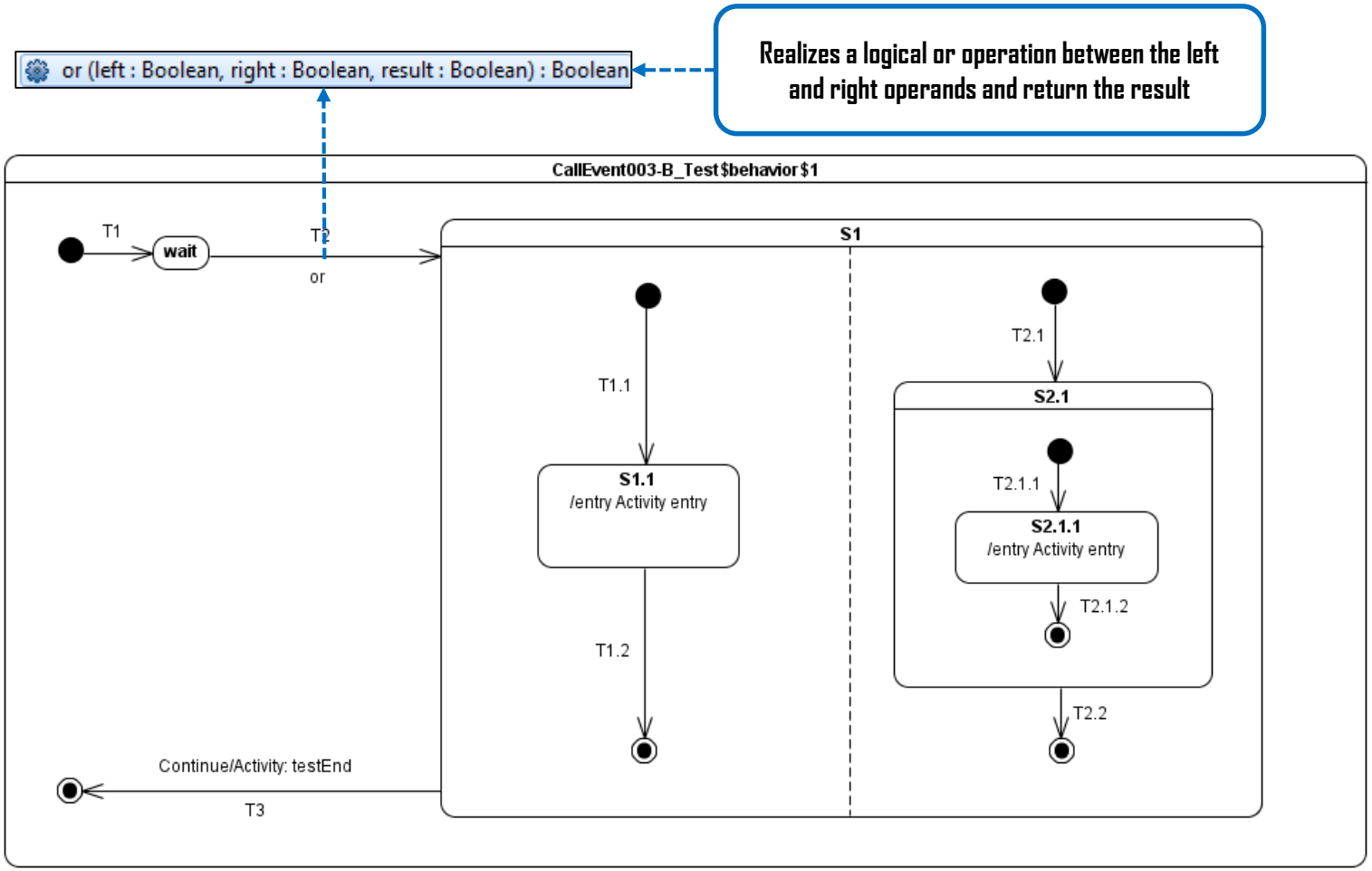
- `getExecutionFor(in Behavior, in EventOccurrence): Execution`

  - If there is no event occurrence specified (i.e. it is null) then a regular execution (i.e. not an `EventTriggeredExecution`) is obtained.

  - If there is an event occurrence specified the an `EventTriggeredExecution` is obtained. This execution wraps the regular execution.

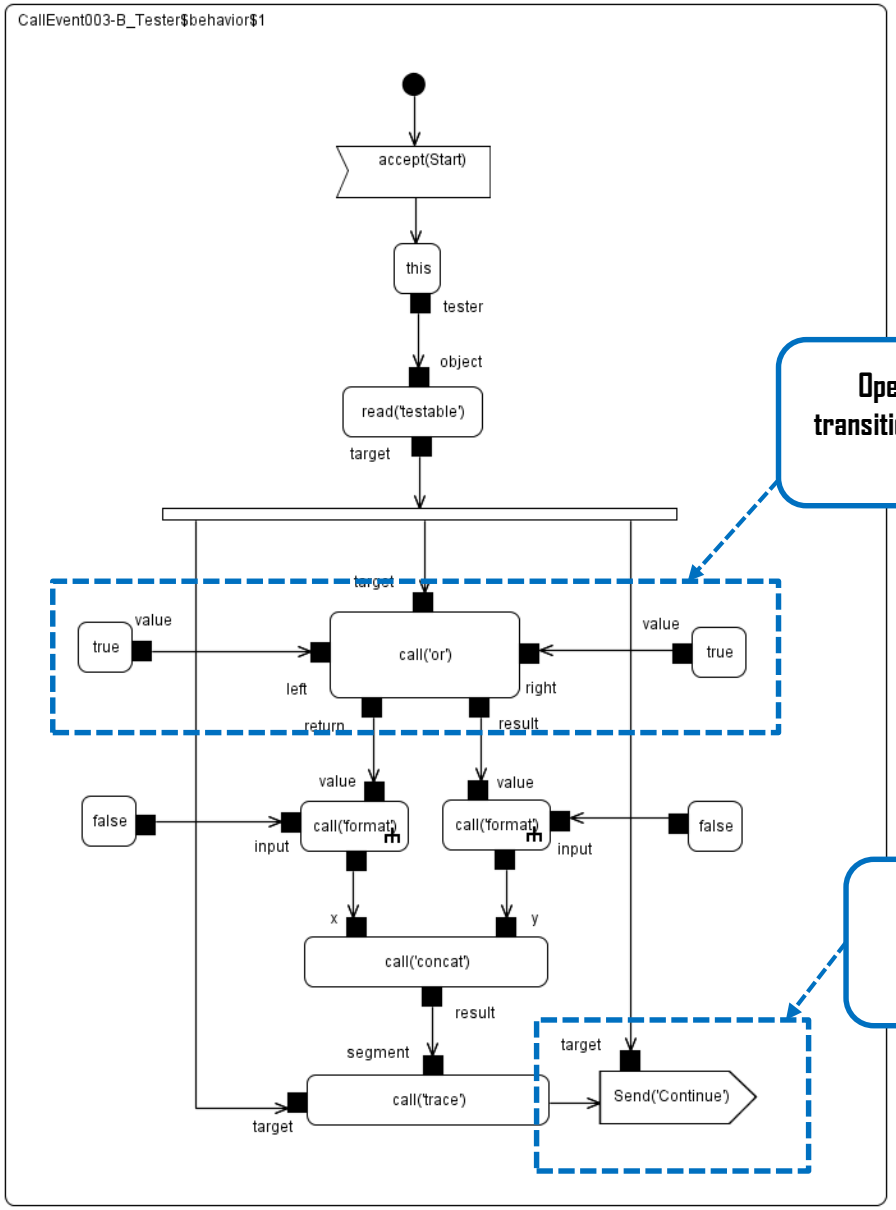
TESTING – CALL EVENT 003-B

EXCERPT FROM THE TEST SUITE



TESTING – CALL EVENT 003-B

EXCERPT FROM THE TEST SUITE

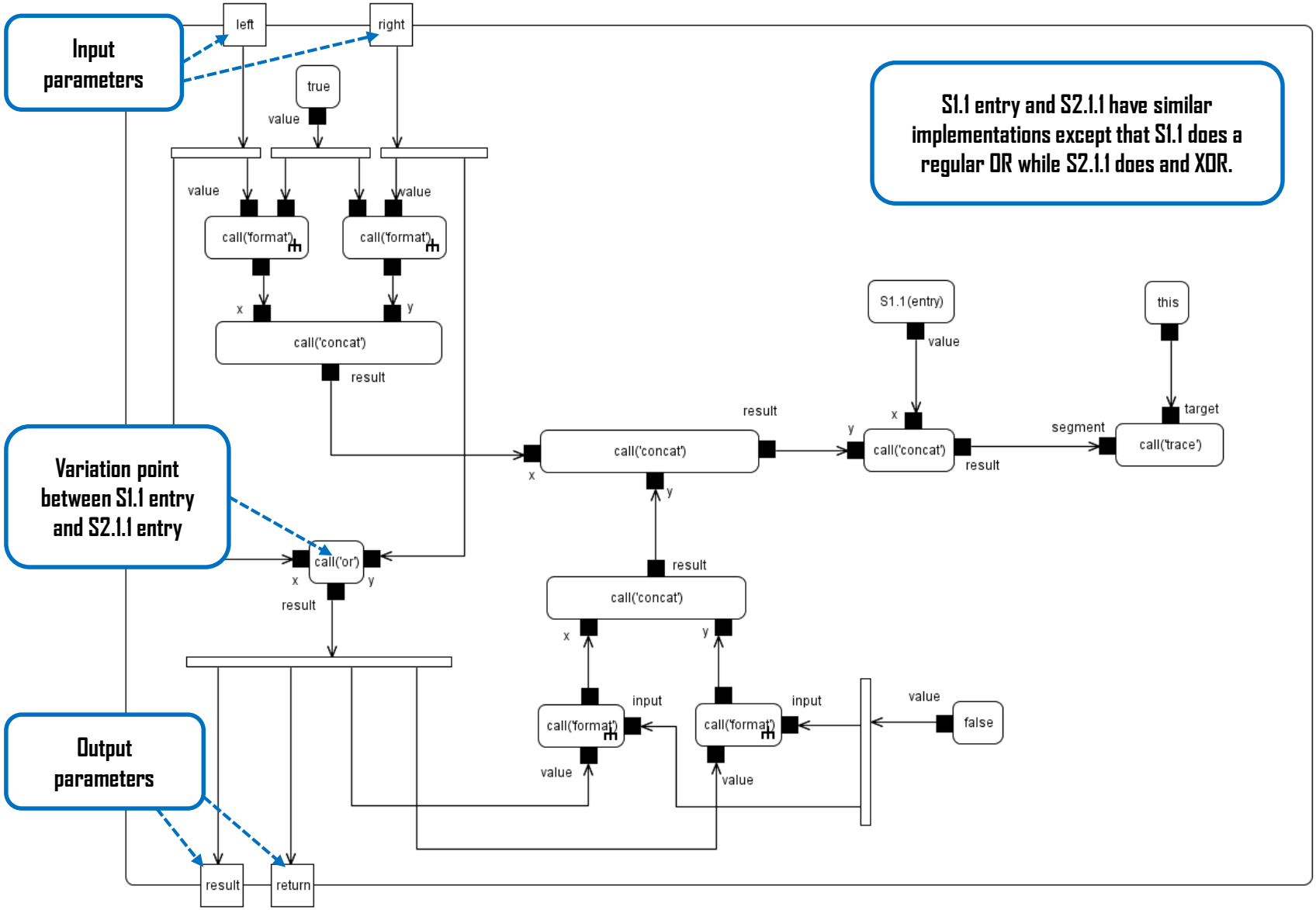


Operation call that will trigger the transition T2. Both left and right operands will have the value true.

Signal sending that will trigger the transition T3.

TESTING – CALL EVENT 003 - B

IMPLEMENTATION OF ENTRY BEHAVIORS



TESTING – CALL EVENT 003 - B

IMPLEMENTATION OF ENTRY BEHAVIORS

	Event pool	Deferred event	Configuration	Transitions
1	[]	[]	[] – Initial step	[T1]
2	[Call(or), **CE(wait)**]	[]	[wait]	[]
3	[**Call(or)**]	[]	[wait]	[T2(T1.1, T2.1(T2.1.1))]
4	[CE(S2.1.1), **CE(S1.1)**]	[]	[S1[S1.1, S2.1[S2.1.1]]]	[T1.2]
5	[**CE(S2.1.1)**]	[]	[S1[S2.1[S2.1.1]]]	[T2.1.2]
6	[**CE(S2.1)**]	[]	[S1[S2.1]]	[T2.2]
7	[Continue, **CE(S1)**]	[]	[S1]	[]
8	[**Continue**]	[]	[S1]	[T3]

This could happen in a different order since there is concurrency. Indeed CE(S2.1.1) may have arrived first at the repository

First execution trace: CE(S1.1) is dispatched first

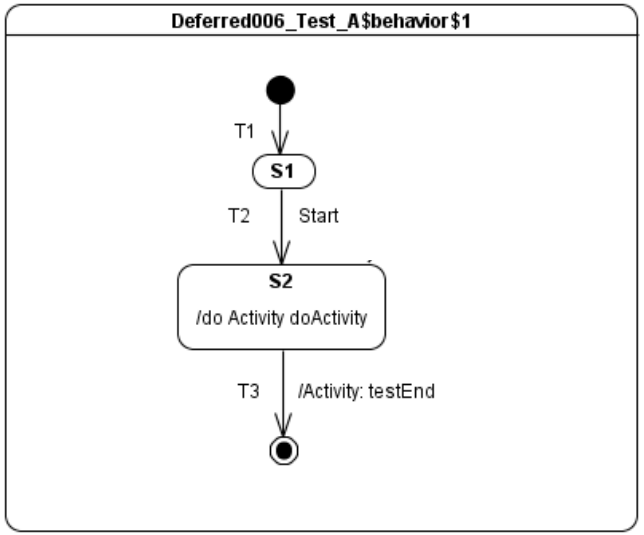
```
/*Test call event 003 - B */
ce003b = new 'CallEvent003-B_SemanticTest'();
ce003b.name = "CallEvent 003 - B";
ce003b.expectedTraces->add("S1.1(entry)[in=true][in=true][out=true][out=true]::S2.1.1(entry)[in=true][in=true][out=false][out=false]::[out=false][out=false]");
ce003b.expectedTraces->add("S2.1.1(entry)[in=true][in=true][out=false][out=false]::S1.1(entry)[in=true][in=true][out=true][out=true]::[out=true][out=true]");
suite.tests->add(ce003b);
```

Second execution trace: CE(S2.1.1) is dispatched first

DO ACTIVITY SEMANTICS

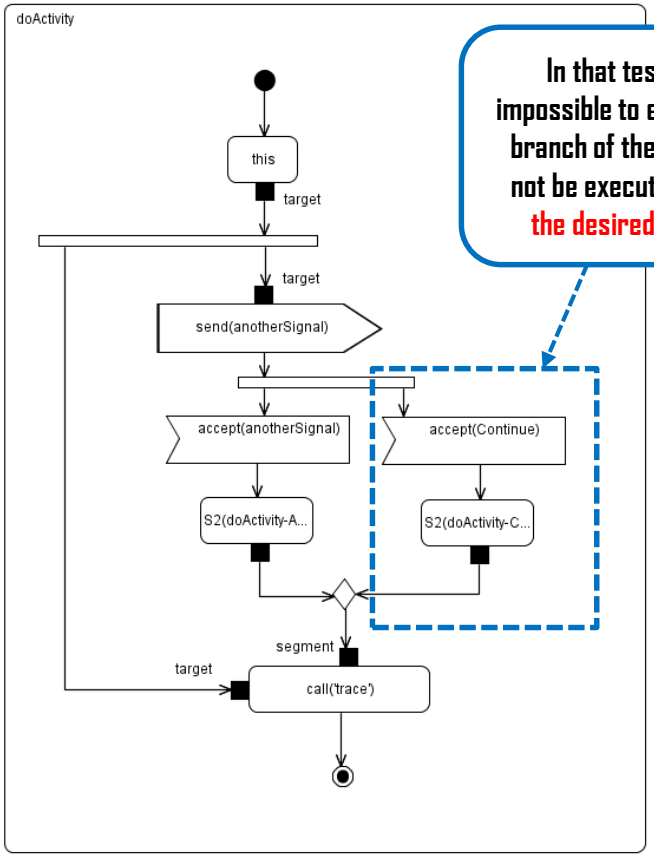
Problem

- Ensure that a *doActivity* invoked from a state is able to consume an event occurrence sent from an external object.




```
private import Util::Protocol::Messages::Start;
private import Util::Protocol::Messages::Continue;
private import Util::Architecture::A_testable_tester;

activity 'Behavior006_Tester_A$behavior$1'() {
   accept(Start);
   this.testable.Continue();
}
```

```
/*Deferred006 A */
d006 = new 'Deferred006_SemanticTest_A'();
d006.name = "Deferred 006 A";
d006.expectedTraces->add("S2(doActivity-AnotherSignal)");
suite.tests->add(d006);
```



In that test case it is impossible to ensure that this branch of the doActvity will not be executed. This is not the desired semantics !

DO ACTIVITY RESOLUTION

PROBLEM STATEMENT - EXECUTION

	Event pool	Deferred events	Configuration	Transitions
1	[]	[]	[] – Initial step	[T1]
2	[Start, **CE(S1)**]	[]	[S1]	[]
3	[**Start**]	[]	[S1]	[T2]
4	[**AnotherSignal**]	[]	[S2]	[]
5	[**Continue**]	[]	[S2]	[]

	Event pool	Wait points	Executed nodes
1	[]	[] – Initial step	[Initial, this, fork, send(AnotherSignal) – Fork – accept(AnotherSignal) – accept(Continue)]
2	[**Continue**]	[accept(AnotherSignal) – **accept(Continue)**]	S1(doActivityContinue) – MergeNode – call('trace') - FinalNode

- Case 1:
  - *AnotherSignal* is lost. *Continue* triggers *accept(Continue)*.
- Case 2:
  - *AnotherSignal* is accepted, it triggers *accept(AnotherSignal).*
- Case 3:
  - Both *AnotherSignal* and *Continue* are lost. The doActivity waits forever.

Resolution

- Rule 1:
  - To avoid a state-machine to consume an event in a certain state then this state must have a deferred trigger for that event type.
- Rule 2:
  - A state-machine is not allowed to defer an event (even if specified) if this event can be accepted by an accepter registered by a *doActivity*.
- Rule 3:
  - A doActivity is allowed to accept an event deferred by the state-machine.

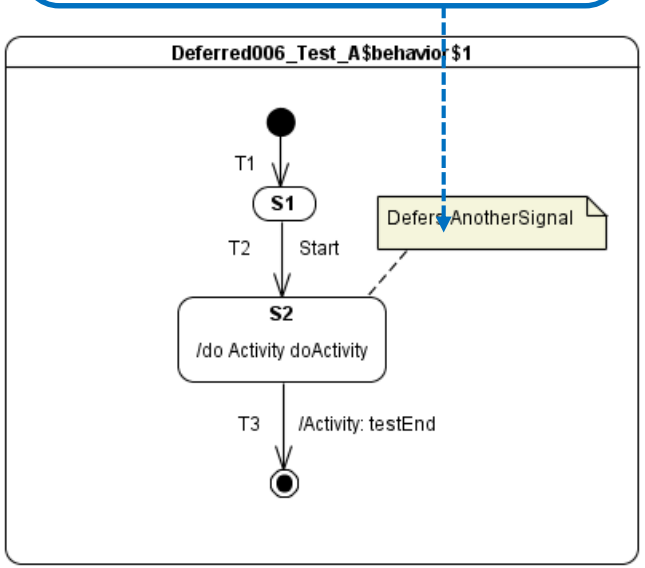
Impact on the semantic model

- StateMachineEventAccepter
  - Rule 2 - `isDeferred(in Eventoccurrence): Boolean`
- DoActivityContextObject
  - Rule 3 – `register(in EventAccepter)`. The event pool is checked, if not matching deferred event could be found then the accepter is register in the context object activation. Otherwise, a new RTC step is scheduled for the doActivity.

DO ACTIVITY RESOLUTION

PROBLEM STATEMENT - EXECUTION

As specified in Rule 1, the AnotherSignal is specified as being deferred in S2 (is the state on which the signal might be lost if not deferred).



DO ACTIVITY RESOLUTION

PROBLEM STATEMENT - EXECUTION

	Event pool	Deferred events	Configuration	Transitions
1	[]	[]	[] – Initial step	[T1]
2	[Start, **CE(S1)**]	[]	[S1]	[]
3	[**Start**]	[]	[S1]	[T2]
4	[**AnotherSignal**]	[]	[S2]	[]
5	[**Continue**]	[AnotherSignal]	[S2]	[]

Accepted from the deferred event pool

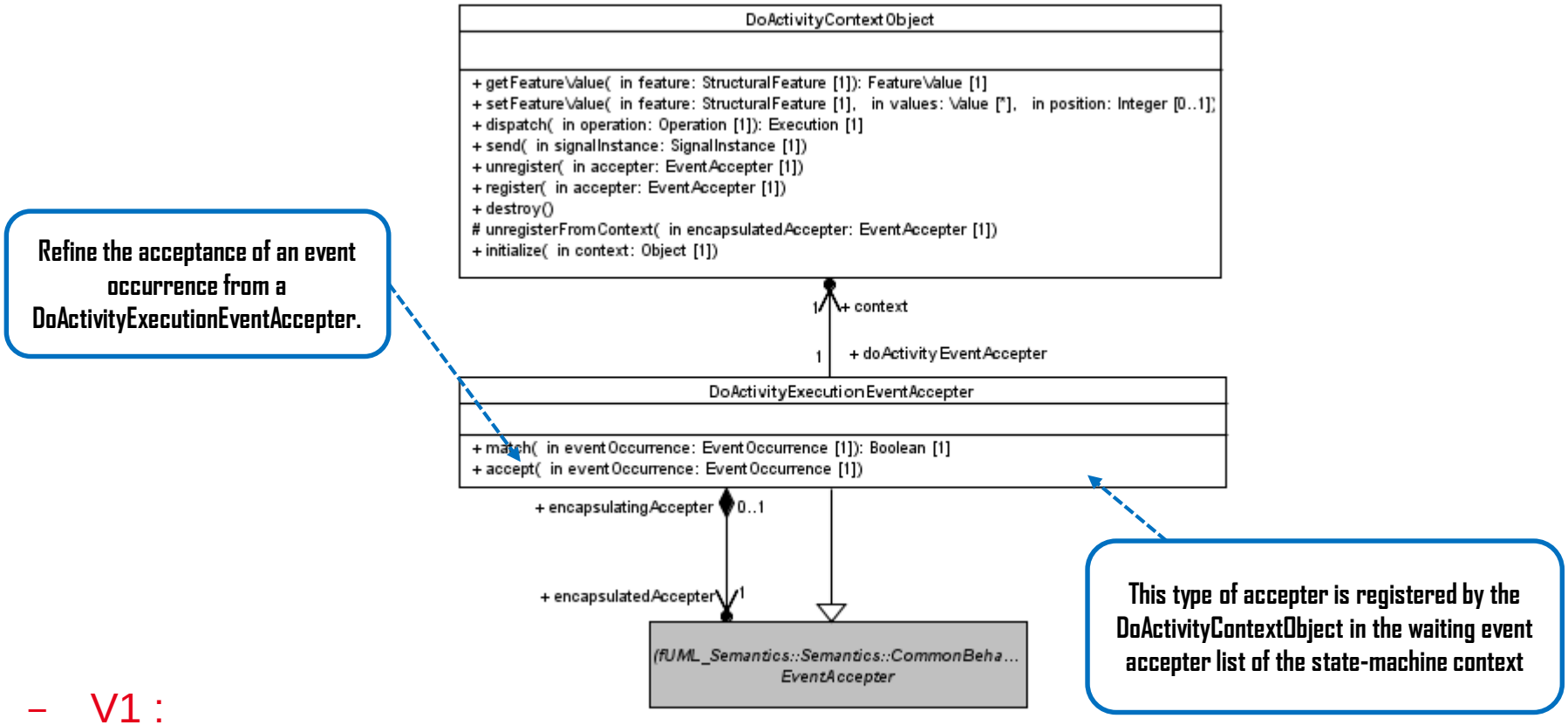
doActivity is invoked from RTC step N°4

	Event pool	Wait points	Executed nodes
1	[]	[] – Initial step	[Initial this, fork, send(AnotherSignal) – Fork – accept(AnotherSignal) – accept(Continue)]
2	[**AnotherSignal**]	[**accept(AnotherSignal)** – accept(Continue)]	S1(doActivityAnotherSignal) – MergeNode – call('trace') - FinalNode

- Case 1:
  - *AnotherSignal* is accepted from the deferred event pool.
- Case 2:
  - *AnotherSignal* is accepted.

DO ACTIVITY RESOLUTION

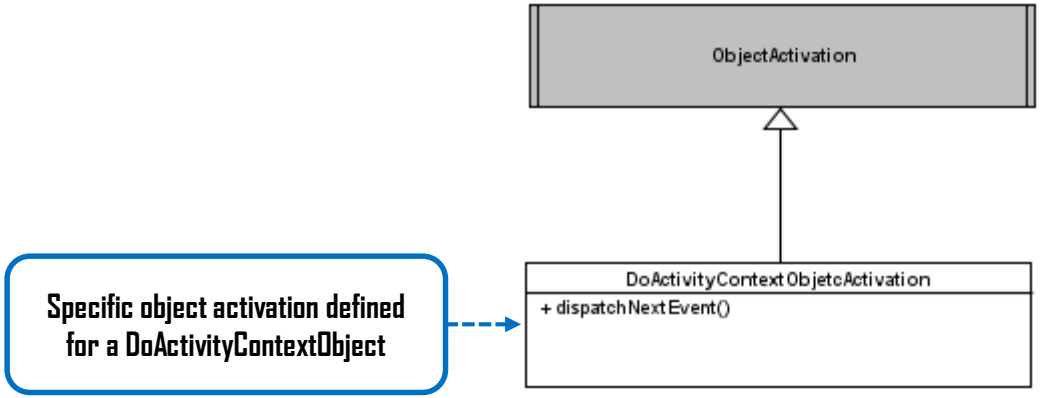
SIDE CORRECTIONS



- V1 :
  - The acceptance was directly delegated to the wrapped event accepter. The side effect is that the RTC step is still realized in the execution thread of the state-machine.
- V2:
  - When accepting the event occurrence the DoActivityExecutionEventAccepter only triggers a RTC step in the doActivity. This is realized by sending the accepted event occurrence to the doActivity object activation.

DO ACTIVITY RESOLUTION

SIDE CORRECTIONS



- `dispatchNextEvent()`
  - Preserve fUML semantics. Implements the dispatch.
  - Add the possibility to notify the state which invoked the doActivity that this latter has completed its execution.
  - Check for execution completion is base on the analysis of the presence of registered accepter at the end of each RTC step.

STATE-MACHINE REDEFINITION SEMANTICS

STATE-MACHINE REDEFINITION SEMANTICS

WHAT SAYS THE RFP

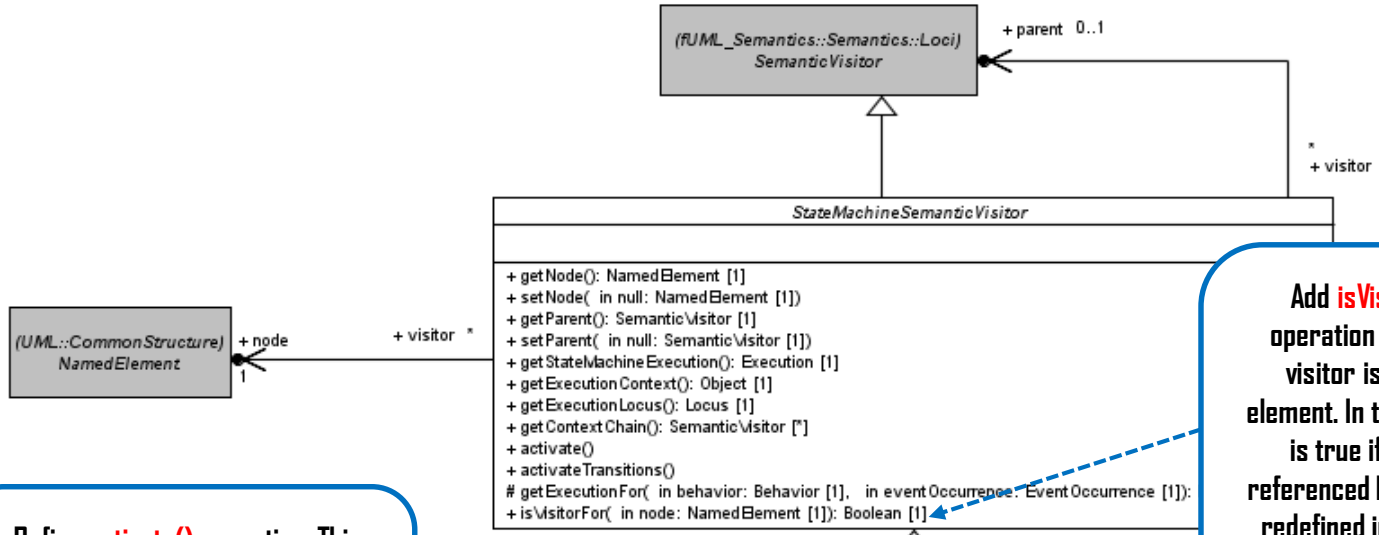
Precise semantics

- Proposals may provide precise semantics for state machine redefinition, as represented by the following meta-associations:
  - A_extendedRegion_region
  - A_extendedStateMachine_stateMachine
  - A_redefinedState_state
  - A_redefinedTransition_transition
  - A_redefinitionContext_region
  - A_redefinitionContext_state
  - A_redefinitionContext_transition

- Agenda: June 2016

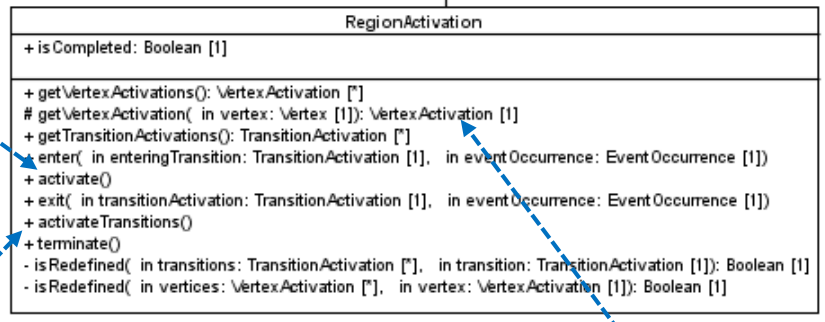
STATE-MACHINE REDEFINITION SEMANTICS

CHANGES IN THE SEMANTIC MODEL



Add isVisitorFor(...) operation. This operation assesses if a given semantic visitor is an interpreter for a model element. In the basic version this assertion is true if the given node is the node referenced by the visitor. This operation is redefined in StateActivation to cope with possibly redefined state.

Refine activate() operation. This operation recursively go through the different extended state-machines and only instantiate vertices that are not redefined.

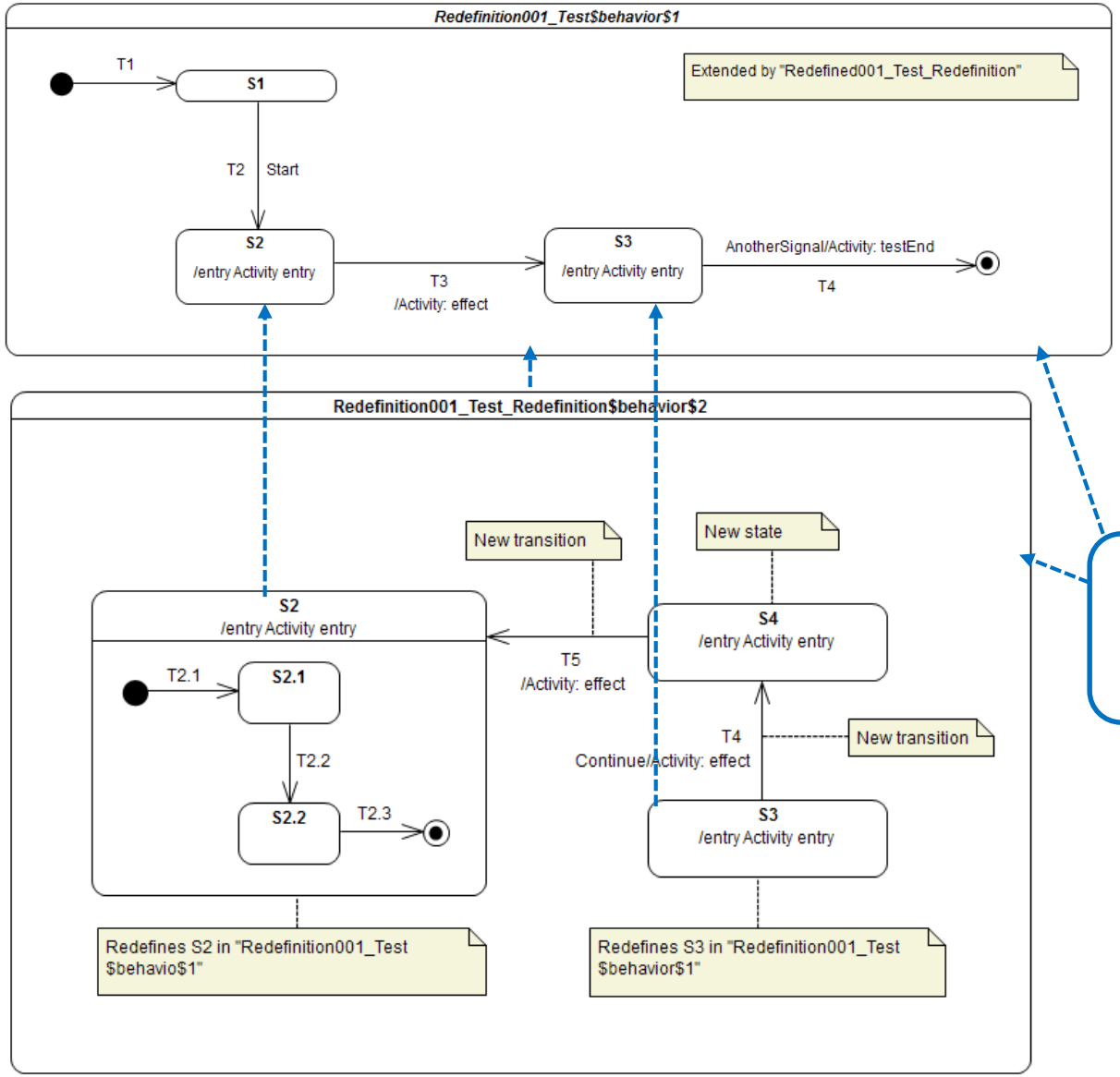


Refine activateTransitions() operation. This operation recursively go through the different extended state-machines and only instantiate transitions that are not redefined.

Refine getVertexActivation (...) operation. This operation now relies on isVisitorFor(...) during the search phase to figure out what is the corresponding visitor in the hierarchy

TESTING – REDEFINITION 001

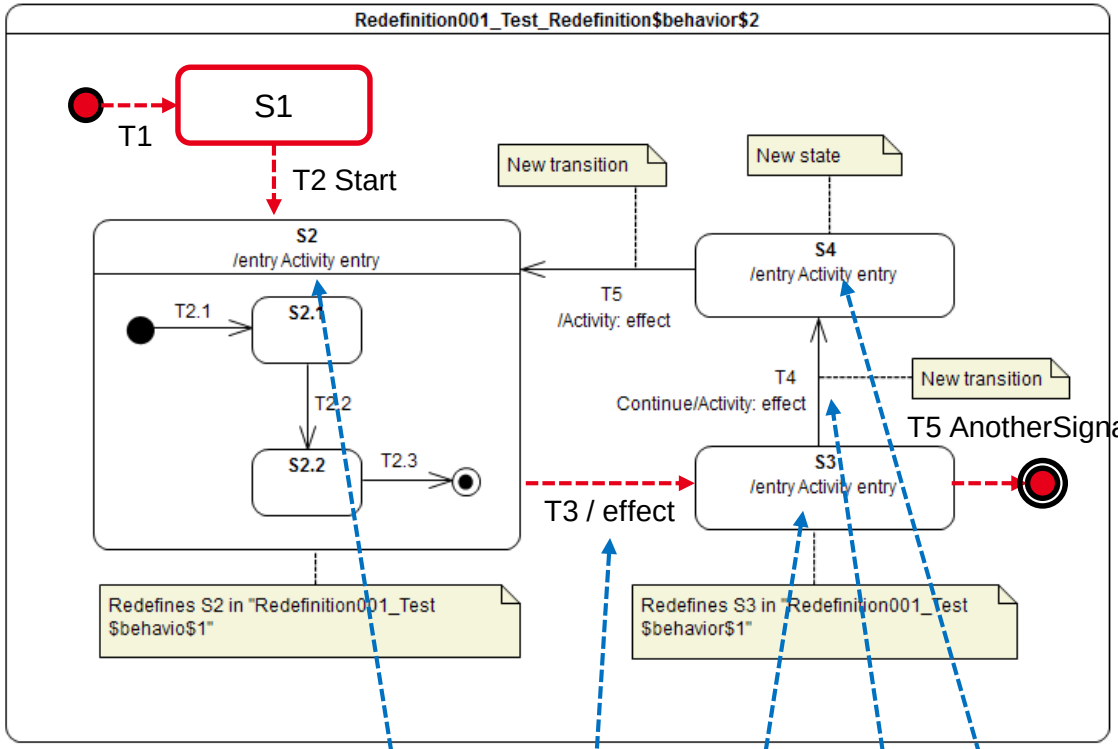
SINGLE LEVEL REDEFINITION



This example is exactly the one discussed in clause 8.5.3 in PSSM initial submission

TESTING – REDEFINITION 001

SINGLE LEVEL REDEFINITION



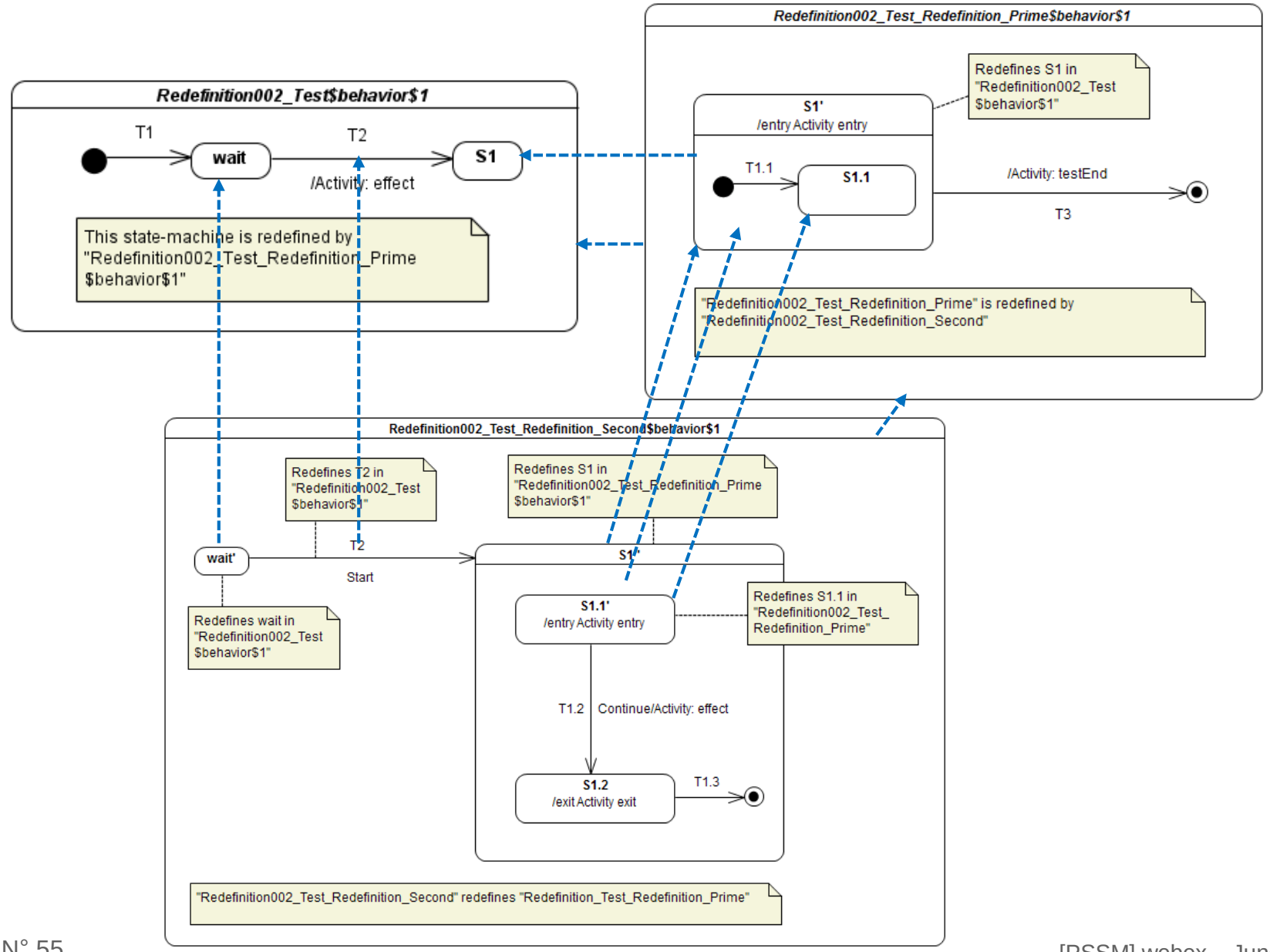

```
activity 'Redefinition001_Tester$behavior$1'() {
  accept(Start);
  this.testable.Continue();
  this.testable.AnotherSignal();
}
```

Finally executed state-machine.

```
/*Redefinition 001*/
r001 = new 'Redefinition001_SemanticTest'();
r001.name = "Redefinition 001";
r001.expectedTraces->add("S2(entry)-redefined::T3(effect)::S3(entry)-redefined::T4(effect)::S4(entry)::T5(effect)::S2(entry)-redefined::T3(effect)::S3(entry)-redefined");
suite.tests->add(r001);
```

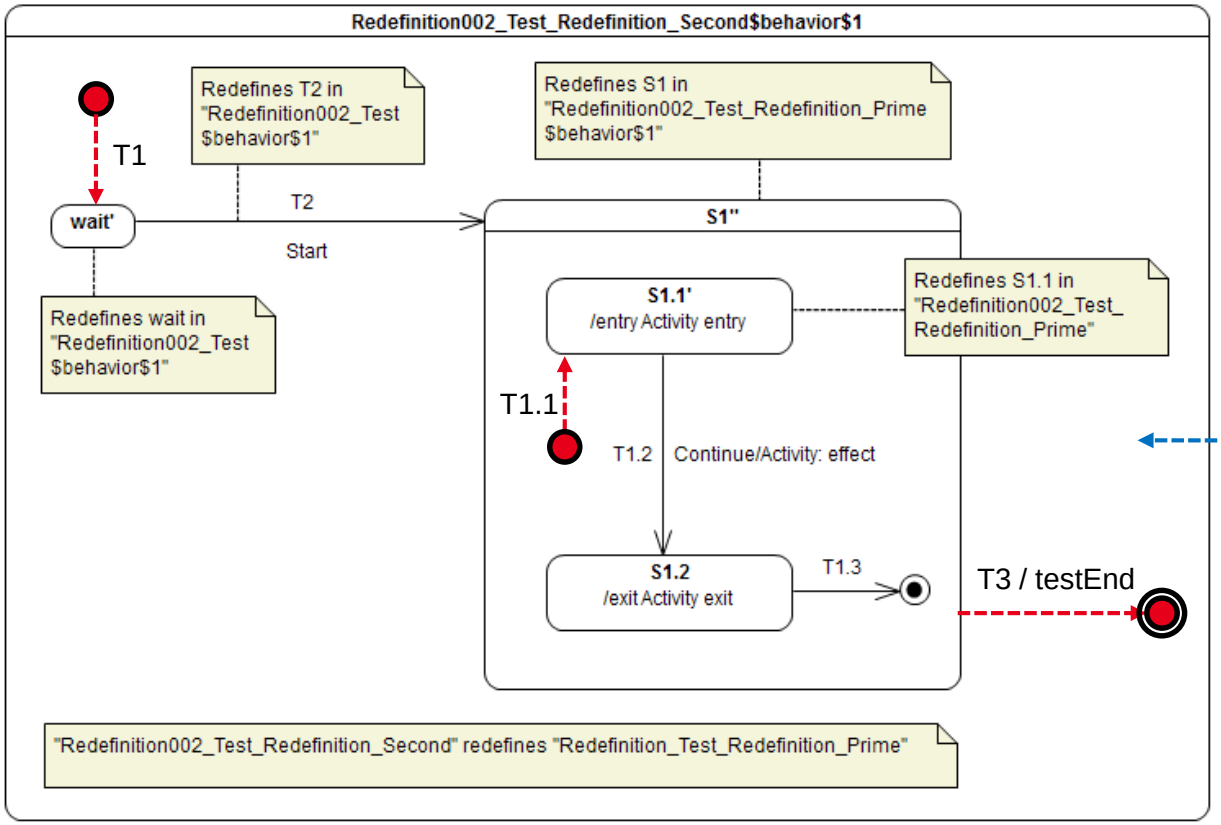
TESTING – REDEFINITION 002

MULTI-LEVEL REDEFINITION



TESTING – REDEFINITION 002

MULTI-LEVEL REDEFINITION




```
activity 'Redefinition002_Tester$behavior$1'() {
  accept(Start);
  this.testable.Continue();
}
```

Finally executed state-machine.

```
/*Redefinition 002*/
r002 = new 'Redefinition002_SemanticTest'();
r002.name = "Redefinition 002";
r002.expectedTraces->add("S1.1(entry)-redefined-second::T1.2(effect)::S1.2(exit)");
suite.tests->add(r002);
```

OUR CURRENT STATUS

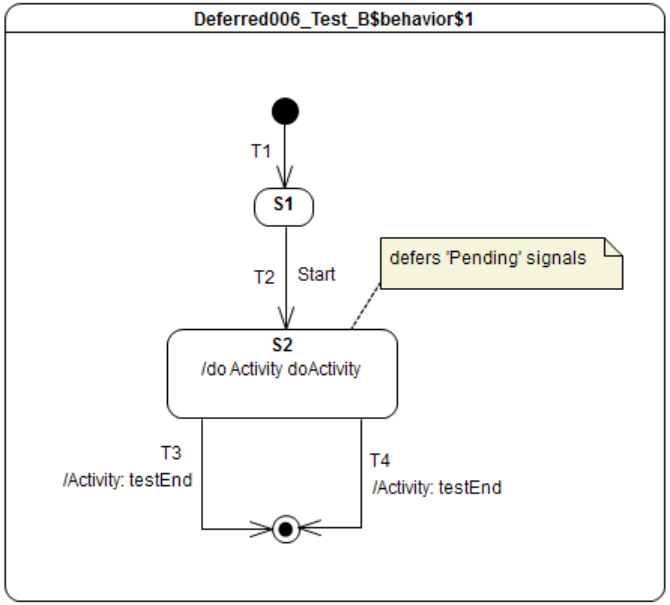
STATUS

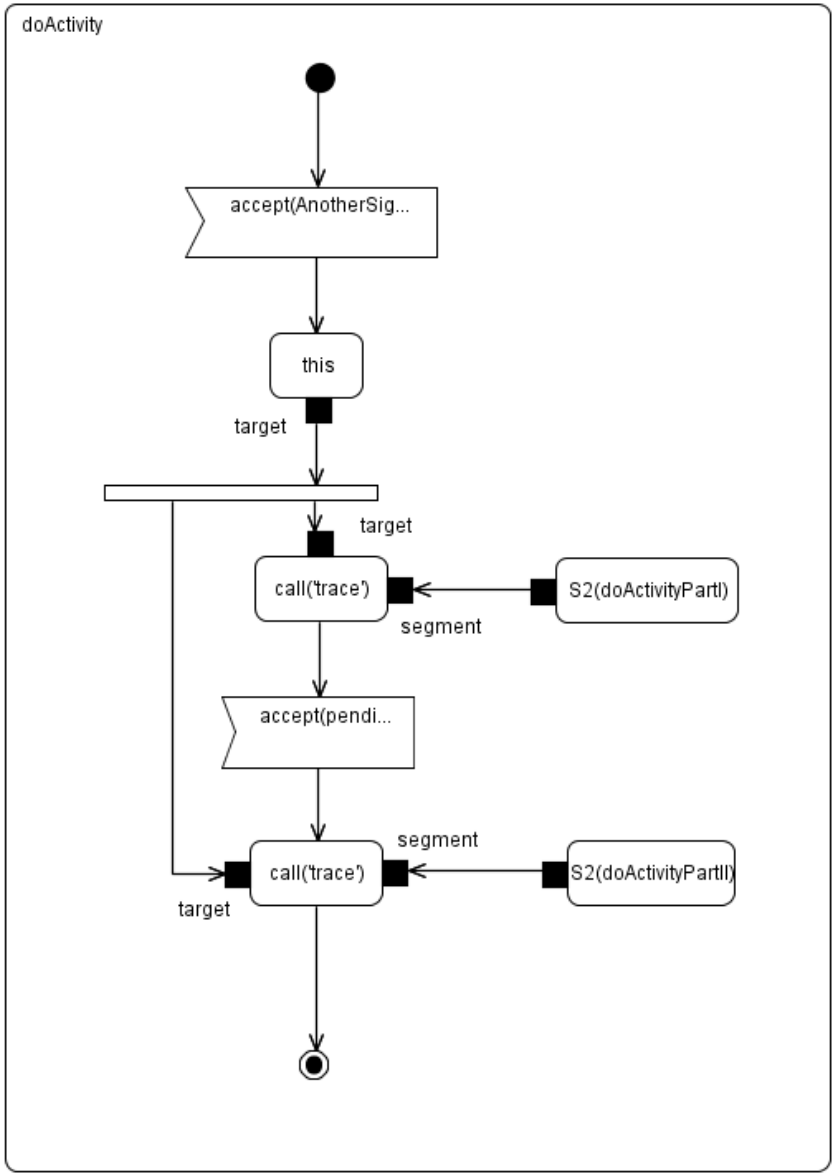
- Deferred events semantics completion OK
- State machine configuration analysis OK
- Test suite refactoring OK
- Call event semantics OK
- Event data passing ⚠
  - Fix: doActivity is not allowed to return values
  - Fix: data are for the moment not passed to guards (UML 2.6 RTF)
- DoActivity OK
- State machine redefinition ⚠
  - Needs more intensive testing
- Standalone state-machines ⚠
  - Needs more intensive testing
- Implementation conformance to bUML ✕
  - We have the list of changes to apply
- Pseudo states ✕
  - History (deep and shallow)
  - Junction


Papyrus

BACKUP

DEFERRED 006 - B

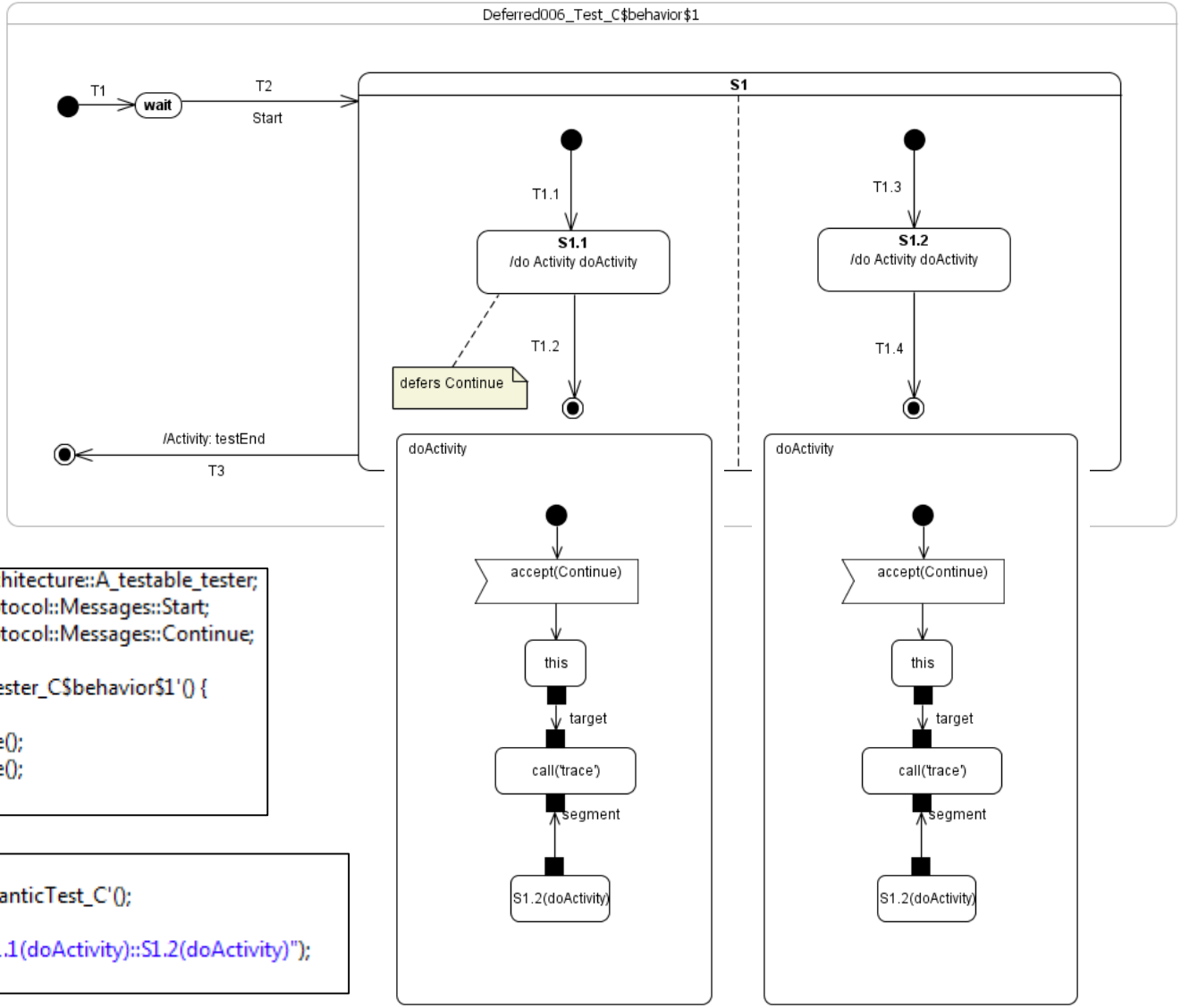





```
private import Util::Architecture::A_testable_tester;

activity 'Deferred006_Tester_B$behavior$1'() {
   accept(Start);
   this.testable.Pending();
   this.testable.AnotherSignal();
}
```

```
/*Deferred006 B */
d006b = new 'Deferred006_SemanticTest_B'();
d006b.name ="Deferred 006 B";
d006b.expectedTraces->add("");
d006b.expectedTraces->add("S2(doActivityPartI)::S2(doActivityPartII)");
suite.tests->add(d006b);
```

DEFERRED 006 - C




```
private import Util::Architecture::A_testable_tester;
private import Util::Protocol::Messages::Start;
private import Util::Protocol::Messages::Continue;

activity 'Deferred006_Tester_C$behavior$1'() {
   accept(Start);
   this.testable.Continue();
   this.testable.Continue();
}
```

```
/*Deferred006 C */
d006c = new 'Deferred006_SemanticTest_C'();
d006c.name = "Deferred 006 C";
d006c.expectedTraces->add("S1.1(doActivity)::S1.2(doActivity)");
suite.tests->add(d006c);
```