Format Requirements Specification – MSE Project

Doug Smith

This document presents an OCL specification of the system I am building as part of this project. Section 1 contains the requirements model, Section 2 contains a test definition, instantiation, and execution of a process using a parallel gateway and join, and Section 3 contains a test of a process test definition, instantiation, and execution of a process using an exclusive gateway and join.

Section 1 – Requirements Model

model MSE

enum ActivityState {pending, active, complete}
enum GatewayType { exclusive, parallel }
enum PropertyType { boolean, string, date, decimal, integer }
-- Process metadata

--Process definition
class ProcessDefinition
attributes
    name : String
    description : String

operations
    retrieveDefinition(name : String) : ProcessDefinition
end

--Swimlane definition
class Swimlane
attributes
    name : String
end

--Connectable object. This is an abstract base class for process definition
--elements that can be connected.
abstract class Connectable
end

--Gateway definition
class GatewayDefinition < Connectable
attributes
    type : GatewayType
end

association ProcessGateways between
    ProcessDefinition[1]
    GatewayDefinition[*]
end

--Connector definition
class ConnectorDefinition
attributes
    source : Connectable
    sink : Connectable
    expression : String
end

-- USE does not support OCL 2 send message and response received
-- constructs, and thus we can't write post conditions that can
-- say the connector expression has been evaluated true. We will
-- abstract this in the specification via a boolean property.
--
-- In OCL 2, calling a method can be specified in a post condition,
-- and testing that the call returned with a specific value is
-- supported, e.g.
--
-- context Person::giveSalary(amount : Integer)
-- post: let message : OclMessage =
--     company^^getMoney(amount)->any( true )
--     in message.hasMoreReturned() and
--     message.result() = true
-- (Example from The Object Constraint Language, Second Edition,
--   Jos Warmer and Anneke Kleppe, Addison-Wesley, 2003

connExpressionIsTrue : Boolean

operations
    expressionSatisfied(activity: ActivityInstance) : Boolean
end

association ProcessConnections between
    ProcessDefinition[1]
    ConnectorDefinition[*] role connectors
end

-- Activity Definition
class ActivityDefinition < Connectable
attributes
    name : String
    description : String
    isStart : Boolean
    isEnd : Boolean
end

association ProcessActivities between
    ProcessDefinition[1]
    ActivityDefinition[*] role activities
end

association SwimlaneActivities between
    Swimlane[1]
    ActivityDefinition[*]
end

-- Property definition
class PropertyDefinition
attributes
    name : String
    type : PropertyType
    description : String
operations
retrieveList() : Set(PropertyDefinition)
updateDescription( name : String, newDescription : String)
end

association ActivityProperties between
  ActivityDefinition[*]
  PropertyDefinition[*] role properties
end

--Role definition
class Role
attributes
  name : String
  description : String
end

association ActivityRoles between
  ActivityDefinition[*]
  Role[*] role activityRoles
end

--User definition
class User
attributes
  username : String
  userid : String
end

association UserRoles between
  User[*]
  Role[*] role userRoles
end

-- Process instances
--Process Factory for instantiating new process instances
class ProcessFactory
operations
  instantiateProcess(processName : String) : ProcessInstance
end

--Process instance
class ProcessInstance
attributes
  instanceId : Integer
end

association ProcessInstanceActivities between
  ProcessInstance[1]
  ActivityInstance[*] role processActivities
end

association ProcessInstanceDefinition between
  ProcessInstance[*]
  ProcessDefinition[1] role processDefinition
end

--Activity instance
class ActivityInstance
attributes
  activityId : Integer
swimlaneName : String
state : ActivityState

operations
  listActivitiesAtASwimlane( swimlaneName : String)
    : Set(ActivityInstance)
  claimActivity( aUserId : String, claimedActivityId : Integer )
    : ActivityInstance
  executeActivity( aUserId : String )
  releaseClaim( aUserId : String, activityId : Integer )
end

association ActivityInstancesProperties between
  ActivityInstance[1]
  PropertyValue[*] role propertyValues
end

association ActivityInstanceDefinition between
  ActivityInstance[*]
  ActivityDefinition[1] role activityDefinition
end

-- Property values
class PropertyValue
attributes
  name : String
  value : String
end

-- Activity claims. An activity claim is a locking mechanism to prevent
two users to work on the same activity simultaneously.
class ActivityClaim
attributes
  claimant : User
  claimed : ActivityInstance
end

--------------------------------------------------
-- Model Constraints
--------------------------------------------------
constraints

-- User
context User
  -- User IDs are unique
  inv uniqueIds: User.allInstances() ->isUnique(userid)

-- Swimlane
context Swimlane
  -- Swimlane names are unique
  inv uniqueNames: Swimlane.allInstances() ->isUnique(name)

-- ConnectorDefinition
--
context ConnectorDefinition
   --Nothing is connected to itself
   inv noConnectionToSelf:
      ConnectorDefinition.allInstances() -> select(c|
         c.source = c.sink) -> isEmpty()

   --Gateways are not connected to gateways. This represents a simplification
   --of the system.
   inv noGatewayToGateway:
      ConnectorDefinition.allInstances() -> select(c|
         c.source.oclIsKindOf(GatewayDefinition)
         and
         c.sink.oclIsKindOf(GatewayDefinition)) -> isEmpty()

   --The source and sink of all connector definitions must reference
   --a connectable definition
   inv sourceAndSinkReferenceStuff:
      ConnectorDefinition.allInstances() -> forAll(cd|
         Connectable.allInstances() -> includes(cd.source) and
         Connectable.allInstances() -> includes(cd.sink))

   --
   -- Connectable
   --
context Connectable
   --All connectable model elements are in fact connected.
   inv allConnected:
      Connectable.allInstances() -> forAll(c|
         ConnectorDefinition.allInstances() -> includes(c) or
         ConnectorDefinition.allInstances() -> includes(c) )

   --
   -- ActivityDefinition
   --
context ActivityDefinition
   --An activity can be a start activity or a stop activity, but not both.
   inv startOrStop:
      ActivityDefinition.allInstances() -> forAll(ad|
         (ad.isStart = true implies ad.isEnd = false)
         and
         (ad.isEnd = true implies ad.isStart = false))

   --
   -- GatewayDefinition
   --
context GatewayDefinition
   --Gateways can be fan-on or fan-out, but not both.
   inv oneInManyOutOrManyInOneOut:
      GatewayDefinition.allInstances() -> forAll(gd|
         (ConnectorDefinition.allInstances() -> select(sink=gd)
          -> size() = 1
         implies
         ConnectorDefinition.allInstances() -> select(source=gd)
          -> size() >= 1)
         and
         (ConnectorDefinition.allInstances() -> select(sink=gd)
          -> size() > 1
         implies
ConnectorDefinition.allInstances().select(source=gd)->size() = 1

-- ProcessDefinition

context ProcessDefinition

-- Each process can have a single start activity associated with it
inv oneStart:
  self.activities->select(isStart = true)->size() = 1

-- Each process can have a single end activity associated with it
inv oneEnd:
  self.activities->select(isEnd = true)->size() = 1

-- No redundant/duplicate connectors
inv noDuplicates:
  self.connectors->forAll(c|
    self.connectors->select(source = c.source and
      sink = c.sink)->size() = 1)

-- Process definitions have unique names
inv uniqueNames:
  ProcessDefinition.allInstances()->isUnique(name)

context ProcessDefinition::retrieveDefinition(name : String) :
  ProcessDefinition

-- Can't retrieve a process definition that does not exist
pre mustExist:
  ProcessDefinition.allInstances()->select(name=name)->size() = 1

post resultOk:
  result = ProcessDefinition.allInstances()->select(name=name)->asSequence()->first()

-- ProcessInstance

context ProcessInstance

-- Each process instance has a unique id
inv uniqueInstanceIds:
  ProcessInstance.allInstances()->isUnique(instanceId)

context ProcessFactory::instantiateProcess(processName : String) :
  ProcessInstance

-- Instantiate only known processes
pre knownProcs:
  ProcessDefinition.allInstances()->exists(p|p.name = processName)

-- Post condition is there's one more process instance of that type.
post oneMore:
  let newProcess : ProcessInstance =
    (ProcessInstance.allInstances()-@pre)
      ->asSequence()->first()
  in
    newProcess.processDefinition.name = processName

-- New start activity instantiated with an active status
post startActive:
  let newInstance : ProcessInstance =
    (ProcessInstance.allInstances()-
ProcessInstance.allInstances().@pre ->asSequence()->first()
in
newInstance.processActivities
->select(activityDefinition.isStart=true)
->size() = 1
and
newInstance.processActivities
->select(activityDefinition.isStart=true)
->asSequence()->first().state = #active

--The non-start activities are created in a pending state
post nonStartPending:
let newInstance : ProcessInstance =
(ProcessInstance.allInstances() -
ProcessInstance.allInstances().@pre)
->asSequence()->first() in let
procDef: ProcessDefinition =
ProcessDefinition.allInstances()->select(name=processName)
->asSequence()->first()
in
(procDef.activities->select(isStart=false) -
newInstance.processActivities.activityDefinition
->select(isStart=false)->asSet())
 isEmpty()
and
newInstance.processActivities
->select(activityDefinition.isStart=false)
->forall(state=#pending)

--New instance is created
post resultOk:
let newInstance : ProcessInstance =
(ProcessInstance.allInstances() -
ProcessInstance.allInstances().@pre)
->asSequence()->first() in
result = newInstance

-- PropertyDefinition
context PropertyDefinition

--Property definitions have unique names
inv uniqueName:
    PropertyDefinition.allInstances()->isUnique(name)

--PropertyDefinition retrieveList returns a list of all property definitions
context PropertyDefinition::retrieveList() : Set(PropertyDefinition)
post getAllPropertyDefs:
    result=PropertyDefinition.allInstances()

context PropertyDefinition::updateDescription(name : String,
newDescription : String)

--A property of the given name must exist
pre mustExist:
    PropertyDefinition.allInstances().select(name=name)
->size() = 1

--Property description changed to new description
post descriptionUpdated:
    PropertyDefinition.allInstances().select(name=name)
->asSequence()->first().description=newDescription
context ActivityClaim
   inv activityMustExist:
      ActivityInstance.allInstances().includesAll(
         ActivityClaim.allInstances().collect(claimed))

   inv userMustExist:
      User.allInstances().includesAll(
         ActivityClaim.allInstances().collect(claimant))

context ActivityInstance
   inv uniqueId:
      ActivityInstance.allInstances().isUnique(activityId)

   inv swimlanesExist:
      ActivityInstance.allInstances().forAll(ai|
         Swimlane.allInstances().select(name=ai.swimlaneName).size() = 1)

context ActivityInstance::listActivitiesAtASwimlane( swimlaneName : String) : Set(ActivityInstance)
   pre mustExist:
      Swimlane.allInstances().select(name=swimlaneName).size() = 1
   post resultOk:
      result = ActivityInstance.allInstances().select(swimlaneName=swimlaneName)

context ActivityInstance::claimActivity( aUserId : String, claimedActivityId : Integer ) : ActivityInstance
   pre mustExist:
      ActivityInstance.allInstances().select(activityId=claimedActivityId).size() = 1
   pre notClaimed:
      let theActivity : ActivityInstance = ActivityInstance.allInstances().asSequence().first()
      in ActivityClaim.allInstances().select(claimed=theActivity).size() = 0
   pre active:
ActivityInstance.allInstances()
->select(activityId=claimedActivityId)
->asSequence()->first().state = #active

--User must have appropriate role
pre userHasRole:
  self.activityDefinition.activityRoles->intersection(
    User.allInstances()->select(userid=aUserId)
    ->asSequence()->first().userRoles)
->notEmpty()

--Activity instance now has a claim
post activityClaimed:
  let claim : ActivityClaim =
    ActivityClaim.allInstances()->select(claimed=self)
    ->asSequence()->first()
  in
  claim.claimant =
    User.allInstances()
    ->select(userid=aUserId)
    ->asSequence()->first()
  and
  ActivityClaim.allInstances()
  ->select(claimed=self)
  ->size() = 1

context ActivityInstance::releaseClaim(aUserId : String, activityId : Integer )
--Instance must be claimed by the user
pre claimedByUser:
  let claim : ActivityClaim =
    ActivityClaim.allInstances()->select(claimed=self)
    ->asSequence()->first()
  in
  claim.claimant =
    User.allInstances()
    ->select(userid=aUserId)
    ->asSequence()->first()

--Instance is no longer claimed
post noLongerClaimed:
  ActivityClaim.allInstances()->select(claimed=self)->isEmpty()

context ActivityInstance::executeActivity( aUserId : String )
--User must have claim to activity
pre claimedByUser:
  let claim : ActivityClaim =
    ActivityClaim.allInstances()->select(claimed=self)
    ->asSequence()->first()
  in
  claim.claimant =
    User.allInstances()
    ->select(userid=aUserId)
    ->asSequence()->first()

--All properties associated with the activity must be included
pre allPropertiesIncluded:
  self.activityDefinition.properties->collect(name)->asSet() =
  self.propertyValues->collect(name)->asSet()

--Claim has been released.
post claimRelease:
  ActivityClaim.allInstances()->select(claimed=self)
  ->isEmpty()
--Activity state is complete
post activityComplete:
  self.state = #complete

--Directly connected activities are now active
post directConnectActive:
  let dirConnected : Set(Connectable) =
  self.processInstance.processDefinition.connectors
  ->select(c| c.source=self.activityDefinition
  and
  c.sink.oclIsKindOf(ActivityDefinition)
  and
  c.connExpressionIsTrue=true)
  .sink->asSet()
  in
  self.processInstance.processActivities
  ->select(a| a<>self and
  dirConnected->includes(a.activityDefinition))
  ->forAll(state=#active)

--If the activity was connected to a gateway, activities on the other
--side of the gateway are now active if the connection expression
--is true. There may be an explicit connection conditional expression
--(e.g. for exclusive gateways) or an implicit condition (which is
--always true for parallel gateways).
post splitterGateways:
  --Get splitter gateways
  let splitterGatewayOutConns: Set(ConnectorDefinition)=
  self.processInstance.processDefinition.connectors
  ->select(c| self.processInstance.processDefinition.connectors
  ->select(source=self.activityDefinition
  and
  sink.oclIsKindOf(GatewayDefinition))
  ->collect(sink)->asSet()->includes(c.source))
  in
  --Get splitter gateway sink activity defs
  let sinkActivityDefs: Set(Connectable) =
  splitterGatewayOutConns->select(
  sink.oclIsKindOf(ActivityDefinition)
  and
  connExpressionIsTrue=true)
  ->collect(sink)->asSet()->includes(c.source)
  in
  self.processInstance.processActivities
  ->select(a| a<>self and
  sinkActivityDefs
  ->includes(a.activityDefinition))
  ->forAll(state=#active)

--If the gateway is a parallel join, and the connection expressions
--for all connectors are true (which means the connected activity
--was completed, then activity on the other side of the gateway is
--active.
post allJoin:
  let toJoinGatewayConns: Set(ConnectorDefinition)=
  self.processInstance.processDefinition.connectors
  ->select(c| c.source=self.activityDefinition
  and
c.sink.oclIsKindOf(GatewayDefinition)
and c.sink.oclAsType(GatewayDefinition).
type=#parallel

and
((self.processInstance.processDefinition
  .connectors
  ->select(sink=c.sink).source->asSet()
  ->size() > 1))

in
gatewayDefs: Set(Connectable)=
toJoinGatewayConns->collect(sink)->asSet()
in
gatewayDefs->forall(gw|
  self.processInstance.processDefinition.connectors->
  select(sink=gw and connExpressionIsTrue=true)
  ->size() > 0 and
  (self.processInstance.processDefinition.connectors->
   select(sink=gw and connExpressionIsTrue=true)
   =
   self.processInstance.processDefinition
   .connectors->
   select(sink=gw))

implies

--the process activity associated with the
--activity definition gw connects to
--has an active state

self.processInstance.processActivities
  ->select(pa|
    (self.processInstance.processDefinition
     .connectors->select(source=gw))
    ->collect(sink)->asSet()
    ->includes(pa.activityDefinition))
  ->forall(state=#active))

--If any of the connection expressions for inputs to an
--exclusive gateway are true, then the activity connected to the
--output of the gateway will now be active.

post oneJoin:
  let toJoinGatewayConns: Set(ConnectorDefinition)=
    self.processInstance.processDefinition.connectors
    ->select(c|c.source=self.activityDefinition
      and
      c.sink.oclIsKindOf(GatewayDefinition)
      and c.sink.oclAsType(GatewayDefinition).
type=#exclusive

and
((self.processInstance.processDefinition
  .connectors
  ->select(sink=c.sink).source->asSet()
  ->size() > 1))

in
gatewayDefs: Set(Connectable)=
toJoinGatewayConns->collect(sink)->asSet()
in
gatewayDefs->forall(gw|
  (self.processInstance.processDefinition
   .connectors->
   select(sink=gw))->size()
(self.processInstance.processDefinition
.connectors->
select(sink=gw and connExpressionIsTrue=true))
->size() > 0)
implies
--the process activity associated with the
--activity definition gw connects to
--has an active state
self.processInstance.processActivities
->select(pa|
(self.processInstance.processDefinition
.connectors->select(source=gw))
->collect(sink)->asSet()
->includes(pa.activityDefinition))
->forAll(state=#active))

--For exclusive gateways, only one of the connection expressions may
--evaluate to true
post oneEnabledExclusiveConn:
self.processInstance.processDefinition.connectors
->select(c|c.source.oclIsKindOf(GatewayDefinition)
and c.source.oclAsType(GatewayDefinition).type=#exclusive
and c.sink.oclIsKindOf(ActivityDefinition)
and
self.processInstance.processActivities->select(state=#active)
->collect(activityDefinition)->asSet()
->includes(c.sink.oclAsType(ActivityDefinition))
and
(self.processInstance.processDefinition.connectors
->select(source=c.source)->size()==1)->size() > 0
implies
self.processInstance.processDefinition.connectors
->select(self.processInstance.processDefinition.connectors
->select(c|c.source.oclIsKindOf(GatewayDefinition)
and c.source.oclAsType(GatewayDefinition).type=#exclusive
and c.sink.oclIsKindOf(ActivityDefinition)
and
self.processInstance.processActivities
->select(state=#active)
->collect(activityDefinition)->asSet()
->includes(c.sink.oclAsType(ActivityDefinition))
->collect(source)->asSet()
->includes(sink))->collect(connExpressionIsTrue)
->count(true)=1

post allParallelEnabled:
self.processInstance.processDefinition.connectors
->select(c|c.source.oclIsKindOf(GatewayDefinition)
and c.source.oclAsType(GatewayDefinition).type=#parallel
and c.sink.oclIsKindOf(ActivityDefinition)
and
processInstance.processActivities->select(state=#active)
->collect(activityDefinition)->asSet()
->includes(c.sink.oclAsType(ActivityDefinition))
and
(self.processInstance.processDefinition.connectors
 ->select(source=c.source)->size()=1))
->collect(source)->asSet()->size() > 0
implies
self.processInstance.processDefinition.connectors->
  select(
  self.processInstance.processDefinition.connectors
  ->select(c|c.source.oclIsKindOf(GatewayDefinition)
  and c.source.oclAsType(GatewayDefinition).type=#parallel
  and c.sink.oclIsKindOf(ActivityDefinition)
  and
  self.processInstance.processActivities->select(state=#active)
  ->collect(activityDefinition)->asSet()
  ->includes(c.sink.oclAsType(ActivityDefinition))
  and
  (self.processInstance.processDefinition.connectors
   ->select(source=c.source)->size()=1))
  ->collect(source)->asSet()
  ->includes(sink))=}collect(connExpressionIsTrue)
  ->count(false) = 0

Section 2 – Test Based on Parallel Gateway

!create p1, p2, p3, p4 : PropertyDefinition

!set p1.name := 'p1'
!set p1.type := #integer
!set p1.description := 'property 1'

!set p2.name := 'p2'
!set p2.type := #string
!set p2.description := 'property 2'

!set p3.name := 'p3'
!set p3.type := #string
!set p3.description := 'property 3'

!set p4.name :'p4'
!set p4.type := #string
!set p4.description := 'property 4'

!create a1, a2, a3, a4 : ActivityDefinition

!set a1.name := 'A'
!set a1.description := 'activity A'
!set a1.isStart := true
!set a1.isEnd := false

!set a2.name := 'B'
!set a2.isStart := false
!set a2.isEnd := false

!set a3.name := 'C'
!set a3.isStart := false
!set a3.isEnd := false
!set a4.name := 'D'
!set a4.isStart := false
!set a4.isEnd := true

!create processDef : ProcessDefinition
!set processDef.name := 'p1'

!insert (processDef, a1) into ProcessActivities
!insert (processDef, a2) into ProcessActivities
!insert (processDef, a3) into ProcessActivities
!insert (processDef, a4) into ProcessActivities

!insert (a1, p1) into ActivityProperties
!insert (a2, p2) into ActivityProperties
!insert (a3, p3) into ActivityProperties
!insert (a4, p4) into ActivityProperties

!create s1, s2, s3 : Swimlane

!set s1.name := 'Lane 1'
!set s2.name := 'Lane 2'
!set s3.name := 'Lane 3'

!insert (s1, a1) into SwimlaneActivities
!insert (s1, a4) into SwimlaneActivities
!insert (s2, a2) into SwimlaneActivities
!insert (s3, a3) into SwimlaneActivities

!create split, join : GatewayDefinition
!set split.type := #parallel
!set join.type := #parallel

!insert (processDef, split) into ProcessGateways
!insert (processDef, join) into ProcessGateways

!create a1ToSplit, splitToA2, splitToA3 : ConnectorDefinition
!create a2ToJoin, a3ToJoin, joinToA4 : ConnectorDefinition

!insert (processDef, a1ToSplit) into ProcessConnections
!insert (processDef, splitToA2) into ProcessConnections
!insert (processDef, splitToA3) into ProcessConnections
!insert (processDef, a2ToJoin) into ProcessConnections
!insert (processDef, a3ToJoin) into ProcessConnections
!insert (processDef, joinToA4) into ProcessConnections

!set a1ToSplit.source := a1
!set a1ToSplit.sink := split

!set splitToA2.source := split
!set splitToA2.sink := a2
!set splitToA2.connExpressionIsTrue := true

!set splitToA3.source := split
!set splitToA3.sink := a3
!set splitToA3.connExpressionIsTrue := true

!set a2ToJoin.source := a2
!set a2ToJoin.sink := join
!set a2ToJoin.connExpressionIsTrue := false

!set a3ToJoin.source := a3
!set a3ToJoin.sink := join
!set a3ToJoin.connExpressionIsTrue := false
!set joinToA4.source := join
!set joinToA4.sink := a4
!set joinToA4.connExpressionIsTrue := false

!create alice, bob, carol, doug : User
!set alice.userid := 'u1'
!set bob.userid := 'u2'
!set carol.userid := 'u3'
!set doug.userid := 'u4'

!create scanner, worker, approver : Role
!insert (alice, scanner) into UserRoles
!insert (bob, worker) into UserRoles
!insert (carol, worker) into UserRoles
!insert (doug, approver) into UserRoles

!insert (a1, scanner) into ActivityRoles
!insert (a2, worker) into ActivityRoles
!insert (a3, worker) into ActivityRoles
!insert (a4, approver) into ActivityRoles

-- Instantiate process instance
!create factory : ProcessFactory

!openter factory instantiateProcess('p1')
!create processInstance : ProcessInstance
!set processInstance.instanceId := 1
!insert (processInstance, processDef) into ProcessInstanceDefinition
!create ai1, ai2, ai3, ai4 : ActivityInstance
!insert (processInstance, ai1) into ProcessInstanceActivities
!insert (processInstance, ai2) into ProcessInstanceActivities
!insert (processInstance, ai3) into ProcessInstanceActivities
!insert (processInstance, ai4) into ProcessInstanceActivities

!set ai1.activityId := 1
!set ai1.state := #active
!set ai1.swimlaneName := 'Lane 1'

!set ai2.activityId := 2
!set ai2.state := #pending
!set ai2.swimlaneName := 'Lane 2'

!set ai3.activityId := 3
!set ai3.state := #pending
!set ai3.swimlaneName := 'Lane 2'

!set ai4.activityId := 4
!set ai4.state := #pending
!set ai4.swimlaneName := 'Lane 3'

!insert (ai1, a1) into ActivityInstanceDefinition
!insert (ai2, a2) into ActivityInstanceDefinition
!insert (ai3, a3) into ActivityInstanceDefinition
!insert (ai4, a4) into ActivityInstanceDefinition

!opexit processInstance

---------- Claim activity 1 ----------
!openter ai1 claimActivity('u1', 1)
!create claim : ActivityClaim
!set claim.claimant := alice
!set claim.claimed := ai1
!opexit ai1

-- Set the property associated with the activity
!create pv1 : PropertyValue
!set pv1.name := 'p1'
!insert (ai1, pv1) into ActivityInstancesProperties

------------- Execute activity 1 -------------
!openter ai1 executeActivity('u1')
!destroy claim
!set ai1.state := #complete
!set ai2.state := #active
!set ai3.state := #active
!opexit

----- Claim activity 2 -----
!openter ai2 claimActivity('u2', 2)
!create claim2 : ActivityClaim
!set claim2.claimant := bob
!set claim2.claimed := ai2
!opexit ai2

-- Set the property associated with the activity
!create pv2 : PropertyValue
!set pv2.name := 'p2'
!insert (ai2, pv2) into ActivityInstancesProperties

------------- Execute activity 2 -------------
!openter ai2 executeActivity('u2')
!destroy claim2
!set ai2.state := #complete
!set a2ToJoin.connExpressionIsTrue := true
!opexit

----- Claim activity 3 -----
!openter ai3 claimActivity('u3', 3)
!create claim3 : ActivityClaim
!set claim3.claimant := carol
!set claim3.claimed := ai3
!opexit ai3

-- Set the property associated with the activity
!create pv3 : PropertyValue
!set pv3.name := 'p3'
!insert (ai3, pv3) into ActivityInstancesProperties

------------- Execute activity 3 -------------
!openter ai3 executeActivity('u3')
!destroy claim3
!set ai3.state := #complete
!set a3ToJoin.connExpressionIsTrue := true
!set ai4.state := #active
!opexit

----- Claim activity 4 -----
!openter ai4 claimActivity('u4', 4)
!create claim4 : ActivityClaim
!set claim4.claimant := doug
!set claim4.claimed := ai4
!opexit ai4
-- Set the property associated with the activity
!create pv4 : PropertyValue
!set pv4.name := 'p4'
!insert (ai4, pv4) into ActivityInstancesProperties

----------- Execute activity 4 ---------------
!openter ai4 executeActivity('u4')
!destroy claim4
!set ai4.state := #complete
!opexit

Section 3 – Test Based on Exclusive Gateway

!create p1, p2, p3, p4 : PropertyDefinition

!set p1.name := 'p1'
!set p1.type := #integer
!set p1.description := 'property 1'

!set p2.name := 'p2'
!set p2.type := #string
!set p2.description := 'property 2'

!set p3.name := 'p3'
!set p3.type := #string
!set p3.description := 'property 3'

!set p4.name :='p4'
!set p4.type := #string
!set p4.description := 'property 4'

!create a1, a2, a3, a4 : ActivityDefinition

!set a1.name := 'A'
!set a1.description := 'activity A'
!set a1.isStart := true
!set a1.isEnd := false

!set a2.name := 'B'
!set a2.isStart := false
!set a2.isEnd := false

!set a3.name := 'C'
!set a3.isStart := false
!set a3.isEnd := false

!set a4.name := 'D'
!set a4.isStart := false
!set a4.isEnd := true

!create processDef : ProcessDefinition
!set processDef.name := 'p1'

!insert (processDef, a1) into ProcessActivities
!insert (processDef, a2) into ProcessActivities
!insert (processDef, a3) into ProcessActivities
!insert (processDef, a4) into ProcessActivities
!create s1, s2, s3 : Swimlane
!set s1.name := 'Lane 1'
!set s2.name := 'Lane 2'
!set s3.name := 'Lane 3'

!insert (s1, a1) into SwimlaneActivities
!insert (s1, a4) into SwimlaneActivities
!insert (s2, a2) into SwimlaneActivities
!insert (s3, a3) into SwimlaneActivities

!create split, join : GatewayDefinition
!set split.type := #exclusive
!set join.type := #exclusive

!insert (processDef, split) into ProcessGateways
!insert (processDef, join) into ProcessGateways

!create a1ToSplit, splitToA2, splitToA3 : ConnectorDefinition
!create a2ToJoin, a3ToJoin, joinToA4 : ConnectorDefinition

!insert (processDef, a1ToSplit) into ProcessConnections
!insert (processDef, splitToA2) into ProcessConnections
!insert (processDef, splitToA3) into ProcessConnections
!insert (processDef, a2ToJoin) into ProcessConnections
!insert (processDef, a3ToJoin) into ProcessConnections
!insert (processDef, joinToA4) into ProcessConnections

!set a1ToSplit.source := a1
!set a1ToSplit.sink := split
!set splitToA2.source := split
!set splitToA2.sink := a2
!set splitToA2.connExpressionIsTrue := false

!set splitToA3.source := split
!set splitToA3.sink := a3
!set splitToA3.connExpressionIsTrue := false

!set a2ToJoin.source := a2
!set a2ToJoin.sink := join
!set a2ToJoin.connExpressionIsTrue := false

!set a3ToJoin.source := a3
!set a3ToJoin.sink := join
!set a3ToJoin.connExpressionIsTrue := false

!set joinToA4.source := join
!set joinToA4.sink := a4
!set joinToA4.connExpressionIsTrue := false

!create alice, bob, carol, doug : User

!set alice.userid := 'u1'
!set bob.userid := 'u2'
!set carol.userid := 'u3'
!set doug.userid := 'u4'
create scanner, worker, approver : Role
insert (alice, scanner) into UserRoles
insert (bob, worker) into UserRoles
insert (carol, worker) into UserRoles
insert (doug, approver) into UserRoles
insert (a1, scanner) into ActivityRoles
insert (a2, worker) into ActivityRoles
insert (a3, worker) into ActivityRoles
insert (a4, approver) into ActivityRoles

-- Instantiate process instance
create factory : ProcessFactory

openter factory instantiateProcess('p1')
create processInstance : ProcessInstance
set processInstance.instanceId := 1
insert (processInstance, processDef) into ProcessInstanceDefinition
create ai1, ai2, ai3, ai4 : ActivityInstance
insert (processInstance, ai1) into ProcessInstanceActivities
insert (processInstance, ai2) into ProcessInstanceActivities
insert (processInstance, ai3) into ProcessInstanceActivities
insert (processInstance, ai4) into ProcessInstanceActivities

set ai1.activityId := 1
set ai1.state := #active
set ai1.swimlaneName := 'Lane 1'

set ai2.activityId := 2
set ai2.state := #pending
set ai2.swimlaneName := 'Lane 2'

set ai3.activityId := 3
set ai3.state := #pending
set ai3.swimlaneName := 'Lane 2'

set ai4.activityId := 4
set ai4.state := #pending
set ai4.swimlaneName := 'Lane 3'

insert (ai1, a1) into ActivityInstanceDefinition
insert (ai2, a2) into ActivityInstanceDefinition
insert (ai3, a3) into ActivityInstanceDefinition
insert (ai4, a4) into ActivityInstanceDefinition

opexit processInstance

---------- Claim activity 1 ----------

openter ai1 claimActivity('u1', 1)
create claim : ActivityClaim
set claim.claimant := alice
set claim.claimed := ai1
opexit ai1

-- Set the property associated with the activity
create pv1 : PropertyValue
set pv1.name := 'p1'
insert (ai1, pv1) into ActivityInstancesProperties

---------- Execute activity 1 ----------
!openter a1 executeActivity('u1')
!destroy claim
!set a1.state := #complete
!set splitToA2.connExpressionIsTrue := true
!set a2.state := #active
!opexit

----- Claim activity 2 ---------
!openter a2 claimActivity('u2', 2)
!create claim2 : ActivityClaim
!set claim2.claimant := bob
!set claim2.claimed := a12
!opexit a2

-- Set the property associated with the activity
!create pv2 : PropertyValue
!set pv2.name := 'p2'
!insert (ai2, pv2) into ActivityInstancesProperties

---------- Execute activity 2 ----------
!openter a2 executeActivity('u2')
!destroy claim2
!set a2.state := #complete
!set a2ToJoin.connExpressionIsTrue := true
!set a4.state := #active
!opexit

----- Claim activity 4 ---------
!openter a4 claimActivity('u4', 4)
!create claim4 : ActivityClaim
!set claim4.claimant := doug
!set claim4.claimed := a14
!opexit a4

-- Set the property associated with the activity
!create pv4 : PropertyValue
!set pv4.name := 'p4'
!insert (ai4, pv4) into ActivityInstancesProperties

---------- Execute activity 4 ----------
!openter a4 executeActivity('u4')
!destroy claim4
!set a4.state := #complete
!opexit