H Annex H: Precise Semantics of SysML

(informative)

H.1 Overview

This annex defines the precise semantics of the abstract syntax of a subset of SysML stereotypes. This semantic definition is given as an extension to the semantic model for PSCS (see [PSCS], Clause 8), which is itself an extension of the execution model for fUML (see [fUML], Clause 8). This annex includes only the extensions to the PSCS model necessary for SysML. However, the full semantics for SysML are given by the fUML execution model as extended for PSCS, which is then a complete, executable fUML model of the operational semantics for the combined PSCS and SysML subset.

The SysML execution model is given as an extension of the PSCS model in order to ensure that SysML semantics are compatible with PSCS semantics.

The SysML semantics specified by this annex does not depend on PSSM. However it is possible for an execution engine to conform to both PSSM and this specification.

The circularity of defining SysML semantics by extending the fUML execution model, which is itself an fUML model, is handled as it is in fUML. That is, the execution model is defined using only the further subset of fUML whose semantics are separately specified by the fUML base semantics (see [fUML], Clause 10), which is not extended further for the purposes of SysML. This further subset, known as Base UML (or “bUML”) includes a subset of UML activity modeling that is used to specify the detailed behavior of all concrete operations in the execution model. However, rather than using activity diagram notation to represent such activity models, they are specified in the execution model extensions for SysML using the Java-syntax textual notation whose mapping to UML is given in Annex A of [fUML].

The SysML extensions to the PSCS execution model are organized into five packages. Figure H.1 shows each of these packages and their dependencies on packages from the SysML profile and from fUML and PSCS semantic models. These dependencies are represented as package-import relationships, which also make the unqualified names of the necessary syntactic and semantics elements visible for use in the detailed behavioral code of each of the SysML semantics packages.

The subsequent clauses in this annex describe each of the SysML semantics packages in turn. The description includes a class model for the contents of the package and an explanation of the operational semantics defined by the functionality of the classes in the model. Those packages are organizes as follows:

- the "Actions" package specifies additional constraints on the UML::Actions package that restrict the scope of models on which this operational semantics applies. It defines also a set of semantics visitors that extends some from the fUML::Semantics::Actions package according to semantics defined by SysML stereotypes.
- The "Activities" package defines a set of semantics visitors that extends some from the fUML::Semantics::Activities package according to semantics defined by SysML stereotypes.
- The "Blocks" package extends the CS_Object visitor define by the PSCS specification and defines a set of construct that can support the semantics defined by SysML stereotypes from the SysML::Blocks package.
- The "PortsAndFlows" package specifies additional constraints on ports, Flow properties and directed features that restrict the scope of models on which this operational semantics applies.
- The "Loci" package is added for specifying necessary extensions of the Loci package of PSCS together with a set of utility operations that simplify the specification of teh semantics visitors

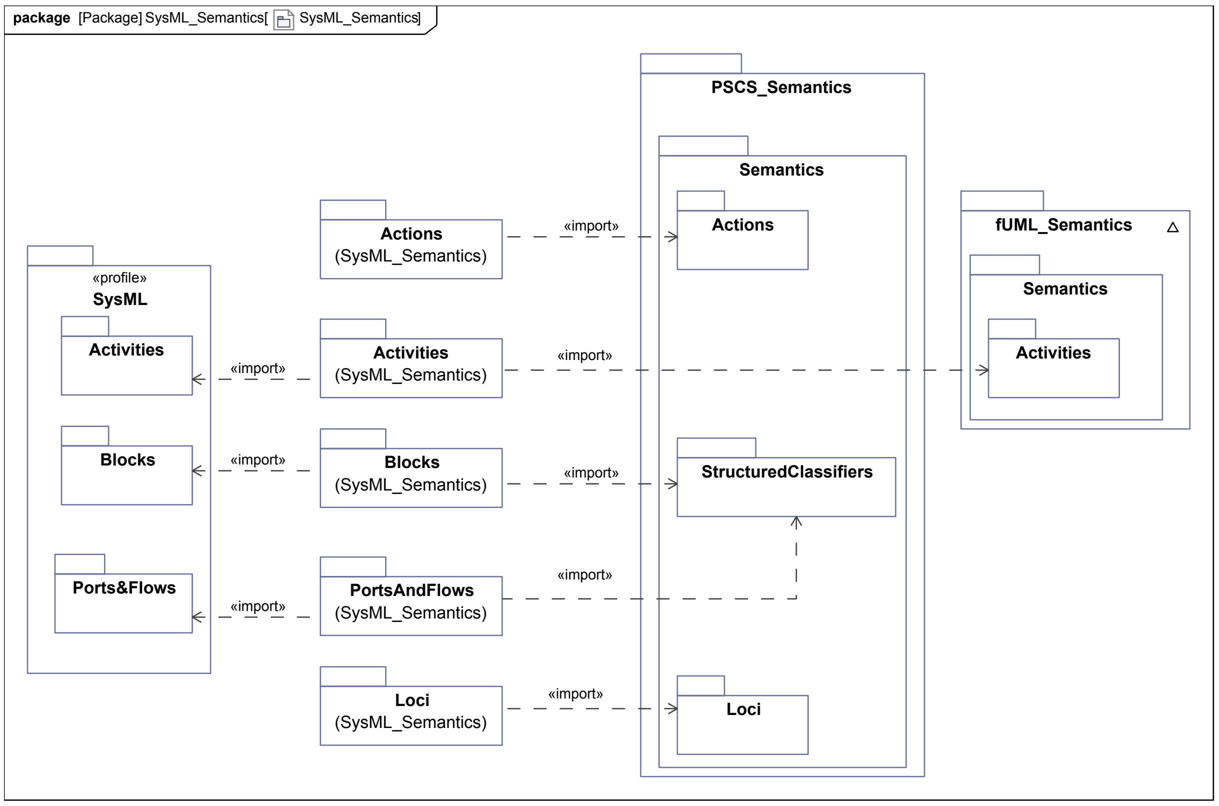


Figure H.1. SysML_Semantics

H.2 References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For versioned references, subsequent amendments to, or revisions of, any of these publications do not apply.

[fUML] Semantics of a Foundational Subset for Executable UML Models (fUML), version 1.5, http://www.omg.org/spec/FUML

[PSCS] Precise Semantics of UML Composite Structures (PSCS), version 1.2, http://www.omg.org/spec/PSCS

[PSSM] Precise Semantics of UML State Machines, version 1.0, https://www.omg.org/spec/PSSM/1.0/PDF

[UML] Unified Modeling Language, version 2.5.1, https://www.omg.org/spec/UML/2.5.1/PDF

H.3 Semantics

This clause is organized in sub-clauses that include this overview and a set of sub-clauses chapter that specifies the structural and behavioral constructs of this specification and/or a sub-clause that defines additional constraints that restrict the scope on which the semantics defined by this specification applies.

Those semantics are defined as an extension of the PSCS semantics that are themselves defined as an extension of fUML. A SysML model that syntactically conforms to this subset shall have an abstract syntax representation that consists solely of instances of metaclasses that are (imported) members of the either the fUML_Syntax::Syntax or the PCSC_Syntax packages, as described in the corresponding specifications. Also only the SysML Stereotypes listed in the sub-clauses below shall be used.

H.3.1 Actions

H.3.1.1 Overview

The Actions package introduces extensions to various fUML action activation classes defined in PSCS or in fUML. SysML does not specify any stereotype for actions. However the semantics of number of SysML stereotypes actually impact the semantics of some actions that are performed on elements those stereotypes are applied on. For instance, binding connectors can link together a pair of properties so that their values shall be the same at any time. The operational consequence of this semantics is that any action modifying the value of one of those properties shall be replicated to the value of the property it is bound to.

H.3.1.2 Additional Constraints

- `upperbound_equal_upper`
  The value of a Pin for its upperbound and upper properties shall be the same

```
context Pin inv: self.upperBound = self.upper
```

H.3.1.3 Class descriptions

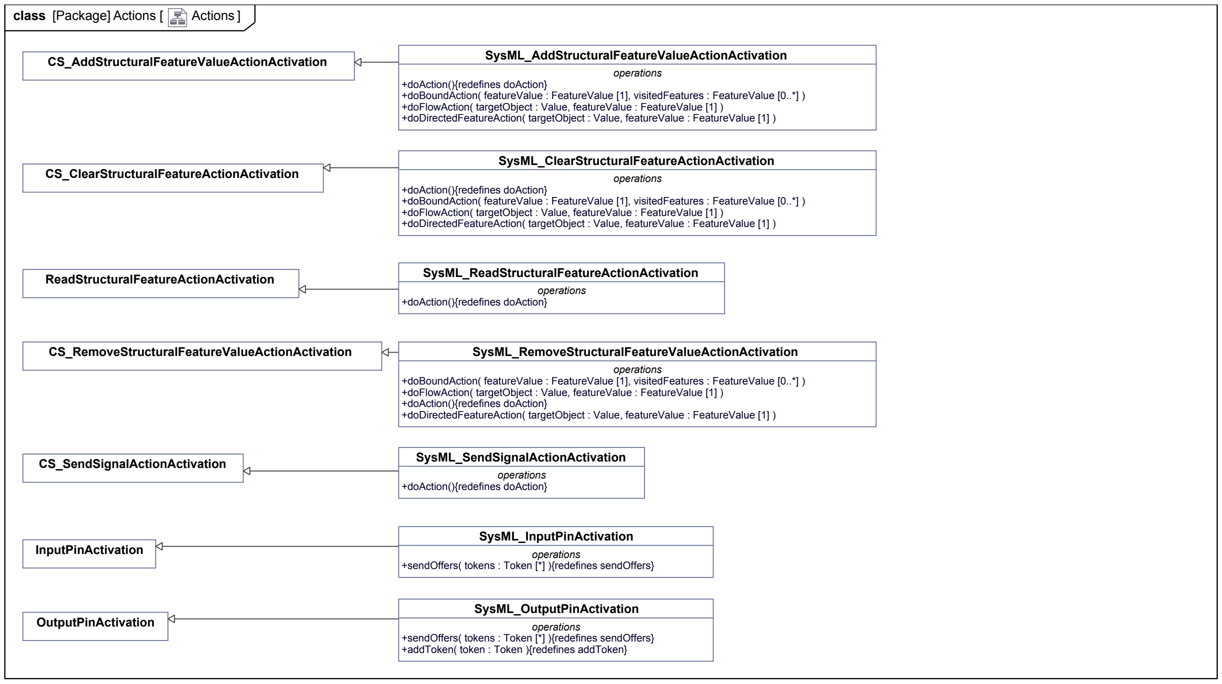

class [Package] Actions [Actions]
CS_AddStructuralFeatureValueActionActivation
SysML_AddStructuralFeatureValueActionActivation
operations
+doAction(){redefines doAction}
+doBoundAction(featureValue : FeatureValue [1], visitedFeatures : FeatureValue [0..*])
+doFlowAction(targetObject : Value, featureValue : FeatureValue [1])
+doDirectedFeatureAction(targetObject : Value, featureValue : FeatureValue [1])
CS_ClearStructuralFeatureActionActivation
SysML_ClearStructuralFeatureActionActivation
operations
+doAction(){redefines doAction}
+doBoundAction(featureValue : FeatureValue [1], visitedFeatures : FeatureValue [0..*])
+doFlowAction(targetObject : Value, featureValue : FeatureValue [1])
+doDirectedFeatureAction(targetObject : Value, featureValue : FeatureValue [1])
ReadStructuralFeatureActionActivation
SysML_ReadStructuralFeatureActionActivation
operations
+doAction(){redefines doAction}
CS_RemoveStructuralFeatureValueActionActivation
SysML_RemoveStructuralFeatureValueActionActivation
operations
+doBoundAction(featureValue : FeatureValue [1], visitedFeatures : FeatureValue [0..*])
+doFlowAction(targetObject : Value, featureValue : FeatureValue [1])
+doAction(){redefines doAction}
+doDirectedFeatureAction(targetObject : Value, featureValue : FeatureValue [1])
CS_SendSignalActionActivation
SysML_SendSignalActionActivation
operations
+doAction(){redefines doAction}
InputPinActivation
SysML_InputPinActivation
operations
+sendOffers(tokens : Token [*]){redefines sendOffers}
OutputPinActivation
SysML_OutputPinActivation
operations
+sendOffers(tokens : Token [*]){redefines sendOffers}
+addToken(token : Token){redefines addToken}

Figure H.2. Actions

H.3.1.3.1 SysML_AddStructuralFeatureValueActionActivation

Description

This semantics visitor extends the PSCS CS_AddStructuralFeatureValueActionActivation class in order to support semantics of binding connectors, flow properties and directed features.

Generalizations

- CS_AddStructuralFeatureValueActionActivation (from Actions)

Operations

- doAction () {redefines doAction}

```
// If the feature has a binding connector attached
// a coordinated update is done
// otherwise, behaves as usual

// get the value of the target pin before the super.doAction() is
called
AddStructuralFeatureValueAction action =
(AddStructuralFeatureValueAction) (this.node);
Value target = this.getToken(action.object).getValue(0);

super.doAction();

StructuralFeature feature = action.structuralFeature;
if (feature instanceof Property & object instanceof StructuredValue) {
  FeatureValue featureValue = action.object.getFeatureValue(feature);

  FeatureValueList visitedFeatures = new FeatureValueList();

  this.doBoundAction(featureValue, visitedFeatures);

  //Flow property management
  this.doFlowAction(target, featureValue);

  //Directed feature management
  this.doDirectedFeatureAction(target, featureValue);
}
```

- doBoundAction (in featureValue : FeatureValue, in visitedFeatures : FeatureValue)

```
// check that this feature value has not been visited yet
// otherwise stop the recursion here
for (int k=0; k < visitedFeature.size(); k++) {
  if (featureValue == visitedFeature.get(k)) {
    return;
  }
}

// add the feature value to the visited list
visitedFeatures.addValue(featureValue);

// retrieve all the bindings for this feature value
SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();
```

```
ValueBindingList bindings = locus.getAllValueBindings(featureValue);

for (int i = 0; i < bindings.size(); i++) {
  // get the feature value bound by this binding
  FeatureValue otherFeatureValue =
bindings.get(i).getOppositeBoundFeatureValue(featureValue);

  // Loop on values...
  for (int j = 0; j < featureValue.values.size(); j++) {

    otherFeatureValue.values = featureValue.values;
  }

  // execute recursively
  doBoundAction(otherFeatureValue, visitedFeatures);
}
```

- doDirectedFeatureAction (in targetObject : Value, in featureValue : FeatureValue)

```
// If the feature is a required feature the value has to be added to
the matched feature, if any
Feature feature = featureValue.feature;
SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

if (feature instanceof Property && locus.isRequiredFeature((Property)
feature) && targetObject instanceof StructuredValue) {

  // retrieve the matching feature value
  FeatureValue matchingFeatureValue =
locus.getMatchingFeatureValue(targetObject, feature);

  if (matchingFeatureValue != null) {
    // Loop on values...
    for (int j = 0; j < featureValue.values.size(); j++) {

      matchingFeatureValue.values = featureValue.values;
    }

    // trigger binding connections, if any
    FeatureValueList visitedFeatures = new FeatureValueList();
    doBoundAction(matchingFeatureValue, visitedFeatures);
  }

}
```

- doFlowAction (in targetObject : Value, in featureValue : FeatureValue)

```
// Looks for the value of the owner of the property,
// i.e. typicaly the value passed to the action
// using its "target" input pin.
// The link to be used connects this "target"
// rather than the feature value itself.
// It is check whether it is a flow property
Feature feature = featureValue.feature;
SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

if (feature instanceof Property &&
  locus.isFlowProperty((Property) feature) &&
  targetObject instanceof StructuredValue) {

  // retrieve the matching feature value
  FeatureValue matchingFeatureValue =
locus.getMatchingFeatureValue(targetObject, feature);

  if (matchingFeatureValue != null) {
    // Loop on values...
    for (int j = 0; j < featureValue.values.size(); j++) {

      matchingFeatureValue.values.get(j) = featureValue.values.get(j);
    }

    // trigger binding connections, if any
    FeatureValueList visitedFeatures = new FeatureValueList();
    doBoundAction(matchingFeatureValue, visitedFeatures);
  }

}
```

H.3.1.3.2 SysML_CallOperationActivation

Description

This semantics visitor extends the PSCS CS_CallOperationActionActivation class in order to support semantics of binding connectors, flow properties and directed features.

Generalizations

- CallOperationActionActivation (from Actions)

Operations

- getCallExecution () : Execution [1] {redefines getCallExecution}

```
// Check whether the operation is a required feature.
// If so, call from the matching feature instead, if any.
// If it is not a required feature, invoke the regular getCallExecution

CallOperationAction action = (CallOperationAction) (this.node);
Value target = this.takeTokens(action.target).getValue(0);
Execution execution = null;

if (action.operation != null) {
  // If the operation is a required feature the matching feature shall
be called
  SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

  if (locus.isRequiredFeature((Property) feature)) {

    // retrieve the matching feature value
    FeatureValue matchingOperation =
locus.getMatchingFeatureValue(target, action.operation);

    target = locus.getObjectWithFeatureValue(matchingOperation);

    execution = ((Reference) target).dispatch(matchingOperation);
  }
}
else {
  execution = super.getCallExecution();
}

return execution;
```

H.3.1.3.3 SysML_ClearStructuralFeatureActionActivation

Description

This semantics visitor extends the PSCS CS_ClearStructuralFeatureActionActivation class in order to support semantics of binding connectors, flow properties and directed features.

Generalizations

- CS_ClearStructuralFeatureActionActivation (from Actions)

Operations

- doAction () {redefines doAction}

```
// If the feature has a binding connector attached
// a coordinated update is done
// otherwise, behaves as usual

// get the value of the target pin before the super.doAction() is
called
ClearStructuralFeatureValueAction action =
(ClearStructuralFeatureValueAction) (this.node);
Value target = this.getToken(action.object).getValue(0);

super.doAction();
```

```
StructuralFeature feature = action.structuralFeature;
if (feature instanceof Property && object instanceof StructuredValue) {
  FeatureValue featureValue = action.object.getFeatureValue(feature);

  FeatureValueList visitedFeatures = new FeatureValueList();

  this.doBoundAction(featureValue, visitedFeatures);

  //Flow property management
  this.doFlowAction(target, featureValue);

  //Directed feature management
  this.doDirectedFeatureAction(target, featureValue);

}
```

- doBoundAction (in featureValue : FeatureValue, in visitedFeatures : FeatureValue)

```
// Check that this feature value has not been visited yet
// otherwise stop the recursion here
for (int k=0; k < visitedFeature.size(); k++) {
  if (featureValue == visitedFeature.get(k)) {
    return;
  }
}

// add the feature value to the visited list
visitedFeatures.addValue(featureValue);

// retrieve all the bindings for this feature value
SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();
ValueBindingList bindings = locus.getAllValueBindings(featureValue);

for (int i = 0; i < bindings.size(); i++) {
  // get the feature value bound by this binding
  FeatureValue otherFeatureValue =
bindings.get(i).getOppositeBoundFeatureValue(featureValue);

  otherFeatureValue.values = new ValueList();

  // execute recursively
  doBoundAction(otherFeatureValue, visitedFeatures);
}
```

- doDirectedFeatureAction (in targetObject : Value, in featureValue : FeatureValue)

```
// If the feature is a required feature the values of the matched
feature, if any,
// have to be cleared
Feature feature = featureValue.feature;
```

```
SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

if (feature instanceof Property && locus.isRequiredFeature((Property)
feature) && targetObject instanceof StructuredValue) {

  // retrieve the matching feature value
  FeatureValue matchingFeatureValue =
locus.getMatchingFeatureValue(targetObject, feature);

  if (matchingFeatureValue != null) {
    matchingFeatureValue.values = new ValueList();

    // trigger binding connections, if any
    FeatureValueList visitedFeatures = new FeatureValueList();
    doBoundAction(matchingFeatureValue, visitedFeatures);
  }

}
```

- doFlowAction (in targetObject : Value, in featureValue : FeatureValue)

```
// Get the value of the owner of the property,
// i.e. typicaly the value passed to the action using its "target"
input pin.
// The link to be used connects this "target" rather than the feature
value itself.
// check whether this is a flow property
Feature feature = featureValue.feature;
SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

if (feature instanceof Property &&
  locus.isFlowProperty((Property) feature) &&
  targetObject instanceof StructuredValue) {

  // retrieve the matching feature value
  FeatureValue matchingFeatureValue =
locus.getMatchingFeatureValue(targetObject, feature);

  if (matchingFeatureValue != null) {
    matchingFeatureValue.values = new ValueList();

    // trigger binding connections, if any
    FeatureValueList visitedFeatures = new FeatureValueList();
    doBoundAction(matchingFeatureValue, visitedFeatures);
  }

}
```

H.3.1.3.4 SysML_InputPinActivation

Description

This semantics visitor extends the fUML InputPinActivation class in order to support semantics of the NoBuffer stereotype.

Generalizations

- InputPinActivation (from Actions)

Operations

- sendOffers (in tokens : Token) {redefines sendOffers}

```
// call the original sendOffer operation
// then, if the NoBuffer stereotype is applied,
// discard remaining tokens, if any

super.sendOffers(tokens);

ObjectNode node = (ObjectNode) this.node;

if ( node.owner instanceof StructuredActivityNode) {
  SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

  if (locus.isNoBuffer(node)) {
    this.clearToken();
  }
}
```

H.3.1.3.5 SysML_OutputPinActivation

Description

This semantics visitor extends the fUML OutputPinActivation class in order to support semantics of both the NoBuffer and the Overwrite stereotypes.

Generalizations

- OutputPinActivation (from Actions)

Operations

- addToken (in token : Token) {redefines addToken}

```
// if the Overwrite stereotype is applied and the node holds at least
one token,
// remove the "oldest" token in the list,
// depending on the node ordering
// then call the original addToken operation

ObjectNode node = (ObjectNode) this.node;
```

```
SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

if (locus.isOverwrite(node) && his.heldTokens.size() > 0) {
  //this.clearToken();
  if (node.ordering == ObjectNodeOrderingKind::FIFO) {
    this.heldTokens.remove(0);
  }
  else {
    if (node.ordering == ObjectNodeOrderingKind::LIFO) {
      this.heldTokens.remove(this.heldTokens.size()-1);
    }
  }
}

super.addToken(tokens);
```

- sendOffers (in tokens : Token) {redefines sendOffers}

```
// call the original sendOffer operation
// then, if the NoBuffer stereotype is applied,
// discard remaining tokens, if any

super.sendOffers(tokens);

ObjectNode node = (ObjectNode) this.node;
SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

if (locus.isNoBuffer(node)) {
  this.clearToken();
}
```

H.3.1.3.6 SysML_ReadStructuralFeatureActionActivation

Description

This semantics visitor extends the fUML ReadStructuralFeatureActionActivation class in order to support semantics of required directed features.

Generalizations

- ReadStructuralFeatureActionActivation (from Actions)

Operations

- doAction () {redefines doAction}

```
// Check whether the feature is a required feature
// if so, get the value from a matching feature, if any.
// If it is not a required feature, invoke the regular doACtion
```

```
ReadStructuralFeatureAction action = (ReadStructuralFeatureAction)
(this.node);
StructuralFeature feature = action.structuralFeature;

if (feature != null && action.object instanceof StructuredValue) {

  // If the feature is a required feature,
  // the values of the matched feature, if any, have to be cleared
  SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

  if (locus.isRequiredFeature((Property) feature)) {

    // retrieve the matching feature value
    FeatureValue matchingFeatureValue =
locus.getMatchingFeatureValue(targetObject, feature);

    if (matchingFeatureValue != null) {
      matchingFeatureValue.values = new ValueList();
      this.putTokens(action.result, matchingFeatureValue.values);
    }

  }
  else {
    super.doAction();
  }
}
```

H.3.1.3.7 SysML_RemoveStructuralFeatureValueActionActivation

Description

This semantics visitor extends the PSCS CS_RemoveStructuralFeatureValueActionActivation class in order to support semantics of binding connectors, flow properties and directed features.

Generalizations

- CS_RemoveStructuralFeatureValueActionActivation (from Actions)

Operations

- doAction () {redefines doAction}

```
// If the feature has a binding connector attached
// a coordinated update is done
// otherwise, behaves as usual

// get the value of the target pin before the super.doAction() is
called
RemoveStructuralFeatureValueAction action =
(RemoveStructuralFeatureValueAction) (this.node);
```

```
Value target = this.getToken(action.object).getValue(0);

super.doAction();

StructuralFeature feature = action.structuralFeature;
if (feature instanceof Property && object instanceof StructuredValue) {
  FeatureValue featureValue = action.object.getFeatureValue(feature);

  FeatureValueList visitedFeatures = new FeatureValueList();

  this.doBoundAction(featureValue, visitedFeatures);

  //Flow property management
  this.doFlowAction(target, featureValue);

  //Directed feature management
  this.doDirectedFeatureAction(target, featureValue);


}
```

- doBoundAction (in featureValue : FeatureValue, in visitedFeatures : FeatureValue)

```
// check that this feature value has not been visited yet
// otherwise stop the recursion here
for (int k=0; k < visitedFeature.size(); k++) {
  if (featureValue == visitedFeature.get(k)) {
    return;
  }
}

// add the feature value to the visited list
visitedFeatures.addValue(featureValue);

// retrieve all the bindings for this feature value
SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();
ValueBindingList bindings = locus.getAllValueBindings(featureValue);

for (int i = 0; i < bindings.size(); i++) {
  // get the feature value bound by this binding
  FeatureValue otherFeatureValue =
bindings.get(i).getOppositeBoundFeatureValue(featureValue);

  // Loop on values...
  otherFeatureValue.values = new ValueList()
  for (int j = 0; j < featureValue.values.size(); j++) {

    otherFeatureValue.values.get(j) = featureValue.values.get(j);
  }
```

```
    // execute recursively
    doBoundAction(otherFeatureValue, visitedFeatures);
  }
```

- doDirectedFeatureAction (in targetObject : Value, in featureValue : FeatureValue)

```
  // If the feature is a required feature the values of the matched
  feature, if any,
  // have to be cleared
  Feature feature = featureValue.feature;
  SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

  if (feature instanceof Property &&
    locus.isRequiredFeature((Property) feature) &&
    targetObject instanceof StructuredValue) {

    // retrieve the matching feature value
    FeatureValue matchingFeatureValue =
  locus.getMatchingFeatureValue(targetObject, feature);

    if (matchingFeatureValue != null) {
      matchingFeatureValue.values = matchingFeatureValue.values;

      // trigger binding connections, if any
      FeatureValueList visitedFeatures = new FeatureValueList();
      doBoundAction(matchingFeatureValue, visitedFeatures);
    }

  }
```

- doFlowAction (in targetObject : Value, in featureValue : FeatureValue)

```
  // Get the value of the owner of the property, i.e. typicaly the value
  passed to teh action using its "target" input pin
  // the link to be used will connect this "target" rather than the
  feature value itself.
  //check whether this is a flow property
  Feature feature = featureValue.feature;
  SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

  if (feature instanceof Property && locus.isFlowProperty((Property)
  feature) && targetObject instanceof StructuredValue) {

    // retrieve the matching feature value
    FeatureValue matchingFeatureValue =
  locus.getMatchingFeatureValue(targetObject, feature);

    if (matchingFeatureValue != null) {
      matchingFeatureValue.values = featureValue.values;
    }
```

```
      // trigger binding connections, if any
      FeatureValueList visitedFeatures = new FeatureValueList();
      doBoundAction(matchingFeatureValue, visitedFeatures);
    }

  }
```

H.3.1.3.8 SysML_SendSignalActionActivation

Description

This semantics visitor extends the PSCS CS_RemoveStructuralFeatureValueActionActivation class in order to support semantics of proxy ports. Note: the final target of the Signal shall be have Reception for this Signal in order to trigger a behavior when the signal occurrence is received.

Generalizations

- CS_SendSignalActionActivation (from Actions)

Operations

- doAction () {redefines doAction}

```
// If onPort is not specified, behaves like in fUML/PSCS
// If onPort is specified:
//  - if it is a behavior port,
//    get the value from the onPort pin.
//  - else (i.e. if it is not a behavior port),
//  get the value from the target pin.
// If the value is not a reference then do nothing.
// Otherwise, looks for all links connected to the referenced object
// if links are found, construct a signal using the values from the
argument pins
// and send it to the referenced object on the opposite side of each of
those links

SendSignalAction action = (SendSignalAction) (this.node);
Port port = action.getOnPort();

if (port == null) {
  // Behaves like in fUML
  super.doAction();
} else {

FeatureValueList actualTargets;

// Get all links (available at the locus of this object) that are
attached to this port
// (i.e. the port is an end such links)
// and get their opposite ends as actual targets
// Note: SysML links are binary
ExtensionalValueList extensionalValues = this.locus.extensionalValues ;
Integer i = 1 ;
while (i <= extensionalValues.size()) {
  ExtensionalValue value = extensionalValues.getValue(i-1) ;
  if (value instanceof CS_Link) {
    CS_Link link = (CS_Link)value;
    if (link.getFeatureValues.size() > 1) {
      if (link.getFeatureValues.get(0).feature == port) {
        actualTargets.addValue(link.getFeatureValues.get(1));
      }
      else {
        if (link.getFeatureValues.get(1).feature == port) {
          actualTargets.addValue(link.getFeatureValues.get(0));
        }
      }
    }
  }
  i = i + 1 ;
}

// Send the a signal instance to all the targets identified that are
CS_References
for (int j=0; j < actualTargets.size(); j++) {

  Value target = actualTargets.get(j).value;

  if (target instanceof CS_Reference) {
    // Constructs the signal instance
    Signal signal = action.getSignal();
    SignalInstance signalInstance = new SignalInstance();
    signalInstance.type = signal;

    List attributes = signal.getOwnedAttributes();
    List argumentPins = action.getArguments();
    Integer j = 0;
    while (j < attributes.size()) {
      Property attribute = attributes.get(j);
      InputPin argumentPin = argumentPins.get(j);
      List values = this.takeTokens(argumentPin);
      signalInstance.setFeatureValue(attribute, values, 0);
      j = j + 1;
    }

    CS_Reference targetReference = (CS_Reference) target;
    targetReference.send(signalInstance);
  }
}
```

H.3.2 Activities

H.3.2.1 Overview

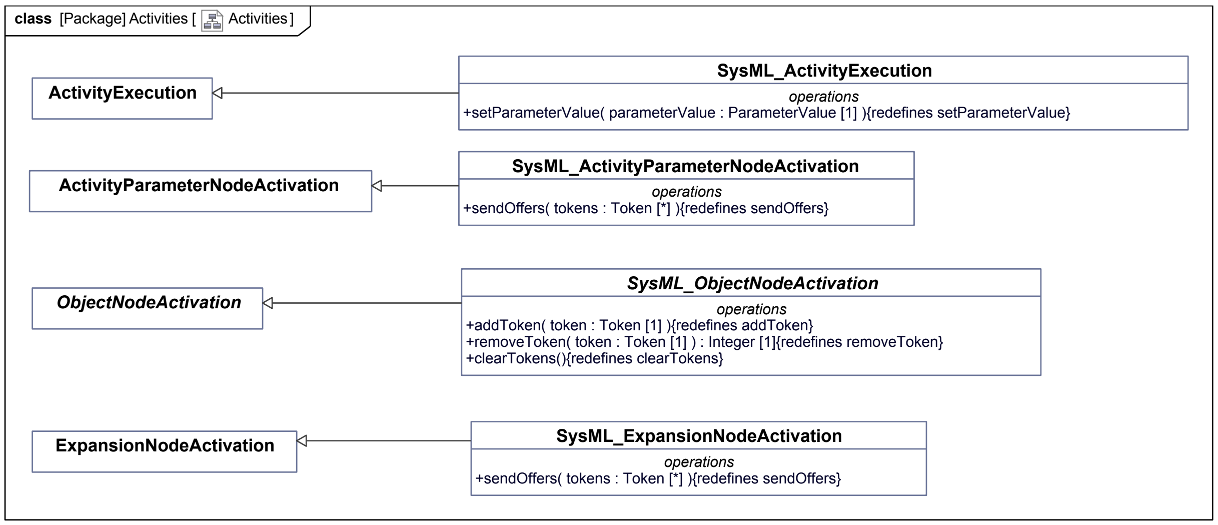
This sub-clause addresses the semantics of both the NoBuffer and the Overwrite stereotypes from the Activities package of SysML. The fact that fUML does not includes foundational semantics for time prevent from describing those for the stereotypes Rate, Discrete and Continuous. Also the way the fUML execution model is built would not make it possible to describe the semantics of ControlOperator without an in-deep revision. The semantics of the Optional stereotype is redundant with that of the multiplicity lower bound and so, already handled in fUML. The semantics of Probability have no direct impact on the model execution even if it can be exploited by analysis tools.

The semantics of NoBuffer, is described in the extensions of both InputPinActivation and OutputPinActivation. Their sendOffers operations is redefined so that remaining tokens are removed if the NoBuffer stereotype is applied. The same extension is done for ActivityParameterNodeActivation but will be effective only for Input parameter nodes.

With the Overwrite stereotype applied on an ObjectNode, a conforming execution engine shall replace tokens stored in a "full" object node by incoming tokens. "Full" means that the number of tokens held within the node is equal to the value of its upperBound property. The tokens to be removed depend whether it has a FIFO or a LIFO ordering. This is supported by the redefinition of the addToken() operation in the SysML_OutputPinActivation. It shall also be done for InputPin, CentralBuffer, and activity parameter nodes (Datastore already has an overwrite semantics).

SysML Stereotypes Supported: NoBuffer, Overwrite

H.3.2.2 Class descriptions



class [Package] Activities [Activities]
ActivityExecution
SysML_ActivityExecution
operations
+setParameterValue(parameterValue : ParameterValue [1]){redefines setParameterValue}
ActivityParameterNodeActivation
SysML_ActivityParameterNodeActivation
operations
+sendOffers(tokens : Token [*]){redefines sendOffers}
ObjectNodeActivation
SysML_ObjectNodeActivation
operations
+addToken(token : Token [1]){redefines addToken}
+removeToken(token : Token [1]) : Integer [1]{redefines removeToken}
+clearTokens(){redefines clearTokens}
ExpansionNodeActivation
SysML_ExpansionNodeActivation
operations
+sendOffers(tokens : Token [*]){redefines sendOffers}

Figure H.3. Activities

H.3.2.2.1 SysML_ActivityExecution

Description

This semantics visitor extends the fUML ActivityExecution class in order to support semantics of adjunct properties.

Generalizations

- ActivityExecution (from Activities)

Operations

- setParameterValue (in parameterValue : ParameterValue) {redefines setParameterValue}

```
// Call the regular SetParalmeterValue first
super.setParameterValue(parameterValue);

// then find looks for any adjunct bindings
SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();
AdjunctBindingList bindings = locus.getAllAdjunctBindings(link);

for (int i = 0; i < bindings.size(); i++) {
  // get the feature value bound by this binding
  FeatureValue adjunctFeatureValue =
bindings.get(i).adjunctFeatureValue;

  // then copy its value to those of the adjunct feature
  adjunctFeatureValue.values = parameterValue.values;
}
```

H.3.2.2.2 SysML_ActivityParameterNodeActivation

Description

This semantics visitor extends the fUML ActivityParameterNodeActivation class in order to support semantics of adjunct properties.

Generalizations

- ActivityParameterNodeActivation (from Activities)

Operations

- sendOffers (in tokens : Token) {redefines sendOffers}

```
// Call the original sendOffer operation.
// Then, if the NoBuffer stereotype is applied,
// discard remaining tokens, if any

super.sendOffers(tokens);

ObjectNode node = (ObjectNode) this.node;

SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

if (locus.isNoBuffer(node)) {
  this.clearToken();
}
```

H.3.2.2.3 SysML_ExpansionNodeActivation

Description

This semantics visitor extends the fUML ExpansionNodeActivation class in order to support semantics of the NoBuffer stereotype.

Generalizations

- ExpansionNodeActivation (from Actions)

Operations

- sendOffers (in tokens : Token) {redefines sendOffers}

```
// Call the original sendOffer operation.
// Then, if the NoBuffer stereotype is applied,
// discard remaining tokens, if any

super.sendOffers(tokens);

ObjectNode node = (ObjectNode) this.node;

SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

if (locus.isNoBuffer(node)) {
  this.clearToken();
}
```

H.3.2.2.4 SysML_ObjectNodeActivation

Description

This semantics visitor extends the fUML ObjectNodeActivation class in order to support semantics of adjunct properties.

Generalizations

- ObjectNodeActivation (from Activities)

Operations

- addToken (in token : Token) {redefines addToken}

```
// Execute a addToken as defined in the base class
// then add the corresponding value to the adjunct property

super.addToken(token);

// retrieve all the adjuncts for this node
SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();
```

```
AdjunctBindingList bindings = locus.getAllAdjunctBindings(this);

for (int i = 0; i < bindings.size(); i++) {
  // get the feature value bound by this adjunct binding
  FeatureValue adjunctFeatureValue =
bindings.get(i).adjunctFeatureValue;

  // add the token value
  adjunctFeatureValue.values.addValue(token.getValue());
}
```

- clearTokens () {redefines clearTokens}

```
// call the clearTokens operation of the base class and remove all
// the values from the adjunct property

super.clearTokens();

// retrieve all the adjuncts for this node
SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();
AdjunctBindingList bindings = locus.getAllAdjunctBindings(this);

for (int i = 0; i < bindings.size(); i++) {
  // get the feature value bound by this adjunct binding
  FeatureValue adjunctFeatureValue =
bindings.get(i).adjunctFeatureValue;

  // clear all the token values
  adjunctFeatureValue.values.clear();
}
```

- removeToken (in token : Token) : Integer [1] {redefines removeToken}

```
// Call the base class version of removeToken then
// if it return a index > 1 then remove the value at that position in
the adjunct property
// Note that index in "1 based" so adjust for java arrays that are "0
based"

int i = super.removeToken(token);
if (i > 0) {
  // retrieve all the adjuncts for this fnode
  SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();
  AdjunctBindingList bindings = locus.getAllAdjunctBindings(this);

  for (int j = 0; j < bindings.size(); j++) {
    // get the feature value bound by this adjunct binding
    FeatureValue adjunctFeatureValue =
bindings.get(j).adjunctFeatureValue;
```

```
      // remove the token value at i-1
      adjunctFeatureValue.values.removeValue(i-1);
      }
    }
    return i;
```

H.3.3 Blocks

H.3.3.1 Overview

The Blocks sub-clause is focused on the semantics for AdjunctProperty and BindingConnector that link together values of the elements they involve. The Block PropertySpecificType, DistributedProperty and ValueType stereotypes do not add any specific executable semantics to Class, Property and DataType, respectively. BoundReference, NestedConnectorEnd, EndPathMultiplicity, DirectedRelationshipPropertyPath and ElementPropertyPath provide mechanisms that allow extending the UML syntax but they have no semantics implication by themselves.

The semantics for ConnectorProperty is redundant to that of an AdjunctProperty having a Connector as its principal. Also, the semantics of ParticipantProperty is linked to AssociationBlock, but AssociationClass is not included in fUML. AdjunctProperty for Connector would also require AssociationClass. They will not be addressed further in this annex.

The semantics specified for BindingConnector is based on those given to FeatureValue by fUML. A FeatureValue owns its value (composite aggregation) and so it cannot share it with another FeatureValue. So, the only way to realize the binding connector semantics is to have one distinct value for each and to maintain them as exact copies. Feature that are typed by Classes have references to objects as values. Changing their value means changing that reference, so the copy mechanism used for ValueProperties will work as well. Based on that approach, the BindingConnector semantics is fully handled by actions modifying the value of the bound properties. That is: AddStructuralFeatureValue, ClearStructuralFeatureValue and RemoveStructuralFeatureValue.

The semantics of AdjunctProperty are quite similar to those of BindingConnector. However this sub-clause excludes adjunct for AssociationBlocks , InteractionUse and Variables, because fUML does not support them. It excludes also CallAction because it would need to either override the doAction() operation of CallActionActivation semantic visitor class which would implies a significant amount of rework of some classes of the fUML execution model that would require a new version of this standard. The semantics of ClassifierBehaviorProperty is not included in this annex for the same reason. The semantics of Adjunct for SubmachineState are also out of scope of this subclause in order to avoid inducing a dependency on PSSM.

In order to support the semantics of AdjunctProperty, an AdjunctBinding abstract class is provided. It is specialized for each kind of principal for which the semantics is described. That is: Parameter and ObjectNode. The adjunctFeatureValue of an AdjunctBinding shall refer to the feature value that is the adjunct for that model element. When the value referred by the principalValue property is modified, that value is copied to the value referred by adjunctFeature.

Note: semantics for parameter adjunct property is provided for parameters owned by activities only.

In addition the following classes of the execution fUML model are extended (see Actions and Activities paragraphs in this annex):
- for supporting adjunct of a Parameter: ActivityExecution
- for supporting adjunct of an ObjectNode: ObjectNodeActivation, ActivityParameterNodeActivation, CentralBufferNodeActivation, ExpansionNodeActivation, PinActivation, InputPinActivation, OutputPinActivation

Supported stereotypes: BindingConnector, AdjunctProperty

H.3.3.2 Class descriptions

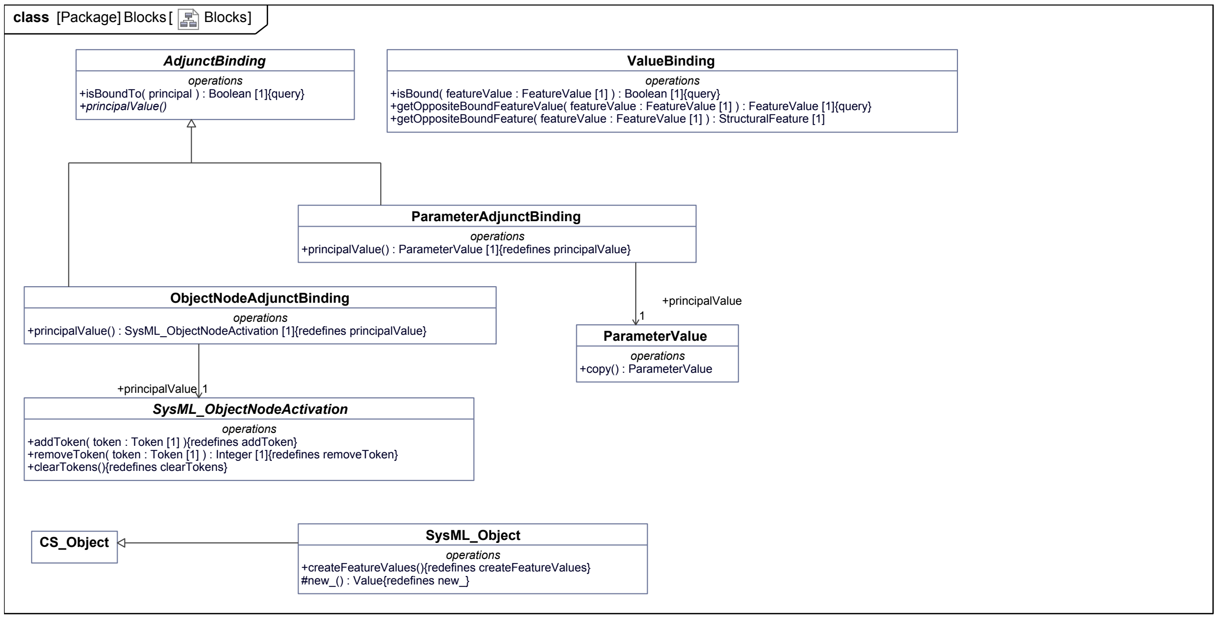

class [Package] Blocks [Blocks]
AdjunctBinding
operations
+isBoundTo(principal) : Boolean [1]{query}
+principalValue()
ValueBinding
operations
+isBound(featureValue : FeatureValue [1]) : Boolean [1]{query}
+getOppositeBoundFeatureValue(featureValue : FeatureValue [1]) : FeatureValue [1]{query}
+getOppositeBoundFeature(featureValue : FeatureValue [1]) : StructuralFeature [1]
ParameterAdjunctBinding
operations
+principalValue() : ParameterValue [1]{redefines principalValue}
+principalValue
1
ObjectNodeAdjunctBinding
operations
+principalValue() : SysML_ObjectNodeActivation [1]{redefines principalValue}
ParameterValue
operations
+copy() : ParameterValue
+principalValue 1
SysML_ObjectNodeActivation
operations
+addToken(token : Token [1]){redefines addToken}
+removeToken(token : Token [1]) : Integer [1]{redefines removeToken}
+clearTokens(){redefines clearTokens}
CS_Object
SysML_Object
operations
+createFeatureValues(){redefines createFeatureValues}
#new_() : Value{redefines new_}

Figure H.4. Blocks

H.3.3.2.1 AdjunctBinding

Description

This class is added in order to support semantics of adjunct properties. Note: bUML does not allow property redefinition, only operation redefinition => principalValue shall be defined as an operation that will return an untyped value (because ParameterValue and Link are not semantic visitors).isBound parameter shall also have no type (for the exact same reason)

Generalizations

- ExtensionalValue (from StructuredClassifiers)

Association Ends

- adjunctFeatureValue : FeatureValue [1]

Operations

- isBoundTo (in principal) : Boolean [1]

  ```
  return this.principalValue == principal;
  ```

- principalValue () [1]
  Abstract operation intended to return the value of the principal

H.3.3.2.2 ObjectNodeAdjunctBinding

Description

This class is added in order to support semantics of adjunct properties for object nodes.

Generalizations

- AdjunctBinding (from Blocks)

Association Ends

- principalValue : SysML_ObjectNodeActivation [1]

Operations

- principalValue () : SysML_ObjectNodeActivation [1] {redefines principalValue}

```
return this.principalValue;
```

H.3.3.2.3 ParameterAdjunctBinding

Description

This class is added in order to support semantics of adjunct properties for parameters. Note: the changes of parameter values (and so update of the adjunct property) are managed within SysML_ActivityExecution by overriding the setParameterValue() operation.

Generalizations

- AdjunctBinding (from Blocks)

Association Ends

- principalValue : ParameterValue [1]

Operations

- principalValue () : ParameterValue [1] {redefines principalValue}

```
return this.principalValue;
```

H.3.3.2.4 SysML_FeatureValue

Description

Generalizations

- FeatureValue (from SimpleClassifiers)

Attributes

- path : StructuralFeature [0..*]

H.3.3.2.5 SysML_Object

Description

This semantics visitor extends the PSCS CS_Object class in order to support semantics of proxy ports.

Generalizations

- CS_Object (from StructuredClassifiers)

Operations

- createFeatureValues () {redefines createFeatureValues}

```
// Create empty feature values for all structural features of the types
// of this structured value and all its supertypes (including private
// features that are not inherited).

super.createFeatureValues();

SysML_Locus locus = (SysML_Locus) this.getExecutionLocus();

//Initialize the values for behavioral proxy ports only
for (int i=0; i < this.featureValues.size(); i++) {

  Port port = (Port) this.featureValues.get(i);

  if (port != null && locus.isProxyPort(port) {
    if (port.isBehavior) {
      port.values = new ValueList(this);
    }
  }
}
```

- new_ () : Value [1] {redefines new_}

```
// Create a new object with no type, feature values or locus.
SysML_Object newObject = new SysML_Object_();
```

H.3.3.2.6 SysML_ReferencePropertyPair

Description

Association Ends

- property : Property [1]
- reference : Reference [1]

H.3.3.2.7 SysML_StructuredValue

Description

Generalizations

- StructuredValue (from SimpleClassifiers)

Operations

- addFeatureValuesForType (in type : Classifier, in oldFeatureValues : FeatureValue) {redefines addFeatureValues}

```
// Add feature values for all structural features of the given type and
// all of its supertypes (including private features that are not
// inherited). If a feature has an old feature value in the given list,
// then use that to initialize the values of the corresponding new
// feature value. Otherwise leave the values of the new feature value
// empty.

// Set feature values for the owned structural features of the given
// type. (Any common structural values that have already been added
// previously will simply have their values set again.)
NamedElementList ownedMembers = type.ownedMember;
for (int j = 0; j < ownedMembers.size(); j++) {
  NamedElement ownedMember = ownedMembers.getValue(j);
  if (ownedMember instanceof StructuralFeature) {
    this.setFeatureValue((StructuralFeature) ownedMember,
      this.getValues(ownedMember, oldFeatureValues), 0);
  }
}

// Add feature values for the structural features of the supertypes
// of the given type. (Note that the feature values for supertype
// features always come after the feature values for owned features.)
ClassifierList supertypes = type.general;
for (int i = 0; i < supertypes.size(); i++) {
  Classifier supertype = supertypes.getValue(i);
  this.addFeatureValuesForType(supertype, oldFeatureValues);
}
```

- getBoundElements (in feature) : ConnectableElement [0..*]

```
//Check whether there is a binding connector attached to this feature
ConnectableElementList = new ConnectableElementList();

if (feature instanceof ConnectableElement) {
  ConnectableElement connectableElement = (ConnectableElement) feature;

  for (int i = 0; i < connectableElement.end.size(); i++) {
    ConnectorEnd thatEnd = connectableElement.end.getValue(i);
    Connector connector = (Connector) thatEnd.owner;

    if (

  }
}
```

```
return result;
```

H.3.3.2.8 ValueBinding

Description

This class is added in order to support semantics of binding connectors.

Generalizations

- ExtensionalValue (from StructuredClassifiers)

Association Ends

- boundFeatureValues : FeatureValue [2]

Operations

- getOppositeBoundFeature (in featureValue : FeatureValue) : StructuralFeature [1]

```
StructuralFeature oppositeFeature = null ;
FeatureValue oppositeFeatureValue =
this.getOppositeFeatureValue(featureValue) ;

if (oppositeFeatureValue != null) {
  oppositeFeature = oppositeFeatureValue.feature ;
}

return oppositeFeature ;
```

- getOppositeBoundFeatureValue (in featureValue : FeatureValue) : FeatureValue [1]

```
FeatureValue oppositeFeatureValue = null ;

if (this.boundFeatureValue.get(0) == featureValue) {
  oppositeFeatureValue = this.boundFeatureValue.get(1) ;
}
else if (this.boundFeatureValue.get(1) == featureValue) {
  oppositeFeatureValue = this.boundFeatureValue.get(0) ;
}

return oppositeFeatureValue ;
```

- isBound (in featureValue : FeatureValue) : Boolean [1]

```
return this.boundFeatureValue.get(0) == featureValue ||
this.boundFeatureValue.get(1) == featureValue;
```

H.3.4 Loci

H.3.4.1 Overview

The Loci package includes extensions to fUML CS_Locus and CS_ExecutionFactory in order to account for new semantic visitors introduced by this specification. The extended Locus class also provides an additional set of utility operations that facilitate the specification of semantic visitors' operations.

H.3.4.2 Class descriptions

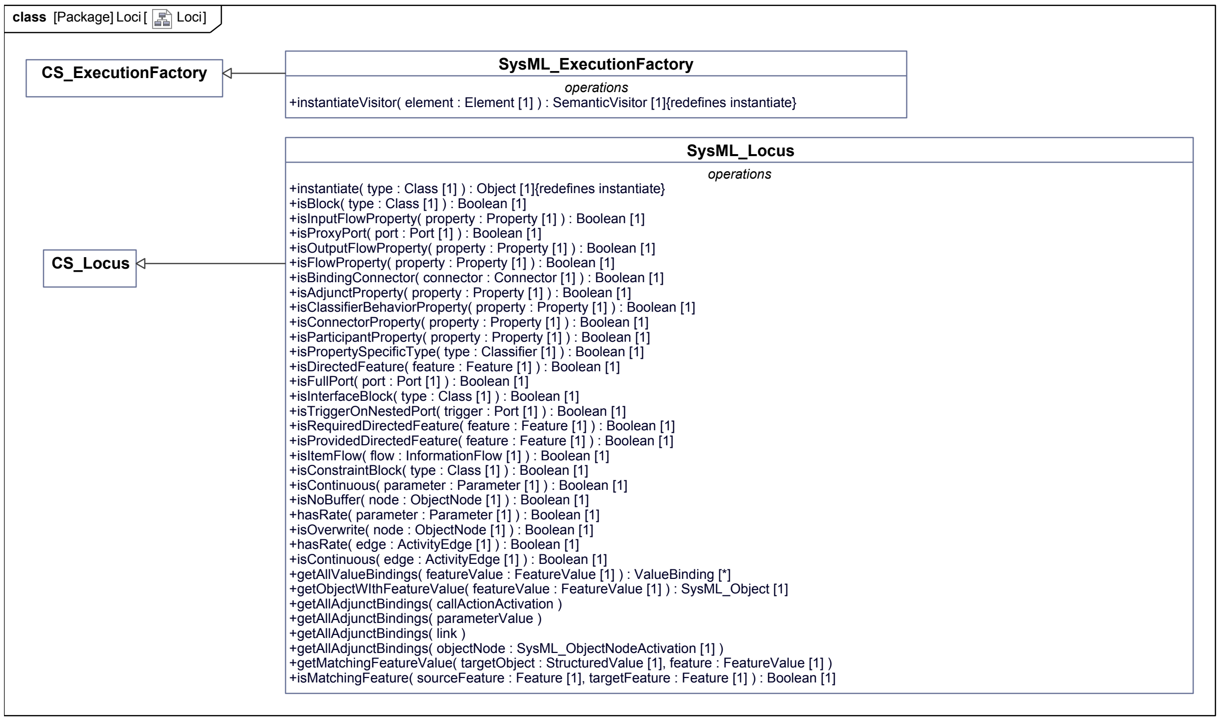

class [Package] Loci [Loci]
CS_ExecutionFactory
SysML_ExecutionFactory
operations
+instantiateVisitor(element : Element [1]) : SemanticVisitor [1]{redefines instantiate}
CS_Locus
SysML_Locus
operations
+instantiate(type : Class [1]) : Object [1]{redefines instantiate}
+isBlock(type : Class [1]) : Boolean [1]
+isInputFlowProperty(property : Property [1]) : Boolean [1]
+isProxyPort(port : Port [1]) : Boolean [1]
+isOutputFlowProperty(property : Property [1]) : Boolean [1]
+isFlowProperty(property : Property [1]) : Boolean [1]
+isBindingConnector(connector : Connector [1]) : Boolean [1]
+isAdjunctProperty(property : Property [1]) : Boolean [1]
+isClassifierBehaviorProperty(property : Property [1]) : Boolean [1]
+isConnectorProperty(property : Property [1]) : Boolean [1]
+isParticipantProperty(property : Property [1]) : Boolean [1]
+isPropertySpecificType(type : Classifier [1]) : Boolean [1]
+isDirectedFeature(feature : Feature [1]) : Boolean [1]
+isFullPort(port : Port [1]) : Boolean [1]
+isInterfaceBlock(type : Class [1]) : Boolean [1]
+isTriggerOnNestedPort(trigger : Port [1]) : Boolean [1]
+isRequiredDirectedFeature(feature : Feature [1]) : Boolean [1]
+isProvidedDirectedFeature(feature : Feature [1]) : Boolean [1]
+isItemFlow(flow : InformationFlow [1]) : Boolean [1]
+isConstraintBlock(type : Class [1]) : Boolean [1]
+isContinuous(parameter : Parameter [1]) : Boolean [1]
+isNoBuffer(node : ObjectNode [1]) : Boolean [1]
+hasRate(parameter : Parameter [1]) : Boolean [1]
+isOverwrite(node : ObjectNode [1]) : Boolean [1]
+hasRate(edge : ActivityEdge [1]) : Boolean [1]
+isContinuous(edge : ActivityEdge [1]) : Boolean [1]
+getAllValueBindings(featureValue : FeatureValue [1]) : ValueBinding [*]
+getObjectWIthFeatureValue(featureValue : FeatureValue [1]) : SysML_Object [1]
+getAllAdjunctBindings(callActionActivation)
+getAllAdjunctBindings(parameterValue)
+getAllAdjunctBindings(link)
+getAllAdjunctBindings(objectNode : SysML_ObjectNodeActivation [1])
+getMatchingFeatureValue(targetObject : StructuredValue [1], feature : FeatureValue [1])
+isMatchingFeature(sourceFeature : Feature [1], targetFeature : Feature [1]) : Boolean [1]

Figure H.5. Loci

H.3.4.2.1 SysML_ExecutionFactory

Description

This class extends the PSCS CS_ExecutionFactory class in order to support the semantics visitors added by this annex.

Generalizations

- CS_ExecutionFactory (from Loci)

Operations

- instantiateVisitor (in element : Element) : SemanticVisitor [1] {redefines instantiate}
  <<TextualRepresentation>>public instantiateVisitor (in element : Element) : SemanticVisitor { // TODO return super.instantiateVisitor(element) ; }

```
// Extends CS_ExecutionFactory to instantiate
// SysML semantic visitors

SemanticVisitor visitor = null ;
if (element instanceof Activity) {
  visitor = new SysML_ActivityExecution() ;
}
else if (element instanceof ActivityParameterNode) {
  visitor = new SysML_ActivityParameterNodeActivation() ;
}
else if (element instanceof AddStructuralFeatureValueAction) {
  visitor = new SysML_AddStructuralFeatureValueActionActivation() ;
}
else if (element instanceof CallOperationAction) {
  visitor = new SysML_CallOperationActionActivation() ;
}
else if (element instanceof ClearStructuralFeatureAction) {
  visitor = new SysML_ClearStructuralFeatureActionActivation() ;
}
else if (element instanceof ExpansionNode) {
  visitor = new SysML_ExpansionNodeActivation() ;
}
else if (element instanceof InputPin) {
  visitor = new SysML_InputPinActivation() ;
}
else if (element instanceof ObjectNode) {
  visitor = new SysML_ObjectNodeActivation() ;
}
else if (element instanceof OutputPin) {
  visitor = new SysML_OutputPinActivation() ;
}
else if (element instanceof ReadStructuralFeatureAction) {
  visitor = new SysML_ReadStructuralFeatureActionActivation() ;
}
else if (element instanceof RemoveStructuralFeatureValueAction) {
  visitor = new SysML_RemoveStructuralFeatureValueActionActivation() ;
}
else if (element instanceof SendSignalAction) {
  visitor = new SysML_SendSignalActionActivation() ;
}
else {
  visitor = super.instantiateVisitor(element) ;
}
return visitor ;
```

H.3.4.2.2 SysML_Locus

Description

This class extends the PSCS CS_Locus class in order to provide a set of utility operations for SysML stereotypes.

Generalizations

- CS_Locus (from Loci)

Operations

- getAllAdjunctBindings (in callActionActivation) [0..*]
- getAllAdjunctBindings (in link) [0..*]
- getAllAdjunctBindings (in objectNode : SysML_ObjectNodeActivation) [0..*]

```
// Return the set of ajunct bindings at this locus which involve the
// given object node
getAllAdjunctBindings bindings = new AdjunctBindingList();

ExtensionalValueList extensionalValues = this.extensionalValues;
for (int i = 0; i < extensionalValues.size(); i++) {
  ExtensionalValue value = extensionalValues.getValue(i);

  if (value instanceof ObjectNodeAdjunctBinding) {
    ObjectNodeAdjunctBinding binding = (ObjectNodeAdjunctBinding)
value;

    if (binding.isBound(objectNode)) {
      bindings.addValue(binding);
    }
  }
}

return bindings;
```

- getAllAdjunctBindings (in parameterValue) [0..*]

```
// Return the set of ajunct bindings at this locus which involve the
// given parameter
getAllAdjunctBindings bindings = new AdjunctBindingList();

ExtensionalValueList extensionalValues = this.extensionalValues;
for (int i = 0; i < extensionalValues.size(); i++) {
  ExtensionalValue value = extensionalValues.getValue(i);

  if (value instanceof ParameterAdjunctBinding) {
    ParameterAdjunctBinding binding = (ParameterAdjunctBinding) value;

    if (binding.isBound(parameterValue)) {
      bindings.addValue(binding);
    }
  }
}

return bindings;
```

- getAllValueBindings (in featureValue : FeatureValue) : ValueBinding [0..*]

```
// Return the set of value bindings at this locus which involve the
// given feature value

ValueBindingList bindings = new ValueBindingList();

ExtensionalValueList extensionalValues = this.extensionalValues;
for (int i = 0; i < extensionalValues.size(); i++) {
  ExtensionalValue value = extensionalValues.getValue(i);

  if (value instanceof ValueBinding) {
    ValueBinding binding = (ValueBinding) value;

    if (binding.isBound(featureValue)) {
      bindings.addValue(binding);
    }
  }
}

return bindings;
```

• getMatchingFeatureValue (in targetObject : StructuredValue, in feature : FeatureValue)

```
// First check whether the property provided as a parameter is a flow
property
// or a required feature
// if so look for the links attached to the targetObject
// for each link found, check whether there is a property on the other
side that is a "matching" flow property
// according to SysML, "matching" flow properties have compatible
directions and conforming types

//

FeatureValueList matchingFeatures = new FeatureValueList();

if (feature instanceof Property && (this.isOutFlowProperty((Property)
feature))
      || this.isRequiredFeature(feature) {
  LinkList links = new LinkList();

  ExtensionalValueList extensionalValues = this.extensionalValues;

  for (int i = 0; i < extensionalValues.size(); i++) {
    ExtensionalValue value = extensionalValues.getValue(i);

    if (value instanceof Link) {
      Link link = (Link) value;
      FeatureValueList linkFeatureValues = link.getFeatureValues();
      FeatureValue candidateFeatureValue = null;
```

```
        if (linkFeatureValues.getValue(0).equals(targetObject)) {
          candidateFeatureValue = linkFeatureValues.getValue(1);
        } else if (linkFeatureValues.getValue(1).equals(targetObject)) {
          candidateFeatureValue = linkFeatureValues.getValue(0);
        }

        if (candidateFeatureValue != null ) {
          //now we can check whether this feature "matches"
          if (this.isMatchingFeature(feature,
candidateFeatureValue.feature)) {
            matchingFeatures.addValue(candidateFeatureValue);
          }
        }

      }
    }
  }

  return matchingFeatures;
```

- getObjectWIthFeatureValue (in featureValue : FeatureValue) : SysML_Object [1]

```
// Return the object at this locus which owns the
// given feature value

SysML_Object object = null;

ExtensionalValueList extensionalValues = this.extensionalValues;
int i = 0;
while (i < extensionalValues.size() && object = null) {
  ExtensionalValue value = extensionalValues.getValue(i);

  if (value instanceof SysML_Object) {
    SysML_Object candidate = (SysML_Object) value;
    FeatureValueList featureValues = candidate.featureValues;
    int j = 0;

    while (j < featureValues.size() && object = null) {
      if (featureValues.get(j) == featureValue) {
        object = candidate;
      }
      j++;
    }
  }
  i++;
}

return object;
```

- hasRate (in edge : ActivityEdge) : Boolean [1]
  Check whether the activity edge has the Rate stereotype applied. // The algorithm of this operation is implementation specific
- hasRate (in parameter : Parameter) : Boolean [1]
  Check whether the parameter has the Rate stereotype applied. // The algorithm of this operation is implementation specific
- instantiate (in type : Class) : Object [1] {redefines instantiate}

```
// If the type is a Block, instantiate a SysML_Object.
// Otherwise behaves like in CS_Locus
if (isBlock(type)) {
  Object_ object = null;
  object = new SysML_Object() ;
  object.types.add(type);
  this.add(object);
  object.createFeatureValues();
  this.assignBehaviorProxyPorts(object);
  return object;
}
else {
  return super.instantiate(type);
}
```

- isAdjunctProperty (in property : Property) : Boolean [1]
  Check whether the property has the AdjunctProperty stereotype applied // The algorithm of this operation is implementation specific
- isBindingConnector (in connector : Connector) : Boolean [1]
  Check whether the connector has the Block stereotype applied. // The algorithm of this operation is implementation specific
- isBlock (in type : Class) : Boolean [1]
  Check whether the class has the Block stereotype applied. // The algorithm of this operation is implementation specific
- isClassifierBehaviorProperty (in property : Property) : Boolean [1]
  Check whether the property has the ClassifierBehaviorProperty stereotype applied // The algorithm of this operation is implementation specific
- isConnectorProperty (in property : Property) : Boolean [1]
  Check whether the property has the ConnectorProperty stereotype applied // The algorithm of this operation is implementation specific
- isConstraintBlock (in type : Class) : Boolean [1]
  Check whether the class has the ConstraintBlock stereotype applied. // The algorithm of this operation is implementation specific
- isContinuous (in edge : ActivityEdge) : Boolean [1]
  Check whether the activity edge has the Continuous stereotype applied. // The algorithm of this operation is implementation specific
- isContinuous (in parameter : Parameter) : Boolean [1]
  Check whether the parameter has the Continuous stereotype applied. // The algorithm of this operation is implementation specific
- isDirectedFeature (in feature : Feature) : Boolean [1]
  Check whether the feature has the DirectedFeature stereotype applied // The algorithm of this operation is implementation specific
- isFlowProperty (in property : Property) : Boolean [1]
  Check whether the property has the FlowProperty stereotype applied // The algorithm of this operation is implementation specific

- isFullPort (in port : Port) : Boolean [1]
  Check whether the port has the FullPort stereotype applied. // The algorithm of this operation is implementation specific
- isInputFlowProperty (in property : Property) : Boolean [1]
  Check whether the property has the FlowProperty stereotype applied and the flow direction is "in" // The algorithm of this operation is implementation specific
- isInterfaceBlock (in type : Class) : Boolean [1]
  Check whether the class has the InterfaceBlock stereotype applied. // The algorithm of this operation is implementation specific
- isItemFlow (in flow : InformationFlow) : Boolean [1]
  Check whether the information flow has the ItemFlow stereotype applied. // The algorithm of this operation is implementation specific
- isMatchingFeature (in sourceFeature : Feature, in targetFeature : Feature) : Boolean [1]

```
//"Matching" applies to flow properties and directed features
//Flow properties "match" when they have opposite directions and
compatible types. That is:
// - the source flow property shall be out or inout
// - the target flow property shall be in or inout
// - the type of the source flow property shall be the same or a
specialization of the type of the target flow property

boolean result = false;
boolean directionChk = false
boolean typeChk = false;


if (this.isFlowProperty(sourceFeature) &&
this.isFlowProperty(targetFeature)) {
  FlowDirectionKind srcDirection = this.getDirection(sourceFeature);
  FlowDirectionKind tgtDirection = this.getDirection(targetFeature);

  Type srcType = ((StructuralFeature) sourceFeature).type;
  Type tgtType = ((StructuralFeature) targetFeature).type;

  directionChk = (srcDirection == FlowDirectionKind.out || srcDirection
== FlowDirectionKind.inout) &&
    (tgtDirection == FlowDirectionKind.in || tgtDirection ==
FlowDirectionKind.inout);

  typeChk = (tgtType == null || srcType != null &&
srcType.conformsTo(tgtType));

  result = directionChk & typeChk;
}
else {
  if (this.isDirectedFeature(sourceFeature) &&
this.isDirectedFeature(targetFeature)) {
    FeatureDirectionKind srcDirection =
this.getFeatureDirection(sourceFeature);
    FeatureDirectionKind tgtDirection =
this.getFeatureDirection(targetFeature);

    Type srcType = ((StructuralFeature) sourceFeature).type;
    Type tgtType = ((StructuralFeature) targetFeature).type;


    directionChk = (srcDirection == FeatureDirectionKind.provided ||
srcDirection == FeatureDirectionKind.provrequired) &&
      (tgtDirection == FeatureDirectionKind.required || tgtDirection ==
FeatureDirectionKind.provrequired);

    if (sourceFeature instanceof BehavioralFeature and targetFeature
instanceof BehavioralFeature) {
      BehavioralFeature sourceBFeature = (BehavioralFeature)
sourceFeature;
      BehavioralFeature targetBFeature = (BehavioralFeature)
targetFeature;

      boolean paramChk = sourceBFeature.ownedParameter.size() ==
targetBFeature.ownedParameter.size();

      for (int i=0; paramChk && i <
sourceBFeature.ownedParameter.size(); i++) {

        Parameter sourceParam = sourceBFeature.ownedParameter.get(i);
        Parameter targetParam = targetBFeature.ownedParameter.get(i);

        paramChk = paramChk &&
sourceParam.type.conformsTo(targetParam.type);

        paramChk = paramChk && sourceParam.lower >= targetParam.lower;
        paramChk = paramChk && sourceParam.upper <= targetParam.upper;

        paramChk = paramChk && sourceParam.direction ==
targetParam.direction;
      }

      result = directionChk && paramChk;
    }
    else {
      if (sourceFeature instanceof StructuralFeature and targetFeature
instanceof StructuralFeature) {

        StructuralFeature sourceSFeature = (StructuralFeature)
sourceFeature;
        StructuralFeature targetSFeature = (StructuralFeature)
targetFeature;

        typeChk = sourceSFeature.type.conformsTo(targetSFeature.type)
&&
          sourceSFeature.lower >= targetSFeature.lower &&
          targetSFeature.upper <= targetSFeature.upper;


        result = directionChk && typeChk;
      }
    }

  }
}

return result;
```

- isNoBuffer (in node : ObjectNode) : Boolean [1]
  Check whether the object node has the NoBuffer stereotype applied. // The algorithm of this operation is implementation specific
- isOutputFlowProperty (in property : Property) : Boolean [1]
  Check whether the property has the FlowProperty stereotype applied and the flow direction is "out" // The algorithm of this operation is implementation specific
- isOverwrite (in node : ObjectNode) : Boolean [1]
  Check whether the object node has the Overwrite stereotype applied. // The algorithm of this operation is implementation specific
- isParticipantProperty (in property : Property) : Boolean [1]
  Check whether the property has the ParticipantProperty stereotype applied // The algorithm of this operation is implementation specific
- isPropertySpecificType (in type : Classifier) : Boolean [1]
  Check whether the classifier has the PropertySpecific stereotype applied // The algorithm of this operation is implementation specific
- isProvidedDirectedFeature (in feature : Feature) : Boolean [1]
  Check whether the feature has the DirectedFeature stereotype applied with direction "provided" // The algorithm of this operation is implementation specific
- isProxyPort (in port : Port) : Boolean [1]
  Check whether the port has the ProxyPort stereotype applied. // The algorithm of this operation is implementation specific
- isRequiredDirectedFeature (in feature : Feature) : Boolean [1]
  Check whether the feature has the DirectedFeature stereotype applied with direction "required" // The algorithm of this operation is implementation specific
- isTriggerOnNestedPort (in trigger : Port) : Boolean [1]
  Check whether the port has the TriggerOnNestedPort stereotype applied. // The algorithm of this operation is implementation specific

H.3.5 Ports and Flows

H.3.5.1 Overview

This clause specifies executable semantics for FlowProperty and ProxyPort. With regard to the executable semantics, a FullPort is the same a a classical part.

Writing a value to an "out" flow property is the same as writing this value to a matching "in" flow property, if there is one and only one. This can be realized by extending WriteStructuralFeatureActionActivation using a mechanism similar to the one use for the binding connectors, taking care to avoid infinite loop in case of "inout" flow properties. In order to avoid inconsistencies an additional constraint prevents flow properties to have a composite aggregation kind. It is assumed that a flow may occur if there is a link, whatever the way it has been created. So, there is no need to retrieve the corresponding connector.

A proxy port stands for another element in the model that can be: either the port owner, if the port is behavioral (i.e. its isBehavior property is true), or a part of the block owning the port, if it is not behavioral. This can be realized by initializing the value of a proxy port with a reference to its owner, if it is behavioral, or with the the reference to its bound part otherwise. It is managed in the extension SysML_Object.

In order to avoid inconsistencies with proxy ports, the following constraints shall be enforced.

- In case of a behavioral port, the type of that port shall also classify the owner of the port
- A non behavioral proxy-port shall be bound to a part of its owner
- In case of a non behavioral port, the type of the port shall also classify the part to which that port is bound

Supported stereotypes: FlowProperty, ProxyPort

H.3.5.2 Additional Constraints

- `behavioral_port_owner_has_compatible_type`
  For a behavioral port, the type of that port shall also classify the owner of the port

  ```
  context ProxyPort inv: self.base_Port.isBehavior implies
  self.base_Port.class.conformsTo(self.base_Port.type))
  ```

- `bound_part_has_compatible_type`
  In case of a non behavioral port, the type of the port shall also classify the part to which that port is bound

  ```
  context ProxyPort inv: not self.base_Port.isBehavior implies
  BindingConnector.allInstances() ->exists(b | b.base_Connector.end->exists(e1
  | e1.role = self.base_Port) and b.base_Connector.end->exists(e2 | e2.role <>
  self.base_Port and e2.role.type.conformsTo(self.base_Port.type)) )
  ```

- `bound_to_owner_part`
  A non behavioral proxy-port shall be bound to a part of its owner

  ```
  context ProxyPort inv: let internalParts: Set(Property) =
  self.base_Port.owner.allFeatures() ->selectByKind(Property)->reject(f |
  f.oclIsKindOf(Port) in not self.base_Port.isBehavior implies
  BindingConnector.allInstances() ->exists(b | b.base_Connector.end->exists(e1
  | e1.role = self.base_Port) and b.base_Connector.end->exists(e2 | e2.role <>
  self.base_Port and internalParts->includes(e2.role))
  ```

- `flowproperty_not_composite`
  Flow properties shall not have a composite aggregation kind

  ```
  context FlowProperty inv: not self.base_Property.isComposite
  ```

- `provrequired_not_supported`
  No semantics is specified for features with direction providedRequired

  ```
  context Feature inv: let df: DirectedFeature =
  ```

```
DirectedFeature.allInstances()->any(f | f.base_Feature = self) in
df.oclIsUndefined() or df.direction <> DirectedFeatureKind#providedRequired
```

H.3.5.3 Class descriptions