Developing Data-Intensive Cloud Applications with Iterative Quality Enhancements

Transformations to Analysis Models — Companion Document

Deliverable 3.1

Copyright © 2016, DICE consortium – All rights reserved

Contents

List of Listings

Copyright © 2016, DICE consortium – All rights reserved

Copyright © 2016, DICE consortium – All rights reserved

1 QVT Transformation for the UML Activity Diagram ath the DPIM level

Listing 1: Transformation for the UML Activity Diagram

```
import es.unizar.disco.pnml.utils.PnmlDiceUtils;

import helpers;

modeltype UML uses 'http://www.eclipse.org/uml2/5.0.0/UML';
modeltype PNML uses 'http:///ptnet.ecore';
modeltype TYPES uses 'http://es.unizar.disco/simulation/datatypes/1.0';
modeltype TRACE uses 'http://es.unizar.disco/simulation/traces/1.0';
modeltype CONST uses 'http://es.unizar.disco/pnconstants/1.0';
modeltype ECORE uses 'http://www.eclipse.org/emf/2002/Ecore';


transformation ad2pnml(in ad : UML, in vars : TYPES, out res : PNML, out traces :
    TRACE);

/**
  Main method:
    1.- Create the root elements (PetriNetDoc, PetriNet, Page)
    2.- Transform ActivityNodes, creating subnets of places and transitions
    2.- Transform ControlFlows, linking the previously created subnets
    2.- Complete transformation with the information of the workload in the initial
        nodes
*/
main() {
  // Transform top-level elements
  ad.scenario().map model2doc();

  // Transform net contents
  ad.scenario().node[UML::ActivityNode] -> map activityNode2subNet();
  ad.scenario().edge[UML::ControlFlow] -> map controlFlow2arc();

  // Transform deployment
  ad->objectsOfType(ActivityPartition)->flatten()[ActivityPartition] -> map
      partitions2resources();

  // Transform workload descriptions
  var initialNodes := ad.scenario().node[UML::InitialNode];
  assert fatal (initialNodes->size() = 1) with log ("Only Activities with a single
      Initial Node are supported");
  initialNodes[getGaWorkloadEvent().oclIsUndefined().not()] -> map
      initialNodes2AnalysisType();

  // Set time metadata
  if (resolveoneIn(UML::NamedElement::model2net,
      PNML::PetriNet).toolspecifics->notEmpty()) {
    log("Base time unit is 's'");
    log("Base frequency unit is 'Hz'");
  } else {
    log("Base time unit is 'tick'");
    log("Base frequency unit is 'events per tick'");
  }
}

```

Copyright © 2016, DICE consortium – All rights reserved

```
48 /**
49   Create the PetriNetDoc and the PetriNet
50 */
51 mapping UML::NamedElement::model2doc() : PNML::PetriNetDoc {
52   nets := self.map model2net();
53 }
54
55 mapping UML::NamedElement::model2net() : PNML::PetriNet {
56   id := createRandomUniqueId();
57   name := object PNML::Name {
58     text := self.name;
59   };
60   pages := self.map model2page();
61 }
62
63 /**
64   Create the page
65 */
66 mapping UML::NamedElement::model2page() : PNML::Page {
67   id := createRandomUniqueId();
68 }
69
70 /**
71   Sets the metadata for the base time unit on the PNML file
72 */
73 mapping baseTimeUnit(unit : String) {
74   var net := resolveoneIn(UML::NamedElement::model2net, PNML::PetriNet);
75   net.toolspecifics += baseTimeUnitToolInfo(unit);
76 }
77
78 /**
79   Get the initial node's information to call the proper transformation
80   rule according to the kind of workload
81 */
82 mapping UML::InitialNode::initialNodes2AnalysisType() {
83   var pattern := self.getGaWorkloadEvent_pattern();
84   switch {
85     case (pattern.oclIsUndefined())
86       assert warning (false) with log ("Unparseable workload pattern");
87     case (pattern.oclIsTypeOf(ClosedPattern))
88       self.map model2closedNet();
89     case (pattern.oclIsTypeOf(OpenPattern))
90       self.map model2openNet();
91     else {
92       assert warning (false) with log ("Unknown workload pattern: " +
           pattern._rawExpression);
93     };
94   }
95 }
96
97 /**
98   Transform the model to a closed net using the initial node
99 */
100 mapping UML::InitialNode::model2closedNet() {
101   var pattern := self.getGaWorkloadEvent_pattern().oclAsType(ClosedPattern);
102   var place := self.resolveoneIn(UML::ActivityNode::activityNode2place);
103   var closingTransitions := ad.scenario().node[UML::ActivityFinalNode].resolveIn(
       UML::ActivityNode::activityNode2transition);
```

Copyright © 2016, DICE consortium – All rights reserved

```
104   assert warning (closingTransitions->size() = 1) with log ("More than one closing
          transition found, this can be problematic to compute the throughput of the
          system");
105   closingTransitions->forEach(transition) {
106     map arc(transition, place);
107     // Add tracing information
108     ad.scenario().map trace(transition, "model2closedNet");
109   };
110   assert fatal (pattern.population_.value() <= 32767) with log ("Population must be
          an integer value under 32767, error in expression '" + pattern._rawExpression +
          "'");
111   place.initialMarking := object PNML::PTMarking {
112     text := pattern.population_.value();
113   };
114   // The extDelay is set in the time of the transition just next to the
          InitialNodePlace
115   var transition := self.resolveoneIn(UML::ActivityNode::activityNode2transition);
116   transition.toolspecifics += expTransitionToolInfo(1 / pattern.extDelay.value());
117   transition.toolspecifics += infServerTransitionToolInfo();
118 }
119
120 /**
121   Transform the model to an open net using the initial node
122 */
123 mapping UML::InitialNode::model2openNet() {
124   var pattern := self.getGaWorkloadEvent_pattern().oclAsType(OpenPattern);
125   var transition := self.resolveoneIn(UML::ActivityNode::activityNode2transition);
126   // Set the rate of the previously generated transition
127   // This transition will generate the workload directly
128   assert fatal (pattern.arrivalRate.oclIsUndefined() xor
          pattern.interArrivalTime.oclIsUndefined())
129     with log ("Only one 'arrivalRate' xor 'interArrivalTime' should be defined");
130   if (pattern.arrivalRate.oclIsUndefined().not()) {
131     transition.toolspecifics += expTransitionToolInfo(pattern.arrivalRate.value());
132     transition.toolspecifics += oneServerTransitionToolInfo();
133   } else if (pattern.interArrivalTime.oclIsUndefined().not()) {
134     transition.toolspecifics += expTransitionToolInfo(1 /
            pattern.interArrivalTime.value());
135     transition.toolspecifics += oneServerTransitionToolInfo();
136   };
137
138   var finalTransitions := ad.scenario().node[UML::ActivityFinalNode].resolveIn(
          UML::ActivityNode::activityNode2transition);
139   assert warning (finalTransitions->size() = 1) with log ("More than one final
          transition found, this can be problematic to compute the throughput of the
          system");
140   finalTransitions->forEach(finalTransition) {
141     // Add tracing information
142     ad.scenario().map trace(finalTransition, "model2openNet");
143   };
144
145 }
146
147 mapping UML::ActivityPartition::partitions2resources()
148 when {
149   self.represents.getPaLogicalResource_poolSize().oclIsUndefined().not();
150 }{
151   var poolSize := self.represents.getPaLogicalResource_poolSize();
152
```

Copyright © 2016, DICE consortium – All rights reserved

```
153  var partitions := self->closure(subpartition)->including(self);
154
155  var containedNodes := ad.objectsOfKind(UML::ActivityNode)->select(node :
         ActivityNode | partitions->intersection(node.inPartition)->notEmpty());
156
157  var incomingEdges := ad.objectsOfKind(UML::ActivityEdge)
158            ->select(edge : ActivityEdge |
                 containedNodes->includes(edge.source).not())
159            ->select(edge : ActivityEdge | containedNodes->includes(edge.target));
160  var outgoingEdges := ad.objectsOfKind(UML::ActivityEdge)
161            ->select(edge : ActivityEdge | containedNodes->includes(edge.source))
162            ->select(edge : ActivityEdge |
                 containedNodes->includes(edge.target).not());
163
164  var place := self.map namedElement2place();
165  place.initialMarking := object PNML::PTMarking {
166    text := poolSize.value();
167  };
168
169  // For each edge that enters the Activity Partition, we need to modify
170  // the subnet that was previously generated by adding an intermediate place
171  // and an intemediate immediate transition that receives the tokent from the
172  // resource's place, i.e.
173  // [T]-->[P]
174  // is transformed into:
175  // [T]-->[P']-->[T']-->[P]
176  //              ^
177  //              |
178  //      [Resources-Place]
179
180  incomingEdges->forEach(edge) {
181    // Get the original transition
182    var originalSourceTrans :=
         edge.source.resolveoneIn(UML::NamedElement::namedElement2transition);
183    // Ensure that we have a transition with a single outgoing arc
184    assert fatal (originalSourceTrans.oclIsUndefined().not()) with log ("Could not
         find a source transition while transforming edge '" + edge.toString() + "' in
         partition '" + self.toString() + "'");
185    assert fatal (originalSourceTrans.OutArcs->size() = 1) with log ("Unexpected
         number of outgoing arcs in transition '" + originalSourceTrans.toString() +
         "'");
186
187    // Create the intermediate nodes
188    var intermediatePlace := object PNML::Place {
189      containerPage := resolveoneIn(UML::NamedElement::model2page);
190      id := createRandomUniqueId();
191    };
192    var intermediateTrans := object PNML::Transition {
193      containerPage := resolveoneIn(UML::NamedElement::model2page);
194      id := createRandomUniqueId();
195    };
196
197
198    var arc := originalSourceTrans.OutArcs->asSequence()->first();
199    var originalTargetPlace := arc.target;
200
201    // Relink the original arc
202    arc.target := intermediatePlace;
203    // Add the intermediate nodes
```

Copyright © 2016, DICE consortium – All rights reserved

```
204     map arc(intermediatePlace, intermediateTrans);
205     map arc(intermediateTrans, originalTargetPlace);
206     // Attach the resource's place
207     map arc(place, intermediateTrans);
208   };
209
210
211   // For each edge that leaves the Activity Partition, we need to add an arc to the
212   // resource's place
213   outgoingEdges->forEach(edge) {
214     // Get the original transition
215     var originalSourceTrans :=
          edge.source.resolveoneIn(UML::NamedElement::namedElement2transition);
216     // Ensure that we have a transition
217     assert fatal (originalSourceTrans.oclIsUndefined().not()) with log ("Could not
          find a source transition while transforming edge '" + edge.toString() + "' in
          partition '" + self.toString() + "'");
218     // Add an arc from the transition to the resource's place
219     map arc(originalSourceTrans, place);
220   };
221
222   // Add tracing information
223   self.represents.map trace(place, "partitions2resources");
224 }
225
226 mapping UML::ActivityNode::activityNode2subNet() disjuncts
227 UML::InitialNode::initialNode2subNet,
228 UML::DecisionNode::decisionActivityNode2subNet,
229 UML::JoinNode::joinActivityNode2subNet,
230 UML::ActivityNode::basicActivityNode2subNet {};
231
232 /**
233   Transform a generic ActivityNodel into a simple [place]->[transition] subnet
234 */
235 mapping UML::ActivityNode::basicActivityNode2subNet() {
236   var place := self.map activityNode2place();
237   var transition := self.map activityNode2transition();
238   var arc := map arc(place, transition);
239   // Add tracing information
240   self.map trace(place, "basicActivityNode2place");
241   self.map trace(transition, "basicActivityNode2transition");
242 }
243
244 /**
245   Transforms an InitialNode into a pair of [place]-->[transition] if the workload
        pattern
246   is undefined or closed, and only a single [transition] if the workload pattern is
        open
247 */
248 mapping UML::InitialNode::initialNode2subNet() {
249   var transition := self.map activityNode2transition();
250   if (self.getGaWorkloadEvent_pattern().oclIsTypeOf(OpenPattern).not()) {
251     var place := self.map activityNode2place();
252     var arc := map arc(place, transition);
253   };
254 }
255
256 /**
257   Transform a generic Decision ActivityNode into a single [place].
```

Copyright © 2016, DICE consortium – All rights reserved

```
  Transitions will be handled in the transformation of the ControlFlows
*/
mapping UML::DecisionNode::decisionActivityNode2subNet() {
  var place := self.map activityNode2place();
}

/**
  Transform a generic Join ActivityNode into a single [transition].
  Places will be handled in the transformation of the ControlFlows
*/
mapping UML::JoinNode::joinActivityNode2subNet() {
  var transition := self.map activityNode2transition();
}


/**
  Transform a generic NamedElement into a Place
*/
mapping UML::NamedElement::namedElement2place() : PNML::Place {
  containerPage := resolveoneIn(UML::NamedElement::model2page);
  id := createRandomUniqueId();
  if (self.name.oclIsUndefined().not()) {
    name := object PNML::Name {
      text := self.name;
    };
  };
}

/**
  Transform a generic ActivityNode into a Place
*/
mapping UML::ActivityNode::activityNode2place() : PNML::Place
inherits UML::NamedElement::namedElement2place {
}

/**
  Transform a generic NamedElement into a Transition
*/
mapping UML::NamedElement::namedElement2transition() : PNML::Transition {
  containerPage := resolveoneIn(UML::NamedElement::model2page);
  id := createRandomUniqueId();
  if (self.name.oclIsUndefined().not()) {
    name := object PNML::Name {
      text := self.name;
    };
  };
}

/**
  Transform a generic ActivityNode into a Transition and
  creates any additional ToolInfo depending on the ActivityNode
  subtype (e.g., OpaqueActions with hostDemand may create
  exponential transitions)
*/
mapping UML::ActivityNode::activityNode2transition() : PNML::Transition
inherits UML::NamedElement::namedElement2transition {
  toolspecifics += self[OpaqueAction].map opaqueActionHostDemand2toolInfo();
}

```

Copyright © 2016, DICE consortium – All rights reserved

```
/**
  Transformas an OpaqueAction with a hostDemand annotation to a ToolInfo element
*/
mapping UML::OpaqueAction::opaqueActionHostDemand2toolInfo() : List ( PNML::ToolInfo )
when {
    self.getGaStep_hostDemand().oclIsUndefined().not();
}{
  var hostDemand := self.getGaStep_hostDemand();
  result += expTransitionToolInfo( 1 / hostDemand.value());
  result += infServerTransitionToolInfo();
}

/**
  Creates an Arc from 'src' to 'tgt'
*/
mapping arc(in src : PNML::Node, in tgt : PNML::Node) : PNML::Arc {
  containerPage := resolveoneIn(UML::NamedElement::model2page);
  id := createRandomUniqueId();
  source := src;
  target := tgt;
}

/**
  Transforms a ControlFlow between two ActivityNodes to different subnets,
  being the most simple an arc between the subnets corresponding to the
  ActivityNodes contected by the ControlFlow
*/
mapping UML::ControlFlow::controlFlow2arc() disjuncts
UML::ControlFlow::decisionControlFlow2arc,
UML::ControlFlow::joinControlFlow2arc,
UML::ControlFlow::basicControlFlow2arc {};

/**
  Transforms a ControlFlow between two ActivityNodes to an Arc between a
  transition and a place in the form:
  UML:
    [AN1] --> [AN2]
  PN:
    ( [PlaceAN1]-- ... -->[TransAN1] )--->( [PlaceAN2]-- ... -->[TransAN2] )
     -------------------------------  -------------------------------
               Subnet AN1                          Subnet AN2
*/
mapping UML::ControlFlow::basicControlFlow2arc() {
  assert warning (self.getGaStep_prob().oclIsUndefined())
    with log ("Only ControlFlows departing from a DecisionNode should define a
        probability. " +
          "Ignoring annotation in the context element '" + self.toString() + "'");
  var transition :=
      self.source.resolveoneIn(UML::ActivityNode::activityNode2transition);
  var place := self.target.resolveoneIn(UML::ActivityNode::activityNode2place);
  map arc(transition, place);
}

/**
  Transforms a ControlFlow between a DecissionNode and a generic ActivityNode.
  The DecissionNode has been previously transformed as a single place, and now
  we need to create a probabilistic immediate transition for each departing
      ControlFlow
*/
```

Copyright © 2016, DICE consortium – All rights reserved

```
373 mapping UML::ControlFlow::decisionControlFlow2arc() when {
374   self.source.oclIsKindOf(DecisionNode)
375 }{
376   var decisionPlace :=
          self.source.resolveoneIn(UML::ActivityNode::activityNode2place);
377   var nextNodePlace :=
          self.target.resolveoneIn(UML::ActivityNode::activityNode2place);
378   var transition := self.map namedElement2transition();
379   transition.toolspecifics += self.map probabilisticControlFlow2toolInfo();
380
381   map arc(decisionPlace, transition);
382   map arc(transition, nextNodePlace);
383 }
384
385 /**
386   Transforms a ControlFlow between a generic ActivityNode and a Join.
387   The Join has been previously transformed as a single transition, and now
388   we need to create a Place for each incoming ControlFlow to enable the
389   synchronization of the different execution flows
390 */
391 mapping UML::ControlFlow::joinControlFlow2arc() when {
392   self.target.oclIsKindOf(JoinNode)
393 }{
394   var prevNodeTrans :=
          self.source.resolveoneIn(UML::ActivityNode::activityNode2transition);
395   var joinTrans :=
          self.target.resolveoneIn(UML::ActivityNode::activityNode2transition);
396   var place := self.map namedElement2place();
397
398   if (place.name.oclIsUndefined() and self.target.name.oclIsUndefined().not()) {
399     place.name := object PNML::Name {
400       text := self.target.name.addSuffixNumber();
401     };
402   };
403
404   map arc(prevNodeTrans, place);
405   map arc(place, joinTrans);
406 }
407
408 /**
409   Transformas ControlFlow with a prob annotation to a ToolInfo element
410 */
411 mapping UML::ControlFlow::probabilisticControlFlow2toolInfo() : PNML::ToolInfo
412 when {
413     self.getGaStep_prob().oclIsUndefined().not();
414 }{
415   init {
416     result := probTransitionToolInfo(self.getGaStep_prob().value());
417   }
418 }
419
420
421 /*****************************************************************************
422   Traceability mappings
423 *****************************************************************************/
424
425 mapping OclAny::trace(to : OclAny) : TRACE::Trace {
426   init {
427     result := object TRACE::Trace {
```

Copyright © 2016, DICE consortium – All rights reserved

```
428       fromDomainElement := self.eObject();
429       toAnalyzableElement := to.eObject();
430     }
431   }
432 }
433
434 mapping OclAny::trace(to : OclAny, text : String) : TRACE::Trace {
435   init {
436     result := object TRACE::Trace {
437       fromDomainElement := self.eObject();
438       toAnalyzableElement := to.eObject();
439       rule := text;
440     }
441   }
442 }
443
444 /*************************************************************************
445   Navigation helpers
446   Helpers on domains are only valid in the context of a transformations and
447   cannot be moved to a library
448 *************************************************************************/
449
450 helper UML::scenario() : UML::Activity {
451   // When running the transformation from the simulation tool, the UML domain must
452   // contain a single activity at its root
453   assert warning (self.rootObjects()[UML::Activity]->size() = 1) with log ("No single
        Activity instance was found at the root of the UML input model, trying to use
        the first Activity in the model instead");
454
455   if (self.rootObjects()[UML::Activity]->isEmpty().not()) {
456     return self.rootObjects()[UML::Activity]->asOrderedSet()->first();
457   };
458   // This execution path is useful when running the transformation at development time
459   return self.objectsOfType(UML::Activity)->asOrderedSet()->first();
460 }
461
462 helper TYPES::vars() : Set ( PrimitiveVariableAssignment ) {
463   return self.rootObjects()[PrimitiveVariableAssignment];
464 }
465
466 helper TYPES::PrimitiveVariableAssignment::asDict() : Dict(String, Real) {
467   var vars : Dict (String, Real) := Dict {};
468   self->forEach(assignment) {
469     vars->put(assignment.variable, assignment.value.toString().toReal());
470   };
471   return vars;
472 }
473
474 helper TRACE::set() : TRACE::TraceSet {
475   return self.rootObjects()[TRACE::TraceSet]->asSequence()->first();
476 }
477
478 /*************************************************************************
479   Intermediate classes
480   Sadly, intermediate classes cannot be shared among libraries or
481   transformations.
482 *************************************************************************/
483
484 intermediate class ArrivalPattern {
```

Copyright © 2016, DICE consortium – All rights reserved

```
  _rawExpression : String;
}

intermediate class ClosedPattern extends ArrivalPattern {
  population_ : NFP_Integer;
  extDelay : NFP_Real;
}

intermediate class OpenPattern extends ArrivalPattern {
  interArrivalTime : NFP_Duration;
  arrivalRate : NFP_Frequency;
  arrivalProcess : String;
}

intermediate class NFP_CommonType {
  _rawExpression : String;
  expr : String;
  source : String;
  statQ : String;
  dir : String;
  mode : String;
}

intermediate class NFP_Integer extends NFP_CommonType {
  value : Integer;
}

intermediate class NFP_Real extends NFP_CommonType{
  value : Real;
}

intermediate class NFP_Duration extends NFP_Real {
  unit : String;
  clock : String;
  precision : Real;
  worst : Real;
  best : Real;
}

intermediate class NFP_Frequency extends NFP_Real {
  unit : String;
  precision : Real;
}

/*******************************************************************************
  Tagged values utilities
*******************************************************************************/

/**
  Helper that parses a VSL tuple containing a NFP_CommonType
*/
helper String::toNfpCommonType() : NFP_CommonType {
  var res := object NFP_CommonType {
    _rawExpression := self;
    statQ := null;
    expr := null;
    source := null;
    dir := null;
    mode := null;
```

Copyright © 2016, DICE consortium – All rights reserved

```
544   };
545   if (self.isTuple()) {
546     var entries := self.asTuple();
547     res.expr := entries->get("expr");
548     res.statQ := entries->get("statQ");
549     res.source := entries->get("source");
550     res.dir := entries->get("dir");
551     res.mode := entries->get("mode");
552   } else {
553     res.expr := self;
554   };
555   assert warning (res.statQ.oclIsUndefined() or res.statQ = 'mean')
556     with log ("Expression '" + self + "' defines an unknown 'statQ' value, expected
          empty or 'mean'");
557   assert warning (res.source.oclIsUndefined() or res.source = 'est' or res.source =
        'meas')
558     with log ("Expression '" + self + "' defines an unsupported 'source' for an input
          parameter, expected 'est' or 'meas'.");
559   assert warning (res.dir.oclIsUndefined())
560     with log ("Expression '" + self + "' defines a value for the unsupported 'dir'
          property");
561   assert warning (res.mode.oclIsUndefined())
562     with log ("Expression '" + self + "' defines a value for the unsupported 'mode'
          property");
563   return res;
564 }
565
566 /**
567   Helper that parses a VSL tuple containing a NFP_Integer
568 */
569 helper String::toNfpInteger() : NFP_Integer {
570   var nfp := self.toNfpCommonType();
571   var res := object NFP_Integer {
572     _rawExpression := nfp._rawExpression;
573     expr := nfp.expr;
574     statQ := nfp.statQ;
575     source := nfp.source;
576     dir := nfp.dir;
577     mode := nfp.mode;
578     value := null;
579   };
580   if (self.isTuple()) {
581     var entries := self.asTuple();
582     res.value := entries->get("value").toInteger();
583   };
584   assert fatal (res.value.oclIsUndefined() xor res.expr.oclIsUndefined())
585     with log ("Expression '" + self + "' must define either a valid 'value' or a valid
          'expr'");
586   return res;
587 }
588
589 /**
590   Helper that parses a VSL tuple containing a NFP_Real
591 */
592 helper String::toNfpReal() : NFP_Real {
593   var nfp := self.toNfpCommonType();
594   var res := object NFP_Real {
595     _rawExpression := nfp._rawExpression;
596     expr := nfp.expr;
```

Copyright © 2016, DICE consortium – All rights reserved

```
597     statQ := nfp.statQ;
598     source := nfp.source;
599     dir := nfp.dir;
600     mode := nfp.mode;
601     value := null;
602   };
603   if (self.isTuple()) {
604     var entries := self.asTuple();
605     res.value := entries->get("value").toReal();
606   };
607   assert fatal (res.value.oclIsUndefined() xor res.expr.oclIsUndefined())
608     with log ("Expression '" + self + "' must define either a valid 'value' or a valid
          'expr'");
609   return res;
610 }
611
612
613 /**
614   Helper that parses a VSL tuple containing a NFP_Duration
615 */
616 helper String::toNfpDuration() : NFP_Duration {
617   var nfp := self.toNfpReal();
618   var res := object NFP_Duration {
619     _rawExpression := nfp._rawExpression;
620     value := nfp.value;
621     expr := nfp.expr;
622     statQ := nfp.statQ;
623     source := nfp.source;
624     dir := nfp.dir;
625     mode := nfp.mode;
626     unit := null;
627     clock := null;
628     precision := null;
629     worst := null;
630     best := null;
631   };
632   if (self.isTuple()) {
633     var entries := self.asTuple();
634     res.unit := entries->get("unit");
635     res.clock := entries->get("clock");
636     res.precision := entries->get("precision").toReal();
637     res.worst := entries->get("worst").toReal();
638     res.best := entries->get("best").toReal();
639   };
640   assert warning (res.unit.oclIsUndefined().not())
641     with log ("Expression '" + self + "' does not define a 'unit', assumming the
          default base unit (see complete log)");
642   assert warning (res.dir.oclIsUndefined())
643     with log ("Expression '" + self + "' defines a value for the unsupported 'clock'
          property");
644   assert warning (res.dir.oclIsUndefined())
645     with log ("Expression '" + self + "' defines a value for the unsupported
          'precision' property");
646   assert warning (res.dir.oclIsUndefined())
647     with log ("Expression '" + self + "' defines a value for the unsupported 'worst'
          property");
648   assert warning (res.dir.oclIsUndefined())
649     with log ("Expression '" + self + "' defines a value for the unsupported 'best'
          property");
```

Copyright © 2016, DICE consortium – All rights reserved

```
650
651   return res;
652 }
653
654 /**
655   Helper that parses a VSL tuple containing a NFP_Frequency
656 */
657 helper String::toNfpFrequency() : NFP_Frequency {
658   var nfp := self.toNfpReal();
659   var res := object NFP_Frequency {
660     _rawExpression := nfp._rawExpression;
661     value := nfp.value;
662     expr := nfp.expr;
663     statQ := nfp.statQ;
664     source := nfp.source;
665     dir := nfp.dir;
666     mode := nfp.mode;
667     unit := null;
668     precision := null;
669   };
670   if (self.isTuple()) {
671     var entries := self.asTuple();
672     res.unit := entries->get("unit");
673     res.precision := entries->get("precision").toReal();
674   };
675   assert warning (res.unit.oclIsUndefined().not())
676     with log ("Expression '" + self + "' does not define a 'unit', assumming the
          default base unit (see complete log)");
677   assert warning (res.dir.oclIsUndefined())
678     with log ("Expression '" + self + "' defines a value for the unsupported 'clock'
          property");
679   assert warning (res.dir.oclIsUndefined())
680     with log ("Expression '" + self + "' defines a value for the unsupported
          'precision' property");
681   assert warning (res.dir.oclIsUndefined())
682     with log ("Expression '" + self + "' defines a value for the unsupported 'worst'
          property");
683   assert warning (res.dir.oclIsUndefined())
684     with log ("Expression '" + self + "' defines a value for the unsupported 'best'
          property");
685
686   return res;
687 }
688
689 helper NFP_Integer::value() : Integer {
690   if (self.value.oclIsUndefined().not()) {
691     return self.value;
692   };
693   return self.expr.eval(vars.vars()).toInteger();
694 }
695
696 helper NFP_Real::value() : Real {
697   if (self.value.oclIsUndefined().not()) {
698     return self.value;
699   };
700   return self.expr.eval(vars.vars()).toReal();
701 }
702
703 helper NFP_Duration::value() : Real {
```

Copyright © 2016, DICE consortium – All rights reserved

```
704   var value : Real;
705   if (self.value.oclIsUndefined().not()) {
706     value := self.value;
707   };
708   value := self.expr.eval(vars.vars()).toReal();
709   if (self.unit.oclIsUndefined().not()) {
710     map baseTimeUnit("s");
711     value := value.convert(self.unit, "s");
712   };
713   return value;
714 }
715
716 helper NFP_Frequency::value() : Real {
717   var value : Real;
718   if (self.value.oclIsUndefined().not()) {
719     value := self.value;
720   };
721   value := self.expr.eval(vars.vars()).toReal();
722   if (self.unit.oclIsUndefined().not()) {
723     map baseTimeUnit("s");
724     value := value.convert(self.unit, "Hz");
725   };
726   return value;
727 }
728
729 /*****************************************************************************
730   Getters for tagged values
731 *****************************************************************************/
732
733 helper UML::Element::getGaWorkloadEvent_pattern() : ArrivalPattern {
734   if (self.getGaWorkloadEvent() = null) {
735     return null;
736   };
737   var patternString := self.getValue(self.getGaWorkloadEvent(),
        "pattern").oclAsType(String);
738   var patternName := patternString.key();
739   var patternValue := patternString.value();
740   switch {
741     case (patternName = "closed") {
742       return object ClosedPattern {
743         _rawExpression := patternValue;
744         population_ := patternValue.asTuple()->get("population").toNfpInteger();
745         extDelay := patternValue.asTuple()->get("extDelay").toNfpDuration();
746       };
747     } case (patternString.key() = "open") {
748       return object OpenPattern {
749         _rawExpression := patternValue;
750         interArrivalTime :=
              patternValue.asTuple()->get("interArrivalTime").toNfpDuration();
751         arrivalRate := patternValue.asTuple()->get("arrivalRate").toNfpFrequency();
752         arrivalProcess := patternValue.asTuple()->get("arrivalProcess");
753       };
754     } else {
755       assert fatal (false) with log ("Unknown ArrivalPattern: " + patternString);
756     }
757   };
758   return null;
759 }
760
```

Copyright © 2016, DICE consortium – All rights reserved

```
761 helper UML::Element::getGaStep_hostDemand() : NFP_Duration {
762   if (self.getGaStep() = null) {
763     return null;
764   };
765   var hostDemandStrings := self.getValue(self.getGaStep(),
        "hostDemand").oclAsType(Collection(String));
766   assert warning (hostDemandStrings->size() = 1)
767     with log ("Unexpected number of 'hostDemand' tagged values found, expected 1. "+
768           "Only the first 'mean' value will be used (if found). " +
769           "The context element is '" + self.toString() + "'");
770   return hostDemandStrings.toNfpDuration()->
771       select(demand | demand.statQ.oclIsUndefined() or demand.statQ = 'mean')->
772       asSequence()->first();
773 }
774
775 helper UML::Element::getGaStep_prob() : NFP_Real {
776   if (self.getGaStep() = null) {
777     return null;
778   };
779   var prob := self.getValue(self.getGaStep(), "prob").oclAsType(String);
780   return prob.toNfpReal();
781 }
782
783 helper UML::Element::getPaLogicalResource_poolSize() : NFP_Integer {
784   if (self.getPaLogicalResource() = null) {
785     return null;
786   };
787   var prob := self.getValue(self.getPaLogicalResource(),
        "poolSize").oclAsType(String);
788   return prob.toNfpInteger();
789 }
```

Copyright © 2016, DICE consortium – All rights reserved

2 QVT Transformation for the UML Sequence Diagram ath the DPIM level

Listing 2: Transformation for the UML Sequence Diagram

```
1  import es.unizar.disco.pnml.utils.PnmlDiceUtils;
2
3  import helpers;
4
5  modeltype UML uses 'http://www.eclipse.org/uml2/5.0.0/UML';
6  modeltype PNML uses 'http:///ptnet.ecore';
7  modeltype TYPES uses 'http://es.unizar.disco/simulation/datatypes/1.0';
8  modeltype TRACE uses 'http://es.unizar.disco/simulation/traces/1.0';
9  modeltype CONST uses 'http://es.unizar.disco/pnconstants/1.0';
10 modeltype ECORE uses 'http://www.eclipse.org/emf/2002/Ecore';
11
12
13 transformation sd2pnml(in sd : UML, in vars : TYPES, out res : PNML, out traces :
       TRACE);
14
15
16 helper validate() {
17   // Validate all ExecutionSpecifications start and end in the same parent
         InteractionFragment
18   assert fatal
         (sd.scenario().allSubobjectsOfKind(UML::ExecutionSpecification)->forAll(
         start.namespace = finish.namespace))
19     with log ("Malformed input model: Start and Finish events for all
           ExecutionSpecifications must belong to the same namespace");
20
21   // Validate all InteractionFragments belong to a lifeline
22   assert fatal (sd.scenario().allSubobjectsOfKind(UML::InteractionFragment)->forAll(
         covered->notEmpty()))
23     with log ("Malformed input model: All InteractionFragments must belong to a
           lifeline");
24   assert fatal (sd.scenario().allSubobjectsOfKind(UML::InteractionFragment)->forAll(
         covered->notEmpty()))
25     with log ("Malformed input model: All InteractionFragments must belong to a
           lifeline");
26
27   // Validate that the interaction starts sending a message
28   var first := sd.scenario().fragment->asOrderedSet()->first().oclAsType(
         UML::MessageOccurrenceSpecification);
29   assert fatal (first.message.sendEvent = first)
30     with log ("The first fragment of the interaction must be a send message event
           (MessageOccurrenceSpecification)");
31
32   // Validate that the interaction finishes receiving a message
33   var last := sd.scenario().fragment->asOrderedSet()->last().oclAsType(
         UML::MessageOccurrenceSpecification);
34   assert fatal (last.message.receiveEvent = last)
35     with log ("The last fragment of the interaction must be a receive message event
           (MessageOccurrenceSpecification)");
36
37   // Validate that the first and last interaction fragment belong to the same
         lifeline (the "actor"" lifeline)
38   assert fatal (first.getLifeline() = last.getLifeline())
```

Copyright © 2016, DICE consortium – All rights reserved

```
39      with log ("The last message of the Interaction must be received by the lifeline
            sending the first message");
40  }
41
42  /**
43    Main method:
44  */
45  main() {
46
47    // Validate that the model is well-formed
48    validate();
49
50    // Transform top-level elements
51    sd.scenario().map model2doc();
52
53    // Transform net contents
54    sd.scenario().map interaction2subnet();
55
56    // Transform deployment
57
58    // Transform workload descriptions
59
60    // Set Time metadata
61    if (resolveoneIn(UML::NamedElement::model2net,
          PNML::PetriNet).toolspecifics->notEmpty()) {
62      log("Base time unit is 's'");
63      log("Base frequency unit is 'Hz'");
64    } else {
65      log("Base time unit is 'tick'");
66      log("Base frequency unit is 'events per tick'");
67    }
68  }
69
70  /**
71    Create the PetriNetDoc and the PetriNet
72  */
73  mapping UML::NamedElement::model2doc() : PNML::PetriNetDoc {
74    nets := self.map model2net();
75  }
76
77  mapping UML::NamedElement::model2net() : PNML::PetriNet {
78    id := createRandomUniqueId();
79    name := object PNML::Name {
80      text := self.name;
81    };
82    pages := self.map model2page();
83  }
84
85  /**
86    Create the page
87  */
88  mapping UML::NamedElement::model2page() : PNML::Page {
89    id := createRandomUniqueId();
90  }
91
92  /**
93    Sets the metadata for the base time unit on the PNML file
94  */
95  mapping baseTimeUnit(unit : String) {
```

Copyright © 2016, DICE consortium – All rights reserved

```
96    var net := resolveoneIn(UML::NamedElement::model2net, PNML::PetriNet);
97    net.toolspecifics += baseTimeUnitToolInfo(unit);
98  }
99
100 mapping UML::Interaction::interaction2subnet() {
101   // Create a transition to close the non-actor lifelines
102   self.map namedElement2transition();
103   // Transform each one of the inner interaction fragments to their corresponding
          subnets
104   self.allSubobjectsOfKind(UML::InteractionFragment).map interactionFragment2subnet();
105   // Connect each one of the inner interaction fragments to the previous fragment
106   self.allSubobjectsOfKind(UML::InteractionFragment).map connectToPrev();
107   // Create the messages subnets
108   self.message->map message2subnet();
109   // Transform the lifelines
110   self.lifeline->map lifeline2subnet();
111
112 }
113
114 mapping UML::Lifeline::lifeline2subnet() disjuncts
115 UML::Lifeline::actorLifeline2subnet,
116 UML::Lifeline::regularLifeline2subnet {};
117
118 mapping UML::Lifeline::actorLifeline2subnet()
119 when {
120   self.isActor();
121 }{
122   // Create the lifeline's initial transition
123   var initialTransition := object PNML::Transition {
124     containerPage := resolveoneIn(UML::NamedElement::model2page);
125     id := createRandomUniqueId();
126   };
127
128   // Create the lifeline's last place
129   var finalPlace := object PNML::Place {
130     containerPage := resolveoneIn(UML::NamedElement::model2page);
131     id := createRandomUniqueId();
132     name := object PNML::Name {
133       text := self.name + '_last';
134     };
135   };
136
137   // Create the final transition representing the lifeline
138   var finalTransition := self.map namedElement2transition();
139
140   // Connect the initial transition to the first element subnet
141   var firstElementPlace :=
          self.fragments()->first().resolveoneIn(UML::NamedElement::namedElement2place);
142   map arc(initialTransition, firstElementPlace);
143
144   // Connect last element subnet to the last place
145   var lastElementTransition :=
          self.fragments()->last().resolveoneIn(UML::NamedElement::namedElement2transition);
146   map arc(lastElementTransition, finalPlace);
147
148   // Connect the last place to the last transition
149   map arc(finalPlace, finalTransition);
150
151   // Check that there's no poolSize definition
```

Copyright © 2016, DICE consortium – All rights reserved

```
152   assert warning (self.getPaRunTInstance_poolSize().oclIsUndefined())
153     with log ("Actor Lifelines should not define a poolSize. Lifeline '" +
          self.toString() + "' does not respect this restriction");
154
155
156   // Configure the workload
157   var pattern := self.fragments()->first().oclAsType(
          UML::MessageOccurrenceSpecification).message.getGaWorkloadEvent_pattern();
158   if (pattern.oclIsUndefined()) {
159     // Undefined workload
160     assert warning (false) with log ("Unparseable workload pattern");
161   } else if (pattern.oclIsKindOf(ClosedPattern)) {
162     // Closed pattern: Create the initial place with the population, set
163     // the first transition's delay and connect with the rest of the subnet
164     var closedPattern := pattern.oclAsType(ClosedPattern);
165     var initialPlace := object PNML::Place {
166       containerPage := resolveoneIn(UML::NamedElement::model2page);
167       id := createRandomUniqueId();
168       name := object PNML::Name {
169         text := self.name;
170       };
171       initialMarking := object PNML::PTMarking {
172         text := closedPattern.population_.value();
173       };
174     };
175     initialTransition.toolspecifics += expTransitionToolInfo(1 /
          closedPattern.extDelay.value());
176     initialTransition.toolspecifics += infServerTransitionToolInfo();
177     map arc(initialPlace, initialTransition);
178     map arc(finalTransition, initialPlace);
179   } else if (pattern.oclIsKindOf(OpenPattern)) {
180     // Open pattern: set the load in the initial transtion rate
181     var openPattern := pattern.oclAsType(OpenPattern);
182     assert fatal (openPattern.arrivalRate.oclIsUndefined() xor
          openPattern.interArrivalTime.oclIsUndefined())
183       with log ("Only one 'arrivalRate' xor 'interArrivalTime' should be defined");
184     if (openPattern.arrivalRate.oclIsUndefined().not()) {
185       initialTransition.toolspecifics +=
            expTransitionToolInfo(openPattern.arrivalRate.value());
186       initialTransition.toolspecifics += oneServerTransitionToolInfo();
187     } else if (openPattern.interArrivalTime.oclIsUndefined().not()) {
188       initialTransition.toolspecifics += expTransitionToolInfo(1 /
            openPattern.interArrivalTime.value());
189       initialTransition.toolspecifics += oneServerTransitionToolInfo();
190     };
191   } else {
192     assert warning (false) with log ("Unknown workload pattern: " +
          pattern._rawExpression);
193   };
194 }
195
196 mapping UML::Lifeline::regularLifeline2subnet()
197 when {
198   self.isActor().not();
199 }{
200   assert fatal (self.getPaRunTInstance_poolSize().oclIsUndefined().not()) with log
        ("Lifeline '" + self.toString() + "' does not define a poolSize");
201
202   var initialPlace := object PNML::Place {
```

Copyright © 2016, DICE consortium – All rights reserved

```
203     containerPage := resolveoneIn(UML::NamedElement::model2page);
204     id := createRandomUniqueId();
205     name := object PNML::Name {
206       text := self.name;
207     };
208     initialMarking := object PNML::PTMarking {
209       text := self.getPaRunTInstance_poolSize().value();
210     };
211   };
212
213   // Create the lifeline's initial transition
214   var initialTransition := object PNML::Transition {
215     containerPage := resolveoneIn(UML::NamedElement::model2page);
216     id := createRandomUniqueId();
217   };
218
219   // Create the lifeline's last place
220   var finalPlace := object PNML::Place {
221     containerPage := resolveoneIn(UML::NamedElement::model2page);
222     id := createRandomUniqueId();
223     name := object PNML::Name {
224       text := self.name + '_last';
225     };
226   };
227
228   // Retrieve the final transition that synchronizes the lifelines
229   var finalTransition :=
          self.interaction.resolveoneIn(UML::NamedElement::namedElement2transition);
230
231   // Connect the initial transition to the first element subnet
232   var firstElementPlace :=
          self.fragments()->first().resolveoneIn(UML::NamedElement::namedElement2place);
233   map arc(initialTransition, firstElementPlace);
234
235   // Connect last element subnet to the last place
236   var lastElementTransition := self.fragments()->last().resolveoneIn(
          UML::NamedElement::namedElement2transition);
237   map arc(lastElementTransition, finalPlace);
238
239   // Connect the initial place to the initial transition
240   map arc(initialPlace, initialTransition);
241
242   // Connect the last place to the last transition
243   map arc(finalPlace, finalTransition);
244
245   // Connect the last transtion to the initial place
246   map arc(finalTransition, initialPlace);
247
248 }
249
250 mapping UML::InteractionFragment::connectToPrev() {
251   // Connect the place to the previous' element transition
252   --assert fatal (self.covered->size() = 1) with log ("Element '" + self.toString() +
          "' is not covered by a single lifeline");
253   self.covered->forEach(lifeline) {
254     var prev := self.prev(lifeline);
255     if (prev.oclIsUndefined().not()) {
256       var prevTransition :=
            prev.resolveoneIn(UML::NamedElement::namedElement2transition);
```

Copyright © 2016, DICE consortium – All rights reserved

```
257       var place := self.resolveoneIn(UML::NamedElement::namedElement2place);
258       map arc(prevTransition, place);
259     };
260   };
261 }
262
263 mapping UML::InteractionFragment::interactionFragment2subnet() disjuncts
264 UML::ExecutionSpecification::executionSpecification2subnet,
265 UML::InteractionFragment::genericInteractionFragment2subnet {};
266
267
268 mapping UML::InteractionFragment::genericInteractionFragment2subnet() {
269   var place := self.map namedElement2place();
270   var transition := self.map namedElement2transition();
271   var arc := map arc(place, transition);
272
273 }
274
275 mapping UML::ExecutionSpecification::executionSpecification2subnet() {
276   var place := self.map namedElement2place();
277   var transition := self.map executionSpecification2transition();
278   var arc := map arc(place, transition);
279 }
280
281 mapping UML::Message::message2subnet() when {
282   // Ignore messages that are sent and received by the same lifeline
283   if (self.sendEvent.oclIsInvalid().not() and self.receiveEvent.oclIsInvalid().not())
        then
284     self.sendEvent.oclAsType(UML::MessageOccurrenceSpecification).getLifeline() <>
          self.receiveEvent.oclAsType(UML::MessageOccurrenceSpecification).getLifeline()
285   else
286     true
287   endif
288 } {
289   assert fatal (self.messageSort <> MessageSort::createMessage) with log
        ("Unsupported Message type: '" + self.toString() + "'");
290   assert fatal (self.messageSort <> MessageSort::deleteMessage) with log
        ("Unsupported Message type: '" + self.toString() + "'");
291   assert fatal (self.messageSort <> MessageSort::asynchSignal) with log ("Unsupported
        Message type: '" + self.toString() + "'");
292   assert warning (self.messageSort = MessageSort::asynchCall) with log ("Unsupported
        Message type: '" + self.toString() + "', processing it as an asynch message");
293
294   var startTransition :=
        self.sendEvent.resolveoneIn(UML::NamedElement::namedElement2transition);
295   var endTransition :=
        self.receiveEvent.resolveoneIn(UML::NamedElement::namedElement2transition);
296   var place1 := object PNML::Place {
297     containerPage := resolveoneIn(UML::NamedElement::model2page);
298     id := createRandomUniqueId();
299     name := object PNML::Name {
300       text := self.name + "_inbox";
301     };
302   };
303   var place2 := object PNML::Place {
304     containerPage := resolveoneIn(UML::NamedElement::model2page);
305     id := createRandomUniqueId();
306     name := object PNML::Name {
307       text := self.name + "_outbox";
```

Copyright © 2016, DICE consortium – All rights reserved

```
308     };
309   };
310   var transition := self.map message2transition();
311
312   map arc(startTransition, place1);
313   map arc(place1, transition);
314   map arc(transition, place2);
315   map arc(place2, endTransition)
316 }
317
318
319 mapping UML::NamedElement::namedElement2place() : PNML::Place {
320   containerPage := resolveoneIn(UML::NamedElement::model2page);
321   id := createRandomUniqueId();
322   if (self.name.oclIsUndefined().not()) {
323     name := object PNML::Name {
324       text := self.name;
325     };
326   };
327 }
328
329 mapping UML::NamedElement::namedElement2transition() : PNML::Transition {
330   containerPage := resolveoneIn(UML::NamedElement::model2page);
331   id := createRandomUniqueId();
332   if (self.name.oclIsUndefined().not()) {
333     name := object PNML::Name {
334       text := self.name;
335     };
336   };
337 }
338
339 /**
340   Transform an ExecutionSpecification into a Transition and
341   creates any additional ToolInfo depending on the InteractionFragment
342   subtype (e.g., ExecutionSpecifications with hostDemand may create
343   exponential transitions)
344 */
345 mapping UML::ExecutionSpecification::executionSpecification2transition() :
      PNML::Transition {
346   init {
347     result := self.map namedElement2transition();
348   }
349   toolspecifics += self[ExecutionSpecification].map executionSpecification2toolInfo();
350   if (toolspecifics->notEmpty()) {
351     // This is a timed ExecutionSpecification, there should not be any event between
          the start and the end
352     // However, self messages (which can be used to specify that the execution is
          self-requested), may be declared between the start and the end events.
353     do {
354       assert fatal (self.covered->size() = 1) with log ("Element '" + self.toString()
            + "' is not covered by a single lifeline");
355       var lifeline := self.covered![UML::Lifeline];
356       var current := self.next(lifeline);
357       while (current <> self.finish and current.oclIsUndefined().not()) {
358         assert fatal (current.oclAsType(
              UML::MessageOccurrenceSpecification).message.isSelfMessage())
359           with log ("A timed ExecutionSpecification cannot have events between its
                start and its finish events '" + self.toString() + "'");
360         current := current.next(lifeline);
```

Copyright © 2016, DICE consortium – All rights reserved

```
361       };
362     };
363   }
364 }
365
366 /**
367   Transform a Message into a Transition and creates any additional ToolInfo (e.g.,
368   Messages with hostDemand may create exponential transitions)
369 */
370 mapping UML::Message::message2transition() : PNML::Transition {
371   containerPage := resolveoneIn(UML::NamedElement::model2page);
372   id := createRandomUniqueId();
373   if (self.name.oclIsUndefined().not()) {
374     name := object PNML::Name {
375       text := self.name;
376     };
377   };
378   toolspecifics += self.map message2toolInfo();
379 }
380
381 /**
382   Transforms an ExecutionSpecification with a hostDemand annotation to a ToolInfo
        element
383 */
384 mapping UML::ExecutionSpecification::executionSpecification2toolInfo() : List (
      PNML::ToolInfo )
385 when {
386     self.getGaStep_hostDemand().oclIsUndefined().not();
387 }{
388   var hostDemand := self.getGaStep_hostDemand();
389   result += expTransitionToolInfo( 1 / hostDemand.value());
390   result += infServerTransitionToolInfo();
391 }
392
393 /**
394   Transforms an Message with and annotation to a ToolInfo element
395 */
396 mapping UML::Message::message2toolInfo() : List ( PNML::ToolInfo ) disjuncts
397 UML::Message::messageGaStepHostDemand2toolInfo,
398 UML::Message::messageGaCommStepHostDemand2toolInfo
399 {};
400
401 /**
402   Transforms a Message with a GaStep.hostDemand annotation to a ToolInfo element
403 */
404 mapping UML::Message::messageGaStepHostDemand2toolInfo() : List ( PNML::ToolInfo )
405 when {
406     self.getGaStep_hostDemand().oclIsUndefined().not();
407 }{
408   var hostDemand := self.getGaStep_hostDemand();
409   result += expTransitionToolInfo( 1 / hostDemand.value());
410   result += infServerTransitionToolInfo();
411 }
412
413 /**
414   Transforms a Message with a GaCommStep.hostDemand annotation to a ToolInfo element
415 */
416 mapping UML::Message::messageGaCommStepHostDemand2toolInfo() : List ( PNML::ToolInfo )
417 when {
```

Copyright © 2016, DICE consortium – All rights reserved

```
418     self.getGaCommStep_hostDemand().oclIsUndefined().not();
419 }{
420   var hostDemand := self.getGaCommStep_hostDemand();
421   result += expTransitionToolInfo( 1 / hostDemand.value());
422   result += infServerTransitionToolInfo();
423 }
424
425
426
427 /**
428   Creates an Arc from 'src' to 'tgt'
429 */
430 mapping arc(in src : PNML::Node, in tgt : PNML::Node) : PNML::Arc {
431   assert warning (src.oclIsUndefined().not()) with log ("Creating an arc without a
          source");
432   assert warning (tgt.oclIsUndefined().not()) with log ("Creating an arc without a
          target");
433   containerPage := resolveoneIn(UML::NamedElement::model2page);
434   id := createRandomUniqueId();
435   source := src;
436   target := tgt;
437 }
438
439 /*****************************************************************************
440   Traceability mappings
441 *****************************************************************************/
442
443 mapping OclAny::trace(to : OclAny) : TRACE::Trace {
444   init {
445     result := object TRACE::Trace {
446       fromDomainElement := self.eObject();
447       toAnalyzableElement := to.eObject();
448     }
449   }
450 }
451
452 mapping OclAny::trace(to : OclAny, text : String) : TRACE::Trace {
453   init {
454     result := object TRACE::Trace {
455       fromDomainElement := self.eObject();
456       toAnalyzableElement := to.eObject();
457       rule := text;
458     }
459   }
460 }
461
462 /*****************************************************************************
463   Navigation helpers
464   Helpers on domains are only valid in the context of a transformations and
465   cannot be moved to a library
466 *****************************************************************************/
467
468 helper UML::scenario() : UML::Interaction {
469   // When running the transformation from the simulation tool, the UML domain must
470   // contain a single activity at its root
471   assert warning (self.rootObjects()[UML::Interaction]->size() = 1) with log ("No
          single Interaction instance was found at the root of the UML input model, trying
          to use the first Interaction in the model instead");
472
```

Copyright © 2016, DICE consortium – All rights reserved

```
473   if (self.rootObjects()[UML::Interaction]->isEmpty().not()) {
474     return self.rootObjects()![UML::Interaction];
475   };
476   // This execution path is useful when running the transformation at development time
477   return self.objectsOfType(UML::Interaction)![UML::Interaction];
478 }
479
480 helper TYPES::vars() : Set ( PrimitiveVariableAssignment ) {
481   return self.rootObjects()[PrimitiveVariableAssignment];
482 }
483
484 helper TYPES::PrimitiveVariableAssignment::asDict() : Dict(String, Real) {
485   var vars : Dict (String, Real) := Dict {};
486   self->forEach(assignment) {
487     vars->put(assignment.variable, assignment.value.toString().toReal());
488   };
489   return vars;
490 }
491
492 helper TRACE::set() : TRACE::TraceSet {
493   return self.rootObjects()![TRACE::TraceSet];
494 }
495
496 helper UML::OccurrenceSpecification::getLifeline() : UML::Lifeline
497 {
498   -- As declared in the standard, the 'covered' association end for
499   -- 'OccurrenceSpecification'' is redefined, and the multiplicity is [1..1]
500   return self.covered![UML::Lifeline];
501 }
502
503 helper UML::Lifeline::fragments() : OrderedSet ( UML::InteractionFragment ) {
504   return self.interaction.fragment[covered->includes(self)];
505 }
506
507 /**
508   Returns the element that precedes the self InteractionFragment in the given
        UML::Lifeline
509 */
510 helper UML::InteractionFragment::prev(lifeline : UML::Lifeline) :
      UML::InteractionFragment {
511   switch {
512     case (self.namespace.oclIsKindOf(UML::Interaction)) {
513       var namespace := self.namespace![UML::Interaction];
514       var lifelineFragments := namespace.fragment[covered->includes(lifeline)];
515       var index := lifelineFragments->indexOf(self);
516       return lifelineFragments->at(index - 1);
517     }
518     case (self.namespace.oclIsKindOf(UML::InteractionOperand)) {
519       var namespace : UML::InteractionOperand :=
            self.namespace![UML::InteractionOperand];
520       var operandFragments := namespace.fragment[covered->includes(lifeline)];
521       if (operandFragments->first() = self) {
522         return namespace.owner![UML::CombinedFragment].prev(lifeline);
523       } else {
524         var index := operandFragments->indexOf(self);
525         return operandFragments->at(index - 1);
526       }
527     }
528   };
```

Copyright © 2016, DICE consortium – All rights reserved

```
529   assert fatal (false) with log ("Unknow namespace Kind for '" + self.toString() +
          "'");
530   return null;
531 }
532
533 /**
534   Returns the element that follows the self InteractionFragment in the given
          UML::Lifeline
535 */
536 helper UML::InteractionFragment::next(lifeline : UML::Lifeline) :
        UML::InteractionFragment {
537   switch {
538     case (self.namespace.oclIsKindOf(UML::Interaction)) {
539       var namespace := self.namespace![UML::Interaction];
540       var lifelineFragments := namespace.fragment[covered->includes(lifeline)];
541       var index := lifelineFragments->indexOf(self);
542       return lifelineFragments->at(index + 1);
543     }
544     case (self.namespace.oclIsKindOf(UML::InteractionOperand)) {
545       var namespace := self.namespace![UML::InteractionOperand];
546       var operandFragments := namespace.fragment[covered->includes(lifeline)];
547       if (operandFragments->last() = self) {
548         return namespace.owner![UML::CombinedFragment].next(lifeline);
549       } else {
550         var index := operandFragments->indexOf(self);
551         return operandFragments->at(index + 1);
552       }
553     }
554   };
555   return null;
556 }
557
558 helper UML::Lifeline::isActor() : Boolean
559 {
560   var first := sd.scenario().fragment->asOrderedSet()->first().oclAsType(
561   UML::MessageOccurrenceSpecification);
562   var last := sd.scenario().fragment->asOrderedSet()->last().oclAsType(
563   UML::MessageOccurrenceSpecification);
564
565   return first.getLifeline() = self and last.getLifeline() = self;
566 }
567
568 helper UML::Message::isSelfMessage() : Boolean
569 {
570   assert fatal (self.sendEvent.oclIsKindOf(UML::MessageOccurrenceSpecification)) with
          log ("Unexpected MessageEnd type: '" + self.sendEvent.toString() + "'");
571   assert fatal (self.receiveEvent.oclIsKindOf(UML::MessageOccurrenceSpecification))
          with log ("Unexpected MessageEnd type: '" + self.receiveEvent.toString() + "'");
572   return self.sendEvent.oclAsType(UML::MessageOccurrenceSpecification).covered =
          self.receiveEvent.oclAsType(UML::MessageOccurrenceSpecification).covered;
573 }
574
575 /*
576 helper OrderedSet(UML::InteractionFragment)::prev(fragment :
        UML::InteractionFragment) : UML::InteractionFragment
577 {
578   var current := self->indexOf(fragment);
579   return self->at(current - 1);
580 }
```

Copyright © 2016, DICE consortium – All rights reserved

```
581
582 helper OrderedSet(UML::InteractionFragment)::next(fragment :
        UML::InteractionFragment) : UML::InteractionFragment
583 {
584   var current := self->indexOf(fragment);
585   return self->at(current + 1);
586 }
587 */
588 /*******************************************************************************
589   Intermediate classes
590   Sadly, intermediate classes cannot be shared among libraries or
591   transformations.
592 *******************************************************************************/
593
594 intermediate class ArrivalPattern {
595   _rawExpression : String;
596 }
597
598 intermediate class ClosedPattern extends ArrivalPattern {
599   population_ : NFP_Integer;
600   extDelay : NFP_Real;
601 }
602
603 intermediate class OpenPattern extends ArrivalPattern {
604   interArrivalTime : NFP_Duration;
605   arrivalRate : NFP_Frequency;
606   arrivalProcess : String;
607 }
608
609 intermediate class NFP_CommonType {
610   _rawExpression : String;
611   expr : String;
612   source : String;
613   statQ : String;
614   dir : String;
615   mode : String;
616 }
617
618 intermediate class NFP_Integer extends NFP_CommonType {
619   value : Integer;
620 }
621
622 intermediate class NFP_Real extends NFP_CommonType{
623   value : Real;
624 }
625
626 intermediate class NFP_Duration extends NFP_Real {
627   unit : String;
628   clock : String;
629   precision : Real;
630   worst : Real;
631   best : Real;
632 }
633
634 intermediate class NFP_Frequency extends NFP_Real {
635   unit : String;
636   precision : Real;
637 }
638
```

Copyright © 2016, DICE consortium – All rights reserved

```
/*
intermediate class CombinedFragmentStart extends UML::InteractionFragment {
  combinedFragment : UML::CombinedFragment;
  coveredLifeline : UML::Lifeline;
}

intermediate class CombinedFragmentEnd extends UML::InteractionFragment {
  combinedFragment : UML::CombinedFragment;
  coveredLifeline : UML::Lifeline;
}
*/

/*******************************************************************************
  Tagged values utilities
*******************************************************************************/

/**
  Helper that parses a VSL tuple containing a NFP_CommonType
*/
helper String::toNfpCommonType() : NFP_CommonType {
  var res := object NFP_CommonType {
    _rawExpression := self;
    statQ := null;
    expr := null;
    source := null;
    dir := null;
    mode := null;
  };
  if (self.isTuple()) {
    var entries := self.asTuple();
    res.expr := entries->get("expr");
    res.statQ := entries->get("statQ");
    res.source := entries->get("source");
    res.dir := entries->get("dir");
    res.mode := entries->get("mode");
  } else {
    res.expr := self;
  };
  assert warning (res.statQ.oclIsUndefined() or res.statQ = 'mean')
    with log ("Expression '" + self + "' defines an unknown 'statQ' value, expected
        empty or 'mean'");
  assert warning (res.source.oclIsUndefined() or res.source = 'est' or res.source =
      'meas')
    with log ("Expression '" + self + "' defines an unsupported 'source' for an input
        parameter, expected 'est' or 'meas'.");
  assert warning (res.dir.oclIsUndefined())
    with log ("Expression '" + self + "' defines a value for the unsupported 'dir'
        property");
  assert warning (res.mode.oclIsUndefined())
    with log ("Expression '" + self + "' defines a value for the unsupported 'mode'
        property");
  return res;
}

/**
  Helper that parses a VSL tuple containing a NFP_Integer
*/
helper String::toNfpInteger() : NFP_Integer {
  var nfp := self.toNfpCommonType();
```

Copyright © 2016, DICE consortium – All rights reserved

```
693   var res := object NFP_Integer {
694     _rawExpression := nfp._rawExpression;
695     expr := nfp.expr;
696     statQ := nfp.statQ;
697     source := nfp.source;
698     dir := nfp.dir;
699     mode := nfp.mode;
700     value := null;
701   };
702   if (self.isTuple()) {
703     var entries := self.asTuple();
704     res.value := entries->get("value").toInteger();
705   };
706   assert fatal (res.value.oclIsUndefined() xor res.expr.oclIsUndefined())
707     with log ("Expression '" + self + "' must define either a valid 'value' or a valid
          'expr'");
708   return res;
709 }
710
711 /**
712   Helper that parses a VSL tuple containing a NFP_Real
713 */
714 helper String::toNfpReal() : NFP_Real {
715   var nfp := self.toNfpCommonType();
716   var res := object NFP_Real {
717     _rawExpression := nfp._rawExpression;
718     expr := nfp.expr;
719     statQ := nfp.statQ;
720     source := nfp.source;
721     dir := nfp.dir;
722     mode := nfp.mode;
723     value := null;
724   };
725   if (self.isTuple()) {
726     var entries := self.asTuple();
727     res.value := entries->get("value").toReal();
728   };
729   assert fatal (res.value.oclIsUndefined() xor res.expr.oclIsUndefined())
730     with log ("Expression '" + self + "' must define either a valid 'value' or a valid
          'expr'");
731   return res;
732 }
733
734
735 /**
736   Helper that parses a VSL tuple containing a NFP_Duration
737 */
738 helper String::toNfpDuration() : NFP_Duration {
739   var nfp := self.toNfpReal();
740   var res := object NFP_Duration {
741     _rawExpression := nfp._rawExpression;
742     value := nfp.value;
743     expr := nfp.expr;
744     statQ := nfp.statQ;
745     source := nfp.source;
746     dir := nfp.dir;
747     mode := nfp.mode;
748     unit := null;
749     clock := null;
```

Copyright © 2016, DICE consortium – All rights reserved

```
750     precision := null;
751     worst := null;
752     best := null;
753   };
754   if (self.isTuple()) {
755     var entries := self.asTuple();
756     res.unit := entries->get("unit");
757     res.clock := entries->get("clock");
758     res.precision := entries->get("precision").toReal();
759     res.worst := entries->get("worst").toReal();
760     res.best := entries->get("best").toReal();
761   };
762   assert warning (res.unit.oclIsUndefined().not())
763     with log ("Expression '" + self + "' does not define a 'unit', assumming the
          default base unit (see complete log)");
764   assert warning (res.dir.oclIsUndefined())
765     with log ("Expression '" + self + "' defines a value for the unsupported 'clock'
          property");
766   assert warning (res.dir.oclIsUndefined())
767     with log ("Expression '" + self + "' defines a value for the unsupported
          'precision' property");
768   assert warning (res.dir.oclIsUndefined())
769     with log ("Expression '" + self + "' defines a value for the unsupported 'worst'
          property");
770   assert warning (res.dir.oclIsUndefined())
771     with log ("Expression '" + self + "' defines a value for the unsupported 'best'
          property");
772
773   return res;
774 }
775
776 /**
777   Helper that parses a VSL tuple containing a NFP_Frequency
778 */
779 helper String::toNfpFrequency() : NFP_Frequency {
780   var nfp := self.toNfpReal();
781   var res := object NFP_Frequency {
782     _rawExpression := nfp._rawExpression;
783     value := nfp.value;
784     expr := nfp.expr;
785     statQ := nfp.statQ;
786     source := nfp.source;
787     dir := nfp.dir;
788     mode := nfp.mode;
789     unit := null;
790     precision := null;
791   };
792   if (self.isTuple()) {
793     var entries := self.asTuple();
794     res.unit := entries->get("unit");
795     res.precision := entries->get("precision").toReal();
796   };
797   assert warning (res.unit.oclIsUndefined().not())
798     with log ("Expression '" + self + "' does not define a 'unit', assumming the
          default base unit (see complete log)");
799   assert warning (res.dir.oclIsUndefined())
800     with log ("Expression '" + self + "' defines a value for the unsupported 'clock'
          property");
801   assert warning (res.dir.oclIsUndefined())
```

Copyright © 2016, DICE consortium – All rights reserved

```
802     with log ("Expression '" + self + "' defines a value for the unsupported
            'precision' property");
803   assert warning (res.dir.oclIsUndefined())
804     with log ("Expression '" + self + "' defines a value for the unsupported 'worst'
            property");
805   assert warning (res.dir.oclIsUndefined())
806     with log ("Expression '" + self + "' defines a value for the unsupported 'best'
            property");
807
808   return res;
809 }
810
811 helper NFP_Integer::value() : Integer {
812   if (self.value.oclIsUndefined().not()) {
813     return self.value;
814   };
815   return self.expr.eval(vars.vars()).toInteger();
816 }
817
818 helper NFP_Real::value() : Real {
819   if (self.value.oclIsUndefined().not()) {
820     return self.value;
821   };
822   return self.expr.eval(vars.vars()).toReal();
823 }
824
825 helper NFP_Duration::value() : Real {
826   var value : Real;
827   if (self.value.oclIsUndefined().not()) {
828     value := self.value;
829   };
830   value := self.expr.eval(vars.vars()).toReal();
831   if (self.unit.oclIsUndefined().not()) {
832     map baseTimeUnit("s");
833     value := value.convert(self.unit, "s");
834   };
835   return value;
836 }
837
838 helper NFP_Frequency::value() : Real {
839   var value : Real;
840   if (self.value.oclIsUndefined().not()) {
841     value := self.value;
842   };
843   value := self.expr.eval(vars.vars()).toReal();
844   if (self.unit.oclIsUndefined().not()) {
845     map baseTimeUnit("s");
846     value := value.convert(self.unit, "Hz");
847   };
848   return value;
849 }
850
851 /*****************************************************************************
852   Getters for tagged values
853 *****************************************************************************/
854
855 helper UML::Element::getGaWorkloadEvent_pattern() : ArrivalPattern {
856   if (self.getGaWorkloadEvent() = null) {
857     return null;
```

Copyright © 2016, DICE consortium – All rights reserved

```
858   };
859   var patternString := self.getValue(self.getGaWorkloadEvent(),
          "pattern").oclAsType(String);
860   var patternName := patternString.key();
861   var patternValue := patternString.value();
862   switch {
863     case (patternName = "closed") {
864       return object ClosedPattern {
865         _rawExpression := patternValue;
866         population_ := patternValue.asTuple()->get("population").toNfpInteger();
867         extDelay := patternValue.asTuple()->get("extDelay").toNfpDuration();
868       };
869     } case (patternString.key() = "open") {
870       return object OpenPattern {
871         _rawExpression := patternValue;
872         interArrivalTime :=
              patternValue.asTuple()->get("interArrivalTime").toNfpDuration();
873         arrivalRate := patternValue.asTuple()->get("arrivalRate").toNfpFrequency();
874         arrivalProcess := patternValue.asTuple()->get("arrivalProcess");
875       };
876     } else {
877       assert fatal (false) with log ("Unknown ArrivalPattern: " + patternString);
878     }
879   };
880   return null;
881 }
882
883 helper UML::Element::getGaStep_hostDemand() : NFP_Duration {
884   if (self.getGaStep() = null) {
885     return null;
886   };
887   var hostDemandStrings := self.getValue(self.getGaStep(),
          "hostDemand").oclAsType(Collection(String));
888   assert warning (hostDemandStrings->size() = 1)
889     with log ("Unexpected number of 'hostDemand' tagged values found, expected 1. "+
890           "Only the first 'mean' value will be used (if found). " +
891           "The context element is '" + self.toString() + "'");
892   return hostDemandStrings.toNfpDuration()->
893       select(demand | demand.statQ.oclIsUndefined() or demand.statQ = 'mean')->
894       asSequence()->first();
895 }
896
897 helper UML::Element::getGaCommStep_hostDemand() : NFP_Duration {
898   if (self.getGaCommStep() = null) {
899     return null;
900   };
901   var hostDemandStrings := self.getValue(self.getGaCommStep(),
          "hostDemand").oclAsType(Collection(String));
902   assert warning (hostDemandStrings->size() = 1)
903     with log ("Unexpected number of 'hostDemand' tagged values found, expected 1. "+
904           "Only the first 'mean' value will be used (if found). " +
905           "The context element is '" + self.toString() + "'");
906   return hostDemandStrings.toNfpDuration()->
907       select(demand | demand.statQ.oclIsUndefined() or demand.statQ = 'mean')->
908       asSequence()->first();
909 }
910
911 helper UML::Element::getGaStep_prob() : NFP_Real {
912   if (self.getGaStep() = null) {
```

Copyright © 2016, DICE consortium – All rights reserved

```
913     return null;
914   };
915   var prob := self.getValue(self.getGaStep(), "prob").oclAsType(String);
916   return prob.toNfpReal();
917 }
918
919 helper UML::Element::getPaLogicalResource_poolSize() : NFP_Integer {
920   if (self.getPaLogicalResource() = null) {
921     return null;
922   };
923   var prob := self.getValue(self.getPaLogicalResource(),
          "poolSize").oclAsType(String);
924   return prob.toNfpInteger();
925 }
926
927 helper UML::Element::getPaRunTInstance_poolSize() : NFP_Integer {
928   if (self.getPaRunTInstance() = null) {
929     return null;
930   };
931   var prob := self.getValue(self.getPaRunTInstance(), "poolSize").oclAsType(String);
932   return prob.toNfpInteger();
933 }
```

Copyright © 2016, DICE consortium – All rights reserved

QVT Helper Functions

Listing 3: Helper Functions

```
import es.unizar.disco.pnml.utils.PnmlDiceUtils;

modeltype UML uses 'http://www.eclipse.org/uml2/5.0.0/UML';
modeltype PNML uses 'http:///ptnet.ecore';
modeltype TRACE uses 'http://es.unizar.disco/simulation/traces/1.0';
modeltype TYPES uses 'http://es.unizar.disco/simulation/datatypes/1.0';
modeltype CONST uses 'http://es.unizar.disco/pnconstants/1.0';
modeltype ECORE uses 'http://www.eclipse.org/emf/2002/Ecore';

library helpers;

/*******************************************************************************
  Types helpers
*******************************************************************************/

helper OclAny::eObject() : ECORE::EObject {
  return self.oclAsType(ECORE::EObject);
}

/*******************************************************************************
  Getters for stereotypes
*******************************************************************************/

helper UML::Element::getGaWorkloadEvent() : UML::Stereotype {
  if (self.isStereotypeApplied(self.getApplicableStereotype(
      "MARTE::MARTE_AnalysisModel::GQAM::GaWorkloadEvent"))) {
    return
        self.getAppliedStereotype("MARTE::MARTE_AnalysisModel::GQAM::GaWorkloadEvent");
  };
  return null;
}

helper UML::Element::getGaStep() : UML::Stereotype {
  if (self.isStereotypeApplied(self.getApplicableStereotype(
      "MARTE::MARTE_AnalysisModel::GQAM::GaStep"))) {
    return self.getAppliedStereotype("MARTE::MARTE_AnalysisModel::GQAM::GaStep");
  };
  return null;
}

helper UML::Element::getGaCommStep() : UML::Stereotype {
  if (self.isStereotypeApplied(self.getApplicableStereotype(
      "MARTE::MARTE_AnalysisModel::GQAM::GaCommStep"))) {
    return self.getAppliedStereotype("MARTE::MARTE_AnalysisModel::GQAM::GaCommStep");
  };
  return null;
}

helper UML::Element::getPaLogicalResource() : UML::Stereotype {
  if (self.isStereotypeApplied(self.getApplicableStereotype(
      "MARTE::MARTE_AnalysisModel::PAM::PaLogicalResource"))) {
    return
        self.getAppliedStereotype("MARTE::MARTE_AnalysisModel::PAM::PaLogicalResource");
  };
  return null;
```

Copyright © 2016, DICE consortium – All rights reserved

```
50 }
51
52 helper UML::Element::getPaRunTInstance() : UML::Stereotype {
53   if (self.isStereotypeApplied(self.getApplicableStereotype(
         "MARTE::MARTE_AnalysisModel::PAM::PaRunTInstance"))) {
54     return
           self.getAppliedStereotype("MARTE::MARTE_AnalysisModel::PAM::PaRunTInstance");
55   };
56   return null;
57 }
58
59
60
61 /*********************************************************************************
62   ToolInfo utilities
63 *********************************************************************************/
64
65 /**
66   Creates the ToolInfo that identifies an exponential timed transition,
67   i.e., CONST::TransitionKind::Exponential
68 */
69 helper expTransitionToolInfo(rate : Real) : PNML::ToolInfo {
70   return object PNML::ToolInfo {
71     tool := CONST::ToolInfoConstants::toolName.toString();
72     version := CONST::ToolInfoConstants::toolVersion.toString();
73     toolInfoGrammarURI := CONST::TransitionKind::Exponential.toString().createURI();
74     formattedXMLBuffer := ("<value grammar=\"" +
           CONST::TransitionKind::Exponential.toString() + "\">" + rate.toString() +
           "</value>").createLongString();
75   };
76 }
77
78 /**
79   Creates the ToolInfo that identifies an InfiniteServer timed transition,
80   i.e., CONST::ServerType::InfiniteServer
81 */
82 helper infServerTransitionToolInfo() : PNML::ToolInfo {
83   return object PNML::ToolInfo {
84     tool := CONST::ToolInfoConstants::toolName.toString();
85     version := CONST::ToolInfoConstants::toolVersion.toString();
86     toolInfoGrammarURI := CONST::ServerType::InfiniteServer.toString().createURI();
87     formattedXMLBuffer := ("<value grammar=\"" +
           CONST::ServerType::InfiniteServer.toString() + "\"/>").createLongString();
88   };
89 }
90
91 /**
92   Creates the ToolInfo that identifies a OneServer timed transition,
93   i.e., CONST::ServerType::OneServer
94 */
95 helper oneServerTransitionToolInfo() : PNML::ToolInfo {
96   return object PNML::ToolInfo {
97     tool := CONST::ToolInfoConstants::toolName.toString();
98     version := CONST::ToolInfoConstants::toolVersion.toString();
99     toolInfoGrammarURI := CONST::ServerType::OneServer.toString().createURI();
100    formattedXMLBuffer := ("<value grammar=\"" +
           CONST::ServerType::OneServer.toString() + "\"/>").createLongString();
101  };
102 }
```

Copyright © 2016, DICE consortium – All rights reserved

```
103
104 /**
105   Creates a ToolInfo that identifies a probabilistic immediate transition,
106   i.e., CONST::TransitionKind::Immediate
107 */
108 helper probTransitionToolInfo(prob: Real) : PNML::ToolInfo {
109   return object PNML::ToolInfo {
110     tool := CONST::ToolInfoConstants::toolName.toString();
111     version := CONST::ToolInfoConstants::toolVersion.toString();
112     toolInfoGrammarURI := CONST::TransitionKind::Immediate.toString().createURI();
113     formattedXMLBuffer := ("<value grammar=\"" +
          CONST::TransitionKind::Immediate.toString() + "\">" + prob.toString() +
          "</value>").createLongString();
114   };
115 }
116
117 /**
118   Creates a ToolInfo for the passed base time unit
119   i.e., CONST::BaseUnitsConstants::baseTimeUnit
120 */
121 helper baseTimeUnitToolInfo(unit : String) : PNML::ToolInfo {
122   return object PNML::ToolInfo {
123     tool := CONST::ToolInfoConstants::toolName.toString();
124     version := CONST::ToolInfoConstants::toolVersion.toString();
125     toolInfoGrammarURI :=
          CONST::BaseUnitsConstants::baseTimeUnit.toString().createURI();
126     formattedXMLBuffer := ("<value grammar=\"" +
          CONST::BaseUnitsConstants::baseTimeUnit.toString() + "\">" + unit +
          "</value>").createLongString();
127   };
128 }
129
130
131 /*****************************************************************************
132   Tagged values utilities
133 *****************************************************************************/
134
135 /**
136   Helper to get the key from a string in the form 'key=value'
137 */
138 helper String::key() : String {
139   assert fatal (self.indexOf("=") <> -1) with log ("Unexpected number of tokens in "
        + self);
140   return self.substringBefore("=").trim()
141
142 }
143
144 /**
145   Helper to get the value from a string in the form 'key=value'
146 */
147 helper String::value() : String {
148   assert fatal (self.indexOf("=") <> -1) with log ("Unexpected number of tokens in "
        + self);
149   return self.substringAfter("=").trim()
150 }
151
152 /**
153   Helper that determines if a given String represents a Tuple
154 */
```

Copyright © 2016, DICE consortium – All rights reserved

```
155 helper String::isTuple() : Boolean {
156   var trimmed := self.trim();
157   return trimmed.startsWith("(") and trimmed.trim().endsWith(")");
158 }
159
160 /**
161   Helper that parses a VSL Tuple and returns a Dictionary
162 */
163 helper String::asTuple() : Dict (String, String) {
164   var trimmed := self.trim();
165   assert warning (trimmed.startsWith("(")) with log ("Tuple string '" + self + "'
        does not start with '('");
166   assert warning (trimmed.trim().endsWith(")")) with log ("Tuple string '" + self +
        "' does not end with ')'");
167
168   var segments : List (String) := List {};
169   var pars : Integer := 0;
170   var segment : String;
171   trimmed.substring(2, trimmed.size() - 1).characters()->forEach(c) {
172     switch {
173       case (c = '(') {
174         pars := pars + 1;
175       } case (c = ')') {
176         pars := pars - 1;
177       } case (c = ',') {
178         if (pars = 0) {
179           segments->add(segment);
180           segment := '';
181           continue;
182         }
183       }
184     };
185     segment := segment.concat(c);
186   };
187   segments->add(segment);
188   var entries : Dict (String, String) := Dict {};
189   segments->forEach(entry) {
190     entries->put(entry.key(), entry.value());
191   };
192   return entries;
193 }
```

Copyright © 2016, DICE consortium – All rights reserved