

E.T.S. INGENIERÍA INFORMÁTICA  
UNIVERSIDAD DE VALLADOLID  
Mención en tecnologías de la información



**Hewlett Packard**  
Enterprise

# Workflow Designer

Trabajo fin de grado

**Autor:**

Fernando Merino Cejudo

**Tutores:**

Benjamín Sahelices Fernández (UVA)

Carlos Caño y Diego Próspero (HPE)

## Resumen

Este documento relatará el proceso seguido para implementar una solución viable a un problema al que se enfrentaba la empresa HPE a la hora de desarrollar flujos para uno de sus productos que tenían actualmente en producción *HPE Activator*.

El problema se basaba en la necesidad de implementar una nueva capa lógica donde poder exportar la información relacionada con la posición de los nodos de los flujos que se ejecutan durante un proceso en la aplicación *HPE Activator*, para así, poder limitar la información contenida en los nodos a las operaciones que realiza el propio nodo.

Durante el proyecto se estudiará el estándar cedido por HPE y el creado por OMG (BPMN), ya que este último será el que se utilice para la creación de dicha capa. Así mismo, una vez realizado el estudio pertinente se desarrollará un traductor que permita la conversión de un workflow de un estándar a otro de una manera fácil y cómoda para los desarrolladores. Para este último punto se desarrollará un plug-in para el IDE Eclipse.

## Abstract

This document will explain development process followed to implement a practical solution about a problem which HPE enterprise has when he has to develop flows for one of his products, in this case *HPE Activator*.

The problem mentioned above relies on the implementation of the one more layer which could wrap the situation logic of the nodes on a specific workflow who could be executed in a *HPE Activator* application. In this way, we could limit the information contained inside the nodes to only the information related with the operations performed by the node itself.

During the whole project, we will study the HPE standard and the BPM standards developed by OMG since the latter will give to us the notation to make the layer. Likewise, after study, we will develop a translator who translates one workflow from one standard to the other one by a nice and friendly way for the developers. To perform the this last requisite, we will develop a plug-in for the Eclipse IDE.

## **Agradecimientos**

Agradecer la posibilidad de poder haber realizado este trabajo a la empresa HPE y a mi tutor Benjamín Sahelices por organizarlo todo y darme la oportunidad de realizar este trabajo conjuntamente con la empresa HPE.

A todo el profesorado que durante el transcurso de la carrera ha conseguido transmitirme los diferentes puntos de vista que existen en el ámbito de la informática a base de conocimiento.

Y por último y posiblemente más importante, a todas esas personas que se han cruzado en mi vida durante el transcurso de la carrera y que han hecho posible de una manera u otra que haya podido realizar este proyecto.

# Índice

<b>1. Introducción</b>	<b>1</b>
<b>2. Objetivos</b>	<b>4</b>
<b>3. Planificación</b>	<b>5</b>
3.1 Organización del trabajo . . . . .	5
3.2 Roles y responsabilidades . . . . .	5
3.3 Fases del proyecto . . . . .	6
3.3.1 Transcurso del proyecto . . . . .	7
3.4 Análisis de riesgos . . . . .	9
3.5 Análisis de recursos . . . . .	13
3.6 Análisis de costes . . . . .	14
3.6.1 Real . . . . .	14
3.6.2 Hipotético . . . . .	15
<b>4. Análisis</b>	<b>17</b>
4.1 Análisis de requisitos . . . . .	17
4.1.1 Requisitos de usuario . . . . .	17
4.1.2 Requisitos funcionales . . . . .	17
4.1.3 Requisitos no funcionales . . . . .	17
4.2 Casos de uso . . . . .	18
4.2.1 CU-01 Traducir flujo HPE->BPMN . . . . .	19
4.2.2 CU-02 Traducir flujo BPMN->HPE . . . . .	20
4.3 Matriz de trazabilidad . . . . .	21
4.4 Modelo de dominio . . . . .	21
4.5 Diagrama de secuencia . . . . .	22
4.6 Entidades del sistema <i>HPE Service Activator</i> . . . . .	23
4.7 Business Process Model and Notation (BPMN) . . . . .	26
4.8 Tabla de correspondencias . . . . .	29
<b>5. Diseño</b>	<b>30</b>
5.1 Process-Node . . . . .	30
5.1.1 Acciones (Como extender la funcionalidad de BPMN) . . . . .	34
5.2 Rule/Switch Node . . . . .	37
5.3 Case-Packet e Initial-Case-Packet . . . . .	39
5.3.1 Case-Packet . . . . .	39
5.3.2 Initial-Case-Packet . . . . .	42
5.4 Handlers . . . . .	43
5.5 Workflow . . . . .	46

5.6	Diagramas de despliegue . . . . .	49
5.7	Diagrama de componentes . . . . .	50
5.8	Diagrama de clases . . . . .	51
<b>6.</b>	<b>Implementación</b>	<b>53</b>
6.1	Librerías utilizadas y necesarias . . . . .	54
6.2	Software utilizado . . . . .	55
<b>7.</b>	<b>Pruebas</b>	<b>56</b>
7.1	Pruebas unitarias . . . . .	56
7.2	Pruebas de integración . . . . .	59
7.3	Pruebas del sistema . . . . .	61
<b>8.</b>	<b>Control y seguimiento</b>	<b>62</b>
8.1	Objetivos conseguidos . . . . .	62
8.2	Problemas encontrados . . . . .	63
8.3	Versiones . . . . .	64
8.4	Estado actual del proyecto . . . . .	67
<b>9.</b>	<b>Conclusiones y trabajo futuro</b>	<b>68</b>
9.1	Conclusiones . . . . .	68
9.2	Trabajo futuro . . . . .	68
	<b>Bibliografía</b>	<b>69</b>
	<b>Iconografía</b>	<b>69</b>
	<b>Anexo I: Ejemplo de un workflow según BPMN</b>	<b>70</b>
	<b>Anexo II: Estructuras mencionadas de BPMN</b>	<b>71</b>
	<b>Contenido del CD</b>	<b>80</b>

## Índice de figuras

1	Workflow Manager [1, p. 6] . . . . .	1
2	Diagrama casos de uso . . . . .	18
3	Modelo de dominio . . . . .	21
4	Diagrama secuencia CU01-02 . . . . .	22
5	Diagrama de entidades que conforman un flujo de trabajo . . . . .	25
6	Visibilidad de un <i>Data Object</i> . . . . .	39
7	Diagrama de despliegue . . . . .	49
8	Diagrama de componentes . . . . .	50
9	Diagrama de clases . . . . .	51
10	Ejemplo Workflow BPMN . . . . .	70

## Índice de tablas

1	Asignación de roles y responsabilidades . . . . .	5
2	Fases del proyecto . . . . .	6
3	Planificación original . . . . .	7
4	Planificación final . . . . .	7
5	Riesgo-01 . . . . .	9
6	Riesgo-02 . . . . .	9
7	Riesgo-03 . . . . .	10
8	Riesgo-04 . . . . .	10
9	Riesgo-05 . . . . .	11
10	Riesgo-06 . . . . .	11
11	Riesgo-07 . . . . .	12
12	Riesgo-08 . . . . .	12
13	Análisis de recursos . . . . .	13
14	Análisis de costes (Fase de Inicio) . . . . .	15
15	Análisis de costes (Fase de elaboración) . . . . .	15
16	Análisis de costes (Fase de construcción) . . . . .	16
17	Análisis de costes (Fase de transición) . . . . .	16
18	Matriz de trazabilidad . . . . .	21
19	Estructuras HPE a entidades BPMN . . . . .	29
20	Process Node a BPMN . . . . .	34
21	Action a BPMN . . . . .	36
22	Rule and Switch node to BPMN . . . . .	38
23	Case-Packet a BPMN . . . . .	41
24	Initial-Case-Packet a BPMN . . . . .	42
25	Handlers a BPMN . . . . .	45

26	Atributos de un workflow a BPMN . . . . .	47
27	PU-01 Creación de un flujo HPE limpio . . . . .	56
28	PU-02 Conversión Process-Node . . . . .	57
29	PU-03 Adición de atributos a Process-Node . . . . .	57
30	PU-04 Traducción de Rule-Node con sus atributos correspondientes	57
31	PU-05 Traducción de los atributos de un workflow . . . . .	58
32	PU-06 Traducción de los Gateways tipo Switch con sus atributos correspondientes . . . . .	58
33	PI-01 Integración de un flujo HPE limpio con Process-Nodes . . .	59
34	PI-02 Traducción de un flujo BPMN con Gateways y Tasks . . . .	59
35	PI-03 Traducción de un flujo BPMN con Gateways, Tasks y atributos en un Workflow . . . . .	59
36	PI-04 Traducción de un flujo BPMN con Gateways, Tasks, Handlers y atributos en un Workflow . . . . .	60
37	PI-05 Traducción de un flujo BPMN completo y añadido el sistema de roles . . . . .	60
38	PI-06 Intento de traducción de un flujo erróneo mediante un plug-in para Eclipse . . . . .	60
39	PS-01 Traducción de un flujo BPMN completo . . . . .	61
40	PS-02 Traducción de un flujo HPE completo . . . . .	61
41	PS-03 Traducción de un flujo cualquiera (HPE/BPMN) por medio del plug-in para Eclipse . . . . .	61



### 1. Introducción

En el siguiente documento se expone en forma trabajo de fin de grado todo el proceso seguido para la resolución de un problema que existía en el software de la empresa HPE, concretamente con su paquete de servicios *Service Activator*, un sistema orientado a las empresas de telecomunicaciones que buscan automatizar todo el proceso de altas, bajas y configuraciones para equipos en remoto de clientes que se suceden diariamente en cualquier empresa de esta índole.

Dado que tanto su tamaño como complejidad es elevada, dicho sistema se divide en diferentes partes, cada una enfocada a la resolución de un problema concreto y necesario para que su conjunto funcione correctamente. En mi caso nos centramos en una parte del sistema que compone el *Service Activator*, denominada *Workflow Manager* que se encarga de interpretar flujos de trabajo creados previamente y constatar su correcta finalización o informar de los errores ocurridos durante la ejecución del mismo.

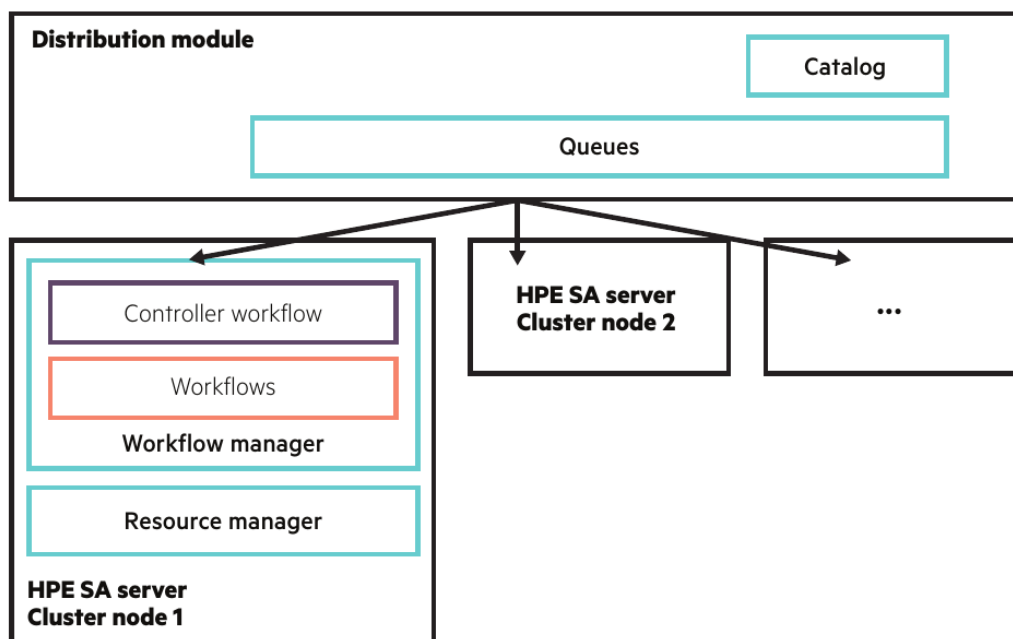


Figura 1: Workflow Manager [1, p. 6]

Un flujo de trabajo consiste en una serie de nodos que conforman una lista enlazada donde cada nodo realiza una operación muy concreta y sencilla (sumar, multiplicar, ejecutar una función en java...) pero que juntos permiten realizar operaciones tan complejas como dar de alta un nuevo servicio a un cliente de cualquier empresa que tenga contratado el *Service Activator*, siempre de una manera

## Introducción

---

totalmente automatizada.

El problema al que se enfrentaba dicho sistema radica en conocer la posición que ocupa un nodo o simplemente editar el orden en el que aparece dicho nodo dentro de un flujo de trabajo ya creado, el trabajo que había que realizar era enorme ya que al tratarse de una lista enlazada, la información relativa al orden que ocupa cada nodo es inexistente dado que únicamente se conoce el nodo siguiente y el anterior de un mismo nodo; indicar que a mayores existían numerosos problemas a la hora de gestionar los nodos y workflows mediante un gestor de versiones tipo *git*, ya que al estar juntas tanto la información de la posición del nodo como la información relativa a su función, al modificar cualquier nodo intermedio del workflow todos los nodos relacionados con el sufrían modificaciones y dichas modificaciones quedarían registradas en el control de versiones provocando inconsistencias en los casos donde estuvieran dos o más personas modificando un mismo workflow simultáneamente, eso por no hablar de todos datos relacionados con la interfaz gráfica que a mayores se verían modificados con cada cambio de orden de los nodos.

Como solución a dicho problema se propuso la creación de una separación entre la lógica del propio nodo y el orden que este tenía en un workflow, en otras palabras la lógica del workflow. Para ello, como iremos viendo según avancemos por el documento utilizaremos la notación **BPMN** (Business Process Model and Notation) que nos provee de un sistema muy flexible y escalable para diseñar y modelar flujos de trabajo y nos ayudaremos de las transformaciones xsl (XSLT) para convertir los flujos actuales del *WorkflowManager* a este nuevo sistema finalizando con el desarrollo de un plugin para el IDE *Eclipse* que automatice dicha conversión (bidireccional) durante el mismo proceso de edición del workflow.

El trabajo está dividido en ocho apartados:

- **Objetivos:** Donde se expondrán los hitos principales por los que pasará el proyecto.
- **Planificación:** Donde se mostrará la planificación original y la seguida hasta la finalización del proyecto.
- **Análisis:** Veremos como es posible aplicar la notación BPMN a los flujos HPE y comprobar el alcance de la solución al problema.
- **Diseño:** En esta sección podremos comprobar como se enfocará la implementación de la solución tras el análisis teniendo en cuenta diversos aspectos como la escalabilidad, la flexibilidad y el rendimiento.

## Introducción

---

- **Implementación:** Como su nombre indica veremos como se ha implementado la solución propuesta en el apartado anterior y los problemas encontrados durante el desarrollo de la misma.
- **Pruebas** Donde se expondrá la batería de pruebas seguida y los resultados obtenidos.
- **Control y seguimiento:** Expondremos los objetivos cumplidos y los problemas encontrados durante el transcurso del proyecto así como las versiones obtenidas durante la fase de implementación y el estado actual del proyecto.
- **Conclusiones y trabajo futuro:** Terminaremos haciendo un informe final de lo que ha supuesto realizar el proyecto y el trabajo futuro que se puede realizar para ampliar la funcionalidad del mismo.
- **Apéndices:** Donde podremos ver algunos ejemplos así como al estructura de numerosas entidades del modelado BPMN que iremos citando durante el transcurso del documento.

Añadir que durante el desarrollo de todo el proyecto únicamente se utilizaron herramientas de código abierto a excepción del Astah para el desarrollo del primer diagrama.

## 2. Objetivos

El desarrollo del proyecto se dividió en cuatro claros objetivos.

1. **Tabla de correspondencias:** Primer hito del proyecto, en el cual se estudió la documentación aportada por HPE para obtener las estructuras más representativas que conforman los workflows y que necesitaríamos para contemplar las entidades a representar mediante la notación BPMN. Como objetivo final de este hito se pidió la realización de una tabla de correspondencias entre las entidades existentes en la tecnología de HPE y las ofrecidas por la notación BPMN.
2. **Estudio en profundidad del estándar BPMN y de posibles implementaciones.** Tras un primer análisis tendremos que realizar un estudio mucho más específico del estándar BPMN para intentar comprender con mayor detalle la responsabilidad de cada estructura y poder relacionar con mayor precisión los diferentes elementos de ambas plataformas. También deberemos estudiar las facilidades que nos aportan las diferentes herramientas existentes en el ámbito de la informática a la hora de implementar el traductor y que nos permitan generar un producto final con ciertas garantías.
3. **Implementación del traductor** Donde pasaremos a implementar el traductor en base al diseño previo. Al usar tecnologías comunes (ambas tecnologías utilizan XML para persistir los datos), se crearán dos convertidores (HPE->BPM y BPM ->HPE) basados en XSLT para obtener ambas representaciones.
4. **Desarrollo del plugin para *Eclipse*** Una vez que se comprobara el correcto funcionamiento de ambos transformadores con workflows reales, se procedería al desarrollo de un plugin que automatizara el proceso de conversión y que se integrara con la plataforma actual que están utilizando para la edición de los flujos; hablamos de *Eclipse*. Mediante este plugin, el IDE sería capaz de convertir de una notación cualquier workflow de una manera cómoda e intuitiva.

Como se puede comprobar todos estos objetivos estaban orientados a completar un gran objetivo final que consistía en implementar una nueva representación de los workflows actuales de HPE que permitiese la separación de la lógica de los nodos y del propio workflow, así como el desarrollo de un sistema que facilitase la conversión entre estas dos representaciones que culminó con su integración en *Eclipse* mediante el desarrollo de un plugin.

### 3. Planificación

#### 3.1. Organización del trabajo

El desarrollo de todo el proyecto se realizará por Fernando Merino Cejudo, a excepción de las tareas de planificación que serán compartidas con el jefe de proyecto, que en este caso será la empresa HPE. Por tanto durante el transcurso del proyecto adoptaré diferentes roles según sea necesario.

#### 3.2. Roles y responsabilidades

Rol	Responsabilidades	Encargado
Jefe de Proyecto	Dirigir Planificar Supervisar tareas Identificar recursos Gestionar riesgos Determinar costes	HPE
Analista	Identificar objetivos Obtener requisitos Modelar los casos de uso	Fernando M.C.
Arquitecto de software	Determinar tecnologías a usar Aplicación de patrones (si procede) Gestión de los requisitos no funcionales Supervisión de la calidad de la solución Creación de diagramas	Fernando M.C.
Programador	Comprensión del diseño indicado Implementación de la solución Documentación del código generado Mantenimiento de dicho código	Fernando M.C.
Testeador	Diseño de las pruebas Implementación de las pruebas Interpretación de los resultados Creación de informes sobre la calidad del producto	Fernando M.C.

Tabla 1: Asignación de roles y responsabilidades

### 3.3. Fases del proyecto

El proyecto está definido en cuatro fases:

Fase	Hito
Fase de inicio	Se estudia el problema a resolver y ambas tecnologías (HPE y BPMN). Esta fase finaliza con la entrega de un documento donde se analizan las estructuras más importantes del modelo de HPE junto con su análogo en BPMN
Fase de elaboración	En esta fase se realizará un estudio más exhaustivo del estándar BPMN concretando las estructuras que se usarán para su implementación y como se usarán. Así mismo se estudiará la tecnología a usar para implementar una solución al problema visto en la anterior fase. Esta fase terminará con un documento donde se podrá estudiar en detalle la representación de cada entidad de HPE en BPMN
Fase de construcción	Donde se implementará la solución definida en la anterior fase según lo dispuesto en el documento generado. También se realizarán en esta fase el conjunto de pruebas necesario para comprobar la calidad de la solución creada. Esta fase finalizará con la implementación de un traductor por línea de comandos para la realización de las pruebas.
Fase de transición	Se generó la documentación final y finalizará con la creación del plugin para eclipse una vez testeado el traductor implementado en la anterior fase.

Tabla 2: Fases del proyecto

El tiempo acordado para la realización de cada fase se midió en meses, ya que al realizar todo el desarrollo solo una persona y sin poder dedicarse esta a tiempo completo en la realización de la misma, se procedió a ampliar los márgenes de tiempo existentes entre fase y fase, lo que conllevó aumentar el número de reuniones de seguimiento entre las fases para comprobar el estado del proyecto durante las mismas. El proyecto se presentó a finales de noviembre y como fecha límite para su entrega se propuso el mes de junio, por lo que habría 6 meses para la realización de este proyecto.

Para cumplir los plazos asignados a cada fase del proyecto nos auto planificábamos en base a hitos, es decir estudiábamos la cantidad de trabajo que llevaría aproximadamente cada fase y la dividíamos internamente en diferentes puntos. De este modo podíamos ir generando una realimentación progresiva donde podíamos contrastar el tiempo que restaba de la fase con los hitos pendientes.

### 3.3.1. Transcurso del proyecto

En un primer momento la planificación general del proyecto se dividió de la siguiente forma:

Fase	Tiempo
Fase de inicio	2 meses (Diciembre-Enero)
Fase de elaboración	2 meses (Febrero-Marzo)
Fase de construcción	2 meses (Abril-Mayo)
Fase de transición	1 mes (Junio)

Tabla 3: Planificación original

Pero según avanzó el proyecto dicha planificación se sufrió varios cambios como veremos más adelante quedando de la siguiente manera:

Fase	Tiempo
Fase de inicio	2 meses (Diciembre-Enero)
Fase de elaboración	4 meses (Febrero-Mayo)
Fase de construcción	1 mes (Junio)
Fase de transición	1 mes (Julio)

Tabla 4: Planificación final

Como vemos, existen dos grandes cambios en la planificación:

1. En la fase de elaboración, se duplicó el tiempo estimado para su elaboración (lo que llevó que se retrasara la entrega del proyecto un mes). Esto se debió a que a la hora de escoger las estructuras más adecuadas para la traducción de las entidades entre HPE y BPMN, este último estándar resultó bastante más complejo de lo esperado ya que al abarcar un gran espectro de flujos de trabajo todas sus estructuras se basaban en conceptos abstractos donde no siempre quedaba bien definida su responsabilidad, teniendo que redefinir continuamente la relación entre ambas plataformas. Algo que también dificultó bastante su comprensión, es que existía muy poca documentación sobre la lógica interna de BPMN que era necesaria para modelar correctamente un flujo, tan solo había documentación (a excepción del propio estándar, donde se detallaban por pormenores del mismo) sobre los elementos más importantes a la hora de crear un flujo como pueden ser las *Gateways* o las *Activities*.

Aparte de la complejidad que llevó el estudio del estándar BPMN, también hubo que añadir el tiempo que llevó el estudio de las diferentes tecnologías empleadas en esta fase como en la posterior (XSD, XSL...) ya que antes de

proceder a implementar la solución en una tecnología tuvimos que realizar un estudio de viabilidad sobre la misma. A grandes rasgos, gran parte del retraso que se sufrió en esta parte se debió a que había numerosas tecnologías nuevas que había que comprender a mayores de la complejidad inherente de BPMN.

2. En la fase de construcción vemos también un cambio sustancial, en este caso relativo a un adelanto de un mes, debido en gran parte a el trabajo previo realizado en la anterior fase, ya que al estudiar todas las tecnologías necesarias para su implementación y realizar un estudio exhaustivo de BPMN, tan solo hubo que realizar unos pocos cambios en el diseño y comprender algo más afondo el estilo de programación relativo al XSL. Exceptuando esos pequeños problemas, su desarrollo se realizó en escaso mes.

En el resto de fases no hubo contratiempos ni imprevistos significantes por lo que se realizaron en el tiempo previsto.



### 3.4. Análisis de riesgos

Organizados en base a su impacto en el proyecto.

Riesgo-01	Fallo en el disco duro
Descripción	Fallo en el disco duro donde se aloja el trabajo realizado actualmente
Efecto	Pérdida de la información contenida en dicho disco duro
Frecuencia	Baja
Impacto	Alto
Detención	Alto
Plan de contingencia	Generar backups periódicos del trabajo realizado en un soporte externo
Análisis Coste/Beneficio	Como coste tendríamos la adquisición de una memoria complementaria más el esfuerzo que conlleva realizar backups periódicamente, un coste cada vez más bajo frente a la posible pérdida total del trabajo realizado hasta la fecha

Tabla 5: Riesgo-01

Riesgo-02	Fallo en alguno de los equipos de desarrollo
Descripción	En alguno de los equipos de desarrollo se estropea una parte fundamental del mismo que impide el inicio del sistema (por ejemplo se quema la placa base)
Efecto	Imposibilidad de seguir desarrollando en dicho equipo
Frecuencia	Muy baja
Impacto	Medio
Detención	Muy Alto
Plan de contingencia	Mantener todo el trabajo en otro soporte (preferiblemente la nube) y disponer de otro equipo que posibilite el desarrollo. Al contrario que el riesgo-01, éste tiene un coste asociado al plan de contingencia muy alto, si también es cierto que su impacto es medio, su probabilidad es tan baja que no merece la pena asumir los costes que llevaría aplicar el plan de contingencia
Análisis Coste/Beneficio	

Tabla 6: Riesgo-02

Riesgo-03	Fallo en la planificación
Descripción	Realizar una planificación de las fases del proyecto incorrecta, lo que puede conllevar un retraso en la entrega del proyecto
Efecto	Aumento de los costes del proyecto y una posible cancelación del mismo
Frecuencia	Media
Impacto	Alto
Detención	Media
Plan de contingencia	Revisar la planificación e ir modificando los recursos necesarios en cada hito en cada momento
Análisis Coste/Beneficio	Como coste asociado al plan de contingencia podría conllevar la contratación de nuevos integrantes del equipo frente al posible fracaso del proyecto, por lo que sería interesante aplicar el plan de contingencia.

Tabla 7: Riesgo-03

Riesgo-04	Comprensión incorrecta del problema en la fase de análisis
Descripción	Se tiene una idea inexacta del problema que debe de solucionar el proyecto una vez finalizado
Efecto	De desarrolla el proyecto de una forma incorrecta, con la consiguiente pérdida de tiempo y recursos
Frecuencia	Medio
Impacto	Muy Alto
Detención	Medio
Plan de contingencia	Realizar una fase de análisis muy exhaustiva con una toma de requisitos (tanto funcionales como no funcionales) contrastada con el cliente y revisada en todo momento.
Análisis Coste/Beneficio	Comparado con el coste en recursos y tiempo que supondría no realizar el análisis (el cual supondría a su vez un coste de tiempo y recursos también ) merece la pena realizar dicha inversión ya que a parte de ser menor el coste asociado al análisis frente al del riesgo, generar dicho análisis crea una sensación de seguridad muy positiva a lo largo del proyecto.

Tabla 8: Riesgo-04

Riesgo-05	Diseño incoherente
Descripción	Modelar al futura solución aplicando herramientas poco adecuadas o violando alguno de los principios básicos del diseño de software como un alto acoplamiento o no asignar coherentemente las responsabilidades de cada entidad que compongan el sistema final
Efecto	Necesidad de rediseñar el sistema (si es posible) o la creación de limitaciones permanentes en la solución final
Frecuencia	Media
Impacto	Media
Detención	Baja
Plan de contingencia	Realizar un diseño coherente y bien documentado que permita crear una solución adaptable a los cambios posteriores que pudieran aparecer
Análisis Coste/Beneficio	Como coste tendríamos la inversión de tiempo que conlleva realizar un diseño tan profundo del sistema y el estudio de metodologías de desarrollo del software si fuera necesario.

Tabla 9: Riesgo-05

Riesgo-06	Ausencia de documentación del producto final
Descripción	Creación de una documentación escasa o de mala calidad sobre el producto final
Efecto	Imposibilidad de dar mantenimiento a la plataforma generada o con un coste asociado muy alto
Frecuencia	Alta
Impacto	Medio
Detención	Bajo
Análisis Coste/Beneficio	Ir documentando el código según se va escribiendo. El coste de este plan de contingencia radicaría en un aumento del tiempo destinado al desarrollo del sistema, pero que repercutiría en una disminución del tiempo necesario para mantener el código en un futuro

Tabla 10: Riesgo-06

Riesgo-07	Indisposición de algún recursos disponible del proyecto
Descripción	Imposibilidad de contar con un recursos que se creía disponible
Efecto	Retraso en el tiempo de desarrollo del proyecto imprevisto en al mayor parte de los casos
Frecuencia	Media
Impacto	Bajo
Detención	Alto
Plan de contingencia	Contratar a otro integrante que supla su ausencia temporalmente
Análisis Coste/Beneficio	El coste de de este plan de contingencia es variable, ya que depende del recurso que no esté disponible. Su aplicación dependerá también del retraso/coste que suponga la ausencia de dicho recurso para el cómputo global del proyecto

Tabla 11: Riesgo-07

Riesgo-08	Creación de una mala batería de pruebas
Descripción	Diseño erróneo de la batería de pruebas o directamente la no ejecución de las mismas
Efecto	Perdida de conocimiento sobre la calidad del producto final
Frecuencia	Alta
Impacto	Medio
Detención	Medio
Plan de contingencia	Generar una batería de pruebas previa al desarrollo del componente del sistema.
Análisis Coste/Beneficio	Una vez más el coste del plan de contingencia se basa en el aporte a mayores de tiempo que tenemos que invertir en la etapa de desarrollo en este caso en pro de una mayor seguridad en cuanto a la calidad del producto final.

Tabla 12: Riesgo-08

## 3.5. Análisis de recursos

Fase	Recursos Materiales	Recursos Humanos
Fase de inicio	Ordenador	Fernando M.C
	Acceso a Internet	
	Herramientas de ofimática ( $\text{\LaTeX}$ )	Tutor UVA
	Documentación (HPE y BPMN)	
	Almacenamiento de backup	
Fase de elaboración	Herramienta de modelado (Astah)	Tutores HPE
	Ordenador	
	Acceso a Internet	
	Herramientas de ofimática ( $\text{\LaTeX}$ )	Fernando M.C.
	Almacenamiento de backup	
Fase de construcción	Documentación (HPE y BPMN)	
	Diversa bibliografía (XSD XSLT..)	
	Ordenador	
	Acceso a Internet	Fernando M.C
	Herramientas de ofimática ( $\text{\LaTeX}$ )	
Fase de transición	Documentación (HPE y BPMN)	
	Almacenamiento de backup	
	Diversa Bibliografía (XSD XSLT..)	Apoyo externo UVA <sup>1</sup>
	Herramientas de control de versiones (Git)	
	Intérprete XLST (Saxon)	
Fase de transición	Ordenador	
	Acceso a Internet	Fernando M.C
	Herramientas de ofimática ( $\text{\LaTeX}$ )	
	Almacenamiento de backup	
	Intérprete XLST (Saxon)	
Fase de transición	Documentación (Eclipse y Plug-ins)	Tutores HPE.
	Eclipse IDE	

Tabla 13: Análisis de recursos

A mayores indicar que durante el transcurso del proyecto se realizaron diversas reuniones de seguimiento tanto con los tutores de HPE como con el tutor de la UVA que en la anterior tabla no están contempladas ya que creemos que las revisiones con el cliente son imprescindibles y por eso su presentación explícita se puede obviar. Para dichas reuniones se utilizó la plataforma *Skype* y la página Web *Trello* como medios para compartir información.

<sup>4</sup> Fué necesaria la ayuda de un experto en XSLT para contrastar la comprensión obtenida sobre XSLT y poder realizar su desarrollo conforme a las buenas prácticas que esta tecnología exige.

### 3.6. Análisis de costes

Sobre el coste asociado al desarrollo del proyecto expondremos dos versiones: la real (obtenida durante nuestro desarrollo del proyecto) y una hipotética donde cuantificaremos cada recurso suponiendo que no estuviéramos bajo el amparo de la Universidad de Valladolid (por ejemplo en el caso que fuésemos una empresa aparte que actúe como cliente para HPE).

#### 3.6.1. Real

El coste final del desarrollo del proyecto fue cero, veamos el porqué:

- **Ordenador** ya se disponía del el antes de realizar el proyecto, por lo que no ha sido necesaria realizar su compra.
- **Acceso a Internet** todo el proyecto se había podido realizar en las instalaciones de la UVA, por lo que su acceso sería gratuito.
- **Herramientas de ofimática (L<sup>A</sup>T<sub>E</sub>X)**, gratuitas.
- **Almacenamiento de backup**, al igual que con el ordenador, ya se disponía previamente de un soporte externo.
- **Herramientas de modelado (Astar)**, ya que la universidad de Valladolid adquirió la licencia pertinente para uso lectivo, esta no ha supuesto coste alguno.
- **Documentación HPE y BPMN**, cedida y libre respectivamente.
- **Bibliografía utilizada**, cedida por la biblioteca de la universidad de Valladolid.
- **Herramientas destinadas al control de versiones (Git)** gratuitas.
- **Intérprete de XST (Saxón)**, gratuito, ya que se ha utilizado una versión libre en vez de la licenciada.
- **Eclipse IDE**, gratuito.
- **Tiempo aportado por el tutor de la UVA**, sin coste para el alumno ya que está subvencionado por la universidad de Valladolid.
- **Tiempo aportado por Fernando M.C** sin coste ya que obtiene la presentación de su TFG como contraprestación.

- **Tiempo aportado por el apoyo externo cedido por la UVA**, sin coste ya que lo cedió altruistamente.

Como podemos ver muchos de los costes contemplados anteriormente han sido subvencionados por la UVA, si no posiblemente hubiera supuesto un coste adicional la realización de dicho proyecto.

### 3.6.2. Hipotético

Suponiendo que no tuviéramos el amparo de la UVA para la realización de dicho proyecto y fuésemos una empresa que tuviera como cliente a HPE, los costes podrían ser los siguientes:

- **Fase de inicio:**

Recursos	Unidades	Total
Ordenador	1	400€
Acceso a Internet	1	20€
L <sup>A</sup> T <sub>E</sub> X	1	0€
Documentación (HPE y BPMN)	1	0€
Almacenamiento de backup	1	60€
Fernando M.C (horas)	120	4200€
Total		4.680 €

Tabla 14: Análisis de costes (Fase de Inicio)

- **Fase de elaboración:**

Recursos	Unidades	Total
Ordenador	1	0€
Acceso a Internet	1	20€
L <sup>A</sup> T <sub>E</sub> X	1	0€
Documentación (HPE y BPMN)	1	0€
Almacenamiento de backup	1	60€
Diversa bibliografía	2	100€
Fernando M.C (horas)	100	3500€
Total		3.680 €

Tabla 15: Análisis de costes (Fase de elaboración)

El coste del ordenador al igual que el del almacenamiento externo (backup) sería de cero euros ya que se habrían adquirido en la anterior fase del proyecto.

■ **Fase de construcción:**

Recursos	Unidades	Total
Ordenador	1	0€
Acceso a Internet	1	20€
L <sup>A</sup> T <sub>E</sub> X	1	0€
Almacenamiento de backup	1	0€
Documentación (HPE y BPMN)	1	0€
Diversa bibliografía	2	0€
Git	1	0€
Saxon	1	0€
Fernando M.C (horas)	100	3500€
Apoyo experto	10	500€
Total		4.020 €

Tabla 16: Análisis de costes (Fase de construcción)

■ **Fase de transición:**

Recursos	Unidades	Total
Ordenador	1	0€
Acceso a Internet	1	20€
L <sup>A</sup> T <sub>E</sub> X	1	0€
Almacenamiento de backup	1	0€
Documentación (Eclipse y Plug-ins)	2	120€
Eclipse IDE	1	0€
Saxon	1	0€
Fernando M.C (horas)	50	1750€
Total		1.890 €

Tabla 17: Análisis de costes (Fase de transición)

Todo esto nos hace un total de 14.270 € suponiendo que la hora se cotizara a treinta y cinco euros.

Hemos supuesto treinta y cinco euros basándonos en la propuesta realizada por el tutor Bejamín para intentar mostrar lo que creemos sería más justo tanto para el cliente como para el ingeniero de desarrollo, aunque si bien es cierto que se valora mucho más la hora del jefe de proyecto que la de un programador junior, en este supuesto donde existe una única persona que va pasando por los diferentes roles, no merece la pena ir seleccionando las horas necesarias para cada rol.



## 4. Análisis

Como comentábamos previamente, el problema a solucionar residía en la necesidad de separar la lógica de cada nodo de la lógica del workflow, es decir tener por una parte la representación de la funcionalidad del nodo y la representación del propio workflow. Esto aparte de simplificar el trabajo que llevaría editar un workflow, permitiría un mejor control de la información en el gestor de versiones ya que al tener por una parte toda la información relativa a los nodos, al modificar un workflow, dicha información permanecería inalterable, lo que nos permitiría la reutilización de código y la posibilidad de crear/editar diferentes flujos de trabajo simultáneamente sin temor a que existan colisiones entre ambas ediciones.

Otro aspecto a tener en cuenta, es que al transformar el flujo de HPE a la notación BPMN, podremos acceder a una gran cantidad de herramientas ya desarrolladas (**Bonita BPM**, **Modelio**, o hasta **Microsoft Visio**) que nos permiten editar cualquier workflow gráficamente y de una forma bastante más elaborada ahorrándonos la necesidad de desarrollar nosotros mismos dichas aplicaciones.

### 4.1. Análisis de requisitos

#### 4.1.1. Requisitos de usuario

- RU-01 El usuario será capaz de traducir un flujo HPE a otro tipo flujo que sea capaz de llevar la lógica sobre la posición de los nodos a otra capa.
- RU-02 El usuario podrá obtener un flujo HPE idéntico al que tenía en un principio desde otro flujo al que se le ha realizado previamente la conversión indicada en el anterior punto.

#### 4.1.2. Requisitos funcionales

- RF-01 El sistema permitirá traducir un flujo desde el estándar HPE hacia otro estándar que permita agregar otra capa de información donde se indique la posición de los nodos.
- RF-02 El sistema será capaz de traducir un flujo desde la nueva notación hacia de la HPE generando un flujo idéntico al original ( de HPE)

#### 4.1.3. Requisitos no funcionales

- RNF-01 El traductor se integrará con la última versión Eclipse (*Neon*) mediante un plug-in.

- RNF-02 El traductor deberá realizar la conversión en un tiempo razonable (menos de un minuto).
- RNF-03 Toda la documentación generada deberá estar en inglés.
- RNF-04 El código generado deberá estar correctamente documentado y organizado.
- RNF-05 El proceso de conversión deberá ser intuitivo para el usuario.
- RNF-06 Ambos flujos podrán ser validados por cualquier herramienta diseñada para dicho propósito (ej. *xmlint*)

## 4.2. Casos de uso

A continuación pasaremos a estudiar los tres únicos casos de uso que implementará nuestro sistema :

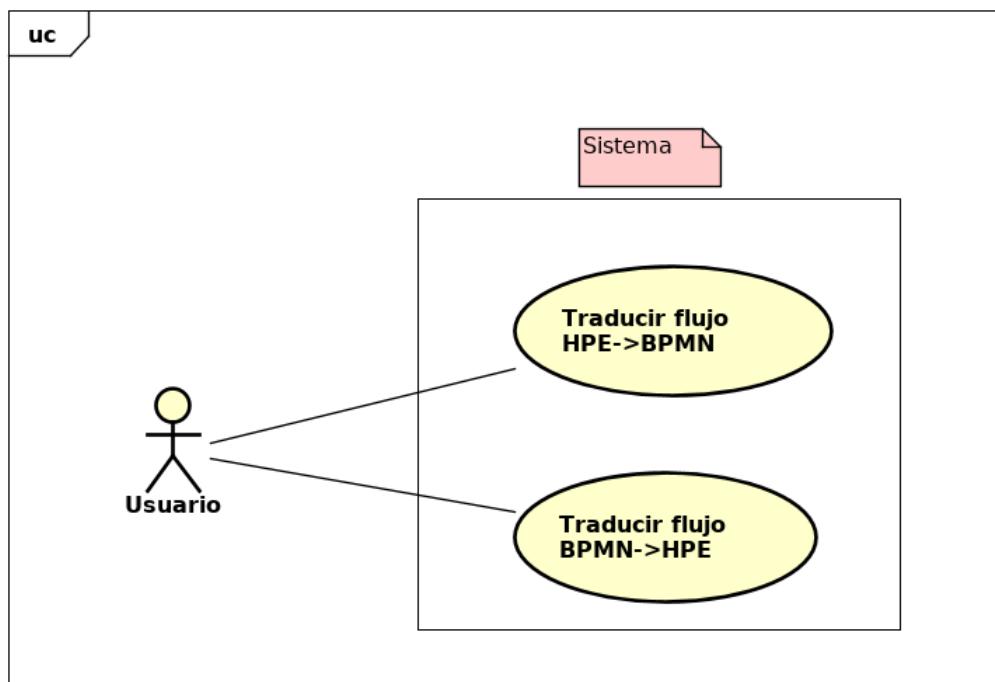


Figura 2: Diagrama casos de uso

**4.2.1. CU-01 Traducir flujo HPE->BPMN****Descripción:**

El usuario quiere traducir un flujo modelado bajo el estándar de HPE y convertirlo en un flujo idéntico pero modelado por BPMN.

**Precondición:**

El usuario debe de disponer previamente de un flujo modelado por HPE y una versión de Eclipse igual o superior a *Neon*.

**Secuencia:**

**Paso 1** El usuario ejecuta su versión de Eclipse.

**Paso 2** El usuario busca el flujo a convertir a BPMN.

**Paso 3** El usuario pulsa con el secundario sobre el flujo a convertir.

**Paso 4** El sistema muestra un botón que permita convertir el flujo seleccionado a BPMN.

**Paso 5** El usuario clickea con el botón izquierdo de su ratón sobre el botón indicado anteriormente.

**Paso 6** El sistema crea en la misma carpeta donde se encontraba el flujo original otro flujo con el mismo nombre y la extensión *.bpmn*.

**Paso 7** El sistema mostrará una notificación indicando que se ha traducido a BPMN.

**Excepciones:**

**Paso 6** El sistema no es capaz de traducir correctamente el flujo, por tanto creará un flujo de la misma forma que lo haría si hubiera procedido correctamente, solo que dicho fichero consistiría en un fichero vacío.

**Paso 6** En dicha carpeta ya existe otro archivo con ese nombre y esa extensión. En este caso el sistema sobrescribirá dicho archivo con el nuevo flujo.

**Postcondiciones:**

El usuario obtiene otro fichero en la misma localización que su flujo original con una extensión *.bpmn* bien con el contenido del flujo HPE traducido o bien vacío, dependiendo de si se ha realizado correctamente o no.

**4.2.2. CU-02 Traducir flujo BPMN->HPE****Descripción:**

El usuario quiere traducir un flujo modelado bajo el estándar de BPMN y convertirlo en un flujo idéntico pero modelado por HPE.

**Precondición:**

El usuario debe de disponer previamente de un flujo modelado por BPMN y una versión de Eclipse igual o superior a *Neon*.

**Secuencia:**

**Paso 1** El usuario ejecuta su versión de Eclipse.

**Paso 2** El usuario busca el flujo a convertir a HPE.

**Paso 3** El usuario pulsa con el secundario sobre el flujo a convertir.

**Paso 4** El sistema muestra un botón que permita convertir el flujo seleccionado a HPE.

**Paso 5** El usuario clickea con el botón izquierdo de su ratón sobre el botón indicado anteriormente.

**Paso 6** El sistema crea en la misma carpeta donde se encontraba el flujo original otro flujo con el mismo nombre y la extensión *.xml*.

**Paso 7** El sistema mostrará una notificación indicando que se ha traducido a HPE.

**Excepciones:**

**Paso 6** El sistema no es capaz de traducir correctamente el flujo, por tanto creará un flujo de la misma forma que lo haría si hubiera procedido correctamente, solo que dicho fichero consistiría en un fichero vacío.

**Paso 6** En dicha carpeta ya existe otro archivo con ese nombre y esa extensión. En este caso el sistema sobrescribirá dicho archivo con el nuevo flujo.

**Postcondiciones:**

El usuario obtiene otro fichero en la misma localización que su flujo original con una extensión *.xml* bien con el contenido del flujo BPMN traducido o bien vacío, dependiendo de si se ha realizado correctamente o no.

### 4.3. Matriz de trazabilidad

A continuación se muestra la matriz de trazabilidad donde se podrá comprobar en que casos de uso se incluyen los requisitos funcionales identificados anteriormente:

	CU-01	CU-02
RF-01	✓	
RF-02		✓

Tabla 18: Matriz de trazabilidad

### 4.4. Modelo de dominio

Pasemos a ver el diagrama de dominio para comprender los principales componentes del sistema:

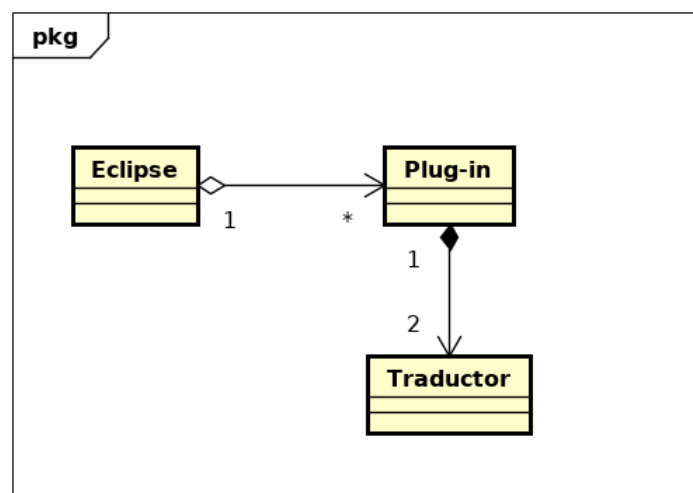


Figura 3: Modelo de dominio

Únicamente tenemos tres instancias importantes:

- **Eclipse**, que representa al IDE Eclipse sobre el cual basaremos el plug-in que desarrollaremos.
- **Plug-in**, representa la estructura que permite ampliar la funcionalidad de *Eclipse* y nos permite embeber en dicha plataforma nuestro traductor.
- **Traductor**, el objetivo final del proyecto que realizará las traducciones de un estándar a otro.

Aclarar en que el anterior diagrama de dominio se ha indicado que un plug-in se compone de dos traductores, esto es cierto únicamente en nuestro sistema, dado que un plug-in puede ampliar de cualquier modo la funcionalidad de eclipse, no únicamente añadiendo traductores.

## 4.5. Diagrama de secuencia

Dado que únicamente tenemos dos casos de uso, y que como podemos ver en su definición únicamente cambia el traductor empleado a la hora de traducir un flujo, mostraremos únicamente un único diagrama de secuencia que ayude a ejemplificar ambos casos:

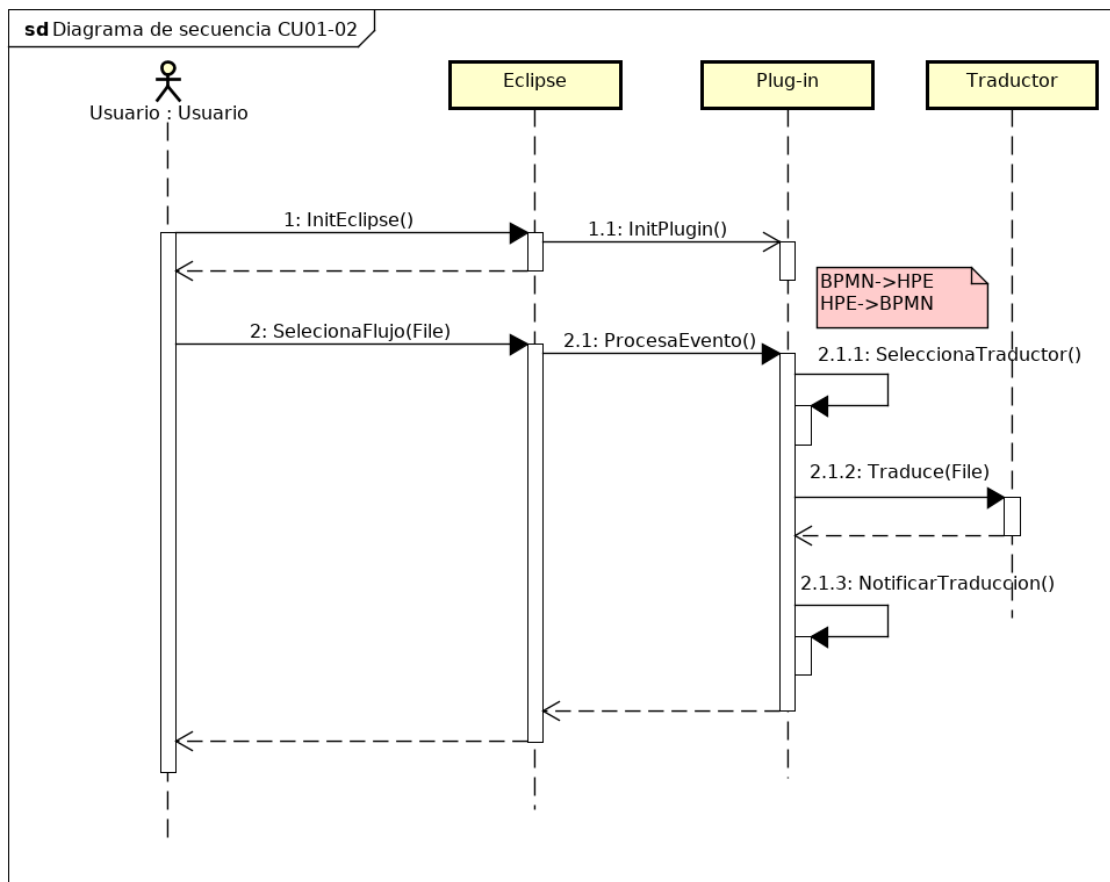


Figura 4: Diagrama secuencia CU01-02

Como vemos, la diferencia entre ambos casos de uso de encuentra en el paso 2.1.1, donde en base a la extensión del flujo, el sistema escogerá aplicar un traductor u otro.

## 4.6. Entidades del sistema *HPE Service Activator*

Para conseguir portar el diseño actual de los flujos de HPE a la notación BPMN, lo primero que realizamos (tras la obtención de los requisitos) fue la recolección de las estructuras básicas que componen dicho flujo.

**Workflow**, representa un tipo de proceso de trabajo o la activación de un proceso que ha sido automatizado. Cada workflow está dividido en porciones discretas llamadas nodos [1, p. 18].

**Nodos**, el componente más pequeño de un flujo de trabajo que realiza una función específica[1, p. 28]. Existen tres tipos de nodos:

- **Process Node**: o nodo proceso, el cual realiza algún tipo de tarea que normalmente modifica el estado del *case-packet* asociado al workflow.
- **Rule Node**: caracterizado por realizar una simple comprobación permitiendo tomar un camino u otro dentro del mismo flujo de trabajo. Similar a la función que realiza un if en java.
- **Switch Node**: similar a la función que tiene el switch de java, este nodo toma un camino de entre dos o más opciones en base a una comprobación que realiza generalmente de las variables que conforman el *case-packet*.

**Handlers**, similar a los nodos en un flujo de trabajo pero que no aparecen en el flujo estándar del mismo [1, p. 31]. Hay tres tipos de disparadores:

- **Error Handlers**, son llamados cuando un proceso en un workflow genera una excepción. Acto seguido finaliza el flujo.
- **End Handlers**, son llamados cuando el flujo de trabajo finaliza, comúnmente para liberar los recursos que el propio flujo haya reservado.

**Case-packet**, es una colección de variables globales para todo el workflow donde cada nodo puede acceder a ellas modificando u obteniendo sus valores según se vaya necesitando [1, p. 33]. Esta colección de variables puede ser de cinco tipos:

- String, cuyo valor por defecto es un string vacío.
- Boolean, por defecto falso.
- Integer, 0
- Float, 0,0

- Object, null

**Roles**, que especifican quién puede ejecutar una operación o interactuar con el workflow [1, p. 27] Existen cuatro tipos diferentes de roles:

- **Default Role**, indica quién está autorizado para interactuar con el flujo de trabajo si no se especifica ningún otro rol para las siguientes acciones, *Iniciar/Rastrear/Finalizar*. También expone quién puede leer los mensajes asociados al flujo.
- **Start Role**, indica a quién le está permitido iniciar ese flujo, si no está especificado obtendrá el valor del rol por defecto.
- **Trace Role**, indica quién puede comprobar el estado actual del flujo.
- **Kill Role**, indica a quién le está permitido finalizar el flujo actual.

Nota: También se puede especificar un rol a nivel de nodo que define a los usuarios que pueden interactuar con dicho nodo o recibir mensajes del mismo. Esta opción a nivel de nodo sobrescribe el rol por defecto del workflow y generalmente solo es apropiada por nodos que que necesiten interactuar con el usuario como *AskFor* or *PutMessage*.

**Queue**, o colas permiten que diversos nodos de cualquier flujo de trabajo creen mensajes o realicen preguntas que requieran una respuesta por parte del usuario; todas estas peticiones/mensajes se almacenan en colas cuyos items no tienen asociado un orden implícito.[1, p. 43].

Hay dos tipos de colas:

- **Message queues**, destinadas a los mensajes que no necesiten ninguna respuesta. Estos mensajes se componen de una simple cadena y no requieren que el flujo de trabajo espere a enviar el mensaje..
- **Request queue**, orientadas a las peticiones creadas por el workflow. Existe un API externa que realiza dicha petición y al contrario que en la cola de mensajes, el flujo espera hasta que o bien recibe una respuesta o se supera un timeout preestablecido.



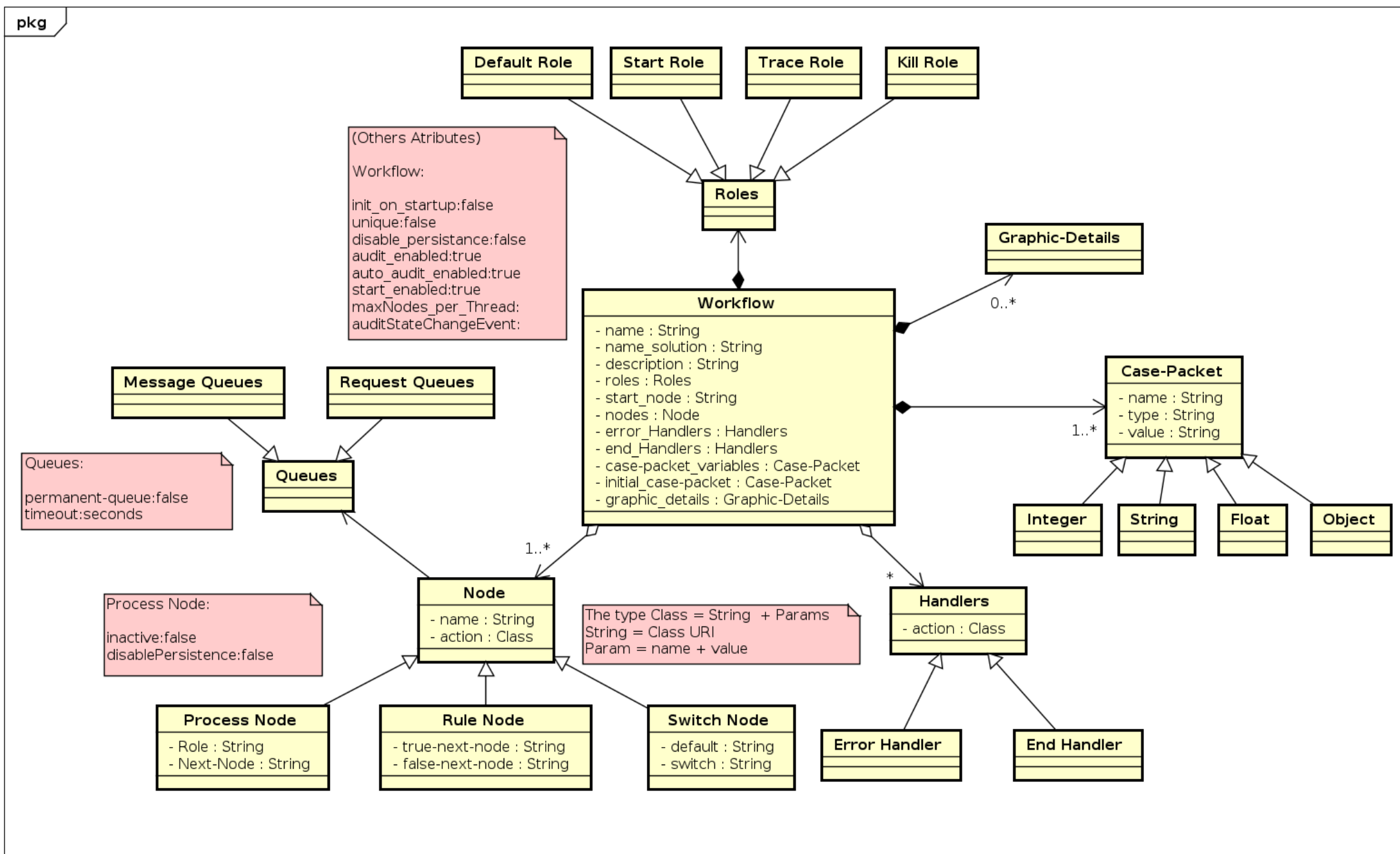


Figura 5: Diagrama de entidades que conforman un flujo de trabajo

## 4.7. Business Process Model and Notation (BPMN)

Tras extraer todas las entidades más representativas que conforman un flujo de trabajo en la implementación de HPE, pasemos a ver cuales son las estructuras semejantes existentes en la notación BPMN que nos permitan realizar la conversión entre ambas plataformas.

**Activity**, consiste en un trabajo que es realizado durante el proceso de trabajo (así se denomina en la notación BPMN a un flujo de trabajo), una actividad puede ser atómica o no atómica (una composición de estas). Los tipos de actividad existentes en un proceso son: [2, p. 151]

- **Task**, una actividad atómica en un flujo de trabajo. La entidad tarea es usada cuando no se puede dividir un trabajo en un nivel de detalle menor [2, p. 151]
- **Sub-Process** otra actividad cuyos detalles internos se han definido usando actividades, puertas de entrada, eventos y flujos de secuencia. [2, p. 173]
- **Call Activity** identifica al punto en un proceso donde se invoca a otro proceso ( de carácter global) o se ejecuta una tarea global.[2, p. 183]

**Data object**, elementos de BPMN cuya finalidad es almacenar y transportar información. Es similar a la función que tienen las variables en muchos lenguajes. [2, p. 203].

Hay seis tipos diferentes de objetos.

- **Data Objects** son la estructura principal para modelar los datos en un flujo. [2, p. 205].
- **Data Objects References** son una manera para reutilizar los objetos de tipo dato en el mismo diagrama. Pueden especificar diferentes estados del mismo objeto en diferentes momentos del proceso. Las referencias de los objetos datos no pueden almacenar información y por contra los objetos de tipo dato no pueden expresar los estados de dichos objetos. [2, p. 205].
- **Data Stores** proveen de un mecanismo a las actividades para acceder o actualizar la información almacenada que persistirá más allá de proceso. A grandes rasgos actúa como un controlador para acceder a la base de datos del sistema. [2, p. 208].
- **Properties**, tienen que formar parte de un elemento de flujo <sup>2</sup> y no tienen una representación gráfica en el diagrama del proceso. Estas propiedades

---

<sup>2</sup>Definición de elemento de flujo

son elementos de datos cuyo cometido es actuar como contenedor de los datos asociados que pudieran necesitar los diferentes elementos flujo de un proceso. [2, p. 210]

- **Data Inputs** es la declaración de un tipo particular de datos que será usado como entrada en la especificación *InputOutputSpecification*. Existen diferentes tipos de datos de entrada preestablecidos asociados a *InputOutputSpecification*. [2, p. 213]
- **Data Outputs**, al igual que los datos de entrada, consisten en la declaración de un tipo particular de dato que actuará como salida en *InputOutputSpecification*. [2, p. 215]

NOTA: Existen más tipos de datos llamados *Data Associations*, los cuales son usados para mover información entre objetos de tipo dato, propiedades, datos de entrada y salida así como de procesos o actividades globales. [2, p. 221]

**Events**, algo que ocurre durante la ejecución de un proceso. Estos eventos suelen afectar al flujo en el que se encuentran y en muchos casos es necesaria una reacción por parte del proceso. [2, p. 233]

Existen tres tipos de eventos:

- **Start Events**, indican donde empieza un proceso.
- **End Events**, indican donde finaliza un proceso.
- **Intermediate Events**, indican que algo ha sucedido en algún punto del proceso entre su inicio y fin.

Estos tres tipos de eventos los podemos encontrar en dos sabores diferentes:

- **Eventos que recogen un evento**. Todos los eventos de inicio y algunos eventos intermedios recogen otros eventos.
- **Eventos que lanzan eventos**. Todos los eventos fin y algunos eventos intermedios lanzan otros eventos que normalmente recogen los eventos mencionados anteriormente.

**Gateways**, o puertas de enlace, son usadas para controlar los diferentes desvíos que un flujo puede tener en un proceso. Algo similar a un `if` en java. [2, p. 287].

Existen seis tipos diferentes de puertas de enlace:

- **Exclusive** que es usada para crear caminos alternativos en un proceso. Actúa básicamente como punto de bifurcación en un flujo. [2, p. 290]

- **Basados en eventos** representan una bifurcación en un proceso donde en vez de evaluar una expresión y adoptar un camino en base a su resultado, se evalúan los eventos que han ocurrido al llegar a esta puerta. [2, p. 297]
- **Parallel** es usada para sincronizar (combinar) diferentes flujos paralelos y/o crear nuevos flujos paralelos.[2, p. 293]
- **Parallel Event-Based** es una puerta de enlace que permite ejecutar diferentes caminos al mismo tiempo, pero al contrario que en la paralela, dichos caminos están basados a su vez en eventos.
- **Inclusive** puede ser usada para crear flujos alternativos incluso paralelos en un mismo proceso. Al contrario que la exclusiva, todas las condiciones son evaluadas. [2, p. 292]
- **Complex** pueden ser usadas para modelar comportamientos de sincronización complejos. Una expresión del tipo *ActivationCondition* es la utilizada para describir esa precisa condición.[2, p. 295]

**Resources**, se usan para especificar recursos que pueden ser referenciados por actividades. Estos recursos pueden ser tanto recursos humanos, como cualquier otro recurso asociado a una actividad durante la ejecución del flujo de trabajo.

## 4.8. Tabla de correspondencias

Tras extraer y analizar todos los conceptos más importantes tanto de una implementación como del modelo BPMN, ya pudimos realizar la primera tabla de correspondencias (primer hito del proyecto) entre ambas. En dicha tabla podemos ver las estructuras necesarias a modelar de HPE, junto con las entidades cuya función final es la más indicada para su conversión

HP	OMG BPMN
Process Node	Activity (Task o Call activity)
Rule / Switch Node	Gateway
Case-Packet	ExtensionElements <sup>3</sup>
End Handler	End Event
Error Handler	Error Event
Default / Start / Trace / Kill Role	Resource (ResourceParameter) <sup>4</sup>

Tabla 19: Estructuras HPE a entidades BPMN

Indicar que esta tabla se ha ido revisando paulatinamente a lo largo del proceso de diseño e implementación tal y como se explica en la nota a pié de página, ya que al tratarse de conceptos muy abstractos no quedaba clara la responsabilidad de cada elemento hasta no cruzar su definición con su descriptor.

<sup>3</sup>En un primer momento, pensé en modelar el *Case-Packet* como un *DataObject*, pero más tarde cuando estaba intentando implementarlo, me dí cuenta de que era totalmente inviable modelarlo de dicha forma por lo que finalmente lo modelé como un elemento de extensión.

<sup>4</sup>Modeladas anteriormente como propiedades, pasaron a ser recursos ya que siendo recursos puedes definir en el flujo de trabajo (workflow) el tipo de roles que necesitas de manera global y te permite especificar dichos roles más tarde, por ejemplo a nivel de nodo.

## 5. Diseño

Para intentar explicar mejor las diferencias existentes entre la implementación de HPE y el estándar de OMG vamos a comparar la estructura una por una para comprobar como todos los elementos de HPE pueden ser modelados según el estándar de OMG.

Indicar que para definir la implementación actual de HPE se está usando la notación DTD, por contra, la notación utilizada para el BPMN se basa en los XMLSchemas, tecnología que adoptaremos para implementar nuestra solución.

### 5.1. Process-Node

Como podemos ver, en la parte izquierda de la página tenemos el DTD que define la estructura que tiene un nodo proceso, pudiendo observar en el lado derecho el XSD completo que define una tarea y una llamada a una actividad.

```

1 <!ELEMENT Name (#PCDATA)>
2 <!ELEMENT Description (#PCDATA)>
3 <!ELEMENT Next-Node (#PCDATA)>
4 <!ELEMENT State (#PCDATA)>
5 <!ELEMENT Role (#PCDATA)>
6 <!ELEMENT Class-Name (#PCDATA)>
7 <!-- Actions -->
8 <!ELEMENT Action (Class-Name, Param*)>
9 <!ELEMENT Param (#PCDATA)>
10 <!-- Param name CDATA #REQUIRED -->
11 <!-- Param value CDATA #REQUIRED -->
12 <!-- Process-Node -->
13 <!ELEMENT Process-Node
14 (Name, Description?, State?,
15 Role?, Action, Next-Node?)>
16 <!-- Process-Node inactive CDATA
17 #IMPLIED -->
18 <!-- Process-Node
19 disablePersistence CDATA #IMPLIED -->

```

Código 1: Process-Node

```

1 <xsd:element name="task" type="tTask"
2 substitutionGroup="flowElement"/>
3 <xsd:complexType name="tTask">
4 <xsd:complexContent>
5 <xsd:extension base="tActivity"/>
6 </xsd:complexContent>
7 </xsd:complexType>

```

Código 2: Task

```

1 <xsd:element name="callActivity" type="
2 tCallActivity" substitutionGroup="
3 flowElement"/>
4 <xsd:complexType name="tCallActivity">
5 <xsd:complexContent>
6 <xsd:extension base="tActivity"/>
7 <xsd:attribute name="calledElement
8 " type="xsd:QName" use="optional"/>
9 </xsd:complexContent>
10 </xsd:complexType>

```

Código 3: CallActivity

Como comentábamos antes, un nodo proceso puede ser una tarea o una llamada a otra actividad, dependiendo si queremos que un nodo realice únicamente una acción o llamee un proceso global (o tarea global cuando queremos sobrescribir las propiedades y atributos del elemento que acabamos de llamar). Una tarea no puede realizar per se ninguna acción, es una clase de la que heredan otras clases

como *ServiceTask* o *ScriptTask*<sup>5</sup>

```

1 <xsd:element name="serviceTask" type="tServiceTask"
  substitutionGroup="flowElement"/>
2 <xsd:complexType name="tServiceTask">
3   <xsd:complexContent>
4     <xsd:extension base="tTask">
5       <xsd:attribute name="implementation" type="
  tImplementation" default="##WebService"/>
6       <xsd:attribute name="operationRef" type="xsd:QName" use="
  optional"/>
7     </xsd:extension>
8   </xsd:complexContent>
9 </xsd:complexType>

```

Código 4: ServiceTask

Como puedes ver en el código anterior, este tipo de nodo tiene un atributo llamado *implementation* donde podemos definir la URI que identifica la otra tecnología o el protocolo de coordinación para enviar y recibir mensajes. En nuestro caso en este atributo definiremos la clase que realizará la acción por ejemplo *com.hp.ov.activator.mwfm.component.builtin.Add*.

Ambos, *Task* y *CallActivity* extienden de *Activity* por lo que heredan todas sus atributos y restricciones que podemos ver aquí:

```

1 <xsd:element name="activity" type="tActivity"/>
2 <xsd:complexType name="tActivity" abstract="true">
3   <xsd:complexContent>
4     <xsd:extension base="tFlowNode">
5       <xsd:sequence>
6         <xsd:element ref="ioSpecification" minOccurs="0"
  maxOccurs="1"/>
7         <xsd:element ref="property" minOccurs="0" maxOccurs="
  unbounded"/>
8         <xsd:element ref="dataInputAssociation" minOccurs="0"
  maxOccurs="unbounded"/>
9         <xsd:element ref="dataOutputAssociation" minOccurs="0"
  maxOccurs="unbounded"/>
10        <xsd:element ref="resourceRole" minOccurs="0"
  maxOccurs="unbounded"/>
11        <xsd:element ref="loopCharacteristics" minOccurs="0"/>
12      </xsd:sequence>
13      <xsd:attribute name="isForCompensation" type="xsd:boolean"
  default="false"/>

```

<sup>5</sup>Aunque se ha elegido la clase *Service Task* como nodo por defecto, disponemos de una gran cantidad de tipos diferentes de tareas que nos aportan mayor expresividad al flujo, por ejemplo si necesitamos incluir un *AskFor* típico de HPE podemos definir la tarea como de tipo *UserTask* lo que implica que dicha tarea será realizada por una persona

```

14     <xsd:attribute name="startQuantity" type="xsd:integer"
15       default="1"/>
16     <xsd:attribute name="completetionQuantity" type="xsd:
17       integer" default="1"/>
18     <xsd:attribute name="default" type="xsd:IDREF" use="
19       optional"/>
17   </xsd:extension>
18 </xsd:complexContent>
19 </xsd:complexType>

```

Código 5: Activity

Y *Activity* también hereda de *Flow Node* así como *Flow Node* hereda de *Flow Element* heredando a su vez de *Base Element*, todas estas descripciones están en el Anexo II: Estructuras mencionadas de BPMN.

Al contrario que en la implementación de HPE, la secuencia de los nodos en los procesos modelados según la notación BPMN no reside en el nodo, si no fuera, en otra capa, la cual está definida en los elementos de tipo *Flow Node*.

Si miramos de cerca la estructura de *Flow Node*, veremos que tiene dos elementos, *incoming* y *outgoing*, estos elementos identifican el orden que tiene cada nodo en el flujo, definiendo los posibles nodos anteriores así como los nodos posteriores. Esta capa es muy útil cuando estamos leyendo el flujo para hacernos una idea de en que punto estamos de dicho flujo y hacia donde podemos ir.

Sobre los roles que podemos tener en los nodos, como ya comentamos anteriormente, hemos modelado dicha funcionalidad como *Resource* para poder asignar cualquier rol en cualquier nodo en cualquier punto del workflow. Para ejemplificar mejor este detalle veamos su estructura:

Primero definimos el recurso rol en nuestro flujo.

```

1 <bpmn:resource id="node_role" name="Node Rol">
2   <bpmn:resourceParameter id="Rol" isRequired="false" name="Node
3     Rol" type="xsd:string"/>
4 </bpmn:resource>

```

Código 6: Ejemplo de resource

Acto seguido, en el contexto de un nodo, ya podemos referenciar el recursos que acabamos de crear y fijar su valor:

```

1 <bpmn:resourceRole>
2   <bpmn:resourceRef>node_role</bpmn:resourceRef>
3   <bpmn:resourceParameterBinding parameterRef="Rol">
4     <bpmn:formalExpression>engineer</bpmn:formalExpression>
5   </bpmn:resourceParameterBinding>
6 </bpmn:resourceRole>

```

Código 7: Ejemplo de referencia a un recurso



Como se puede ver, la primera clase a la que se llama *Resource Role*, hereda de *Activity* y hace referencia a un recurso con la finalidad de definir su valor a ingeniero.

Para traducir los atributos *Inactive* o *DisablePersistence*, hemos usado categorías (*Category*) para con la finalidad de poder filtrar los nodos por dichas categorías una vez importado el flujo a un IDE como *Bonita*.

Las categorías son de la siguiente forma:

```

1 <xsd:element name="category" type="tCategory"
  substitutionGroup="rootElement"/>
2 <xsd:complexType name="tCategory">
3   <xsd:complexContent>
4     <xsd:extension base="tRootElement">
5       <xsd:sequence>
6         <xsd:element ref="categoryValue" minOccurs="0"
9         maxOccurs="unbounded"/>
7       </xsd:sequence>
8       <xsd:attribute name="name" type="xsd:string"/>
9     </xsd:extension>
10   </xsd:complexContent>
11 </xsd:complexType>

```

Código 8: Category

Y tienen un elemento llamado *categoryValue* que se define mediante el siguiente esquema:

```

1 <xsd:element name="categoryValue" type="tCategoryValue"/>
2 <xsd:complexType name="tCategoryValue">
3   <xsd:complexContent>
4     <xsd:extension base="tBaseElement">
5       <xsd:attribute name="value" type="xsd:string" use="
6       optional"/>
7     </xsd:extension>
8   </xsd:complexContent>
9 </xsd:complexType>

```

Código 9: Category Value

Por tanto, si queremos definir cualquiera de esos dos atributos, primero definiríamos la categoría ( muy similar a lo realizado para los roles):

```

1 <bpmn:category name="disablePersistence">
2   <bpmn:categoryValue value="true" id="disablePersistence.true
3   "> </bpmn:categoryValue>
4   <bpmn:categoryValue value="false" id="disablePersistence.
5   false"> </bpmn:categoryValue>
6 </bpmn:category>

```

Código 10: Creating the category

Y una vez en la definición de nuestro nodo tan solo nos bastaría con hacerla referencia.

```
1 <bpmn:categoryValueRef>disablePersistence.true
2 </bpmn:categoryValueRef>
```

Código 11: Referencing category

Ahora que ya tenemos algo más claro que entidades nos permiten representar cada uno de los puntos que definen un nodo proceso, ya podemos realizar su correspondiente tabla a modo de resumen:

Process Node	BPMN	Situación (XSD)
Name	Name (línea nº10)	Flow Element
Description <sup>6</sup>	Documentation (línea nº4)	Base Element
Next-Node	TargetRef (línea nº9)	Sequence Flow
State	State	Activity <sup>7</sup>
Role	Resource Role	Resource Role
Action	See next section	-
Inactive (Attribute)	Category	Category
DisablePersistence (Attribute)	Category	Category

Tabla 20: Process Node a BPMN

### 5.1.1. Acciones (Como extender la funcionalidad de BPMN)

Dado que una acción tiene más de un campo, hemos tenido que separar dicha funcionalidad en dos partes, el nombre de la clase y los parámetros que necesita la clase para poder ejecutarse.

El nombre de la clase le podemos definir en el atributo *Implementation* de *ServiceTask* o *User Task*, pero si queremos definir sus parámetros necesitamos extender la funcionalidad del estándar BPMN.

Para extender la funcionalidad de esta notación, el propio estándar provee una estructura especial para estos casos llamada *Extension Elements* que heredan de *Base Element*

Gracias a estructura podemos agregar cualquier lógica a mayores y seguir pudiendo validar correctamente el workflow en la notación BPMN.

Durante el transcurso de este proyecto utilizaremos estos elementos para implementar cuatro funcionalidades diferentes:

<sup>6</sup>Si queremos una descripción de todo el flujo podemos generar una interfaz del mismo e incluirla en *supportedInterfaceRefs* (*CallableElement*)

<sup>7</sup>Estos valores están disponibles únicamente en tiempo de ejecución [2, p. 153]

- Para definir los parámetros que necesitamos en los elementos *Action* de cualquier tarea.
- Para implementar las variables que se incluyen en el *CasePacket*.
- Para inicializar las variables existentes en el `\{ }textit{CasePacket}` .
- Para añadir atributos a un flujo de workflow.

```

1 <xsd:element name="extensionElements" type="tExtensionElements
  "/>
2 <xsd:complexType name="tExtensionElements">
3   <xsd:sequence>
4     <xsd:any namespace="##any" processContents="lax"
      minOccurs="0" maxOccurs="unbounded"/>
5   </xsd:sequence>
6 </xsd:complexType>

```

Código 12: Extension Elements

Para ilustrar un poco más en detalle su funcionamiento veamos un ejemplo:

Lo primero será crear nuestro propio xsd con nuestro propio espacio de nombres, para este proyecto lo hemos llamado "extension\_HPE" y su esquema XML es el siguiente:

```

1 <xsd:element name="actionParams" type="hpe:tActionParams"/>
2 <xsd:element name="param" type="hpe:tParam"/>
3
4 <xsd:complexType name="tActionParams">
5   <xsd:sequence>
6     <xsd:element ref="hpe:param" minOccurs="0" maxOccurs="
      unbounded"/>
7   </xsd:sequence>
8 </xsd:complexType>
9 <xsd:complexType name="tParam">
10   <xsd:attribute name="name" type="xsd:string"/>
11   <xsd:attribute name="value" type="xsd:string"/>
12 </xsd:complexType>

```

Código 13: Action Params Schema

Nota: Este esquema forma parte de otro esquema mucho más grande que puedes ver en: *HPE extension Schema*.

Una vez que hemos definido la estructura que tendrán nuestros datos que conformarán la ampliación, tan solo tenemos que incluirlos dentro del contenedor *Extension Elements*. Veamos un ejemplo:

```

1 <bpmn:extensionElements>
2   <hpe:actionParams>
3     <hpe:param name="queue" value="operator_input"/>
4     <hpe:param name="variable0" value="number_1"/>
5     <hpe:param name="variable1" value="number_2"/>
6     <hpe:param name="required1" value="true"/>
7     <hpe:param name="response" value="Thanks for insert the
  numbers"/>
8   </hpe:actionParams>
9 </bpmn:extensionElements>

```

Código 14: Action Params Schema

Si llegados a este punto quisieras validar el workflow, bastaría con crear un contenedor que incluyese todos los esquemas (tanto los propios como los de BPMN), en este proyecto dicho contenedor se ha llamado "BPMN\_HPE.xsd"

Para finalizar resumimos las correspondencias en la tradicional tabla:

Action	BPMN	Situación (XSD)
Class-Name	Implementation (línea nº5)	ServiceTask
Param name	tParam(línea nº19)	HPE extension Schema
Param value	tParam(línea nº20)	HPE extension Schema

Tabla 21: Action a BPMN

## 5.2. Rule/Switch Node

Para implementar estos nodos usaremos las conocidas *gateways* que según la notación BPMN se asemejarían más a evento que a nodos propiamente dichos.

```

1 <!--ELEMENT True-Next-Node (#PCDATA)>
2 <!--ELEMENT False-Next-Node (#PCDATA)>
3 <!--ELEMENT Rule-Node (Name, Description
  ?, State?, Action, True-Next-Node,
  False-Next-Node)>
4 <!--ATTLIST Rule-Node
  disablePersistence CDATA #IMPLIED>

```

Código 15: Rule Node

```

1 <!--ELEMENT Default (#PCDATA)>
2 <!--ELEMENT Switch (#PCDATA)>
3 <!--ATTLIST Switch name CDATA #IMPLIED>
4 <!--ELEMENT Switch-Node (Name,Description
  ?, State?, Action, Switch*, Default)>
5 <!--ATTLIST Switch-Node
  disablePersistence CDATA #IMPLIED>

```

Código 16: Switch Node

Ambos pueden ser modeladas usando puertas exclusivas, las cuales heredan de la clase *Gateway*

```

1 <xsd:element name="exclusiveGateway" type="tExclusiveGateway"
  substitutionGroup="flowElement"/>
2 <xsd:complexType name="tExclusiveGateway">
3   <xsd:complexContent>
4     <xsd:extension base="tGateway">
5       <xsd:attribute name="default" type="xsd:IDREF" use="
  optional"/>
6     </xsd:extension>
7   </xsd:complexContent>
8 </xsd:complexType>

```

Código 17: ExclusiveGateway

```

1 <bpmn:sequenceFlow id="sf_4137e1f4" targetRef="egR_ea5d80d6"
  sourceRef="egR_42532d83">
2   <bpmn:conditionExpression id="cp_7856c1d2" xsi:type="bpmn:
  tFormalExpression" evaluatesToTypeRef="java:java.lang.Boolean
  " language="com.hp.ov.activator.mwfm.component.builtin.
  LowerThan">True
3   </bpmn:conditionExpression>
4 </bpmn:sequenceFlow>
5 <bpmn:sequenceFlow id="sf_538fc791" targetRef="st_1e745e2a"
  sourceRef="egR_42532d83">
6   <bpmn:conditionExpression id="cp_f6cc4f8a" xsi:type="bpmn:
  tFormalExpression" evaluatesToTypeRef="java:java.lang.Boolean
  " language="com.hp.ov.activator.mwfm.component.builtin.
  LowerThan">False
7   </bpmn:conditionExpression>
8 </bpmn:sequenceFlow>

```

Código 18: Ejemplo de Sequence Flows

En BPMN después de definir cualquier elemento en un proceso, (evento,puerta de enlace o un nodo), hay que definir uno o varios elementos de tipo *Sequence*

*Flow* que actúan como tokens a la hora de relacionar un nodo con otro para realizar el camino que puede tomar un flujo del workflow. En nuestro caso, después de cada *Gateway* tan solo tenemos que definir un elemento por comprobación.

Indicar que el atributo *gatewayDirection* definido en *Gateway* nos permite modelar el tipo de flujo que queremos, por ejemplo, si necesitamos que converjan diferentes flujos, asignaremos a dicho atributo el valor "Converging", y automáticamente podremos definir dos o más flujos de entrada y un único flujo de salida.

```

1 <bpmn:exclusiveGateway id="egR_42532d83" name="
  GreaterOrLowerThan20Decision" gatewayDirection="Diverging">
2   <bpmn:extensionElements>
3     <hpe:actionParams>
4       <hpe:param name="op1" value="number_1"/>
5       <hpe:param name="op2" value="constant:20"/>
6     </hpe:actionParams>
7   </bpmn:extensionElements>
8   <bpmn:categoryValueRef>disablePersistence.true</bpmn:
  categoryValueRef>
9   <bpmn:incoming>sf_7dfb7d6e</bpmn:incoming>
10  <bpmn:outgoing>sf_4137e1f4</bpmn:outgoing>
11  <bpmn:outgoing>sf_538fc791</bpmn:outgoing>
12 </bpmn:exclusiveGateway>

```

Código 19: Ejemplo de gateway (XML)

Algo similar pasa con el atributo *ConditionExpression*, si una puerta tiene dos o más flujos de salida y dichos *sequenceFlows* tienen una *conditonExpression*, antes de seguir ningún camino, el procesador BPMN evalúa cada una de las condiciones para elegir el flujo correcto. A mayores podemos definir un camino por defecto que tomará únicamente en el caso en el caso que sea imposible tomar algún otro camino.

Rule/Switch Node	BPMN	Situación (XSD)
Name	Name (línea nº10)	Flow Element
Description	Documentation (línea nº4)	Base Element
State	A implementar	- <sup>8</sup>
Action (Class-Name)	Condition Expression (línea nº6)	Sequence Flow
True/False-Next-Node	TargetRef (línea nº9) <sup>9</sup>	Sequence Flow
Default	Default (línea nº5)	ExclusiveGateway
DisablePersistence (Attribute)	Category	Category

Tabla 22: Rule and Switch node to BPMN

<sup>8</sup>Al contrario que las tareas, que pueden heredar el estado de una actividad, las puertas no disponen de ese estado ya que únicamente juntan o separan flujos de trabajo.

<sup>9</sup>Junto con una *Conditional Expression*

### 5.3. Case-Packet e Initial-Case-Packet

#### 5.3.1. Case-Packet

El conjunto de variables denominado *Case-Packet* pueden ser implementados mediante la estructura *Data Object*, pero a mayores tendremos que definir el tipo de datos que queremos usar.

```

1 <!--ELEMENT Variable (#PCDATA)>
2 <!--ATTLIST Variable name CDATA #REQUIRED>
3 <!--ATTLIST Variable type CDATA #REQUIRED>
4 <!--ELEMENT Case-Packet (Variable*)>
5 <!--ATTLIST Variable disablePersistence CDATA #IMPLIED>

```

Código 20: Case-Packet

Hemos elegido *Data Object* sobre todo porque *Data Object* es la estructura principal para modelar datos en un workflow y su ciclo de vida en el flujo de trabajo nos da la flexibilidad que necesitamos.

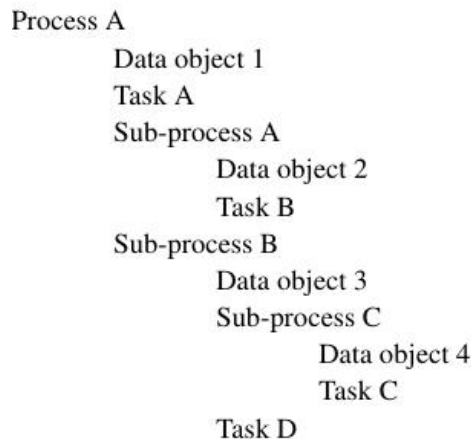


Figura 6: Visibilidad de un *Data Object*

Como podemos ver en la figura anterior, si definimos un *Data Object* en el primer proceso, el mismo objeto de datos será visible para todas y cada unas de las actividades y subprocesos que se pertenezcan a dicho proceso, en cambio si definimos ese objeto en un subproceso, todos las actividades que no pertenezcan a ese subproceso no podrán ver el objeto. Por ejemplo. "Data object 1 " podrá ser accedido por "Process A", "Task A", "Sub-Process A "," Task B"," Sub-Process B"," Sub-process C", "Task C " y " Task D", pero no podrá ser accedido por "Sub-Process B", "Sub-Process C", "Task C", y "Task D".

Las variables del *Case-Packet* se declaran al principio del flujo de trabajo y perduran durante todo el flujo (aunque sus valores puedan variar), por lo que su

definición residirá junto a la propia definición del proceso. En la línea once dentro del xsd que define a un *Process* podemos ver un elemento de tipo *Flow Element* que nos permite declarar el *Data Object*; esto es posible ya que *Data Object* hereda de *Flow Element* como podemos ver en la estructura siguiente:

```

1  <xsd:element name="dataObject" type="tDataObject" />
2  <xsd:complexType name="tDataObject">
3    <xsd:complexContent>
4      <xsd:extension base="tFlowElement">
5        <xsd:sequence>
6          <xsd:element ref="dataState" minOccurs="0" maxOccurs="1"
7          />
8        </xsd:sequence>
9        <xsd:attribute name="itemSubjectRef" type="xsd:QName"/>
10       <xsd:attribute name="isCollection" type="xsd:boolean"/>
11     </xsd:extension>
12   </xsd:complexContent>
13 </xsd:complexType>

```

Código 21: Data Object

En primer lugar, antes de poblar de información dichos objetos necesitamos definir el tipo de información que incluirán; para ello, nos apoyaremos en *Item-Definition*:

```

1  <xsd:element name="itemDefinition" type="tItemDefinition"
2    substitutionGroup="rootElement"/>
3  <xsd:complexType name="tItemDefinition">
4    <xsd:complexContent>
5      <xsd:extension base="tRootElement">
6        <xsd:attribute name="structureRef" type="xsd:QName"/>
7        <xsd:attribute name="isCollection" type="xsd:boolean"
8        default="false"/>
9        <xsd:attribute name="itemKind" type="tItemKind" default=
10        "Information"/>
11      </xsd:extension>
12    </xsd:complexContent>
13  </xsd:complexType>
14  <xsd:simpleType name="tItemKind">
15    <xsd:restriction base="xsd:string">
16      <xsd:enumeration value="Information"/>
17      <xsd:enumeration value="Physical"/>
18    </xsd:restriction>
19  </xsd:simpleType>

```

Código 22: ItemDefinition

En la línea nº5, se define una atributo llamado *structureRef* que acepta valores del tipo *xsd:QName* o en otras palabras cualquier tipo de URI, por ejemplo la



definición de nuestra estructura *Case-Packet* que podéis encontrar a partir de la línea veintiuno en *HPE extension Schema*

Una vez definido el tipo de datos que va a albergar nuestro *Data Object* y correctamente referenciada en el mismo, este debería tener un aspecto similar a siguiente:

```

1  <bpmn:dataObject id="do_d68362be" name="casePacket"
2    isCollection="false" itemSubjectRef="id_38d3a61f">
3    <bpmn:documentation>CasePacketVariables</bpmn:documentation>
4    <bpmn:extensionElements>
5      <hpe:casePacket>
6        <hpe:variable name="Cancel" type="Boolean"/>
7        <hpe:variable name="Finish" type="Boolean"/>
8        <hpe:variable name="module_name" type="String"/>
9      </hpe:casePacket>
10   </bpmn:extensionElements>
11 </bpmn:dataObject>

```

Código 23: Ejemplo DataObject

*ItemSubjectRef* nos permite relacionar el *Data Object* con la definición del objeto que estamos guardando en nuestro caso referenciará al *ItemDefinition* donde se encuentra nuestro *hpe:casePacket*

Si queremos usar este tipo de objetos en un flujo de trabajo debemos definir diferentes asociaciones como *dataObjectReference*, las cuales nos permiten relacionar aspectos como el estado del propio objeto con el estado del workflow. Pero ya que la finalidad de este proyecto es únicamente desarrollar una nueva forma de almacenamiento, se ha pospuesto su desarrollo para revisiones posteriores.

Case-Packet	BPMN	Situación (XSD)
Variable-Name	Name (línea nº30)	HPE extension Schema
Variable-Type	Type (línea nº31)	HPE extension Schema
DisablePersistence (Attribute)	DisablePersistence(line nº32)	HPE extension Schema

Tabla 23: Case-Packet a BPMN

### 5.3.2. Initial-Case-Packet

El caso del *Initial-Case-Packet*, se parece al anterior, pero es tratado de manera diferente. Mientras el *Case-Packet* viene definido por un *Data Object*, para el valor inicial que tomarán dichas variables necesitamos la interfaz que nos aporta *DataInput* ya que nos permite definir el valor inicial de las variables del *Case-Packet* antes de que ningún nodo se haya ejecutado.

Para usar los *dataInputs* primero tenemos que declarar el entorno *IoSpecification*, requisito necesario en la lógica BPMN que permite definir un *InputSet* y un *OutputSet* posterior. El *InputSet* en nuestro caso consistirá en un *DataInput* donde albergaremos el *Initial-Case-Packet* mientras que el *OutputSet* será una simple formalidad sin información importante asociada.

Como dato interesante y como veréis en el siguiente ejemplo, ambas estructuras *Case-Packet* y *Initial-Case-Packet* son referenciadas por el mismo *ItemDefinition*, aunque pueda parecer un error, esto es correcto ya que aunque se definan en diferentes puntos del workflow y tengan estructuras diferentes, representan el mismo conjunto de variables.

```

1  <bpmn:ioSpecification id="io_30e87ed1">
2    <bpmn:dataInput id="di_e4abf4de" itemSubjectRef="id_38d3a61f
3      ">
4      <bpmn:extensionElements>
5        <hpe:initial_casePacket>
6          <hpe:variable_value name="Cancel" value="true"/>
7          <hpe:variable_value name="Finish" value="false"/>
8        </hpe:initial_casePacket>
9      </bpmn:extensionElements>
10    </bpmn:dataInput>
11    <bpmn:inputSet id="is_b4532ca8">
12      <bpmn:dataInputRefs>di_e4abf4de</bpmn:dataInputRefs>
13    </bpmn:inputSet>
14    <bpmn:outputSet id="os_890f7b8d"/>
15  </bpmn:ioSpecification>

```

Código 24: Ejemplo DataInput

Initial-Case-Packet	BPMN	Situacion (XSD)
Variable-Value name	Name (línea nº42)	HPE extension Schema
Variable-Value value	Type (línea nº43)	HPE extension Schema

Tabla 24: Initial-Case-Packet a BPMN

## 5.4. Handlers

Para intentar traducir los disparadores de HPE, necesitaremos usar los dos tipos de eventos que nos ofrece BPMN, los disparadores y los receptores. Veamos primero como es un disparador en el estándar de HPE:

```
1 <!ELEMENT End-Handler (Name?, Class-Name, Param*)>
2 <!ELEMENT Error-Handler (Name?, Class-Name, Param*)>
```

Código 25: Handlers

Ambos, tanto los disparadores de errores como el evento fin de flujo serán implementados por eventos, concretamente *End-Handler* es un tipo de evento *ThrowEvent*<sup>10</sup> cuya estructura es la siguiente:

```
1 <xsd:element name="throwEvent" type="tThrowEvent"/>
2 <xsd:complexType name="tThrowEvent" abstract="true">
3   <xsd:complexContent>
4     <xsd:extension base="tEvent">
5       <xsd:sequence>
6         <xsd:element ref="dataInput" minOccurs="0" maxOccurs="unbounded"/>
7         <xsd:element ref="dataInputAssociation" minOccurs="0" maxOccurs="unbounded"/>
8         <xsd:element ref="inputSet" minOccurs="0" maxOccurs="1"/>
9         <xsd:element ref="eventDefinition" minOccurs="0" maxOccurs="unbounded"/>
10        <xsd:element name="eventDefinitionRef" type="xsd:QName" minOccurs="0" maxOccurs="unbounded"/>
11      </xsd:sequence>
12    </xsd:extension>
13  </xsd:complexContent>
14 </xsd:complexType>
```

Código 26: ThrowEvent

Existen nueve tipos diferentes de *End-Events*, pero en nuestro caso tan solo necesitaremos dos, eso sí, en dos sabores cada uno.

En primer lugar necesitamos un evento para comunicar al procesador BPMN que algo ha ocurrido, esto puede ser notificado de dos formas:

1. Cada nodo envía su propio evento ( tan solo válido para implementar el *Error-Handler*).
2. Se define un evento común para todos los nodos y que seguirán ( a modo de camino) siempre y cuando se produzca cualquier anomalía en el flujo.

<sup>10</sup>Eventos que siempre envían un resultado en contraposición de *Catch Event* quienes se activan al recoger un resultado enviado (casi siempre) por un *ThrowEvent*

Si elegimos la primera opción necesitaremos declarar un *Boundary Event* para cada nodo que pueda fallar. Esto generaría un gran número de líneas de código innecesarias si lo único que queremos es almacenar información, no procesar un flujo de trabajo directamente.

Por contra, si optamos por la segunda opción, podremos representar todos esos caminos una única vez y reutilizarlos durante todo el flujo lo cual es mucho más intuitivo y necesita muchas menos líneas de código que la primera opción, aunque tiene por contra que no todos los nodos la soportan.

Se ha elegido la segunda opción ya que en mi opinión, actualmente resuelve la finalidad del proyecto sin añadirle una mayor complejidad innecesaria y permite que, si en el futuro necesitáramos añadir las funcionalidades que nos da *Boundary Event* tan solo tendríamos que añadir unas pocas líneas por nodo para dar soporte a los eventos de error individuales.

Volviendo a la implementación, una vez que se tiene un evento que informa al workflow de que ha ocurrido algo, tan solo es necesario otro evento para recoger dicha información y empezar el camino alternativo.

Para contemplar los casos en los que un workflow puede finalizar, ya sea porque bien finaliza el workflow o porque finaliza tras lanzar un error, usaremos dos tipos diferentes de eventos finales:

- ***errorEventDefinition***, quien envía un mensaje de error con intención de que sea recogido por un receptor

```
1 <xsd:element name="errorEventDefinition" type="
  tErrorEventDefinition" substitutionGroup="eventDefinition"
  />
2 <xsd:complexType name="tErrorEventDefinition">
3   <xsd:complexContent>
4     <xsd:extension base="tEventDefinition">
5       <xsd:attribute name="errorRef" type="xsd:QName"/>
6     </xsd:extension>
7   </xsd:complexContent>
8 </xsd:complexType>
```

Código 27: Error Event Definition

- ***messageEventDefinition***, envía un mensaje genérico que será recogido por un receptor de mensajes.

```
1 <xsd:element name="messageEventDefinition" type="
  tMessageEventDefinition" substitutionGroup="
  eventDefinition"/>
2 <xsd:complexType name="tMessageEventDefinition">
3   <xsd:complexContent>
4     <xsd:extension base="tEventDefinition">
5       <xsd:sequence>
```

```

6      <xsd:element name="operationRef" type="xsd:QName"
      minOccurs="0" maxOccurs="1"/>
7      </xsd:sequence>
8      <xsd:attribute name="messageRef" type="xsd:QName"/>
9      </xsd:extension>
10     </xsd:complexContent>
11 </xsd:complexType>
12

```

Código 28: MessageEventDefinition

El encargado de recoger este tipo de eventos esperará durante todo la ejecución del proceso hasta que encuentre que alguien haya lanzado un evento con ese mensaje. Dicho mensaje se puede definir por medio de *ItemDefinition* de manera similar a la empleada en el *Case-Packet*

Recordemos que ambos tipos de eventos siguen siendo eventos finales, únicamente cambia su definición (indicada anteriormente) por lo que ejecutarán la misma acción, finalizar un proceso, solo que para realizar dicha finalización serán recorridos caminos diferentes dependiendo de quien recoja el evento (es decir, el receptor cuya definición sea la misma que el emisor).

Ya que el *Error-Handler* de HPE siempre termina con la ejecución del workflow, el evento fin que finaliza el flujo de trabajo se llamará siempre.

Para implementar la lógica de estos handlers, se han usado subprocesos, uno por cada tipo de disparador ( error y fin). Un subproceso se comporta igual que un workflow, tiene un evento de inicio, una serie de nodos que ejecutan tareas (en nuestro caso un *ServiceTask*) y un nodo fin que concluye el subproceso.

En nuestro caso los eventos de inicio corresponderían con los receptores de los eventos anteriores y los dos eventos fin corresponderían a las dos definiciones diferentes que tendríamos de cada evento fin anterior.

Handlers	BPMN	Situación (XSD)
Name	Name (line nº10)	Flow Element
Class-Name	Implementation (line nº5)	ServiceTask
Param	tParam	HPE extension Schema

Tabla 25: Handlers a BPMN

## 5.5. Workflow

Tras definir todos los elementos que componen un flujo de trabajo ya podemos ver como juntarles para así crear nuestro primer workflow HPE sobre el estandar BPMN.

```

1  <!--ELEMENT Workflow (Name,Solution?, Description?, Default-Role
    ?, Start-Role?, Trace-Role?, Kill-Role?,Start-Node, Nodes,
    Error-Handler*, End-Handler*, Case-Packet,
    Initial-Case-Packet?, Coordinates?)-->
2  <!--ATTLIST Workflow Init-On-Startup CDATA #IMPLIED>
3  <!--ATTLIST Workflow Unique CDATA #IMPLIED>
4  <!--ATTLIST Workflow statEnabled CDATA #IMPLIED>
5  <!--ATTLIST Workflow auditEnabled CDATA #IMPLIED>
6  <!--ATTLIST Workflow autoAuditEnabled CDATA #IMPLIED>
7  <!--ATTLIST Workflow disablePersistence CDATA #IMPLIED>
8  <!--ATTLIST Workflow maxNodesPerThread CDATA #IMPLIED>
9  <!--ATTLIST Workflow auditStateChangeEvent CDATA #IMPLIED>

```

Código 29: Workflow

Ya que muchos de estos parámetros ya han sido explicados anteriormente, únicamente les recordaremos en la tabla resumen tradicional de estas secciones.

Nuestro primer problema a la hora de modelar un flujo de trabajo era incluir el campo *solution*, aunque podemos describir dicho campo utilizando la estructura *Interface* junto con *Operation* que nos permite definir la finalidad del flujo de trabajo actual, hemos preferido añadir dicho campo como un campo más en la documentación del propio flujo ya que, a parte de ser campos optativos y hacer mucho más legible el xml, no aportaría ninguna funcionalidad a mayores frente a esta forma. No obstante lo indicamos por si más adelante fuera necesario plasmar dicha información aplicando interfaces y operaciones.

Aunque el nodo de inicio en la implementación de HPE es un elemento especial, en los flujos BPMN es tan solo un evento inicial como cualquier otro (puede venir en diferentes versiones) y que tiene la siguiente estructura:

```

1  <xsd:element name="startEvent" type="tStartEvent"
    substitutionGroup="flowElement"/>
2  <xsd:complexType name="tStartEvent">
3    <xsd:complexContent>
4      <xsd:extension base="tCatchEvent">
5        <xsd:attribute name="isInterrupting" type="xsd:boolean"
        " default="true"/>
6      </xsd:extension>
7    </xsd:complexContent>
8  </xsd:complexType>

```

Código 30: Start Event

En los procesos BPMN, hemos definido por una parte la lógica de los nodos (actividades en el caso de BPMN ) y por otra parte la posición que dichos nodos ocupan dentro del flujo. Es por esto, que una vez hayamos definido las actividades que queremos que tenga un workflow, únicamente necesitaremos unir estas actividades por medio de unas estructuras llamadas *Sequence Flow*.

Por último, para modelar los atributos se ha echado mano una vez más de los *Extension Elements* de BPMN, ya que la mayoría de atributos son atributos orientados únicamente a la lógica de HPE y que no tienen ninguna relación con el enfoque de BPMN. Sin embargo los atributos como *auditEnabled* y *auditStateChangeEvent* podrían ser relacionados con BPMN utilizando diferentes estructuras, pero ya que son atributos muy específicos los que podríamos modelar mediante estructuras ya predefinidas por BPMN habría que definir los diferentes atributos en diferentes posiciones del flujo, lo que dificultaría en gran medida su legibilidad y por consiguiente su mantenimiento, por tanto se decidió modelar todos los atributos utilizando la misma estructura anteriormente comentada.

```

1 <bpmn:extensionElements>
2   <hpe:workFlow_attributes>
3     <hpe:Unique>true</hpe:Unique>
4     <hpe:maxNodesPerThread>12</hpe:maxNodesPerThread>
5   </hpe:workFlow_attributes>
6 </bpmn:extensionElements>

```

Código 31: Atributos del workflow

Workflow	BPMN	Situación (XSD)
Name	Name (línea nº10)	Flow Element
Solution	Documentation (línea nº4)	Base Element
Description	Documentation (línea nº4)	Base Element
Roles	Resource Role	Resource Role
Start-Node	Start Event	Start Event
Nodes	Activities	Activity
Error-Handler	Error Event	Error Event Definition
End-Handler	End Event	MessageEventDefinition
Case-Packet	Data Object	Data Object
Attributes	tWf_attributes	HPE extension Schema

Tabla 26: Atributos de un workflow a BPMN

Para representar los nodos en un flujo BPMN gráficamente necesitamos, o bien definir las coordenadas donde se sitúan dichos nodos o definirlos en un conjunto de canales. Si optamos por la primera opción conseguiremos una representación más organizada que la segunda, pero por contra añadiremos mucha más información por ítem de la que podríamos desear, aunque si optamos por la segunda opción

conseguiremos algo intermedio, podremos representar el flujo gráficamente (como se puede ver en el *Anexo I: Ejemplo de un workflow según BPMN*) con tan solo unas pocas líneas que necesitaremos para definir el *laneSet* al que pertenece. Indicar que aunque parezca que mediante esta segunda opción disponemos de un flujo ordenado, es todo fachada es decir, de manera interna este flujo no tendrá orden alguno, por lo que si quisiéramos exportar un flujo, tendríamos que exportar todo el proyecto, no únicamente el xml donde se define.



## 5.6. Diagramas de despliegue

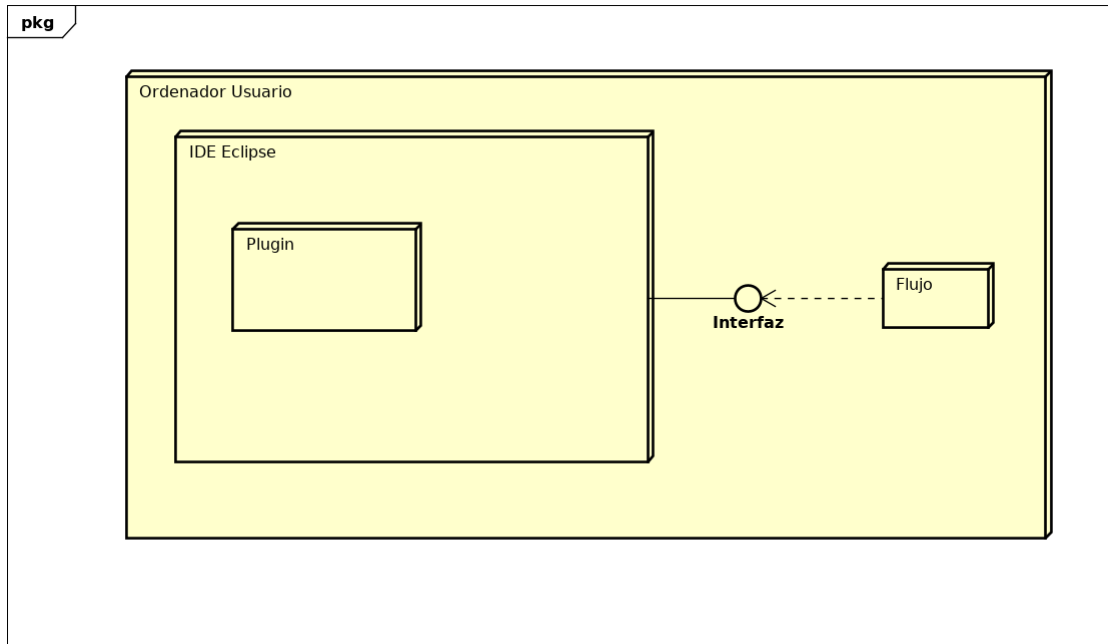


Figura 7: Diagrama de despliegue

En él podemos apreciar que todo el sistema corre en el ordenador del usuario, donde tiene que disponer de una versión instalada de *Eclipse* que, a su vez, tiene que tener instalado el plugin que actúa como traductor de flujos. Una vez que toda la plataforma está funcionando y a través de la interfaz de *Eclipse* un flujo cualquiera (HPE o BPMN) podrá comunicarse con el sistema para realizar la traducción pertinente.

### 5.7. Diagrama de componentes

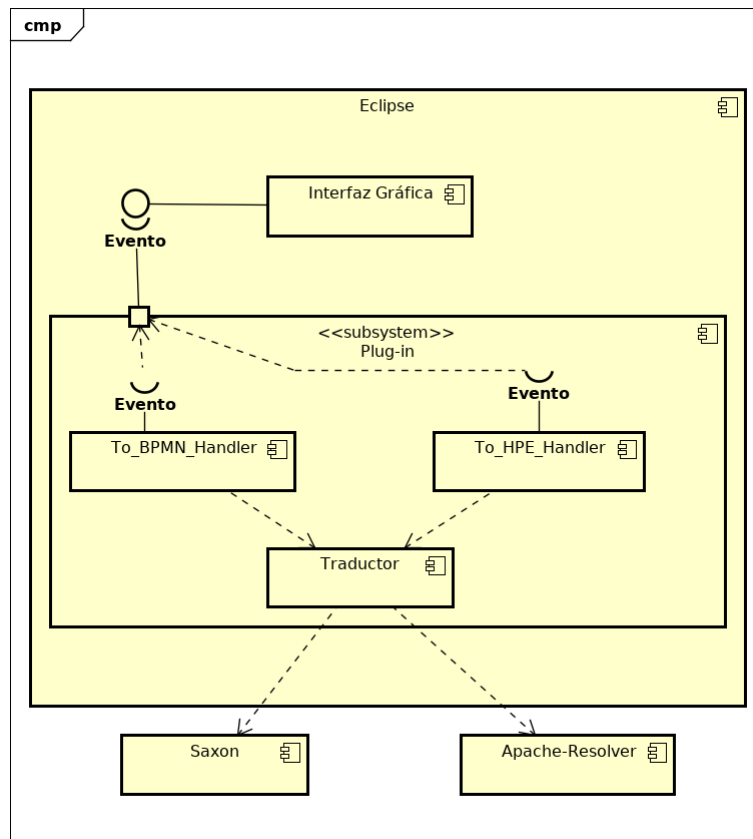


Figura 8: Diagrama de componentes

En este diagrama podemos entrever el funcionamiento interno del plugin comentado para *Eclipse*. Una vez instalado el plugin para el IDE, la interfaz gráfica generaría los eventos necesarios para alimentar al plugin (por ejemplo cuando el usuario selecciona un flujo para convertir en otra notación). Acto seguido dicho evento sería capturado por alguno de los Handlers, que dependiendo de su naturaleza, inicializaría los recursos pertinentes (obtendría por ejemplo los xslt necesarios para cada traducción) y realizaría la traducción..

Para realizar la traducción se hace uso de la librería saxón, la cual está incluida en el propio plugin y de la librería apache resolver, que también está incluida en el plugin.

## 5.8. Diagrama de clases

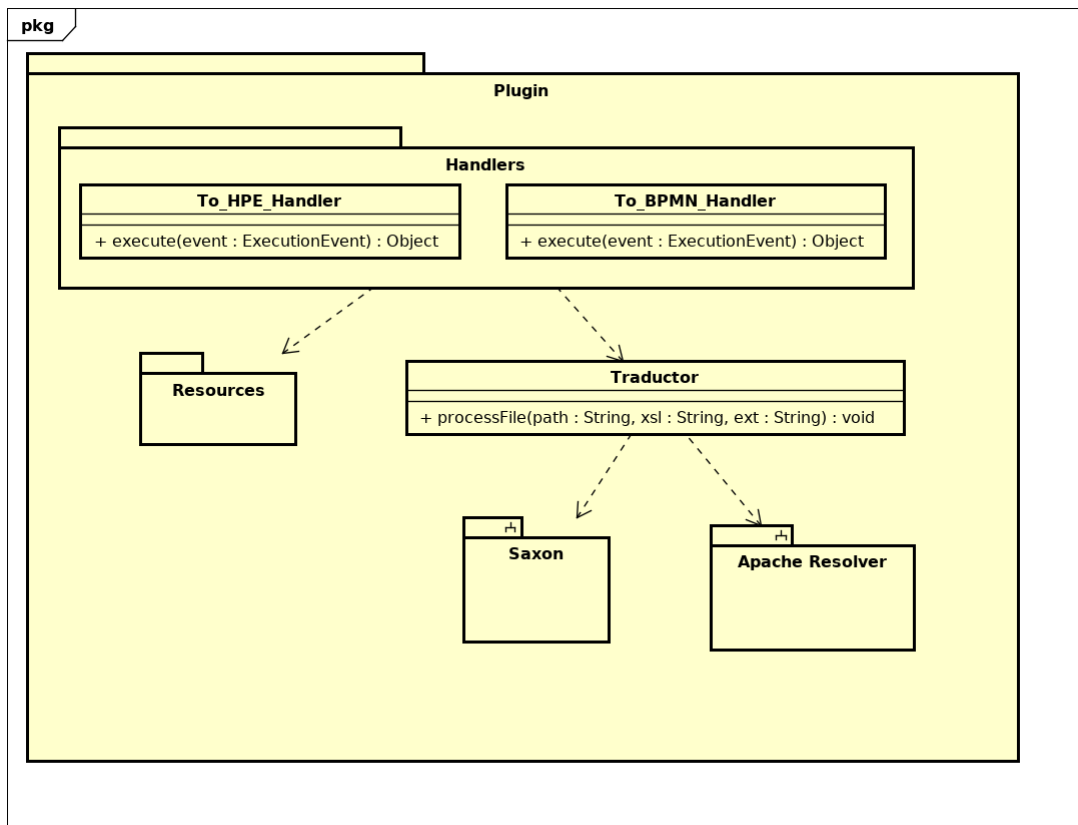


Figura 9: Diagrama de clases

Veamos la función de cada apartado:

- **Handlers**, Uno por cada tipo de traducción, la función de estos handlers es la de procesar cada evento según su naturaleza, es decir si el usuario necesitara traducir un flujo HPE a BPMN se ejecutaría el handler "To\_BPMN\_Handler", esta relación está especificada en el fichero *plugin.xml*. Cada uno de los handlers obtendrán los ficheros donde se encuentran los flujos a traducir, obtendrán el archivo XSLT necesario para su traducción y procederán a su traducción, finalizando con el envío de un mensaje informativo al usuario indicándole que se ha realizado la traducción.
- **Resources**, se trata de un repositorio donde se encuentra todo lo necesario para realizar la traducción, esto es, tanto los jars *Saxon* y *Apache Resolver*, los logos para la interfaz gráfica así como los archivos XSLT necesarios

que comentábamos anteriormente. En el anterior diagrama se indicó una dependencia hacia este repositorio únicamente como medio para indicar que ambos handlers dependen de los ficheros XSLT, ya que sin ellos, a la hora de llamar al traductor, este no realizaría ningún cambio en el flujo original.

- **Traductor**, es el encargado de leer el fichero original cedido por los handlers, ejecutar los cambios obtenidos del XSLT, y crear el nuevo flujo.
- **Saxón** Para aplicar los cambios necesarios en la traducción el traductor ejecuta el intérprete saxón, de ahí la necesidad de esta librería, incluida en el plugin como un jar.
- **Apache Resolver** previo a la ejecución del intérprete saxón, permite incluir los archivos DTD y realizar las importaciones de otros archivos XSLT en otros XSLT.

### 6. Implementación

Para implementar el conversor HPE -> BPMN y viceversa, hemos elegido la tecnología XSLT ya que ambos diseños se almacenan de la misma manera (mediante XMLs) y XSLT nos permite realizar transformaciones directas siempre y cuando los documentos de entrada sean XMLs. Otra posible implementación podría haber consistido en la generación de un conversor basado en perl (o en algún otro lenguaje de programación que esté enfocado en el desarrollo y uso de expresiones regulares) que leyera el origen y en base a una lógica preestablecida y expresiones regulares diera como resultado la otra definición del flujo, pero ya que existe una tecnología tan directa y enfocada a nuestro objetivo como es XSLT, preferimos obviar la segunda opción en pro el rendimiento y mantenibilidad que nos puede ofrecer la primera, es más, si implementamos el traductor por medio de XSLT únicamente tendremos que desarrollar y mantener dos plantillas, una destinada a convertir un fichero XML del estándar HPE a BPMN y otro para su viceversa.

El fichero xsl principal que traduce de BPMN a HPE se ha llamado "BPMN\_to\_Activator.xsl", que a su vez depende del fichero "functions\_bpm.xsl" que actúa como librería para el primero almacenando diversas funciones que irá llamando paulatinamente el archivo principal. Se optó por extraerlas en otro archivo ya que manteniendo las funciones más complejas en un archivo externo conseguimos que el archivo principal no quede tan largo y tedioso de mantener; a parte, si el día de mañana necesitáramos que esas funciones se reutilizaran para otro traductor, podríamos importarlas directamente de la misma forma que lo hemos realizado aquí.

"Activator\_to\_BPMN.xsl" traduce de HPE a BPMN pero al contrario que el anterior conversor, este no necesita de una librería, ya que las funciones que utiliza no son tan complejas ni largas como las utilizadas en el otro traductor, sin embargo necesita llamar a ciertas funciones de java que permiten generar los números aleatorios que necesitaremos para generar los ids de cada elemento del workflow.

Si queremos validar el resultado obtenido tras su ejecución necesitaremos algunos ficheros que podemos descargarnos de la documentación oficial de BPMN, pero que en nuestro caso podemos encontrar en la carpeta denominada "Desarrollado". Todos los \*.xsd que encontramos en esta carpeta son los ficheros desarrollados por BPMN y necesarios para la validación que comentamos, a mayores y dado que utilizamos varias estructuras que extienden la funcionalidad de BPMN también necesitamos los ficheros "HPE.xsd" y "BPMN\_HPExsd".

El archivo "HPE.xsd" es donde definimos las estructuras de los elementos que extendemos como por ejemplo las variables del *Case-Packet* mientras que en el fichero "BPMN\_HPExsd" actúa como contenedor de ambos ficheros BPMN y HPE, ya que al validar un XML únicamente podemos validarlo frente a un schema,

en este caso este fichero, que incluye a su vez el resto de schemas necesarios para su validación.

## 6.1. Librerías utilizadas y necesarias

Como comentábamos en el apartado de diseño, para la realización del plugin han sido necesarias la importación de dos librerías:

- **Saxon-B**, utilizado para interpretar los XSLT que guardan la lógica de la traducción. Se ha elegido esta versión ya que para hacer uso de funciones externas dentro de un xsl, por ejemplo las llamadas a las funciones de java que comentamos anteriormente para la generación de números aleatorios, saxon propone una versión de pago *saxonEE/PE* pero también ofrece otra versión *saxon-B* que no está tan actualizada como la de pago (9.1.0.8 frente a 9.7) pero que no requiere una licencia de pago a mayores. Esta versión puede descargarse del siguiente enlace:

<https://sourceforge.net/projects/saxon/files/Saxon-B/9.1.0.8/saxonb9-1-0-8j.zip/download>.

- **Apache Resolver**, necesario únicamente si vamos a ejecutar el traductor por medio del plugin para *Eclipse* ya que nos permite mapear los ficheros extra que necesitamos para realizar las traducciones como pueden ser los DTDs o las importaciones de archivos xsl dentro de otros xsls. Se puede descargar del siguiente enlace:

<http://xerces.apache.org/mirrors.cgi>.

- **Xmllint**, necesario únicamente si queremos validar los diferentes flujos que vayamos creando (HPE o BPMN), ya sea mediante DTDs o XSDs. Se puede descargar e instalar desde cualquier distribución/versión de linux/mac respectivamente sin problemas, no obstante si trabajásemos en un sistema Windows habría que seguir las indicaciones del siguiente enlace:

<https://stackoverflow.com/questions/19546854/installing-xmllint#21227833>.

## 6.2. Software utilizado

Durante todo el transcurso del proyecto se ha utilizado el siguiente software:

- **S.O:** Arch linux.
- **IDE:** Eclipse Neon.
- **Editor de texto:** Vim.
- **Gestor de versiones:** Git.
- **Herramientas ofimática:**  $\text{\LaTeX}$ .
- **Herramienta de modelado:** Astah.
- **Intérprete XSLT:** Saxon-B 9.1.0.8.
- **Validador XML:** Xmlint.
- **Lector de pdfs:** Okular.
- **Navegador Web:** Firefox.
- **Comparador de ficheros:** Meld.

## 7. Pruebas

A continuación veremos la batería de pruebas que se ha ido realizando en las distintas fases de la implementación, así como sus resultados.

Para la realización de las pruebas se creó un flujo de trabajo de ejemplo especialmente diseñado para estas, ya que contemplaba casi todos los casos que se pueden dar en un flujo comercial. Este flujo de ejemplo lo podéis encontrar en el Anexo I: Ejemplo de un workflow según BPMN

Primero expondremos las pruebas unitarias que realizamos para comprobar que cada componente que se iba desarrollando funcionara correctamente. Acto seguido pasaremos a ver las de integración, que aunque se realizaron paralelamente con las unitarias, las hemos agrupado todas en una sección para facilitar su organización. Por último, pasaremos a las pruebas del sistema que certificarán el correcto funcionamiento de la solución desarrollada.

Indicar que únicamente se expondrán las pruebas realizadas para un traductor, ya que las pruebas realizadas para el otro traductor son exactamente iguales (sin fallos notables a excepción de la prueba de sistema realizada con flujos reales que si mostraremos en dicha sección) y que por lo tanto no aportan nada nuevo al documento.

### 7.1. Pruebas unitarias

#### PU-01 Creación de un flujo HPE limpio

Descripción	Se procede a crear un flujo HPE válido únicamente con las cabeceras propias de HPE
Entrada	Un flujo válido BPMN
Resultado esperado	Obtención de un flujo HPE válido sin ningún nodo únicamente con las cabeceras propias de HPE
Resultado obtenido	Se crea correctamente un flujo limpio y válido
Estado	✓

Tabla 27: PU-01 Creación de un flujo HPE limpio



**PU-02 Conversión Process-Node**

Descripción	Se procede a traducir el nodo <i>Task</i> de BPMN a un <i>Process-Node</i> propio de HPE
Entrada	Un flujo válido BPMN con un nodo tipo <i>Task</i>
Resultado esperado	Obtención de un nodo tipo <i>Process-Node</i> en un nuevo flujo HPE
Resultado obtenido	Se traduce correctamente el nodo <i>Task</i>
Estado	✓

Tabla 28: PU-02 Conversión Process-Node

**PU-03 Adición de atributos a Process-Node**

Descripción	Se procede a añadir los atributos de un <i>Process-Node</i> desde un nodo tipo <i>Task</i>
Entrada	Un flujo válido BPMN con un nodo tipo <i>Task</i>
Resultado esperado	Obtención de un nodo tipo <i>Process-Node</i> en un nuevo flujo HPE con sus atributos asociados
Resultado obtenido	Se traduce correctamente el nodo <i>Task</i> con sus atributos asociados
Estado	✓

Tabla 29: PU-03 Adición de atributos a Process-Node

**PU-04 Traducción de Rule-Node con sus atributos correspondientes**

Descripción	Se procede a crear los nodos <i>Rule-Node</i> de sus <i>Gateways</i> correspondientes
Entrada	Un flujo válido BPMN con un nodo tipo <i>exclusiveGateway</i>
Resultado esperado	Obtención de un nodo tipo <i>Rule-Node</i> en un nuevo flujo HPE con sus atributos asociados
Resultado obtenido	Se traduce correctamente el nodo <i>exclusiveGateway</i> con sus atributos asociados pero se duplica el atributo <i>Next-node</i> cuando es el último nodo
Resultado corregido	Se traduce correctamente el nodo <i>exclusiveGateway</i> con sus atributos asociados
Estado	✓

Tabla 30: PU-04 Traducción de Rule-Node con sus atributos correspondientes

**PU-05 Traducción de los atributos de un workflow**

Descripción	Se procede a traducir los atributos relacionados con un workflow
Entrada	Un flujo válido BPMN con los atributos típicos de un workflow
Resultado esperado	Obtención de un flujo HPE con los atributos del flujo de trabajo correctamente traducidos
Resultado obtenido	Se traducen correctamente todos los atributos
Estado	✓

Tabla 31: PU-05 Traducción de los atributos de un workflow

**PU-06 Traducción de los Gateways tipo Switch con sus atributos correspondientes**

Descripción	Se procede a crear los nodos <i>Switch-Node</i> de sus <i>Gateways</i> correspondientes
Entrada	Un flujo válido BPMN con uno nodo tipo <i>exclusiveGateway</i>
Resultado esperado	Obtención de un nodo tipo <i>Switch-Node</i> en un nuevo flujo HPE con sus atributos asociados
Resultado obtenido	Se traduce correctamente el nodo <i>exclusiveGateway</i> con sus atributos asociados pero se duplican los nodos comunes tras converger de nuevo los caminos
Resultado corregido	Se traduce correctamente el nodo <i>exclusiveGateway</i> con sus atributos asociados sin duplicidad de nodos
Estado	✓

Tabla 32: PU-06 Traducción de los Gateways tipo Switch con sus atributos correspondientes

## 7.2. Pruebas de integración

### PI-01 Integración de un flujo HPE limpio con Process-Nodes

Descripción	Se procede a integrar un Process-Node a un flujo HPE limpio y comprobar que sigue validando el flujo
Entrada	Un flujo válido BPMN con nodos de tipo <i>Task</i>
Resultado esperado	Obtención de un flujo HPE válido con uno o más <i>Process-Node</i>
Resultado obtenido	Se crea correctamente un flujo válido
Estado	✓

Tabla 33: PI-01 Integración de un flujo HPE limpio con Process-Nodes

### PI-02 Traducción de un flujo BPMN con Gateways y Tasks

Descripción	Se procede traducir un flujo BPMN que contiene Gateways y Tasks
Entrada	Un flujo válido BPMN con nodos de tipo <i>Task</i> y <i>Exclusive-Gateway</i>
Resultado esperado	Obtención de un flujo HPE válido con uno o más <i>Process-Node</i> y <i>Rule-Nodes</i>
Resultado obtenido	Se crea correctamente un flujo válido
Estado	✓

Tabla 34: PI-02 Traducción de un flujo BPMN con Gateways y Tasks

### PI-03 Traducción de un flujo BPMN con Gateways, Tasks y atributos en un Workflow

Descripción	Se procede traducir un flujo BPMN que contiene Gateways y Tasks y atributos en un Workflow
Entrada	Un flujo válido BPMN con nodos de tipo <i>Task</i> y <i>Exclusive-Gateway</i> y atributos asociado a un flujo de trabajo
Resultado esperado	Obtención de un flujo HPE válido con todo traducido
Resultado obtenido	Se crea correctamente un flujo válido
Estado	✓

Tabla 35: PI-03 Traducción de un flujo BPMN con Gateways, Tasks y atributos en un Workflow

#### PI-04 Traducción de un flujo BPMN con Gateways, Tasks, Handlers y atributos en un Workflow

Descripción	Se procede traducir un flujo BPMN que contiene Gateways, Tasks, Handlers y atributos en un Workflow
Entrada	Un flujo válido BPMN con nodos de tipo <i>Task</i> , <i>Handlers</i> , <i>ExclusiveGateway</i> y atributos asociado a un flujo de trabajo
Resultado esperado	Obtención de un flujo HPE válido con todo traducido
Resultado obtenido	Se crea correctamente un flujo válido
Estado	✓

Tabla 36: PI-04 Traducción de un flujo BPMN con Gateways, Tasks, Handlers y atributos en un Workflow

#### PI-05 Traducción de un flujo BPMN completo y añadido el sistema de roles

Descripción	Se procede traducir un flujo BPMN completo al que se le ha añadido el sistema de roles
Entrada	Un flujo válido BPMN con el sistema de roles a mayores
Resultado esperado	Obtención de un flujo HPE válido con todo traducido
Resultado obtenido	Se crea correctamente un flujo con los nodos desordenados
Resultado corregido	Se traduce todo el flujo correctamente con los nodos ordenados, generando un flujo totalmente válido
Estado	✓

Tabla 37: PI-05 Traducción de un flujo BPMN completo y añadido el sistema de roles

#### PI-06 Intento de traducción de un flujo erróneo mediante un plug-in para Eclipse

Descripción	Se procede traducir un flujo erróneo mediante el plug-in desarrollado para eclipse
Entrada	Un flujo válido incorrecto
Resultado esperado	Obtención de un flujo vacío
Resultado obtenido	Se crea un archivo nuevo vacío con el nombre del flujo que se ha intentado convertir
Estado	✓

Tabla 38: PI-06 Intento de traducción de un flujo erróneo mediante un plug-in para Eclipse

### 7.3. Pruebas del sistema

En esta sección ya se realizaron pruebas con los flujos reales cedidos por HPE, *HP1\_activator.xml* y *HP2\_activator.xml*.

#### PS-01 Traducción de un flujo BPMN completo

Descripción	Se procede traducir un flujo BPMN completo y realizar su posterior validado
Entrada	Un flujo BPMN válido completo
Resultado esperado	Obtención de un flujo HPE válido con todo traducido
Resultado obtenido	Generado un flujo HPE válido con errores en los handlers
Resultado corregido	Generado un flujo HPE completo y validado
Estado	✓

Tabla 39: PS-01 Traducción de un flujo BPMN completo

#### PS-02 Traducción de un flujo HPE completo

Descripción	Se procede traducir un flujo HPE completo y realizar su posterior validado
Entrada	Un flujo HPE válido y completo
Resultado esperado	Obtención de un flujo BPMN válido con todo traducido
Resultado obtenido	Fallo al procesar el segundo switch (bucle infinito)
Resultado corregido	Generado un flujo BPMN completo y validado
Estado	✓

Tabla 40: PS-02 Traducción de un flujo HPE completo

#### PS-03 Traducción de un flujo cualquiera (HPE/BPMN) por medio del plug-in para Eclipse

Descripción	Se procede traducir un flujo HPE o BPMN completo mediante un plug-in desarrollado para <i>Eclipse</i>
Entrada	Un flujo HPE o BPMN válido y completo
Resultado esperado	Obtención de un flujo válido con todo traducido
Resultado obtenido	Se obtiene un flujo válido y traducido
Estado	✓

Tabla 41: PS-03 Traducción de un flujo cualquiera (HPE/BPMN) por medio del plug-in para Eclipse

## 8. Control y seguimiento

### 8.1. Objetivos conseguidos

Una vez finalizada la fase de implementación, si miramos a tras y recapitulamos los objetivos conseguido nos damos cuenta de que son muchos más los objetivos que no vemos a simple vista que los enumerados en un principio, pero que sin embargo resultan imprescindibles para lograr una calidad mínima al realizar los principales. Es el caso por ejemplo de una buena documentación, tanto en forma de un documento como en el propio código, pero este es solo un ejemplo dentro de muchos otros.

A continuación veamos algunos de estos objetivos que hemos intentado realizar:

1. Hemos conseguido traducir los flujos HPE a la notación BPMN, lo que nos permite añadir esa capa de abstracción que nos pedían como objetivo principal al principio del documento. También se ha implementado el traductor inverso, que consigue replicar un flujo HPE de su representación en BPMN.
2. Se ha conseguido desarrollar un plugin para el IDE *Eclipse* que permite realizar la traducción de los flujos cómodamente, directamente de la interfáz gráfica de *Eclipse*, facilitando su edición y posterior traducción desde la misma herramienta.
3. Se han desarrollado las estructuras XSD necesarias para la posibilitar la correcta validación de los flujos.
4. Se ha abierto la puerta a posibles ampliaciones e integraciones con el estándar BPMN, lo cual es un grán avance ya que existen numerosas herramientas que implementan dicho estándar enfocadas a la edición de flujos a un nivel mucho más complejo que el existente actualmente en HPE.
5. Se ha intentado documentar tanto el código como el proyecto en general de una forma lo más comprensible posible y en inglés, lo que facilita su mantenimiento por cualquier trabajador de HPE, ya sea de habla hispana o no.
6. Se ha intentado mantener en todo momento una política DRY en la implementación del código lo que facilita a su vez su futuro mantenimiento.
7. Al elegir la tecnología XSLT e intentar utilizarla mediante sus "buenas prácticas" (funciones recursivas, llamadas a funciones..) hemos podido obtener traducciones de tres mil líneas de código instantáneas, por lo que podemos decir que se ha conseguido optimizar bastante su implementación.

## 8.2. Problemas encontrados

Durante el transcurso del proyecto nos hemos encontrado con numerosos problemas. Los primeros aparecieron junto al estándar BPMN, que resultó ser bastante más complejo de lo esperado, sobre todo porque nunca había tenido que comprender un estándar tan afondo, y que encima su documentación únicamente radicaba en el documento facilitado por OMG donde se definía el propio estándar. Si a esto le unimos el problema de que no tenía realimentación exterior apenas, la sensación de incertidumbre era generalizada durante el transcurso del proyecto, motivada sobre todo por diversos rediseños que se sucedieron a lo largo del diseño final donde primero se pensaba que la responsabilidad de una estructura *x* era una y luego al intentar implementar dicha estructura resultaba ser otra. Un ejemplo de eso fue la estructura *case-packet*, que en un principio la veía viable implementarla mediante *DataObjects* pero que más tarde resultó ser inviable por el formato de dicha estructura.

Una vez con el diseño correcto y dispuestos a implementar el sistema, al crear el primer nodo vimos que le teníamos que asignar un id, y dicho id tenía que ser obviamente único, no solo en el flujo, si también en diferentes flujos ( con intención de en un futuro, editar los flujos mediante herramientas de modelado BPMN), por esta razón encontramos que la mejor forma de generar estos números aleatorios era la inclusión de funciones java en el propio XSL, hasta ahí correcto, el problema radicaba en el intérprete que utilizábamos para aplicar los cambios del XSL ya que dicha funcionalidad estaba disponible únicamente para versiones de pago. No obstante conseguimos una versión concreta que permitía disponer de dicha funcionalidad con una licencia de código abierto.

Otro problema al que nos enfrentamos en esta misma fase fue la creación de los *sequenceFlows*, ya que al ser elementos que relacionan un nodo con otro nodo ( nodos que no siempre son colindantes), necesitábamos conocer que ids son los que tenían asignados cada uno, el problema consistía en que dichos ids se generaban al crear el propio nodo, pero en muchos casos el *sequenceFlow* hacía referencia a un nodo que aún no se había creado, por ejemplo al crear el *sequenceFlow* de una *exclusiveGateway*, en el momento de creación del *sequenceFlow* dicha *exclusiveGateway* al próximo nodo que apunta aún no se ha creado, luego no tiene un id establecido. Una posible opción hubiera sido almacenar dicho id en este momento hasta la creación del nodo, pero tampoco sería una solución válida ya que ese id se puede reutilizar para otros nodos cercanos a el, por lo que se realizó un pre parseado del flujo a traducir donde se irían generando y almacenando los diferentes ids de los nodos en un mapa donde la clave es su nombre y el valor el propio id y los *sequenceFlow* en otra estructura, que almacenaría una referencia al nombre actual del nodo, el nombre del próximo nodo y generaría el id de dicha realación es decir, el id del *SequenceFlow*.

### 8.3. Versiones

Todas las versiones han sido gestionadas gracias a la herramienta *GIT*, donde podemos ver la evolución que ha seguido la implementación de los traductores y cuando se ha realizado:

---

Git Log

---

```
commit 84d596ade527e6d0146624016c4bfbdcff865aa5
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>
Date:   Sun Jun 12 21:14:39 2016 +0200
```

Added lanes and updated the readme

```
commit 7bd0f76f99a890bd2c675eb82f969eff96a3ea17
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>
Date:   Sun Jun 12 19:27:22 2016 +0200
```

Done, it makes a identical HPE activator files and the BPMN  
validates GREAT

```
commit bbe4336d424e41cbb0b260a4951e4810ea77d7e9
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>
Date:   Sun Jun 12 00:02:32 2016 +0200
```

Can pass HPE fist lopp, but explodes in the next  
wich is more greater than the first

```
commit f51fc397340b089537917328ad7c68a8f5e48ba6
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>
Date:   Sat Jun 11 22:23:48 2016 +0200
```

Completed the activator\_to\_BPMN but can not pass HPE workflows

```
commit a2cf7648e83045bf85e3efce6264b6030cd21fd4
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>
Date:   Fri Jun 10 12:57:32 2016 +0200
```

Completed the most of the Activator\_to BPMN except the nodes

```
commit 298d9722090accc4475915437646a27963d235d6
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>
Date:   Thu Jun 9 19:25:18 2016 +0200
```

Lower changes in the BuyerProcess



commit 02a121b6bc70617dca09d33e9a5156f687c45c7a  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Thu Jun 9 18:34:04 2016 +0200

Removed lanes

commit c9c570c5158bdc11d7cfd392a5e39c042efc738d  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Thu Jun 9 18:30:39 2016 +0200

Created a real HPE workflow, corrected the Handlers, renamed  
hp to hpe and started Activator\_to\_BPMN

commit c1a4042fc0fcc6745da4598377c8673d994ba45f  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Sun Jun 5 18:50:40 2016 +0200

Realing the elements in order to validate vs HP dtd

commit c8b7829ae6a864b241b8094710bf0f1f9aea0437  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Sun Jun 5 13:58:58 2016 +0200

Second Version sent to HP observatory: First real Beta

commit f53ba411a168d82a5aa292ee04910fd4052137d4  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Sat Jun 4 20:59:42 2016 +0200

Aded solution element to the workflow

commit 8346b39b31edb5ae676cfc1d03b5fa5f94603ce3  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Sat Jun 4 20:27:08 2016 +0200

Aded Roles and some elements of the workflow

commit 8c3d084833c3a364033672436c18a0bf44e0b779  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Sat Jun 4 12:57:27 2016 +0200

Second Version sent to HP observatory: Aded Case-Packet  
and some minor improvements

commit eb5c8c110fd22f08d8ec73ec8b13b617ba6472f9  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Fri Jun 3 18:27:48 2016 +0200

Aded handlers

commit b012e54c295eee81eaeb3d77b328293680366a6c  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Thu Jun 2 14:51:47 2016 +0200

Aded post\_processing to remove duplicated nodes and removed  
next-node in the process-node when the next-node it is  
the final node

commit d77563bac4e6c7e384fe5215e8f2fd2a25f7ff2e  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Wed Jun 1 16:17:49 2016 +0200

Aded switch node and some improvements

commit 5548c083b37b9815f59d4d613038b287e9718fea  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Mon May 30 15:47:52 2016 +0200

Aded workflow attributes

commit 203f5a02383ae9106debdff1ff62e7bc7ea50f605  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Thu May 26 19:32:01 2016 +0200

First Version sended to HP observatory: Implemented the whole  
node logic but with many things to do apart of this

commit fce57bde3d44847714ab7383b6e3984e34e6e1bb  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Wed May 25 18:40:03 2016 +0200

I redesigned the whole xsl and implemented again the filled  
the process-nodes except the state and roles attributes

commit 5bf00604a50f9217728c6a0dd7a89b5948f7528c  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Mon May 23 14:11:28 2016 +0200

Real inital commit, Now, we can build any tag from any node  
with all his attributes

commit 5b33b6cbde75ff32a6c745dea34c7af36a133c29  
Author: Fernando Merino <fernando.merino.cejudo@alumnos.uva.es>  
Date: Sun May 22 14:31:34 2016 +0200

Commit inicial, en versión prueba

### **8.4. Estado actual del proyecto**

Ya que el proyecto se realizó hace un año y no se ha podido presentar hasta ahora, hemos podido preguntar a los tutores de HPE el estado actual de este proyecto para conocer de primera mano el resultado del mismo.

Actualmente forma parte de otro proyecto que está teniendo continuidad este año, donde se desarrollará un plug-in para Eclipse que permita comparar los cambios entre dos versiones diferentes de un mismo flujo de HPE y realizar el merge pertinente. Esta comparación se realiza en el formato BPMN, por lo que resulta imprescindible los traductores que se han desarrollado en este proyecto.

Sobre los problemas encontrados, me indicaron que únicamente existían dos, el primero relacionado con la codificación de los caracteres equivalentes a las comillas que solucionaron en el propio plugin de eclipse, y un fallo de duplicidad de ids en los sequenceFlows de los switch que está en fase de resolución. Por lo demás funciona correctamente.

## 9. Conclusiones y trabajo futuro

### 9.1. Conclusiones

Tras finalizar el proyecto con éxito, podemos concluir que dicho proyecto nos ha formado en una gran cantidad de aspectos, para empezar dicho proyecto se ha realizado conjuntamente con la empresa HPE, lo cual junto con las prácticas en empresa es un gran complemento a la hora de formarte una visión sobre cómo es una empresa y como es trabajar con ella.

Otro aspecto en el que la resolución de este proyecto que me ha ayudado bastante ha sido en la comprensión necesaria para enfrentarse a un estándar como BPMN, entenderlo y sobretodo saber aplicarle para desarrollar lo que necesites en un momento específico.

También me ha formado en las diferentes tecnologías que han sido necesarias para su desarrollo como son los XML schemas o los XSLs, tecnologías que aunque se ven de paso por la carrera, resultan muy útiles a la hora de almacenar datos y convertirlos más tarde. De hecho el estándar XML resulta uno de los mejores almacenes de datos a la hora de compartir datos, incluso mejor que Json para algunos casos.

Concluyendo ya, este proyecto no solo nos ha servido para resolver un problema que tenía la empresa HPE, si no que nos ha dado una visión más global de todo el proceso que conlleva realizar cualquier proyecto relacionado con el desarrollo del software así como un conocimiento mucho más profundo de las tecnologías utilizadas para su resolución.

### 9.2. Trabajo futuro

Como trabajo futuro a mayores del que se está llevando actualmente, proponemos que se amplíe la base de conocimiento de los traductores para que sean capaces de definir con mayor detalle la naturaleza de un *Process-Node* por ejemplo que sean capaces de discernir cuando se trata de un nodo que requiere intervención del usuario para poderle modelar con estructuras más específicas como puede ser para este caso la *User-Task* para ampliar el rango de opciones cuando se dé el paso a utilizar editores de flujos basados en BPMN.

Y como no, dar el paso a estos editores y a toda la funcionalidad que nos permiten añadir, ya que al estar destinada esta notación a modelar casi cualquier flujo que pueda darse, tiene soporte para infinidad de casos, los cuales podrían automatizar muchísimas tareas que posiblemente se estén realizando manualmente en la actualidad por la imposibilidad de ser modeladas según el estándar de HPE. Eso sin hablar de la cantidad de herramientas de edición disponibles en el mercado que posiblemente sean más intuitivas y eficientes que las utilizadas actualmente.

## Referencias

- [1] HP INVENT *HPE Service Activator*, Workflows and the Workflow Manager, V60-1A, Noviembre 28,2011.
- [2] OMG *Business Process Model and Notation (BPMN)* [ONLINE], Disponible en : <http://www.omg.org/spec/BPMN/2.0.2/PDF> [Accedido:20-febrero-2016]
- [3] OMG *BPMN v2.0 Examples machine readable files* [ONLINE], Disponible en : <http://www.omg.org/spec/BPMN/20100602/> [Accedido:20-febrero-2016]
- [4] M.VON ROSING, N. KEMP, M. ARZUMANYAN *The Complete Business Process Handbook* Understanding Business Process Management Roles [ONLINE], Disponible en: <http://www.sciencedirect.com/science/article/pii/B9780127999593000136> [Accedido:21-febrero-2016]
- [5] ERIK T. RAY *Learning XML* , Guide to Creating Self-Describing Data , O'Reilly Media, 2nd edition, 2003.
- [6] VLIST & ERIC VAN DER *XML Schema* , The W3C's Object-Oriented Descriptions for XML , O'Reilly Media, 2002.
- [7] DOUG TIDWELL *XSLT* , Mastering XML Transformations , O'Reilly Media, 2nd edition 2008.
- [8] ALEX BLEWITT *Eclipse 4 Plug-in Development by Example* , How to develop build, test package, and release Eclipse plug-ins with features for Eclipse 3.x and Eclipse 4.x , PACKT, 2013.
- [9] WIKIPEDIA *Software testing* [ONLINE], Disponible en : [https://en.wikipedia.org/wiki/Software\\_testing](https://en.wikipedia.org/wiki/Software_testing) [Accedido:2-julio-2017]

## Iconografía

- [1] HPE SERVICE ACTIVATOR 8.0 SOFTWARE, HPE ServiceActivator cluster architecture example for a high volume mobile subscriber activation solution [FIGURA 4] Recuperado de <https://www.hpe.com/h20195/v2/getpdf.aspx/4AA5-4013ENW.pdf?ver=1.0> [Accedido 13-mayo-2017].

## Anexo I: Ejemplo de un workflow según BPMN

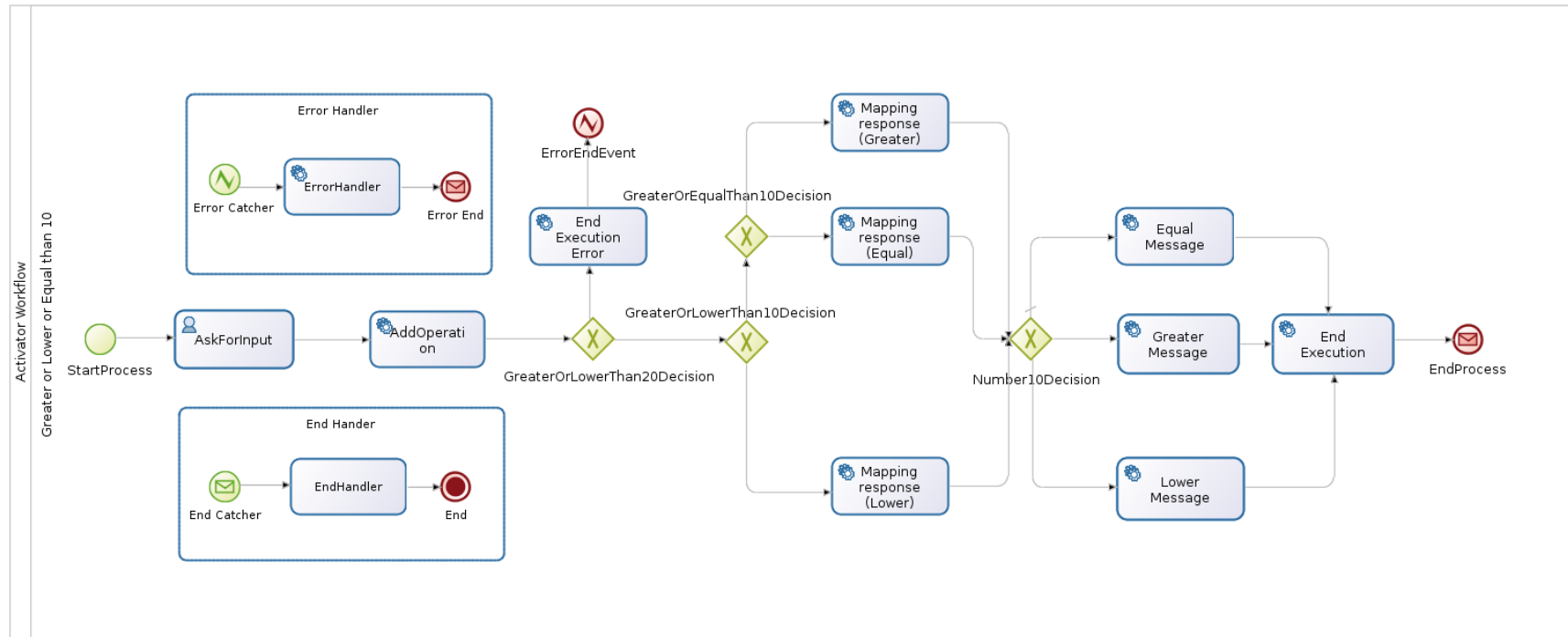


Figura 10: Ejemplo Workflow BPMN

Puedes consultar el código que genera el diagrama anterior en el fichero "BuyerProcess.bpmn" que se encuentra en la documentación.

## Anexo II: Estructuras mencionadas de BPMN

```

1 <xsd:element name="flowNode" type="tFlowNode"/>
2 <xsd:complexType name="tFlowNode" abstract="true">
3   <xsd:complexContent>
4     <xsd:extension base="tFlowElement">
5       <xsd:sequence>
6         <xsd:element name="incoming" type="xsd:QName" minOccurs=
7           "0" maxOccurs="unbounded"/>
8         <xsd:element name="outgoing" type="xsd:QName" minOccurs=
9           "0" maxOccurs="unbounded"/>
10      </xsd:sequence>
11    </xsd:extension>
12  </xsd:complexContent>
13 </xsd:complexType>

```

Código 32: Flow Node

```

1 <xsd:element name="flowElement" type="tFlowElement"/>
2 <xsd:complexType name="tFlowElement" abstract="true">
3   <xsd:complexContent>
4     <xsd:extension base="tBaseElement">
5       <xsd:sequence>
6         <xsd:element ref="auditing" minOccurs="0" maxOccurs="1"/>
7         <xsd:element ref="monitoring" minOccurs="0" maxOccurs="1"
8           "/>
9         <xsd:element name="categoryValueRef" type="xsd:QName"
10           minOccurs="0" maxOccurs="unbounded"/>
11       </xsd:sequence>
12       <xsd:attribute name="name" type="xsd:string"/>
13     </xsd:extension>
14   </xsd:complexContent>
15 </xsd:complexType>

```

Código 33: Flow Element

```

1 <xsd:element name="baseElement" type="tBaseElement"/>
2 <xsd:complexType name="tBaseElement" abstract="true">
3   <xsd:sequence>
4     <xsd:element ref="documentation" minOccurs="0" maxOccurs="unbounded"/>
5     <xsd:element ref="extensionElements" minOccurs="0" maxOccurs="1"/>
6   </xsd:sequence>
7   <xsd:attribute name="id" type="xsd:ID" use="optional"/>
8   <xsd:anyAttribute namespace="##other" processContents="lax"/>
9 </xsd:complexType>

```

Código 34: Base Element

```

1 <xsd:complexType name="tDocumentation" mixed="true">
2   <xsd:sequence>
3     <xsd:any namespace="##any" processContents="lax" minOccurs="
4       0"/>
5   </xsd:sequence>
6   <xsd:attribute name="id" type="xsd:ID" use="optional"/>
7   <xsd:attribute name="textFormat" type="xsd:string" default="
8     textplain"/>
9 </xsd:complexType>

```

Código 35: Documentation

```

1 <xsd:element name="sequenceFlow" type="tSequenceFlow"
2   substitutionGroup="flowElement"/>
3 <xsd:complexType name="tSequenceFlow">
4   <xsd:complexContent>
5     <xsd:extension base="tFlowElement">
6       <xsd:sequence>
7         <xsd:element name="conditionExpression" type="
8           tExpression" minOccurs="0" maxOccurs="1"/>
9       </xsd:sequence>
10      <xsd:attribute name="sourceRef" type="xsd:IDREF" use="
11        required"/>
12      <xsd:attribute name="targetRef" type="xsd:IDREF" use="
13        required"/>
14      <xsd:attribute name="isImmediate" type="xsd:boolean" use="
15        optional"/>
16    </xsd:extension>
17  </xsd:complexContent>
18 </xsd:complexType>

```

Código 36: Sequence Flow

```

1 <xsd:element name="callableElement" type="tCallableElement"/>
2 <xsd:complexType name="tCallableElement">
3   <xsd:complexContent>
4     <xsd:extension base="tRootElement">
5       <xsd:sequence>
6         <xsd:element name="supportedInterfaceRef" type="xsd:
7           QName" minOccurs="0" maxOccurs="unbounded"/>
8         <xsd:element ref="ioSpecification" minOccurs="0"
9           maxOccurs="1"/>
10        <xsd:element ref="ioBinding" minOccurs="0" maxOccurs="
11          unbounded"/>
12      </xsd:sequence>
13      <xsd:attribute name="name" type="xsd:string"/>
14    </xsd:extension>
15  </xsd:complexContent>
16 </xsd:complexType>

```

Código 37: Callable Element



```

1 <xsd:element name="process" type="tProcess" substitutionGroup="
  rootElement"/>
2 <xsd:complexType name="tProcess">
3   <xsd:complexContent>
4     <xsd:extension base="tCallableElement">
5       <xsd:sequence>
6         <xsd:element ref="auditing" minOccurs="0"maxOccurs="1"/>
7         <xsd:element ref="monitoring" minOccurs="0" maxOccurs="1
  "/>
8         <xsd:element ref="processRole" minOccurs="0" maxOccurs="
  unbounded"/>
9         <xsd:element ref="property" minOccurs="0" maxOccurs="
  unbounded"/>
10        <xsd:element ref="laneSet" minOccurs="0" maxOccurs="
  unbounded"/>
11        <xsd:element ref="flowElement" minOccurs="0" maxOccurs="
  unbounded"/>
12        <xsd:element ref="artifact" minOccurs="0" maxOccurs="
  unbounded"/>
13        <xsd:element ref="resourceRole" minOccurs="0" maxOccurs=
  "unbounded"/>
14        <xsd:element ref="correlationSubscription" minOccurs="0"
  maxOccurs="unbounded"/>
15        <xsd:element name="supports" type="xsd:QName" minOccurs=
  "0" maxOccurs="unbounded"/>
16      </xsd:sequence>
17      <xsd:attribute name="processType" type="tProcessType"
  default="None"/>
18      <xsd:attribute name="isExecutable" type="xsd:boolean" use="
  optional"/>
19      <xsd:attribute name="isClosed" type="xsd:boolean" default=
  "false"/>
20      <xsd:attribute name="definitionalCollaborationRef" type="
  xsd:QName" use="optional"/>
21    </xsd:extension>
22  </xsd:complexContent>
23 </xsd:complexType>
24 <xsd:simpleType name="tProcessType">
25   <xsd:restriction base="xsd:string">
26     <xsd:enumeration value="None"/>
27     <xsd:enumeration value "Public"/>
28     <xsd:enumeration value="Private"/>
29   </xsd:restriction>
30 </xsd:simpleType>

```

Código 38: Process

```

1 <xsd:element name="resource" type="tResource" substitutionGroup=
  "rootElement"/>
2 <xsd:complexType name="tResource">
3   <xsd:complexContent>
4     <xsd:extension base="tRootElement">
5       <xsd:sequence>
6         <xsd:element ref="resourceParameter" minOccurs="0"
          maxOccurs="unbounded"/>
7       </xsd:sequence>
8       <xsd:attribute name="name" type="xsd:string" use="required"
          />
9     </xsd:extension>
10  </xsd:complexContent>
11 </xsd:complexType>

```

Código 39: Resource

```

1 <xsd:element name="gateway" type="tGateway" abstract="true"/>
2 <xsd:complexType name="tGateway">
3   <xsd:complexContent>
4     <xsd:extension base="tFlowElement">
5       <xsd:attribute name="gatewayDirection" type="
          tGatewayDirection" default="Unspecified"/>
6     </xsd:extension>
7   </xsd:complexContent>
8 </xsd:complexType>
9 <xsd:simpleType name="tGatewayDirection">
10  <xsd:restriction base="xsd:string">
11    <xsd:enumeration value="Unspecified"/>
12    <xsd:enumeration value="Converging"/>
13    <xsd:enumeration value="Diverging"/>
14    <xsd:enumeration value="Mixed"/>
15  </xsd:restriction>
16 </xsd:simpleType>

```

Código 40: Gateway

```

1 <xsd:element name="rootElement" type="tRootElement"/>
2 <xsd:complexType name="tRootElement" abstract="true">
3   <xsd:complexContent>
4     <xsd:extension base="tBaseElement"/>
5   </xsd:complexContent>
6 </xsd:complexType>

```

Código 41: Root Element

```

1 <xsd:element name="resourceRole" type="tResourceRole"/>
2 <xsd:complexType name="tResourceRole">
3   <xsd:complexContent>
4     <xsd:extension base="tBaseElement">

```

```

5      <xsd:choice>
6      <xsd:sequence>
7          <xsd:element name="resourceRef" type="xsd:QName"
minOccurs="0" maxOccurs="1"/>
8          <xsd:element ref="resourceParameterBinding" minOccurs="
"0" maxOccurs="unbounded"/>
9      </xsd:sequence>
10     <xsd:element ref="resourceAssignmentExpression"
minOccurs="0" maxOccurs="1"/>
11 </xsd:choice>
12 </xsd:extension>
13 </xsd:complexContent>
14 </xsd:complexType>

```

Código 42: Resource Role

```

1 <xsd:element name="event" type="tEvent" substitutionGroup="
flowElement"/>
2 <xsd:complexType name="tEvent" abstract="true">
3 <xsd:complexContent>
4 <xsd:extension base="tFlowNode"/>
5 </xsd:complexContent>
6 </xsd:complexType>

```

Código 43: Event

```

1 <xsd:element name="eventDefinition" type="tEventDefinition"/>
2 <xsd:complexType name="tEventDefinition" abstract="true">
3 <xsd:complexContent>
4 <xsd:extension base="tBaseElement"/>
5 </xsd:complexContent>
6 </xsd:complexType>

```

Código 44: Event Definition

```

1 <xsd:element name="dataInput" type="tDataInput" />
2 <xsd:complexType name="tDataInput">
3 <xsd:complexContent>
4 <xsd:extension base="tBaseElement">
5 <xsd:attribute name="name" type="xsd:string" use="optional
" />
6 <xsd:attribute name="itemSubjectRef" type="xsd:QName" />
7 <xsd:attribute name="isCollection" type="xsd:boolean"
default="false"/>
8 <xsd:attribute name="dataState" type="xsd:IDREF"/>
9 </xsd:extension>
10 </xsd:complexContent>
11 </xsd:complexType>

```

Código 45: DataInput

```

1 <xsd:element name="message" type="tMessage" substitutionGroup="
  rootElement"/>
2 <xsd:complexType name="tMessage">
3   <xsd:complexContent>
4     <xsd:extension base="tRootElement">
5       <xsd:attribute name="name" type="xsd:string"/>
6       <xsd:attribute name="itemRef" type="xsd:QName"/>
7     </xsd:extension>
8   </xsd:complexContent>
9 </xsd:complexType>

```

Código 46: Message

```

1 <xsd:element name="intermediateCatchEvent" type="
  tIntermediateCatchEvent" substitutionGroup="flowElement"/>
2 <xsd:complexType name="tIntermediateCatchEvent">
3   <xsd:complexContent>
4     <xsd:extension base="tCatchEvent"/>
5   </xsd:complexContent>
6 </xsd:complexType>

```

Código 47: Intermediate Catch Event

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:
  hpe="extension_HPE" targetNamespace="extension_HPE"
  elementFormDefault="qualified" attributeFormDefault="
  unqualified">
3
4   <xsd:element name="actionParams" type="hpe:tActionParams"/>
5   <xsd:element name="param" type="hpe:tParam"/>
6   <xsd:element name="casePacket" type="hpe:tCasePacket"/>
7   <xsd:element name="variable" type="hpe:tVariable"/>
8   <xsd:element name="initial_casePacket" type="hpe:
  tInitial_casePacket"/>
9   <xsd:element name="variable_value" type="hpe:tVariable_value
  "/>
10  <xsd:element name="workflow_attributes" type="hpe:
  tWf_attributes"/>
11
12  <xsd:complexType name="tActionParams">
13    <xsd:sequence>
14      <xsd:element ref="hpe:param" minOccurs="0"
  maxOccurs="unbounded"/>
15    </xsd:sequence>
16  </xsd:complexType>
17  <xsd:complexType name="tParam">
18    <xsd:attribute name="name" type="xsd:string"/>
19    <xsd:attribute name="value" type="xsd:string"/>
20  </xsd:complexType>

```

```

21     <xsd:complexType name="tCasePacket">
22         <xsd:sequence>
23             <xsd:element ref="hpe:variable" minOccurs="0"
maxOccurs="unbounded"/>
24         </xsd:sequence>
25     </xsd:complexType>
26
27     <xsd:complexType name="tVariable">
28         <xsd:attribute name="name" type="xsd:string"/>
29         <xsd:attribute name="type" type="xsd:string"/>
30         <xsd:attribute name="disablePersistence" type="xsd:
boolean"/>
31     </xsd:complexType>
32
33     <xsd:complexType name="tInitial_casePacket">
34         <xsd:sequence>
35             <xsd:element ref="hpe:variable_value" minOccurs="0"
maxOccurs="unbounded"/>
36         </xsd:sequence>
37     </xsd:complexType>
38
39     <xsd:complexType name="tVariable_value">
40         <xsd:attribute name="name" type="xsd:string"/>
41         <xsd:attribute name="value" type="xsd:string"/>
42     </xsd:complexType>
43
44     <xsd:complexType name="tWf_attributes">
45         <xsd:sequence>
46             <xsd:element name="Init-On-Startup" type="xsd:
boolean" minOccurs="0" maxOccurs="1"/>
47             <xsd:element name="Unique" type="xsd:boolean"
minOccurs="0" maxOccurs="1"/>
48             <xsd:element name="statEnabled" type="xsd:boolean"
minOccurs="0" maxOccurs="1"/>
49             <xsd:element name="auditEnabled" type="xsd:boolean"
minOccurs="0" maxOccurs="1"/>
50             <xsd:element name="autoAuditEnabled" type="xsd:
boolean" minOccurs="0" maxOccurs="1"/>
51             <xsd:element name="disablePersistence" type="xsd:
boolean" minOccurs="0" maxOccurs="1"/>
52             <xsd:element name="maxNodesPerThread" type="xsd:
integer" minOccurs="0" maxOccurs="1"/>
53             <xsd:element name="auditStateChangeEvent" type="xsd:
boolean" minOccurs="0" maxOccurs="1"/>
54         </xsd:sequence>
55     </xsd:complexType>
56 </xsd:schema>

```

Código 48: HPE extension Schema

```

1 <xsd:element name="ioSpecification" type="
  tInputOutputSpecification" />
2 <xsd:complexType name="tInputOutputSpecification">
3   <xsd:complexContent>
4     <xsd:extension base="tBaseElement">
5       <xsd:sequence>
6         <xsd:element ref="dataInput" minOccurs="0" maxOccurs="
          unbounded"/>
7         <xsd:element ref="dataOutput" minOccurs="0" maxOccurs="
          unbounded"/>
8         <xsd:element ref="inputSet" minOccurs="1" maxOccurs="
          unbounded"/>
9         <xsd:element ref="outputSet" minOccurs="1" maxOccurs="
          unbounded"/>
10        </xsd:sequence>
11      </xsd:extension>
12    </xsd:complexContent>
13  </xsd:complexType>

```

Código 49: IoSpecification

```

1 <xsd:element name="inputSet" type="tInputSet" />
2 <xsd:complexType name="tInputSet">
3   <xsd:complexContent>
4     <xsd:extension base="tBaseElement">
5       <xsd:sequence>
6         <xsd:element name="dataInputRefs" type="xsd:IDREF"
          minOccurs="0" maxOccurs="unbounded"/>
7         <xsd:element name="optionalInputRefs" type="xsd:IDREF"
          minOccurs="0" maxOccurs="unbounded"/>
8         <xsd:element name="whileExecutingInputRefs" type="xsd:
          IDREF" minOccurs="0" maxOccurs="unbounded"/>
9         <xsd:element name="outputSetRefs" type="xsd:IDREF"
          minOccurs="0" maxOccurs="unbounded"/>
10        </xsd:sequence>
11        <xsd:attribute name="name" type="xsd:string" />
12      </xsd:extension>
13    </xsd:complexContent>
14  </xsd:complexType>

```

Código 50: InputSet

```

1 <xsd:element name="boundaryEvent" type="tBoundaryEvent"
  substitutionGroup="flowElement"/>
2 <xsd:complexType name="tBoundaryEvent">
3   <xsd:complexContent>
4     <xsd:extension base="tCatchEvent">
5       <xsd:attribute name="cancelActivity" type="xsd:boolean"
          default="true"/>
6       <xsd:attribute name="attachedToRef" type="xsd:QName"/>

```

```
7 </xsd:extension>
8 </xsd:complexContent>
9 </xsd:complexType>
```

Código 51: Boundary Event

```
1 <xsd:element name="catchEvent" type="tCatchEvent"/>
2 <xsd:complexType name="tCatchEvent" abstract="true">
3   <xsd:complexContent>
4     <xsd:extension base="tEvent">
5       <xsd:sequence>
6         <xsd:element ref="dataOutput" minOccurs="0" maxOccurs="
unbounded"/>
7         <xsd:element ref="dataOutputAssociation" minOccurs="0"
maxOccurs="unbounded"/>
8         <xsd:element ref="outputSet" minOccurs="0" maxOccurs="1"/>
9         <xsd:element ref="eventDefinition" minOccurs="0"
maxOccurs="unbounded"/>
10        <xsd:element name="eventDefinitionRef" type="xsd:QName"
minOccurs="0" maxOccurs="unbounded"/>
11      </xsd:sequence>
12      <xsd:attribute name="parallelMultiple" type="xsd:boolean"
default="false"/>
13    </xsd:extension>
14  </xsd:complexContent>
15 </xsd:complexType>
```

Código 52: Catch Event

## Contenido del CD

---

## Contenido del CD

El contenido del CD adjuntado es el siguiente:

- Versión PDF de la memoria llamado "memoria.pdf"
- Una carpeta con todo el software desarrollado llamado "Desarrollado"
- Un manual de Usuario llamado "ManualUsuario.pdf"
- Una carpeta llamada "Documentación" donde podremos encontrar el estándar BPMN.