University of Kent

Kent Academic Repository

Akehurst, David H. (2004) *Validating BPEL Specifications using OCL.* Technical report. kent university

Downloaded from

https://kar.kent.ac.uk/14114/ The University of Kent's Academic Repository KAR

The version of record is available from

This document version

UNSPECIFIED

DOI for this version

Licence for this version

UNSPECIFIED

Additional information

Versions of research works

Versions of Record

If this version is the version of record, it is the same as the published version available on the publisher's web site. Cite as the published version.

Author Accepted Manuscripts

If this document is identified as the Author Accepted Manuscript it is the version after peer review but before type setting, copy editing or publisher branding. Cite as Surname, Initial. (Year) 'Title of article'. To be published in *Title of Journal* , Volume and issue numbers [peer-reviewed accepted version]. Available at: DOI or URL (Accessed: date).

Enquiries

If you have questions about this document contact ResearchSupport@kent.ac.uk. Please include the URL of the record in KAR. If you believe that your, or a third party's rights have been compromised through this document please see our Take Down policy (available from https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies).

Computer Science at Kent

Validating BPEL Specifications using OCL

D.H.Akehurst

Technical Report No. 15-04
August 2004

Copyright © 2004 University of Kent at Canterbury
Published by the Computing Laboratory,
University of Kent, Canterbury, Kent CT2 7NF, UK

Contents

1 Introduction

The Business Process Execution Language for Web Services (BPEL) is a specification language for modelling executable business processes. The BPEL standard [1] defines the structure an XML document should take in order to represent a BPEL specification. This document contains a number of constraints written in informal text and it is a time consuming and error prone task to check that all of these textual constraints have been met each time a BPEL specification is written.

This report gives a UML model of the structure for a BPEL document and provides a formal version of each informal constraint using the Object Constraint Language (OCL) [2]. Given this formalisation of the constraints, it is possible, using tools developed at Kent along with IBM's Eclipse Modelling Framework (EMF) [3] to convert the OCL constraints into Java code that forms an automatic validation tool for BPEL documents.

2 BPEL 1.1 Metamodel

The model of the BPEL language defined here is in accordance with the version 1.1 of the BPEL standard.

All yes/no options from the specification are mapped to Booleans; with ‘true’ representing ‘yes’.

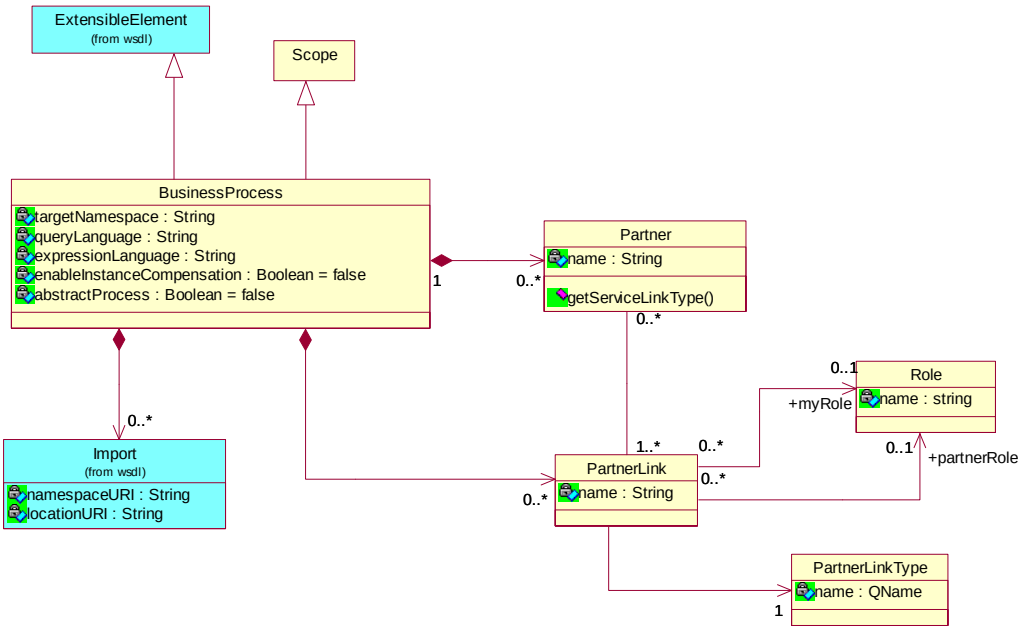

Figure 1 Business Process

The BusinessProcess class extends the class Scope. There is a large overlap between the two classes (partly due to changes from version 1.0 to 1.1), and the extension simplifies the model. The BusinessProcess class now contains PartnerLinks, these were not part of the BPEL 1.0 specification. The notion of ‘imports’ to a business process have been added (for the purpose of this work) in order to aid the reading of BPEL specifications and their reference to other documents.

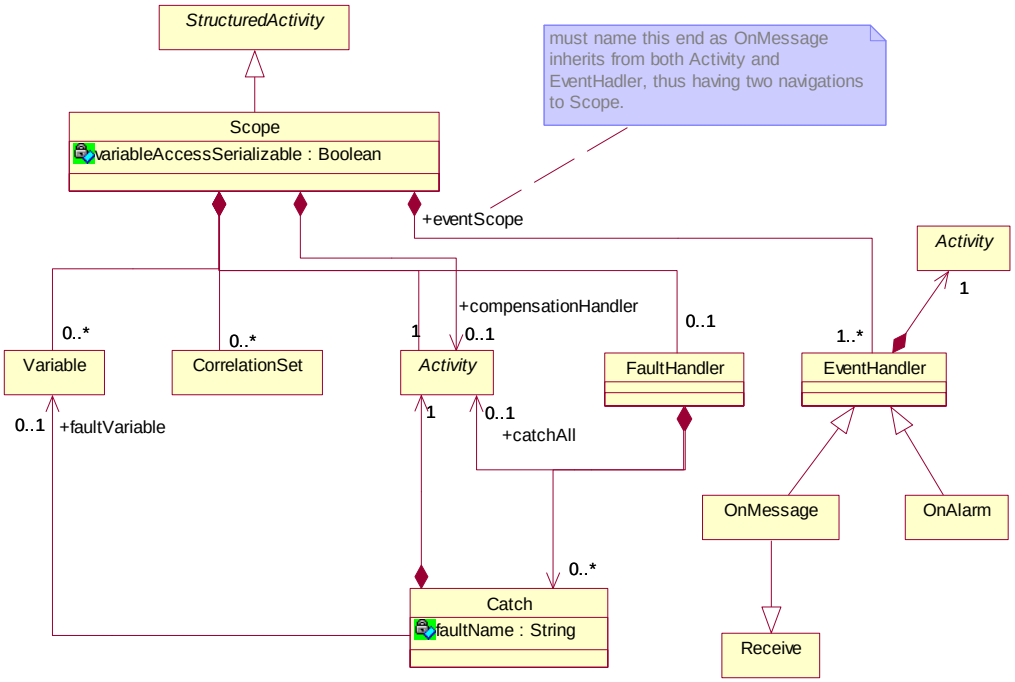

Figure 2 Scope

As of BPEL version 1.1, the Scope construct contains a collection of Variables (replacement construct for Containers), a collection of CorrelationSets and an optional EventHandler.

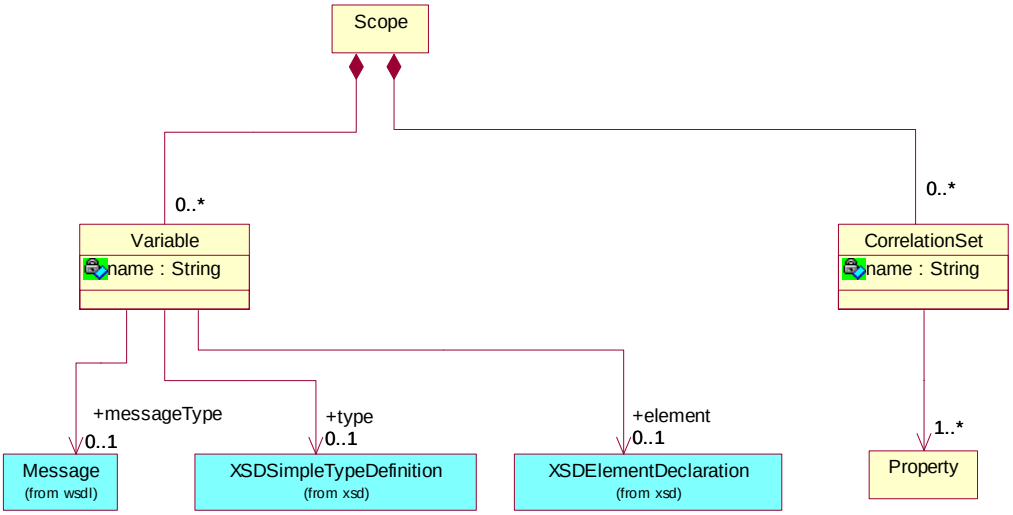

Figure 3 Variable and CorrelationSet

A Variable refers to one of the three optional parts, Message, SimpleType, Element. An OCL constraint restricts the reference to one of these. It might be better to model them as subtypes rather than exclusive-or optional references.

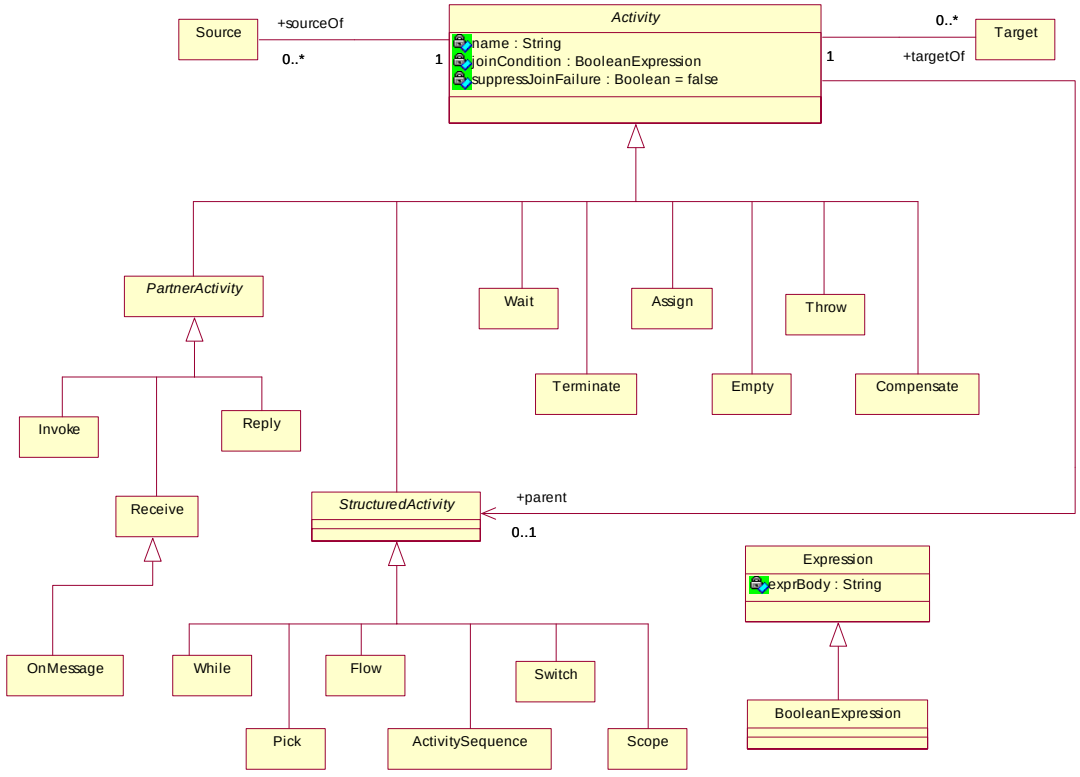

Figure 4 Activity Hierarchy and Standard Parts

An additional Layer has been added to the activity hierarchy. The StructuredActivity and BasicActivity classer partitions the activities into those that contain sub-activities and those that don't. This helps with defining constraints to model the restrictions on links and the boundary crossing conditions.

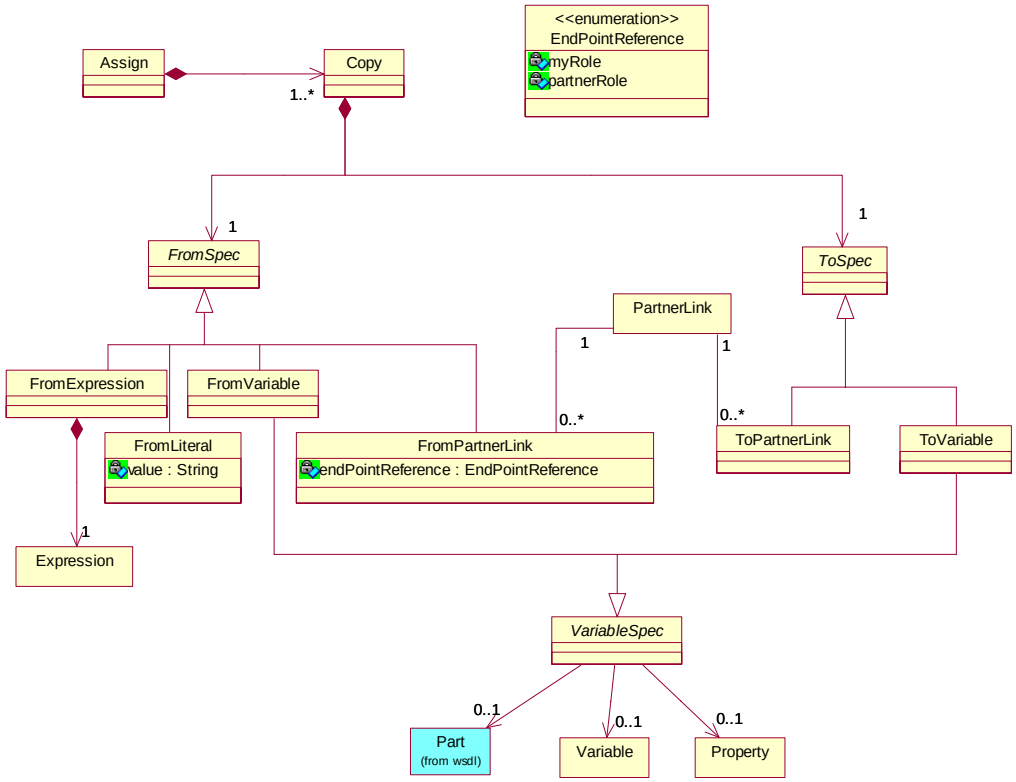

Figure 5 Assign Activity

The alternative options for From and To specs in the Copy construct are modelled as sub-types.

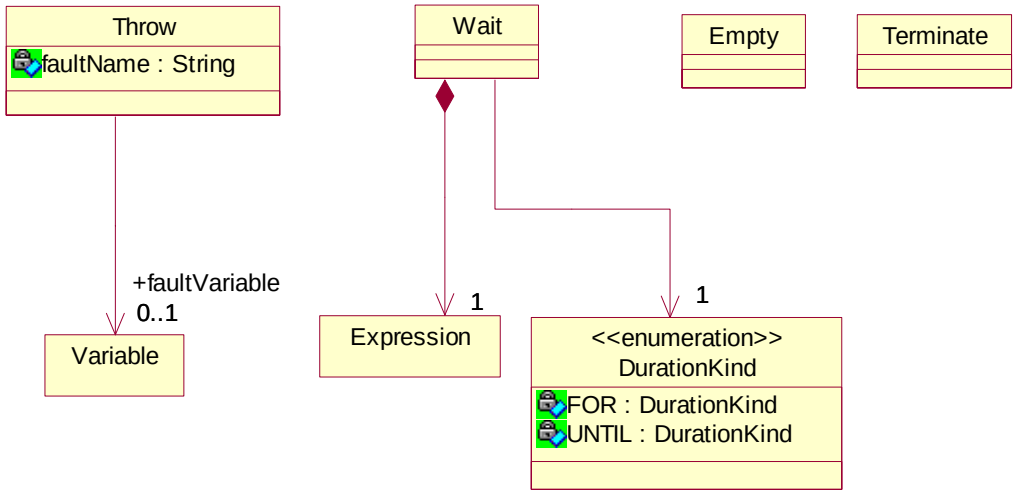

Figure 6 Basic Activities

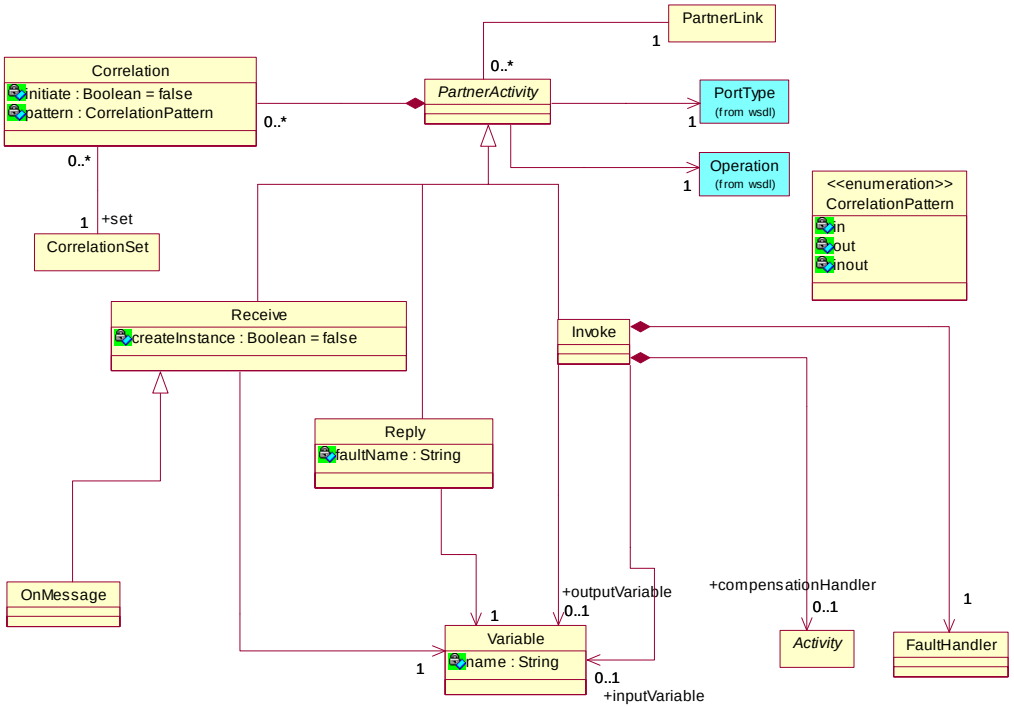

Figure 7 Partner Activities

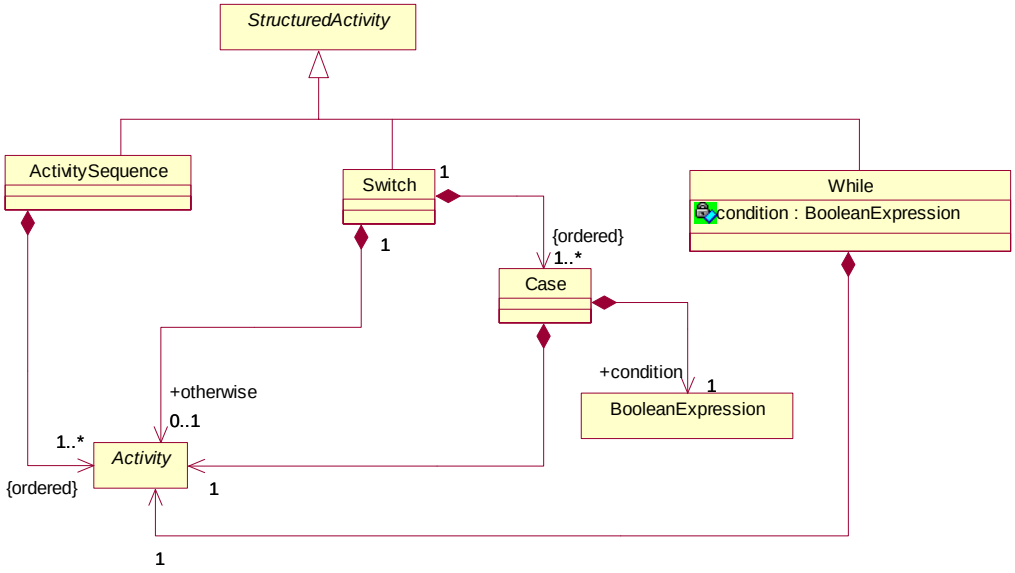

Figure 8 Structured Activities 1

The class modelling the construct for a sequence of activities is renamed ActivitySequence (originally Sequence) as the original name clashes with the OCL type Sequence.

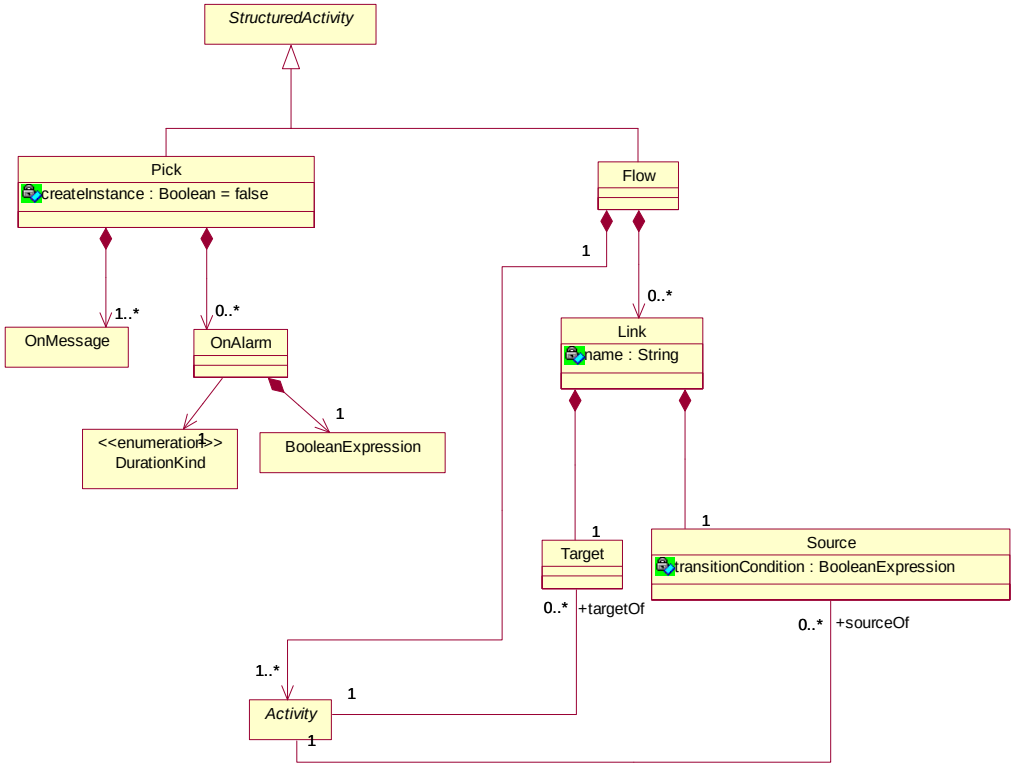

Figure 9 Structured Activities 2

The OnAlarm construct is modelled with a single expression and an enumeration to indicate whether it is an ‘until’ or ‘for’ expression; rather than two exclusive-or expression attributes.

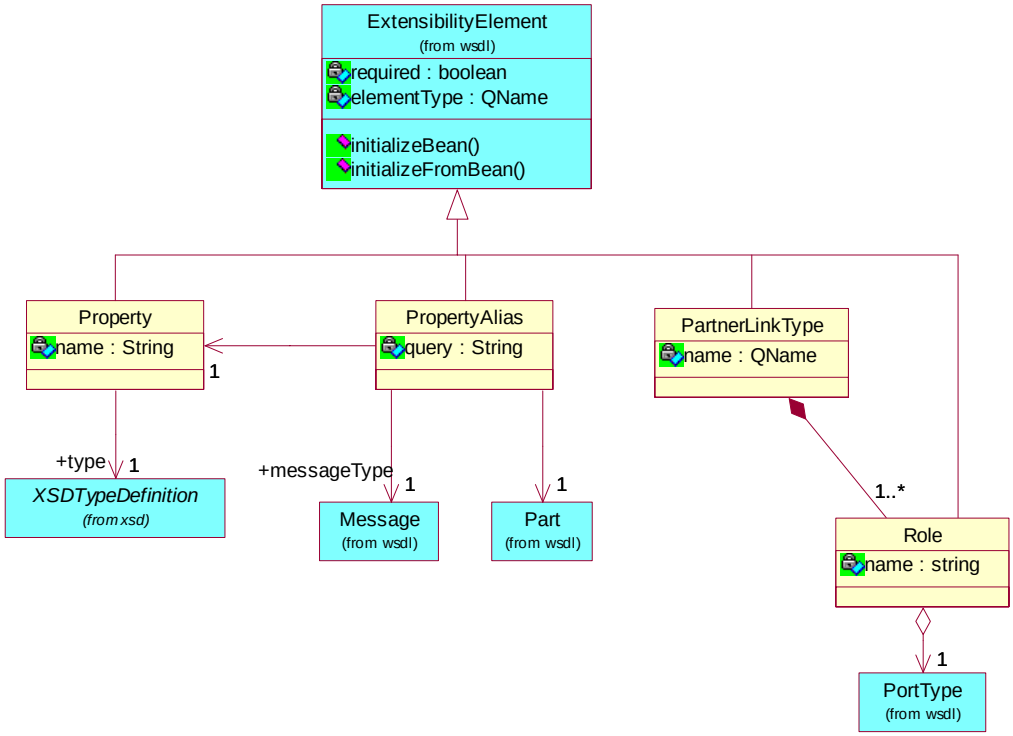

Figure 10 WSDL Extensions

PartnerLink is the version 1.1 replacement for the version 1.0 ServiceLinkType.

3 Constraints

The following subsections contain quotations from the BPEL specification document [1]. Each quotation is a piece of descriptive text that places constraints on to a BPEL specification that are not captured by the XML structure of a BPEL document.

The quoted textual constraints are translated into OCL constraints placed in the context of classes from the BPEL metamodel.

Each subsection corresponds to a set of constraints taken from one subsection of the standard document.

3.1 Business Process

> "6.2. The Structure of a Business Process
> …
> <partnerLinks>?
> <!-- Note: At least one role must be specified. -->"

```
context bpel::PartnerLink
  inv atLeastOneRoleMustBeDefined :
    not ( self.myRole.oclIsUndefined() and
          self.partnerRole.oclIsUndefined() )
```

> "…
> <faultHandlers>?
> <!-- Note: There must be at least one fault handler or default. -->"

```
context bpel::FaultHandler
  inv atLeastOneFaultHandlerOrDefault :
    self.catchAll.oclIsUndefined() implies self.catch->notEmpty()
```

> "…
> <eventHandlers>?
> <!-- Note: There must be at least one onMessage or onAlarm handler. -->"

```
context bpel_11::Scope
  inv atLeastOneOnMessageOrOnAlarmHandler :
    self.eventHandler.oclIsUndefined() implies
      self.eventHandler->notEmpty()
```

The BPEL metamodel defines a Business Process to be a subtype of Scope in order to reuse the structure of the Scope element. However, a Business Process is not mentioned in the BPEL specification as being a sub Activity of any other activity, thus we place an additional constraint as follows:

```
context bpel::BusinessProcess
  inv processIsNotASubActivity :
    self.parent.oclIsUndefined()
```

3.2 Partner definitions must not overlap

> "7.3 Business Partners
> … Partner definitions MUST NOT overlap, that is, a partner link MUST NOT appear in more than one partner definition."

```
context bpel::Process
  inv partnerDefinitionsMustNotOverlap :
    let
      x=self.partner.partnerLink
    in
      x->asSet()->asBag() = x->asBag()
```

The union of partnerLink objects from each partner in a process is a Set; i.e. each partnerLink in that union is unique.

NB. This constraint could be forced by the model, however, the model would then not represent the XML structure and the constraint could not be checked. I.e. If we model the relationship between Partner and PartnerLink as an [0..1] – to – [0..*] association, an EMF implementation of this model would enforce the [0..1] multiplicity when a partnerLink is added to a partner by changing the partner referred to by the partnerLink; no notification is given when this occurs. Hence the model defines a [0..*] – to – [0..*] association, and an OCL constraint is given.

3.3 getLinkStatus Function

> "9.1 Expressions
> … [bpws:getLinkStatus ('linkName')] function MUST NOT be used anywhere except in a join condition. The linkName argument MUST refer to the name of an incoming link for the activity associated with the join condition. These restrictions MUST be statically enforced."

To statically check this constraint it would be necessary to parse the body of an Expression construct; it would be in appropriate to attempt this using OCL.

3.4 Variable Options

> "9.2 Variables
> … The `messageType`, `type` or `element` attributes are used to specify the type of a variable. Exactly one of these attributes must be used."

```
context bpel::Variable
  inv variableRefersToOneItem :
    Bag{ not self.messageType.oclIsUndefind(),
         not self.type.oclIsUndefind(),
         not self.element.oclIsUndefind() }->count(true) = 1
```

NB. It might be better to model this as separate subtypes of Variable.

3.5 Assignment from-spec and to-specs

> "9.3 Assignment
> … The assign activity copies a type-compatible value from the source ("from-spec") to the destination ("to-spec"). The *from-spec* MUST be one of the following forms except for the opaque form available in abstract processes:
>
> <from variable="ncname" part="ncname"?/>
> <from partnerLink="ncname" endpointReference="myRole|partnerRole"/>
> <from variable="ncname" property="qname"/>
> <from expression="general-expr"/>
> <from> ... literal value ... </from>"

This is captured in the model by the four sub-types of the `FromSpec` class.

> "… The *to-spec* MUST be one of the following forms:
>
> <to variable="ncname" part="ncname"?/>
> <to partnerLink="ncname"/>
> <to variable="ncname" property="qname"/>"

This is captured in the model by the two sub-types of the `ToSpec` class.

> "… In the first *from-spec* and *to-spec* variants the variable attribute provides the name of a variable. If the type of the variable is a WSDL messge type the optional part attribute MAY be used to provide the name of a part within that variable. When the variable is defined using XML Schema simple type or element, the part attribute MUST NOT be used."

```
context bpel::VariableSpec
  inv useOfVariableSpecPartProperty :
    self.variable.message.oclIsUndefined() implies
    self.part.oclIsUndefined()
```

> "…In the case of from-specs, the role must also be specified"

```
context bpel::FromPartnerLink
  inv specificatinOfFromSpecRole :
    not self.endPointReference.oclIsUndefined()
```

> "… The fifth *from-spec* variant allows a literal value to be given as the source value to assign to a destination. The type of the literal value MUST be the type of the destination (to-spec)."

This constraint is covered by the Type Compatibility constraints below.

> "…
> 9.3.1. Type Compatibility in Assignment

For an assignment to be valid, the data referred to by the from and to specifications MUST be of compatible types. The following points make this precise:

- The from-spec is a variable of a WSDL message type and the to-spec is a variable of a WSDL message type. In this case both variables MUST be of the same message type, where two message types are said to be equal if their qualified names are the same.
- The from-spec is a variable of a WSDL message type and the to-spec is not, or *vice versa*. This is not legal because parts of variables, selections of variable parts, or endpoint references cannot be assigned to/from variables of WSDL message types directly.
- In all other cases, the types of the source and destination are XML Schema types or elements, and the constraint is that the source value MUST possess the element or type associated with the destination. Note that this does not require the types associated with the source and destination to be the same. In particular, the source type MAY be a subtype of the destination type. In the case of variables defined by reference to an element, moreover, both the source and the target MUST be the same element."

The type of from and to specs is given by a property 'type'. This property may be set or calculated when populating the model. There does not appear to be a mechanism within the BPEL spec that covers the concept of a Type.

For the classes FromVariable and ToVariable, the BPEL spec defines the notion of type, used to define the type property of the classes as follows:

```
context bpel_11::FromSpec
  def: type : OclAny = 'undefinedType'

context bpel_11::ToSpec
  def: type : OclAny = 'undefinedType'

context bpel_11::FromVariable
  def: type : wsdl::QName = self.variable.messageType.qName

context bpel_11::ToVariable
  def: type : wsdl::QName = self.variable.messageType.qName
```

The notion of type for the other constructs is not defined. We assume that there will be a mechanism for setting the type property for from and to specs in these other cases. The OCL for checking type compatibility is as follows:

```
context bpel::Copy
  inv typeCompatibilityInAssignment :
    self.fromSpec.type = self.toSpec.type
```

3.6 Binding first Activities in Correlation Sets

"10.1 Message Correlation
Both initiator and followers must mark the first activity in their respective groups as the activity that binds the correlation set."

```
context Scope
  inv firstActivityToUseACorrelationSetMustBindIt :
    let
      partnerActs : Sequence(bpel_11::PartnerActivity) =
        self.allOrderedSubActivities->flatten()->select(act |
          act.oclIsKindOf(bpel_11::PartnerActivity)
        )->oclAsType(Sequence(bpel_11::PartnerActivity))
    in
      self.correlationSet->forAll( corSet |
        let
          firstUse = partnerActs->select( act |
                       act.correlation.set->includes(corSet)
                     )->first()
        in
          firstUse.correlation->any(c|c.set=corSet).initiate
      )

  inv followingActivitiesToUseACorrelationSetMustNotBindIt :
    let
      partnerActs : Sequence(bpel_11::PartnerActivity)
        = self.allOrderedSubActivities->flatten()->select(act |
            act.oclIsKindOf(bpel_11::PartnerActivity)
          )->oclAsType(Sequence(bpel_11::PartnerActivity))
    in
```

```
      self.correlationSet->forAll( corSet |
        let
          useage = partnerActs->select( act |
                     act.correlation.set->includes(corSet)),
          followingUse = useage->subSequence(2,useage->size())
        in
          followingUse->forAll( a |
            not a.correlation->any(c|c.set=corSet).initiate)
      )
```

3.7 Source and Target of Activities

"11.2. Standard Elements for Each Activity
... An activity MAY declare itself to be the *source* of one or more links by including one or more <source> elements. Each <source> element MUST use a distinct link name. Similarly, an activity MAY declare itself to be the *target* of one or more links by including one or more <target> elements. Each <source> element associated with a given activity MUST use a link name distinct from all other <source> elements at that activity. Each <target> element associated with a given activity MUST use a link name distinct from all other <target> elements at that activity."

```
context bpel::Activity
  inv eachSourceElementMustUseDistinctLinkName :
    self.sourceOf.link->isUnique(s|s.name)

  inv eachTargetElementMustUseDistinctLinkName :
    self.targetOf.link->isUnique(s|s.name)
```

3.8 Instantiation of a Process

"11.4. Providing Web Service Operations
... The only way to instantiate a business process in BPEL4WS is to annotate a `receive` activity with the `createInstance` attribute set to "`yes`" (see 12.4. Pick for a variant). The default value of this attribute is "`no`". A `receive` activity annotated in this way MUST be an initial activity in the process, that is, the only other basic activities may potentially be performed prior to or simultaneously with such a `receive` activity MUST be similarly annotated `receive` activities."

```
context bpel_11::Receive
  inv allReceivesMarkedAsCreateInstanceMustBeInitialActivities :
     self.createInstance
    implies
      self.process.initialActivities->includes( self )

context bpel_11::BusinessProcess
  inv allInstantiationActivitiesAreMarkedCreateInstanceAsYes :
    self.instantiationActivities->forAll( act |
         act.oclIsTypeOf(bpel_11::Receive)
        and
         act.oclAsType(bpel_11::Receive).createInstance
      )
```

NB. There is an issue here regarding 'empty' initial activities, or structured activities that resolve to 'empty'. It also does not handle the 'Pick variant'.

"... It is permissible to have the `createInstance` attribute set to "`yes`" for a *set* of concurrent initial activities. In this case the intent is to express the possibility that any one of a set of *required* inbound messages can create the process instance because the order in which these messages arrive cannot be predicted. All such `receive` activities MUST use the same correlation sets (see 10. Correlation). Compliant implementations MUST ensure that only one of the inbound messages carrying the same correlation set tokens actually instantiates the business process (usually the first one to arrive, but this is implementation dependent). The other incoming messages in the concurrent initial set MUST be delivered to the corresponding receive activities in the already created instance."

```
context bpel_11::Flow
  inv AllConcurrentInstantiationActivitiesUseTheSameCorrelationSets :
    let
      instantiations: Set(bpel_11::Activity) = self.allSubActivities
      ->intersection(self.process.instantiationActivities )
    in
      instantiations->oclAsType(Set(bpel_11::Receive))
      .correlation.set->asSet()->size() <= 1
```

"... A business process instance MUST NOT simultaneously enable two or more `receive` activities for the same partnerLink, portType, operation and correlation set(s). Note that `receive` is a blocking activity in the sense that it will not complete until a matching message is received by the process instance. The semantics of a process in which two or more `receive` actions for the same partnerLink, portType, operation and correlation set(s) may be simultaneously enabled is undefined. For the purposes of this constraint, an `onMessage` clause in a `pick` and an onMessage event handler are equivalent to a `receive` (see 12.4. Pick and 13.5.1. Message Events)."

```
context bpel_11::Flow
  def : noSimilarReceives( set : Set(bpel_11::Activity) ) : Boolean =
    set->select(a|a.oclIsTypeOf(bpel_11::Receive))
    .oclAsType(Set(bpel_11::Receive))
    ->isUnique( rec |
      Tuple { partnerLink = rec.partnerLink,
              portType = rec.portType,
              operation = rec.operation,
              correlationSet = rec.correlation.set }
    )

  inv no_simultaneous_similar_receive_activities :
    self.nextGroups->forAll( ngs |
      self.noSimilarReceives( ngs )
    )
```

Note: This constraint makes use of the defined property 'nextGroups' which requires the use of a 'transitiveClosure' operation that is not part of the standard OCL.

"... The correlation between a request and the corresponding reply is based on the constraint that more than one outstanding synchronous request from a specific partner link for a particular portType, operation and correlation set(s) MUST NOT be outstanding simultaneously. The semantics of a process in which this constraint is violated is undefined. For the purposes of this constraint, an `onMessage` clause in a `pick` is equivalent to a `receive` (see 12.4. Pick)."

```
context bpel_11::Receive
  inv noSimultaneousOutstandingSynchronousRequests  :
    let
      request = self,
      actionSeq = self.allNext->flatten(),
      replySeq = actionSeq->select( a | a.oclIsTypeOf(bpel_11::Reply) )
                  .oclAsType(Sequence(bpel_11::Reply)),
      reply = replySeq->select( r |
                r.portType= request.portType and
                r.operation = request.operation and
                r.correlation = request.correlation
              )->first(),
      replyIndex = actionSeq->indexOf(reply)
    in
      not actionSeq->subsequence(0,replyIndex)
          ->select(a|a.oclIsTypeOf(bpel_11::Receive))
          .oclAsType(Sequence(bpel_11::Receive))
          ->exists( rec |
           rec.portType= request.portType and
           rec.operation = request.operation and
           rec.correlation = request.correlation
          )
```

Note: This constraint makes use of the defined property 'allNext' which requires the use of a 'transitiveClosure' operation that is not part of the standard OCL.

"... Moreover, a `reply` activity must always be preceded by a `receive` activity for the same partner link, portType and (request/response) operation, such that no reply has been sent for that `receive` activity. The semantics of a process in which this constraint is violated is undefined."

```
context bpel_11::Reply
  inv noExtraReplies  :
    let
      reply = self,
      actionSeq = self.allPrev->flatten(),
      requestSeq = actionSeq->select(a|a.oclIsTypeOf(bpel_11::Receive))
                    .oclAsType(Sequence(bpel_11::Receive)),
      request = requestSeq->select( r |
                r.portType= reply.portType and
                r.operation = reply.operation and
                r.correlation = reply.correlation
```

```
            )->first(),
      requestIndex = actionSeq->indexOf(request)
    in
      not actionSeq->subsequence(0,requestIndex)
          ->select(a|a.oclIsTypeOf(bpel_11::Reply))
          .oclAsType(Sequence(bpel_11::Reply))
          ->exists( rep |
           rep.portType= request.portType and
           rep.operation = request.operation and
           rep.correlation = request.correlation
          )
```

Note: This constraint makes use of the defined property ‘allPrev’ which requires the use of a ‘transitiveClosure’ operation that is not part of the standard OCL.

3.9 Pick

> “12.4 Pick
> … A special form of `pick` is used when the creation of an instance of the business process could occur as a result of receiving one of a set of possible messages. In this case, the `pick` itself has a `createInstance` attribute with a value of `yes` (the default value of the attribute is `no`). In such a case, the events in the `pick` must all be inbound messages and each of those is equivalent to a `receive` with the attribute "`createInstance=yes`". No alarms are permitted for this special case.“

```
context bpel::Pick
  inv createInstancePickImpliesAllEventsAreCreateInstanceReceives :
     self.createInstance
    implies
      self.onMessage->forAll( act | act.createInstance )
     and
      self.onAlarm->isEmpty()
```

> “… Each pick activity MUST include at least one `onMessage` event.”

This constraint is imposed by the 1..* multiplicity on the Pick-OnMessage association. However as EMF does not generate code that enforces this we add an invariant to check it.

```
context bpel::Pick
  inv pick_onMessage_MultiplicityAtLeastOne :
    self.onMessage->size() >= 1
```

3.10 Flows and Links

> “12.5 Flow
> … A link has a name and all the links of a flow activity MUST be defined separately within the flow activity.”

This constraint is imposed by the Model. Links are contained by a Flow (and can’t be included anywhere else). Links only have two ends, therefore each link is separate.

> “…The source of the link MUST specify a `source` element specifying the link's name and the target of the link MUST specify a `target` element specifying the link's name.”

This constraint is imposed by the Model. Within a Link, Source and Target elements are not optional.

> “… Every link declared within a flow activity MUST have exactly one activity within the flow as its source and exactly one activity within the flow as its target. The source and target of a link MAY be nested arbitrarily deeply within the (structured) activities that are directly nested within the flow, except for the boundary-crossing restrictions.
> …
> In general, a link is said to *cross the boundary* of a syntactic construct if the source activity for the link is nested within the construct but the target activity is not, or *vice versa*, if the target activity for the link is nested within the construct but the source activity is not.”

To express this constraint in OCL, we require the method `subActivities` to be defined for each subtype of Activity. The method returns a set containing all activities directly nested within that Activity. Also required is a method `allSubActivities` which returns all nested and sub-nested activities. For basic Activities this set will typically be empty. These operations are defined in section 4.

The constraint requiring the source and target activity for each link of a flow to be contained with the flow is expressed as follows:

```
context bpel_11::Flow
  inv sourceAndTargetActivitiesAreContainedWithinTheFlow :
    self.link->forAll( lnk |
        self.allSubActivities->includes( lnk.source.activity )
      and
        self.allSubActivities->includes( lnk.target.activity )
    )
```

"... A link MUST NOT cross the boundary of a while activity, a serializable scope, an event handler or a compensation handler (see 13. Scopes for the specification of event, fault and compensation handlers)."

```
context bpel::While
  inv boundryCrossing :
    allSubActivities->includesAll(self.allSubActivities.sourceOf.activity)
   and
    allSubActivities->includesAll(self.allSubActivities.targetOf.activity)
```

```
context bpel::Scope
  inv boundryCrossing :
     not self.variableAccessSerializable.oclIsUndefined()
    implies (
      self.variableAccessSerializable
     implies
       self.allSubActivities->includesAll(
                               self.allSubActivities.sourceOf.activity)
      and
       self.allSubActivities->includesAll(
                               self.allSubActivities.targetOf.activity)
    )
```

```
context bpel::EventHandler
  inv boundryCrossing :
     self.activity.allSubActivities->includesAll(
                   self.activity.allSubActivities().sourceOf.activity)
    and
     self.activity.allSubActivities->includesAll(
                   self.activity.allSubActivities().targetOf.activity)
```

"... In addition, a link that crosses a fault-handler boundary MUST be outbound, that is, it MUST have its source activity within the fault handler and its target activity within a scope that encloses the scope associated with the fault handler."

```
context bpel_11::FaultHandler
  inv boundryCrossing :
   let allSubActivities =
     Set { self.catchAll }
     ->union( self.catch.activity->asSet() ).allSubActivities
   in
      allSubActivities->includesAll( allSubActivities.sourceOf.activity )
     and
      allSubActivities.sourceOf.link.target.activity->forAll( tgtAct |
        self.scope.allParents->exists( act |
          act.allSubActivities->includes(tgtAct)
        )
      )
```

"... Finally, a link MUST NOT create a control cycle, that is, the source activity must not have the target activity as a logically preceding activity, where an activity A logically precedes an activity B if the initiation of B semantically requires the completion of A. Therefore, directed graphs created by links are always acyclic."

```
context bpel_11::Link
  inv noControlCycles :
    self.source.allPrev->flatten()->excludes(self.target)
```

Note: This constraint makes use of the defined property 'allPrev' which requires the use of a 'transitiveClosure' operation that is not part of the standard OCL.

"12.5.1. Link Semantics

... The expression for a join condition for an activity MUST be constructed using *only* Boolean operators and the `bpws:getLinkStatus` function (see 9.1. Expressions) applied to incoming links at the activity."

To statically check this constraint it would be necessary to parse the body of an Expression construct; it would be in appropriate to attempt this using OCL.

4 Defined Properties and Operations

To construct the constraints defined in the previous section a number of additional operations and properties have been defined on classes within the BPEL package. These definitions are given using OCL in the following subsections. Figure 11 gives a class diagram overview of the additional operations and properties.

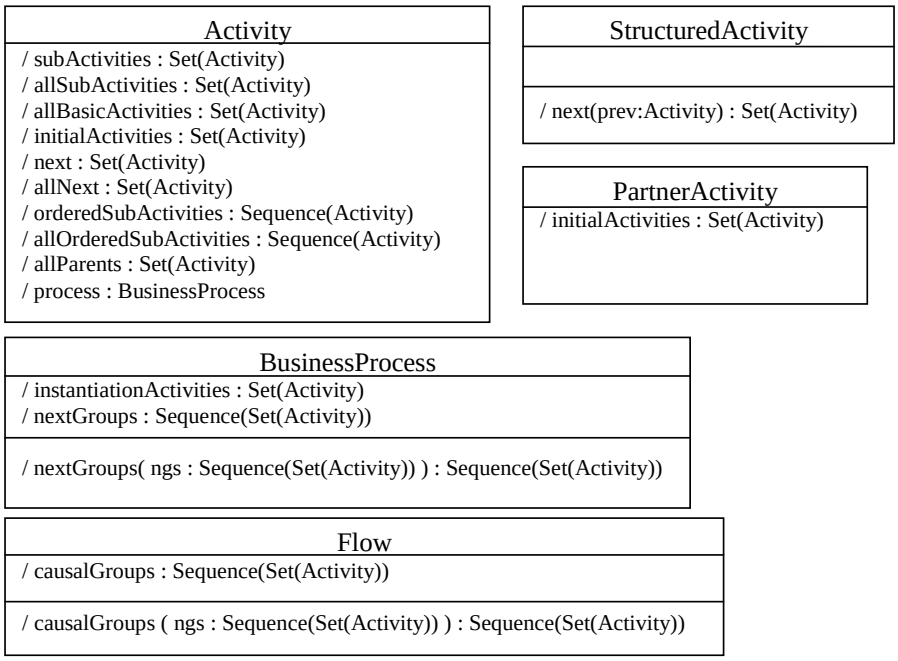

Figure 11

4.1 Define Property - subActivities : Set

```
context bpel_11::Activity
  def: subActivities : Set(bpel_11::Activity) = Set {}
```

This property is defined as returning an empty Set for the class Activity. Each subtype that contains sub activities redefines the property as follows:

Note: activities in compensation, fault and event handlers *are* treated as subActivities.

```
context bpel_11::OnMessage
  def: subActivities : Set(bpel_11::Activity) = Set { self.activity }

context bpel_11::Invoke
  def: subActivities : Set(bpel_11::Activity) =
    let
      -- undefined's are not put into sets, so these are empty
      -- if navigations are undefined
      compHndlr : Set(bpel_11::Activity) = Set{self.compensationHandler},
      fltHndlr : Set(bpel_11::Activity)  =
        Set { self.faultHandler.catchAll }
        ->union( self.faultHandler.catch.activity->asSet() )
    in
      fltHndlr->union( compHndlr )

context bpel_11::While
  def: subActivities : Set(bpel_11::Activity) = Set { self.activity }

context bpel_11::Flow
  def: subActivities : Set(bpel_11::Activity) = self.activity->asSet()

context bpel_11::ActivitySequence
  def: subActivities : Set(bpel_11::Activity) = self.activity->asSet()

context bpel_11::Pick
  def: subActivities : Set(bpel_11::Activity) =
    self.onAlarm.activity->union( self.onMessage )->asSet()

context bpel_11::Switch
  def: subActivities : Set(bpel_11::Activity) =
    self.case.activity->asSet()->union( Set{self.otherwise} )
```

```
context bpel_11::Scope
  def: subActivities : Set(bpel_11::Activity) =
    let
      -- undefined's are not put into sets, so these are empty
      -- if navigations are undefined
      compHndlr : Set(bpel_11::Activity) = Set{self.compensationHandler},
      fltHndlr  : Set(bpel_11::Activity) =
        Set { self.faultHandler.catchAll }
        ->union( self.faultHandler.catch.activity->asSet() ),
      evntHndlr : Set(bpel_11::Activity) =
        Set { self.eventHandler.activity }->flatten()
    in
      Set{self.activity}
      ->union(compHndlr)
      ->union(fltHndlr)
      ->union(evntHndlr)
```

4.2 Define Property - allSubActivities : Set

```
context bpel_11::Activity
  def: allSubActivities : Set(bpel_11::Activity) =
    self.subActivities
    ->union( self.subActivities.allSubActivities->asSet() )
```

4.3 Definition of Property - initialActivities : Set

This property is intended to return a Set containing all possible activities that could occur if this activity is expected to occur. For basic activities this typically returns a set containing itself. In addition structured activities do not typically included themselves; rather they include the initial activities of those that are contained within.

4.3.1 Basic Activities

```
context bpel_11::Activity
  def: initialActivities : Set(bpel_11::Activity) = Set{self}
```

4.3.2 Partner Activities

```
context bpel_11::Receive
  def: initialActivities : Set(bpel_11::Activity) = Set{self}

context bpel_11::Reply
  def: initialActivities : Set(bpel_11::Activity) = Set{self}

context bpel_11::Invoke
  def: initialActivities : Set(bpel_11::Activity) = Set{self}
```

4.3.3 Structured Activities

```
context bpel_11::While
  def: initialActivities : Set(bpel_11::Activity) =
    self.activity.initialActivities

context bpel_11::Flow

  def: initialActivities : Set(bpel_11::Activity) =
    self.activity->select( a |
      a.targetOf->isEmpty()
    ).initialActivities->asSet()

context bpel_11::ActivitySequence
  def: initialActivities : Set(bpel_11::Activity) =
    self.activity->first().initialActivities

context bpel_11::Pick
  def: initialActivities : Set(bpel_11::Activity) =
    self.onAlarm.activity.initialActivities
    ->union( self.onMessage.initialActivities )
    ->asSet()

context bpel_11::Switch
  def: initialActivities : Set(bpel_11::Activity) =
    let otherW = if self.otherwise.oclIsUndefined() then
                   Set{}
                 else
                   self.otherwise.initialActivities
                 endif
    in
```

```
      self.case.activity.initialActivities->asset()
      ->union( otherW )->asSet()

context bpel_11::Scope
  def: initialActivities : Set(bpel_11::Activity) =
    self.activity.initialActivities
```

4.4 Definition of Property -allBasicActivities : Set

4.4.1 Basic Activities

```
context bpel 11::Activity
  def: allBasicActivities : Set(bpel_11::Activity) = Set{self}
```

4.4.2 Partner Activities

```
-- OnMessage is a subtype of Recieve

context bpel_11::Receive
  def: allBasicActivities : Set(bpel_11::Activity) = Set{self}

context bpel 11::Reply
  def: allBasicActivities : Set(bpel_11::Activity) = Set{self}

context bpel 11::Invoke
  def: allBasicActivities : Set(bpel_11::Activity) = Set{self}
```

4.4.3 Structured Activities

```
context bpel 11::While
  def: allBasicActivities : Set(bpel 11::Activity) =
    self.activity.allBasicActivities->asSet()

context bpel 11::Flow
  def: allBasicActivities : Set(bpel_11::Activity) =
    self.activity.allBasicActivities->asSet()

context bpel_11::ActivitySequence
  def: allBasicActivities : Set(bpel_11::Activity) =
    self.activity.allBasicActivities->asSet()

context bpel_11::Pick
  def: allBasicActivities : Set(bpel 11::Activity) =
    self.onAlarm.activity.allBasicActivities
    ->union( self.onMessage.allBasicActivities )->asSet()

context bpel 11::Switch
  def: allBasicActivities : Set(bpel 11::Activity) =
    let otherW = if self.otherwise.oclIsUndefined() then
                   Set{}
                 else
                   self.otherwise.allBasicActivities
                 endif
    in
     self.case.activity.allBasicActivities->asSet()
     ->union( otherW )->asSet()

context bpel 11::Scope
  def: allBasicActivities : Set(bpel_11::Activity) =
    self.activity.allBasicActivities
```

4.5 Definition of Property - next : Set

Property ‘prev’ is similarly defined; however the definition is not explicitly given.

4.5.1 Basic and Partner Activities

```
context bpel 11::Activity
  def: next : Set(bpel 11::Activity) =
    self.parent.next(self)->collect( n |
      if n.oclIsTypeOf(bpel_11::StructuredActivity) then
        n.initialActivities
      else
        Set{n}
      endif
    )->flatten()->asSet()
```

4.5.2 Structured Activities

```
context bpel_11::StructuredActivity
  def: next(prev:bpel_11::Activity) : Set(Activity) = Set{}

context bpel_11::While
  def: next(prev:bpel_11::Activity) : Set(Activity) =
    self.activity.initialActivities->union( self.parent.next(self) )

context bpel_11::Pick
  def: next(prev:bpel_11::Activity) : Set(Activity) =
    Set { self.parent.next(self) }->flatten()

context bpel_11::Flow
  def: next(prev:bpel_11::Activity) : Set(Activity) =
    self.activity->reject( a |
      a.allBasicActivities->includes(prev)
    ).allBasicActivities->asSet()

context bpel_11::ActivitySequence
  def: next(prev:bpel_11::Activity) : Set(Activity) =
    let
      i = self.activity->indexOf(prev)
    in
      Set{ self.activity->at(i+1) }.initialActivities->asSet()

context bpel_11::Switch
  def: next(prev:bpel_11::Activity) : Set(Activity) =
    Set { self.parent.next(self) }->flatten()

context bpel_11::Scope
  def: next(prev:bpel_11::Activity) : Set(Activity) =
    Set { self.parent.next(self) }->flatten()
```

4.6 Definition of Property - allNext : Set

```
context bpel_11::Activity
  def: allNext : Sequence(Set(bpel_11::Activity)) =
    Sequence{Set{self}}->transitiveClosure(x|x.next->asset())
```

4.7 Definition of Property - orderedSubActivities : Set

Note: activities in compensation, fault and event handlers *are* treated as subActivities.

4.7.1 Basic Activities

```
context bpel_11::Activity
  def: orderedSubActivities : Sequence(bpel_11::Activity) = Sequence {}
```

4.7.2 Partner Activities

```
context bpel_11::OnMessage
  def: orderedSubActivities : Sequence(bpel_11::Activity) =
    Sequence { self.activity }

context bpel_11::Invoke
  def: orderedSubActivities : Sequence(bpel_11::Activity) =
    let
      -- undefined's are not put into sets, so these are empty
      --  if navigations are undefined
      compHndlr : Sequence(bpel_11::Activity) =
        Sequence { self.compensationHandler },
      fltHndlr : Sequence(bpel_11::Activity)  =
        Sequence { self.faultHandler.catchAll }
        ->union( self.faultHandler.catch.activity->asSequence() )
    in
      fltHndlr->union( compHndlr )
```

4.7.3 Structured Activities

```
context bpel_11::While
  def: orderedSubActivities : Sequence(bpel_11::Activity) =
    Sequence { self.activity }

context bpel_11::Flow
  def: orderedSubActivities : Sequence(bpel_11::Activity) =
    self.activity->asSequence()
```

```
context bpel_11::ActivitySequence
  def: orderedSubActivities : Sequence(bpel_11::Activity) =
    self.activity

context bpel_11::Pick
  def: orderedSubActivities : Sequence(bpel_11::Activity) =
    self.onAlarm.activity->asSequence()
    ->union( self.onMessage->asSequence() )

context bpel 11::Switch
  def: orderedSubActivities : Sequence(bpel 11::Activity) =
    self.case->collectNested(c|c.activity)->including( self.otherwise )

context bpel 11::Scope
  def: orderedSubActivities : Sequence(bpel_11::Activity) =
    let
      -- undefined's are not put into sets, so these are empty
      --  if navigations are undefined
      compHndlr : Sequence(bpel 11::Activity) =
        Sequence { self.compensationHandler },
      fltHndlr  : Sequence(bpel_11::Activity) =
        Sequence { self.faultHandler.catchAll }
        ->union( self.faultHandler.catch.activity->asSequence() ),
      evntHndlr : Sequence(bpel_11::Activity) =
        self.eventHandler.activity->asSequence()
    in
      Sequence{self.activity}
      ->union(compHndlr)
      ->union( fltHndlr )
      ->union(evntHndlr)
```

4.8 Definition of Property - allOrderedSubActivities : Set

```
context bpel 11::Activity
  def: allOrderedSubActivities : Sequence(OclAny) =
    if self.oclIsKindOf(bpel 11::StructuredActivity) then
     self.orderedSubActivities->collectNested(act |
        act.allOrderedSubActivities ).oclAsType(Sequence(OclAny))
    else
      Sequence{self}->union( self.orderedSubActivities )
    endif
```

4.9 Definition of Property - allParents : Set

```
context bpel_11::Activity
  def: allParents : Set(bpel_11::Activity) =
    self.parent->union(self.parent.allParents->asSet())
```

4.10 Definition of Property - process : BusinessProcess

```
context bpel 11::Activity
  def: process : bpel 11::BusinessProcess =
    if self.parent.oclIsUndefined() then
      self.oclAsType(bpel_11::BusinessProcess)
    else
      self.parent.process
    endif
```

4.11 Definition of Property - instantiationActivities : Set

Note: The Pick Variant is not handled.

```
context bpel_11::BusinessProcess
  def : instantiationActivities : Set(bpel_11::Activity) =
    self.allBasicActivities->select( a |
       a.oclIsKindOf(bpel_11::Receive)
      and
       a.oclAsType(bpel_11::Receive).createInstance
    )
```

4.12 Definition of Property - causalGroups : Set

```
context bpel 11::Flow
  def : causalGroups( cgs : Sequence(Set(bpel_11::Activity)) )
        : Sequence(Set(bpel_11::Activity)) =
```

```
  let
    ng : Set(bpel_11::Activity) = cgs->last().next->asSet()
  in
    if ng->isEmpty() then
      cgs
    else
      self.causalGroups( cgs->append(ng) )
    endif

def : causalGroups : Sequence(Set(bpel_11::Activity)) =
 self.causalGroups( Sequence{ self.initialActivities } )
```

5 Generating the BPEL Validation Code

To create the Java Code for the OCL constraints defined in sections 3 and 4 above we use the OCL library developed at the University of Kent. The following code defines a command line application that will create appropriate java code for a set of OCL constraints.

```
package uk.ac.kent.cs.bpel;

import org.eclipse.xsd.XSDPackage;
import com.ibm.mda.bpel_11.BPELPackage;
import com.ibm.mda.wsdl.WSDLPackage;
import uk.ac.kent.cs.kmf.util.ILog;
import uk.ac.kent.cs.kmf.util.OutputStreamLog;
import uk.ac.kent.cs.ocl20.OclProcessor;
import uk.ac.kent.cs.ocl20.bridge4emf.CodeGenerator;
import uk.ac.kent.cs.ocl20.bridge4emf.EmfOclProcessorImpl;

public class CreateCodeFromOclFile {

  public static void main(String[] args) {
    // Initialize the model and log
    ILog log = new OutputStreamLog(System.out);
    OclProcessor processor = new EmfOclProcessorImpl(log);
    processor.addModel(XSDPackage.eINSTANCE);
    processor.addModel(WSDLPackage.eINSTANCE);
    processor.addModel(BPELPackage.eINSTANCE);

    // Generate code
    CodeGenerator codeGen =
      new CodeGenerator(processor, args[0],args[1],args[2],args[3], log);
    codeGen.generate();

    // Print the compilation report
    log.printMessage("Done.\n");
    log.finalReport();
  }

}
```

The parameters required by the CodeGenerator class, and hence by this command line application are as follows:

processor – an instance of the OclProcessor class (not a command line parameter).
inputFileName – a string denoting the file containing OCL expressions to generate code from.
outputDir – the directory into which the generated Java class files should be put.
pkgName – the name of the package into which the generated Java classes should be put.
outputClassName – the name of the primary generated java class containing code for the constraints.
log – an instance of the ILog class for output from the OCL library (not a command line parameter).

For example:

```
java CreateCodeFromOclFile ocl/bpel.ocl src bpel_11 Invariants
```

Will generate code for the OCL constraints contained in the file ‘ocl/bpel.ocl’. It will generate a class ‘bpel_11.Invariants’ and place it in the directory ‘src’.

6 The generated Code

The code generated for the OCL constraints is structured into a number of static methods contained in sub classes of a main containing class. In the context of the example shown in the previous section, the main class was named ‘Invariants’. The generated code is placed in a class with this name in a package named ‘bpel_11’.

Each invariant specified on a class is placed in a nested sub class of *Invariants* in a method with the name of the invariant (or a default name if no invariant name was specified). The name of the nested sub class matches that of the context class for the invariant; e.g. the invariant:

```
context bpel::BusinessProcess
  inv processIsNotASubActivity :
    self.parent.oclIsUndefined()
```

generates the following code:

```
package bpel 11;
public class Invariants

  public static class BusinessProcess {

    public static java.lang.Boolean processIsNotASubActivity(
                      com.ibm.mda.bpel 11.BusinessProcess self) {
      /* self.parent.oclIsUndefined() */
      try {
         return new Boolean((self.getParent() == null));
      } catch (Exception e) {
         return null;
      }
    }

  }

}
```

and to evaluate this invariant for a particular object we could use the code snipet:

```
BusinessProcess busProc;

... // some code to assign busProc a value

Boolean b = Invariants.BusinessProcess.processIsNotASubActivity(busProc);
```

In addition there is a method generated for each nested sub class named ‘evaluateAll’ which will evaluate all invariants defined for a particular object and return a java.util.Map containing the result of evaluating each invariant along side a key with the name of that invariant. E.g.

```
BusinessProcess busProc;

... // some code to assign busProc a value

Map m = Invariants.BusinessProcess.evaluateAll(busProc);
```

7 A BPEL Validation Application

Using the generated code for the defined OCL constraints along with:

- code generated by EMF for the BPEL 1.1 metamodel and
- a bespoke reader that imports a BPEL XML into instances of the EMF generated code

we can write a small application that will validate any BPEL document.
The following code is just such an application:

```
package uk.ac.kent.cs.bpel;

import java.lang.reflect.Method;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

import org.eclipse.emf.common.util.TreeIterator;
import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.EObject;
import org.eclipse.emf.ecore.resource.Resource;
import org.eclipse.emf.ecore.resource.ResourceSet;
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;

import com.ibm.mda.bpel.resource.BPELResource;
import com.ibm.mda.bpel.resource.BPELResourceFactoryImpl;
import com.ibm.mda.bpel_11.impl.BPELPackageImpl;

public class Validator {

  public static void main(String[] args) {
    String bpelFileName = args[0];
    BPELPackageImpl.init();
    Resource.Factory.Registry reg = Resource.Factory.Registry.INSTANCE;
    Map m = reg.getExtensionToFactoryMap();
    m.put("bpel", new BPELResourceFactoryImpl());
    ResourceSet resSet = new ResourceSetImpl();
    Resource res = resSet.getResource(URI.createURI(bpelFileName), true);

    System.out.println("\n----------------------------------------");
    System.out.println("\nValidating contents of resource "+res.getURI());
    System.out.println("\n----------------------------------------\n");
    Validator v = new Validator();
    TreeIterator te = res.getAllContents();
    while (te.hasNext()) {
      Object o = te.next();
      if (o instanceof EObject) {
         v.validate((EObject) o);
      }
    }
    System.out.println("Done.");
  }

  public void validate(EObject obj) {
    EClass objCls = obj.eClass();
    Set allClasses = new HashSet(objCls.getEAllSuperTypes());
    allClasses.add(objCls);
    Iterator i = allClasses.iterator();
    while (i.hasNext()) {
      EClass cls = (EClass)i.next();
      String clsName = cls.getName();
      try {
        Class invCls = getInvClass(clsName);
        if (invCls != null) {
          Method evalAll = invCls.getMethod("evaluateAll",
                              new Class[] { cls.getInstanceClass()});
          Map m = (Map) evalAll.invoke(null, new Object[] { obj });
          print(m, obj, clsName);
        }
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }
```

```
  Map invClasses = new HashMap();
  public Class getInvClass(String clsName) {
    if (!invClasses.containsKey(clsName)) {
      String invClsName = "bpel_11.Invariants";
      try {
        Class ic = null;
        Class inv = Class.forName(invClsName);
        Iterator i = Arrays.asList(inv.getClasses()).iterator();
        while (i.hasNext()) {
          Class c = (Class) i.next();
          if (c.getName().equals(invClsName + "$" + clsName))
            ic = c;
        }
        if (ic == null) {
          invClasses.put(clsName, null);
        } else {
          invClasses.put(clsName, ic);
        }
      } catch (ClassNotFoundException e) {
        System.out.println("No " + invClsName + " Class found");
      }
    }
    return (Class) invClasses.get(clsName);
  }

  static public void print(Map m, Object self, String clsName) {
    String selfId = "";
    if(self instanceof EObject) {
      EObject obj = (EObject)self;
      BPELResource o = (BPELResource)obj.eResource();
      selfId = o.getURIFragment(obj);
    }

    Map notTrues = new HashMap();
    Iterator i = m.entrySet().iterator();
    while (i.hasNext()) {
      Map.Entry me = (Map.Entry) i.next();
      if (me.getValue() instanceof Boolean &&
          me.getValue().equals(Boolean.TRUE) ) {
      } else {
          notTrues.put(me.getKey(), me.getValue());
      }
    }

    if ( ! notTrues.entrySet().isEmpty() ) {
      System.out.println("\ncontext self = " + self + "\n[" +selfId+ "]");
      System.out.println(" : "+clsName);
      i = notTrues.entrySet().iterator();
      while (i.hasNext()) {
        Map.Entry me = (Map.Entry) i.next();
        System.out.println(me.getKey() + ":\n" + me.getValue());
      }
    }
  }

}
```

This application takes a single command line parameter which is the name of the BPEL document to validate; e.g.

```
java Validator examples/ShippingService/process.bpel
```

8 Files and Resources

To reproduce the production of the BPEL validator it is necessary to have the following resources:

- **bpel.ocl** - text file containing the specification OCL statements that constraint a BPEL document.
- **BPEL_1.1.cat** – Rational Rose file containing the specification of the BPEL version 1.1 metamodel. This file make used of these additional files, all included in **bpel_1.1.mdl**.
    - **WSDL_1.0.2.cat**
    - org.eclipse.XSD.cat
    - org.eclipse.emf.Ecore.cat
- The Eclipse Modelling framework, to generate code for the BPEL model. Code for the WSDL and XSD models are found in the following jars
    - **wsdl.jar** and wsdl4j.jar
    - xsd.jar
- **bpel_11.jar** - Hand written BPEL reader and resource factory for reading BPEL XML files into the EMF generated code for the BPEL metamodel. The jar includes the EMF generated code for the BPEL 1.1 metamodel.
- **OCLforEMF plugins**, including the following jars
    - OclCommon.jar – standard parts of the Kent OCL library.
    - Ocl4EMF.jar – bridge for using the Kent OCL library with EMF code.
    - KMFpatterns.jar, KMF_Util.jar – utilities used by the Kent OCL library.
    - CUPRuntime.jar – code used by the Kent OCL parser, generated using the CUP parser generator.

These jars are contained in the following list of plugins:

Supplier	Name	Version	Id
University of Kent	BPEL Validator	1.0.0	uk.ac.kent.cs.bpel.validator
University of Kent	BPEL Model	1.1.0	uk.ac.kent.cs.bpel
IBM	com.ibm.mda.wsdl	1.0.2	com.ibm.mda.wsdl
Eclipse.org	XML Schema Infoset Model (XSD)	2.0.0	org.eclipse.xsd
…	other eclipse and EMF plugins	…	…
University of Kent	OCL for EMF	1.1.2	uk.ac.kent.cs.forEMF
University of Kent	OCL Common	1.2.2	uk.ac.kent.cs.ocl.common
University of Kent	OCL Utils	1.1.0	uk.ac.kent.cs.ocl.utils

9 Testing

The validator and constraints have been tested on the following BPEL examples taken from the BPEL standard:

- Initial example
- ShippingService example

and on the following examples supplied by IBM:

- SyncHelloWorld
- FlightService
- HotelService
- TripHandlingSimple
- TripHandlingConcurrent
- Philosopher[1..5]
- RandomHalting
- PhilosopherTableFixedLinkFive
- PhilosopherTableSimpleLinkFive
- PhilosopherTableSimpleLinkThree

Bibliography

[1] IBM, "Business Process Execution Language for Web Services," 2003.
[2] OMG, "Response to the UML 2.0 OCL Rfp (ad/2000-09-03), Revised Submission, Version 1.6," Object Management Group ad/2003-01-07, January 2003 2002.
[3] IBM, "Eclipse Modeling Framework," http://www.eclipse.org/emf/, 2003.