Estándar BPMN 2.0

Proyecto de grado - Anexo

Emiliano Alonzo - Rodrigo Damiano

Supervisor responsable

Andrea Delgado - Daniel Calegari

Instituto de Computación

Facultad de Ingeniería - Universidad de la República

Montevideo - Uruguay

1 Representación del estándar BPMN 2.0

El objetivo de este documento es mostrar las diferentes representaciones que ofrece el estándar BPMN 2.0. Para esto iremos presentando los diferentes componentes que consideramos que son importantes para entender el proyecto de grado, mostrando para cada uno su representación gráfica y su representación en formato XML.

La ruta de trabajo para este análisis es la siguiente:

- Un proceso vacío
- Proceso funcional con una tarea de usuario.
- Complemento sobre tareas de usuario
- Otras actividades: Servicios, Manuales y SubProcesos.
- Eventos: Tipos y definiciones.
- Compuertas

Nuestro principal objetivo es definir un proceso, principal objetivo del modelado BPMN 2.0. Para que este sea ejecutable hay que asociarlo a un pool y ambos, tanto el proceso como un pool deben ir dentro de una etiqueta de definición como se ve en la figura 1 el resultado es el siguiente:

```
<definitions>
 <collaboration id="Collaboration_0znjdps">
  <participant id="idPool" name="NombrePool" processRef="idProceso" />
 </collaboration>
 <process id="idProceso" isExecutable="true">
  <!-- Definición del proceso -->
 </process>
<definitions>
```

Figura 1. Representación en formato XML de un proceso vacío.

Con esta estructura básica es posible modelar un proceso con un pool (piscina del inglés), capaz de contener la definición estructural de un proceso. Se pueden apreciar en negrita los elementos destacados de esta definición que son, el nombre del pool y la referencia en el pool a la definición del proceso mediante su identificador.

El pool cumple el rol de contenedor de proceso y este puede, estar asociado a uno definido detalladamente como los que planteamos a continuación, o pueden estar asociados a procesos externos que desconocemos su estructura pero sabemos su comportamiento, por ejemplo, cómo responden frente a determinados mensajes. Y si bien es la primera forma de asignación de procesos, y por ende, de todos los elementos que lo contienen, hay otra forma de asignar tareas dentro de un proceso de manera más específica, como lo son los lanes. Un lane (carril del inglés) es un componente que se utiliza para asignar roles internos dentro de un pool y se utiliza para realizar asignación de roles a los componentes de flujo dentro de un proceso. En la figura 2 se

puede ver la definición de diferentes lanes asociados a un procesos.

```
<definitions>
  <!-- Definición de colaboraciones -->
  <process id="idProceso" isExecutable="true">
    <laneSet>
      <lane id="Lane_0zulgpl" name="Lane 2" />
      <lane id="Lane_0xi4xgm" name="Lane 1" />
    </laneSet>
  </process>
<definitions>
```

Figura 2. Definición de lanes en un proceso.

Una vez definidos los contenedores, pools y lanes, comenzamos a analizar los elementos de flujo de un proceso, que pueden ser actividades, eventos y compuertas.

Observamos el modelo de un proceso sencillo en la Figura 3, y pasaremos a analizar los elementos básicos que lo componen.

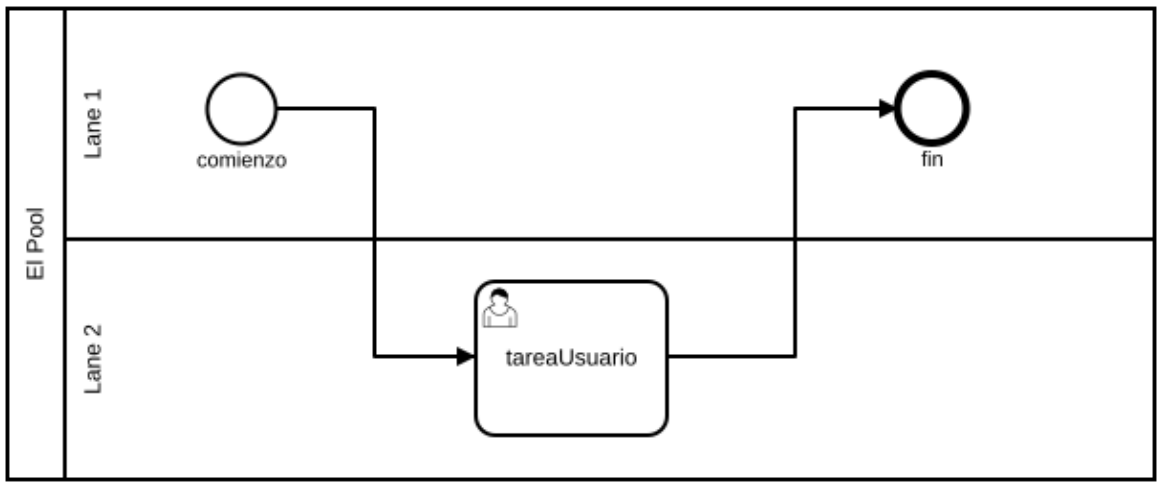

Figura 3. Ejemplo de un proceso sencillo

Dentro de los elementos de flujo del diagrama de la figura 3 encontramos un evento de inicio y un evento de fin, los cuales se representan el comienzo y la finalización de un proceso. Para ambos es posible asociar un nombre que será desplegado a la hora de renderizar el proceso en un modelador.

También encontramos una tarea de usuario o User task, este componente es la tarea más común dentro de los procesos de negocio y son realizadas por un usuario con la asistencia de una aplicación de software, como un motor de ejecución. A este tipo de tareas es posible asignarles objetos de datos, con el fin que éstas despliegan campos de un formulario o muestren información de solo lectura pero nos ocuparemos de estos componentes de datos más adelante.

en la figura 4 se puede ver la representación de este proceso en formato XML.

```
<definitions>
 <!-- Definición de colaboraciones -->
 <process id="idProceso" isExecutable="true">
     <laneSet>
    <lane id="id_lane_2" name="Lane 2">
     <flowNodeRef>Task_0km2n4u</flowNodeRef>
    </lane>
    <lane id="id_lane_1" name="Lane 1">
     <flowNodeRef>StartEvent_1dyyswn</flowNodeRef>
     <flowNodeRef>EndEvent_0dtn6r7</flowNodeRef>
    </lane>
  </laneSet>
  <startEvent id="StartEvent_1dyyswn" name="comienzo"></startEvent>
  <endEvent id="EndEvent_0dtn6r7" name="fin"></endEvent>
  <userTask id="Task_0km2n4u" name="tareaUsuario"></userTask>
  <sequenceFlow id="SequenceFlow_1hh7r47"
     sourceRef="StartEvent_1dyyswn" targetRef="Task_0km2n4u" />
  <sequenceFlow id="SequenceFlow_1m4hwx6"
     sourceRef="Task_0km2n4u" targetRef="EndEvent_0dtn6r7" />
 </process>
<definitions>
```

Figura 4. Representación en formato XML del proceso sencillo de la figura 3.

Se puede apreciar que los elementos de flujo son asociados a un lane utilizando su identificador

Finalmente resta analizar los componentes de conexión, en este caso el sequenceFlow (o flujo de secuencia) que es el componente utilizado para conectar dos elementos de flujo. Para ello se debe indicar el id de origen y el id de destino. En la figura 4, se pueden ver estos elementos, que conectan el evento de inicio con la tarea de usuario, y la tarea de usuario con el evento de fin.

Nuestro siguiente paso en el proceso de análisis es especificar las restantes actividades que incluimos dentro del dominio del problema a resolver. Para esto observamos el modelo en la Figura 5 en el cual se pueden ver una tarea manual, una tarea de servicio y un subproceso.

Figura 5. Ejemplo de modelo con diferentes actividades

Se puede ver la diferencia gráfica entre las diferentes actividades, donde las tareas de usuario tienen el dibujo de una persona, la tarea manual el dibujo de una mano, para las tareas de servicio

un par de engranajes y para los subprocesos un cuadrado con el símbolo “+”. De forma similar, en la representación en formato XML, cad auna de estas actividades tiene un tag diferente como se puede ver en la figura 6.

```
<definitions>
  <!-- Definición de colaboraciones -->
  <process id="idProceso" isExecutable="true">
    <laneSet>
      <!-- Asignación de lanes -->
    </laneSet>
    <manualTask id="Task_0" name="tarea manual"></manualTask>
    <serviceTask id="Task_1" name="tarea de servicio"></serviceTask>
    <subProcess id="Task_2" name="sub proceso">
      <!-- definición del subproceso -->
    </subProcess>
    <!-- Definición de secuence flow -->
  </process>
<definitions>
```

Figura 6. Representación en formato XML de la figura 5.

Las tareas manuales modelan aquellas tareas que se resuelven sin la interacción con un motor de ejecución o una aplicación. Las tareas de servicio corresponden aquellas tareas que se resuelven mediante la ejecución de un servicio o aplicación externa, pudiendo ser por ejemplo la invocación a un webservice. Las actividades de tipo subproceso modelan la ejecución completa de un proceso embebido.

Como se puede ver, la diferencia esencial entre cada tipo de tarea es el tipo de etiqueta que utiliza. Todos comparten un atributo común, name, que es utilizado para desplegar un nombre, que cumple un rol documentador a la hora de desplegar el diagrama gráficamente.

Los *subprocess* requieren tener dentro la definición del subproceso, esto implica, que dentro de la etiqueta subproceso, se debe colocar dentro todas las etiquetas que componen la estructura del proceso, como si se tratara de la etiqueta *process*, a excepción de la etiqueta correspondiente a lane.

Dado que parte de nuestro problema ser capaces de definir capturar y desplegar información acerca de los proceso, para definir por ejemplo formularios, mostraremos cómo asociar componentes de datos a las tareas, en particular, a tareas de usuario. En la figura 7 se puede ver esta definición, la cual no es visible en la representación gráfica del estándar.

```
<process id='id_tareas_variadas' isExecutable='true'>
  <property id='descripcion' itemSubjectRef='xsd:string' name='descripcion'/>
  <!-- Definición de lanes y elementos asociados -->
  <userTask id='_4' name='envia el expediente'>
    <ioSpecification>
      <dataOutput id='data_4_descripcion' itemSubjectRef='xsd:string'
        name='data_4_descripcion'/>
```

```
    <inputSet></inputSet>
    <outputSet>
     <dataOutputRefs>data_4_descripcion</dataOutputRefs>
    </outputSet>
   </ioSpecification>
   <dataOutputAssociation>
    <sourceRef>data_4_descripcion</sourceRef>
    <targetRef>descripcion</targetRef>
   </dataOutputAssociation>
  </userTask>
  <userTask id='_5' name='confirma expediente'>
   <ioSpecification>
    <dataInput id='data_5_descripcion' itemSubjectRef='xsd:string'
     name='data_5_descripcion'/>
    <inputSet>
     <dataInputRefs>data_5_descripcion</dataInputRefs>
    </inputSet>
    <outputSet></outputSet>
   </ioSpecification>
   <dataInputAssociation>
    <sourceRef>descripcion</sourceRef>
    <targetRef>data_5_descripcion</targetRef>
   </dataInputAssociation>
  </userTask>
  <!--  Otros elementos de flujo  -->
  <!-- Flujo de secuencias -->
 </process>
```

Figura 7. Asociación de datos a una tarea de usuario

La asociación de campos a tareas no es trivial y se puede realizar mediante la asociación de datos a una tarea de usuario. Para ello se necesitan 3 componentes: una propiedad asociada al proceso, que va a ser la que toma el rol de contener la información de modo que pueda ser vista por todos los elementos del proceso; una ioSpecification que indica cómo se va a realizar la entrada y salida de datos de una tarea; y un dataInputAssociation/dataOutputAssociation que va a ser el componente que asocia los últimos dos componentes definidos indicando el origen y el destino de la asociación.

Como siguiente actividad analizamos los distintos tipos de eventos, que se utilizan para modelar cosas que suceden. Existen cuatro tipos de eventos y cada uno de estos con sus opciones particulares. Además de los tipos de eventos, existen tipos de sucesos a los que reaccionar, ya sea un mensaje, un error, un período de tiempo, entre otros. En la Figura 8 se puede ver un modelo donde aparecen estos eventos que mencionamos y en la figura 9 vemos la representación en formato XML de la figura 8.

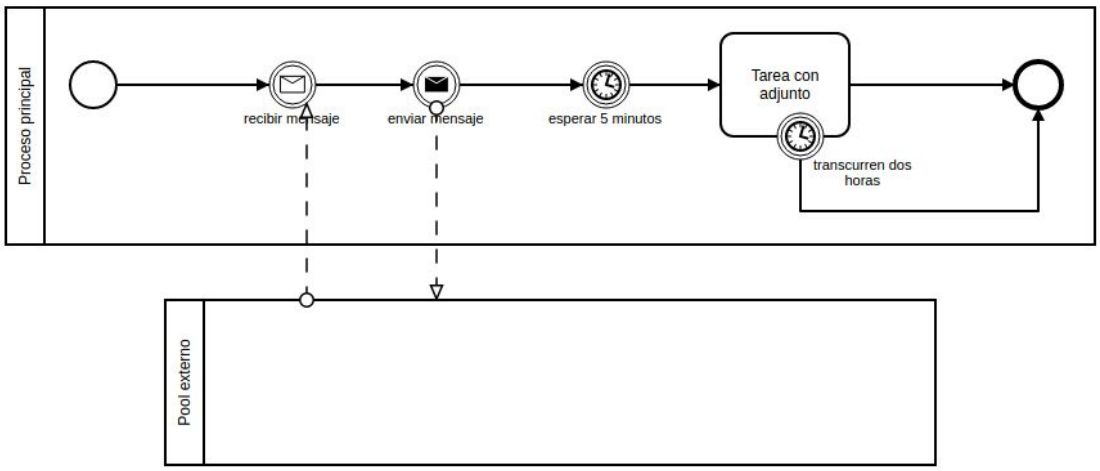

Figura 8. Ejemplo de modelo con eventos

```
<collaboration id="Collaboration_0jg3gms">
  <participant id="Part_0" name="Proceso principal" processRef="Process_1" />
  <participant id="Part_1" name="Pool externo"/>
  <messageFlow id="MsgFlow_0" sourceRef="InteThrowEv_1" targetRef="Part_1" />
  <messageFlow id="MsgFlow_1" sourceRef="Part_1" targetRef="InteThrowEv_2" />
</collaboration>
<process id="Process_1" isExecutable="false">
  <startEvent id="StartEvent_1"></startEvent>
  <intermediateCatchEvent id="InteThrowEv_2" name="recibir mensaje">
    <messageEventDefinition />
  </intermediateCatchEvent>
  <intermediateThrowEvent id="InteThrowEv_1" name="enviar mensaje">
    <messageEventDefinition />
  </intermediateThrowEvent>
  <intermediateCatchEvent id="InteThrowEv_0" name="esperar 5 minutos">
    <timerEventDefinition id="_7_ED_1">
      <timeDuration><![CDATA[PT5M]]></timeDuration>
    </timerEventDefinition>
  </intermediateCatchEvent>
  <task id="Task_1" name="Tarea con adjunto"></task>
  <endEvent id="EndEv_1"></endEvent>
  <boundaryEvent id="BoundEv_1" name="transcurren dos horas" attachedToRef="Task_1">
    <timerEventDefinition id="_7_ED_1">
      <timeDuration><![CDATA[PT2H]]></timeDuration>
    </timerEventDefinition>
  </boundaryEvent>
  <!-- definicion de secuence flow -->
</process>
```

Figura 9. Representación en formato XML de la figura 8

Como ya expresamos, existen dos tipos de eventos, StartEvent y EndEvent que se utilizan para comenzar y finalizar la ejecución de un proceso respectivamente. Además BPMN 2.0 incluye un evento intermedio y un tipo de evento adjunto.

El evento intermedio se utiliza para esperar o disparar un evento en un momento concreto del flujo de un proceso.

El evento adjunto se utiliza para esperar por un evento mientras se ejecuta una determinada actividad (tarea o subproceso) y tiene dos posibles comportamientos: una vez disparado el evento se interrumpe la tarea o se espera a que esta finalice.

Una vez analizados los 4 tipos de eventos podemos pasar a analizar la definición de cada uno de ellos. La definición está relacionado con la naturaleza del estímulo frente al cual reacciona o emite el evento. Dentro del alcance del proyecto incluimos los eventos de mensaje y eventos de timer.

Los eventos de mensaje son aquellos que esperan o emiten un mensaje. Para ello hay que asignarle un tipo de datos al mensaje y un nombre. También es esperable que un evento de mensaje tenga una contraparte que lo reciba o envíe, externo al proceso modelado. Para ello debe definirse dentro de la colaboración tanto el participante externo, sin necesidad de detallar el proceso; como el flujo de mensaje indicando las actividades o procesos involucradas .

Los eventos de timer son aquellos que esperan por un transcurso de tiempo actuando como un mecanismo de retraso. Hace falta indicar el período de tiempo por el cual se está esperando.

Los tipos de definición de eventos que quedan fuera del alcance son CancelEventDefinition, CompensationEventDefinition, LinkEventDefinition, SignalEventDefinition, ErrorEventDefinition, ConditionalEventDefinition, EscalationEventDefinition y TerminateEventDefinition.

Con los elementos analizados hasta el momento se es capaz de definir actividades básicas que conforman el estándar BPMN 2.0 de manera secuencial, es decir, ejecutándose una actividad o evento luego de otro. Una de las características de BPMN 2.0 es que es capaz de controlar el flujo mediante compuertas. En las Figuras 10.1 y 10.2 se pueden ver ejemplos de los dos tipos de compuertas que daremos soporte.

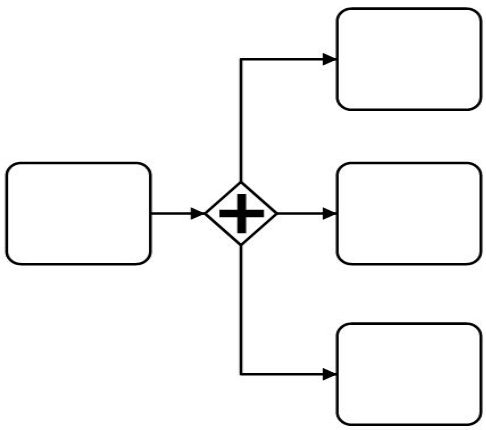

Figura 10.1. Compuerta paralelo o compuerta AND

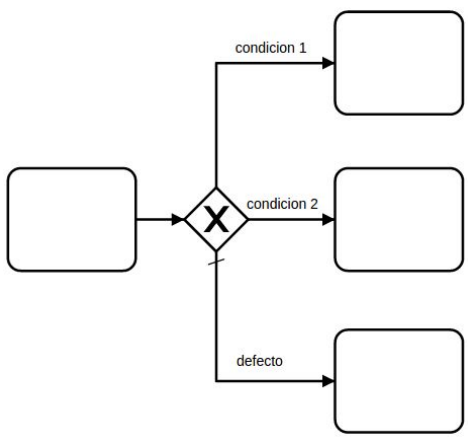

Figura 10.2. Compuerta exclusiva o compuerta XOR

En las figuras 11 y 12 se pueden ver las representaciones en formato XML de una compuerta AND y de una compuerta XOR.

```
<process id="Process_1" isExecutable="false">
  <task id="Task_4"></task>
  <task id="Task_3"></task>
  <task id="Task_2"></task>
  <task id="Task_1"></task>
  <parallelGateway id="ParallelGw_0"></parallelGateway>
  <sequenceFlow id="SeqFlow_1xxzt2z" sourceRef="ParallelGw_0" targetRef="Task_4" />
  <sequenceFlow id="SeqFlow_1fswj6z" sourceRef="ParallelGw_0" targetRef="Task_3" />
  <sequenceFlow id="SeqFlow_1m3ml7x" sourceRef="ParallelGw_0" targetRef="Task_2" />
  <sequenceFlow id="SeqFlow_1piekj4" sourceRef="Task_1" targetRef="ParallelGw_0" />
</process>
```

Figura 11. Representación en formato XML de una compuerta AND

La finalidad de una compuerta paralela es esperar por todos los flujos que tiene marcados como entrada para largar al en paralelo todos los flujos que tiene marcado como salida. En el ejemplo de la Figura 10.1, una vez le llega el flujo a la compuerta, está lo manda en paralelo a las 3 tareas posteriores, adoptando el patrón de flujo de "bifurcación paralelo". El sentido opuesto, cuando la compuerta tiene varios flujos de entrada adopta el patrón de "sincronización".

```
<process id="Process_1" isExecutable="false">
  <task id="Task_4"></task>
  <task id="Task_3"></task>
  <task id="Task_2"></task>
  <task id="Task_1"></task>
  <exclusiveGateway id="ExclGw_0" default="SeqFlow_1xxzt2z"></exclusiveGateway>
  <sequenceFlow id="SeqFlow_1xxzt2z"  name="defecto" sourceRef="ExclGw_0"
targetRef="Task_4" />
  <sequenceFlow id="SeqFlow_1fswj6z" name="condicion1" sourceRef="ExclGw_0"
targetRef="Task_3">
    <conditionExpression xsi:type='tFormalExpression'>
      <![CDATA[condicion1]]>
    </conditionExpression>
  </sequenceFlow>
  <sequenceFlow id="SeqFlow_1m3ml7x" name="condicion2" sourceRef="ExclGw_0"
targetRef="Task_2">
    <conditionExpression xsi:type='tFormalExpression'>
      <![CDATA[condicion2]]>
    </conditionExpression>
  </sequenceFlow>
  <sequenceFlow id="SeqFlow_1piekj4" sourceRef="Task_1" targetRef="ExclGw_0" />
</process>
```

Figura 12. Representación en formato XML de una compuerta XOR.

La compuerta exclusiva tiene un comportamiento similar, pero de manera exclusiva. Es decir, por cada flujo que le entra, elige una sola de las posibles salidas, tomando la primera que cumpla la condición asignada o tomando el camino por defecto. Cuando opta por una de las posibles salidas decimos que cumple con el patrón de flujo “elección exclusiva”. Cuando toma el primer flujo de los que le entran decimos que cumple el patrón “unión simple”.