

**ESTUDIO COMPARATIVO DE LENGUAJES DE
MODELADO DE PROCESOS DE NEGOCIO
PARA SU INTEGRACIÓN EN
PROCESOS DE DESARROLLO DE SOFTWARE
DIRIGIDO POR MODELOS**

**ÁREA DE LENGUAJES Y SISTEMAS INFORMÁTICOS
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD CARLOS III DE MADRID**



**INGENIERÍA EN INFORMÁTICA
PROYECTO FIN DE CARRERA**

Autor: Sara Magaña Orúe

Dirigido por: Prof. María Cruz Valiente Blázquez

AGRADECIMIENTOS:

A mis padres, José Antonio y Emeri, y a mis hermanos, Emilio, Pedro, Carmen y Ana, que a pesar de la distancia entre La Rioja y Madrid siempre me han dado su apoyo y cariño.

A mis sobrinos, Sofía y Nicolás que han sido mi motivación estos últimos meses.

A las hermanas de la residencia de estudiantes María Inmaculada y en especial a la hermana Charo, Jesusa, Gracia, Angelines y por supuesto a Vicenta María. También a todas mis compañeras y amigas de residencia, por las horas de estudio y tantos momentos compartidos. Gracias a todas ellas por apoyo y compañía en los años de carrera.

Y a Fran por su apoyo, consejos y paciencia en estos últimos seis meses de esfuerzo y estudio.

ÍNDICE DE CONTENIDOS

ÍNDICE DE CONTENIDOS	3
ÍNDICE DE IMÁGENES	5
ÍNDICE DE TABLAS	6
0. INTRODUCCION	8
0.1 Motivación	9
0.2 Planteamiento del problema	10
0.3 Objetivos	11
0.4 Descripción del trabajo	11
0.5 Organización de este documento	11
1. ESTADO DEL ARTE	13
1.1 Procesos De Negocio	15
1.2 Flujos De Trabajo	18
1.2.1 ¿Qué son los flujos de trabajo?	19
1.2.2 Conceptos básicos de Flujos de Trabajo	21
1.2.3 Sistemas de Gestión de Flujos de Trabajo (WfMS)	27
1.3 Patrones	28
1.3.1 Patrones de Flujo de Control	29
1.3.2 Patrones de Datos	57
1.3.3 Patrones de Recursos	61
1.4 Lenguajes De Modelado Para Representar Procesos De Negocio	64
1.4.1 BPMN	66
1.4.1.1 Introducción	67
1.4.1.2 Definición del Metamodelo de un Proceso de Negocio	68
1.4.1.3 Notación para el Modelado de Procesos de Negocio	71
1.4.1.4 Conformidad	73
1.4.1.5 Alcance de BPMN	75
1.4.1.6 Usos de BPMN	75
1.4.1.7 Diagramas de Procesos de Negocio	77
1.4.1.8 Uso de Texto, Color, Tamaño y Líneas en un Diagrama	99

1.4.1.9	Reglas de Conexión para los Objetos de Flujo	99
1.4.1.10	Reglas del Flujo de Secuencia	100
1.4.1.11	Reglas del Flujo de Mensajes	101
1.4.1.12	Herramientas para el modelado con BPMN	101
1.4.2	UML	103
1.4.2.1	Introducción	104
1.4.2.2	Conformidad	105
1.4.2.3	Unidades de Lenguaje.....	105
1.4.2.4	Niveles de Cumplimiento	106
1.4.2.5	Significado y tipos de conformidad	107
1.4.2.6	Semántica en Tiempo de Ejecución de UML	108
1.4.2.7	La Arquitectura de la Semántica.....	108
1.4.2.8	Conceptos de UML – Diagramas de Actividad	110
1.4.2.9	Herramientas para el modelado con UML.....	132
1.4.3	Redes De Petri	133
1.4.3.1	Introducción	133
1.4.3.2	Elementos	135
1.4.3.3	Definición formal.....	135
1.4.3.4	Representación Gráfica.....	136
1.4.3.5	Reglas de disparo	137
1.4.3.6	Propiedades y Métodos de análisis	139
1.4.3.7	Propiedades de comportamiento.....	139
1.4.3.8	Extensión de las redes de Petri básicas.....	140
1.4.3.9	Redes de Petri para modelar Procesos de Negocio.....	141
1.4.3.10	YAWL	141
1.4.3.11	Herramientas para el modelado con YAWL	145
2.	ANALISIS COMPARATIVO DE LOS LENGUAJES DE MODELADO DE PROCESOS DE NEGOCIO.....	147
3.	CONCLUSIONES	167
4.	LINEAS FUTURAS DE TRABAJO.....	171
5.	BIBLIOGRAFÍA	172
6.	SOFTWARE.....	176

ANEXOS	177
A. ANEXO: PRESUPUESTO	177

ÍNDICE DE IMÁGENES

Figura 1. Evolución de los sistemas de Información	18
Figura 2. Relación entre conceptos básicos de un flujo de trabajo.....	21
Figura 3. Y-Divisor.....	23
Figura 4. O-Divisor.....	23
Figura 5. Y-Unión.....	24
Figura 6. O-Unión.....	24
Figura 7. Tipo de Datos y flujo de información en un flujo de trabajo	25
Figura 8. Componentes de la estructura WfMS.....	28
Figura 9. Elementos de un flujo de trabajo basado en patrones	30
Figura 10. Patrón Unión Parcial de Cancelación	102
Figura 11. Ejemplo de reserva viaje con BPMN.	103
Figura 12. Arquitectura semántica.....	109
Figura 13. AcceptEventAction	112
Figura 14. Action	113
Figura 15. Action con condiciones.	113
Figura 16. Activity.....	115
Figura 17. ActivityEdge.....	116
Figura 18. ActivityFinalNode	116
Figura 19. ActivityNode	117
Figura 20. ActivityParametreNode	117
Figura 21. ActivityParameterNode	117
Figura 22. ActivityPartitió n	118
Figura 23. AddVariableValueAction.....	119
Figura 24. CallBehaviourAction.....	120
Figura 25. CallBehaviourAction.....	120
Figura 26. CallOperationAction	120
Figura 27. CentralBufferNode	121
Figura 28. ControlFlow	121

Figura 29. ControlNode	122
Figura 30. DataStoreNode	122
Figura 31. DecisionNode	123
Figura 32. ExceptionHandler	123
Figura 33. ExpansionRegion	125
Figura 34. FinalNode	125
Figura 35. Notación para FlowFinalNode	125
Figura 36. ForkNode.....	126
Figura 37. InitialNode.....	126
Figura 38. InterruptibleActivityRegion	126
Figura 39. JoinNode.....	127
Figura 40. MergerNode	127
Figura 41. ObjectFlow	128
Figura 42. ObjectNode.....	128
Figura 43. ParameterSet.....	129
Figura 44. Pin.....	130
Figura 45. SendSignalAction.....	130
Figura 46. StructuredActivityNode	131
Figura 47. ValueSpecification	131
Figura 48. Ejemplo del proceso de pedido y compra	133
Figura 49. Elementos de una Red de Petri.....	136
Figura 50. Ejemplo de una Red de Petri.	138
Figura 51. Ejemplo de disparo en una red de Petri.....	139
Figura 52. Notación para YAWL	144
Figura 53. Ejemplo de proceso con YAWL	146

ÍNDICE DE TABLAS

Tabla 1. Objetos de Flujo.....	80
Tabla 2. Objetos Conectores.....	81
Tabla 3. Swimlanes.....	82
Tabla 4. Artefactos.....	83
Tabla 5. Extensión elementos gráficos	99

Tabla 6. Reglas de Conexión Flujos de Secuencia	100
Tabla 7. Reglas de Conexión de Flujo de Mensajes	101
Tabla 8. Elementos de una red de Petri	135
Tabla 9. Patrones soportados por los lenguajes	170
Tabla 10. Total de horas empleadas	177
Tabla 11. Total euros requeridos en la bibliografía	177
Tabla 12. Total euros personal.....	178
Tabla 13. Total Euros requeridos en herramientas	178
Tabla 14. Total euros sin IVA	179
Tabla 15. Total Presupuesto con IVA.....	179

0. INTRODUCCION

Con la aparición de las nuevas tecnologías, más concretamente lo que se refiere a la informática como ayuda en las actividades del ser humano, la forma de llevar a cabo éstas, ha cambiado y evolucionado de acuerdo a las necesidades que se presentan en la vida diaria. Una de las direcciones de esta evolución ha sido el intento por automatizar las actividades cotidianas con ayuda de las *Tecnologías de la Información* (del inglés *Information Technologies. IT's*). Es así como las organizaciones de diferentes sectores incorporan tecnologías para la realización de sus procesos y objetivos de negocio. Como consecuencia de todo esto, las organizaciones han incorporado sistemas de información con el fin de organizar, automatizar y ejecutar sus procesos de negocio; de esta forma tenemos que la tecnología de *flujos de trabajo* (del inglés *Workflow*), proporciona una plataforma para lograr dicha automatización de procesos.

Esta tecnología de flujos de trabajo engloba diferentes conceptos, uno de los principales es: “*el proceso de flujo de trabajo*”, donde “*Un proceso de flujo de trabajo, es un proceso genérico dentro de una organización [36]. Como ejemplos de estos procesos podemos citar los siguientes: la compra en on-line de un libro, la inscripción de un alumno a un nuevo cuatrimestre, la adquisición de productos en un departamento de compras, etc...*”.

Es evidente que los procesos de flujo de trabajo son procesos comunes que se encuentran en la vida diaria y en las organizaciones, independientemente del giro de éstas. A estos procesos comúnmente también se les define como procesos de negocio. El atractivo de la tecnología flujo de trabajo, es la ejecución de manera automatizada o semi-automatizada de esos procesos de negocio en las organizaciones. Para dicha tarea se emplean los *Sistemas de Gestión de Flujo De Trabajo* (del inglés *Workflow Management Systems, WfMS*), los cuales a partir de la definición del proceso de negocio realizada con alguna técnica de modelado, llevan a cabo la ejecución del proceso, conjuntamente con otras tecnologías de información y de las personas que colaboran en la empresa. Los flujos de trabajo pueden ser modelados con distintos lenguajes como pueden ser: *BPMN, UML o YAWL*, y que son la base de este trabajo.

Para entrar más en detalle de los sistemas de gestión de flujos de trabajo, podemos decir que se caracterizan por tener un ciclo de vida con las siguientes fases:

- Fase de Inicio: Creación y Definición.
- Fase de Verificación y de Representación.
- Fase de Ejecución y Monitorización del proceso.

En este proyecto fin de carrera, el estudio se delimita a la primera fase, relativa a la creación y definición, donde la tarea más importante es el modelado de los procesos de flujo de trabajo que ejecutará el sistema de gestión de flujos de trabajo, las técnicas que se pueden emplear para llevar a cabo esta fase y cuál de ellas es la más recomendada para el modelado y desarrollo de flujos de trabajo.

“El modelado de Flujos de Trabajo consiste en la representación y definición de los procesos”. Un punto fuerte en el modelado de estos procesos son los “Patrones de Flujo de Trabajo”. El objetivo de estos patrones es la de delimitar los requisitos fundamentales que se plantean durante el proceso de elaboración de modelos, facilitando estructuras para modelar las distintas situaciones que pueden plantearse.

Como se describirá en el objetivo de este proyecto, los flujos de trabajo y los patrones serán los grandes protagonistas del tema central del proyecto, con lo que se ha considerado necesario explicar estos dos conceptos al principio de este documento. Los flujos de trabajo van a ser el objeto de este estudio, que se aplicarán sobre los patrones, para obtener conclusiones que no indiquen cuál de los lenguajes de modelado es el más apropiado para modelar procesos de negocio.

0.1 Motivación

Dentro de una organización hay multitud de formas de organizar, representar y gestionar las principales tareas que se llevan a cabo. Muchas organizaciones no siguen ningún estándar ni orden a la hora de realizar su trabajo, obteniendo tareas difusas, además de los problemas de mala gestión que esto con lleva. Otras en cambio, tienen un plan de actuación que les ayuda a controlar en todo momento las actividades realizadas y los productos obtenidos de estas tareas.

Estas organizaciones que llevan un orden en su forma de trabajo, utilizan estándares de planificación, de documentación y de actuación. Pero para ello previamente deben de definir claramente el objetivo principal de la organización. Una vez han definido el objetivo, deben definir las tareas a llevar a cabo por cada uno de los miembros del equipo u organización (se refiere a los roles, que se definirán más adelante).

El objetivo de este documento, es describir distintos estándares de modelado de los procesos de una organización. Se refiere a lenguajes de modelado de procesos de negocio, que de una u otra forma, facilitan la definición y representación de las tareas definidas en la organización.

Más concretamente, se quiere describir los flujos de trabajo como forma de modelar las tareas de una organización y comparar los distintos lenguajes que permiten representarlos gráficamente. Luego se obtendrá una conclusión de esta comparación que facilite la elección del mejor lenguaje para modelar los flujos de trabajo.

En el contexto de las organizaciones, el “*Soporte De Procesos De Negocio*” (BPM, del inglés Business Process Management) ha surgido para encauzar los continuos cambios que dichas organizaciones sufren en su cadena de valor y por lo tanto en sus procesos de negocio. La gestión de procesos de negocio ayuda a los analistas a manejar todos los aspectos relacionados con los procesos de negocio, pero el hueco entre estos analistas y la gente que construye las aplicaciones es demasiado grande. La “*Ingeniería Dirigida Por Modelos*” (MDE, del inglés *Model Driven Engineering*) aparece como una gran ayuda para transferir los cambios en los procesos de negocio a

los sistemas que implementan dichos procesos. Así, usando alguna propuesta de la ingeniería dirigida por modelos, como es la “*Arquitectura Dirigida por Modelos*” (MDA, del inglés *Model Driven Architecture*) se podría mejorar la interacción entre la gente de negocios y los ingenieros del software. En ese marco se encuadra el siguiente documento técnico, donde se recogen las diferentes propuestas, así como una serie de estándares que ayudan al diseño de los procesos de negocio y que encajan dentro de la filosofía de la ingeniería dirigida por modelos. Dentro de este marco, lo que se quiere obtener es, qué lenguaje facilita la integración del modelo de negocio y los ingenieros del software, facilitando así el desarrollo, minimizando la complejidad y el coste.

Bajo esta perspectiva, la MDE pretende ser elemento clave de la ingeniería del software haciendo que los modelos dejen de utilizarse básicamente para documentar, y constituyan los artefactos principales de todo proyecto de desarrollo de software [45]. El término MDE fue propuesto inicialmente por Stuart Kent en [46], pero este término tiene su origen en una iniciativa del OMG conocida como MDA (Model Driven Architecture, es decir, arquitectura dirigida por modelos) [47].

MDE es el resultado de desarrollos recientes en lenguajes de programación, conscientes de la necesidad de metodologías de desarrollo de sistemas y de la necesidad de gestionar proyectos de desarrollo de sistemas cada vez más complejos y de mayor tamaño. Los dos elementos clave en MDE son los lenguajes de modelado y las herramientas de modelado, que permitirán crear y transformar los modelos.

0.2 Planteamiento del problema

A pesar de que existen diversos enfoques para el modelado de procesos de flujos de trabajo, actualmente no existe una metodología gráfica estándar. Además, a través de un marco de trabajo de evaluación conocido como patrones de flujos de trabajo, han sido sometidos los lenguajes de modelado y herramientas de software para el modelado de procesos de flujo de trabajo más utilizados e importantes en la actualidad. En investigaciones tales como [7], [9] se presentan resultados y conclusiones que muestran que ningún de los lenguajes y sistemas estudiados modela todas las posibles situaciones que pueden encontrarse en un proceso de flujo de trabajo y que son representadas a través de patrones de flujos de trabajo.

Una consecuencia de lo anterior, se presenta cuando el modelo de un sistema deja de ser un modelo pequeño y tiende a ser muy complejo mantener la perspectiva de lo modelado (principalmente de manera visual). La importancia de la definición del proceso de flujos de trabajo radica en que en base al modelo obtenido se realiza la ejecución del proceso de manera automatizada con *WfMS*. Es así como la correcta especificación del proceso asegurará en mayor medida la ejecución correcta de este.

Pero lo que más importa, es encontrar herramientas que no solo modelen los flujos de trabajo, sino que también integren los modelos de procesos de negocio en un desarrollo de software dirigido por modelos.

0.3 Objetivos

El objetivo general de este documento es obtener una perspectiva y conocer más sobre los lenguajes de modelado de flujos de trabajo de cara a su integración en un proceso de desarrollo de software dirigido por modelos.

Los objetivos específicos de este documento son los siguientes:

- Conocer más en detalle los lenguajes de modelado de flujos de trabajo más relevantes, especificando la notación utilizada por cada uno de ellos, para el modelado.
- Realizar mediante un marco de trabajo, una evaluación de los distintos lenguajes y compararlos, tomando como base los patrones de flujos de trabajo.
- Nombrar algunas herramientas que soporten los lenguajes de modelado descritos.
- Obtener conclusiones de la comparación de los distintos lenguajes para poder decir cuál es el lenguaje que más conviene a una organización para el modelado de proceso de negocio y la integración de estos para el desarrollo de software de la empresa con un enfoque MDE.

0.4 Descripción del trabajo

Para el desarrollo de este proyecto fin de carrera, se realizaron las siguientes actividades: como primera tarea se realizó una investigación sobre el área de flujos de trabajo, la cual permitió conocer los conceptos y los temas relacionados a la tecnología de flujos de trabajo; de igual forma se llevó a cabo el estudio del modelado de flujos de trabajo y de los patrones que lo conforman. Posteriormente se realizó el estudio de los distintos lenguajes de modelado de flujos de trabajo (BPMN, UML y YAWL), y de las herramientas que los implementan y permiten el desarrollo de modelos con ellos, para acabar con el análisis comparativo de ellos y obtener conclusiones sobre cuál de ellos es el más apropiado para el modelado y futura integración con el desarrollo de software.

0.5 Organización de este documento

La estructura de este documento ha sido dividida en tres apartados principales, tres apartados menos relevantes (líneas futuras, bibliografía y software) y anexos.

Los apartados principales se han organizado de la siguiente forma:

En el primer apartado de Introducción, se ha descrito la motivación de este trabajo, se ha planteado el problema y definido los objetivos a alcanzar al final de este documento.

En el segundo apartado, Estado del Arte, se entra en detalle de los principales conceptos de este documento. Los cuales son los procesos de negocio o flujos de

trabajo, dando una descripción de ellos. Una vez entendidos en qué consisten estos procesos, se ha llevado a cabo la base del análisis, que son los patrones de flujos de trabajo, describiendo los tres tipos que existen y entrando en detalle únicamente en el primer grupo de patrones (patrones de flujo de control).

Una vez realizada la descripción de los patrones de flujo de trabajo, se estudian los distintos lenguajes de modelado para representar flujos de trabajo (BPMN, UML, y YAWL). En cada uno de los apartados que describen estos lenguajes, se ha indicado las principales características que los definen, (en todo momento se hace referencia a la bibliografía en la que se ha basado este documento), las cuales son muy distintas en cada uno de los lenguajes, ya que se encontrará mucha diferencia entre los apartados que componen cada descripción.

En el tercer apartado, se realiza el estudio comparativo de los lenguajes de modelado de procesos de negocio, estudiando qué lenguajes soportan qué patrones.

El cuarto apartado y el más importante, se llegan a las conclusiones obtenidas después de realizar todo el análisis de lenguajes y el análisis comparativo de estos.

Para acabar este documento, se citan futuras líneas de trabajo, la bibliografía utilizada, además del software empleado.

En los anexos se podrá consultar el presupuesto de este estudio.

1. ESTADO DEL ARTE

Durante las últimas dos décadas, los avances en lenguajes de programación y plataformas empleadas han aumentado el nivel de abstracción disponible en la tarea de desarrollo de software. Además debido a la madurez de los lenguajes de tercera generación, los desarrolladores de software están mejor equipados para afrontar y resolver los distintos problemas que se les pueden plantear.

A pesar de todos estos avances, aún quedan problemas importantes que resolver. En el centro de éstos se encuentra el crecimiento de la complejidad de las plataformas, las cuales contienen miles de clases y métodos con dependencias muy complicadas que deben ser conocidas por el desarrollador. El problema se acrecienta cuando estas mismas plataformas crecen rápidamente y además aparecen otras nuevas, con el consiguiente esfuerzo de migración de unas a otras.

En este último caso, cuando la evolución tecnológica de las plataformas o de los sistemas se produce, es importante conservar el mismo modelo conceptual del negocio, es decir, que la lógica del dominio del problema debería ser la misma, sea cual sea la plataforma o lenguaje que implementa dicha lógica. Para manejar el problema del crecimiento de la complejidad de los sistemas, la orientación a objetos no parece ser suficiente. Los lenguajes orientados a objetos han ido perdiendo la simplicidad con la que fueron ideados, el encapsulamiento no es un recurso tan útil como en principio parecía y, sobre todo, la reutilización de los objetos como componentes no ha tenido demasiado éxito en la industria del software [38].

Parece que las propuestas centradas en código no dan respuesta a las demandas de los sistemas actuales. Ésta es la razón por la que ha aparecido una nueva propuesta centrada en modelos.

A esta propuesta se le llama ingeniería dirigida por modelos (MDE), ya definida anteriormente.

Este paradigma combina los siguientes conceptos [39]:

- *Lenguajes de Dominio Específico (DSL, del inglés Domain Specific Language):* que formalizan la estructura de la aplicación, el comportamiento y los requisitos dentro de un dominio particular. Estos lenguajes (DSL) son descritos usando metamodelos, los cuales definen relaciones entre elementos dentro de un dominio. Un metamodelo se define en [48] como “*un modelo de un lenguaje de modelado. El metamodelo define la estructura, semántica, y restricciones para una familia de modelos*”. Se considera familia de modelos “*al conjunto de modelos que comparten una sintaxis y una semántica común*”. Tal y como lo define [45], un metamodelo es “*un grafo compuesto de conceptos y relaciones entre esos conceptos*”. Por su parte, en [49] se define un metamodelo como “*un tipo especial de modelo que especifica la sintaxis abstracta de un lenguaje de modelado*”. Por tanto, se

puede decir que el metamodelo va a actuar como filtro para extraer los elementos de un sistema y construir el modelo válido asociado. De esta manera, cualquier característica (concepto o relación) que no aparezca en el metamodelo se debe ignorar cuando se construya el modelo que representa el sistema. Con lo cual, un metamodelo se encuentra a un nivel más alto de abstracción que los modelos, y ha de representar a través de la sintaxis abstracta los conceptos considerados en esos modelos y las relaciones que se dan entre esos conceptos, junto con sus restricciones. De este modo, como indica [50] “*un modelo solamente será válido si es conforme a su metamodelo*”.

- *Motores de transformación y generadores.* Analizan ciertos aspectos de los modelos y después crean varios tipos de artefactos, tal como código fuente, entradas de simulación, descripciones de uso XML, o representaciones alternativas de dicho modelo.

Las herramientas de la ingeniería dirigida por modelos usan los conceptos anteriores y hacen más fácil para los ingenieros del software el soporte a la evolución del software, tanto en su lógica como en su tecnología.

Mediante los lenguajes de dominio específico se consiguen notaciones de modelado distintas para cada tipo de sistema, las cuales están definidas formalmente por su metamodelo. De esta manera, el ingeniero del software tiene herramientas específicas para cada tipo de sistema, lo cual le permite modelarlos de una manera más detallada y de acuerdo al dominio al que pertenecen.

Mediante los motores de transformación se facilita la evolución de modelos, transformando de unos modelos a otros, según la reglas de transformación entre metamodelos.

En el paradigma de la ingeniería dirigida por modelos cualquier concepto debe ser modelado. De esta manera, cualquier cambio o nueva propiedad del sistema debe ser mostrado en su modelo correspondiente. Con este paradigma, la parte de escritura de código es una parte más del proceso de construcción de sistemas (quizás la menos importante), la cual se sugiere que se realice automáticamente.

Ingeniería Dirigida por Modelos: Arquitectura Dirigida por Modelos (MDA)

El consorcio OMG (*Object Management Group*) ha desarrollado la propuesta Arquitectura Dirigida por Modelos o MDA como ejemplo de implementación de la ingeniería dirigida por modelos.

La arquitectura dirigida por modelos nace con la idea establecida de separar la especificación de la lógica operacional de un sistema, de los detalles que definen cómo el sistema usa las capacidades de la plataforma tecnológica donde es implementado.

Teniendo en cuenta lo anterior, los objetivos de la arquitectura dirigida por modelos son la portabilidad, la interoperabilidad y la reusabilidad a través de la separación arquitectural.

El concepto de *independencia de plataforma* aparece frecuentemente. Es la cualidad que tienen los modelos de ser independientes de las características de cualquier tipo de plataforma tecnológica.

Mediante la aplicación de este paradigma se cubre completamente el ciclo de vida de un sistema software, desde la captura de requisitos hasta el mantenimiento del mismo, pasando por la generación del código fuente. Para ello define tres tipos de modelos que se explican a continuación.

Los tipos de modelos que la arquitectura dirigida por modelos propone, son los siguientes:

- ***Modelo Independiente de la Computación (CIM)***: Un CIM no muestra detalles de la estructura del sistema. A veces es llamado modelo de dominio o modelo de negocio. En el CIM se modelan los requisitos que deberá satisfacer el sistema, describiendo la situación en la cual el sistema será usado. Es muy útil tanto para ayudar a comprender el problema como para ejercer de fuente de vocabulario compartido para el uso en otros modelos. Según la arquitectura dirigida por modelos, la especificación de requisitos de un sistema CIM debería ser transformable en un PIM y posteriormente en un PSM y viceversa. El CIM juega un papel importante como puente entre los que son unos expertos en el dominio del problema y sus requisitos y aquellos que son expertos en el diseño y construcción de artefactos software.
- ***Modelo Independiente de la Plataforma (PIM)***: Un modelo PIM muestra el grado de independencia de plataforma necesario para poder ser usado en diferentes plataformas tecnológicas de un tipo similar. Este modelo debe tener tal nivel de abstracción que no cambie, sea cual sea la plataforma elegida para su implementación. Con este modelo se representa la lógica del sistema y sus interacciones con el mundo exterior, sin entrar en detalle de que tipo de tecnología implementará cada parte y cómo se adapta a una plataforma específica.
- ***Modelo Específico de Plataforma (PSM)***: El modelo PSM es una vista del sistema para una plataforma específica. Éste combina la especificación del sistema hecha en el PIM, con los detalles que especifican la manera en que dicho sistema usa una plataforma particular.

1.1 Procesos De Negocio

El Diccionario Oxford de la lengua Inglesa define proceso como “*toda acción continua y regular o sucesión de acciones que tiene lugar de una forma determinada y que conducen a un resultado determinado*”. Por su parte, la RAE (*Real Academia*

Española) define un proceso como “conjunto de las fases sucesivas de un fenómeno natural o de una operación artificial”.

Un *proceso de negocio* (aunque también se puede denominar simplemente proceso) se define en [51] como “un conjunto de actividades interrelacionadas, iniciadas en respuesta a un evento, que permite obtener un resultado específico para el cliente del proceso”. El *evento* se correspondería con la petición específica del resultado que genera el proceso, y el *cliente* sería el beneficiario del resultado generado por el proceso de negocio. Aclarar que no se entiende como cliente solamente a quien compra bienes o servicios (es decir, productos), sino, en un concepto más amplio, a todos aquéllos ya sean internos o externos a la organización que encuentran valor en el resultado del proceso. El montaje de un coche en una fábrica, y la tramitación de pedidos en una empresa constituyen ejemplos de procesos de negocio. Para [52] todo proceso de negocio tiene un objetivo definido, por lo que todo proceso de negocio debe definir cómo la organización consigue alcanzar ese objetivo.

Por su parte, la organización sin ánimo de lucro que escribe sobre estándares de flujos de trabajo y BPM (*Business Process Modeling*, es decir, modelado de procesos de negocio), WfMC (*Workflow Management Coalition*) [2], define un proceso de negocio como “un conjunto de uno o más procedimientos o actividades que de manera conjunta permiten alcanzar un objetivo de negocio, normalmente dentro del contexto de una estructura organizacional que define roles y relaciones”.

Otra definición mucho más simplista se encuentra en [53] que define un proceso de negocio como “un conjunto de actividades internas llevadas a cabo para servir a un cliente”. Una vez más, cliente entendido en el sentido amplio del término. De acuerdo a [53], el objetivo de cada proceso de negocio ha de ser el de proporcionar al cliente el producto que realmente necesita, con un presupuesto, duración, calidad y resultado aceptables.

En definitiva, se puede decir de manera general que “un proceso es una forma de organizar trabajo y recursos para alcanzar unos objetivos determinados, a través de un conjunto de actividades que se llevan a cabo en un determinado orden”.

De acuerdo a [51] “el flujo de información y de control del proceso de negocio vendrá definido a través de un flujo de trabajo”. De la misma manera, para [52] todo proceso de negocio constituye un flujo de trabajo tratado en forma de escenario. Es decir, un escenario va a mostrar el flujo de trabajo que se da en el proceso de negocio. Por tanto, se puede deducir que un flujo de trabajo sería de manera simplificada un *modelo de un proceso de negocio*, denominado que define el qué (*objetivo del proceso*), cómo (*actividades*), quién (*responsables*), y cuándo (*orden*) de un proceso de negocio real.

El objetivo de un modelo de proceso de negocio es describir (abstraer) de una realidad compleja aquellas actividades que son clave en el proceso para facilitar su comprensión. Por ello, diversos autores justifican la conveniencia de utilizar técnicas de modelado orientadas a objetos no sólo para describir sistemas software complejos, sino

también para describir procesos de negocio. Para [53], por ejemplo, un buen argumento para utilizar la orientación a objetos cuando se modelan organizaciones es el hecho de que las organizaciones se encuentran muy ligadas al mundo real. Es decir, el hueco semántico que se da entre el modelo de negocio de la organización y el mundo real es mínimo. Por tanto, modelos orientados a objetos que representen procesos de negocio resultarán sencillos y fáciles de entender. Además, tal y como destaca [53], al modelar tanto el proceso de negocio como el sistema software con un lenguaje de modelado orientado a objetos ambos modelos se podrán relacionar de una manera mucho más directa y efectiva, ya que objetos del modelo de negocio se corresponderán con objetos del sistema. De esta manera se consigue una trazabilidad entre los dos modelos (negocio y sistema), lo que reduce el riesgo de introducir errores en la transición de un modelo al otro.

Por su parte, en [54] se enumeran un conjunto de ventajas que se obtienen cuando se adopta una orientación a objetos a la hora de modelar los procesos de negocio:

- *Conceptos similares.* Un negocio se puede describir en forma de procesos que alcanzan objetivos a través de la colaboración de diferentes objetos. Los objetos del mundo real y los objetos del modelo de proceso se encuentran muy vinculados por lo que se podrán relacionar de manera muy sencilla.
- *Técnicas consolidadas.* La programación orientada a objetos y el modelado orientado a objetos llevan utilizándose con éxito durante varios años para gestionar el desarrollo de sistemas complejos.
- *Notación estándar.* UML, por ejemplo, es un estándar de modelado muy utilizado para desarrollar sistemas. Con lo cual, la gran variedad de herramientas que trabajan con este estándar se podrían adaptar de una manera muy sencilla para poder incluir el modelado de procesos de negocio utilizando UML.
- *Facilidad de aprendizaje.* Los mismos conceptos que se utilizan a la hora de describir sistemas de información (objetos, clases, etc.) se pueden utilizar para describir procesos de negocio. De esta manera se minimiza el hueco de comunicación existente entre los modeladores de negocio y los modeladores del sistema. Se puede decir que ahora todos hablarán el mismo “idioma”, lo cual supone un gran beneficio.
- *Métodos nuevos y más sencillos de ver una organización o un negocio.* Las técnicas orientadas a objetos pueden mostrar los procesos de negocio de manera sencilla y resultan tan eficaces como los métodos tradicionales de describir y ver una organización.

1.2 Flujos De Trabajo

En la década de los 90's surgió la tecnología flujo de trabajo en el contexto de los sistemas de información, como una nueva solución para la administración o gestión de los procesos que se realizan dentro de una organización. Con el auge de Internet y de las tecnologías de la información, el flujo de trabajo ha evolucionado de manera rápida y su uso se ha incrementado en los negocios de diversas industrias. Su característica principal es la automatización de procesos organizacionales (conocidos también como procesos de negocio), los cuales integran por lo general una combinación de personas y actividades basadas en máquinas, que además, en su trabajo interactúan con aplicaciones de software y tecnologías de información.

La evolución de los sistemas de información puede verse por etapas en la figura 1 obtenida en [1]. La figura 1 describe la arquitectura de los sistemas de información en términos de componentes.

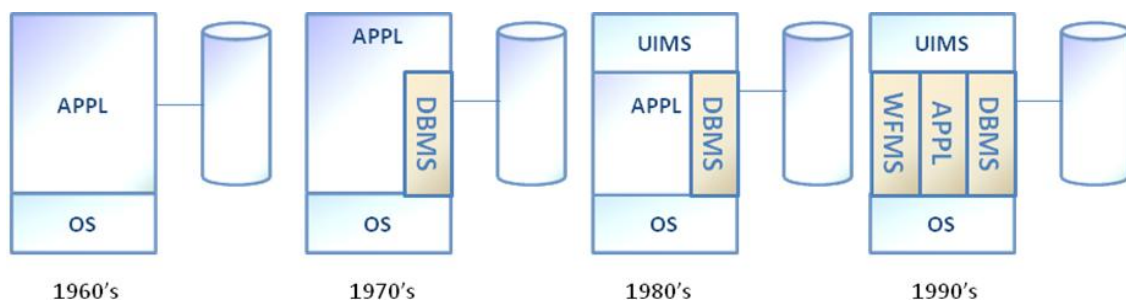


Figura 1. Evolución de los sistemas de Información

Años 60: Los datos y las aplicaciones campaban juntas, sin hacer ninguna distinción, dificultando el tratamiento de la información.

Años 70: Los datos fueron separados de las aplicaciones. Para ello se creó los *Sistemas Administradores De Base De Datos* (del inglés Database Management System, DBMSs), existían diversos sistemas de información en los cuales se manejaba y administraba toda la información necesaria para llevar a cabo la producción en las empresas. Se lograron automatizar tareas que antes eran realizadas manualmente.

Años 80: Surgió la necesidad de mejorar el flujo de la información, y de obtener la información de manera rápida y eficiente, además de optimizar la productividad, acortar tiempos en la realización de los procesos y tener control sobre éstos con el fin de reducir costes y mejorar la gestión en la organización. Con la gestión de *Interfaces De Usuarios* (UIMS, del inglés *User Interface Management Systems*) cada vez más interactivas y menos susceptibles de aceptar errores humanos. Se logró mayor facilidad para utilizar los sistemas por parte de los usuarios finales. Esto permitió la proliferación de los sistemas de información basados en ordenadores, al ser más accesibles en cuanto a la utilización para las personas en general.

Años 90: Aparece la gestión de flujos de trabajo, que ayuda en la administración del control y ejecución de procesos de negocio en las organizaciones.

Para entender mejor lo ocurrido en los 90's, cuando aparecen los flujos de trabajo, es necesario definir lo que significa un proceso de negocio formalmente. El Workflow Management Coalition (WfMC) principal organización en el mundo sobre el tema de flujos de trabajo lo define de la siguiente manera: “*un proceso de negocio es el conjunto de uno o más procedimientos o actividades relacionadas, que colectivamente realizan un objetivo del negocio, normalmente, lo anterior es dentro del contexto de una estructura organizacional que define roles funcionales y relaciones entre los mismos*” [2]. También es necesario definir el término flujo de trabajo, el WfMC lo define como: “*la automatización informatizada de un proceso de negocio, de manera total o parcial, en la cual documentos, información o tareas son pasadas de un participante a otro para efectos de su procesamiento. Lo anterior se realiza de acuerdo a un conjunto de reglas establecidas (patrones)*” [2].

La gestión de flujos de trabajo no es una solución que se implementa únicamente con la instalación de un programa de software para después esperar la obtención de resultados de forma inmediata; por el contrario, una solución de este tipo representa el tener tecnologías de la información, sistemas de software, políticas organizacionales, cooperación de los empleados de la empresa, entre otros elementos más que trabajan de acuerdo a la implementación necesaria para automatizar los procesos de negocio de la empresa; además, para la implementación se requiere de un análisis de la forma en que se realizan actualmente los procesos y en ciertos casos es necesario la aplicación de re-ingeniería en los procesos, lo que significa volver a plantear la manera en que éstos se realizan con el fin de mejorar [6]. Entre los sistemas de software necesarios, se encuentra una componente principal para la implementación de una solución basada en flujos de trabajo, que es el Sistema Gestión de Workflow (WfMS, del inglés *Workflow Management System, Flujo*).

1.2.1 ¿Qué son los flujos de trabajo?

Como ya se ha dicho el objetivo principal de un flujo de trabajo es automatizar la secuencia de acciones, tareas o actividades con la información y los documentos implicados en la ejecución de un proceso de negocio y además realizar el seguimiento de los estados de cada una de las etapas. Tanto los estados como las etapas del proceso de negocio, serán definidos por los responsables del proceso de negocio.

Un flujo de trabajo puede ser organizado manualmente, pero en la práctica se organiza dentro de un sistema en las tecnologías de la información para proporcionar soporte a los procedimientos de automatización.

Algunas instancias de procesos individuales pueden estar operativas durante el proceso de ejecución de un flujo de trabajo, en otros casos se puede asociar un cambio de un dato relevante como instancia de un proceso individual (algo que produce un cambio se puede modelar de forma individual).

La mayoría de las reglas procedimentales suelen ser definidas con anterioridad en la producción de flujos de trabajo, en flujos de trabajo ad-hoc y pueden ser modificadas y creadas durante el proceso.

Los flujos de trabajo suelen estar asociados a la *reingeniería de procesos de negocio* (del inglés, *Business Process Re-engineering*), que se encarga de la obtención de requisitos, análisis, diseño y definición del núcleo de un proceso de negocio. Pero no todas las actividades de un proceso de negocio se implementan con flujos de trabajo. Es apropiado utilizar flujos de trabajo cuando se pretende separar los procedimientos lógicos de negocio y el soporte operacional de las TI's, permitiendo incorporar cambios posteriores en el ejercicio de un proceso de negocio, para lo cual se ha definido. Esto quiere decir que en un proceso de negocio existirán dos partes claramente diferenciadas. Por una parte se encontrará la lógica de negocio que se modelará mediante procesos y será la actividad concreta de la empresa, modelada mediante flujos de trabajo y por otro lado está el mantenimiento de las propias herramientas y la gestión operacional de las TI's como ya se ha dicho.

Las funciones más comunes que realizan los flujos de trabajo son:

- Automatización de las secuencias de los procesos de negocio y optimización de las mismas.
- Control y seguimiento de dichos procesos.
- Agilización de los procesos de negocio y lo que resulta en un mejor servicio al cliente.
- Modificación de instancias de procesos sobre la marcha.
- Asignación de tareas al personal.
- Aviso al personal de tareas pendientes.
- Permitir la colaboración en las tareas comunes.
- Optimización de recursos humanos y técnicos alineándolos a la estrategia de la empresa.

Las funciones citadas anteriormente aportan diversos beneficios a la organización. Estos han convertido al flujo de trabajo en una tecnología imprescindible dentro los negocios de organizaciones actuales. Algunos de los beneficios más importantes son:

- Mejora en servicio y atención al cliente.
- Incremento en la ejecución de actividades en paralelo.
- Minimización en el tiempo para acceso a documentación, aplicaciones y bases de datos.
- Disminución del tiempo de transferencia de información entre tareas.
- Se asegura la completa colaboración del personal de trabajo.
- Permite tener conocimiento inmediato del estado de alguna tarea o proceso.
- Mejora en la gestión y optimización de procesos.

De lo anterior, lo más específico de la tecnología de flujo de trabajo son los *Sistemas de Gestión de Flujos de Trabajo (WfMS)*. En siguientes secciones se proporciona una definición y las funciones de este tipo de sistemas, además de su arquitectura, la cual actualmente se encuentra estandarizada.

1.2.2 Conceptos básicos de Flujos de Trabajo

Se ha tomado la especificación [3] de *WfMC* para establecer los conceptos básicos de flujo de trabajo. Es importante conocer y entender bien estos conceptos para poder entender el estudio que se va a realizar más adelante. Los conceptos más importantes que se van a definir son los siguientes: Caso, Proceso, Tarea, Enrutamiento, Datos, Personas, Roles, Grupos, Eventos, Plazos. Estos términos se han obtenido de los documentos [2] y [3]. Existen más conceptos de los aquí citados, pero no van a ser definidos debido a que no se consideran relevantes para entender este documento. En la figura 2 se puede ver la relación entre los siguientes conceptos.

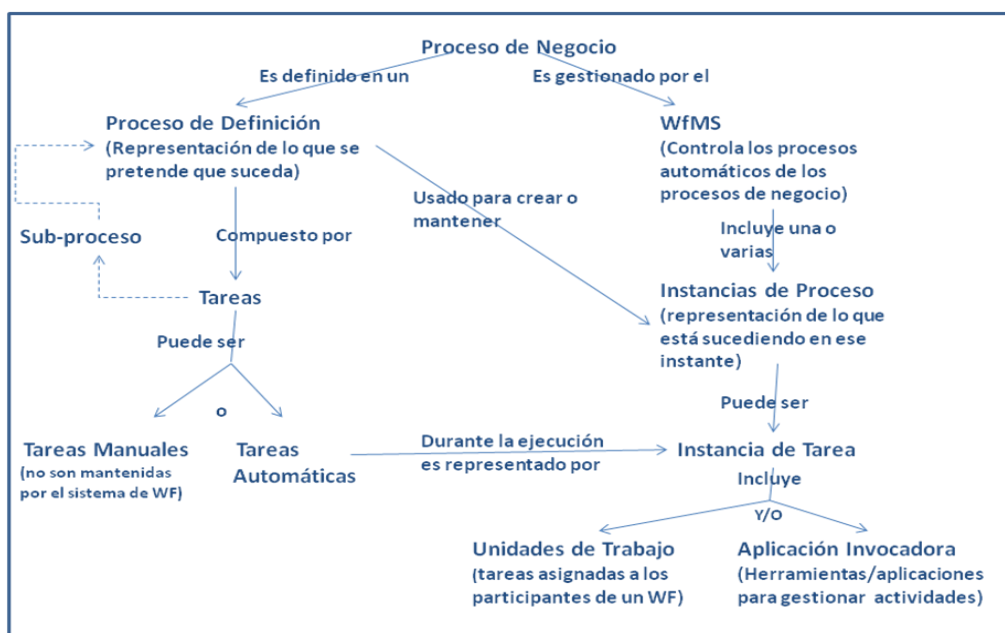


Figura 2. Relación entre conceptos básicos de un flujo de trabajo

De acuerdo a [3]:

Caso: Un caso es un elemento físico y abstracto el cual es producido o modificado por algún tipo de trabajo. Un caso siempre tiene un inicio y un final. La modelización de este tipo de elementos se llama *Casos de Uso*, aunque en este documento no se va a entrar en tal detalle. Trabajo, Job, Producto y Servicio serían sinónimos de este concepto.

Proceso: Un proceso está compuesto por un número de tareas que necesitan ser realizadas y un conjunto de condiciones que determinan el orden de las tareas. Red de Actividad, Grafo dirigido, redes de Petri, Modelos, Hoja de Instrucción serían sinónimos de este concepto.

Tarea: Las tareas son unidades de trabajo formadas por un conjunto de acciones o actividades, que son realizadas por uno o más recursos (un recurso puede ser una persona o una máquina) en un intervalo de tiempo predeterminado. Una tarea es atómica, lo que quiere decir que no es divisible en tareas más pequeñas y que tiene que completarse en su totalidad. Es importante este concepto de tarea ya que para la fase de modelado es necesario tener previamente identificadas las tareas. Esto se logra a través del análisis del proceso como un flujo de trabajo. Podemos considerar los siguientes ejemplos de tareas: evaluar un examen, contestar una llamada telefónica, entre otras.

Las tareas pueden ser manuales, automáticas y semiautomáticas. Las tareas manuales son realizadas exclusivamente por personas. Por el contrario una tarea automática es en la que no existe participación alguna de personas. Mientras que, en las tareas semiautomáticas, se tiene una combinación de las dos anteriores. Algunos sinónimos que pueden ser utilizados para referirse a tareas son los siguientes: paso, nodo, elemento de trabajo, elemento de proceso, operación o instrucción.

Enrutamiento: En un proceso de un flujo de trabajo la información fluye a través de los distintos elementos que forman el proceso. Es así como la información puede tomar diferentes rutas hasta llegar a completarse. Lo anterior es conocido como flujo de control del flujo de trabajo, además este flujo puede fluir en paralelo, es decir, existen varias tareas ejecutándose a la vez, existiendo más de un *hilo de control* (flujos de ejecución). El flujo normal de información, por el que sólo una tarea está en ejecución se llama secuencial. Para representar esas posibles rutas se han definido los siguientes constructores básicos en el modelo de referencia de flujo de control de la *WfMC* [2] y [3]:

Y-Divisor (del inglés AND-Split): Es un punto del flujo de trabajo donde un simple hilo de control se divide en dos o más hilos, los cuales son ejecutados en paralelo dentro del flujo de control. Es decir, a través de cada uno de ellos puede fluir información en un mismo instante permitiendo que múltiples actividades sean realizadas simultáneamente. Esto lo que antes se definía como un flujo paralelo. Cada *Y-Divisor* debe ser complementado con un *Y-Unión*. En algunos sistemas un *Y-Divisor* recibirá hilos que pueden proceder de diferentes *Y-Divisor*.

En la figura 3 se puede observar cómo sería la forma de un *Y-Divisor*.

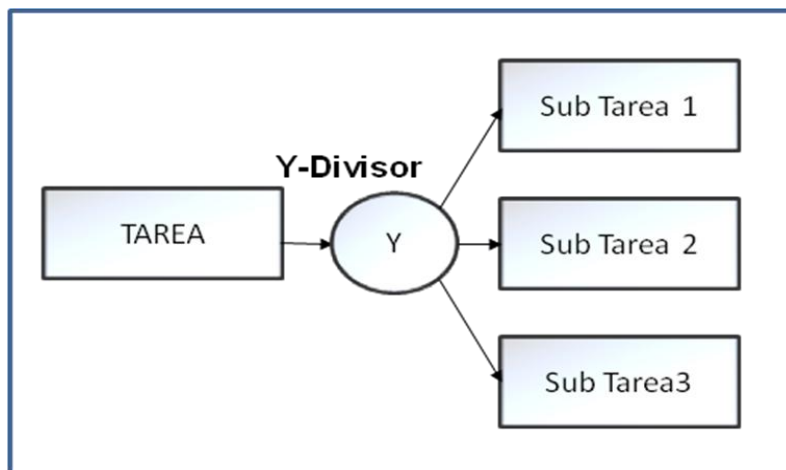


Figura 3. Y-Divisor

O-Divisor (del inglés OR-Split): Es un punto en el flujo de trabajo donde al final de un único hilo de control se toma una decisión con la cual se bifurcará cuando se encuentra con múltiples alternativas. Únicamente podrá tomar una de estas alternativas.

En la figura 4 se muestra como sería el flujo en un O-Divisor.

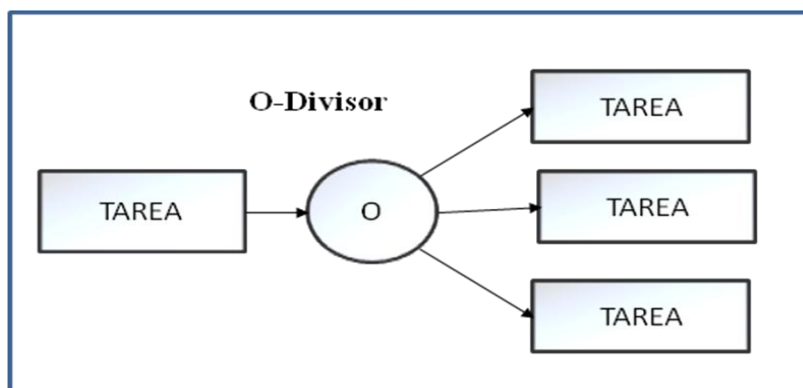


Figura 4. O-Divisor

Y-Union (del inglés AND-Join): Es un punto en el flujo de trabajo donde dos o más actividades que se ejecutan en forma paralela convergen, entonces, gracias a esa convergencia, se puede iniciar otra actividad. Estas actividades que se ejecutan en paralelo pueden proceder uno o varios Y-Divisor.

En la figura 5 se muestra la forma del flujo de control:

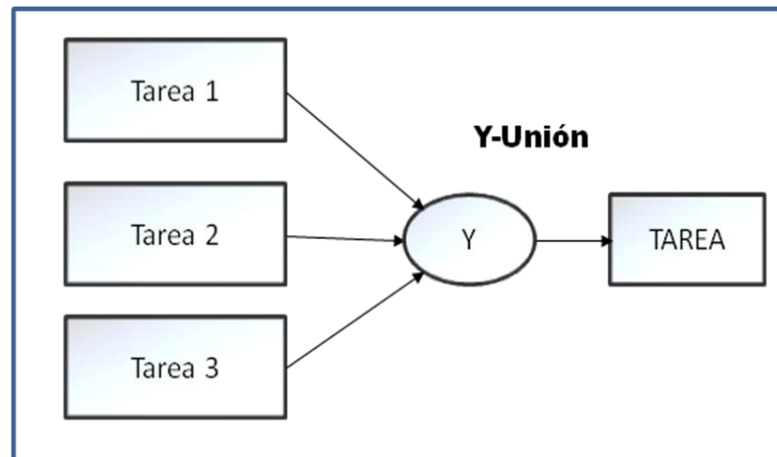


Figura 5. Y-Unión

O-Union (del inglés OR-Join): Es un punto en el flujo de control donde dos o más alternativas para continuar dentro del flujo de control se unen en una actividad simple como el siguiente paso a seguir.

La figura 6 muestra el flujo de control para un O-Union:

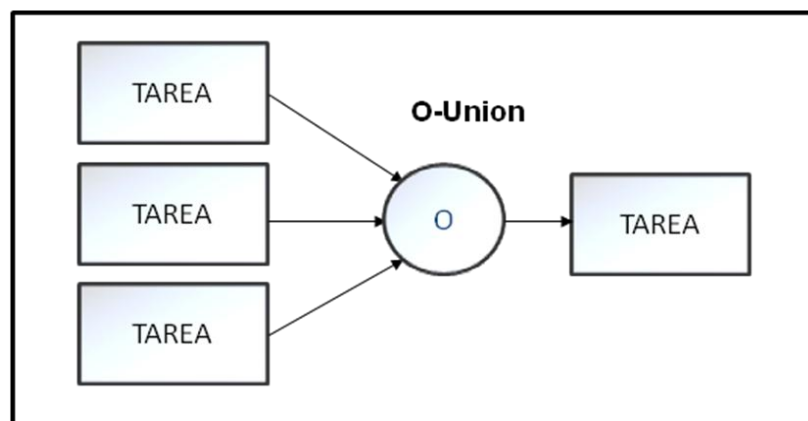


Figura 6. O-Unión

Datos: Los datos son los documentos, archivos, imágenes, registros de la Base de Datos (del inglés Data Bases, BBDD), y otros utilizados como información para llevar a cabo el trabajo. Entre los datos manejados por el flujo de trabajo encontramos los *Datos de Control*, los *Datos de Aplicación* y los *Datos Relevantes*. Los Datos de Control son los datos internos manejados por la lógica del sistema de flujo de trabajo. Es decir, son los datos manejados por el *WfMS* y no suelen ser utilizados por las aplicaciones. Los Datos de Aplicación son los datos específicos de la aplicación, no son utilizados por la lógica de los flujos de trabajo. Generalmente estos datos no suelen ser conocidos por el *WfMS*, por lo que estos datos son administrados por la instancia del proceso que los maneja. Los Datos Relevantes o de Relevancia, son aquellos datos utilizados para determinar el enrutamiento o la dirección de las distintas tareas del sistema, es decir determina el estado al cual tiene que pasar el flujo de trabajo. Estos datos pueden ser pre- y post-condiciones, condiciones de transición o asignadas a los participantes de un flujo de trabajo.

En la figura 7 se muestra donde se emplean cada uno de los tipos de datos en un flujo de trabajo.

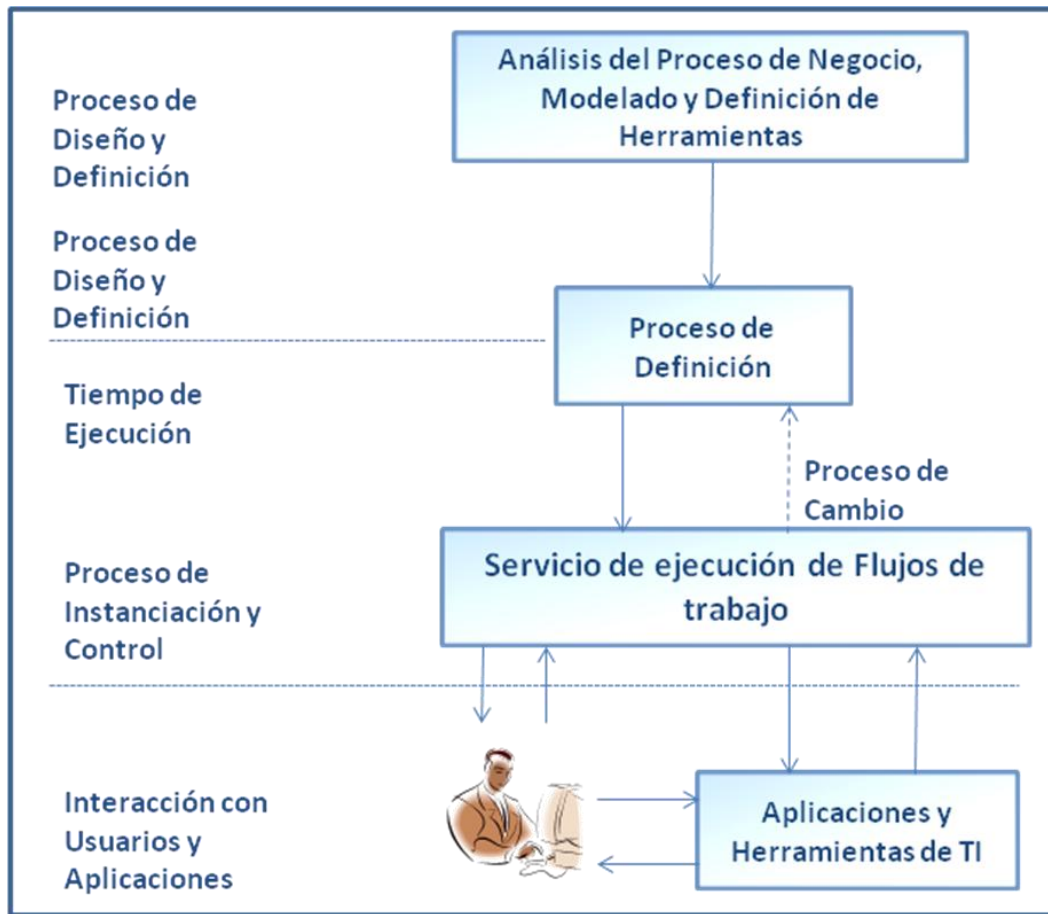


Figura 7. Tipo de Datos y flujo de información en un flujo de trabajo

En cuanto a los datos, es muy utilizada la noción de documento como contenedor de información, que se transmite de una tarea a otra. Por esta razón, cuando se hace referencia a datos manejados por el sistema, es posible referirse a estos datos como documentos. Existen ciertas propiedades que se le pueden asociar a un documento, tales como: la definición de los derechos de acceso a los mismos; las vistas definidas sobre ellos; manejo de accesos concurrentes (es decir, que dos personas o procesos puedan acceder al documento simultáneamente); también es posible definir formas de relacionar datos provenientes de fuentes externas al documento, tales como datos de la aplicación o de la base de datos.

Personas o Participantes del flujo de trabajo: Como se mencionó anteriormente las tareas son realizadas por recursos ya sean máquinas o seres humanos, es así como las personas dentro de una organización son parte importante en lo que a cumplimiento de los procesos se refiere. En muchos procesos las tareas son realizadas en un orden establecido previamente por ciertas personas de acuerdo a su capacidad o rol que desempeñe dentro de la empresa, además de realizarse en base a las condiciones

y reglas de negocio. Una persona es conocida como un actor dentro del escenario de un flujo de trabajo, sin embargo como actor también puede referirse a una máquina. Con lo que se puede definir como actor cualquier cosa o persona que interactúe con el sistema de flujos de trabajo.

Roles: Cada rol define las distintas competencias potenciales que existen en el sistema. Se definen independientemente de los participantes físicos a las cuales se les van a asignar dichos roles. Un participante puede tener más de un rol, siendo un rol un conjunto de características de un actor. En el caso de actores personas, un rol se puede referir a determinadas habilidades, responsabilidades o autoridad que ésta posea.

Grupos: Es un conjunto de actores que realizan el mismo rol. Cuando se asignan tareas a un grupo de actores en vez de a un solo actor, se puede tener mayor flexibilidad en la selección de, quién o qué, puede realizar la tarea, teniendo así una división de las labores más eficiente [5].

Eventos: Un evento es una interrupción que contiene información. Un evento tiene un origen y uno o más destinatarios. La información contenida en el mensaje que se produjo por el evento puede ser implícita o dada por el usuario. Los eventos pueden ser disparados voluntariamente por el usuario; o en forma implícita durante un proceso según el estado de los datos o de decisiones tomadas por el usuario; o en forma automática. Por ejemplo, cuando un gerente de un banco hace una consulta sobre ciertos datos para hacer una auditoria, se dispara un evento que le devuelve la información sobre dicha consulta.

Un evento tiene dos tipos de elementos: *Disparador o Causa* y *Acción Respuesta*. El disparador o causa determina los motivos por los cuales se debe llevar a cabo una acción, en este caso lo que activa un evento. La acción o respuesta es lo que provoca un disparador o causa, es decir es el resultado del evento.

Plazos: Son los tiempos que se le asignan a ciertas tareas para que éstas sean realizadas o estén listas antes de tomar una determinación diferente a la normal, como podría ser su cancelación o la finalización del proceso en sí. Podemos considerar los siguientes ejemplos de plazos: el tiempo máximo que se le asigna a una tarea para que finalice, el tiempo máximo para recorrer una ruta; terminar una tarea antes de cierta fecha, entre otras. A los plazos se les puede asignar eventos, de forma que con el incumplimiento del plazo se dispare dicho evento de forma automática.

Proceso de Definición: La representación de un proceso de negocio de forma que soporte tanto la manipulación automática, como el modelado utilizando el *WfMS*. El proceso de definición consiste en enlazar las actividades y sus relaciones, indicando el comienzo y finalización de los procesos, la información sobre las actividades individuales, sus participantes, asociaciones con las aplicaciones TIs, datos, etc. Se puede hacer referencia a este término cuando se hable de definición del modelo, diagrama de flujo, diagrama de transición de estados, esquema de flujo o escritura del flujo de trabajo [3].

Unidad de Trabajo: Es la representación del trabajo que va a ser realizado en el contexto de una actividad dentro de un proceso. Otros términos que pueden hacer referencia a la unidad de trabajo son: objeto de trabajo, tarea y elemento.

Lista de Trabajo: Conjunto de unidades de trabajo asociados a un participante de flujo de trabajo determinado.

Aplicación Invocadora: Es una aplicación de flujo de trabajo que es invocada por el *WfMS*, para iniciar una actividad, en su totalidad o parte de ella, o para soportar una unidad de trabajo de un participante.

1.2.3 Sistemas de Gestión de Flujos de Trabajo (WfMS)

En el apartado anterior se presentó y describió lo que es un proceso de flujo de trabajo, así como los conceptos que lo conforman. También se hizo una pequeña introducción a lo que es un sistema que ejecuta y administra el proceso del flujo de trabajo. En este apartado se va a definir más detalladamente y a describir las características y arquitectura de estos *WfMS*.

Un sistema gestor de flujos de trabajo es aquel que proporciona una automatización de los procedimientos de un proceso de negocio para el mantenimiento de la secuencia de actividades o tareas y la invocación apropiada de los recursos humanos y tecnológicos asociados a los distintos pasos de cada tarea. El *WfMS* es un sistema que define, crea y gestiona la ejecución de un flujo de trabajo a través de un software, y que permite interpretar el proceso de definición y además interactuar con los participantes del flujo de trabajo y, donde se requiere el uso de las herramientas y aplicaciones de las TIs.

Con lo que se puede decir que un *WfMS* consiste en un conjunto de componentes *SW* que almacenan e interpretan el proceso de definición, creación y mantenimiento de las instancias de los flujos de trabajo, así como su ejecución y el control de su interacción con los participantes y las aplicaciones.

Un Flujo de *WfMS* se caracteriza por dar soporte a tres áreas funcionales distintas:

- ❑ **Funciones en tiempo de diseño y construcción (del inglés Build Time):** Relacionadas con la definición y modelado de cada proceso y actividades que lo constituyen.
- ❑ **Funciones de control en tiempo de ejecución (del inglés Run Time):** relacionadas con la gestión del proceso de flujo de trabajo en el entorno operacional y secuenciando las distintas actividades a ser mejoradas como parte del proceso.
- ❑ **Interacción en tiempo de Ejecución (del inglés Run Time Iteractions):** Entre los usuarios, las aplicaciones y las TIs.

Estas características son reflejadas en los módulos o componentes que forman la arquitectura de un *WfMS*, las cuales se describen en la figura 8.

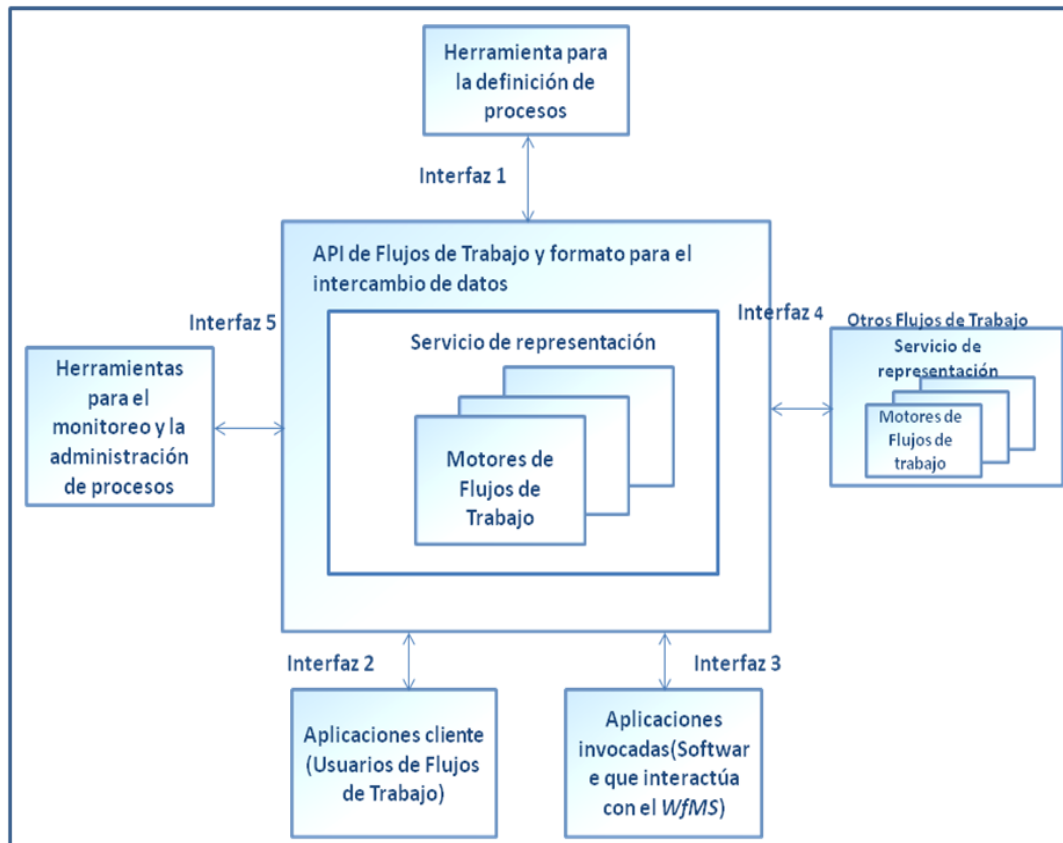


Figura 8. Componentes de la estructura WfMS

1.3 Patrones

La iniciativa de crear *patrones de flujo de trabajo* comenzó con el objetivo de delimitar los requisitos fundamentales que se plantean durante la creación de modelos.

En sí, los patrones sirven para evaluar las capacidades de los diversos sistemas de flujo de trabajo. En este apartado se va a hacer una descripción de los patrones más importantes, para poder aplicarlos más tarde a ejemplos de flujo de trabajo y poder obtener unas conclusiones sobre los distintos lenguajes de modelado.

Como ya se vio anteriormente en el proceso de los *Sistemas de Información* se pueden distinguir distintas perspectivas. La *perspectiva de flujo de control* capta los aspectos relacionados con el control de la información, de las dependencias entre diversas tareas (por ejemplo, el paralelismo, la elección, sincronización, etc). Originalmente se han propuesto veinte patrones para esta perspectiva, pero en la última actualización, han aumentado a más de cuarenta patrones. La perspectiva de los *Datos* se refiere a la transmisión de información, alcance de variables, etc, mientras que la perspectiva de *Recursos* se refiere a recursos para la asignación de tareas, delegación etc. Por último las pautas para el *Manejo de Excepciones* y las distintas acciones que son necesarias adoptar como consecuencia de los posibles problemas que ocurren en forma de excepciones. Este trabajo se va a centrar exclusivamente en la perspectiva de los flujos de control, con lo que se enumerarán los distintos patrones, además de añadir una descripción detallada de cada uno de ellos. De las otras perspectivas, simplemente

se hará una breve descripción general y se enumerarán los patrones sin exponer una descripción de estos ya que si no este documento se alargaría sin ningún fin.

1.3.1 Patrones de Flujo de Control

Son los patrones en los que se centra este trabajo. El objetivo de este apartado es describir la perspectiva de flujos de control. Estos patrones están motivados por el trabajo realizado por la *Iniciativa De Patrones De Flujo De Trabajo* [8] que se describen en el documento [26]. La descripción de cada uno de estos modelos ha sido completamente revisada y evaluada obteniendo el conjunto de patrones final.

En muchos casos, un estudio detallado de un patrón ha indicado que potencialmente hay varias formas en que el patrón original podría ser interpretado y aplicado. Con el fin de resolver estas ambigüedades, en [8] toman la decisión de utilizar la definición revisada, es decir la más restrictiva interpretación de su funcionamiento con el objetivo de limitar otras posibles interpretaciones erróneas del modelo de flujos de trabajo que se quiere diseñar. En el caso en el que fuese necesario utilizar estos escenarios alternativos se presentarán en forma de un nuevo Flujo de Control.

En un esfuerzo por describir las características operativas más rigurosas de cada patrón, en [8] presentan un modelo formal utilizando *Redes de Petri Coloreadas*. Los patrones que se van a describir han sido tomados de la especificación que se hace en [26] y [8]. La *Iniciativa de Patrones para Flujos De Trabajo* [8] definió en un principio un conjunto de veinte patrones que definían el comportamiento básico de flujo de control (estos primeros patrones son los que implementa YAWL). Más adelante tras una segunda revisión, se publicó una relación de los patrones con una descripción más formal (BPMN y UML entran más en detalle y análisis de estos patrones para implementar la mayoría de ellos). Además publicaron veintitrés patrones nuevos que ampliaban la perspectiva del flujo de control permitiendo describir mayor número de situaciones que quieran ser modeladas con flujo de trabajo.

En la figura 9 se muestra los símbolos que utilizan los patrones para representados. Esta simbología es la misma que se utiliza en las redes de Petri según [8]. Aunque no se va a entrar tanto en detalle de los patrones y sólo se va a realizar una descripción textual.

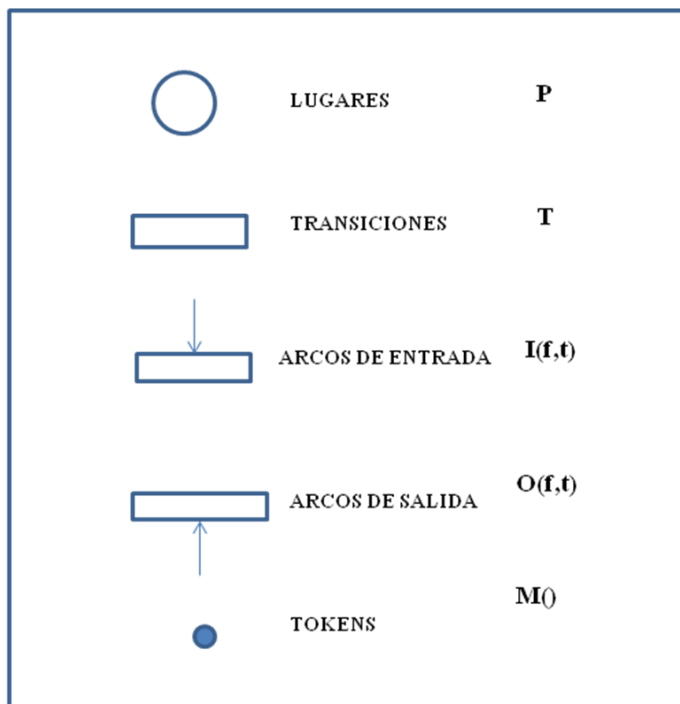


Figura 9. Elementos de un flujo de trabajo basado en patrones

Los patrones de flujo de control se dividen en ocho grupos, los cuales representan las distintas formas que puede tomar el flujo de control en un flujo de trabajo. Estos grupos y sus patrones son los siguientes:

Patrones Básicos de Flujo de Control (del inglés Basic Control Flow Patterns):

- Secuencia (del inglés Sequence)
- División Paralela (del inglés Parallel Split)
- Sincronizador (del inglés Synchronization)
- Elección Exclusiva (del inglés Exclusive Choice)
- Fusión Simple (del inglés Simple Merge)

Patrones Avanzados de Bifurcación y Sincronización (del inglés Advanced Branching and Synchronization Patterns):

- Multi Elección (del inglés Multi-Choice)
- Fusión de Sincronización Estructurada (del inglés Structured Synchronizing Merge)
- Multi-Fusión (del inglés Multi-Merge)
- Discriminador Estructurado (del inglés Structured Discriminator)
- Discriminador de Bloqueo (del inglés Blocking Discriminator)
- Discriminador de Cancelación (del inglés Cancelling Discriminator)
- Unión Parcial Estructurada (del inglés Structured Partial Join)

- Unión Parcial de Bloqueo (del inglés Blocking Partial Join)
- Unión Parcial de Cancelación (del inglés Cancelling Partial Join)
- Unión-Y Generalizada (del inglés Generalised AND-Join)
- Fusión Local de Sincronización (del inglés Local Synchronizin Merge)
- Fusión General de Sincronización (del inglés General Synchronizing Merge)
- Fusión de Hilos (del inglés Thread Merge)
- División de Hilos (del inglés Thread Split)

Patrones de Múltiples Instancias (del inglés Multiple Instance Patterns):

- Múltiples Instancias sin Sincronización (del inglés Multiple Instances without Synchronization)
- Múltiples Instancias con Conocimiento a Priori en Tiempo de Diseño (del inglés Multiple Instances with a Priori Design-Time Knowledge)
- Múltiples Instancias con Conocimiento a Priori en Tiempo de Ejecución (del inglés Multiple Instances wit a Priori Run-Time Knowledge)
- Múltiples Instancias sin Conocimiento a Priori en Tiempo de Ejecución (del inglés Multiple Instances without a Priori Run-Time Knowledge)
- Unión Estática y Parcial para Múltiples Instancias (del inglés Static Partial Join for Muple Instances)
- Unión Parcial de Cancelación para Múltiples Instancias (del inglés Cancelling Partial Join for Multiple Instances)
- Unión Dinámica y Parcial para Múltiples Instancias (del inglés Dynamic Partial Join for Multiple Instances)

Patrones Basados en Estados (del inglés State-based Patterns):

- Elección Aplazada (del inglés Deferred Choice)
- Enrutamiento Paralelo Intercalado (del inglés Interleaved Parallel Routing)
- Hito (del inglés Milestone)
- Sección Crítica (del inglés Critical Section)
- Enrutamiento Intercalado (del inglés Interleaved Routing)

Patrones de Cancelación y Terminación por la Fuerza (del inglés Cancellation and Force Completion Patterns):

- Cancelación de una Tarea (del inglés Cancel Task)
- Cancelación de un Caso (del inglés Cancel Case)
- Cancelación de una Región (del inglés Cancel Region)

- Cancelación de Actividad con Múltiples Instancias (del inglés *Cancel Multiple Instance Activity*)
- Terminación de Actividades de Múltiples Instancias (del inglés *Complete Multiple Instance Activity*)

Patrones de Iteración (del inglés *Iteration Patterns*):

- Ciclos Arbitrarios (del inglés *Arbitrary Cycles*)
- Bucle Estructurado (del inglés *Structured Loop*)
- Recursividad (del inglés *Recursion*)

Patrones de Terminación (del inglés *Termination Patterns*):

- Terminación Explícita (del inglés *Explicit Termination*)
- Terminación Implícita (del inglés *Implicit Termination*)

Patrones Disparadores (del inglés *Trigger Patterns*):

- Desencadenante Disparador Temporal (del inglés *Transient Trigger*)
- Desencadenante Disparador Persistente (del inglés *Persistent Trigger*)

A continuación se van a describir detalladamente los tipos de patrones que hay.

PATRONES BÁSICOS DE FLUJO DE CONTROL

Estos patrones captan los aspectos elementales del control de procesos y son similares a las definiciones de estos conceptos propuestos inicialmente por la *WfMC*[4].

Estos patrones se describen más detalladamente a continuación:

1) Secuencia:

Descripción: Consiste en una tarea que activa otra tarea después de la ejecución de un trabajo anterior o tarea, en el mismo proceso. Es una secuencia de pasos o enrutamiento en serie.

Sinónimos: Encaminamiento secuencial.

Ejemplo: Tarea: Sacar dinero de un cajero:

- I. Paso 1: Introducir la tarjeta en el cajero.
- II. Paso 2: Introducir el número secreto.
- III. Paso 3: Indicar la cantidad a sacar.
- IV. Paso 4: Retirar la tarjeta.
- V. Paso 5: Retirar el dinero.

Motivación: La secuencia es la base fundamental para los procesos. Se utiliza para construir una serie consecutiva de ejecución de tareas una tras otra. Dos

tareas forman parte de una secuencia si un flujo de salida es el flujo de entrada de otra, sin verse afectado por ningún tipo de condición.

Contexto: El contexto de este patrón, está asociado a una condición: una instancia de este patrón no puede iniciarse de nuevo hasta que haya terminado la ejecución del hilo de control anterior.

2) División Paralela:

Descripción: Consiste en la divergencia de una entidad en dos o más ramas o caminos paralelos, cada una de las cuales se puede ejecutar simultáneamente.

Sinónimos: Enrutamiento paralelo, Y-Divisor.

Ejemplo: Una vez el cliente ha pagado por las mercancías y por el envase, entonces se expide el recibo.

Motivación: El patrón de división paralela permite un solo hilo de ejecución, para ser dividido en dos o más ramas de actividad que pueden ejecutar tareas simultáneamente. Estas ramas pueden o no estar sincronizadas en algún momento del futuro.

Contexto: No hay condiciones específicas de contexto para este patrón.

3) Sincronizador:

Descripción: Consiste en la convergencia de dos o más lugares en una sola rama posterior, de tal forma que el hilo de control se pasa a la rama posterior cuando todas las ramas de entrada se han activado.

Sinónimos: Y-Unión.

Ejemplo: El recuento de caja sólo se puede hacer cuando la tienda se ha cerrado y el resumen de la tarjeta de crédito ha sido impreso.

Motivación: La sincronización proporciona un medio de comunicar los hilos de ejecución de dos o más ramas paralelas. En general, estas ramas son creadas usando la división paralela con anterioridad a construir el modelo de proceso. El hilo de control se pasa a la tarea inmediatamente después de que todos los hilos de la sincronización hayan acabado.

Contexto: Para toda señal que se reciba en el lugar, hasta que no haya recibido todas, no pasará al siguiente paso.

4) Elección Exclusiva:

Descripción: Consiste en la divergencia de una sucursal en dos o más ramas de tal manera que cuando el nuevo lugar está habilitado, el hilo de control se pase de inmediato a, precisamente una de las ramas saliente, sobre la base de un mecanismo que puede seleccionar una de las ramas salientes.

Sinónimos: O Exclusivo-Divisor (del inglés XOR-Split), enrutamiento condicional.

Ejemplo: Después de revisar que la tarea de la elección se ha completado, se lleva a cabo, ya sea la tarea de declarar los resultados o la tarea de recuento de votos.

Motivación: El patrón de elección exclusiva permite a los hilos de control ir a una tarea específica dependiendo de la tarea precedente, de los valores de los elementos del proceso, del resultado de una expresión de evaluación o de algún otro mecanismo de selección. La decisión de enrutamiento es dinámica permitiendo ser aplazadas hasta el último momento posible en tiempo de ejecución.

Contexto: Existe una condición para este patrón, el mecanismo que evalúa la elección exclusiva es capaz de acceder a cualquier tipo de datos necesarios, otros elementos o recursos para determinar por cuál rama de las salientes debe dirigirse el hilo de control.

5) Fusión Simple:

Descripción: Consiste en la convergencia de dos o más ramas en una simple rama consecutiva de tal forma que permite que los dos resultados de la rama sean la entrada del hilo de control.

Sinónimos: O Exclusivo- Unión (del inglés XOR-Join), exclusivos o de adhesión.

Ejemplo: Después de la tarea de pago o de crédito, iniciar la tarea de recibir producto.

Motivación: La simple fusión de patrón proporciona un medio de unión de distintas tramas sin necesitar de la sincronización. Permite la simplificación del modelo eliminando la necesidad de replicar explícitamente una secuencia de tareas que es común a dos o más tramas.

Contexto: Hay una condición asociada con la distribución: el lugar donde se produce la fusión es seguro y nunca puede contener más de un token.

PATRONES AVANZADOS DE BIFURCACIÓN Y SINCRONIZACIÓN

Aquí se presentan una serie de pautas que caracterizan una ramificación más compleja y la fusión de conceptos que se plantean en los procesos de negocio. Aunque es relativamente frecuente, en la práctica, estos patrones a menudo no están soportados directamente o incluso muchas herramientas no son capaces de representarlos. Los patrones originales de flujo de control identifican cuatro de estos patrones: Multi-Elección, Fusión Sincronizada, Fusión Múltiple y Discriminador.

Existen catorce patrones para representar paralelismo y sincronización avanzada, que son los siguientes.

6) Multi-Elección:

Descripción: Este Patrón es una variación del patrón de elección exclusiva incluyendo división paralela. Consiste en la divergencia de una rama en dos o más ramas de tal manera que cuando la rama entrante está habilitada, el hilo de control se pasa de inmediato a una o varias de las ramas salientes.

Sinónimos: Encaminamiento condicional, selección, O-Divisor, elección múltiple.

Ejemplo: Dependiendo de la naturaleza de la llamada de emergencia, una o varias tareas de la policía, ambulancia o bomberos se ponen en acción.

Motivación: Este patrón proporciona a un hilo de ejecución la capacidad de divergir en varios hilos concurrentes. La decisión en cuanto a si hay que pasar el hilo de ejecución a una rama u otra se hace en tiempo de ejecución y en la condición de la rama que va a divergir. Puede estar basado en una variedad de factores incluyendo el resultado de una tarea precedente, los valores de los elementos de datos en un proceso, de los resultados de la evaluación de una expresión asociada con la rama saliente o algún otro mecanismo de selección.

Contexto: Hay una condición de contexto asociada a este modelo: el mecanismo que evalúa la multi-elección es capaz de tener acceso a cualquier elemento requerido o recursos necesarios, determinando cuál de las ramas salientes debe ser tomada por el hilo de control.

7) Fusión de Sincronización Estructurada:

Descripción: Este patrón es una combinación entre el patrón anterior multi-elección y el patrón sincronizador. Consiste en la convergencia de dos o más ramas (o entidades) en una sola rama posterior, de tal forma que el hilo de control se pasa a la rama posterior cuando cada rama de los hilos activos que divergen, se activan. Este patrón se produce en un contexto estructurado, es decir, debe existir una única construcción del patrón anterior (multi-elección) con el que este patrón debe estar asociado y que debe fusionar la totalidad de las ramas salientes del patrón de multi-elección.

Sinónimos: Fusión Sincronizada.

Ejemplo: Dependiendo del tipo de emergencia, uno o ambos servicios de policía y ambulancia se inician simultáneamente. Cuando todos los vehículos de emergencia llegan al lugar del accidente, comienza la tarea de traslado de pacientes.

Motivación: Este patrón proporciona el medio de combinar las ramas resultantes del patrón de multi-elección que se construyen en la rama precedente.

Contexto: Hay dos contextos asociados con este patrón: (1) se guarda o memoriza todos los puntos de entrada al patrón sin poder ser modificado hasta que no es reiniciado y (2) una vez que la multi-elección ha sido habilitada

ninguna de las tareas de las ramas que conducen a este patrón pueden ser canceladas. La única excepción a esto es que, es posible para todas las tareas que conducen hasta el patrón para ser cancelado.

8) Multi-Fusión:

Descripción: Consiste en la convergencia de dos o más ramas en una sola rama posterior de tal forma que cada vez que se habilita una rama, los resultados en el hilo de control se transmiten a la siguiente rama.

Sinónimos: No aplica.

Ejemplo: Una cadena de montaje, en la que se obtiene un producto final, como es el lanzamiento de un disco de música a la venta. Un proceso paralelo es la grabación de la maqueta, otro proceso es obtener el elemento físico donde se grabará esa maqueta (un CD, una cinta, etc.) y otro es la fabricación de la carátula.

Motivación: Este patrón proporciona un medio de fusión de diferentes ramas en un proceso de una única rama.

Contexto: Hay una condición asociada a este patrón. Siempre estará precedido por la construcción del patrón multi-elección.

9) Discriminador Estructurado:

Descripción: Este patrón consiste en la convergencia de dos o más entidades en una sola rama posterior, sólo saldrá como resultado de la convergencia el flujo de información de una de las ramas entrantes en el discriminador. Este patrón aparece en entornos estructurados, por ejemplo, debe haber una construcción del patrón división paralela antes del modelo de este patrón con el que se asocia y esto debe combinar todas las ramas que salen del discriminador estructurado. Estas ramas, o bien se derivan de la división paralela al discriminador estructurado sin divisiones o uniones, o tiene que tener una forma estructurada (es decir, equilibrio entre divisiones y uniones).

Sinónimos: Fusión de m-entradas-a-1-salida (del inglés 1-out-of-M join).

Ejemplo: Tareas que puedan finalizar en distintos tiempos, una de ellas continúa gracias al discriminador.

Motivación: Este patrón proporciona un medio de combinar dos o más ramas distintas en una sola rama posterior de tal manera que el primero de ellos se completa en la rama posterior, pero el resto de ramas no tienen ningún efecto sobre dicha rama. Además este patrón proporciona un mecanismo para avanzar la ejecución de un proceso una vez que la primera tarea de un conjunto de tareas concurrentes ha terminado.

Contexto: Existen dos condiciones de contexto asociadas con el uso de este patrón: (1) una vez se ha activado el patrón y si se ha recibido una señal, no

volverá a procesar otra hasta que se reactive y (2) no se podrá cancelar ninguna rama entrante del patrón antes de que este se active o dispare, es decir todas las ramas o tareas entrantes se completaran correctamente.

10) Discriminador de Bloqueo:

Descripción: Este patrón consiste en la convergencia de dos o más ramas procedentes del proceso anterior en una rama posterior. El hilo de control se pasa a la rama posterior cuando la primera subdivisión de las ramas ha sido activada. El bloqueo se produce cuando el discriminador se activa, hasta que el discriminador no finaliza el bloqueo persiste.

Sinónimos: No aplica.

Ejemplo: La tarea de *comprobar cartas de credenciales* puede comenzar una vez, la llegada de *confirmar delegación*, o la tarea de *comprobar seguridad (valor)*, ha finalizado. Aunque estas dos tareas puedan ejecutarse simultáneamente, en la práctica, la tarea de llegada de *confirmar delegación* siempre se finaliza antes de la tarea de *comprobar seguridad (valor)*. Otro caso de la tarea de *comprobar cartas de credenciales* no puede iniciar, si un caso precedente de la tarea aún no ha finalizado. Los casos asimismo subsecuentes de la llegada de *confirmar delegación* y las tareas de *comprobar seguridad (valor)* no pueden ser iniciadas si un caso precedente de la tarea de *comprobar cartas de credenciales* aún no ha finalizado.

Motivación: Este patrón es una variante del modelo del patrón discriminador estructurado que es capaz de funcionar en contextos donde hay varios hilos de ejecución simultáneos (concurrentes) en el mismo caso de proceso. Esta característica permite que se use en bucles y con procesos estructurados donde se pueda recibir más de un hilo de ejecución en el tiempo entre la entrada al discriminador y el bloqueo. Este patrón es más robusto que el patrón discriminador estructurado ya que no está sujeto a que cada rama entrante pueda ser sólo provocada una vez antes del bloqueo. Además mantiene o recuerda el camino de las entradas que han sido desencadenadas y les impide volver a ser activadas hasta que se reinicia el patrón. Otra característica importante es la que permite recibir múltiples disparadores en el mismo lugar de entrada, por ejemplo, donde este patrón se utiliza dentro de un bucle.

Contexto: No hay condiciones específicas asociadas a este patrón.

11) Discriminador de Cancelación:

Descripción: Este patrón es otra variación del discriminador estructurado. Lo que hace es, una vez seleccionada una opción, el resto se eliminan sin llegar a ser evaluadas, es decir se cancelan. El hilo de control se pasa a la rama posterior cuando la primera rama seleccionada de la subdivisión se activa. La activación de este patrón cancela la ejecución de todas las otras ramas entrantes y reinicia el modelo.

Sinónimos: No aplica.

Ejemplo: Después de extraer una muestra de sangre, los distintos componentes de dicha muestra se envían a tres laboratorios distintos para su examen. Una vez que el primero de estos laboratorios completa el análisis de la muestra, la tarea de los otros laboratorios se cancela y continúa el proceso de pruebas.

Motivación: Este modelo proporciona el medio de acelerar un proceso donde una serie de ramas entrantes a una estructura unión tienen que ser sincronizadas pero no es importante que las tareas asociadas a cada rama finalicen.

Contexto: Hay una condición asociada al uso de este patrón. Una vez que la cancelación se ha activado, no es posible recibir otras señales del resto de ramas entrantes.

12) Unión Parcial Estructurada:

Descripción: Es otra variación del discriminador estructurado. En el cuál se puede hacer una combinación de los distintos hilos de entrada. Permite la convergencia de dos o más ramas (m ramas) en una única rama subsecuente después de la divergencia previa correspondiente, tal que el hilo de control se pasa a la siguiente rama cuando se han permitido las n de las ramas entrantes, donde n es menor que m . Este patrón se reinicia cuando todas las ramas entrantes se han habilitado. Se produce en un contexto estructurado, es decir, debe existir una división paralela previa al modelo de este patrón para luego fusionar todas las ramas en una unión.

Sinónimos: No aplica.

Ejemplo: En el proceso de selección de azafatas para una compañía aérea, hay que superar cuatro de las cinco pruebas físicas de selección, para continuar con el proceso. En el momento en el que un candidato apruebe cuatro de las cinco pruebas puede continuar con la prueba escrita.

Motivación: Este patrón proporciona un medio de combinar dos o más ramas resultantes de la división paralela o un Y-Dividido previo en un proceso de flujo de trabajo en una rama simple. La unión no requiere disparadores en cada rama entrante antes de activarse. En cambio tomando un umbral se puede definir las circunstancias en las cuales la unión debería activarse (normalmente esto es presentado como la proporción de ramas entrantes que deben estar activas para habilitar el patrón). Por ejemplo, la unión 2-de-3 significa que se activaría la unión cuando dos de las tres ramas entrantes estén activas. La terminación secuencial de otras ramas entrantes no tendrá ningún efecto sobre la siguiente rama.

Este patrón proporciona un mecanismo para continuar la ejecución de un proceso una vez que un número especificado de tareas simultáneas ha completado sin tener que esperar a que completen todas las tareas.

Contexto: Hay dos condiciones asociadas con el uso de este patrón: (1) una vez que este patrón se ha habilitado y aún no ha acabado, no es posible que reciba otra señal de otra rama y (2) una vez que la división paralela se ha habilitado ninguna de las ramas entrantes a este patrón puede ser anulado antes de que la unión se haya disparado. La única excepción a esto es que es posible que todas las ramas que tienen acceso a la unión parcial se cancelen.

13) Unión Parcial de Bloqueo:

Descripción: Es otra variación del patrón discriminador de bloqueo. Permite la convergencia de dos o más ramas (el número de ramas será m) en una rama posterior después de una o más divergencias previas en el modelo de proceso. El hilo de control pasa a la rama secuencial cuando n de las ramas de entrada se han habilitado (donde $2=n<m$). La unión parcial se reinicia cuando todas las ramas entrantes y activas se han habilitado una vez por proceso. El resto de entradas habilitadas están bloqueadas hasta que la unión parcial acaba.

Sinónimos: No aplica.

Ejemplo: Cuando el primer miembro de la delegación visitante llega, comienza la tarea de comprobar las credenciales. El informe concluye cuando llegan el embajador o el presidente. Debido a limitaciones en el número de personal, sólo se puede llevar a cabo una tarea de comprobación de credenciales a la vez. En el caso en el que llegasen los miembros de otra delegación, la tarea de comprobación de credenciales de esta delegación debería esperar a que la anterior acabase.

Motivación: Este patrón es una variación del anterior unión parcial estructurada que es capaz de ejecutarse en entornos donde los procesos son concurrentes, en concreto donde los procesos tienen varios hilos de ejecución.

Contexto: No hay condiciones específicas para este patrón.

14) Unión Parcial de Cancelación:

Descripción: Es una variación del discriminador de cancelación. Permite la convergencia de dos o más ramas (m) en una rama posterior, después de una o más convergencias previas. El hilo de control se pasa a la siguiente rama cuando n ramas de la entrada se han habilitado, donde $n<m$. Cuando se activa la unión parcial, se cancelan el resto de ramas que están activas.

Sinónimos: No aplica.

Ejemplo: En un examen que consta de tres partes (teoría, problemas y práctica), en el caso en el que no se supere las dos primeras partes (teoría y problemas) el proceso de corrección del examen de prácticas no continúa.

Motivación: Proporciona un medio de agilizar un proceso donde una serie de ramas de entrada necesitan ser sincronizadas pero sólo un subconjunto de las tareas relacionadas finalizan.

Contexto: Hay una condición asociada al uso de este patrón, una vez que la unión parcial se ha activado y no haya acabado, no es posible recibir ninguna otra señal.

15) Y-Union Generalizada:

Descripción: Consiste en la convergencia de dos o más entidades en una rama posterior cuando todas las ramas entrantes se han habilitado. Se pueden recibir desencadenantes adicionales en una o más ramas y son almacenados para continuar más adelante (aunque el tiempo de estos eventos puede variar).

Sinónimos: No aplica.

Ejemplo: Cuando se ha obtenido la firma de todos los directores de una empresa, es decir, la tarea de la firma ha finalizado, se completa la tarea del contrato.

Motivación: Este patrón corresponde con una de las generalidades de la noción de Y-Unión (la otra situación la describe el patrón de sincronización), donde varios caminos de ejecución se sincronizan y combinan en uno. A diferencia del patrón de sincronizador, este patrón soporta que una o varias ramas entrantes puedan recibir varios eventos o disparos en el mismo caso de proceso antes de que Y-Unión se reinicie. La redes de Petri clásicas utiliza una semántica muy parecida a la de este patrón. Esto demuestra que este patrón se puede formular fácilmente. Sin embargo, en la práctica, la semántica tiende a ser poco clara para situaciones en las que el comportamiento no es seguro.

Contexto: No hay condiciones específicas de contexto para este patrón.

16) Fusión Local de Sincronización:

Descripción: Consiste en la convergencia de dos o más ramas de actividad que han divergido en un proceso previo de una sola rama de tal manera que el hilo de control se pasa a la siguiente rama cuando cada rama activa está habilitada. La determinación de cuantas ramas tienen que sincronizarse se hace en base a la información disponible a nivel local para la construcción de la estructura. Esta información puede comunicarse directamente mediante la divergencia previa o, alternativamente, puede ser determinado sobre la base de datos local indicando como los hilos de control llegan a la estructura de fusión.

Sinónimos: No aplica.

Ejemplo: En un proceso paralelo de construcción de un camión de juguete, mientras se construye la cabina del camión y la parte trasera, paralelamente se debe comenzar la unión del eje del camión con las ruedas. Una vez se han finalizado las dos tareas, se procede a unir las dos partes del camión.

Motivación: Este patrón proporciona una semántica determinista para la sincronización que no se basa en un modelo estructurado (como es necesario

para la fusión de sincronización estructurada), sino que tampoco requiere el uso de semántica local en la evaluación.

Contexto: Hay dos condiciones de contexto asociadas con el uso de este patrón: (1) una vez que la estructura del patrón se ha activado y no se haya reiniciado, no es posible recibir otras señales de otras ramas y (2) la estructura del patrón debe ser capaz de determinar cuántas ramas entrantes requiere la sincronización en tiempo de ejecución.

17) Fusión General de Sincronización:

Descripción: Este patrón consiste en la convergencia de dos o más ramas de actividad que previamente divergen en una sola rama y el hilo de control se pasa a la siguiente rama, ya sea cuando (1) cada rama activa se habilita o (2) cuando no es posible que las ramas que todavía no se han habilitado lo hagan en cualquier momento futuro.

Este patrón se puede combinar con el patrón multi-elección, una estructura O Exclusiva, elección aplazada y fusión sincronizada.

Sinónimos: No aplica.

Ejemplo: Cualquier conjunto de subtareas que pueden converger en una tarea o bien cuando están activas o cuando ya no se pueden ejecutar las tareas que faltan por ejecutar.

Motivación: Este modelo ofrece un enfoque general para evaluar la introducción de la sincronización en un proceso. Se puede utilizar de forma no estructurada y para procesos concurrentes que incluyen bucles arbitrariamente.

La dificultad de aplicar este patrón deriva en que su evaluación se basa en el principio de la semántica *No-local* con el fin de determinar cuándo se puede activar. De hecho, es fácil ver que esta construcción puede conducir a la “paradoja del círculo vicioso” donde dos O-Unión depende el uno del otro.

El O-Unión sólo puede ser activado cuando el hilo de control recibe todas las ramas, y esto es así gracias a que el resto de ramas que no se han habilitado nunca podrán hacerlo en cualquier momento posterior. Esto requiere una evaluación de los posibles estados futuros para el proceso actual (caro computacionalmente).

Hay tres cuestiones significativas asociadas con este patrón:

1. A la hora de determinar si un O-Unión debe ser habilitado en un determinado proceso y como deben componerse las tareas precedentes al O-Unión.
2. Como debe ser manejado lo que precede a un O-Unión.
3. Como deben dirigirse todo aquello que afecte a un O-Unión.

Existen documentos que describen la solución a estos problemas, en este documento no se va a entrar en detalle de esta solución.

Contexto: No hay condiciones específicas para este patrón.

18) Fusión de Hilos:

Descripción: Es como si fuera el patrón secuencial, pero solo continúa el hilo en el momento en el que han pasado todas las tareas que se han definido. En un proceso, en un punto dado, un número designado de hilos de ejecución, en una sola rama de la misma instancia de proceso deberían fusionarse en un único hilo de ejecución.

Sinónimos: No aplica.

Ejemplo: Las instancias de la tarea de registro de vehículos se ejecutan de forma independiente unas de otras en la tarea en el proceso de investigación. Los procesos se crean según sean necesarios. Cuando por ejemplo, diez de ellos han terminado, la tarea del proceso de registro de lote se debe ejecutar una vez finalice el sistema de actualización de registros de vehículos.

Motivación: Este modelo proporciona un medio de fusionar múltiples hilos en una instancia de proceso determinada. Es la contraposición al siguiente patrón (*Hilo divisor*), el cual crea múltiples hilos de ejecución de una misma rama.

Contexto: Hay una consideración de contexto para este patrón: el número de hilos (es decir num Items) que se fusionan deber conocerse en tiempo de diseño.

19) División de Hilos:

Descripción: Es la contraposición del anterior, entra un solo hilo y se obtienen tantas tareas como se haya definido. En un punto dado, en un proceso, un número designado de hilos de ejecución puede iniciarse en una sola rama de una misma instancia de proceso.

Sinónimos: No aplica.

Ejemplo: Cualquier tarea que implique en un momento dado dividir el trabajo en varias subtareas.

Motivación: Este patrón proporciona un medio de activación de múltiples hilos de ejecución a lo largo de una rama en una instancia de un proceso determinado. Los nuevos hilos de ejecución se ejecutarán independientemente hasta el final del proceso.

Contexto: Hay una consideración a tener en cuenta en el contexto de este patrón: El número de hilos necesarios (es decir num Items) debe ser conocido en tiempo de diseño.

PATRONES DE MÚLTIPLES INSTANCIAS

Existen muchos ejemplos de patrones que describen situaciones en las que hay múltiples hilos de ejecución activos, en un modelo de proceso que se refiera a la misma

actividad (por lo tanto, comparten la misma definición de implementación). Estos ejemplos se pueden plantear en tres situaciones:

1. Una actividad es capaz de iniciar múltiples instancias de sí misma cuando se activó o disparó (del inglés *trigger*). Se denota esta forma de actividad como *Actividades con múltiples instancias*.
2. Una determinada actividad se inicia varias veces como consecuencia de recibir varios eventos independientes (por ejemplo, como parte de un bucle o en un proceso, por ejemplo, en los que hay varios hilos simultáneos de ejecución, como podría ser el resultado de una fusión).
3. Dos o más actividades en un proceso comparten la misma definición de implementación. Esto puede ser, la misma definición de actividad en el caso de una instancia múltiple de una actividad o un sub-proceso de definición en el caso de un bloque de actividad. Dos (o más) de estas actividades se activan de tal forma que sus ejecuciones se superponen (ya sea parcial o totalmente).

A pesar de todas estas situaciones que pueden implicar múltiples instancias concurrentes de una actividad o sub-proceso, es el primero de ellos el más importante y el que más interesa en este documento, ya que requieren la activación y sincronización de múltiples instancias de actividad concurrentes. Este grupo de patrones se centra en las diversas formas en que estos acontecimientos pueden ocurrir.

Los patrones que se definieron para estas situaciones son los siguientes:

20) Múltiples Instancias sin Sincronización:

Descripción: Este patrón permite que en una instancia de un proceso cree varias instancias de ese mismo proceso. Estas instancias son independientes unas de otras. Cada una de las instancias creadas deben ejecutarse en el contexto del proceso que las creo (es decir, comparten el mismo identificador de caso y tiene acceso a los mismos datos) y como ya se ha dicho antes, cada uno de ellos debe ejecutarse independientemente sin referenciar a la tarea que ellos comenzaron.

Sinónimos: Múltiples hilos no sincronizados.

Ejemplo: Se reciben una lista de infracciones de tráfico (multas) de la Dirección General de Tráfico (DGT). Por cada infracción en la lista se crea una tarea de notificación de infracción. Estas tareas se realizan en paralelo pero no desencadenan las posteriores tareas. Y no necesitan ser sincronizadas para finalizar.

Motivación: Este modelo ofrece un medio de crear múltiples instancias de una determinada tarea. El número de tareas nuevas a crear debe ser conocido previamente. Estas tareas se pueden ejecutar de forma independiente unas de otras y la sincronización posterior no es obligatoria.

Contexto: Hay una condición asociada a este patrón: el número de instancias de tareas que se van a crear (es decir, numInst) debe conocerse en tiempo de diseño y es un valor fijo.

21) Múltiples Instancias con Conocimiento a Priori en Tiempo de Diseño:

Descripción: Permite dentro de una instancia de proceso crear una o varias instancias de una tarea. El número de instancias deberá ser determinado en tiempo de diseño. Estas instancias son independientes una de otras y se ejecutan de forma concurrente. Es necesario que las instancias se sincronicen para finalizar antes de que comience la siguiente tarea.

Sinónimos: No aplica.

Ejemplo: Anualmente una empresa emite un informe sobre las actividades desarrolladas en la empresa ese año y el estado actual de la empresa. Antes de hacer público dicho informe, los seis principales directores de la empresa deben firmarlo.

Motivación: Este patrón proporciona la base para la ejecución concurrente de una tarea, un número predefinido de veces. También asegura que todas las instancias de dichas tareas se completan antes de la ejecución de la siguiente tarea.

Contexto: Hay una condición asociada con este patrón: el número de tareas (es decir, numInst) debe conocerse en tiempo de diseño y debe ser un valor fijo.

22) Múltiples Instancias Con Conocimiento a Priori en Tiempo De Ejecución:

Descripción: Dentro de una instancia de proceso, puede crear varias instancias de tareas. El número de instancias depende de una serie de factores de ejecución, incluidos los datos, la disponibilidad de los recursos y la comunicación entre procesos, pero esta información se debe conocer antes de que se creen las instancias de la tarea. Una vez iniciados, estas instancias son independientes unas de otras y se ejecutan simultáneamente.

Sinónimos: No aplica.

Ejemplo: El sistema de diagnóstico de un coche puede obtener los fallos de un motor. Cuando esto sucede, varias instancias de la tarea de comprobación de sensores se ejecutan simultáneamente. El número de tareas depende del número de mensajes de error que ha obtenido el diagnóstico. Sólo cuando todos los mensajes se han tratado, se puede determinar el porqué del fallo.

Motivación: Este patrón proporciona un medio de ejecutar múltiples instancias de una determinada tarea de manera sincronizada, conociendo en tiempo de ejecución, exactamente cuántas instancias se crearan. Este conocimiento se conoce justo en el instante previo a que se inicie la primera tarea.

Contexto: Hay una condición de contexto asociada con este patrón. El número de instancias de tareas (es decir, numInst) se conoce en tiempo de ejecución y con anterioridad a la creación de instancias de la tarea. Una vez determinada, el número de instancias de tareas tiene un valor fijo.

23) Múltiples Instancias Sin Conocimiento A Priori En Tiempo De Ejecución:

Descripción: Dentro de una instancia de proceso, se pueden crear varias instancias de una tarea. El número de casos puede depender de una serie de factores en tiempo de ejecución, incluidos los datos, la disponibilidad de los recursos, la comunicación entre instancias, etc. Pero esto no se conoce hasta el último momento. Una vez iniciadas, estas instancias son independientes unas de otras y se ejecutan simultáneamente. Además en cualquier momento, mientras las instancias se están ejecutando, es posible añadir instancias adicionales, mientras no existan más de las definidas. Es necesario sincronizar la terminación de las instancias creadas antes de que se habilite cualquiera de las instancias de otras tareas que deben ejecutarse a continuación.

Sinónimos: No aplica.

Ejemplo: El envío de petróleo procedente de una torre de perforación desde las fábricas, implica numerosas tareas de transporte. Estas tareas de transporte se producen simultáneamente y aunque se inician tareas suficientes para cubrir la demanda estimada de envíos de petróleo, siempre es posible hacer más envíos de los estimados, que se iniciarán si hay un déficit en las necesidades del transporte. Una vez que todo el petróleo, procedente de todas las torres petroleras, ha sido transportado, y todas las tareas de envío han finalizado, la próxima tarea puede comenzar (ensamblar torre).

Motivación: Este patrón es una extensión del patrón múltiples instancias con conocimiento a priori en tiempo de ejecución, que remite a la necesidad de determinar cuántas instancias de tareas concurrentes son necesarias en el último momento. Ofrece más flexibilidad en qué pueden crearse instancias adicionales “sobre la marcha” sin la necesidad de cambios en el modelo de proceso o la sincronización de las condiciones de las tareas.

Contexto: Hay una condición de contexto asociada a este patrón. Es el número de instancias de tareas (es decir numInst) que debe ser conocido, en tiempo de ejecución, con anterioridad a la finalización de la tarea de múltiples instancia. Hay que tener en cuenta que no es necesario, conocer el número final de instancias, al inicializar la tarea de múltiples instancias.

24) Unión Estática y Parcial para Múltiples Instancias:

Descripción: Dentro de una instancia de proceso, se pueden crear una tarea de múltiples instancias concurrentes (se representa como m). El número de instancias se conoce cuando la primera instancia de la tarea comienza. Una vez que n de las instancias de tareas se han completado (donde $n < m$), la siguiente

tarea del proceso puede comenzar. Una vez han finalizado n de las m instancias, el resto de instancias son intrascendentes, sin embargo todas las instancias deberán concluir con el fin de que el patrón pueda volver a ser utilizado en el modelo.

Sinónimos: No aplica.

Ejemplo: Examinar diez muestras de la línea de producción, en busca de defectos. Continuar con la siguiente tarea en el momento en el que siete de estos exámenes se han completado.

Motivación: Este patrón es una extensión del patrón *Unión Estática y Parcial para Múltiples Instancias*. Este patrón permite que la instancia de proceso general continúe una vez se hayan completado n de las múltiples instancias, en lugar de exigir que todas las instancias terminen.

Contexto: Existen dos condiciones de contexto asociadas con este patrón. (1) El número de instancias concurrentes en la tarea se conoce antes de que la tarea comience y (2) el número de tareas que es necesario que se complete para que continúe el proceso, y se conoce en el momento previo del inicio de la tarea.

25) Unión Parcial de Cancelación para Múltiples Instancias:

Descripción: Es parecido al anterior, se diferencia en que las instancias pueden ser canceladas. Más concretamente una vez que una de las instancias de la tarea ha terminado, la siguiente tarea en el proceso es disparada y el resto de instancias se cancelan.

Dentro de una instancia de proceso se puede crear una tarea con múltiples instancias (m) concurrentes. El número de instancias que se necesita se conoce cuando la primera instancia de la tarea comienza. Cuando n de las instancias de la tarea se ha completado (donde $n < m$), la siguiente tarea del proceso puede comenzar o dispararse y el resto de las tareas ($m-n$) se cancelan.

Sinónimos: No aplica.

Ejemplo: Se ejecutan 500 instancias de la tarea de test de un patrón con distintas muestras. Una vez que 400 de estas tareas han terminado, el resto de las 500 instancias (es decir 100) que no han llegado a comenzar, se cancelan y se pone en marcha la siguiente tarea del proceso.

Motivación: Este modelo ofrece una variante del patrón *Múltiples Instancias con Conocimiento a priori en tiempo de ejecución*. Esta variante permite más rendimiento ya que cancela las instancias permitiendo que el proceso continúe evitando la necesidad de dedicar más esfuerzo a la ejecución de las instancias que no han concluido y que ya no son necesarias.

Contexto: Este modelo comparte dos condiciones de contexto con el patrón *Unión Estática Y Parcial Para Múltiples Instancias*. (1) El número de instancias concurrentes en la tarea se conoce antes de que la tarea se habilite, (2) el número

de tareas que se necesitan para finalizar y que comience la siguiente tarea del proceso también es conocido antes de que se habilite la tarea con las múltiples instancias.

26) Unión Dinámica y Parcial para Múltiples Instancias:

Descripción: Dentro de una instancia de proceso, múltiples instancias de una tarea pueden ser creadas de forma concurrente. El número de instancias que se requieren depende de una serie de factores en tiempo de ejecución, (es decir no se conocen hasta el final, hasta que la última instancia ha concluido) como son el estado de los datos, la disponibilidad de los recursos y la comunicación entre los distintos procesos. En cualquier momento, mientras que las instancias se ejecutan, es posible que para las instancias adicionales que se han iniciado no se deshabiliten. La condición para completar o finalizar se evalúa cada momento en el que una instancia de tarea se completa. Una vez que la condición que se evalúa es cierta, se activa la siguiente tarea. La finalización de las tareas pendientes es intrascendente y no permite que se creen nuevas instancias.

Sinónimos: No aplica.

Ejemplo: El transporte de una torre de perforación de petróleo desde la fábrica al lugar donde debe realizarse la perforación implica numerosas tareas de transporte del envío. Estas tareas son concurrentes y aunque se inician suficientes tareas para cubrir la estimación inicial del volumen de transporte, siempre es posible añadir tareas adicionales que se iniciarán si hay un déficit en las necesidades de transporte. Una vez que el 90% de las tareas de transporte se completan, la próxima tarea (facturar los gastos de transporte) puede iniciarse. El resto de tareas de transporte continúan hasta que todo el aparejo se ha transportado.

Motivación: Este patrón es una variante del patrón de *Múltiples Instancias Sin Conocimiento a Priori en Tiempo de Ejecución* que proporciona la posibilidad de desencadenar la próxima tarea una vez que la condición de terminación se cumple.

Contexto: Este patrón tiene dos condiciones de contexto: (1) el número de instancias de tareas que pueden ejecutarse concurrentemente debe ser conocido antes de que la tarea comience y (2) debe ser posible acceder a los elementos de datos y a los recursos necesarios para evaluar la condición de finalización para completar todas las instancias de cada tarea.

PATRONES BASADOS EN ESTADOS

Los patrones basados en estados se corresponden a situaciones en las que su solución se acopla mejor para lenguajes que soportan la noción de estado. En este contexto, se considera que el estado de una instancia de proceso incluye la amplia recopilación de datos relacionados con la ejecución actual, incluido el estado de las diversas actividades, así como los procesos de trabajo relevantes, los datos de actividad y elementos de casos de datos.

El patrón original incluye tres patrones en los que el estado actual es el principal determinante en el curso de una acción que será tomada en la perspectiva de un flujo de control. Estos son: *Elección aplazada*, donde la decisión sobre qué entidad tomar, se basa en la interacción con el entorno operativo, el patrón, *Encaminamiento paralelo permitido*, cuando dos o más secuencias de actividades intercaladas se llevan a cabo de tal forma que sólo una instancia, puede ejecutarse en un momento dado, el patrón, *Hito*, donde permite que una determinada actividad sólo se produzca cuando el proceso se encuentra en un estado específico.

Existen cuatro nuevos modelos para este tipo de patrones, y se enumeran a continuación:

27) Elección Aplazada:

Descripción: Donde la decisión sobre qué sucursal tomar, se basa en la interacción con el entorno operativo. Antes de la decisión, todas las ramas representan posibles cursos de ejecución futuros. La decisión se realiza al iniciar la primera tarea en una de las ramas, es decir, no hay elección explícita, sino más bien una carrera entre las diferentes ramas.

Sinónimos: Elección Externa (del inglés External Choice), Elección Implícita (del inglés Implicit Choice) y O Exclusivo Dividido.

Ejemplo: En el comienzo del proceso de resolución de queja, hay una elección entre la tarea de contacto inicial con el cliente y la tarea del gerente. El contacto inicial con el cliente se inicia cuando comienza el servicio de atención al cliente por un miembro del equipo. La tarea del gerente o manager comienza 48 horas después de que el proceso anterior comience. Una vez que una de estas tareas inicia, el otro se retira.

Motivación: Este patrón permite la posibilidad de aplazar el momento de la elección de un proceso, es decir, el momento en el que uno de los distintos y posibles cursos de acción o ramas deben ser elegidos, se retrasa lo más posible hasta el último momento y además depende de factores externos a la instancia de proceso (por ejemplo, mensajes, datos sobre el entorno, la disponibilidad de recursos, tiempos, etc). Hasta el punto en el que se toma la decisión, cualquiera de las alternativas viables representa cursos de acción futura.

Contexto: Hay una condición de contexto asociada con este patrón, sólo una instancia de este patrón puede operar en el tiempo.

28) Enrutamiento Paralelo Intercalado:

Descripción: El conjunto de tareas tiene un orden parcial, que define los requerimientos, en relación con el orden en que deben ser ejecutadas las tareas. Cada tarea del conjunto debe ser ejecutada una vez y pueden concluir en cualquier orden que concuerde con el orden parcial definido. Sin embargo dos

tareas no pueden estar activas en la misma instancia de proceso al mismo tiempo.

Sinónimos: No aplica.

Ejemplo: Al despachar un pedido, la tarea de selección de mercancías, y la preparación del paquete deben completarse. Las tareas de seleccionar y recoger la mercancía deben realizarse antes de la tarea de preparación de paquete. La tarea de preparar la factura puede ocurrir en cualquier momento. Sólo una de estas tareas se puede hacer en cualquier momento y en un orden determinado.

Motivación: Este patrón ofrece la posibilidad de hacer más flexible el orden estricto que un proceso impone a un conjunto de tareas. Hay que notar que este patrón está relacionado con el patrón de exclusión mutua, es decir, es como un semáforo, se encarga de que las tareas no se ejecuten al mismo tiempo sin aplicar un orden particular.

Contexto: Hay una condición asociada con este patrón: las tareas deben ser iniciadas y completadas secuencialmente, no es posible suspender una tarea durante su ejecución para trabajar en otra.

29) Hito:

Descripción: Este patrón permite que una determinada tarea se ejecute cuando el proceso se encuentra en un estado específico. Es decir, una tarea sólo se activa cuando la instancia del proceso (de la que forma parte) se encuentra en un estado específico (normalmente una rama paralela). El estado se supone que es un punto de ejecución (también conocido como un hito) en el modelo de proceso. Cuando se alcanza este punto de ejecución, la tarea nominada puede comenzar. Si la instancia de proceso ha progresado más allá de este estado, entonces la tarea asociada al estado no puede iniciarse en ningún momento (es decir, ha vencido el plazo). La ejecución no influye en el propio estado.

Sinónimos: Arco de prueba, plazo, condición de estado estacionario.

Ejemplo: La mayoría de líneas aéreas permiten realizar una reserva que debe permitir cambiar el billete si este no ha sido expedido. La tarea de inscribir a los estudiantes solo se puede ejecutar al mismo tiempo que la tarea de aceptar nuevas inscripciones. Esto es después de que la tarea de matrícula abierta ha terminado y antes de que la tarea de cierre de matrícula comience.

Motivación: Este patrón proporciona un mecanismo de soporte a la tarea de ejecución condicional de una tarea o sub-proceso donde la instancia de proceso está en el estado correcto. El concepto de estado en general se entiende en el sentido de que el control de flujo ha llegado a un punto designado en la ejecución de la instancia del proceso (es decir, un hito). Como tal proporciona un medio de sincronización entre dos ramas distintas de una instancia de proceso, de manera

que una rama no puede continuar o proceder a menos que el otro grupo o ramas haya llegado a un determinado estado.

Contexto: No existen condiciones específicas para el desarrollo de este patrón.

30) Sección Crítica:

Descripción: Este patrón proporciona la capacidad para prevenir la ejecución concurrente de partes específicas. Dos o más subregiones conectadas de un modelo de proceso se identifican como “*secciones críticas*” y no pueden ejecutarse en paralelo con otras regiones o secciones críticas. En tiempo de ejecución para una determinada instancia de proceso, sólo determinadas tareas pueden estar activas en uno de estos “*puntos críticos*”. Una vez que la ejecución de las tareas en un punto crítico comienza, se debe completar antes de que otro punto crítico pueda comenzar.

Sinónimos: No aplica.

Ejemplo: Tanto la tarea de obtención de un depósito, el pago del seguro y las tareas en el proceso de reserva de vacaciones exigen el uso exclusivo de la tarjeta de crédito en el momento de pago. Por consiguiente, sólo uno de ellos puede ejecutarse en un momento dado.

Motivación: El uso de este patrón proporciona un medio de limitar la ejecución de dos o más secciones en un proceso. En general esto es necesario si las tareas dentro de una sección requieren acceso exclusivo a un recurso común (ya sea de datos o de un recurso físico), necesario para una tarea. Sin embargo también hay situaciones de control o reglamento, que requieren dos tareas que no se producen simultáneamente.

Contexto: Para toda aquella sección crítica, no podrá ser suspendida su ejecución, para la ejecución de otra en ninguno de los casos.

Criterio de Evaluación: En algunas herramientas se puede conseguir una funcionalidad similar a través de programas adicionales de configuración o a través de la extensión de sus actuales construcciones.

31) Enrutamiento Intercalado:

Descripción: Este patrón recoge situaciones en las que un grupo de actividades se pueden ejecutar secuencialmente en cualquier orden. Cada miembro de un conjunto de actividades debe ser ejecutado una sola vez. Se pueden ejecutar en cualquier orden, pero no hay dos tareas que se puedan ejecutar al mismo tiempo.

Sinónimos: No aplica.

Ejemplo: El control de aceite, los tests de alimentación o revisiones, el estudio de las cantidades y las tareas de garantía deben llevarse a cabo como parte del proceso de producción de un producto alimenticio. Sólo uno de ellos puede llevarse a cabo a la vez, sin embargo, se puede ejecutar en cualquier orden.

Motivación: Amplia la aplicación del patrón de Enrutamiento Paralelo Intercalado y permite la ejecución en secuencia de tareas en cualquier orden.

Contexto: Las tareas deben iniciarse y finalizarse de forma secuencial, en particular, no es posible suspender una tarea durante su ejecución para trabajar en otra.

PATRONES DE CANCELACIÓN Y TERMINACIÓN POR LA FUERZA

Este tipo de patrones proporcionan la cancelación de tareas de distintas formas. Existen varias modalidades de cancelación que se describen a continuación en forma de patrones:

32) Cancelación de una Tarea:

Descripción: Una tarea puede ser cancelada antes de que comience su ejecución. Si la tarea ha comenzado, no es posible finalizar la ejecución de la tarea y, cuando sea posible la instancia actual en ejecución se detendrá y se eliminará.

Sinónimos: Retirar tarea.

Ejemplo: La tarea de acceso a los daños está a cargo de los asesores de seguros. Una vez que la primera tarea de evaluación ha finalizado, la segunda se cancela.

Motivación: Este patrón proporciona la posibilidad de finalizar una tarea que se ha activado o que ya se está ejecutando. Esto asegura que no va a comenzar o terminar la ejecución.

Contexto: No hay condiciones específicas de contexto para este patrón.

33) Cancelación de un Caso:

Descripción: Consiste en que una instancia de un proceso completo, se elimina. Esto incluye la ejecución actual de tareas, las que puede realizar en el futuro y todos los sub-procesos. La instancia de proceso, se registra como que ha terminado sin éxito.

Sinónimos: Retirar caso.

Ejemplo: Durante una solicitud de hipoteca, el comprador decide no continuar con la compra de una casa y se retira la solicitud.

Motivación: Este modelo proporciona un medio de detener una instancia de proceso y la retirada de las tareas asociadas a ella.

Contexto: Existe una condición muy importante de contexto asociada con este patrón: la cancelación de la ejecución de un caso debe ser considerada como si hubiera finalizado. Esto significa que aunque el caso se dio por terminado de forma ordenada, quizá incluso llego a su finalización, esto no debe interpretarse en modo alguno como un resultado exitoso. Por ejemplo, si se mantiene un registro de los acontecimientos ocurridos durante el proceso de ejecución, el caso debe registrarse como incompleto o cancelado.

34) Cancelación de una Región:

Descripción: Este patrón tiene la capacidad de desactivar un conjunto de tareas en una instancia de un proceso. Si alguna de las tareas de ejecución ya se ha ejecutado, entonces el resto finalizan.

Una de las cuestiones que puede surgir con la aplicación de este patrón, se produce cuando la tarea se encuentra dentro de la región. A pesar de que la tarea debe ejecutarse hasta el final y provocar la cancelación del resto de tareas de la región, una vez que se ha completado, esta tarea también debe cancelarse.

La solución más eficaz a la cuestión anterior es asegurar que la tarea de cancelación es la última de las que se procesan (es decir, el último en ser terminado) en la región de cancelación, y finaliza. La cancelación real se produce cuando la tarea asociada a la región de cancelación finaliza su ejecución.

Sinónimos: No aplica.

Ejemplo: Eliminar todas las tareas del proceso de reserva de la ruta de un viaje, después de la tarea de presentación de los posibles transportes.

Motivación: La opción de poder cancelar una serie de tareas (potencialmente no relacionadas) es una capacidad muy útil, en particular para el manejo de errores inesperados o para la implementación de formas de manejar las excepciones.

Contexto: No existen condiciones específicas asociadas con este patrón.

35) Cancelación de Actividad con Múltiples Instancias:

Descripción: Dentro de una instancia de un proceso se pueden crear varias instancias de una tarea. El número necesario de casos se conocen en tiempo de diseño. Estos casos son independientes uno de otro y se ejecutan simultáneamente. En cualquier momento, la instancia de múltiples tareas puede ser cancelada y todos los casos que no han terminado se deben retirar o cancelar. Los casos que ya se han completado no se ven afectados.

Sinónimos: No aplica.

Ejemplo: Ejecutar 500 casos de prueba de la tarea de test de proteínas con distintas muestras. Si no ha terminado una hora después de la apertura, se cancelan.

Motivación: Este patrón proporciona un medio de cancelar múltiples instancias de una tarea en cualquier momento de la ejecución, de manera que los restantes casos se cancelan. Sin embargo, los casos que ya han finalizado, no se ven afectados por la cancelación.

Contexto: Hay una condición de contexto asociada con este patrón: se supone que sólo una instancia de cada una de las tareas se ejecuta para un determinado proceso en un momento determinado.

36) Terminación de Actividad de Múltiples Instancias:

Descripción: Dentro de una instancia de proceso se pueden crear varias instancias de una tarea. Es necesario conocer el número de instancias en tiempo de diseño. Estos casos son independientes unos de otros y pueden ser concurrentes. Es necesario sincronizar las instancias antes de que cualquier tarea posterior se pueda activar. Durante el curso de la ejecución es posible que la tarea sea lo que fuerce a completar las restantes instancias para que se retiren y que el hilo de control se transmita a posteriores tareas.

Sinónimos: No aplica.

Ejemplo: Ejecutar 500 casos de prueba de la tarea de test de proteínas con distintas muestras. Si no ha terminado una hora después de la apertura, retirar el resto de casos, e iniciar la siguiente tarea.

Motivación: Este patrón proporciona un medio para la finalización de una tarea que tiene múltiples instancias y que todavía no han terminado, de manera que las restantes instancias, se retiran y el hilo de control se pasa inmediatamente a las siguientes tareas. Los casos que ya han finalizado no se ven afectados por la cancelación.

Contexto: La condición de contexto asociada a este patrón es que sólo una instancia, de las múltiples instancias, de una tarea puede ejecutarse en cualquier momento.

PATRONES DE ITERACIÓN

Estos patrones se encargan del comportamiento repetitivo en un flujo de trabajo. Los patrones de iteración son los siguientes:

37) Ciclos Arbitrarios:

Descripción: Patrón que tiene la capacidad de representar los ciclos en un modelo de proceso que tiene más de una entrada o salida. Utiliza principalmente la estructura O Exclusivo. Además debe ser posible para los distintos puntos de entrada y salida estén asociados con diferentes entidades.

Sinónimos: Bucle no estructurado, iteración, ciclo.

Ejemplo: La comprobación de redundancia cíclica (CRC) es un tipo de función que recibe un flujo de datos de cualquier longitud como entrada y devuelve un valor de longitud fija como salida. El CRC es un código de detección de error cuyo cálculo es una larga división de computación. La mecánica de la informática con su lenguaje binario produce unas CRC simples. Los bits representados de entrada son alineados en una fila, y el (n+1) representa el patrón de bits del divisor CRC (llamado polinomio) se coloca debajo de la parte izquierda de la fila. Si la entrada que está por encima del extremo izquierdo tiene como divisor 0, no hace nada y pasar el divisor a la derecha de uno en uno. Si la

entrada que está por encima de la izquierda tiene como divisor 1, el divisor es O Exclusivo en la entrada. El divisor es entonces desplazado hacia la derecha, y el proceso se repite hasta que el divisor llega a la derecha, en la parte final de la fila de entrada.

Motivación: El patrón de Ciclos Arbitrarios proporciona un medio para soportar la repetición dentro de un mismo modelo de forma no estructurada, sin necesidad de especificar operadores para bucles o añadir restricciones a la estructuración general del modelo de proceso.

La única consideración que hay que tener en cuenta, es que el modelo debe soportar ciclos.

Contexto: No hay ninguna condición específica asociada a este patrón.

38) Bucle Estructurado:

Descripción: Este patrón posee la capacidad de ejecutar una tarea o sub-proceso varias veces. El bucle tiene ya sea, un pre-test o post-test asociados a la condición que va a ser evaluada al principio o al final del bucle para determinar si debe continuar. El bucle tiene una estructura única de entrada y salida.

Sinónimos: No aplica.

Ejemplo: Mientras la máquina de producción de cajas tenga combustible, continuará con el proceso de producción. Sólo se planificarán vuelos si la tarea de predicción de tormenta es propicia.

Motivación: Existen dos formas generales para este modelo,

- El bucle *mientras* (del inglés *while*) que equivale a la forma clásica *mientras... hacer* (del inglés *while...do*), es la prueba previa al bucle que se usa en los lenguajes de programación, este bucle permite repetir la ejecución de una tarea secuencialmente, una o varias veces, dependiendo de una condición que se evalúa a verdadero. La prueba, evalúa la condición antes de la primera iteración del bucle y vuelve a evaluarla antes de cada iteración. Una vez que la condición se evalúa a falso, el hilo de control pasa independientemente a la tarea que continúa después del bucle.
- El bucle *repetir* (del inglés *repeat*) que equivale a la forma *repetir... hasta* (del inglés *repeat...until*), que es la prueba que se realiza después del bucle. Esta estructura permite repetir la ejecución de una tarea o subproceso una o más veces, continuando con la ejecución hasta llegar a la condición que se evaluará a verdadero. La prueba que evalúa la condición se realiza después de la primera iteración del bucle y vuelve a evaluarla al final de cada iteración. Una vez que la prueba de condición evalúa a cierto o verdadero, el hilo de control pasa a la tarea inmediatamente posterior al bucle.

Contexto: Sólo existe una condición de contexto asociada a este patrón. Sólo una instancia del bucle puede estar activa en un momento determinado.

39) Recursividad:

Descripción: La recursividad es la capacidad que tiene una tarea de invocarse a sí misma durante su ejecución o de invocar a una tarea anterior en los términos de la descomposición de la estructura con la que está asociada.

Sinónimos: No aplica.

Ejemplo: En una planta de fabricación de automóviles, la tarea de resolver defectos se inicia para cada problema mecánico que se identifica en un automóvil. Si durante la ejecución de esta tarea se detecta un fallo mecánico y no está relacionado con el problema actual, otra tarea se inicia para resolver dichos fallos mecánicos. Estas sub-tareas pueden a su vez iniciar otras sub-tareas si en su ejecución encuentran algún defecto. Pero lo que es más importante, la tarea principal, la que se inicio al principio, no puede finalizar hasta que las sub-tareas iniciadas durante su ejecución, finalicen.

Motivación: Para algunos tipos de tareas, el uso de la recursividad en vez de la iteración, proporciona soluciones más sencillas y concisas. Cuando se está resolviendo un problema con recursividad, es obligatorio describir la ejecución de la tarea en términos de sí misma, es decir, la tarea que va a llamarse a sí misma tiene que estar bien y completamente definida.

Contexto: No hay condiciones de contexto asociadas con este patrón.

PATRONES DE TERMINACIÓN

Son aquellos patrones que hacen referencia a las circunstancias en que un flujo de trabajo se considera terminado.

40) Terminación Implícita:

Descripción: Este patrón consiste en que cuando si a una instancia de un determinado proceso (o subproceso) no le quedan elementos de trabajo capaces de realizar alguna actividad en cualquier momento, dicha instancia del proceso debe finalizar. Es una forma objetiva para determinar si el proceso ha finalizado satisfactoriamente.

Sinónimos: No aplica.

Ejemplo: En un sistema de producción de cajas de cartón, en el momento en el que no queda material (láminas de cartón), se para el proceso de producción.

Motivación: La justificación para este modelo es que representa el enfoque más realista para determinar cuándo una instancia de un proceso ha finalizado. Esto se produce cuando no hay ninguna actividad pendiente y no existe posibilidad que los elementos de trabajo se ejecuten en algún momento.

Contexto: No hay ninguna condición de contexto asociada a este patrón.

41) Terminación Explícita:

Descripción: La instancia de un proceso (o sub-proceso) debe terminar cuando se alcanza un estado designado. Normalmente este estado se denota como nodo final. Cuando se llega a este nodo final, cualquier trabajo o actividad pendiente en el proceso es anulado y el proceso general finaliza satisfactoriamente independientemente de si había tareas en curso o pendientes ejecución.

Sinónimos: No aplica.

Ejemplo: En una fábrica de coches, salé una nueva gama y deciden realizar 100 productos de esta gama. En el momento en el que cumplen este objetivo, dejan de producir dicha gama de coches.

Motivación: La justificación para este patrón, es que representa una alternativa de definir cuando una instancia de proceso puede declararse como finalizada. Esto es cuando el hilo de control llega a un estado ya definido (nodo final) en el modelo de proceso. Donde hay un único nodo final en el proceso, su inclusión en otras composiciones se simplifica.

Contexto: Existe una condición asociada con este patrón: toda tarea del proceso debe estar en el camino entre el nodo inicial y el nodo final.

PATRONES DESENCADENANTES

Los siguientes patrones se encargan de hacer frente a las señales externas que puedan ser necesarias para iniciar ciertas tareas. Dichos patrones son los siguientes:

42) Disparador Temporal:

Descripción: Es la capacidad para que la instancia de una tarea sea activada por una señal de otra parte del proceso o de una señal del exterior. Estos factores o eventos (disparadores) desencadenantes son de naturaleza transitoria y se pierden si no se produce una recepción inmediata por parte de la tarea. Un disparador sólo puede existir si hay una tarea a la espera en el momento en el que debe recibirse.

Sinónimos: No aplica.

Ejemplo: La tarea de comprobar sensor, debe iniciarse cada vez que la alarma de incendios se haya disparado.

Motivación: Los disparadores transitorios son un medio común de señalización que define los eventos que tiene que ocurrir y se debe comprobar qué se ha producido una respuesta adecuada, ya sea el inicio de una tarea, una secuencia de tareas o un nuevo hilo de ejecución en un proceso. Los disparadores son eventos transitorios que deben ser tratados tan pronto como se hayan recibido. En otras palabras, deben dar inmediatamente inicio a una tarea. Si no se actúa de inmediato, el disparador inmediatamente se pierde.

Contexto: No existe ninguna condición de contexto asociada a este patrón.

43) Disparador Persistente:

Descripción: Es la capacidad de una tarea para ser activada por una señal de otra parte del proceso o del exterior. Estos factores desencadenantes son persistentes en su forma y son retenidos por el proceso hasta que puedan ser recibidos por la tarea.

Sinónimos: No aplica.

Ejemplo: Se debe iniciar una nueva instancia de la tarea de inspección de vehículos para cada señal de servicio que se retrasa.

Motivación: Los disparadores persistentes son de naturaleza duradera, asegurando que la señal no se pierda hasta que no es recibida por una tarea, conservándose en memoria. Esto significa que la tarea que recibe la señal puede estar segura de que el disparador enviará la señal correspondiente.

Contexto: No hay ninguna condición de contexto asociada a este patrón.

1.3.2 Patrones de Datos

Los flujos de trabajo tratan de aportar un vehículo para aplicaciones complejas y procesos de negocio recurrentes. No obstante lo dispuesto en este objetivo común, hay una variedad de rasgos comerciales distintivos ofrecidos por los *WfMS*. Estas diferencias se observan en la capacidad de las distintas herramientas para representar y poner en práctica la multitud de necesidades que puedan surgir en los procesos de negocio. Muchos de estos requisitos se repiten con bastante frecuencia durante la fase de análisis de requisitos para los flujos de trabajo y las abstracciones de estos requisitos sirven como medio para determinar los componentes de las clases de los lenguajes de los flujos de trabajo. En el apartado anterior se ha identificado una serie de patrones para los flujos de trabajo de control, a continuación se describirá una serie de patrones de datos de flujos de trabajo que tienen por objeto captar las distintas formas en que los datos son representados y cómo son utilizados en flujos de trabajo. Estos patrones se van a especificar en un formato independiente de las tecnologías específicas de flujos de trabajo y de lenguajes de modelado, de forma que se puede proporcionar un tratamiento integral de la perspectiva de datos de flujos de trabajo y posteriormente utilizar estos patrones para la base de la comparación.

Desde la perspectiva de los datos, hay una serie de características que se producen repetidamente en los diferentes paradigmas de modelos de flujos de trabajo. Estas características se pueden dividir en cuatro grupos distintos:

- *Visibilidad de los datos:* Relativo a la medida y la forma en que elementos de datos pueden ser vistos por distintos componentes de un proceso de flujos de trabajo.

- *Interacción de datos:* Se centra en la manera en que los datos se comunican entre elementos activos dentro de un flujo de trabajo.
- *Transferencia de datos:* Considera el medio por el cual la transferencia real de los elementos de información se produce entre los componentes del flujo de trabajo y describe los diferentes mecanismos mediante los cuales los elementos de datos pueden transmitirse a través de la interfaz de un componente de un flujo de trabajo. Es un complemento a la característica anterior, con lo que los patrones que la representan no van a ser citados.
- *Base del encaminamiento de datos:* Caracteriza la forma en que elementos de datos pueden influir en el funcionamiento de otros aspectos del flujo de trabajo, en particular la perspectiva de Flujo de Control.

Cada una de estas características se analizan con más detalle mediante patrones en la siguientes secciones. Esta especificación se ha obtenido de [23].

PATRONES DE VISIBILIDAD DE DATOS (del inglés Data Visibility)

En el contexto de un motor de WFs, hay una variedad de maneras distintas en las que los elementos de datos pueden ser definidos y utilizados. Normalmente, estas variaciones se refieren a la forma en que se declaran y el método de construir el WF relacionado. Más importante aún, cómo influyen directamente en la forma en que el elemento de datos puede utilizarse por ejemplo para captar la producción de información, para gestionar el control de datos o para la comunicación con el entorno externo. A continuación se enumeran estos patrones, pero sin hacer una descripción más exhaustiva.

Los patrones de Visibilidad de datos son los siguientes:

- Grupo de Datos (del inglés Task Data)
- Bloque de Datos (del inglés Block Data)
- Ámbito de Aplicación de Datos (del inglés Scope Data)
- Múltiples Instancias de Datos (del inglés Multiple Instance Data)
- Caso de Datos (del inglés Case Data)
- Carpeta de Datos (del inglés Folder Data)
- Datos de Flujo de Datos (del inglés Workflow Data)
- Entorno de Datos (del inglés Environment Data)

PATRONES DE INTERACCIÓN DE DATOS (del inglés Data Interaction)

En este apartado se va a examinar brevemente las diversas formas en que los elementos de datos pueden transmitirse entre los componentes en un proceso de WF y

cómo las características de los componentes individuales pueden influir en la manera en que se produce el tráfico de elementos de datos. Es de particular interés la distinción entre la comunicación de datos entre componentes en un motor de WF y la comunicación de los elementos de WF con el entorno externo.

Estos patrones se engloban en dos grupos, Interacción interna de datos y Interacción externa de datos. A continuación se enumeran los distintos patrones que los componen.

A) Interacción interna de datos (del inglés Internal Data Interaction):

- Interacción Interna de Datos (del inglés Internal Data Interaction)
- Interacción de Datos - Tarea a Tarea (del inglés Data Interaction - Task to Task)
- Interacción de Datos - Bloque de Tareas a Sub Flujos de Trabajo (del inglés Data Interaction - Block Task to Sub-Workflow Decomposition)
- Interacción de Datos - Sub Flujos de Trabajo a Bloque de Tareas (del inglés Data Interaction - Sub-Workflow Decomposition to Block Task)
- Interacción de Datos - A Múltiples Instancias de Tarea (del inglés Data Interaction - to Multiple Instance Task)
- Interacción de Datos - De Múltiples Instancias de Tarea (del inglés Data Interaction - from Multiple Instance Task)
- Interacción de Datos - De Caso a Caso (del inglés Data Interaction - Case to Case)

B) Interacción Externa De Datos (del inglés External Data Interaction):

- Interacción de Datos - De la tarea al entorno - Apilar (del inglés Data Interaction - Task to Environment - Push-Oriented)
- Interacción de Datos - Del entorno a la tarea - Desapilar (del inglés Data Interaction - Environment to Task - Pull-Oriented)
- Interacción de Datos - Del entorno a la tarea - Apilar (del inglés Data Interaction - Environment to Task - Push-Oriented)
- Interacción de Datos - De la tarea al entorno - Desapilar (del inglés Data Interaction - Task to Environment - Pull-Oriented)
- Interacción de Datos - Del caso al entorno - Apilar (del inglés Data Interaction - Case to Environment - Push-Oriented)
- Interacción de Datos - Del entorno al caso - Desapilar (del inglés Data Interaction - Environment to Case - Pull-Oriented)
- Interacción de Datos - Del entorno al caso - Apilar (del inglés Data Interaction - Environment to Case - Push-Oriented)

- Interacción de Datos - Del caso al entorno - Desapilar (del inglés Data Interaction - Case to Environment - Pull-Oriented)
- Interacción de Datos - Del Flujo de Trabajo al Entorno - Apilar (del inglés Data Interaction - Workflow to Environment - Push-Oriented)
- Interacción de Datos - Del entorno al Flujo de Trabajo - Desapilar (del inglés Data Interaction - Environment to Workflow - Pull-Oriented)
- Interacción de Datos - Del entorno al Flujo de Trabajo - Apilar (del inglés Data Interaction - Environment to Workflow - Push-Oriented)
- Interacción de Datos - Del Flujo de Trabajo al Entorno - Desapilar (del inglés Data Interaction - Workflow to Environment - Pull-Oriented)

PATRONES BASADOS EN LA TRANSFERENCIA DE DATOS (del inglés Data Transfer Patterns)

- Transferencia de Datos por Valor - Entrada (del inglés Data Transfer by Value - Incoming)
- Transferencia de Datos por Valor - Salida (del inglés Data Transfer by Value - Outgoing)
- Transferencia de Datos - Copia Entrante/Copia Saliente (del inglés Data Transfer - Copy In/Copy Out)
- Transferencia por Referencia - Sin Bloqueo (del inglés Data Transfer by Reference - Unlocked)
- Transferencia por Referencia - Con Bloqueo (del inglés Data Transfer by Reference - With Lock)
- Transformación de Datos - Entrada (del inglés Data Transformation - Input)
- Transformación de Datos - Salida (del inglés Data Transformation - Output)

PATRONES BASADOS EN EL ENCAMINAMIENTO DE DATOS (del inglés Data-based Routing)

A continuación se enumeran los patrones que representan las distintas formas en que los elementos de datos pueden interactuar con otras perspectivas y cómo influyen sobre el funcionamiento de una instancia de un proceso.

- Precondición de Tarea - Existencia de datos (del inglés Task Precondition - Data Existence)
- Precondición de Tarea - Valor de Datos (del inglés Task Precondition - Data Value)
- Postcondición de Tarea - Existencia de datos (del inglés Task Postcondition - Data Existence)

- Postcondición de Tarea - Valor de Datos (del inglés Task Postcondition - Data Value)
- Basado en Eventos de Activación de Tareas (del inglés Event-based Task Trigger)
- Basado en Dato de Activación de Tareas (del inglés Data-based Task Trigger)
- Enrutamiento Basado en datos (del inglés Data-based Routing)

1.3.3 Patrones de Recursos

No basta simplemente con centrarse en el control y la corriente de datos a la hora de capturar los procesos de negocio, los recursos permitidos que puedan ser necesarios deben ser considerados también. Toda la información que abajo se detalla se ha obtenido de [25].

En este apartado se va a describir la perspectiva de recursos de forma más abreviada, ya que como se ha dicho este documento se va a centra en los patrones de Flujo de Control. La perspectiva de los recursos se centra en la modelización de los recursos y la interacción con los procesos del sistema. Los recursos pueden ser humanos (por ejemplo, un trabajador) o no humanos (por ejemplo, planta y equipo), aunque los recursos que se van a describir se centrará en los recursos humanos. Se va a describir una serie de patrones de flujos de trabajo de recursos que tiene por objetivo captar las diversas formas en que están representados los recursos y como se utilizan en dichos flujos de trabajo.

A continuación se enumeran los patrones de recursos.

PATRONES DE CREACIÓN (del inglés Creation)

La creación de patrones corresponde a las limitaciones sobre la manera en que un elemento de trabajo puede ser ejecutado. Estas se especifican en tiempo de diseño, por lo general están relacionados con una tarea, y sirven para restringir el rango de recursos al que tienen acceso la tarea que corresponden a la tarea. También influye en la manera en que un elemento de trabajo puede ir acompañado de un recurso.

La justificación para los patrones de creación es que proporcionan un grado de claridad acerca de cómo una tarea debe ser manipulada después de su creación durante la asignación de etapas antes de que sea ejecutada. De esta forma se garantiza que la operación de un proceso se ajusta a sus principios de diseño y funciona de la forma más eficiente y determinista posible.

De los patrones de creación se han seleccionado aquellos que se han considerado más importantes por la función que ejercen. Son los siguientes:

- Distribución Directa (del inglés Direct Distribution)
- Distribución Basada en Roles (del inglés Role-Based Distribution)

- Distribución Aplazada (del inglés Deferred Distribution)
- Autorización (del inglés Authorization)
- Serpación de Responsabilidades (del inglés Separation of Duties)
- Manejo de Casos (del inglés Case Handling)
- Retener Parentesco (del inglés Retain Familiar)
- Basado en la Capacidad de Distribución (del inglés Capability-Based Distribution)
- Basado en el Historial de Distribución (del inglés History-Based Distribution)
- Organización de Distribución (del inglés Organisational Distribution)
- Ejecución Automática (del inglés Automatic Execution)

PATRONES DE APILAMIENTO (del inglés Push Patterns):

Estos patrones representan situaciones donde proactivamente se crean nuevas unidades de trabajo (tareas, etc) o el sistema es el que asigna los recursos. Esto puede ocurrir indirectamente mediante advertencias de las unidades de trabajo a determinados recursos mediante una lista de trabajos compartida o directamente con unidades de trabajo que están asignadas a determinados recursos. Sin embargo en ambos casos, es el sistema quien toma la iniciativa y provoca el proceso de distribución.

A continuación se enumeran los distintos patrones:

- Distribución por Oferta - Recurso Simple (del inglés Distribution by Offer - Single Resource)
- Distribución por Oferta - Recursos Múltiples (del inglés Distribution by Offer - Multiple Resources)
- Distribución por Asignación - Recurso Individual (del inglés Distribution by Allocation - Single Resource)
- Asignación Aleatoria (del inglés Random Allocation)
- Vuelta Aleatoria (del inglés Round Robin Allocation)
- Cola Corta (del inglés Shortest Queue)
- Principios de Distribución (del inglés Early Distribution)
- Capacidad de Distribución (del inglés Distribution on Enablement)
- Distribución Tardía (del inglés Late Distribution)

PATRONES DE RETIRADA O DESAPILAMIENTO (del inglés Pull Patterns):

Estos patrones representan la situación en la que recursos individuales están relacionados directamente con alguna unidad o unidades de trabajo específicos, que requieren ejecución, mediante bien una herramienta directa desde el sistema o indirectamente por una lista de unidades de trabajo compartida. El compromiso de realizar una tarea específica o de estar asociado a una tarea es de los propios recursos en vez de por el sistema.

A continuación se enumeran los distintos patrones:

- Iniciación de Asignación de Recursos (del inglés Resource-Initiated Allocation)
- Inicio de Ejecución de Recursos - Asignación Inicial de Trabajo (del inglés Resource-Initiated Execution - Allocated Work Item)
- Inicio de Ejecución de Recursos - Oferta de Trabajo (del inglés Resource-Initiated Execution - Offered Work Item)
- Sistema Determinado por el Trabajo contenido en la Cola (del inglés System-Determined Work Queue Content)
- Recursos Determinados por el trabajo contenido en la Cola (del inglés Resource-Determined Work Queue Content)
- Autonomía de Selección (del inglés Selection Autonomy)

PATRONES DE DESVÍO (del inglés Detour Patterns):

Estos patrones se refieren a situaciones donde la distribución de recursos a Work Items que se ha hecho se interrumpe, ya sea por el sistema o provocado por los recursos. Como consecuencia de este evento, la secuencia normal de transiciones de estados para este Work Item cambia.

- Los patrones de desvío son los siguientes:
- Delegación (del inglés Delegation)
- Escalado (del inglés Escalation)
- Desasignación (del inglés Deallocation)
- Reasignación de Estado (del inglés Stateful Reallocation)
- Desasignación de Estado (del inglés Stateless Reallocation)
- Reanudación-Suspensión (del inglés Suspension-Resumption)
- Saltar (del inglés Skip)
- Reahacer (del inglés Redo)

- Deshacer (del inglés Pre-Do)

PATRONES DE AUTOCOMIENZO (del inglés Auto-Start Patterns):

Estos patrones se refieren a la situación en la cual la ejecución del Work Item es activada por eventos específicos en el ciclo de vida del Work Item o relacionados con el proceso de definición. Tales eventos pueden incluir la creación o la asignación de un Work Item, completar otra instancia del mismo Work Item o un Work Item que precede inmediatamente al anterior.

- Los patrones de auto comienzo son los siguientes:
- Comienzo de Creación (del inglés Commencement on Creation)
- Comienzo de Asignación (del inglés Commencement on Allocation)
- Pila de Ejecución (del inglés Piled Execution)
- Cadena de Ejecución (del inglés Chained Execution)

PATRONES DE VISIBILIDAD (del inglés Visibility Patterns):

Los patrones de visibilidad clasifican los diversos ámbitos de disponibilidad y responsabilidad de los Work Items están en condiciones de ser vistas por los recursos.

Los patrones que implementan estas funciones son los siguientes:

- Desasignación Configurable de Visibilidad de Trabajos (del inglés Configurable Unallocated Work Item Visibility)
- Asignación Configurable de Visibilidad de Trabajos (del inglés Configurable Allocated Work Item Visibility)

PATRONES DE MÚLTIPLES RECURSOS (del inglés Multi Resource Patterns):

- Ejecución Simultanea (del inglés Simultaneous Execution)
- Recursos Adicionales (del inglés Additional Resources)

A continuación se van a describir distintos lenguajes de modelado de flujos de trabajo.

1.4 Lenguajes De Modelado Para Representar Procesos De Negocio

El modelado de procesos de negocio es también conocido como especificación de flujos de trabajo. Para modelar un flujo de trabajo es necesario un lenguaje o herramienta de modelado ya sea para modelado en general o para modelar específicamente flujos de trabajo. De estos lenguajes o técnicas de modelado podemos obtener una clasificación que nos permita observar sus ventajas y desventajas. En primer lugar pueden ser clasificadas en dos grupos: *Formales* y *No formales*. A partir de

dicha clasificación podemos hacer una distinción entre las técnicas que ofrecen un ambiente gráfico para el modelado y las que no lo ofrecen.

La *formalidad* es un principio que tradicionalmente no es bien acogido en el campo de los sistemas de información. Muchos lenguajes han sido propuestos sin tener fundamentos formales. Con el tiempo ha venido a ser claro que está no es una situación del todo deseable. Las técnicas de modelado formales, descansan sobre bases matemáticas que permiten que el modelo obtenido sea validado tanto sintácticamente como semánticamente, ésta es la principal ventaja ya que permite que un modelo sea analizado y verificado, asegurando así en mayor grado la ausencia de errores. Su principal desventaja, es que son técnicas que pueden requerir un poco de más conocimiento o aprendizaje por parte de la persona que modela los procesos.

Ejemplos de este tipo de técnicas son:

- Redes de Petri
- Máquinas de Estados
- Lógica Temporal

Con las técnicas *no formales* se tiene prácticamente lo contrario, es decir, el uso de éstas suele resultar de mayor facilidad y más comprensión al no estar atadas a restricciones matemáticas. Sin embargo, la principal desventaja es el no poder verificar los modelos obtenidos, con lo que es necesario esperar hasta tener el sistema desarrollado e implementado para verificar que todos sus módulos funcionan como se modelaron a través de alguna de las técnicas informales. Lo anterior por supuesto, no es recomendable ya que incrementa los costes de desarrollo.

Ejemplos de técnicas no formales son los siguientes lenguajes:

- UML
- Diagramas de flujo
- Diagramas IDEF
- BPMN

Los lenguajes sin una semántica formal son fácilmente susceptibles de ser inherentemente ambiguos. Una solución que en ocasiones es adoptada, es la de modelar los procesos utilizando una técnica informal para después mediante algún algoritmo de conversión, generar la representación en alguna técnica formal, para de esa manera validar el modelo. Sin embargo, además de tener como resultado un trabajo extra, es muy difícil que el modelo obtenido represente de manera correcta el modelo original.

Después de esta breve introducción a lo que son los lenguajes de modelado y en los dos tipos en los que se pueden clasificar, se va a llevar a cabo una descripción más detallada de los lenguajes de modelado más empleados.

En primer lugar se va a describir dos de los lenguajes más utilizados y que forman parte de los lenguajes no formales. Estos son Notación para el Modelado de Procesos de Negocio o BPMN (del inglés *Business Process Modeling Notation*) y

Lenguaje Unificado de Modelado o UML (del inglés *Unified Modeling Language*). Estos son los lenguajes más utilizados hasta el momento.

Hablando de estos dos lenguajes, la *Iniciativa para el Soporte de Procesos de Negocio* [11] (BPMI, del inglés Business Process Management Initiative) es una organización sin ánimo de lucro que tiene la misión de promover el desarrollo y uso de la administración de procesos de negocio a través del establecimiento de estándares para el diseño, distribución, ejecución, soporte y optimización de procesos.

El Grupo de Gestión de Objetos (*OMGTM*, del inglés Object Management Group) [16] es un consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI o CORBA. Es otra organización sin ánimo de lucro. El grupo está formado por compañías y organizaciones de software como: HP, IBM, Sun Microsystems, Apple Computer.

En Junio de 2005 el BPMI y el *OMGTM* anunciaron su fusión y combinación de sus actividades de gestión de procesos de negocio para proporcionar una dirección de pensamiento y una industria de estándares para el crecimiento de esta industria. Se unieron para ello bajo el nombre de *Business Modeling & Integration DTF* [17]. En esta unión confluyen por un lado una organización con gran experiencia en el modelado de procesos de negocio y con una notación ampliamente difundida, BPMN (del inglés Business Process Modeling Notation), y por otro lado la organización internacional más importante a la hora de crear estándares relacionados con la ingeniería del software. Uno de esos estándares, el más difundido, es UML, una de cuyas partes, los Diagramas de Actividad, tienen entre sus objetivos el modelado de procesos de negocio. Por tanto encontramos dos especificaciones con propósitos similares dentro de la misma organización.

El objetivo conjunto de estas organizaciones es desarrollar un soporte para la gestión de especificaciones y modelos integrados de empresas u otros entornos. Estas especificaciones promocionaron la integración y coalición de inter/intra empresa con las personas, los sistemas y la información de dicha empresa incluyendo los procesos de los socios y los clientes.

Por la parte de lenguajes de modelado formales, se va a describir el lenguaje YAWL, que está basado en las redes de Petri, y que hasta aparecer los dos anteriores, era el más utilizado para modelar procesos de negocio.

1.4.1 BPMN

Muchas personas piensan que el siguiente paso al “boom” del flujo de trabajo en los 90’s fue el “*Soporte de Procesos de Negocio*” (BPM, del inglés *Business Process Management*) [20].

El “*Soporte de procesos de negocio es el encargado del soporte de procesos de negocio usando métodos, técnicas y software para diseñar, representar, controlar y analizar los procesos operacionales que involucran organizaciones, aplicaciones,*

documentos y otras fuentes de información”[20]. Existen muchas definiciones del soporte de procesos de negocio, pero en la mayoría de los casos estas definiciones dejan claro su relación con el *WfMS*. La definición que se acaba de hacer restringe el soporte de procesos a procesos operacionales.

El soporte debe tener unos fuertes cimientos, es decir:

- Los modelos formales no se pueden permitir ninguna posibilidad de que exista alguna ambigüedad,
- y deben aumentar, en lo posible, su potencial de análisis.

Para todo ello el soporte de los procesos debe tener un proceso bien definido que incluya lenguajes de modelado, técnicas, metodologías, sistemas de seguridad (BPMS, del inglés *Business Process Management System*), etc.

A continuación se entra en detalle en la parte de modelado que gestiona el BPMS, que es BPMN.

1.4.1.1 Introducción

Tradicionalmente el modelado de procesos de negocio se ha utilizado en la industria para obtener una visión global de los procesos mediante actividades de soporte, control y monitorización y para otras actividades como el procesado automático de documentos. Sin embargo, durante los últimos años el trabajo de investigación en este campo ha aumentado enormemente y con ello las aplicaciones de esta técnica que se ha empleado en campos tan diversos como:

- La planificación de recursos empresariales (ERP).
- La integración de aplicaciones empresariales (EAI).
- La gestión de flujos de trabajo (WFM).
- La gestión de las relaciones con los clientes (CRM).
- La comunicación entre usuarios del proceso de negocio para facilitar la gestión de requisitos.

Como consecuencia del creciente interés en las técnicas de modelado de procesos de negocio, en Junio del 2005 la BPMI entró a formar parte de la OMG y fruto de esta unión se publicó en febrero del año 2006 la primera versión 1.0 de BPMN [12] (este documento se basa en la versión 1.2 de BPMN). Pese a que según los propios autores “*BPMN se centra en los procesos de negocio y los Diagramas de Actividad de UML se centran en el diseño de software y por tanto no son competidoras, sin diferentes puntos de vista sobre un sistema*”, lo cierto es que en la práctica los diagramas de actividad ya se estaban usando para el modelado de procesos de negocio y que en la actualidad, tras la absorción, todo parece indicar que la propia OMG ha relegado a sus propios diagramas de actividad, eligiendo BPMN para el modelado de procesos de negocio. Como indicador de este movimiento destacar que otras

especificaciones de la propia OMG, como por ejemplo SPEM (del inglés *Software Process Engineering Metamodel 2.0*), que usaba diagramas de actividad para describir comportamientos dinámicos deja abierta, una nueva versión, la elección de notación sugiriendo el uso de BPMN. Todo esto no sólo debe entenderse como una cesión a la organización que ha sido absorbida, sino que se hace necesario un análisis de las notaciones para ver cuál es la más adecuada para la actividad que ocupa este documento, el modelado de procesos de negocio.

Finalmente, se obtuvo la Notación para el Modelado de Procesos de Negocio (BPMN, del inglés *Business Process Model Notation*) y una Definición del Metamodelo de Procesos de Negocio (BPDM, del inglés *Business Process Definition MetaModel*) que en estos momentos está en desarrollo y que será el metamodelo de BPMN. En los siguientes apartados se va a hacer una descripción del metamodelo que el BPMI está desarrollando para la representación de procesos de negocio, en forma de flujos de trabajo y en qué consiste la notación para el modelado de procesos de negocio.

1.4.1.2 Definición del Metamodelo de un Proceso de Negocio

La definición del metamodelo de un proceso de negocio o BPDM (del inglés *Business Process Definition Metamodel*), es un metamodelo que están definiendo para BPMN. En [19] dice: *“BPDM define un metamodelo de negocio comprensible independiente de cualquier lenguaje de modelado; traza múltiples lenguajes incluyendo BPMN y XMI (Intercambio de Metadatos, del inglés Metadata Interchange) que proporciona modelado y capacidad de exportación e importación”*.

En cambio en [15] dicen de BPDM: *“La definición del metamodelo para los procesos de negocio es un marco para entender y especificar los procesos de una organización o comunidad”*. Los procesos de negocio son el principal pilar del perfeccionamiento de los negocios y las tecnologías bajo algunos términos y metodologías como: Ingeniería de Procesos de Negocio, Gestión de Procesos de Negocio, Ejecución de Procesos de Negocio, Total Calidad en la Gestión, Perfeccionamiento de Procesos, Modelado de Procesos de Negocio y Flujos de Trabajo, etc.

Fundiendo las dos definiciones completándola en una sola se puede decir: *“BPDM define un metamodelo para la representación de procesos de negocio, cualquiera que sea su naturaleza y que puede ser representado mediante distintos lenguajes. Es una descripción semántica de las relaciones lógicas entre varios elementos de alguna posible descripción de un proceso de negocio. No es una notación. Simplemente describe relaciones lógicas”*.

BPDM proporciona la capacidad para representar y modelar procesos de negocio independientemente de la notación o metodología, de esta manera fusiona los diferentes enfoques dentro de la capacidad de cohesión. Esto es posible gracias a la aplicación del metamodelo definido en [21], donde dice que *“un metamodelo es el análisis, la construcción y desarrollo de macros, reglas, restricciones, modelos y teorías aplicables*

y usadas para el modelado de una clase de problemas predefinidos”. Este metamodelo pretende capturar el significado entre la notación y la tecnología, en una dirección que puede ayudar a integrar el metamodelo y las ventajas actuales de acción/efecto y el nuevo diseño. El metamodelo de BPDM usa por detrás el estándar de OMG “Meta Object Facility” (MOF) [22] para capturar modelos de procesos de negocio en la dirección general y proporcionar una sintaxis XMI para almacenar y transferir modelos de procesos de negocio entre herramientas e infraestructuras. Varias herramientas, métodos y tecnologías pueden trazar un mapa en esta dirección para ver, entender e implementar procesos mediante BPDM.

El OMG aprobó una especificación que define sólo un metamodelo de BPM (es decir, sólo el conjunto de bloques de construcción) por separado de un lenguaje de modelado. Normalmente cuando se construye un modelo en un lenguaje de modelado utilizando una herramienta para ello, el conjunto de módulos disponibles para realizarlo es el idioma del metamodelo [19].

Lo que el OMG creó fue, BPDM. La estructura y el contenido de BPDM se definieron para corresponder a las realidades de los procesos de negocio y no corresponden a BPMN ni ningún otro lenguaje de modelado. Al concentrarse en el metamodelo por separado, el modelado de lenguaje utilizado para que lo represente las empresas de modelado, algunos expertos fueron capaces de incorporar toda la gama de actividades empresariales en un metamodelo con una rica y sofisticada variedad.

La correspondencia entre los elementos del metamodelo y elementos del lenguaje de modelado es un mapeo (trasladar a un mapa, sistemas o estructuras conceptuales). Es más complicado para los constructores de un lenguaje definir un lenguaje usando un metamodelo externo como BPDM, pero las ventajas justifican el esfuerzo:

- Gracias a los elementos estructurados que permiten la automatización, los modelos basados en él pueden ser analizados y manipulados por el software.
- BPDM puede (y ha sido transportado) transportarse a más de un lenguaje.
- Transforma lenguajes gráficos permitiendo modelado de diagramas usando herramientas gráficas, mientras que para transformaciones de lenguajes basados en texto o formatos como XMI soporta exportación/importación, la red de transferencia y el almacenamiento de los modelos en los repositorios.
- Tomando ventajas de las múltiples transformaciones, el software puede transformar un modelo en BPDM basado en los lenguajes anteriores a otro mediante transformaciones adicionales.
- BPDM define un conjunto de elementos de modelado completo y suficiente para representar los procesos de negocio tanto de intra como de inter-empresa de forma natural y precisa.

El más alto nivel de BPDM soporta dos puntos de vista fundamentales y complementarios, la Orquestación y la Coreografía.

- Orquestación: Ejecución de un proceso dentro de una organización, con una secuencia de flujo en un proceso gestionado por una autoridad reconocida por todos los participantes.
- Coreografía: Colaboración a través de procesos, con la comunicación vía paso de mensajes, y ninguna autoridad de coordinación. La coreografía de un proceso suele ser implementada utilizando Arquitecturas Orientadas a Servicios o SOA (del inglés Service Oriented Architecture).

Muchos de los elementos del lenguaje de modelado del metamodelo, son visibles en el modelo, además, si el metamodelo está bien integrado definirá elementos y atributos no visibles que ayudan a representar más estrechamente el mundo real.

Estos elementos, a pesar de no ser visibles, tienen mucho que decir acerca de cómo un modelo finalizado, es y se comporta. Por ejemplo, el modelo de elementos visibles pueden estar relacionados entre sí de muchas maneras – pueden pertenecer al mismo departamento, o utilizar el mismo protocolo de interacción, o ser medidas en el mismo proceso, y un elemento particular pueden participar en la superposición de muchas relaciones al mismo tiempo. El BPDM incluye una extensa composición de modelo que define un concepto general de relaciones y permite en la representación del modelo la diversidad y la superposición de las relaciones de una forma más realista. Una extensión de la composición del modelo es el rendimiento del Curso modelo, que conecta elementos en el tiempo y se convierte en la base para la representación de un proceso. Los fundamentos de los elementos de esta composición y curso modelo, no tienen representación visible en un lenguaje de modelado como BPMN, e incluso algunas de las relaciones de ese grupo de elementos en el modelo no tienen una representación visual como símbolo. La flecha de una actividad a su sucesor es una excepción a esto.

Por su diseño, BPDM apoya procesos de muchos enfoques y notaciones. He aquí una lista extraída de la especificación:

- Flujo de trabajo, y otros conceptos más generales del proceso.
- Las actividades, tareas y procesos sub-incluidos, los ejecutados por una combinación de medios humanos y automatizados de los participantes.
- Ejecución condicional de caminos.
- Procesos paralelos.
- Flujos de procesos y flujo de datos.
- Eventos y condiciones temporales
- Condiciones de las transacciones.

- Funciones.
- Responsabilidades y colaboraciones.

1.4.1.3 Notación para el Modelado de Procesos de Negocio

El BPMN es la nueva notación para modelar flujos de procesos de negocio. Fue creado pensando en dos objetivos principales [12]:

- Proporcionar una notación que sea entendida por todos los usuarios del negocio. Estos van desde los gerentes de área y jefes de división que “tienen el negocio en la cabeza”, pasando por los analistas del negocio que crean los bocetos iniciales del proceso hasta los desarrolladores de sistemas encargados de implementar los aspectos tecnológicos para la ejecución de dichos procesos.
- Es una notación común para los lenguajes de ejecución de procesos de negocio. BPMN pretende garantizar que los lenguajes de procesos XML diseñados para la ejecución de los procesos de negocio, como BPEL4WS (del inglés *Business Process Execution Language for Web Services*) y BPML (del inglés *Business Process Modelling Language*), puedan ser visualmente expresados a través de una notación común. Esto implicaría que los modelos creados como BPMN puedan ser directamente ejecutados en BPML o mapeados a BPEL.

Una tercera característica fundamental de la notación BPMN es que debe proveer la capacidad de modelar procesos complejos y sistemas de procesos, es decir, debe contar una semántica y una sintaxis suficientemente ricas como para representar todos los aspectos de un proceso.

BPMN consiste en un diagrama, llamado Diagrama de Procesos de Negocio (BPD, del inglés *Business Process Diagram*). A diferencia de otros diagramas de procesos de negocio, el BPD fue creado con los lenguajes de ejecución de procesos y los Servicios Web (del inglés *Web Services*) en mente. Notaciones especiales han sido añadidas al diagrama para describir eventos basados en mensajes y paso de mensajes entre organizaciones, permitiendo el modelado de B2B y B2C.

La siguiente explicación de Stephen White [43] del proceso de modelado a través de BPMN permite entender por qué la notación es defendida como “fácil de entender”. Leyendo esta explicación cualquier usuario del negocio podría entender un modelo BPMN. Sin embargo esto en ningún momento garantiza que los usuarios (en especial la alta gerencia) estén en disposición de aprender el estándar para lograr crear especificaciones de procesos completas, que permitan definir las características detalladas de implementación de un proceso y lograr así ejecutarlo directamente.

“Para modelar el flujo del proceso de negocio, simplemente se modelan los eventos que deben ocurrir para que el proceso empiece, el proceso que debe ejecutarse y los resultados finales del flujo del proceso. Las decisiones del negocio y ramas del

flujo son modeladas usando pasarelas (del inglés gateways). Una pasarela es similar a un símbolo de decisión en un diagrama de flujo.

Adicionalmente, un proceso puede contener sub-procesos, los cuales pueden ser mostrados gráficamente por otro BPD conectado a través de un hipervínculo a un símbolo de proceso. Si un proceso no se descompone en sub-procesos, se considera una tarea (el nivel más bajo de proceso). Una marca '+' en el símbolo del proceso denota que el proceso se descompone, si no tienen la marca, es una tarea.

Mientras se profundiza más en el análisis del negocio, es posible especificar quién hace qué ubicando los eventos y procesos en áreas compartidas llamadas pool¹ que denotan quien desempeña el proceso. Es posible también dividir las pools en lanes. Una pool típicamente representa una organización, y una lane un departamento dentro de la organización, sin embargo, pueden representar otras cosas como funciones, aplicaciones o sistemas.”

Un BPD está compuesto por un grupo de elementos gráficos que se organizan en cuatro categorías básicas y que se desarrollaran más adelante:

- **Objetos de flujo:** Eventos, actividades y pasarelas.
- **Objetos conectores:** Flujo de secuencia, flujo de mensaje y asociación.
- **Swimlanes:** Pool, lane.
- **Artefactos:** Objetos de datos, grupo y anotación.

Como se vio anteriormente los estándares de BPMI.org trabajan con el concepto de proceso e-business para el que define una interfaz pública entre los participantes y las implementaciones privadas. Por esta razón en BPMN existen dos tipos básicos de modelos que pueden ser creados con BPD:

- **Procesos colaborativos B2B (públicos):** Describen las interacciones, relaciones y transiciones entre dos o más negocios. Estos BPD son especificaciones generales en las que no se tienen en cuenta los detalles de los procesos de un participante en particular, sino que se muestran las interacciones entre los participantes. Las interacciones son descritas como una secuencia de actividades y el intercambio de patrones de mensajes entre participantes.

¹ En muchos de los términos de este trabajo se ha traducido del inglés y como tienen correspondencia directa con el castellano no se ha considerado necesario poner su traducción en inglés. En otros términos se ha traducido al castellano intentando encontrar una correspondencia de significado lo más fidedigna posible y por ello se ha puesto su traducción también en inglés. Además se encontrarán algunos términos en inglés, sin que hayan sido traducidos al castellano, debido a que una posible traducción podría llevar a una interpretación errónea del término, así como de su significado en este contexto.

- **Procesos de negocio internos (privados):** El concepto de proceso y de cadena de valor difícilmente permite un proceso interno sin interacción con el mundo exterior (aunque existen), sin embargo para los procesos o subprocesos que se desarrollan internamente se deben definir las actividades que generalmente no son visibles al público y son por ende, actividades privadas.

En los siguientes apartados se va a describir la nomenclatura para procesos de negocio, así como las reglas que utiliza BPMN para los flujos de trabajo.

1.4.1.4 Conformidad

A continuación se describen las características que tienen que cumplir las aplicaciones para soportar BPMN.

Apariencia Visual

Un elemento clave de BPMN es la elección de la forma y los iconos a usar para los elementos gráficos identificados en la especificación [18]. Se ha intentado crear un lenguaje visual estándar que todos los modeladores de procesos reconozcan y entiendan.

En [12] se especifican un conjunto de cláusulas que debe cumplir un diagrama y modelo en BPMN. En este trabajo, no interesa entrar en tan alto detalle de especificación, con lo que no se van a describir dichas cláusulas.

El estándar descrito en [12] permite que los diagramas de BPMN se extiendan de la siguiente forma:

- Se pueden añadir nuevas marcas o indicadores. Pueden ser usados para resaltar la especificación de un atributo o para representar un subtipo de algún concepto.
- Se pueden añadir nuevas formas para representar un tipo de artefacto, pero esta nueva forma no puede entrar en conflicto con ningún otro elemento o marca de BPMN.
- Se pueden pintar o colorear los elementos, y los colores pueden significar una semántica especial que amplía la información de los elementos.
- El estilo de la línea de los elementos gráficos puede cambiarse, pero el cambio no debe entrar en conflicto con ningún otro estilo de línea de otros elementos definidos en el estándar de BPMN.

Una extensión no debe cambiar la forma especificada para la definición de los elementos gráficos o marcas (por ejemplo, cambio de un cuadrado a un triángulo, o el cambio de esquinas redondeadas a esquinas cuadradas).

Conformidad Estructural

En [12] definen un conjunto de cláusulas que tienen que definir los modelos para obtener la conformidad estructural que define BPMN. Donde se requiere o permite conexiones condicionales se representadas con el atributo correspondiente al concepto,

la aplicación deberá garantizar la correspondencia entre las conexiones y los valores de los atributos.

En general, estas conexiones y las relaciones tienen una interpretación semántica, que especifican las interacciones entre los conceptos de los procesos representados por los elementos gráficos. Las relaciones condicionales basadas en atributos específicos representan variaciones en el comportamiento. La conformidad estructural, por lo tanto, garantiza la interpretación correcta del diagrama como una especificación de proceso, en términos de flujos de control y de información.

Elementos Semánticos

La especificación define los conceptos semánticos utilizados en la definición de procesos, y asociaciones gramaticales con elementos, marcas y conexiones. En la medida en que una aplicación ofrece una interpretación de BPMN como una especificación semántica de un proceso, dicha interpretación debe ser coherente con la gramática descrita en el estándar [12].

Lo que se intenta es que un diagrama BPMN se utilice como la especificación de un flujo de trabajo que tiene la interpretación especificada en el estándar [12], el cual se aplica o reduce por las características de los sistemas de flujos de trabajo.

Atributos y Propiedades

En [12] definen un número de atributos y propiedades de los objetos semánticos representados (elementos, marcas y conexiones). Algunos de estos atributos son puramente representativos y son como marcas, y algunos requieren representación. Algunos atributos son de carácter obligatorio, pero no tienen representación o es opcional. Otros atributos son de carácter opcional.

Para todo atributo o propiedad que sea de carácter obligatorio, las aplicaciones deben proporcionar algún mecanismo para crear y representar estos elementos. Este mecanismo debe permitir crear o ver los valores para cada objeto especificado con estos atributos o elementos.

Donde una representación gráfica de atributos o propiedades opcionales, la aplicación puede usar otra representación gráfica u otro mecanismo.

Donde no hay representación gráfica de atributos o propiedades, la aplicación puede usar otra representación gráfica u otro mecanismo. Si se utiliza una representación gráfica, no debe existir conflicto entre dicha representación y otro objeto de BPMN.

Elementos Opcionales y Ampliación

Una aplicación no requiere soportar ningún elemento o atributo que no esté especificado en la normativa [12] o que sea informativo.

Las aplicaciones no están obligadas a soportar características que se clasifican como opcionales. En el caso en el que las soporte, debe ser en la forma en la que se especifica.

Una aplicación no soportará ninguna característica opcional en el caso en el que lo único que gestione sea la forma como se muestra dicha característica.

1.4.1.5 Alcance de BPMN

BPMN está obligado a soportar o gestionar solo los conceptos de modelado que se aplican a los procesos de negocio. Esto significa que otros tipos de modelado realizado por las organizaciones con fines comerciales, estará fuera del alcance de BPMN. Por ejemplo, el modelado de lo siguiente, no forma parte de BPMN:

- Estructura y recursos de una organización.
- Errores funcionales.
- Modelo de datos y de información.
- Estrategias.
- Reglas de negocio.

Desde que este tipo de modelado de alto nivel afecta directa e indirectamente a los procesos de negocio, las relaciones entre BPMN y otro nivel de modelado de procesos de negocio será definido más formalmente como BPMN y la otra especificación lo completará.

Señalar qué mientras BPMN muestra el flujo de datos (mensajes), y la asociación entre los datos de los artefactos y las actividades, esto no son diagramas de flujo.

1.4.1.6 Usos de BPMN

El modelado de procesos de negocio se utiliza para comunicar una amplia variedad de información con una amplia cantidad de audiencia. BPMN está diseñado para cubrir los distintos tipos de modelado y permitir la creación de procesos de negocio como comienzo y fin. La estructura de los elementos de BPMN permitirá distinguir fácilmente entre las secciones de un diagrama BPMN.

Existen tres tipos básicos de sub modelos² dentro de los modelos de BPMN:

- *Procesos de negocio privados (Internos)*: Los procesos de negocio privados son los procesos internos especificados por las organizaciones y son el tipo de proceso que suele llamarse flujo de trabajo o proceso BPM. Un único

² La terminología empleada para describir los diferentes tipos de procesos no ha sido estandarizada. Estas definiciones están en un estado de cambio.

proceso de negocio puede ser asignado a uno o más documentos BPEL4WS. Si un *swimlane* es utilizado, entonces los procesos de negocio deben estar en una *pool* individual. La secuencia de flujos del proceso está, por lo tanto, dentro de la *pool* y no puede cruzar las fronteras de ésta. Los flujos de mensajes pueden cruzar la frontera para mostrar la interacción existente entre procesos de negocio privados separados. De esta forma, un simple diagrama de procesos de negocio puede mostrar múltiples procesos de negocio privados, cada uno con correspondencia a BPEL4WS separada.

- *Procesos abstractos (Públicos)*: Representan la interacción entre procesos de negocio internos y otros procesos o participantes. Solo aquellas actividades que se utilizan para comunicarse fuera de los procesos de negocio internos, además del correspondiente mecanismo de control de flujo, se incluyen en los procesos abstractos. El resto de actividades internas del proceso de negocio privado no son representadas como procesos abstractos o públicos. Con lo que los procesos abstractos muestran la secuencia de mensajes que se requieren para interactuar con los procesos de negocio. Estos procesos se representan dentro de una *pool* y pueden ser modelados por separado o dentro de un diagrama BPMN amplio para mostrar el flujo de mensajes entre las actividades de los procesos abstractos y otras entidades. Si el proceso público pertenece al mismo diagrama que un proceso privado, entonces las actividades que son comunes en ambos procesos pueden ser asociadas.
- *Procesos colaboradores (Globales)*: Estos procesos representan la interacción entre dos o más entidades de negocio. Estas interacciones son definidas como una secuencia de actividades que representan los mensajes que se intercambian entre las entidades involucradas. Los procesos globales pueden mostrarse como dos o más procesos abstractos que se comunican entre ellos.

Entre estos sub modelos de BPMN se pueden crear otros tipos de diagramas. En BPMN se pueden modelar varios tipos de procesos de negocio, pero en este documento no es de interés.

Desde que un diagrama BPMN puede describir los procesos para diferentes participantes, cada uno de estos participantes puede ver el diagrama de forma diferente. Esto quiere decir, que cada participante tiene un punto de vista respecto a cómo utiliza o aplica el diagrama sobre el proceso. Algunas de las actividades pueden ser internas para algunos participantes y otras actividades pueden ser externas o públicas para otros. Cada participante tendrá su perspectiva, pudiendo ser interna o externa. En la ejecución, la diferencia entre actividades internas o externas es importante para como un participante puede ver el estado las actividades o mediar sobre algunos problemas. Sin embargo, el diagrama puede solventar los problemas.

Sin embargo el punto de vista de un diagrama es importante para que sea entendible por un observador que quiera ver el comportamiento de un proceso, BPMN

actualmente no especifica ningún mecanismo gráfico para enfatizar estos puntos de vista. Esto queda a merced del modelador o de las herramientas de modelado, deberán ser estos los que deban enfatizar estos puntos de vista en los diagramas.

BPMN se ha creado, también, con el objetivo de ser extensible. Esta propiedad permite a los modeladores añadir elementos estándar o artefactos que satisfagan la especificación necesaria, como el único requerimiento del llamado *Dominio Vertical*. Aunque sea extensible, los diagramas de BPMN deben mantener la base de un diagrama (parecer y sentir, del inglés *look and feel*) de modo que cualquier modelador haga el diagrama entendible para cualquier observador. Ésta es la huella de los elementos de flujo básicos (eventos, actividades y pasarelas), que no deben ser modificados. Tampoco debe ser añadido ningún nuevo elemento de flujo a un diagrama de proceso de negocio, si no se ha especificado como una secuencia y mensaje de flujo que conecte con un nuevo objeto de flujo. Adicionalmente, la transformación a lenguajes de ejecución puede verse afectada si se añaden nuevos elementos de flujo. Para satisfacer el modelado de conceptos adicionales que no forman parte del conjunto básico de elementos de flujos, BPMN proporciona el concepto de artefactos que pueden ser asociados con los objetos de flujo existentes a través de asociaciones. Estos artefactos no afectan al flujo básico de secuencia y mensajes, tampoco afecta a la transformación a un lenguaje de ejecución.

Los elementos gráficos de BPMN han sido diseñados para abrir todas las marcas especializadas que se permiten para transmitir información específica. Por ejemplo, los tres tipos de eventos abren los centros de los marcadores, que BPMN estandariza, así como los marcadores definidos por el usuario.

1.4.1.7 Diagramas de Procesos de Negocio

A continuación se proporciona una descripción de los objetos gráficos y relaciones que define BPMN.

Un objetivo para los desarrolladores de BPMN es que la notación sea simple y fácilmente adoptable para los analistas de negocio. También, hay un conflicto potencia entre los requisitos de BPMN, ya que este proporciona poder para describir procesos de negocio complejos y mapas para lenguajes de ejecución de BPM. Para ayudar a entender como BPMN puede gestionar estos requisitos, se presenta los elementos gráficos en dos grupos.

El primer grupo, es una lista de elementos básicos o el núcleo de objetos que soportará los requisitos de la notación más simple. Estos elementos definen la vista y forma (parecer y sentir) más básica de BPMN. La mayor parte de los procesos de negocio serán modelados con estos elementos. El segundo grupo, es el conjunto completo, e incluye los elementos básicos, los cuales ayudaran a soportar los requisitos más fuertes de la notación para poder reflejar y mantener situaciones un modelado más avanzado. Y para ir más allá, los elementos gráficos serán gestionados por atributos no

gráfico que proporcionarán el resto de información necesaria para obtener un mapa de un lenguaje de ejecución u obtener otros propósitos del modelado de procesos.

Conjunto Núcleo de Elementos de BPD

Cabe destacar que uno de los motores para el desarrollo de BPMN es crear un mecanismo sencillo para la creación de modelos de proceso de negocio, mientras que al mismo tiempo sea capaz de gestionar procesos de negocio inherentemente complejos. El enfoque que toma la gestión de estos dos puntos debe ser organizar los aspectos gráficos de la notación para cada categoría específica. Esto proporciona un conjunto pequeño de categorías de notación de forma que el lector de un diagrama BPMN puede reconocer fácilmente los tipos básicos de elementos y entender el diagrama. Dentro de estas categorías básicas de elementos, hay variaciones e información adicional que se puede añadir para gestionar los requisitos complejos sin cambiar drásticamente la mirada y sensación (parecer y sentir) básica del diagrama o figura. Las cuatro categorías de elementos son las siguientes:

1. Objetos de flujo
2. Objetos conectores
3. *Swimlane*
4. Artefactos

Los objetos de flujo son los principales elementos gráficos para definir el comportamiento de un proceso de negocio. Hay tres objetos de flujo:

1. Eventos
2. Actividades
3. Pasarelas

Existen tres formas de conectar los objetos de flujo entre ellos o con otra información. Hay tres objetos conectores:

1. Flujo de Secuencia
2. Flujo de Mensaje
3. Asociaciones

Hay dos formas de en las que se agrupan los principales elementos a través de los “*swimlanes*”:

1. *Pools*
2. *Lane*








Los artefactos se utilizan para proporcionar información adicional sobre el proceso. En BPMN se han normalizado tres artefactos pero los modeladores o herramientas de modelado son libres de añadir más artefactos, según sea necesario.

Además BPMN se esfuerza por estandarizar un amplio conjunto de artefactos para el uso general o para mercados verticales. El conjunto actual de artefactos incluye:

1. Objetos de Datos.
2. Grupos
3. Anotaciones

En las siguientes tablas se muestra la lista de los elementos de modelado. No sólo se incluyen los elementos básicos y una descripción de estos, sino que también se incluye un desglose de esto, mostrando los subtipos en los que se dividen.

La tabla 1 muestra los objetos de flujo:

Elemento	Descripción	Notación	Sub Conjunto
Evento	Un evento se representa con un círculo. Es algo que “pasa” durante el curso del proceso de negocio. Estos eventos afectan al flujo del proceso y suelen tener una causa (trigger) o un impacto (resultado). Los eventos representados con un círculo con centro abierto permiten a los marcadores internos diferenciar diferentes causas y resultados. Hay tres tipos de eventos, basados en cuando afectan al flujo: <i>Inicio</i> , <i>Intermedio</i> , y <i>Fin</i> .		 Inicio  Intermedio  Fin
Actividad	Una actividad se representa con un rectángulo redondeado y es un término genérico para el trabajo que hace una compañía. Una actividad puede ser atómica o compuesta. Los tipos que hay son: <i>Tarea</i> y <i>Sub-Proceso</i> . El Sub-Proceso se distingue por una pequeña marca de suma en la parte central inferior de la figura.		 Tarea  Suproceso



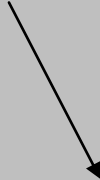


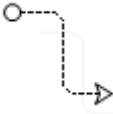
Elemento	Descripción	Notación	Sub Conjunto
Pasarela	Una <i>pasarela</i> se representa por la típica figura de diamante y se usa para controlar la divergencia o convergencia de la secuencia de flujo. Así, esto determina las tradicionales decisiones, así como la creación de nuevos caminos, la fusión de estos o la unión. Los marcadores internos indicarán el tipo de control de comportamiento.		

Tabla 1. Objetos de Flujo

Los objetos de flujo se conectan entre ellos en un diagrama para crear el esqueleto básico de la estructura de un proceso de negocio. Hay tres objetos conectores que hacen esta función.

La tabla 2 muestra los objetos conectores:

Elemento	Descripción	Notación	Sub Conjunto
Flujo de Secuencia	El flujo de secuencia se representa por una línea sólida con una cabeza de flecha sólida y se usa para mostrar el orden (la secuencia) en el que las diferentes actividades se ejecutarán en el Proceso. El término “flujo de control” normalmente no se usa en BPMN.		
Flujo de Mensaje	El flujo de mensaje se representa por un línea discontinua con una punta de flecha hueca y se usa para mostrar el flujo de mensajes entre dos participantes del proceso separados (entidades de negocio o roles de negocio). En BPMN, dos <i>pools</i> separadas en el diagrama representan los dos participantes.		

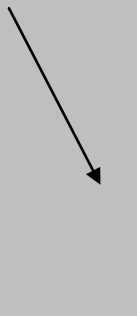

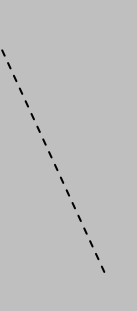

Elemento	Descripción	Notación	Sub Conjunto
Flujo de Secuencia	El flujo de secuencia se representa por una línea sólida con una cabeza de flecha sólida y se usa para mostrar el orden (la secuencia) en el que las diferentes actividades se ejecutarán en el Proceso. El término “flujo de control” normalmente no se usa en BPMN.		
Asociaciones	Una asociación se representa por una línea de puntos con una punta de flecha de líneas y se usa para asociar datos, texto, y otros artefactos con los objetos de flujo. Las asociaciones se usan para mostrar entradas y salidas de las actividades.		


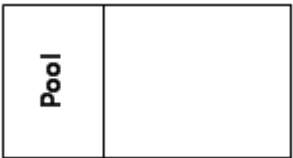
Tabla 2. Objetos Conectores

Para los modeladores que requieren o desean más precisión para crear modelos de proceso por motivos de documentación y comunicación, los elementos básicos más los conectores dan la posibilidad de crear fácilmente diagramas comprensible.

Para los diseñadores que necesiten un nivel más alto de precisión, para análisis detallado o que sean manejados por un Gestor de Sistemas de Procesos de Negocio, existen detalles adicionales que se pueden añadir a los elementos básicos.

Muchas metodologías de modelado de procesos usan el concepto de *swimlanes* como un mecanismo para organizar actividades en categorías separadas visualmente para ilustrar diferentes capacidades funcionales o responsabilidades. BPMN soporta los canales con dos constructores principales.

La tabla 3 muestra los canales:

Elemento	Descripción	Notación	Sub Conjunto
Pool	Una <i>pool</i> representa un participante de un proceso. Además actúa como un contenedor gráfico para particionar un conjunto de actividades desde otras <i>pools</i> , normalmente en el contexto de B2B.		

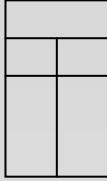

Elemento	Descripción	Notación	Sub Conjunto
Lane	Una <i>lane</i> es una sub-partición dentro de una <i>pool</i> y extiende la longitud de la <i>pool</i> , verticalmente u horizontalmente. Las <i>lanes</i> se usan para organizar y categorizar actividades.		



Tabla 3. Swimlanes.

Las *pools* se usan cuando un diagrama implica dos entidades de negocio o participantes separados y están físicamente separados en el diagrama. Las actividades dentro de las *pools* separadas se consideran procesos *auto contenidos*. Así, el flujo de secuencia no debe cruzar el límite de una *pool*. El flujo de mensajes se define como el mecanismo para mostrar las comunicaciones entre dos participantes, y, de este modo debe conectar dos *pools* (o los objetos dentro de las *pools*).

Las *lanes* están más estrechamente relacionadas con las metodologías tradicionales de los *swimlanes*. Las *lanes* se suelen usar para separar las actividades asociadas con la función o rol de una compañía específica. El flujo de secuencia puede cruzar los límites de las *lanes* dentro de una *pool*, pero el flujo de mensajes no puede ser usado entre objetos de flujo en *lanes* de la misma *pool*.

BPMN fue diseñado para permitir a los modeladores y las herramientas de modelado un poco de flexibilidad a la hora de extender la notación básica y a la hora de habilitar un contexto apropiado adicional según una situación específica, como para un mercado vertical (por ejemplo, seguros o banca). Se puede añadir cualquier número de artefactos a un diagrama como sea apropiado para un contexto de proceso de negocio específico. La versión actual de la especificación de BPMN sólo tiene tres tipos de artefactos BPD predefinidos.

La tabla 4 muestra los artefactos:

Elemento	Descripción	Notación	Sub Conjunto
Objeto de Datos	Los objetos de datos son un mecanismo para mostrar como los datos son requeridos o producidos por las actividades. Están conectados a las actividades a través de		



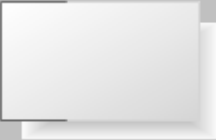









Elemento	Descripción	Notación	Sub Conjunto
	asociaciones.		
Objetos	Un grupo es representado por un rectángulo redondeado con línea discontinua. El agrupamiento se puede usar documentación o análisis, pero no afecta al flujo de secuencia.		
Anotaciones	Las anotaciones son mecanismos para que un modelador pueda dar información textual adicional.		








Tabla 4. Artefactos

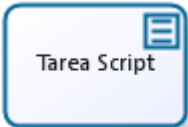





Los modeladores pueden crear sus propios tipos de artefactos, que añaden más detalle sobre cómo se ejecuta el proceso normalmente para mostrar las entradas y las salidas de las actividades del proceso. Sin embargo, la estructura básica del proceso, determinada por las actividades, pasarelas, y flujos de secuencia, no se cambia por añadir artefactos al diagrama.

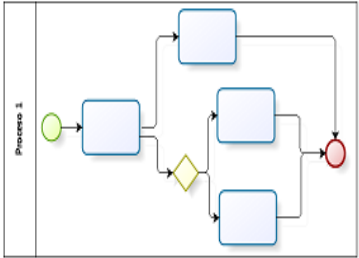




La tabla 5 muestra una lista más extensa de los conceptos de procesos de negocio que podrían ser representados a través de una notación de modelado de procesos de negocio.





Elemento	Descripción	Notación
Eventos	Los eventos de inicio e intermedio definen la causa del evento. Hay múltiples formas por las que estos eventos se pueden disparar. El evento final puede definir un resultado que es consecuencia de la secuencia de flujo que finaliza. El evento inicio solo puede reaccionar (o capturar del	<p>Mensaje</p>  <p>Temporizador</p> 



Elemento	Descripción	Notación
	<p>inglés <i>catch</i>) a un disparador. Los eventos finales solo pueden crear (o obtener o lanzar del inglés <i>throw</i>) un resultado. Los eventos intermedios tanto capturan como lanzan disparadores. Para los eventos que capturan, el marcador está vacío y para los disparadores y los casos en los que se obtiene un resultado, los marcadores están llenos.</p>	<p>Error</p>  <p>Cancelar</p>  <p>Compensación</p>  <p>Condicional</p>  <p>Enlace</p>  <p>Señal</p>  <p>Fin</p>



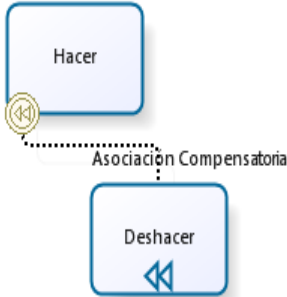

Elemento	Descripción	Notación
		 <p>Múltiple</p> 
<p>Tarea (Atómica)</p>	<p>Una tarea es una actividad que es atómica y que se incluye dentro de un proceso. Una tarea se usa cuando el trabajo en el proceso se quiere reflejar a un nivel de detalle del modelo de procesos y que no puede dividirse en tareas más pequeñas.</p>	    

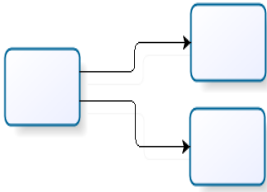
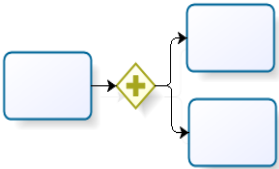
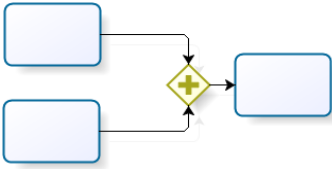
Elemento	Descripción	Notación
		 <p>Tarea Script</p>  <p>Tarea Manual</p>  <p>Tarea Referencia</p>
<p>Procesos/Sub-Procesos (No atómicos)/Sub-proceso Colapsado</p>	<p>Un sub proceso es una actividad compuesta que se incluye dentro de un proceso. Se trata de compuestos en que se pueden desglosar en un fino nivel de detalle (un proceso) a través de un conjunto de sub actividades.</p> <p>Los detalles del sub proceso no son visibles en el diagrama. Un signo “+” en el centro de la forma indica que la actividad es un proceso y que tiene un nivel inferior de detalle.</p>	 <p>SubProceso Embebido</p>  <p>SubProceso Reutilizable</p>  <p>SubProceso de Referencia</p>



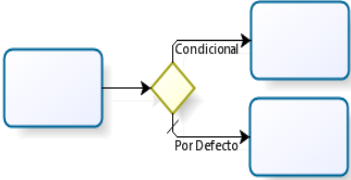
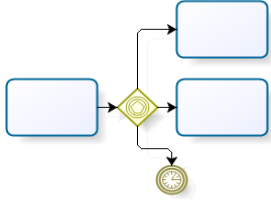
Elemento	Descripción	Notación
<p>Sub Proceso Ampliado</p>	<p>El límite para expandir un sub proceso y hacer visibles los detalles (del proceso) dentro de estos límites. El flujo de secuencia de un sub proceso no puede cruzar los límites.</p>	
<p>Pasarela</p>	<p>Una puerta de enlace o pasarela se utiliza para controlar la divergencia y convergencia de múltiples secuencias de flujo. Por lo tanto, determinará la ramificación, bifurcación, fusión y unión de caminos.</p>	
<p>Tipos de Control de Pasarela</p>	<p>Se representan mediante un icono en forma de diamante, que indican el tipo de comportamiento del flujo de control. Los tipos de control incluyen:</p> <ul style="list-style-type: none"> • Decisión Exclusiva y fusión: tanto lo basado en datos como en eventos. En base a los datos, se muestra con o sin la "X". • Inclusión con decisión y fusión. • Complejas: condición y 	<p>Datos o</p>  <p>Exclusivo</p> <p>Evento</p>  <p>Inclusión</p> 

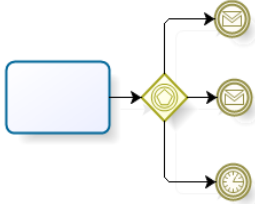
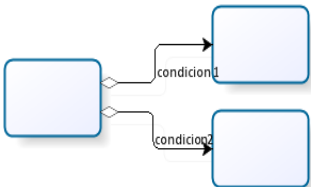
Elemento	Descripción	Notación
	<p>situación.</p> <ul style="list-style-type: none"> • Bifurcación y unión paralelas. <p>Cada uno de los tipos de control afecta tanto a la entrada como a la salida.</p>	<p>Complejo</p>  <p>Paralelo</p> 
Flujo de Secuencia	<p>El flujo de secuencia se usa para mostrar el orden en el que se ejecutan las tareas en el proceso.</p>	<p>Ver las siguientes 3 figuras.</p>
Flujo Normal	<p>El flujo normal de secuencia se refiere al flujo original que va del evento inicial y continúa a través de las actividades por caminos paralelos o alternativos hasta que finaliza en el evento final.</p>	
Flujo Incontrolado	<p>El flujo incontrolado se refiere al flujo que no se ve afectado por ninguna condición o no pasa por ninguna pasarela o puerta de enlace. El ejemplo más simple es cuando el flujo conecta dos actividades. Esto también se puede aplicar a múltiples flujos de secuencia que convergen o divergen en una actividad. Por cada flujo de</p>	

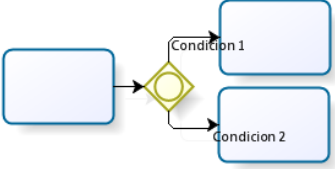
Elemento	Descripción	Notación
	<p>secuencia incontrolado un token se desviará del origen al objetivo.</p>	
<p>Flujo Condicional</p>	<p>Un flujo de secuencia puede tener una expresión condicional que será evaluada en tiempo de ejecución para determinar si el flujo será usado o no:</p> <ul style="list-style-type: none"> • Si el flujo condicional sale de la actividad, entonces la secuencia de flujo tendrá un diamante pequeño al comienzo de la línea. • Si el flujo condicional sale de una pasarela o puerta de enlace, entonces la línea no tendrá un diamante pequeño al principio de la línea. 	
<p>Flujo por Defecto</p>	<p>Para decisiones basadas en exclusión de datos o inclusión, un tipo de flujo es el flujo condicional por defecto. Este flujo solo se usa si todos los flujos condicionales existentes de salida no se cumplen en tiempo de ejecución. Este flujo de secuencia tiene una barra diagonal al inicio de la línea.</p>	

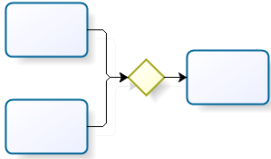

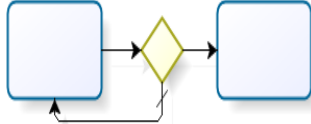
Elemento	Descripción	Notación
Flujo de Excepción	Este flujo ocurre fuera del flujo normal del proceso y está basado en un evento intermedio que ocurre durante la ejecución del proceso.	
Flujo de Mensaje	Un flujo de mensaje se usa para mostrar el flujo de mensajes entre dos entidades que están preparadas para enviar y recibir estos. En BPMN dos <i>pools</i> separadas en el diagrama representan dos entidades.	
Asociación Compensatoria	La asociación compensatoria ocurre fuera del flujo normal de un proceso y está basada en un evento que se dispara a través del fallo de una transacción o un evento compensatorio. La etiqueta de una asociación se debe marcar como una actividad compensatoria.	
Objeto de Datos	Los objetos de datos son considerados artefactos porque no tienen un efecto directo en el flujo de secuencia o en el flujo de mensajes del proceso, pero esto proporciona información sobre qué actividades requieren ser ejecutadas y/o que producen.	




Elemento	Descripción	Notación
<p>Bifurcación</p>	<p>BPMN usa el término bifurcación para referirse a la división de un camino en dos o más caminos paralelos. Esto es un lugar en el proceso donde las actividades pueden ejecutarse concurrentemente, más que secuencialmente. Hay dos opciones:</p> <ul style="list-style-type: none"> • Puede utilizarse la secuencia de flujo con múltiples salidas. Esta representación de flujo incontrolado es el método preferido para la mayoría de las situaciones. • O se puede utilizar una pasarela paralela. Se utiliza raramente. Generalmente se combinan varias pasarelas. 	 
<p>Unión</p>	<p>BPMN usa el término unión para referirse a la combinación de dos o más caminos paralelos en un único camino. Una pasarela paralela se utilizara para mostrar la unión de múltiples flujos.</p>	
<p>Decisión, Punto de Ramificación</p>	<p>La decisión es una pasarela dentro de los procesos donde el flujo de control puede tomar uno o varios caminos alternativos.</p>	<p>Ver las 5 figuras siguientes.</p>


Elemento	Descripción	Notación
<p>Exclusivo</p>	<p>Una pasarela exclusiva restringe el flujo, como que solo una del conjunto de alternativas puede elegirse en tiempo de ejecución. Hay dos tipos de pasarelas exclusivas: basadas en datos y basadas en eventos.</p>	<p>Basados en Datos</p>  <p>Basados en Eventos</p> 
<p>Basados en Datos</p>	<p>Esta decisión representa un punto de ramificación donde las alternativas se basan en una expresión de condición que contiene la salida del flujo de secuencia. Solo se puede elegir una de las alternativas.</p>	
<p>Basados en Eventos</p>	<p>Esta decisión representa un punto de ramificación donde las alternativas a tomar dependen de un evento que ocurre en un punto del proceso. El evento específico, normalmente es un mensaje, y determina qué camino debe tomarse. Otros tipos de eventos puede ser el temporizador. Solo puede tomarse una alternativa. Hay dos opciones</p>	





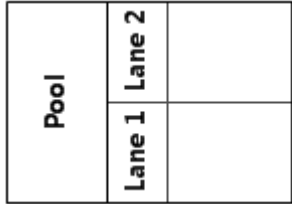
Elemento	Descripción	Notación
	<p>para recibir mensajes:</p> <ul style="list-style-type: none"> • Se puede utilizar la recepción de tareas. • O un evento intermedio con mensajes. 	
<p>Inclusión</p>	<p>Esta decisión representa un punto de ramificación donde las alternativas se basan en una condición que decide sobre la salida del flujo de secuencia. En algún sentido este grupo está relacionado con decisiones binarias (Sí/No) independientes. Desde que cada camino es independiente, toda combinación de caminos puede tomarse desde cero a todos. Sin embargo, esto debe ser definido de forma que al menos se tome un camino. Una condición por defecto podría</p>	

Elemento	Descripción	Notación
	<p>utilizarse para garantizar que al menos uno de los caminos se toma. Existen dos versiones de este tipo de decisiones:</p> <ul style="list-style-type: none">• El primer uso es una colección de flujos de secuencia condicionales, indicado con un diamante pequeño (ver primera figura de la derecha)• El segundo uso es una pasarela inclusiva (ver segunda figura de la derecha).	 <pre>graph LR; A[] --> B{ }; B -- "Condicion 1" --> C[]; B -- "Condicion 2" --> D[]</pre>

Elemento	Descripción	Notación
<p>Combinación</p>	<p>BPMN usa el término <i>combinar</i> para referirse a una combinación exclusiva de dos o más caminos en un solo camino (también se conoce como un O-Unión). Una pasarela con combinación exclusiva se usa para mostrar la combinación múltiple de flujos. Si todas las entradas de un flujo son alternativas, entonces no se necesita esta pasarela. Esto es un flujo incontrolado que proporciona el mismo comportamiento.</p>	
<p>Bucle</p>	<p>BPMN proporciona dos mecanismos de bucle dentro de los procesos.</p>	<p>Ver las dos figuras siguientes.</p>
<p>Actividad Bucle</p>	<p>Este atributo de una tarea y de un sub proceso se determinarán si hay repetición o se ejecuta solo una vez. Hay dos tipos de bucles: estándar y de múltiples instancias. Un bucle se representa con un indicador en la parte inferior central.</p>	
<p>Bucle de Flujo de Secuencia</p>	<p>Los bucles se pueden crear mediante la conexión de un flujo de secuencia con un objeto o pasarela. Un objeto se considera que vuelve al flujo, si el objeto tiene un flujo de secuencia de salida que</p>	

Elemento	Descripción	Notación
	<p>conduce a otro flujo de secuencia, en el cual el último flujo es el de entrada al objeto original.</p>	
<p>Múltiples Instancias</p>	<p>Los atributos de las tareas y de los sub procesos determinan si se repiten o se ejecutan una sola vez. Se muestra estas instancias con un indicador de tres líneas paralelas en el centro de la actividad.</p>	
<p>Proceso en Stand-by (Algo ajeno al proceso hace que este se pare)</p>	<p>Un proceso en Stand-by es una posición en el proceso que muestra donde se espera que se produzca una demora dentro de un proceso. Se utiliza un evento intermedio para mostrar este comportamiento. Adicionalmente, se utiliza un artefacto por modeladores y herramientas de modelado, y se puede asocia con un evento de alto nivel en el lugar donde se espera la demora dentro del flujo.</p>	
<p>Transición</p>	<p>Una transición es un sub proceso que lo soporta un protocolo especial que asegura que todas las partes interesadas están completamente de acuerdo en que la actividad debe ser completada o cancelada. Los atributos de la</p>	

Elemento	Descripción	Notación
	<p>actividad determinan si la actividad es una operación. Una doble línea en la figura indica que el proceso es una transición.</p>	
<p>Sub Proceso Anidado/Embebido (Bloque en Línea)</p>	<p>Un proceso anidado (o embebido) es una actividad que parte del mismo conjunto de datos que el proceso padre. Esto es contrario al sub proceso que es independiente, re-utilizable, y que hace referencia al proceso padre. Es necesario que los datos se pasen por referencia al sub proceso, pero no a procesos anidados.</p>	<p>No hay ninguna figura específica para este tipo de procesos.</p>
<p>Grupo (Una caja alrededor de un grupo de objetos que están en una misma categoría)</p>	<p>Es un conjunto de actividades que están dentro de una misma categoría. Este tipo de grupo no afecta al flujo de secuencia de las actividades dentro del mismo grupo. El nombre de la categoría aparece en el diagrama como el nivel del grupo. Las categorías se pueden usar para documentar o analizar por algún propósito. Los grupos son una dirección en la que los objetos de la categoría se pueden ver en un diagrama.</p>	

Elemento	Descripción	Notación
<p>Conector Fuera de Página</p>	<p>Generalmente se utiliza para imprimir, este objeto mostrará donde el flujo de secuencia deja una sola página y, a continuación, se reinicia en la siguiente página. Se utiliza para mostrar este objeto un evento intermedio de enlace.</p>	
<p>Asociación</p>	<p>La asociación se utiliza para asociar información con los objetos de flujo. Los objetos de texto y gráficos que no son de flujo pueden asociarse con estos objetos de flujo.</p>	
<p>Anotaciones de Texto (Adjunta una asociación)</p>	<p>Las anotaciones de texto son un mecanismo para proporcionar información adicional al lector del diagrama BPMN.</p>	
<p>Pool</p>	<p>Una <i>pool</i> representa a uno de los participantes del proceso. Actúa también como un <i>swimlane</i> y un contenedor gráfico para dividir un conjunto de actividades en otras <i>pools</i>, por lo general en el contexto de situaciones B2B.</p>	
<p>Lane</p>	<p>Una <i>lane</i> es una sub partición dentro de una <i>pool</i> y se extenderá a todo lo largo de la <i>pool</i>, ya sea vertical u horizontalmente. Las <i>lanes</i> se utilizan para organizar y</p>	

Elemento	Descripción	Notación
	categorizar actividades dentro de una <i>pool</i> .	

Tabla 5. Extensión elementos gráficos

1.4.1.8 Uso de Texto, Color, Tamaño y Líneas en un Diagrama

Los objetos que son anotaciones de texto pueden usarse para los modeladores para mostrar información adicional sobre los procesos o los atributos de un objeto dentro de un proceso.

- Los objetos de flujo y el flujo pueden tener niveles (por ejemplo, su nombre y/o otros atributos) colocado dentro de la forma, por encima o por debajo de la forma, en cualquier dirección o ubicación, dependiendo de las preferencias de la herramienta de modelado o del modelador.
- El relleno que se utiliza para los elementos gráficos puede ser de color blanco o un color claro. La notación se puede aplicar con el uso de otros colores para reflejar el propósito del modelador o de la herramienta (por ejemplo, para destacar el valor de un atributo de un objeto).
- Los objetos de flujo y los marcadores pueden ser de cualquier tamaño que se adapte a los efectos de la herramienta de modelado o al modelador.
- Las líneas que se utilizan para pintar elementos gráficos podrán ser negras. La notación se podrá extender y utilizar líneas de colores para reflejar el propósito del modelador o de la herramienta de modelado. También se podrá utilizar otro estilo de línea según sea el propósito, con la condición de que el nuevo estilo de línea no entre en conflicto con el estilo de línea definido por BPMN. Estos estilos de línea son el de flujo de secuencia, el de flujo de mensaje y el de asociación, que no pueden ser modificados.

1.4.1.9 Reglas de Conexión para los Objetos de Flujo

Un flujo de secuencia de entrada se puede conectar a cualquier lugar en un objeto de flujo (a la izquierda, derecha, arriba o abajo). Así mismo, un flujo de secuencia de salida puede conectarse desde cualquier lugar en un objeto de flujo (a la izquierda, derecha, arriba o abajo). El flujo de mensajes también tiene esta capacidad. BPMN permite esta flexibilidad, sin embargo, también recomiendan que los modeladores utilicen sentencias o las buenas prácticas definidas en el estándar, para

conectar los objetos de flujo de forma que los lectores de los diagramas se encuentren un comportamiento claro y fácil de seguir. Esto es aún más importante cuando contiene un diagrama de flujo y de secuencia de flujo de mensajes. En estas situaciones lo mejor es elegir una dirección del flujo de secuencia, ya sea de izquierda a derecha o de arriba a abajo y, a continuación dirigir el flujo de mensajes en un ángulo de 90° a la secuencia de flujo. El diagrama resultante será más fácil de entender.

1.4.1.10 Reglas del Flujo de Secuencia

En la tabla 6 muestra los objetos de flujo de BPMN y muestra como estos objetos pueden conectarse a través del flujo de secuencia. Se utiliza el símbolo de una flecha para indicar que el objeto que se lista en la fila puede conectarse con el objeto en la lista de las columnas. La cantidad de conexiones de entrada y de salida de un objeto está sujeto a la configuración de varias dependencias que no se van a especificar aquí. Hay que destacar que si un sub proceso se expande dentro de un diagrama los objetos dentro de este sub proceso no puede conectarse con los objetos que están fuera del sub proceso. Un flujo de secuencia no puede atravesar el límite de la *pool* a la que pertenece.

La tabla 6 muestra las reglas de conexión de los flujos de secuencia






































De/A						
						
						
						
						
						
						

Tabla 6. Reglas de Conexión Flujos de Secuencia

1.4.1.11 Reglas del Flujo de Mensajes

En la tabla 7 se muestra el modelado de objetos de BPMN y muestra como estos objetos se conectan con otro flujo de mensajes. El símbolo de una flecha hueca indica que el objeto que se lista en la fila se puede conectar con el objeto listado en la columna. La cantidad de conexiones de entrada y de salida de un objeto está sujeto a la configuración de varias dependencias que no se van a especificar aquí.

La tabla 7 muestra las reglas de conexión del flujo de mensaje:


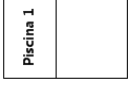





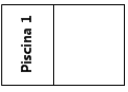





























De/A						
						
						
						
						
						
						

Tabla 7. Reglas de Conexión de Flujo de Mensajes

1.4.1.12 Herramientas para el modelado con BPMN

Debido a la gran expansión y uso de BPMN, se han desarrollado gran cantidad de herramientas para modelar los diagramas de BPMN. Se van a citar las más utilizadas y se va a exponer un ejemplo de la herramienta utilizada para realizar este estudio.

- *Business Process Visual ARCHITECT 3.1*: Es el más utilizado habitualmente por los analistas de negocio que están tratando de mejorar la eficiencia del proceso. Es una herramienta de modelado para la visualización, la comprensión, análisis, mejora y documentación de los procesos de negocio. En Visual Paradigm Suite se pueden encontrar distintas herramientas para el modelado de procesos de negocio según el lenguaje utilizado, además otras

herramientas para el modelado de otro tipo de diagramas y modelos. Se barajó utilizar esta herramienta para el modelado de procesos de negocio con BPMN, pero como más adelante se utilizaría para los diagramas de UML, se consideró mejor probar otra herramienta. En la figura 10 se muestra el patrón Unión Parcial de Cancelación, modelado con esta herramienta.

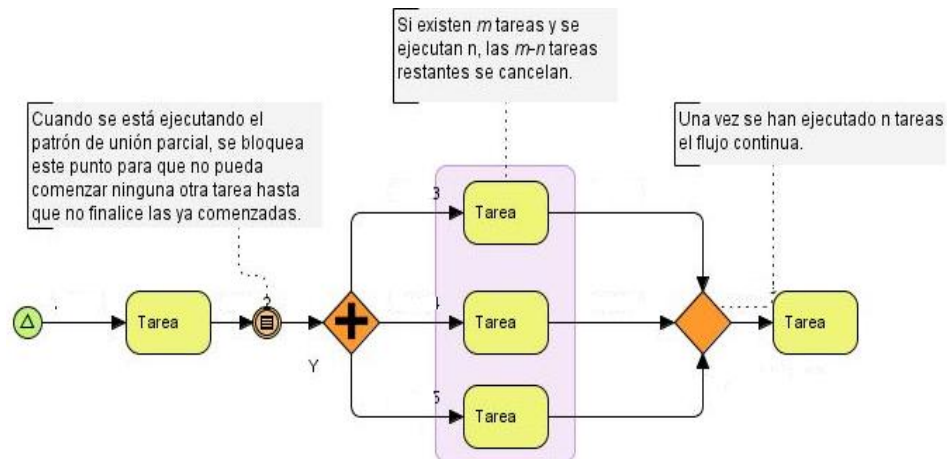


Figura 10. Patrón Unión Parcial de Cancelación

- *BizAgi Process Modeler Versión 14.0.0.162*: Esta herramienta modela procesos de negocio de un modo comprensible para los usuarios finales, para que luego sea automatizada, enganchándolos con una arquitectura de servicio (a través de servicios Web). Tiene la capacidad de simular el comportamiento de un sistema, medirlo y optimizarlo. Para este estudio, ésta ha sido la herramienta elegida, aunque la versión utilizada en este documento es una versión de prueba, que únicamente permite modelar los procesos de negocio.

Un ejemplo que se puede modelar con esta herramienta es la reserva de un viaje a través de Internet. Se definen las tareas de reserva de vuelo y reserva de hotel con la posibilidad de poder cancelar en todo momento tanto la reserva del vuelo como la del hotel. Si se produce un error el control se pasa a una tarea de la que se encarga el servicio de clientes, si se desconecta de la sesión se avisa mediante una notificación. En caso de que la reserva se realice correctamente se hace el cargo de la reserva al comprador. Este proceso de muestra en la Figura 11. Ejemplo de reserva viaje con BPMN.

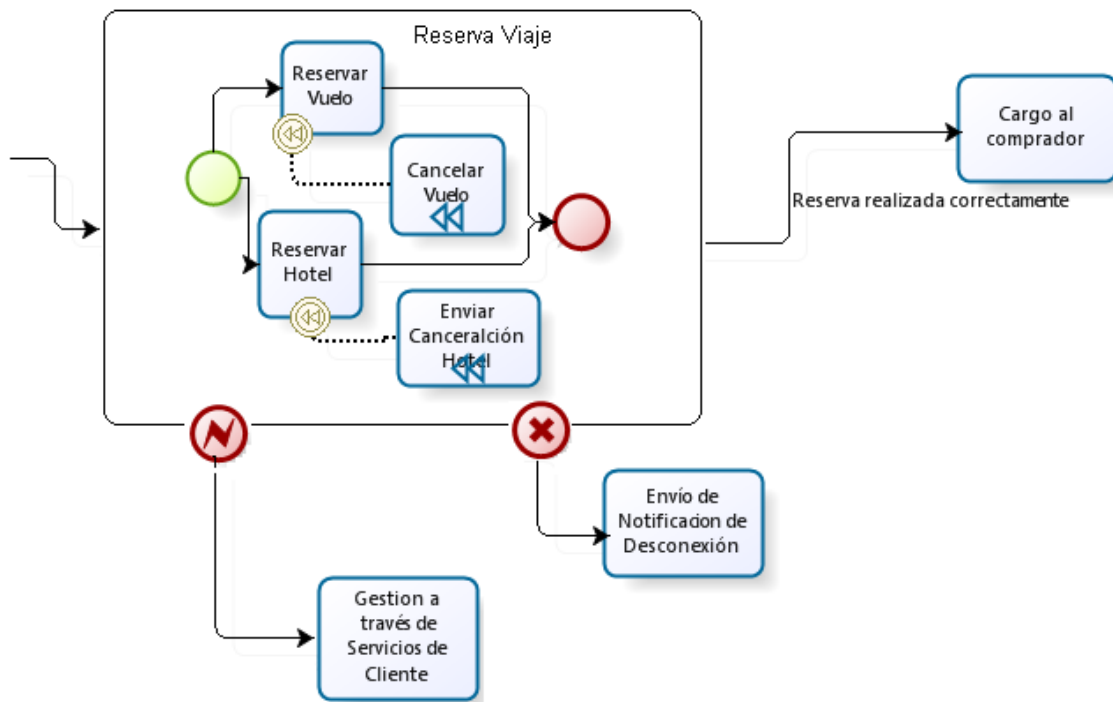


Figura 11. Ejemplo de reserva viaje con BPMN.

1.4.2 UML

El Lenguaje Unificado de Modelado (UML, del inglés *Unified Modeling Language*) es un lenguaje de modelado de sistemas de software y está respaldado por el OMG. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. Ofrece un estándar para describir una visión del sistema (metamodelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquema de bases de datos y componentes reutilizables. Resumiendo en otras palabras, es el lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software ofreciendo gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el *Proceso Unificado Racional* o RUP), pero no especifica en sí mismo qué metodología o proceso usar.

UML no puede compararse con la programación estructurada, pues UML significa lenguaje unificado de modelado, no es programación, solo se dibuja la realidad del uso de un requisito. Mientras que, la programación estructurada, es una forma de programar como lo es la orientación a objetos, sin embargo, la programación orientada a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML sólo para lenguajes orientados a objetos.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos y niveles de la realidad representada. De todos los diagramas que describe

UML, en este trabajo únicamente se va a centrar en los diagramas de actividad, los cuáles permiten modelar procesos de negocio o flujos de trabajo.

1.4.2.1 Introducción

El objetivo de UML es proporcionar una arquitectura de sistema, ingeniería del software, y desarrollo de software con herramientas de análisis, diseño e implementación basada en los sistemas de software tanto para modelado de procesos de negocio, como para procesos similares.

La primera versión de UML se origino con tres métodos orientados a objetos (*Booch*, *OMT*, y *OOSE*), que incorporaba mejores prácticas de lenguajes de modelado de diseños, programación orientada a objetos, y lenguajes de descripción de arquitectura. Relativo a UML 1.0, la revisión (sobre la que se base este documento 2.2) engancha significativamente con definiciones más precisas de las reglas de sintaxis abstractas y semánticas, un lenguaje estructural más modular, y mejora la capacidad para modelar sistemas a gran escala.

Una de las principales metas de UML es avanzar en el estado de la industria mediante la interoperabilidad con herramientas visuales de modelado. Sin embargo, para permitir el intercambio significativo entre la información del modelo y las herramientas, el acuerdo entre la semántica y la notación es obligatorio. Para ello, UML cumple los siguientes requisitos:

- Una definición formal de una estrategia común basada en el metamodelo de MOF (del inglés *Meta-Object Facility*) que especifica el resumen de la sintaxis de UML. La sintaxis abstracta define el conjunto de conceptos del modelado en UML, sus atributos y sus relaciones, así como las reglas para la combinación de estos conceptos para la construcción parcial o completa de los modelos UML.
- Una explicación detallada de la semántica de cada uno de los conceptos del modelado en UML. La semántica define, en una manera de tecnología independiente, como los conceptos de UML deben ser realizados por ordenador.
- Una especificación de la notación, fácilmente legible por las personas, para la representación de los elementos de los distintos conceptos de UML, así como reglas para combinar la variedad de diagramas correspondientes a distintos aspectos del modelado de sistemas.
- Una definición detallada de los caminos por los cuales las herramientas de UML pueden ser compatibles con la especificación. Esto se soporta (en una especificación separada), con una especificación basada en XML (del inglés *Extensible Markup Language*) de los correspondientes formatos de los modelos de intercambio o XMI (del inglés *Metadata Interchange*), que debe ser realizado por herramientas compatibles.

1.4.2.2 Conformidad

UML es un lenguaje con un alcance muy extenso que abarca un amplio y diverso conjunto de ámbitos de aplicación. No todas sus capacidades de modelado son necesariamente útiles en todos los dominios o aplicaciones. Esto sugiere que el lenguaje debe ser estructurado y modular, con la posibilidad de seleccionar únicamente aquellas partes del lenguaje que son de interés directo. Por otro lado, un exceso de este tipo de flexibilidad, aumenta la probabilidad de que dos herramientas diferentes de UML den el apoyo a diferentes subconjuntos del lenguaje, dando lugar a problemas de intercambio entre ellos. En consecuencia, la conformidad o aceptación de la definición de un modelo UML, requiere un equilibrio que debe establecerse entre la modularidad y la facilidad de intercambio.

La experiencia con las versiones anteriores de UML, ha demostrado que la posibilidad de intercambiar modelos entre herramientas, es un instrumento de gran interés para la gran comunidad de usuarios. Por esa razón, la última especificación de UML 2.2 [44] define un pequeño número de niveles de cumplimiento, lo que aumenta la probabilidad de que dos o más herramientas compatibles con herramientas de soporte a UML sean compatibles con subconjuntos de lenguajes. Sin embargo, en el reconocimiento de la necesidad de flexibilidad en el aprendizaje y el uso del lenguaje, UML también proporciona el concepto de unidades de lenguaje.

1.4.2.3 Unidades de Lenguaje

Los conceptos de modelado de UML se agrupan dentro de unidades de lenguaje. Una unidad de lenguaje consiste en una colección fuertemente acoplada de conceptos de modelado que proporcionan a los usuarios la posibilidad de representar aspectos del sistema acorde a un paradigma o formalismo en particular. Por ejemplo las unidades del lenguaje de las máquinas de estado, permite a los modeladores especificar el comportamiento discreto de la dirección de los eventos usando, una variante del formalismo del conocimiento de los estados gráficos, mientras que las unidades del lenguaje de arquitectura proporcionan para modelar el comportamiento, el paradigma basado en flujos de trabajo. Para la perspectiva del usuario, esta división de las formas o técnicas de UML que ellos necesitan, solo les concierne con la parte del lenguaje que ellos consideren necesaria para sus modelos. Si estas necesidades cambian en el tiempo, se añadirán tantas unidades del lenguaje como el usuario requiera. Sin embargo, un usuario de UML no tiene porqué conocer por completo el lenguaje para usarlo de manera efectiva.

Además, cada unidad de trabajo se divide en múltiples partes incrementales, que añaden más capacidades y características a las unidades de las que parten. Esta descomposición de UML sirve para hacer el lenguaje más fácil de aprender y usar, pero cada uno de los segmentos individuales dentro de esta estructura no representa el cumplimiento de los puntos anteriores por separado. Esta estrategia puede llevar a que se cumplan más puntos de los necesarios y produciría problemas de interoperabilidad.

Sin embargo, las agrupaciones proporcionadas por las unidades del lenguaje y sus ampliaciones pueden servir para simplificar el cumplimiento de la definición de UML que se va a describir más adelante.

1.4.2.4 Niveles de Cumplimiento

La estratificación de las unidades de lenguaje es utilizada para definir la conformidad en UML. A saber, que el conjunto de conceptos de modelado de UML se dividen en niveles que aumentan su capacidad horizontalmente, a esto se le llama *Niveles de Conformidad*.

Para facilitar el intercambio de modelos, hay sólo cuatro niveles de conformidad definidos para todo modelo de UML. Los niveles son los siguientes:

- *Nivel 0 (L0)*: Este nivel de cumplimiento se define formalmente en la infraestructura de UML. Únicamente contiene una unidad lingüística que proporciona al modelado de los distintos tipos de clases las estructuras que se encuentran en los lenguajes de programación orientados a objetos. Como tal, proporciona una entrada a la capacidad del nivel de modelado. Lo que es más importante, es que representa un denominador común de coste bajo que puede servir de base para interoperabilidad entre las diferentes categorías de herramientas de modelado.
- *Nivel 1 (L1)*: este nivel añade nuevas unidades de lenguaje y extiende las capacidades que proporciona el nivel 0. Concretamente añade unidades de lenguaje para usar casos, interacciones, estructuras, acciones y actividades. En este nivel es donde entran los diagramas de actividad, que son los que interesan para este trabajo.
- *Nivel 2 (L2)*: este nivel extiende las unidades del lenguaje ya proporcionadas en el nivel 1 y añade unidades de lenguaje para el despliegue, para el modelado de una máquina de estados y perfiles.
- *Nivel 3 (L3)*: este nivel representa por completo UML. Extiende las unidades del lenguaje que proporciona el nivel anterior y añade nuevas unidades de lenguaje para modelar flujos de información, plantillas y paquetes de modelos.

El contenido de las unidades de lenguaje, son definidas por los paquetes del nivel superior del metamodelo de UML, mientras que el contenido de los distintos incrementos se define en los paquetes de segundo nivel dentro de los paquetes de las unidades del lenguaje. Por lo tanto, el contenido de un nivel de conformidad está definido por el conjunto de paquetes del metamodelo al nivel al que pertenece.

Como se ha señalado, los niveles de conformidad se basan en el soporte de estos. El principal mecanismo utilizado en esta especificación para conseguir este paquete es la fusión. Permite combinar el paquete de modelado conceptos definidos en un nivel que será ampliado con nuevas funcionalidades. Lo que es más importante, esto se logra en el contexto dentro del mismo entorno, que permite el intercambio de modelos con diferentes niveles de conformidad.

Por esta razón, todos los niveles de conformidad, se combinaron en un único núcleo de un paquete de modelos UML que define el nombre común que comparten todos los niveles de conformidad.

1.4.2.5 Significado y tipos de conformidad

La conformidad en un nivel conlleva la completa realización de todas las unidades del lenguaje que se definen en el nivel de conformidad. Sin embargo, esto implica la completa realización de todas las unidades de lenguaje en todos los niveles por debajo. La completa realización de las unidades del lenguaje en un nivel determinado significa que soporta el conjunto completo de los conceptos de modelado para ese nivel.

Una herramienta que es compatible con un determinado nivel, debe ser capaz de importar modelos de herramientas que cumplan con niveles más bajos sin pérdida de información.

Hay dos tipos distintos de conformidad. Estos son:

1. *Cumplimiento de sintaxis abstracta*. Para un determinado nivel de conformidad esto supone:
 - El cumplimiento de las metaclasses, sus relaciones estructurales y las limitaciones que se define como parte de la fusión del metamodelo de UML para el nivel de conformidad,
 - La capacidad de producir y leer modelos basados en el esquema correspondiente a XMI para el correspondiente nivel de conformidad.
2. *Cumplimiento de sintaxis concreta*. Para obtener conformidad en un nivel, esto supone:
 - El cumplimiento de la notación. Se define en la notación de las subclases de esta especificación para los elementos de este metamodelo que están definidos como parte de la fusión del metamodelo del nivel de conformidad y, por la implicación, los tipos de diagramas en los que aparecen estos elementos. Y opcionalmente:
 - La capacidad de producción y lectura de diagramas basados en esquemas XMI definidos por el diagrama de intercambio de la notación de ese nivel. Esta opción requiere la conformidad con la sintaxis abstracta y concreta.

La conformidad con la sintaxis concreta no requiere el cumplimiento de las opciones antes presentadas como parte de la notación.

La conformidad en un nivel se puede expresar como:

- Conformidad de sintaxis abstracta.
- Conformidad de sintaxis concreta.

- Conformidad de sintaxis abstracta con sintaxis concreta.
- Conformidad de sintaxis abstracta con sintaxis concreta y conformidad con diagramas de intercambio.

En el caso de las herramientas que generan código de programa de los modelos o las que son capaces de ejecutar los modelos, también es útil para comprender el nivel de soporte a la semántica, los tiempos de ejecución descritos en la semántica de las cláusulas descritas en el pliego de condiciones. Sin embargo. La presencia de numerosos puntos de variación en esta semántica (y el hecho de que informalmente se definen utilizando el lenguaje natural), que sea práctico para definir esto como un tipo formal, ya que el número de combinaciones posibles es muy grande.

Una situación similar existe con las opciones presentadas, ya que diferentes ejecutores pueden hacer distintas elecciones de las soportadas. Por último, se reconoce que el perfil de alguno de estos ejecutores y diseñadores, soporta un subconjunto de las características de los niveles que están por encima del nivel de conformidad formal. Nótese, sin embargo, que sólo puede reclamar el nivel de conformidad que soporta por completo, incluso si la aplicación soporta una gran parte de niveles superiores. Dado el potencial de variedad, es útil que se sea capaz de especificar con claridad y eficiencia, los niveles que se soportan por una determinada aplicación. Con este fin, además de una declaración formal de conformidad, el perfil de los diseñadores y ejecutores podrán también proporcionar soporte informal a las características declaradas. Estas declaraciones de soporte para identificar las características adicionales en términos de unidades del lenguaje y/o del paquete individual del metamodelo, así como una definición precisa de las dimensiones, como la presentación de opciones y puntos de variación semántica.

1.4.2.6 Semántica en Tiempo de Ejecución de UML

El término *en tiempo de ejecución* se utiliza para referirse al entorno de ejecución. La *semántica del tiempo de ejecución*, por lo tanto, se especifica como un mapeo de los conceptos de modelado correspondientes a los fenómenos de ejecución de un programa. Hay, por supuesto, otras especificaciones de semántica relevantes en UML, como la especificación de la superestructura de UML, la semántica de repositorio, que es, como se comporta el modelo de UML en un repositorio de un modelo. Sin embargo, la semántica realmente es la parte de definición del MOF. Sin embargo hay que destacar que no todos los conceptos en los modelos UML es un fenómeno en tiempo de ejecución (por ejemplo, el concepto de *paquete*).

1.4.2.7 La Arquitectura de la Semántica

La figura 12 identifica las áreas claves de la semántica, cubiertos por la norma actual y cómo se relacionan entre sí. Los elementos en las capas superiores dependen de los productos de las capas inferiores, pero no al revés.

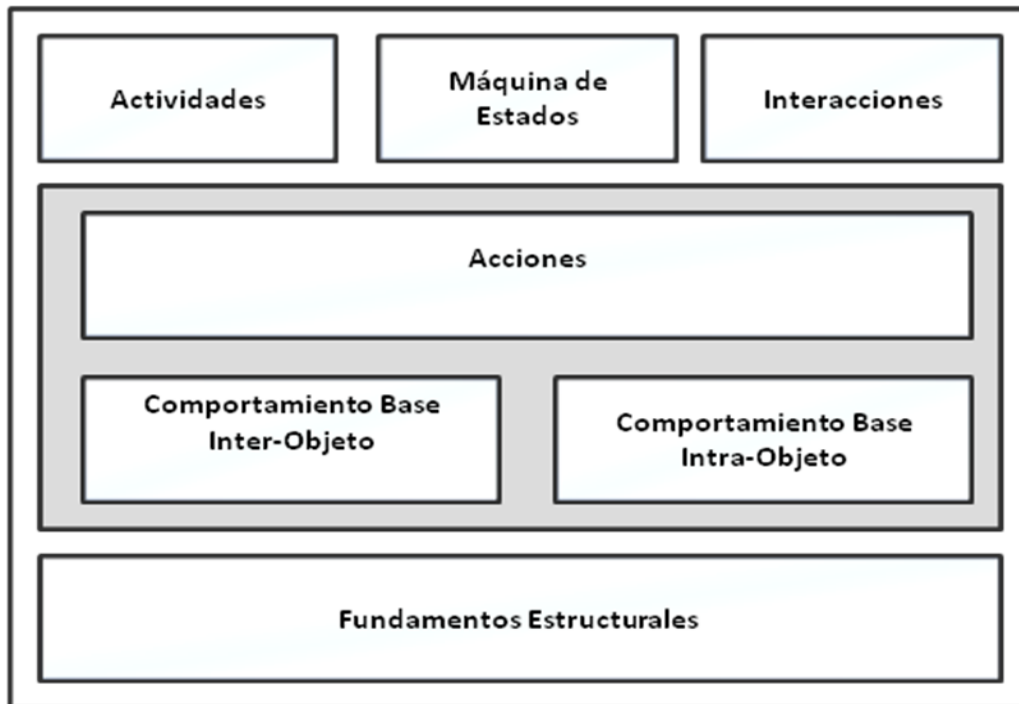


Figura 12. Arquitectura semántica

En más alto nivel de abstracción, es posible distinguir tres estratos distintos compuestos de las definiciones semánticas. El nivel fundamental es el estructural. Este refleja la premisa de que no hay ningún comportamiento incorpóreo en UML (todo el comportamiento es la consecuencia de las acciones de las entidades estructurales).

El siguiente nivel es el comportamiento y proporciona la base para la descripción semántica de todos los demás niveles superiores de *formalismo de comportamiento* (el término formalismo de comportamiento se refiere a un marco formal para describir el comportamiento, tales como máquinas de estado, redes de Petri, gráficos de flujos de datos, etc.). Este nivel representado en la figura 10 por el cuadro sombreado, es la base semántica de comportamiento y consiste en tres sub-áreas separadas distribuidas en dos sub-niveles. El sub-nivel consiste en el comportamiento base de los inter-objetos, que trata de cómo las entidades estructurales se comunican unas con otras, y dentro de la base de la conducta de los objetos, lo que se refiere a la conducta que tiene lugar dentro de las entidades estructurales. El sub-nivel de acciones se coloca en la parte superior de estos dos niveles. Se define la semántica de las acciones individuales. Las acciones son las unidades fundamentales de comportamiento en UML y se utilizan para definir los comportamientos de grano fino. Su resolución y el poder expresivo son comparables a las instrucciones del lenguaje de programación tradicional. Las acciones en este nivel están disponibles para cualquiera de los formalismos de alto nivel que se utiliza para describir las conductas detalladas.

El nivel superior en la jerarquía semántica define la semántica de los formalismos de comportamiento de alto nivel de UML: actividades máquinas de estado y las interacciones. Otros formalismos de comportamiento pueden ser agregados a este nivel en el futuro.

1.4.2.8 Conceptos de UML – Diagramas de Actividad

Los conceptos de UML se agrupan en tres grandes partes:

- Parte I: Conceptos relacionados con el modelado de estructuras.
- Parte II: Conceptos relacionados con el modelado del comportamiento.
- Parte III: Conceptos adicionales.

En cada una de las partes los conceptos están agrupados en cláusulas acordes a la capacidad de modelado. Típicamente una capacidad cubre un formalismo específico de modelado. Por ejemplo, todos los conceptos relacionados con las capacidades de modelado de máquinas de estados, se reúnen en la cláusula de Máquinas De Estado y todos los aspectos relacionados con las capacidades de modelado de las actividades son cláusulas de Actividades. Las cláusulas de Capacidad en cada una de las partes se presentan ordenadas alfabéticamente.

En este documento, se va a exponer una breve descripción formal de cada uno de los conceptos, sin entrar en más detalle de la representación gráfica del concepto. A continuación se va a describir la parte de UML que hace referencia a los flujos de trabajo, es decir el comportamiento, en lo que se refiere a los diagramas de Actividad.

BEHAVIOR - ACTIVITY

El modelado de actividades hace hincapié en la secuencia y las condiciones de coordinación de comportamientos de menor nivel, en lugar de que los clasificadores propios del comportamiento lo hagan. Estos son, comúnmente llamados comportamientos de flujo de control y modelos de objetos de flujo. Las acciones coordinadas por los modelos de actividad se pueden iniciar al terminar la ejecución de otras acciones, porque los objetos y los datos estén disponibles o por que se producen acontecimientos externos al flujo normal.

La ejecución de una acción corresponde a la ejecución de una acción en particular. Igualmente, la ejecución de una actividad es la ejecución de una actividad en particular, a la larga incluye la ejecución de acciones dentro de ellas. Cada acción, dentro de una actividad, puede no ejecutarse, o ejecutarse, una o más de una vez en la ejecución de una actividad. Como mínimo, las acciones necesitan acceso a los datos, necesitan transformarlos y probarlos, y pueden requerir también la secuencia. La especificación de las actividades permite la ejecución de varios hilos de control a la vez y los mecanismos de sincronización para asegurar que las actividades se ejecutan en el orden especificado. La semántica basada en la ejecución concurrente se puede asignar fácilmente a una aplicación distribuida. Sin embargo, el hecho de que UML permite la ejecución de objetos al mismo tiempo, no implica necesariamente una distribución de la estructura de Software. Algunas implementaciones pueden agrupar los objetos en una sola tarea y ejecutar de forma secuencial, siempre y cuando el comportamiento de la aplicación se ajuste a las limitaciones de la secuencia especificada.

Existen muchas formas de implementar la misma especificación y que preserve la información sobre el contenido y el comportamiento que se acepta en la especificación. Debido a que la aplicación puede tener una estructura diferente de la de la especificación, existe una correspondencia entre la especificación y su aplicación. Esta asignación no tiene que ser de uno a uno: una aplicación no necesita utilizar la orientación a objetos, y puede elegir un conjunto diferente de clases de la especificación original.

La especificación [44] no proporciona las sobreimpresiones, ni prevé la generación de código de forma explícita, pero el pliego de condiciones muestra dos enfoques diferentes. A continuación se describen los paquetes que utilizan los diagramas de actividad en UML.

FundamentalActivities: El nivel fundamental define como las actividades contienen los nodos, que incluye acciones. Este nivel es compartido entre el flujo y las formas estructuradas de actividades.

BasicActivities: Este nivel incluye la secuencia de control y de flujo de datos entre las acciones. No especifica, las bifurcaciones explícitas y las divisiones de control, así como las decisiones y las fusiones. Los niveles se basan en estructuras y son ortogonales. Cualquiera puede ser utilizado sin la otra o ambas se pueden utilizar para el soporte de modelado, que incluye tanto los flujos y el control estructurado de las construcciones.

IntermediateActivities: El nivel intermedio, soporta el modelado de diagramas de actividad que incluyen el control concurrente y el flujo de datos, y decisiones. Es compatible con modelos similares a las redes de Petri con cola. Se requiere el nivel básico.

Los niveles intermedios y estructurados son ortogonales. Cualquiera puede ser utilizado sin la otra o ambas pueden ser utilizados para el soporte de modelos que incluyen tanto concurrencia y construcciones de estructuras de control.

CompleteActivities: El nivel completo, añade construcciones que mejoran los modelos de nivel inferior, tales como pesos en los arcos y canalización (del inglés Streaming, que se puede definir como generación de flujo en una dirección).

StructuredActivities: El nivel estructural soporta el modelado tradicional de la construcción con programación estructurada, tales como la secuencia, bucles y condicionales, como una adición a los nodos de *FundamentalActivities*. Se requiere el nivel fundamental. Es compatible con el nivel intermedio y con el nivel completo.

CompleteStructuredActivities: Este nivel incluye soporte para el flujo de salida de datos identificadores, condicionales y bucles o circuitos. Depende del nivel base por los flujos.

ExtraStructuredActivities: El nivel de extra estructuras soporta la ejecución de manejadores de excepciones encontrados en lenguajes de programación tradicionales y

la invocación de comportamientos con un conjunto de valores. Esto requiere el nivel estructurado.

A continuación se listan todas las clases que contiene el paquete de Actividades. Además se describirán cada una de las clases que componen este paquete y la notación (un símbolo si procede y una descripción o no aplica).

1. **AcceptEventAction:**

Descripción: *AcceptEventAction* es una acción que espera la presencia o el acontecimiento de un evento cuando se cumple una condición específica.

Notación: Esta acción se puede representar con un pentágono cóncavo. Una acción con tiempo de espera se simboliza con la forma de un reloj de arena, como se muestra en la figura 13.

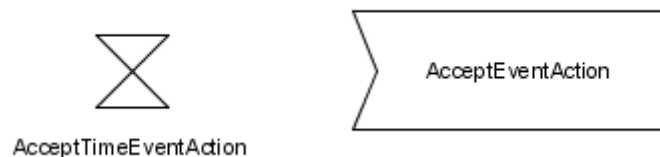


Figura 13. AcceptEventAction

2. *Action*

Descripción: *Action* es una *activity* que representa un simple o único paso dentro de una *activity*, y que no es un rasgo desglosado dentro de la *activity*. Una *activity* representa un comportamiento que se compone de elementos individuales que son las *actions*. Sin embargo, la llamada a un comportamiento de una *action*, puede hacer referencia una *activity*, en cuyo caso la ejecución de la llamada a la *action* envuelve la ejecución de la *activity* y sus *actions* (similar para toda invocación de *actions*). Por lo tanto, una *action* simple desde el punto de vista de la *activity* que contiene la *action*, pero puede ser compleja en sus efectos y no ser atómica. Como una pieza de una estructura dentro de un modelo de *activity*, es un elemento discreto único; como una especificación de su comportamiento, podrá recurrir, arbitrariamente, al comportamiento complejo. Como consecuencia, una *activity* define un comportamiento que puede reutilizarse en otros lugares, mientras que una instancia de una *action* se utilice una vez en un punto particular de la *activity*.

Una *action* puede tener arcos de entrada y salida de la *activity*, que especifican el control de flujo y flujo de datos desde y hacia otros nodos. La ejecución de una *action* no se iniciará hasta que la totalidad de sus condiciones de entrada sean satisfechas. La finalización de la ejecución de una *action* puede permitir la ejecución de un conjunto de nodos sucesores y *actions* que tienen entradas del resultado de la *action*.

El paquete de *CompleteActivities*, permite a la *action* tener pre y post condiciones.

Notación: El uso de la notación de una *action* o de la de una *activity* es opcional. Una anotación de texto se puede utilizar en su lugar. Las *actions* se anotan como un

rectángulo con las esquinas redondeadas. El nombre de la acción u otra descripción de la misma puede aparecer en el centro del símbolo como se muestra en la figura 14.

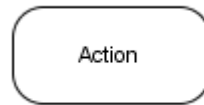


Figura 14. Action

Las precondiciones y postcondiciones se muestran como notas adjuntas a la invocación con la palabra clave `<<localPrecondition>>` y `<<localPostcondition>>` respectivamente. Esto se ve en la figura 15.

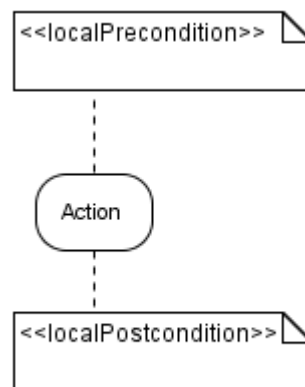


Figura 15. Action con condiciones.

3. ActionInputPin

Descripción: *ActionInputPin* es un tipo de *pin* de entrada de una *action* es una clase de ejecución que determina la entrada de otros.

Notación: Una acción que introduce un *pin*, con una *action ReadVariableAction* como una clase *fromAction* se simboliza como un identificador de entrada con el nombre de la variable escrita junto a él. Una *action* que introduce un *Pin* con un *ReadSelfObject* como un *fromAction* se anota como un *Pin* de entrada con la palabra “*self*” escrita a su lado. Un identificador de entrada de acción con un *ValueSpecification* como un *fromAction* se muestra como un *pin* de entrada con el valor específico escrito al lado de él. No tiene una simbología o un icono asociado.

4. Activity

Descripción: *Activity* es la especificación del comportamiento parametrizado como la secuencia coordinada de las unidades subordinadas cuyos elementos individuales son las *actions*.

Una *activity* especifica la coordinación de las ejecuciones de los comportamientos subordinados, usando un modelo de control y de flujo de datos. Los comportamientos subordinados coordinados por estos modelos pueden ser iniciados por

otros comportamientos en el modelo final de ejecución, ya que los objetos y los datos estén disponibles, o porque se producen acontecimientos externos a la corriente. El flujo de ejecución se modela como nodos de *activity* relacionada de un comportamiento subordinado, como un cálculo aritmético, una llamada a una operación, o la manipulación de los contenidos de un objeto. Los nodos de *activity* también incluyen flujo de control de construcciones, como la sincronización, decisión y control de la concurrencia. Las *activities*, en última instancia, son la solución de las *actions* individuales. En un modelo orientado a objetos, las *activities* son generalmente aludidas indirectamente, como métodos vinculados a las operaciones que son invocadas directamente.

Las *activities* pueden describir un procedimiento computacional. En este contexto, son los métodos correspondientes a las operaciones en las clases. Las *activities* pueden ser aplicadas a modelos de organización de ingeniería de procesos de negocio y flujo de trabajo. En este contexto, los acontecimientos a menudo se originan desde el interior del sistema, tales como la terminación de una tarea, sino también de fuera del sistema, tales como una llamada de un cliente. Las *activities* también pueden ser utilizadas para el sistema de modelado de información para especificar los procesos de nivel de sistema.

Las *activities* pueden incluir *actions* de varios tipos:

- Los casos de las funciones primitivas, como las funciones aritméticas.
- Invocaciones de la conducta, tales como las *activities*.
- *Actions* de comunicación, como el envío de señales.
- La manipulación de objetos, tales como leer o escribir los atributos o asociaciones.

Las *actions* no tienen una descomposición mayor de la *activity* que los contiene. Sin embargo, la ejecución de una acción individual puede provocar la ejecución de muchas otras *actions*. Por ejemplo, una acción de llamada, invoca a una operación que se lleva a cabo por una *activity* que contiene las *actions* que se ejecutan antes de que finalice la acción de llamada.

La mayoría de las construcciones en la cláusula de *activity* tienen diversos mecanismos para ordenar el flujo de control y los datos entre las *actions*, como son:

- Los objetos de flujo para secuencias de ejecución de nodos.
- Los flujos de control para secuencia de la ejecución de nodos.
- Nodos de control de la estructura de control y los objetos de flujo. Estas incluyen las decisiones y se une a la contingencia del modelo. Estos también incluyen los nodos iniciales y finales para iniciar y finalizar los flujos. En las *IntermediateActivities*, incluyen bifurcaciones y fusiones para crear y sincronizar las ejecuciones que surjan de manera concurrente.

- La generalización de la *Activity* para reemplazar nodos y arcos.
- Nodos de objeto para representar objetos y datos que fluyen dentro y fuera de los comportamientos invocados, o para representar la colección de tokens que están esperando para pasar a un nivel inferior.

El paquete de *StructuredActivities* está compuesto de nodos que representan flujos de control estructurados, como bucles y condicionales.

El paquete de *IntermediateActivities* proporciona particiones para organizar las *activities* de menor nivel, de acuerdo a diversos criterios, tales como la organización responsable del ciclo de vida del modelo y de su rendimiento.

El paquete de *CompleteActivities* proporciona regiones interrumpibles y excepciones para representar desviaciones para el principal flujo de control normal.

Notación: El uso de la notación de *activity* y de *action* es opcional. Se puede utilizar una anotación de texto en su lugar. La notación para una *activity* es una combinación de las anotaciones de los nodos y arcos que contiene, además de una frontera y el nombre aparece en la esquina superior izquierda. Los nodos de parámetros de *activity* se muestran en el borde. Las *actions* y los flujos que contiene la *activity* también están representados. Se muestra esta notación en la figura 16.

Las restricciones pre y post condicionales, heredadas del *Behaviour*, se muestran con la palabra clave <<precondition>> y <<postcondition>>, respectivamente. Estos principios se utilizan globalmente para todos los usos de una *activity*.

La palabra clave <<SingleExecution>> se utiliza para *activities* que ejecutan como una ejecución única compartida; de lo contrario, cada llamada se ejecuta en su espacio.

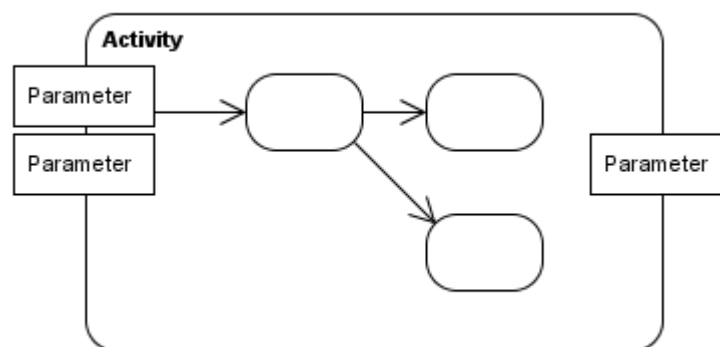


Figura 16. Activity

Las *activities* se introdujeron para coordinar el flujo de los modelos que tienen otros comportamientos, incluyendo otros modelos de flujo. Son compatibles con características de clase para controlar el modelo y el seguimiento de la ejecución de procesos, y relacionarlos con otros objetos (por ejemplo, en un modelo de organización).

5. ActivityEdge

Descripción: *ActivityEdge* es una clase abstracta con conexiones directas entre dos *ActivityNode*. Una *ActivityEdge* se puede definir como los arcos que se utilizan para unir los distintos conceptos de UML, que pueden interconectarse. Se contempla el control y los datos de flujo entre arcos.

Notación: Una *ActivityEdge* se denota por una flecha abierta y una línea que conecta dos nodos de *activity*, como se muestra en la figura 17. Si el arco tiene nombre, este se anota cerca de la flecha.

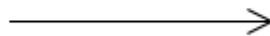


Figura 17. ActivityEdge

6. ActivityFinalNode

Descripción: *ActivityFinalNode* es un tipo de nodo para indicar el final de una *activity*. Una *activity* puede tener más de un nodo final de *activity*. La primera *activity* se detiene cuando todos los flujos han llegado al *ActivityFinalNode*.

Notación: Estos nodos se representan como un círculo sólido con una parte hueca, como se indica en la figura 18. Puede representarse como un objetivo con el centro blanco o como destino.



Figura 18. ActivityFinalNode

7. ActivityGroup

Descripción: *ActivityGroup* es un grupo genérico constructor de nodos y arcos. Los nodos y los arcos pueden pertenecer a más de un grupo. Estos elementos no tienen la semántica inherente y pueden ser utilizados para diversos fines.

Notación: No aplica. No existe ningún icono relacionado con esta clase.

8. ActivityNode

Descripción: *ActivityNode* es un nodo de actividad, que representa una clase abstracta de puntos en el flujo de una *activity* relacionada con los arcos. Un nodo de *activity* es una clase abstracta para representar los pasos de una *activity*. Cubre los nodos ejecutables, nodos de control y los nodos objetos.

Notación: Existen tres tipos de nodos: *ActionNode*, *ObjectNode* y *ControlNode*. En la figura 19 se muestra su representación, donde los *DecisionNode*, *ForkNode*, *InitialNode*, *ActivityFinalNode* y *FlowFinalNode* son *ControlNode*.

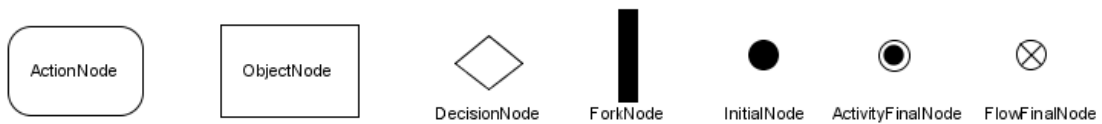


Figura 19. ActivityNode

9. ActivityParameterNode

Descripción: *ActivityParameterNode* es un nodo objeto con parámetros para las entradas y salida a las *activities*. Son nodos objeto del principio y de final de los flujos que proporcionan una forma de aceptar entradas para una *activity* y salidas de una *activity*, a través de los parámetros de *activity*.

Notación: Estos *ActivityParameterNode* se representan con rectángulos. El nombre de etiqueta se coloca dentro del símbolo, donde el nombre lo indica el tipo de nodo objeto, o el nombre y el tipo del nodo en formato "ParametreName:type". Los nodos objetos cuyos casos son conjuntos de tipo "name", están etiquetados como tales tal y como se muestra en la figura 20.

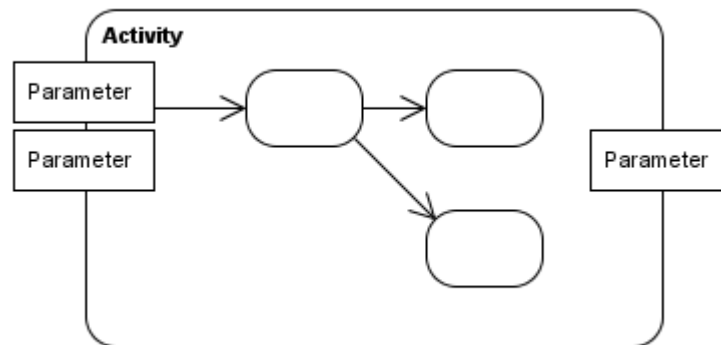


Figura 20. ActivityParametreNode

Para mostrar la notación de los parámetros de *activities* de cadenas y excepciones, se representan igual que con *Pin*, tal y como se muestra en la figura 21.

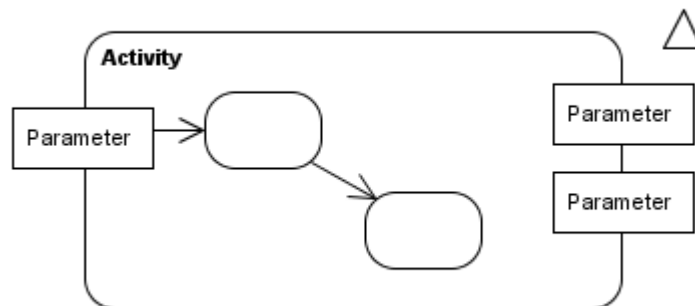


Figura 21. ActivityParameterNode

10. ActivityPartition

Descripción: *ActivityPartition* es un tipo de *ActivityGroup* para identificar *activities* que tienen alguna característica en común. Son particiones de *activities*.

Las particiones dividen los nodos y los arcos por sus restricciones y muestra una vista de los nodos que contiene. Una partición puede conservar o mantener el contenido. Esto suele corresponder con unidades organizacionales en un modelo de negocio. Se puede utilizar para mostrar características o recursos sobre los nodos de una *activity*.

Notación: Por lo general la partición de una *activity* se podría representar con dos líneas paralelas, ya sea horizontal o vertical, y una etiqueta con el nombre de la partición en los extremos del cuadro como se muestra en la figura 22 a). Cualquier nodo *activity* y borde situado entre estas líneas se consideran contenidos de la partición. Las *Swimlanes* pueden expresarse con una división jerárquica que se expresa mediante la representación de los hijos en la jerarquía como una partición más de la partición de los padres. Figura 22 (b) Los diagramas se pueden dividir de forma multidimensional, llamadas *pool*. Además cada celda de la *pool* es una intersección de varias particiones. Figura 22 (c) La especificación para cada dimensión se expresa en el extremo del conjunto de las particiones correspondientes.

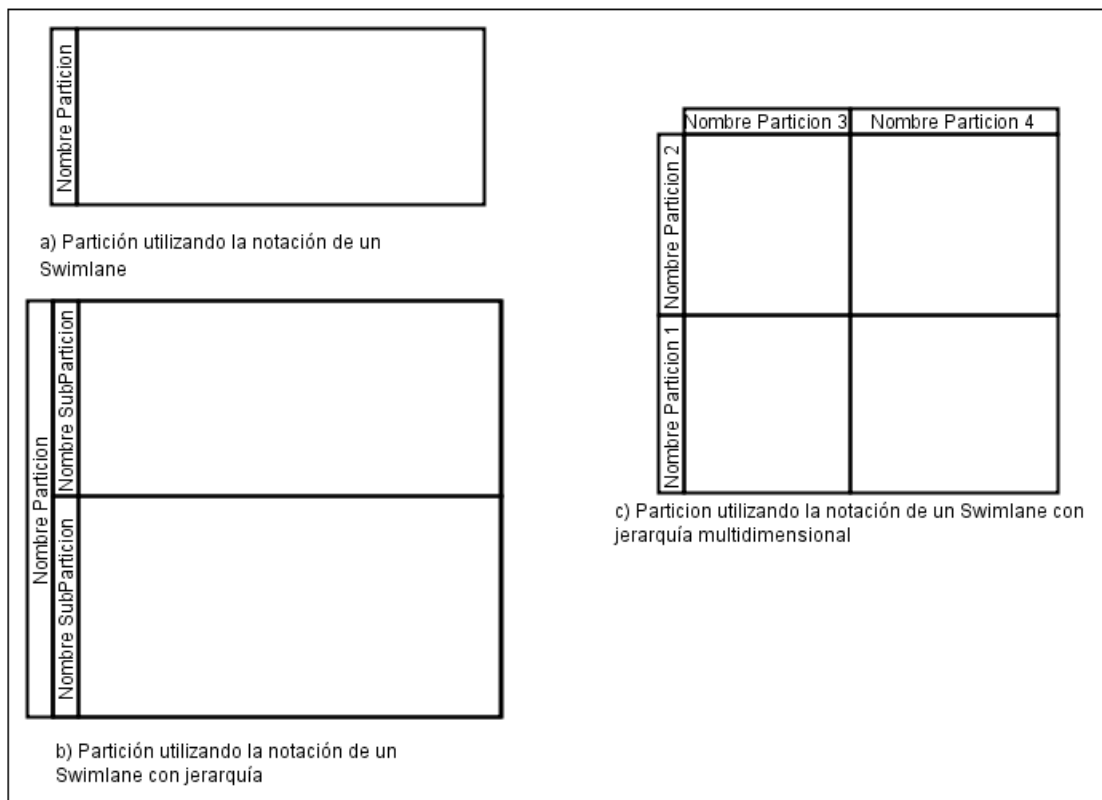


Figura 22. ActivityPartición

En algunas situaciones de diagramas es práctico usar líneas paralelas para delimitar las particiones. Una alternativa es colocar el nombre de la partición entre paréntesis sobre el nombre de la *activity*. Una lista de nombres de particiones separados

por comas, significa que el nodo está contenido en más de una partición. Dos puntos dobles (::) dentro del nombre de la partición indica que está anidada, con particiones más grandes con el nombre cerca de este. Cuando las *activities* se considera que ocurren fuera del dominio de un modelo particular, la partición se puede etiquetar con la palabra clave <<external>>. Siempre que una *activity* en una *pool* esté marcada con la palabra <<external>>, esto anula la *pool* y la designación de dimensión.

11. AddVariableValueAction

Descripción: *AddVariableValueAction* es un tipo *action* de escritura de variables para añadir valores a las variables.

Notación: En la figura 23 se muestra este tipo de *actions*. Si la *action* tiene valores de meta atributos que no son por defecto, se pueden mostrar con una lista de nombres de variables.

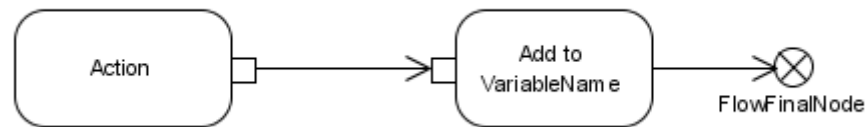


Figura 23. AddVariableValueAction

12. Behavior

Descripción: *Behavior* es una especialización que puede no tener parámetros o tener un conjunto de ellos. El concepto de comportamiento se extiende al conjunto propio de parámetros. Un comportamiento con un conjunto de parámetros de entrada puede solo aceptar una entrada de parámetros por ejecución. Un comportamiento con un conjunto de parámetros de salida solo puede poner un conjunto de parámetros de salida por ejecución.

Notación: Ver la notación de la clase *ParameterSet* en figura 44.

13. BehavioralFeature

Descripción: Es una especialización para cero o más conjunto de parámetros propios.

Notación: Ver la notación de la clase *ParameterSet* en figura 44.

14. CallBehaviorAction

Descripción: Una *CallBehaviourAction* es un tipo de *action* que llama al comportamiento específico de una *action*.

Notación: Se representa con un rectángulo con las esquinas redondeadas y el nombre del comportamiento en el centro, figura 24. Se puede también representar las

precondiciones y postcondiciones, mediante notas y las palabras clave <<precondition>> y <<postcondition>> tal y como se muestra en la figura 25.

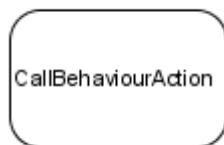


Figura 24. CallBehaviourAction

Una notación alternativa en el caso de una *action* invocadora, es mostrar el contenido de la *activity* invocada dentro de un rectángulo más grande. Los nodos de objetos de parámetros se muestran en el borde de la *action* invocada. El modelo es el mismo independientemente de la elección de la notación. La especificación de UML 2.2 soporta el intercambio de elementos del diagrama y el mapeo de los elementos del modelo. La llamada de una *activity* se representa mediante la colocación del símbolo de un tridente en el rectángulo tal y como se muestra en la figura 25.

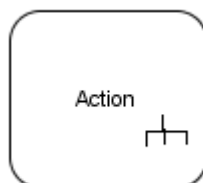


Figura 25. CallBehaviourAction

15. CallOperationAction

Descripción: CallOperationAction es una *action* que llama a una operación.

Notación: Se representa también con un rectángulo con las esquinas redondeadas, figura 26.

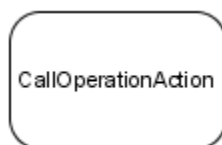


Figura 26. CallOperationAction

Si un nodo tiene un nombre diferente al de la operación, entonces se pone el nombre de la operación entre paréntesis. El nombre de la clase puede aparecer opcionalmente antes del nombre de la operación.

16. CentralBufferNode

Descripción: CentralBufferNode es un objeto nodo para la gestión de los flujos que provienen de múltiples fuentes y tienen múltiples destinos.

Notación: Se representa con un cuadrado y el nombre del nodo en el centro tal y como se muestra en la figura 26.

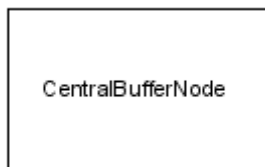


Figura 27. CentralBufferNode

17. Clause

Descripción: *Clause* es un elemento que representa a una única rama de un constructor condicional, incluyendo una prueba y un cuerpo de sección. El cuerpo de la sección se ejecuta sólo si (pero no es necesario) la sección de prueba se evalúa a cierto.

Notación: No aplica. No existe ningún símbolo que represente esta clase.

18. ConditionalNode

Descripción: *ConditionalNode* es un nodo condicional de *activities* estructuradas, que representan una elección exclusiva entre un cierto número de alternativas. *ConditionalNode* consta de una o más cláusulas (*clauses*). Cada cláusula consiste en una sección de prueba y un cuerpo de sección. Cuando el nodo comienza la ejecución condicional, se ejecutan las secciones de prueba de las cláusulas. Si una o más secciones de prueba toman un valor verdadero, uno de los cuerpos de sección será ejecutado. La elección no es determinista, a menos que se especifique la secuencia de prueba de cláusulas.

Notación: No aplica. No existe ningún símbolo que represente esta clase.

19. ControlFlow

Descripción: *ControlFlow* es un arco que se inicia en un nodo de *activity*, después de que esta *activity* haya terminado.

Notación: Se representa mediante una línea con una flecha que conecta dos *actions* tal y como se muestra figura 28.

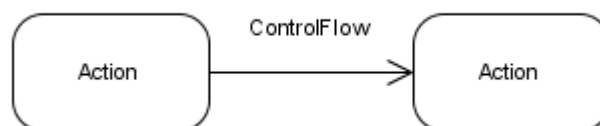


Figura 28. ControlFlow

20. ControlNode

Descripción: *ControlNode* es un nodo de control una *activity* abstracta que coordina los flujos en una *activity*. Además coordina el flujo entre otros nodos. Este nodo controla y engloba el *InitialNode*, *FinalNode* y sus hijos, *ForkNode*, *JoinNode*, *DecisionNode*, y *MergeNode*.

Notación: Los nodos de control o *ControlNode* se representan como se muestra en la figura 29, donde *ActivityFinalNode* y *FlowFinalNode* son *FinalNode*.

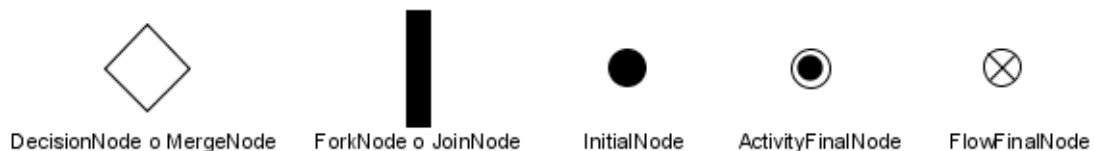


Figura 29. ControlNode

21. DataStoreNode

Descripción: *DataStoreNode* es un nodo de memoria central para la información no transitoria.

Un almacén de datos mantiene o almacena todos los tokens que entran en él, copiándolos cuando es necesario.

Notación: El símbolo de un *DataStoreNode* es un caso especial de la notación de *ObjectNode*, usando la etiqueta <<DataStore>>, tal y como se muestra en la figura 30.



Figura 30. DataStoreNode

22. DecisionNode

Descripción: *DecisionNode* es un tipo de nodo de control que elige o decide entre los flujos de salida.

Notación: Se representa mediante un rombo. El comportamiento de entrada se especifica por la palabra clave <<DecisionInput>>, colocada en una nota en el símbolo. Un flujo de entrada de la decisión se especifica por la palabra clave <<DecisionInputFlow>>, tal y como se muestra en la figura 31.

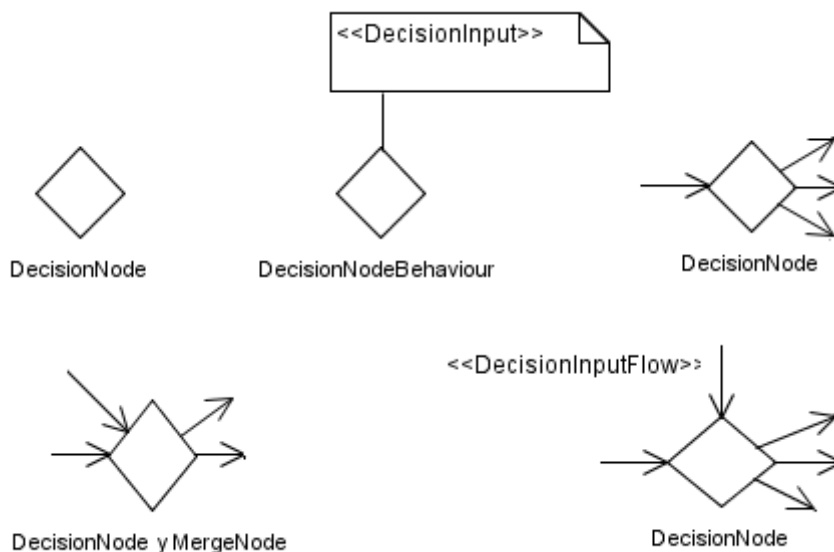


Figura 31. DecisionNode

23. ExceptionHandler

Descripción: *ExceptionHandler* es un manipulador o controlador de excepciones, es un elemento que establece un órgano encargado de ejecutarse en caso de que la excepción específica se produzca durante la ejecución de un nodo protegido.

Notación: Un *ExceptionHandler* para un nodo protegido se muestra con el símbolo de un rayo en la frontera del nodo protegido es un pequeño rectángulo al final del enlace del *ExceptionHandler*, tal y como se muestra en la figura 32. El rectángulo o plaza es el nodo de entrada de excepción, y debe ser propiedad del *ExceptionHandler*. Tanto el nodo como el *ExceptionHandler* tienen que estar al mismo nivel de anidamiento.

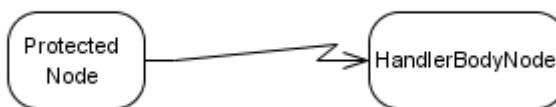


Figura 32. ExceptionHandler

24. ExecutableNode

Descripción: *ExecutableNode* es una clase abstracta para los nodos de *activity* que pueden ser ejecutados. Se utiliza como un punto de conexión de los manejadores de excepciones.

Notación: No aplica. No existe ningún símbolo asociado a la clase.

25. ExpansionKind

Descripción: *ExpansionKind* es un tipo de enumeración que se utiliza para especificar cómo interactúan las múltiples ejecuciones de una región de expansión. Los

literales que puede tomar son; *Parallel*, la ejecución es independiente, es decir se puede producir de forma paralela. Se puede ejecutar de forma concurrente. *Iterative*, la ejecución es dependiente y debe ejecutarse una vez cada momento, de forma iterativa, en el orden de la colección de elementos. *Stream*, es un flujo de una colección de elementos en una ejecución simple, en el orden de la colección de elementos.

Notación: No aplica. No existe ningún símbolo asociado a la clase.

26. ExpansionNode

Descripción: *ExpansionNode* es un objeto nodo utilizado para indicar el flujo a través del límite de una *ExpansionRegion*. Un flujo dentro de una región contiene una colección que se divide en sus elementos individuales dentro de ella, y que se ejecuta una vez por elemento.

Notación: Ver *ExpansionRegion* en figura 33.

27. ExpansionRegion

Descripción: *ExpansionRegion* es una zona o región de expansión de *activities* estructuradas, que se ejecuta varias veces que corresponden a los elementos de una colección de entrada. Una región de expansión es una región que se modela como un *ExpansionNode*. Cada entrada es una colección de valores. Si hay múltiples entradas, cada una de estas debe esperar el mismo tipo de colección, sin embargo los tipos de los elementos en colecciones distintas pueden variar. La *ExpansionRegion* es ejecutada una vez por cada elemento de entrada de la colección.

El número salidas de la colección puede ser distinto al número de entradas de la colección. Cada ejecución de una región se inserta en una colección de salida en la misma posición que los elementos de entrada. Si la ejecución de la región no genera ninguna salida, entonces no se añade ninguna salida a la colección. Cuando esto sucede la colección de salida no tiene el mismo número de elementos que la colección de entrada, con lo que la región de salida actúa como un filtro.

Como ya se ha dicho, las entradas y salidas de una región de expansión se modelan como un *ExpansionNode*. Para las entradas de la región los valores aparecen como elementos de la colección. Los arcos de objetos de flujo conectan *pins* de salida de la región con colecciones de entrada y salida de los *ExpansionNode*. Los arcos de objetos de flujo también conectan *pins* de entrada de la región con entradas y salidas de elementos individuales. Para la entrada de la región, estos nodos son visibles como valores individuales. Si un nodo de expansión tiene un nombre, es el nombre del elemento individual dentro de una región.

Notación: Se muestra como un cuadro con línea discontinua y con un conjunto de cuadrados en el extremo izquierdo y la palabra clave: *parallel*, *iterative* o *stream*.

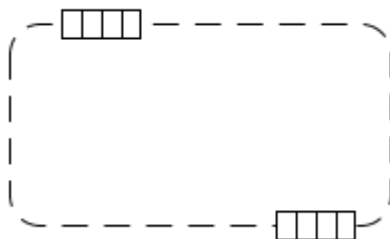


Figura 33. ExpansionRegion

28. FinalNode

Descripción: *FinalNode* es un nodo abstracto de control en el que el flujo se detiene en una *activity*.

Notación: Existen dos tipos de nodos final: *ActivityFinalNode* y *FlowFinalNode* final, tal y como se muestra en la figura 34.

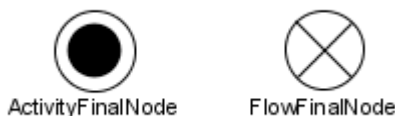


Figura 34. FinalNode

29. FlowFinalNode

Descripción: *FlowFinalNode* es un nodo final que finaliza un flujo.

Notación: La notación para un *FlowFinalNode* se muestra en la figura 35.



Figura 35. Notación para FlowFinalNode

30. ForkNode

Descripción: *ForkNode* es un nodo que representa una bifurcación que es un nodo de control que divide el flujo en múltiples flujos concurrentes.

Notación: La notación para este tipo de nodos es simplemente un segmento de línea, con las distintas variantes de bifurcación que existen, tal y como se muestra en la figura 36. En el uso, sin embargo, el *ForkNode* debe tener una única arco de entrada, y dos o más arcos de salida. La funcionalidad del *JoinNode* y el *ForkNode* puede combinarse usando el mismo símbolo de nodo.

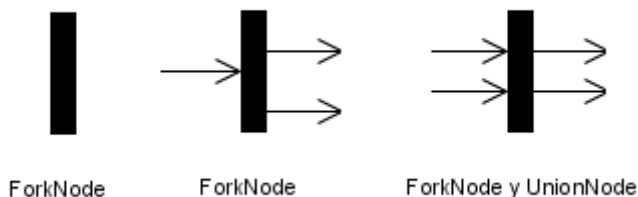


Figura 36. ForkNode

31. InitialNode

Descripción: *InitialNode* es un nodo en el que el control de flujo se inicia cuando se invoca la *activity*.

Notación: Este tipo de nodo se representa con un círculo sólido, tal y como se muestra en la figura 37.



Figura 37. InitialNode

32. InputPin

Descripción: *InputPin* son los nodos que reciben los valores de otras *actions* a través de un flujo de objetos (*ObjectFlow*)

Notación: No aplica. No existe un símbolo que represente esta clase.

33. InterruptibleActivityRegion

Descripción: *InterruptibleActivityRegion* es un grupo de *activities* que soporta la terminación de elementos que fluyen en una parte de una *activity*.

Notación: Esta región se simboliza por una línea discontinua, tal y como se muestra en la figura 38. Una ventaja es la notación con la interrupción de un rayo con una flecha en la punta que da a la *activity*.

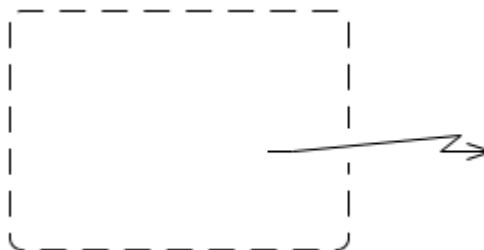


Figura 38. InterruptibleActivityRegion

34. JoinNode

Descripción: *JoinNode* es un nodo de control que sincroniza múltiples flujos. Estos nodos pueden tener múltiples arcos de entrada y un arco de salida.

Notación: Se representa con un segmento de línea, tal y como se muestra en la figura 39. *JoinNode* debe tener uno o más bordes de *activity* entrando en él, y sólo uno de los bordes saliendo de él.

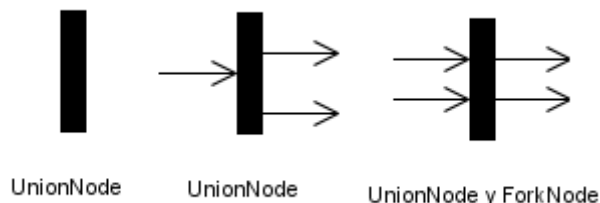


Figura 39. JoinNode

35. LoopNode

Descripción: *LoopNode* es un nodo de *activities* estructuradas, que representan un lazo o bucle con la colección, prueba y cuerpo de sección.

Notación: No aplica. No existe un símbolo que represente esta clase.

36. MergeNode

Descripción: *MergeNode* es un nodo de control que reúne a múltiples flujos alternativos. No se utiliza para sincronizar los flujos concurrentes que aceptan una de las distintas corrientes alternativas.

Notación: La notación para un *MergeNode* es un símbolo en forma de rombo, tal y como se muestra en la figura 40. Este nodo debe tener dos o más bordes de entrada a él y un único flujo saliendo de él. La funcionalidad de este nodo es el mismo que para el *DecisionNode*.

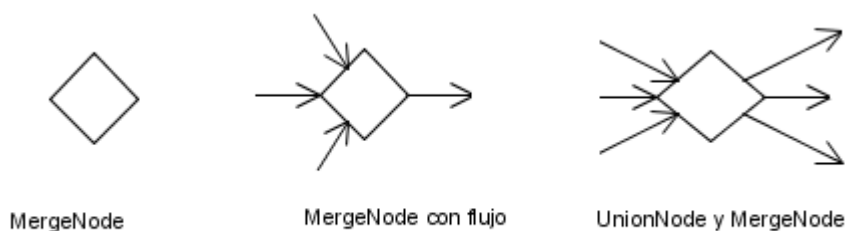


Figura 40. MergerNode

37. ObjectFlow

Descripción: *ObjectFlow* es un arco de *activity* que puede tener objetos o datos que pasan a lo largo de ella.

Notación: Se representa mediante una línea con una flecha, tal y como se muestra en la figura 41.

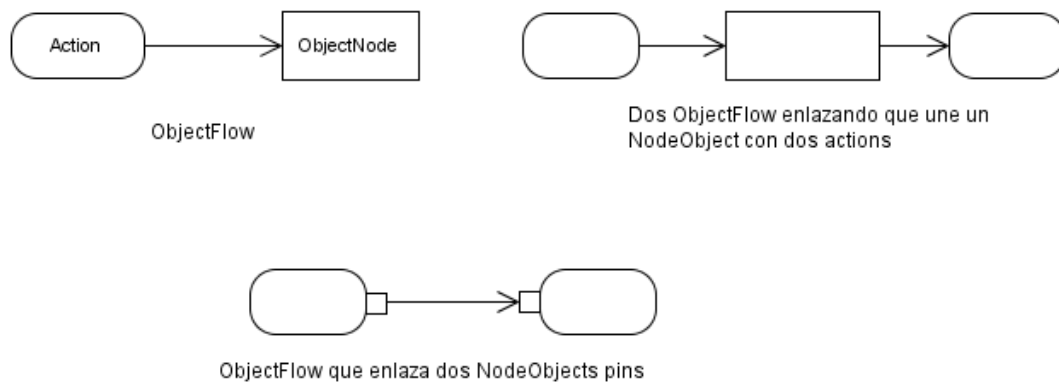


Figura 41. ObjectFlow

38. ObjectNode

Descripción: *ObjectNode* es una nodo abstracto de *activity* que es parte de la definición del flujo objeto en una *activity*.

Notación: Se representan mediante rectángulos, tal y como se muestra en la figura 42. Con una etiqueta con el nombre dentro del símbolo, que indica el tipo de nodo objeto con el formato "name:type".

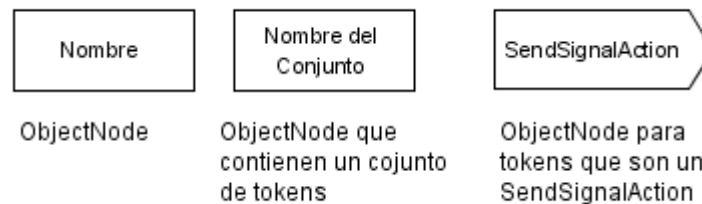


Figura 42. ObjectNode

39. ObjectNodeOrderingKind

Descripción: *ObjectNodeOrderingKind* es como una enumeración de una colección que indica el orden dentro del nodo. Los tipos de orden son: unordered (no ordenado), ordenado (ordered), ultimo en entrar primero en salir (LIFO), primero en entrar primero en salir (FIFO).

Notación: No aplica. No existe un símbolo que represente esta clase.

40. OutputPin

Descripción: *OutputPin* son *pins* de salida que representan nodos objeto que transmiten valores a otras *actions* a través de flujos de objetos.

Notación: No aplica. No existe un símbolo que represente esta clase.

41. Parameter

Descripción: *Parameter* es un tipo de parámetro que se especializan cuando se usan con *activities* completas.

Notación: La notación en los diagramas de clases para las excepciones y los parámetros de transición en las operaciones tiene la palabra clave <<exception>> o <<stream>> en la cadena que identifica la propiedad. Ver la notación de *ActivityParameterNode* en figura 20.

42. ParameterEffectKind

Descripción: *ParameterEffectKind* es un tipo de datos que es una enumeración que indica el efecto de un comportamiento en los valores que se transmite dentro o fuera de sus parámetros. Los tipos de efectos son: crear, leer, actualizar, eliminar.

Notación: No aplica. No existe un símbolo que represente esta clase.

43. ParameterSet

Descripción: *ParameterSet* es un tipo de conjunto de parámetros establece una alternativa de las entradas o salidas que un comportamiento puede utilizar.

Notación: Un objeto con múltiples flujos de entrada o salida de una invocación se tratan con una condición "y". Sin embargo, a veces un grupo de los flujos se permite la exclusión de otros. Se modela como un conjunto de parámetros y simboliza con rectángulos en torno a uno o a más líneas tal y como se muestra en la figura 43.

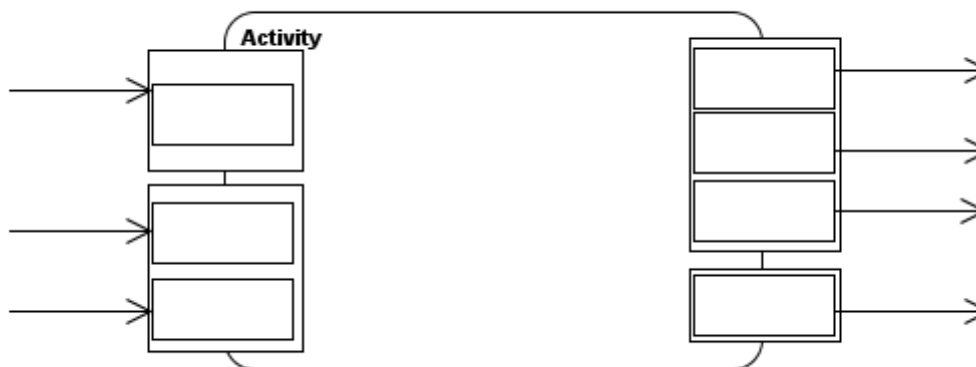


Figura 43. ParameterSet

44. Pin

Descripción: Un *Pin* es un nodo objeto para las entradas y salidas de una *action*.

Notación: Un *pin* se puede representar como pequeños rectángulos que se adjuntan a los rectángulos de *action*, figura 45.

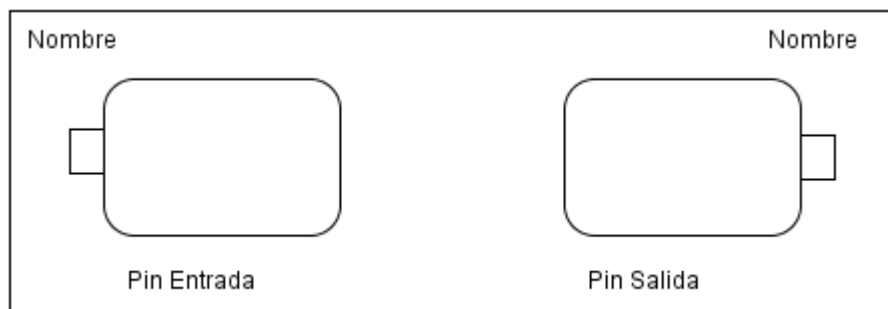


Figura 44. Pin

45. SendObjectAction

Descripción: *SendObjectAction* es un tipo de *action* que transmite de un objeto origen a objeto destino, donde se podrá invocar el comportamiento como el disparo de las transiciones de una máquina de estados o la ejecución de una *activity*. El valor del objeto está disponible para la ejecución del comportamiento.

Notación: No aplica. No existe símbolo asociado a esta clase.

46. SendSignalAction

Descripción: *SendSignalAction* es un tipo de *action* que crea una instancia de señal, y la transmite al objeto destino, donde puede causar el disparo de una máquina de estados o la ejecución de una *activity*. Los valores de los argumentos están disponibles para la ejecución del comportamiento asociado.

Notación: En la figura 45 se muestra la acción de enviar señal.

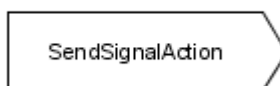


Figura 45. SendSignalAction

47. SequenceNode

Descripción: *SequenceNode* es un tipo de *activity* estructurada, que ejecuta sus *actions* en orden.

Notación: No aplica. No existe un símbolo que represente esta clase.

48. StructuredActivityNode

Descripción: *StructuredActivityNode* es un nodo de *activity* ejecutable que puede tener una expansión en los nodos hijo. Los nodos hijos deben pertenecer a un solo *StructuredActivityNode*, aunque pueden estar anidados.

Notación: Este tipo de nodos se representa con un rectángulo con línea discontinua, y las esquinas redondeadas que encierran los nodos y arcos, con la palabra clave <<structured>> en la parte superior, tal y como se muestra en la figura 46.



Figura 46. StructuredActivityNode

49. UnmarshallAction

Descripción: *UnmarshallAction* es un tipo de *action* que divide un objeto de un tipo conocido en salidas, cada una de ellas es igual a un valor de un rasgo estructural del objeto.

Notación: No aplica. No existe un símbolo que represente esta clase.

50. ValuePin

Descripción: *ValuePin* es un tipo de valor de un *pin* de entrada que ofrece un valor a una *action* que viene de un arco de llamada entrante.

Notación: Un *ValuePin* se representa con como un *InputPin* con la especificación de valor escrito junto a él.

51. ValueSpecificationAction

Descripción: *ValueSpecificationAction* es un tipo de *action* que evalúa un valor específico.

Notación: Este tipo de *action* se etiqueta con el valor de la especificación, tal y como se muestra en la figura 47.

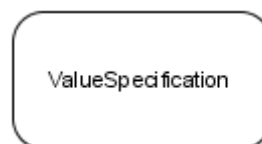


Figura 47. ValueSpecification

52. Variable

Descripción: *Variable* es un elemento para intercambiar datos entre las *actions* de forma indirecta. El resultado de una *action* puede ser escrito en una *variable* y se puede leer para una entrada de una *action* posterior. Una *variable* de memoria local comparte los valores con las *actions* que están dentro del *ActivityGroup* y no son accesibles desde fuera de dicho grupo.

Notación: No aplica. No existe una notación específica para esta clase.

1.4.2.9 Herramientas para el modelado con UML

Debido a la gran utilidad de UML, se han creado gran cantidad de herramientas para modelar los diagramas de UML. Se van a citar las más utilizadas y se va a exponer un ejemplo de la herramienta utilizada para realizar este estudio.

- *UModel de Altova*: Es una herramienta de modelado basada en UML que permite también generación de código Java. Incluye ingeniería inversa con capacidad para leer código fuente en Java y generar modelos UML. Esta herramienta es muy utilizada, por ejemplo por universidades ya que permite crear XMI Schema e integrar UML con otros lenguajes. No se ha podido obtener una versión de esta herramienta debido a que era preciso pagar una licencia.
- *EasyCASE for UML (Rational Rose)*: Es una herramienta de producción y comercialización establecidas por Rational Software Corporation. Rose es un instrumento operativo conjunto que utiliza UML, como medio para facilitar la captura de dominio de la semántica, la arquitectura y el diseño. Al igual que el anterior no era gratuito, con lo que no se ha podido utilizar.
- *Edraw UML Diagram 4.3*: Esta herramienta está diseñada como una gran aplicación útil para los ingenieros de software y diseñadores. Fácil de elaborar todos los diagramas de UML y OLE (del inglés *Object Linking and Embedding*).
- *Visual Paradigm for UML 7.1*: Es Una herramienta CASE que cuenta con una versión gratuita denominada Community Edition. Ésta ha sido la herramienta utilizada para este estudio, ya que era gratuita, además de manejable e intuitiva. Esta herramienta añade conceptos para modelar que no se definen en el estándar de UML 2.2 [44]. A continuación se expone un ejemplo con esta herramienta.

Un ejemplo con la herramienta Visual Paradigm for UML 7.1, es un proceso de compra. Este ejemplo, podría representar cualquier proceso de negocio de una tienda a la que se puede realizar pedidos, tanto por internet, como por teléfono, o en la misma tienda. Como se muestra en la figura 48, se ha creado tres particiones que representan los distintas particiones que se pueden distinguir en el proceso. El primero es el cliente que va a realizar el pedido (acción pedido), esta acción desencadena la acción coger en la partición de ventas, que comprobará y cogerá del almacén el pedido. La acción reponer de la partición almacén, repondrá las existencias del objeto u objetos que se han pedido. Paralelamente a la acción coger, el cliente ha debido llegar a la acción pagar, para que ventas entregue dicho pedido. Una vez ejecutada la entrega, el cliente podrá recoger su pedido.

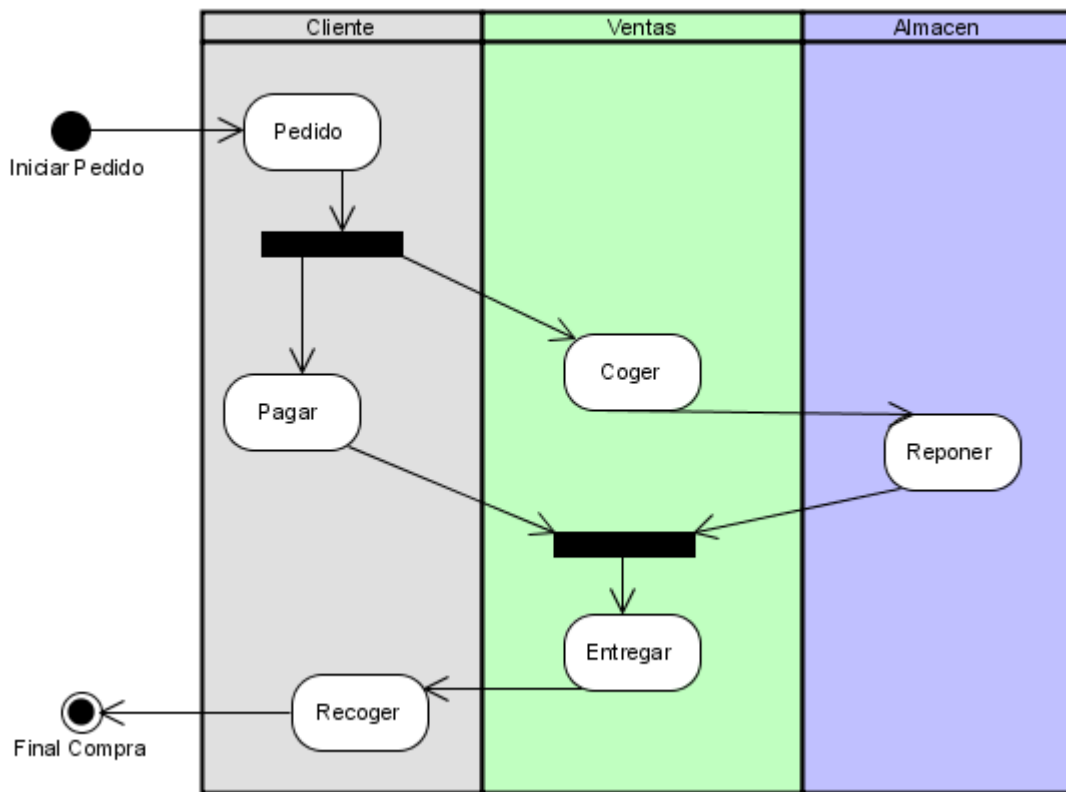


Figura 48. Ejemplo del proceso de pedido y compra

1.4.3 Redes De Petri

Una red de Petri se puede utilizar también para modelar procesos de negocio, y de esta premisa surgió YAWL, que se explicará más adelante como lenguaje de modelado de procesos de negocio. En este apartado se va a hacer una breve descripción de las *redes de Petri*, intentando dar significado a su teoría para poder entender y llevar a cabo el objetivo de este trabajo.

1.4.3.1 Introducción

Las redes de Petri se deben a la tesis doctoral “Kommunikation mit Automaten2” del alemán Carl Adam Petri presentada en la facultad de matemáticas y física de la Universidad Técnica de Darmstadt en 1962.

Lo que Carl Adam creó fue una forma de mantener flujos de trabajo, tal que se controlara, monitorizara, optimizara y soportara los procesos de negocio. La principal característica que destaca en este mantenimiento es la representación lógica de los procesos de negocio de forma que pueda ser soportada por un ordenador.

Una Red de Petri (PN, del inglés *Petri Net*) es una herramienta gráfica y matemática que ayuda a describir relaciones entre condiciones y eventos presentes en los sistemas del mundo real. Las características de un sistema que son modeladas

utilizando PN son algunas de las siguientes: sincronización de procesos, eventos asíncronos, operaciones secuenciales, operaciones concurrentes, conflictos con recursos compartidos, procesos distribuidos, procesos paralelos, procesos no deterministas y/o estocásticos.

Un sistema modelado a través de PN puede ser analizado de manera formal, obteniendo información de su comportamiento dinámico. Además es posible verificar propiedades que debe mantener el sistema. Por ejemplo, se puede verificar que no se encuentren estados inalcanzables en el sistema o verificar que no exista bloqueo de candados (deadlocks) dentro del modelo.

Dos conceptos importantes en el modelado de sistemas dinámicos son los eventos y las condiciones. Los eventos son acciones que se presentan en el sistema y lo llevan a un estado. Es posible describir un estado como un conjunto de condiciones; para que cierto evento ocurra es necesario que ciertas condiciones se cumplan; a éstas se les llama precondiciones del evento, así mismo, la ocurrencia del evento puede llevar a otras condiciones; éstas se conocen como post condiciones.

Al modelar con PN es necesario identificar las condiciones (pre y post condiciones), así como los eventos que pueden suceder en él. De esta manera se traslada el sistema de la vida real al modelo en PN. La tarea de modelado es una actividad subjetiva. Es decir, el modelado de un sistema puede realizarse de diferentes maneras. Así cuando diferentes personas realizan el modelo de un mismo sistema, los modelos resultantes pueden diferir considerablemente; sin embargo ambos estarán modelando el comportamiento del sistema. Al tener las condiciones necesarias para que ciertos eventos del sistema sucedan, es posible modelar de forma modular, y de esta manera relacionar los modelos con otras condiciones; para esto es necesario conocer la estructura de la PN.

En la tabla 8 se muestra [10], cómo, una condición puede ser modelada a través de un lugar y un evento a través de una transición. De esta manera una precondición puede ser modelada empleando un lugar de entrada y una post condición con un lugar de salida. En la tabla se muestran algunas maneras de interpretar los lugares y las transiciones.

Lugares de Entrada	Transiciones	Lugares de Salida
Precondiciones	Evento	Post condiciones
Datos de Entrada	Procesamiento	Datos de Salida

Señales de Entrada	Procesamiento de señales	Señales de Salida
Requerimiento de Recurso	Tarea	Liberación de Recurso
Condiciones	Cláusula Lógica	Conclusiones
Buffers	Procesador	Buffers

Tabla 8. Elementos de una red de Petri

Las redes de Petri poseen una parte matemática y una gráfica. Por lo tanto, pueden ser estudiadas formalmente a través de su estructura matemática y en su forma gráfica a través de su representación visual en forma de grafo dirigido.

1.4.3.2 Elementos

Una red de Petri es un grafo bipartito dirigido y se compone de los siguientes elementos:

- Un conjunto de lugares (nodos).
- Una función de entrada.
- Una función de salida.
- Marcado Inicial.

Las funciones de entrada y de salida relacionan a los lugares con las transiciones y viceversa. La función de entrada es un mapeo de una transición t_j a un conjunto de lugares conocidos como los nodos de entrada de una transición. La estructura de una PN es definida por los nodos, las transiciones, la función de entrada y la función de salida. A continuación se presenta la definición formal de una PN.

1.4.3.3 Definición formal

La estructura de una red de Petri se define como la siguiente quintupla.

$PN = (P, T, F, W, M_0)$ donde:

$P = \{p_1, p_2, \dots, p_m\}$ es un conjunto finito de lugares (nodos), donde $m \geq 0$.

$T = \{t_1, t_2, \dots, t_n\}$ es un conjunto finito de transiciones, donde $n \geq 0$.

$F \subseteq (P \times T) \cup (T \times P)$ es un conjunto de arcos.

$W : F \rightarrow \{1, 2, 3, \dots\}$ es una función de peso.

$M_0 : P \rightarrow \{0, 1, 2, 3, \dots\}$ es la marca inicial.

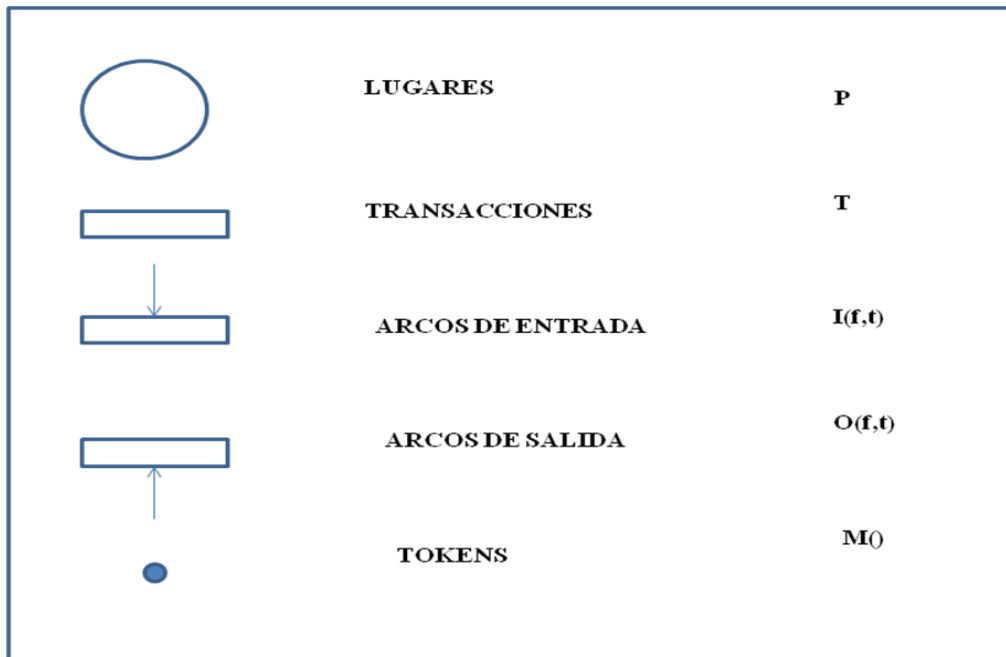


Figura 49. Elementos de una Red de Petri.

Para definir el concepto de marca inicial es necesario referirse a los tokens. Un token es un concepto primitivo que forma parte de las PN, el cual es asignado a cada uno de los lugares, a esta asignación se le conoce como marca, un lugar puede contener n tokens. Los tokens deciden la ejecución en la red de Petri. Por lo tanto la cantidad y la posición de los tokens cambian durante la ejecución. El paso de los tokens a través de las transiciones y los lugares de la PN, es utilizado comúnmente para interpretar la ejecución del sistema modelado.

Como herramienta matemática es posible obtener ecuaciones de estado, ecuaciones algebraicas y otros modelos matemáticos que describen el comportamiento de los sistemas. Mientras que como herramienta gráfica ésta permite obtener un concepto visual del sistema, a continuación se describe la representación gráfica.

1.4.3.4 Representación Gráfica

La representación gráfica de una PN es importante ya que si se observa el modelo del sistema en forma gráfica y la manera en que cambia de un estado a otro, es posible mantener la atención y obtener una perspectiva más clara del análisis del problema. En la figura 49 se muestra la representación gráfica de los elementos que forman una red de Petri.

Los lugares son representados con círculos, las transiciones se representan mediante una barra o un rectángulo, los arcos dirigidos que conectan a los lugares con las transiciones y viceversa son representados con flechas. Los tokens son representados con pequeños círculos de color negro que van dentro de los lugares.

Un arco dirigido desde un lugar p_j (transición t_i) a una transición t_i (lugar p_j) define a p_j como un lugar de entrada (salida) de t_i , el cual describe como $w(p_j, t_i) = w(t_i, p_j) = 1$. Si $w(p_j, t_i) = k$ (ó $w(t_i, p_j) = k$), entonces existen k arcos dirigidos (paralelos) que conectan el lugar p_j con la transición t_i (o que conectan la transición t_i con el lugar p_j). Generalmente en un esquema gráfico, los arcos paralelos que conectan a un lugar (transición) con una transición (lugar) se representan por un solo arco dirigido, etiquetado con el número de arcos que representa a su peso k . Un lugar que contiene un punto en su interior representa a un lugar que contiene un token. La capacidad de almacenamiento de tokens en cada lugar es infinita, y para denotar el número de tokens o marcado de un lugar se utiliza $M(p)$.

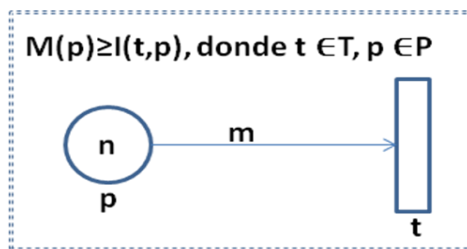
1.4.3.5 Reglas de disparo

La ejecución de una PN es controlada por el número y distribución de los tokens. La ejecución de la red de Petri se activa disparando sus transiciones. Cuando una transición es disparada se remueven tokens de los lugares de entrada y se crean tokens en los lugares de salida. El número de tokens que son removidos es igual al peso del arco que conecta al lugar de entrada con la transición que fue disparada. De igual manera el número de tokens creados está dado por el peso del arco que conecta la transición con el lugar de salida.

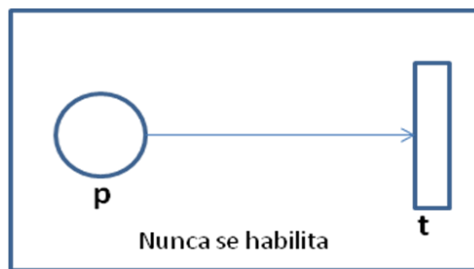
Para que una transición pueda dispararse, requiere estar habilitada, Dicha habilitación depende de una condición conocida como *regla de habilitación*. Esta regla se describe a continuación:

Una transición t se dice que será habilitada si cada lugar de entrada p de t contiene al menos un número de tokens igual al peso k del arco dirigido que conecta a p con t , es decir, que $M(p) \geq w(p, t)$ para algún p en P . En la figura 50 se muestra la regla de disparo y algunos ejemplos.

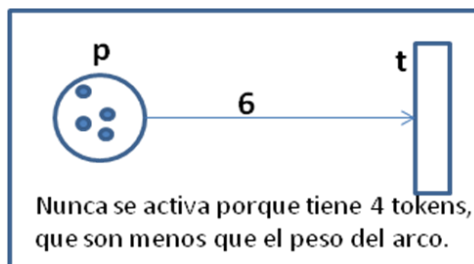
La figura 50 a) muestra la condición de habilitación. Cuando se omite el peso del arco por convención el peso es 1, no sucede así con el número de tokens. Es decir, si no existen tokens el número de tokens es 0. Es así como en el ejemplo mostrado en la figura 50 b) la red no se habilitará nunca ya que el peso del arco es 1 y no es ni mayor ni igual a 0, que es el número de tokens en el lugar. Un caso similar se da en el ejemplo de la figura 50 c). Sin embargo, la figura 50 d) muestra un ejemplo en el que la transición sí se activará ya que el número de tokens en el lugar es mayor al peso de su transición.



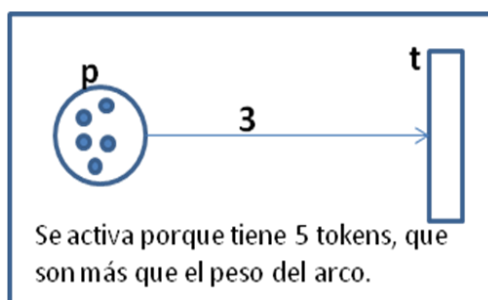
a)



b)



c)



d)

Figura 50. Ejemplo de una Red de Petri.

Una transición activada puede o no dispararse, (dependiendo de la interpretación adicional que tenga la transición). Esencialmente el disparo de las transiciones y el cambio de tokens entre los lugares de la red, es lo que permite ver el comportamiento dinámico de un sistema. A lo anterior se le conoce como la ejecución del juego de tokens de la PN.

En la figura 51 se muestra la ejecución de una PN sencilla. En ella podemos verificar cómo el peso de los arcos decide el número de tokens que son movidos del lugar de entrada y el número de tokens creados en el lugar de salida. También podemos observar que llega un momento en que el lugar de salida queda sin tokens, esto es después del tercer disparo. Esto provoca la deshabilitación de la transición y la ausencia de posteriores disparos en el sistema, hecho que se podría interpretar como la finalización del sistema, o el agotamiento de sus recursos, esto es dependiendo de la interpretación con que se modele el sistema.

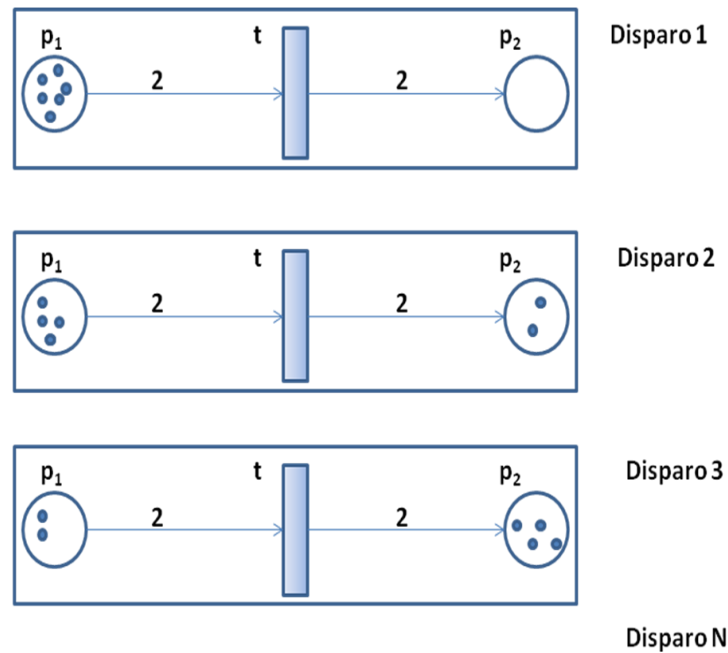


Figura 51. Ejemplo de disparo en una red de Petri.

1.4.3.6 Propiedades y Métodos de análisis

Una de las mayores ventajas de utilizar técnicas de modelado que poseen semántica formal (tal es el caso de las redes de Petri) es que el modelo obtenido, puede ser analizado a través de técnicas matemáticas. Las PN poseen propiedades que permiten analizar la validez del modelo desde distintas perspectivas, principalmente en aspectos de concurrencia y paralelismo.

Dos tipos de propiedades pueden ser estudiadas en un modelo de red de Petri [10]:

- Las que dependen del marcado inicial (comúnmente referidas como propiedades de comportamiento).
- Las que son independientes del marcado inicial (propiedades estructurales).

A continuación se describe brevemente cada una de las propiedades de comportamiento ya que estas propiedades son de más interés en la tarea de modelar los procesos de tipo flujos de trabajo que fueron presentados en capítulos anteriores. Esto es debido al interés de conocer el comportamiento y las propiedades estructurales en un proceso de negocios. En [10] es posible profundizar en el tema de las propiedades de una PN.

1.4.3.7 Propiedades de comportamiento

Alcanzabilidad: Una secuencia de disparos da lugar a una secuencia de marcados. Un marcado M_n se dice que es alcanzable desde un marcado M_0 si existe una secuencia de disparos que transforma M_0 en una red (N, M_n) ; se denota por $R(N, M_0)$.

Un disparo o secuencia de ocurrencias es denotado por $\sigma = M_0 t_1 M_1 t_2 M_2 \dots t_n M_n$ o simplemente $\sigma = t_1 t_2 \dots t_n$. En este caso, M_n es alcanzable desde M_0 a través de σ por lo que se escribe $M_0 [\sigma > M_n]$. El conjunto de todos los posibles marcados que hacen alcanzable a M_0 en una red (N, M_0) es denotado por $L(N, M_0)$ o simplemente por $L(M_0)$.

Acotamiento: Una red (N, M_0) se dice que está k -acotada o simplemente acotada si el número de tokens en cada lugar no excede un número finito k para ningún marcado alcanzable desde M_0 . Una red de Petri se dice que es segura si es 1-acotada. El término “segura” se refiere al conocimiento del número de tokens k que habrá siempre en los lugares de la red. Lo anterior, cierto tipo de sistemas lo requieren con el fin de que los elementos representados por los tokens no tiendan a infinito, con lo cual aseguran estabilidad en el sistema.

Vivacidad: Este concepto está relacionado con la ausencia total de bloqueo de candados o deadlocks como son conocidos en el área de sistemas operativos. El concepto: “viva”, está estrechamente relacionado con la situación de candado mortal (deadlock), el cual ha sido situado en el ámbito de los sistemas operativos. Una red de Petri que modela un sistema y se encuentra libre de candados mortales es viva. Esto implica que para alguna marca alcanzable M , es posible disparar al menos una transición en la red a través de alguna secuencia de disparo. Sin embargo, este requisito debe ser bastante estricto para representar algún sistema real o escenarios donde se presente un comportamiento libre de bloqueo de candados.

Persistencia: Una red de Petri se dice que es persistente si en el caso de tener dos transiciones cualesquiera habilitadas, el disparo de una no deshabilita a la otra transición. La noción de persistencia es utilizada en el contexto de programas paralelos [10].

1.4.3.8 Extensión de las redes de Petri básicas

Las extensiones o modificaciones a las redes de Petri básicas fueron desarrolladas de acuerdo a necesidades que han surgido en el área de investigación referida a modelado de sistemas. Existen algunas extensiones de redes de Petri que presentan ciertas variaciones contra la estructura original. Estas variaciones fueron desarrolladas con la finalidad de cubrir características de sistemas que no manejan únicamente eventos y condiciones debido a que existen sistemas que necesitan incluir información en forma de datos o variables. En ciertos casos es necesario manejar intervalos de tiempo u otras propiedades con el fin de representar las características del sistema modelado.

Algunas extensiones de redes de Petri son las siguientes:

- Red de Petri Coloreada.
- Red de Petri Difusa.
- Red de Petri Estocástica.

- Red de Petri con Tiempo.
- Red de Petri Temporizada..

1.4.3.9 Redes de Petri para modelar Procesos de Negocio

En este apartado se va a realizar una pequeña introducción a las redes de Petri. En apartados posteriores se realizará un estudio más exhaustivo para poder explicar el lenguaje de modelado YAWL. En base a diversos trabajos de investigación realizados desde inicios de los años 90's, que han utilizado diversas modificaciones de redes de Petri para el modelado de flujos de trabajo a continuación mostramos una lista de ventajas que han sido identificadas desde siempre debido a las propiedades inherentes del modelo de red de Petri y su relación con los flujos de trabajo. Así mismo, se muestran algunos argumentos encontrados en la literatura que sugieren que las redes de Petri no son del todo buenas para modelar procesos flujo de trabajo. Aún así se ha visto interesante y necesario describir detenidamente estas redes de Petri, en apartados posteriores se describirán más detalladamente.

1.4.3.10 YAWL

En [29] dicen: *YAWL (Yet Another Workflow Language) es un lenguaje de flujo de trabajo basado en los patrones de flujo de trabajo. Este lenguaje está soportado por un sistema de software que incluye un motor de ejecución y un editor gráfico.* El lenguaje y su sistema de soporte fueron desarrollados originalmente por investigadores de la *Universidad Eindhoven de Tecnología* y la *Universidad Queensland de Tecnología*. Consecuentemente, varias organizaciones tales como *Grupo InterContinental de Hoteles, first:telecom* y *ATOS Worldline* se unieron a esta iniciativa y el sistema YAWL está ahora disponible como software de código abierto bajo la licencia LGPL (*Licencia Pública General Reducida de GNU*).

Los objetivos para obtener YAWL 2.2 eran definir un lenguaje de flujo de trabajo que soportara todo (o la mayoría) de los patrones de flujo de trabajo y que tuvieran una semántica formal. Observando que las redes de Petri se acercaba bastante a dar soporte a la mayoría de los patrones de flujo, los desarrolladores de YAWL decidieron tomar las redes de Petri como un punto de partida y extender esta formalización con tres constructores principales que son los siguientes: *O-unión, grupos de cancelación, y actividades multi-instancia.*

Estos tres conceptos o constructores soportan cinco de los patrones de diseño que no fueron incluidos directamente en las redes de Petri: Fusión local de sincronización, Discriminador estructurado, Unión estática y parcial para múltiples instancias, múltiples instancias con conocimiento no a-priori y Cancelación de Casos. Para completar conceptos, YAWL añade algunos elementos sintácticos a las redes de Petri de forma que sea posible capturar intuitivamente otros patrones de diseño tales como Elección simple (O Exclusivo Divisor), Sincronización (O Exclusivo Unión), y Múlti- elección (O-divisor).

Durante el diseño del lenguaje, puso de manifiesto que algunas de las extensiones que fueron añadidas a las redes de Petri eran difíciles e incluso imposibles de volver a codificar con redes de Petri sencillas. Como resultado, la semántica formal original de YAWL está definida como un sistema de transiciones etiquetadas y no en términos de redes de Petri. El hecho de que YAWL esté basado en una semántica formal ha puesto en marcha la implementación de distintas técnicas para analizar procesos YAWL. En particular, el sistema YAWL incluye una herramienta de análisis estático llamada WofYAWL.

A continuación se va a describir en qué consiste YAWL, su nomenclatura y todos aquellos aspectos que puedan apoyar a una conclusión más concisa. YAWL se basa en redes de Petri, y lo que se ha hecho con YAWL es ampliarlo con funciones adicionales para facilitar el modelado de los patrones de flujos de trabajo más complejos.

Requisitos

Aunque en el apartado de patrones se ha hecho una descripción de los patrones para representar flujos de trabajo, a continuación se va a hacer un breve resumen de las seis categorías en las que se engloban todos los patrones según el documento [30], las cuales se refieren a los patrones que implementa YAWL.

Las seis categorías en las que se clasifican los principales patrones son las siguientes:

1. *Patrones básicos de flujo de control*: Estas son las construcciones mayoritarias en los lenguajes de flujos de trabajo para el modelado secuencial, paralelo y el enrutamiento condicional.
2. *Patrones de ramificación avanzada y sincronización*: estos patrones básicos trascienden de los básicos para permitir tipos más avanzados de divisiones y uniones del flujo.
3. *Patrones estructurales*: estructuras de bloque que identifiquen claramente puntos de entrada y de salida.
4. *Patrones que envuelven instancias múltiples*: en el contexto de un solo caso a veces todas las partes del proceso necesitan ser instanciadas varias veces.
5. *Patrones basados en estados*: típicamente los flujos de trabajo se centran sólo en actividades y eventos y no en estados. Esto limita la expresividad de un lenguaje de flujos de trabajo, ya que no permite disponer de patrones como el de Hito.
6. *Patrones de cancelación*: la ocurrencia de un evento puede dar lugar a la cancelación de actividades. En algunos escenarios en los que existen estos eventos puede causar incluso la retirada de todo el caso.

Sin embargo es importante señalar que hay diferencias notables entre los lenguajes y muchas de las construcciones avanzadas que no pueden realizarse por la base de flujo de control.

Las redes de Petri son la base para desarrollar el lenguaje de flujos de trabajo YAWL, para superar las limitaciones que tienen las redes de Petri a la hora de servir como lenguaje de modelado de flujos de trabajo. El punto de partida para ampliar las redes de Petri es realizar construcciones para hacer frente a *las instancias múltiples, la sincronización avanzada y las modalidades de cancelación*. En esta sección se define el lenguaje YAWL, mostrando la notación empleada para el diseño de flujos de trabajo.

Definición

Cómo ya se ha dicho antes, YAWL se inspira en las redes de Petri pero no es sólo un paquete macro incorporado en la parte superior de nivel más alto de las redes de Petri. YAWL es un lenguaje completamente nuevo con semántica independiente al nivel más alto de redes de Petri.

La especificación de un flujo de trabajo en YAWL es un conjunto de redes de flujo de trabajo ampliado que forman una jerarquía, es decir, es cómo una estructura de árbol. Las tareas son atómicas (únicas) o un conjunto o compuesto de estas. Cada compuesto de tareas se refiere a un único flujo de trabajo en un nivel inferior de la jerarquía. Las tareas atómicas (es decir que se tiene que ejecutar por completo y finalizar individualmente es decir tareas únicas) son las hojas de la estructura de árbol. Hay un *Eflujo de Trabajo-Red* (EWF, del inglés EWorkFlows-nets) que no es un compuesto de tareas, este es el nivel superior del árbol o raíz desde donde comenzaría un flujo de trabajo.

Cada EWF se compone de tareas (ya sean compuestas o atómicas) y las condiciones que pueden interpretarse como lugares. Cada flujo de trabajo cuenta con una única condición de entrada y una única condición de salida. En contraste con las redes de Petri, es posible conectar o tratar “transición-como-objetos” como tareas compuestas y atómicas directamente sin usar un “lugar-como-objeto” (es decir, las condiciones) en el medio. La semántica de esta construcción puede ser interpretada como una condición oculta, es decir, una condición implícita que se añadirá siempre para representar una condición directa o que siempre existirá, pero no es necesario representar.

En la figura 52 se muestra los símbolos que usa YAWL para la representación de procesos de negocio.

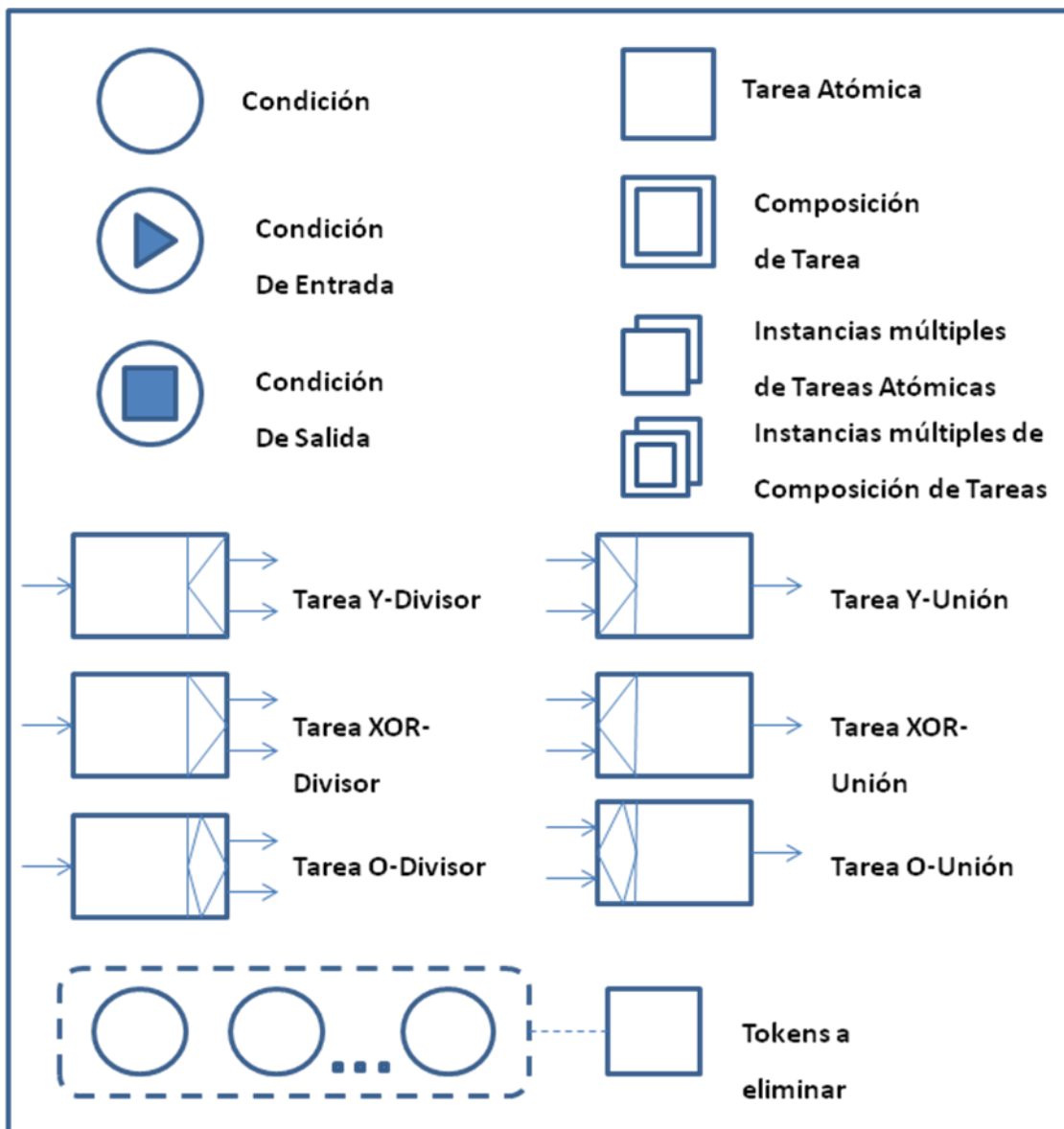


Figura 52. Notación para YAWL

Cada tarea (ya sea compuesta o atómica) puede tener múltiples instancias. Es posible especificar el número máximo (límite máximo) y el mínimo (límite mínimo) de instancias creadas después de iniciar la tarea. Por otra parte, es posible indicar que la tarea termina en el momento en el que se alcanza el umbral de número de instancias cuando todas las que están en ejecución terminan y la tarea se completa. Si no se especifica el umbral, la tarea se completa cuando todas las instancias finalizan. Para terminar, existe un cuarto parámetro que indica si el número de instancias se fija después de crear las primeras instancias. El valor de los parámetros es *estático* si después de la creación de instancias no se pueden añadir nuevas, el valor de los parámetros es *dinámico* si es posible añadir más instancias mientras que todavía hay instancias en ejecución. Destacar que para extender las redes de Petri mediante una construcción se toman estos cuatro parámetros (límite máximo, límite mínimo, umbral y estático/dinámico), directamente soporta patrones que envuelven múltiples instancias, y

adicionalmente, el patrón discriminador bajo el concepto de múltiples instancias de la misma tarea.

En [30] adoptan la notación que se describe en [34] y [35] para *Y/XOR-Divisor/Unión* de la figura 52. Además, introducen *O-Divisor* y *O-Unión* correspondientes respectivamente al patrón de Multi-elección y Fusión de sincronización estructurada.

Finalmente se introduce una notación para permitir eliminar los tokens de los lugares independientemente de los tokens que existan. Todas aquellos tokens que pueden ser eliminados se representan dentro de una zona. La tarea que lo habilita no depende de los tokens dentro de la zona de eliminación. Sin embargo, en el momento en el que la tarea se ejecuta todos los tokens de la zona se eliminan. Es evidente que esta extensión es útil para representar los patrones de cancelación. También se muestra en la figura 52.

1.4.3.11 Herramientas para el modelado con YAWL

Para la representación de procesos de negocio con la notación YAWL, no se han encontrado más que una herramienta que permita la representación de los procesos de negocio. Ésta ha sido adquirida en la propia página principal de YAWL [55]. Esta herramienta maneja datos complejos, transformaciones, la integración con recursos de la organización y la integración de servicios Web. Está construida en Java, y emplea XML Schema y XQuery.

Esta herramienta se llama YAWL4Study_2.0_Windows. A continuación se muestra un ejemplo de proceso de negocio modelado con esta herramienta.

El siguiente ejemplo que tiene en cuenta el rol de tiempo y los eventos.

Se define una tarea pagar, que es una tarea evento que desencadena automáticamente un evento (por ejemplo, la llegada de un mensaje). La tarea fin_tiempo es una tarea de tiempo, por ejemplo una tarea que sólo espera una cantidad de tiempo (segundos, minutos, horas...) para ser completada. Puesto que esto es una simple tarea y no parte del lenguaje YAWL, puede haber varias tareas predefinidas ofreciéndose como servicios. Se muestra la situación donde la tarea fin_tiempo cancela el proceso de pago cuando se completa el tiempo. Notar que igual si el pago comienza la primera bifurcación puede ser cancelada. También destacar que la condición previa a Envío_factura y pago es explícita. Ver la figura 53.

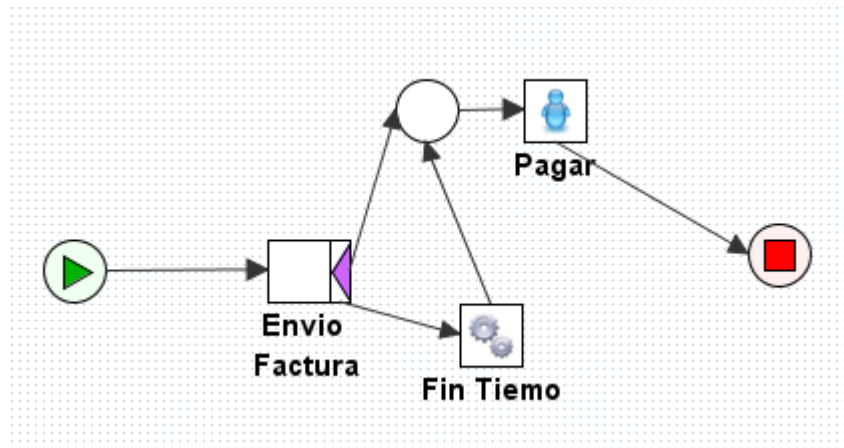


Figura 53. Ejemplo de proceso con YAWL

2. ANALISIS COMPARATIVO DE LOS LENGUAJES DE MODELADO DE PROCESOS DE NEGOCIO

Una vez realizado una descripción de la notación de BPMN, UML y YAWL, llega el momento de realizar el análisis y comparación de estos lenguajes, tomando como base de la comparativa los patrones de diseño (ver descripciones de cada patrón en el apartado 1.3.1).

Para ello se va a enumerar cada uno de los patrones de flujo de control, y se va a mostrar una tabla por cada uno de ellos, donde las columnas indican el lenguaje (BPMN, UML o YAWL), la versión sobre la que nos basamos, si dicho lenguaje soporta el patrón (sí o no) y la motivación que lleva al lenguaje a implementar o soportar el patrón.

1) Secuencia

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Lo implementa utilizando enlaces entre actividades con flujos de secuencia.
YAWL	1.0	Sí	Lo implementa uniendo los distintos lugares secuenciales con transiciones. Utilizando arcos de entrada para llegar entrar en un lugar y arcos de salida para salir de este.
UML	2.2	Sí	Lo implementa con la utilización de <i>ActivityEdges</i> entre nodos.

2) División paralela

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Para implementar este patrón utiliza la forma de Y-Divisor, que en este caso se representa con una pasarela paralela.
YAWL	2.2	Sí	Soportado mediante la tarea Y-Divisor.
UML	2.2	Sí	Soportado mediante la construcción de un <i>ForkNode</i> . También puede ser representado implícitamente por una <i>action</i> o <i>activity</i> que se une a varias <i>actions</i> o <i>activities</i> posteriores.

3) Sincronizador

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Soportado mediante la pasarela Y-Unión, que en este caso es otra pasarela paralela.
YAWL	2.2	Sí	Soportado mediante la tarea Y-Unión.
UML	2.2	Sí	Soportado mediante la construcción de un <i>JoinNode</i> . También puede ser representado implícitamente.

4) Elección Exclusiva

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Soportado por la pasarela O Exclusivo Dividido, que se representa con una pasarela exclusiva.
YAWL	2.2	Sí	Se soporta mediante el uso de una tarea O Exclusivo Dividida, con condiciones de salida.
UML	2.2	Sí	Soportado por la construcción de un <i>DecisionNode</i> donde las condiciones de salida son excluyentes.

5) Fusión Simple

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Soportado por la pasarela O Exclusivo Fusión, que se representa con una pasarela exclusiva.
YAWL	2.2	Sí	Lo soporta mediante la tarea de O Exclusivo Unión.
UML	2.2	Sí	Soportado por la construcción de un <i>MergeNode</i> .

6) Multi-Elección

Puede ser representado implícitamente o explícitamente. El modelado y los procesos de negocio tienden a favorecer el empleo explícito para representar el modelo:

UML 2.2 mediante la bifurcación de nodos con condiciones (de guardia) en las ramas o arcos de salida. BPMN proporciona tres representaciones alternativas incluyendo el empleo de división implícita con condiciones en los arcos, un O-Divisor o una entrada compleja.

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Soportado de tres formas distintas: <ul style="list-style-type: none"> • Una con una pasarela de división implícita (división implícita) con condiciones en los arcos. • Con un O-Divisor, una pasarela dividida y una pasarela compleja.
YAWL	2.2	Sí	Se implementa mediante el uso de combinaciones y el uso de una tarea O-Divisor.
UML	2.2	Sí	Soportado por la construcción de un <i>ForkNode</i> , con las condiciones en los extremos salientes, que como ya se ha dicho se representa añadiendo unas notas textuales indicando las condiciones

7) Fusión de Sincronización Estructurada

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Soportado a través de la pasarela O-Unión, representado mediante una pasarela que utiliza la combinación.
YAWL	2.2	Sí	Se implementa mediante el uso de combinaciones y el uso de una tarea O-Unión.
UML	2.2	Sí	A pesar de que en la página oficial de patrones [8] se indique que no lo soporta, sí es posible implementar este patrón combinando la clase <i>CallBehaviourAction</i> . <i>Activity</i> implementa este tipo de <i>action</i> .

8) Multi-Fusión

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Soportado a través de la pasarela O Exclusiva-Unión basada en datos o en eventos.
YAWL	2.2	Sí	Lo soporta mediante la tarea O Exclusiva Unión.
UML	2.2	Sí	Se implementa con un <i>MergeNode</i> .

9) Discriminador Estructurado

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	A pesar de que en [8], consideran que el soporte a este patrón es parcial debido a que la pasarela compleja de unión no especifica la condición del discriminador, en la versión 1.2, esta pasarela permite definir claramente la condición, utilizando una decisión basada en datos.
YAWL	2.2	Sí	Lo implementa gracias al uso de condiciones, junto con O-Divisor.
UML	2.2	Sí	Aun qué al igual que en BPMN, en un principio [8], se dice que no lo implementa, la actual versión de UML, sí permite definir claramente la condición del discriminador, utilizando un <i>DecisionNode</i> .

10) Discriminador de Bloqueo

Para este caso es importante tener en cuenta que en el caso en el que varios procesos concurrentes traten de iniciar al mismo tiempo el bloqueo discriminador, es necesarios realizar un seguimiento de dichos procesos y de la entrada de las ramas que han desencadenado el bloqueo, además de los hilos de ejecución que son consecuencia del bloqueo hasta que se completan. En esto se basa la decisión de saber si estos lenguajes implementan el patrón.

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí/No	Aunque la especificación de BPMN 1.2 si lo incluye no está claro como se especifica la expresión de la condición de entrada a la pasarela de la unión compleja, pero como ya se ha dicho es necesario llevar un seguimiento de los procesos para saber, cuales están accediendo al discriminador. En este caso BPMN no proporciona implementación total a este patrón.
YAWL	2.2.	No	No lo soporta debido a que no hay ninguna estructura o regla que permita guardar un control de las tareas que han comenzado.
UML	2.2	Sí	No está clara la especificación de la condición para lograr la unión, pero en esta versión de UML, esta construcción es básica, con lo que hay que centrarse en la premisa de control de los procesos. Lo que UML si implementa gracias al <i>CentralBufferNode</i> .

11) Discriminador de Cancelación

Con el fin de aplicar este patrón, es necesario que haya un medio que indique las ramas entrantes que deben ser canceladas. Esto puede requerir el patrón de “*Región de Cancelación*”, aunque sólo se requiere de una forma limitada en la zona donde todas las ramas se unan en el patrón de cancelación. En BPMN se consigue incorporando las ramas entrantes y este patrón en un subproceso que tiene un evento de error asociado a él. Este evento se dispara cancelando las ramas restantes en el subproceso en el momento en el que este patrón se activa por la primera rama entrante. UML soporta el modelo de un modo similar, incluyendo todas las ramas entrantes en una región incorruptible que se cancelan cuando se activa este patrón.

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Soportado gracias a que incluye actividades y O-unión en un sub-proceso que pasa el control a la siguiente actividad una vez que la primera rama ha terminado, así como la cancelación de las restantes actividades en el sub-proceso usando un error de tipo intermedio.

Lenguaje	Versión	Implementado	Motivación
YAWL	2.2	Sí	Lo soporta gracias a las condiciones y las tareas de cancelación que implementa.
UML	2.2	Sí	Soportado gracias a la incorporación de arcos de entrada a una <i>ExpansionRegion</i> .

12) Unión Parcial Estructurada

Una de las dificultades para construir este patrón es que requiere condiciones específicas y la claridad del modelo para poder representar la unión para que se represente de la forma más clara y sencilla posible y no dificulte su uso. Los lenguajes de modelado de procesos de negocio, como BPMN, soportan este patrón mediante la construcción de una entrada compleja, pero no permite el detalle en la condición de entrada y los modelos diseñados con BPMN se complican. UML también sufre una carencia similar de detalle en la condición de entrada.

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí/No	La especificación de este patrón para BPMN 1.2 no queda clara la especificación de la expresión de la nueva condición para la pasarela compleja de la unión.
YAWL	2.2	Sí	Lo soporta gracias al empleo del Y-Unión y el empleo de condiciones.
UML	2.2	Sí/No	La especificación de la expresión de la condición del <i>JoinNode</i> no se especifica de forma clara.

13) Unión Parcial de Bloqueo

La realización de este patrón es muy parecida a la del anterior excepto que la unión parcial se activa cuando n ramas entrantes se han desencadenado. BPMN y UML 2.2 lo soportan parcialmente.

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí/No	Aunque BPMN 1.2 soporta este patrón, no queda claro cómo especifica la expresión de la condición de entrada para la pasarela compleja de unión.

Lenguaje	Versión	Implementado	Motivación
YAWL	2.2	Sí	Mediante el uso de Y-Unión y condiciones, se puede representar este patrón.
UML	2.2	Sí	Aunque la condición el <i>JoinNode</i> , según [8] no se puede especificar claramente, esta nueva versión de UML, si declara las condiciones de forma clara, proporcionando la clase <i>ConditionalNode</i> .

14) Unión Parcial de Cancelación

El enfoque de la aplicación de este modelo es esencialmente el mismo que el de la *Cancelación Discriminatoria*. UML, no tiene problemas en representar este tipo de patrón, en cambio BPMN sólo obtiene el soporte parcial.

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	A pesar de que en [8] indiquen que no se puede implementar este patrón, la nueva versión con las pasarelas y los flujos de control basados en condiciones permiten especificar correctamente la condición que permite que se soporte este patrón.
YAWL	2.2	Sí	Gracias a Y-Unión y con las tareas de cancelación que ofrece.
UML	2.2	Sí	Lo soporta gracias a la incorporación de ramas de entrada al <i>JoinNode</i> de una <i>InterruptibleActivityRegion</i> .

15) Y-Union Generalizada

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Soportado con la pasarela Y-Unión de combinación.
YAWL	2.2	Sí	Soportado con la tarea Y-Unión.
UML	2.2	No	No lo soporta. La semántica del <i>JoinNode</i> previene que se derive a esta situación.

16) Fusión Local de Sincronización

Este patrón puede representarse mediante la combinación de varios patrones, Multi-elección, Fusión Sincronizada (los cuales ya se han visto antes) y la Elección Aplazada (que se verá más adelante).

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Según [8], no se puede implementar debido a que la estructura de la pasarela O-Unión no lo contempla, pero sí se puede combinar varios patrones (los ya indicados más arriba) y que este lenguaje sí soporta, dando por tanto soporte a este patrón también.
YAWL	2.2	Sí	Mediante la combinación de los patrones anteriores, se permite la implementación en YAWL de este.
UML	2.2	Sí	En [8] se indica que el soporte o la implementación es parcial, pero se puede implementar fácilmente con la combinación de los tres patrones que sí implementa, con lo que este patrón también.

17) Fusión General de Sincronización

Este patrón se puede representar con la combinación de varios patrones, Multi-elección, Fusión de Sincronización, Elección Aplazada y la estructura O Exclusiva.

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	A pesar de que explícitamente, no existe ningún concepto que represente este patrón, por la combinación de los patrones antes citados y que sí implementa este lenguaje, perfectamente se puede implementar este patrón.
YAWL	2.2	Sí	Mediante la combinación de los patrones anteriores, se permite la implementación en YAWL de este.

Lenguaje	Versión	Implementado	Motivación
UML	2.2	Sí	A pesar de qué explícitamente, no existe ningún concepto que represente este patrón, por la combinación de los patrones antes citados y que sí implementa este lenguaje, perfectamente se puede implementar este patrón.

18) Fusión de Hilos

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Es soportado mediante el establecimiento en el inicio de la cantidad de atributos de las actividades después de la actividad de inicio.
YAWL	2.2	Sí	Mediante instancias múltiples de tareas atómicas y condiciones de entrada.
UML	2.2	Sí	Incluye la ponderación de las transiciones entre <i>activities</i> de inicio para cualquier actividad posterior, mediante el uso de variables con valores.

19) Division de Hilos

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Es soportado mediante el establecimiento en el inicio de la cantidad de atributos en la secuencia de flujo saliente de una actividad.
YAWL	2.2	Sí	Mediante instancias múltiples de tareas atómicas y condiciones de salida.
UML	2.2	Sí	Incluye la ponderación de las transiciones entre <i>activities</i> de inicio para cualquier <i>activity</i> posterior mediante el uso de valores en las variables.

20) Múltiples Instancias sin Sincronización

En la mayoría de los casos que soportan este patrón, se hace mediante un bucle.

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Lo implementa mediante múltiples instancias de tareas iniciales con una condición de flujo y sin atributo definido.
YAWL	2.2	Sí	Emplea para ello parámetros dinámicos, límite máximo y mínimo y un umbral, con la ayuda de la composición de tareas.
UML	2.2	Sí	Lo realiza generando nuevas instancias de <i>activity</i> fuera de un <i>LoopNode</i> .

21) Múltiples Instancias con Conocimiento a Priori en Tiempo de Diseño

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Lo implementa mediante múltiples instancias de tareas iniciales con una condición de flujo y definiendo todos los atributos.
YAWL	2.2	Sí	Emplea para ello parámetros dinámicos, límite máximo y mínimo y un umbral, con la ayuda de la composición de tareas.
UML	2.2	Sí	Lo soporta con el uso de <i>ExpansionRegion</i> .

22) Múltiples Instancias Con Conocimiento a Priori en Tiempo De Ejecución

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Lo soporta mediante múltiples instancias de tareas y atributos que se requieren en tiempo de ejecución.
YAWL	2.2	Sí	Emplea para ello parámetros dinámicos, límite máximo y mínimo y un umbral, con la ayuda de la composición de tareas.
UML	2.2	Sí	Lo soporta con el uso de <i>ExpansionRegion</i> .

23) Múltiples Instancias Sin Conocimiento A Priori En Tiempo De Ejecución

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	No	No soportado. No se puede añadir múltiples instancias de una tarea una vez iniciada la ejecución.
YAWL	2.2	Sí	Emplea para ello parámetros dinámicos, límite máximo y mínimo y un umbral, con la ayuda de la composición de tareas.
UML	2.2	No	No soportado. No se puede añadir nuevos casos ni instancias de multi-loop una vez iniciada la ejecución.

24) Unión Estática y Parcial para Múltiples Instancias

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí/No	Supuestamente se soporta a través de múltiples instancias de tareas con la condición inicial del flujo que es el conjunto de atributos complejos y la complejidad de la condición del flujo es una expresión que evalúa a verdadero cuando exactamente M instancias han terminado y pasan a una única transición para la siguiente actividad. Sin embargo no está claro cómo debe definirse la compleja condición del flujo.
YAWL	2.2	Sí	Emplea para ello parámetros dinámicos, límite máximo y mínimo y un umbral, con la ayuda de la composición de tareas.
UML	2.2	No	No lo soporta. La <i>activity</i> inicial sólo puede ser completada cuando N instancias en la <i>ExpansionRegion</i> han finalizado.

25) Unión de Cancelación Parcial para Múltiples Instancias

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	BPMN teóricamente lo alcanza a través de una tarea en la que se ha producido un error que ha sido desencadenado por un tipo de evento intermedio en la frontera. Después de la tarea se pasa a una actividad que emite un evento cancelar que pone fin a cualquier actividad que no ha finalizado.
YAWL	2.2	Sí	Emplea para ello parámetros dinámicos, límite máximo y mínimo y un umbral, con la ayuda de la composición de tareas.
UML	2.2	No	UML no lo soporta. La <i>activity</i> inicial sólo puede terminar cuando todos las N instancias especificadas en la <i>ExpansionRegion</i> han terminado.

26) Unión Dinámica y Parcial para Múltiples Instancias

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	No	No lo soporta. No hay capacidad para añadir dinámicamente instancias a una actividad multi-instancia.
YAWL	2.2	Sí	Emplea para ello parámetros dinámicos, límite máximo y mínimo y un umbral, con la ayuda de la composición de tareas.
UML	2.2	No	No lo soporta. No hay forma de añadir dinámicamente instancias a <i>activity</i> .

27) Elección Aplazada

En este caso para BPMN 1.2 y UML 2.2 la elección se hace basándose en los mensajes de interacción de eventos.

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Soportado mediante eventos basados en pasarelas exclusivas seguidas por cualquier evento intermedio usando mensajes basados en disparadores o recibiendo tareas.
YAWL	2.2	Sí	Lo implementa gracias a la utilización de condiciones y el uso de un Y-Divisor.
UML	2.2	Sí	Se soporta con una <i>ForkNode</i> y un conjunto de acciones <i>SendSignalAction</i> , uno anterior a cada elección.

28) Enrutamiento Paralelo Intercalado

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	No	Incluye una tarea sencilla y un proceso ad-hoc, pero no apoya a grupos entrelazados o secuencias de tareas.
YAWL	2.2	No	Las redes de Petri se centran en tareas, eventos y no en estados. Con lo que no representan los patrones que necesiten de estados para su estructura.
UML	2.2	No	No lo soporta. No hay noción de estado dentro de UML 2.2.

29) Hito

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	No	No soportado. No tiene soporte para estados.
YAWL	2.2	No	Las redes de Petri se centran en tareas, eventos y no en estados. Con lo que no representan los patrones que necesiten de estados para su estructura.

Lenguaje	Versión	Implementado	Motivación
UML	2.2	No	No lo soporta. No hay noción de estado dentro de UML 2.2. Ni tampoco se consigue con la combinación de otras construcciones.

30) Sección Crítica

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	No	No soportado. No hay forma de limitar la ejecución simultánea de un conjunto de actividades.
YAWL	2.2	No	No soportado. No hay forma de limitar la ejecución simultánea de un conjunto de actividades. Además Las redes de Petri se centran en tareas, eventos y no en estados. Con lo que no representan los patrones que necesiten de estados para su estructura.
UML	2.2	No	No lo soporta. No hay forma de prevenir la ejecución simultánea de un conjunto de <i>activities</i>

31) Enrutamiento Intercalado

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí/No	Lo soporta gracias a un grupo de procesos ad-hoc que contiene todas las actividades que se han entrelazado con el conjunto ad-hoc secuencial, sin embargo no está claro lo que requiere la condición de terminación, ya que se debe garantizar que cada actividad se ejecuta con precisión una vez.
YAWL	2.2	Sí	Sí lo permite mediante el uso de condiciones y la tarea O Exclusivo Divisor y la tarea Y Exclusivo Unión.

Lenguaje	Versión	Implementado	Motivación
UML	2.2	Sí	Se puede implementar mediante el uso del <i>InitialNode</i> y <i>FinalNode</i> , combinándolo con <i>InputPin</i> y <i>OutputPin</i> .

32) Cancelación de una Tarea

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Lo soporta gracias a un evento de tipo error desencadenante intermedio que se adjunta a la frontera de las actividades que tienen que ser canceladas.
YAWL	2.2	Sí	Para soportar este patrón define la región con tokens a eliminar, donde se introducirán las actividades o tareas que pueden ser canceladas.
UML	2.2	Sí	Incorpora la <i>InterruptibleActivityRegion</i> desencadenando por señales o por la ejecución de otra <i>activity</i> .

33) Cancelación de un Caso

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Se incluye todo el proceso en una transición. Se hace un disparo final para cancelar el evento asociado con la transacción de manera efectiva a todas las actividades relacionadas con la instancia del proceso.
YAWL	2.2	Sí	Para soportar este patrón define la región con tokens a eliminar, donde se introducirán las tareas que pueden ser canceladas.

Lenguaje	Versión	Implementado	Motivación
UML	2.2	Sí	Incluye todas las <i>activities</i> en una <i>InterruptibleActivityRegion</i> y las cancela provocando una señal o mediante la ejecución de otra tarea.

34) Cancelación de una Región

El concepto de cancelación de regiones no es mayoritariamente soportado. Los diagramas de actividad de UML 2.2 son los únicos que soportan este patrón. BPMN ofrece soporte parcial adjuntando las tareas que potencialmente pueden ser canceladas en un subproceso y asocia un suceso de error al subproceso de cancelación para activarlo cuando sea necesario.

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí/No	Lo soporta mediante la inclusión de una región de cancelación en un sub-proceso y asociando un evento de error con el sub-proceso para permitir la cancelación. Esta región está limitada por el proceso general del modelo.
YAWL	2.2	Sí/No	Para soportar este patrón define la región con tokens a eliminar, donde se introducirán las actividades o tareas que pueden ser canceladas.
UML	2.2	Sí	Lo soporta la construcción de una <i>InterruptibleActivityRegion</i> .

35) Cancelación de Actividad con Múltiples Instancias

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Se alcanza a través de múltiples instancias de una tarea, que tiene un tipo de error intermedio de un disparador de error en el borde o límite de una región. Cuando han finalizado las múltiples instancias de la tarea, el evento cancelar es activado para terminar cualquier instancia múltiple de la tarea.

Lenguaje	Versión	Implementado	Motivación
YAWL	2.2	Sí	Para soportar este patrón define la región con tokens a eliminar, donde se introducirán las actividades o tareas que pueden ser canceladas.
UML	2.2	Sí	Se incluye la construcción de una <i>InterruptibleActivityRegion</i> .

36) Terminación de Actividad de Múltiples Instancias

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	No	No lo soporta. No hay forma de cancelar las restantes múltiples instancias de actividad que no han sido canceladas.
YAWL	2.2	No	No permite la terminación de actividades con múltiples instancias, ya que las múltiples instancias las implementa con tareas compuestas y no pueden terminarse estas tareas individualmente.
UML	2.2	No	No lo soporta. No hay forma de finalizar las instancias de la actividad una vez han comenzado múltiples instancias de una <i>activity</i> .

37) Ciclos Arbitrarios

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Soporta directamente la recepción no estructurada. Lo soporta mediante una actividad bucle.
YAWL	2.2	Sí	Se implementa mediante condiciones de entrada y salida, además de emplear la composición de tareas.
UML	2.2	Sí	Soporta ciclos no estructurados. Mediante <i>LoopNode</i> .

38) Bucle Estructurado

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Tanto el bucle <i>mientras</i> como <i>repetir</i> se soportan con una actividad bucle.
YAWL	2.2	Sí	Se implementa mediante condiciones de entrada y salida, además de emplear la composición de tareas.
UML	2.2	Sí	Se soporta con la construcción de un <i>LoopNode</i> .

39) Recursividad

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	No	No lo soporta. No hay medio para especificar la composición recursiva con un modelo de proceso.
YAWL	2.2	Sí	Se implementa mediante condiciones de entrada y salida, además de emplear la composición de tareas.
UML	2.2	Sí	Lo soporta gracias al empleo de un <i>LoopNode</i> y con la especificación de un <i>ConditionalNode</i> , con las condiciones necesarias por el modelo.

40) Terminación Implícita

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Es soportado mediante un evento final.
YAWL	2.2	No	No lo soporta. No hay ninguna forma de terminar un proceso que ha comenzado la ejecución sin llegar al final.
UML	2.2	Sí	Es soportado mediante la construcción de un nodo de actividad final.

41) Terminación Explícita

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Soportado por cada hilo que termina con un evento final. Cuando el último token generado por el evento de inicio acaba, la instancia del proceso terminará.
YAWL	2.2	Sí	Se puede decir que sí lo soporta, cuando un proceso acaba de ejecutarse.
UML	2.2	Sí	Se soporta con la construcción de un <i>FlowFinalNode</i> .

42) Disparador Temporal

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Esta versión de BPMN implementa una gran variedad de eventos que lanzan disparadores que permiten la implementación de este patrón. En este caso se podría implementar con el evento temporizador.
YAWL	2.2	Sí	Lo implementa mediante tareas de tiempo. Tareas que pueden esperar un evento, tareas que pueden ser en sí un evento, etc.
UML	2.2	Sí	Proporciona la capacidad de desechar las señales, cuando no se recoge la señal inmediatamente. Designa la característica <i>AcceptEventAction</i> que tramita todas las señales.

43) Disparador Persistente

BPMN proporciona un mecanismo de este tipo de activación a través de mensajes y en todos los casos los mensajes son de naturaleza duradera (disparadores persistentes) y pueden desencadenar tareas independientes o pueden permitir que una tarea se bloquee a la espera de la recepción del evento o mensaje para continuar. UML 2.2 ofrece un servicio similar mediante la utilización de señales.

Lenguaje	Versión	Implementado	Motivación
BPMN	1.2	Sí	Soportado mediante el uso de eventos de mensajes.
YAWL	2.2	Sí	Lo implementa mediante tareas de tiempo. Tareas que pueden esperar un evento, tareas que pueden ser en sí un evento, etc.
UML	2.2	Sí	Soportado mediante el uso de señales, con la acción <i>SendSignalAction</i> .

Llegando al final de este apartado para finalizar este trabajo, se van a describir las conclusiones a las que se ha llegado después del análisis realizado a BPMN, UML y YAWL, para poder decidir cuál de los anteriores lenguajes es el más adecuado para modelar procesos de negocio.

Se ha intentado llevar un orden en las conclusiones basándose en el orden en el cual se han descrito los lenguajes.

3. CONCLUSIONES

El objetivo de estas conclusiones, como ya se ha dicho, es determinar cuál de los lenguajes de modelado es el más adecuado para modelar procesos de negocio, permitiendo mejorar el desarrollo de software dirigido por modelos.

A la vista de este documento, se puede decir que BPMN proporciona una notación más sencilla de entender y emplear para el modelado de procesos de negocio. Además, en la actualidad, BPMN es el estándar *de facto* de modelado de procesos de negocio. Esto se puede deber a que únicamente considera un diagrama para la representación de los procesos de negocio. Una consecuencia directa de la sencillez para modelar y entender la notación, es que facilita a los observadores del modelo entenderlo, así como a los propios desarrolladores del modelo y a otros, adaptarlo tanto al cambio como a posibles extensiones de éste.

BPMN permite establecer una dirección de mejora y eficiencia continua al convertir actividades ineficientes en reducción de gastos y reducción de tiempo a través del uso de los procesos de negocio. Además reduce costes futuros en integración y mantenimiento.

BPMN sirve de gran ayuda a las empresas, ya que les permite hacer un mejor uso de los procesos de negocio, regula el método de notación que sirve como ayuda en la automatización de estos. Además, al estar todos los procesos organizados de una manera unificada y estandarizada, permite a la organización conocer las actividades de la empresa, pudiendo facilitar en cierto modo el trabajo. Se muestra esto último como una ventaja ya que la empresa puede aprovechar la oportunidad de mejorar los procesos en el caso de que algún paso no estuviera del todo claro.

Esta notación (BPMN) tan internacionalmente utilizada y por ello entendida, nos permite comprender y gestionar los procesos de cualquier empresa. Por ello, este hecho supone un aumento de la eficiencia entre distintas compañías e incluso un valor en la cadena productiva tanto de la empresa como del sector, ya que tendrán mayor facilidad a la hora de integrarse.

BPMN reduce los errores que cualquier empresa pueda tener en su diseño de procesos de negocio, haciendo que estos se comporten siempre del mismo modo y dando elementos que permitan visualizar el estado de los mismos. Teniendo en cuenta el tiempo y costes que supone cambiar algún orden de proceso, BPMN reduce tanto uno como otro. Es bastante favorable para las empresas que están constantemente adaptándose a los cambios del sector, la tecnología e incluso avances de la sociedad. Como prueba de ello, se puede mencionar la compra de vuelos, las reservas de hotel o la compra de entradas por internet entre otros. Detrás de todas ellas, hay una metodología BPM y BPMN, que son las que hacen posible la misma reserva teniendo en cuenta la disponibilidad en las fechas seleccionadas, el descuento por comprar con antelación, etc.

El punto de vista no tan positivo es que es difícil que un proceso pueda reflejar todas las variables y excepciones que se puedan dar en la vida real o en el problema a

modelar. En el caso de que fuera posible, podría llegar a ser bastante complejo. Al establecer un BPM en una empresa, no tiene porque tener un gran impacto en la empresa, todo depende del sistema BPM que se elija y del tiempo de adaptación al nuevo sistema, pudiéndose hacer poco a poco. También, BPMN está pensado para ser asignado con naturalidad a lenguajes de ejecución gracias a BPEL4WS (del inglés *Business Process Execution Language For Web Services*).

Sin embargo, el problema que presenta es que BPMN no tiene definido un metamodelo que defina reglas sintácticas y semánticas para su construcción, y que por tanto permita la construcción de herramientas compatibles entre sí y la integración coherente y uniforme con lenguajes de programación, ya que BPMN puede ser interpretado de distinta manera según el modelador y el observador del modelo. Esto hace que BPMN no se pueda considerar un lenguaje formal. En la actualidad se está trabajando actualmente en el metamodelo de BPMN, pero mientras no exista un metamodelo definido resulta muy complicado integrar los procesos de negocio modelados con BPMN adoptando el paradigma MDE (es decir, el enfoque del desarrollo de software dirigido por modelos), donde los metamodelos juegan un papel muy significativo de cara a la interoperabilidad de los modelos en distintos entornos.

Respecto a UML, es un lenguaje más complejo de emplear debido a que se trata de un lenguaje de propósito general para modelado de software con una semántica un tanto ambigua. Sin embargo, UML presenta notación específica para modelar procesos de negocio y gracias al metamodelo de UML, los modelos son más robustos y portables, siendo esto el punto fuerte de UML 2.2. Además, ayuda al reconocimiento rápido de errores, ahorro de tiempo en el desarrollo de software si se ha conseguido modelar un proceso de negocio que represente los objetivos establecidos, ya que se puede seguir utilizando el mismo lenguaje UML para modelar el sistema software a partir del proceso de negocio al que da soporte. Además, UML facilita la extensibilidad y las modificaciones del modelo, lo que favorece la comunicación de los programadores y modeladores.

Por otro lado, UML es un lenguaje ya consolidado y ampliamente conocido, ya que es considerado el estándar de facto para modelado de software y que, como se mencionaba antes, permite modelar con distintos diagramas todo el sistema software completo (estructura y comportamiento) y no sólo los procesos de negocio (con diagramas de actividad) que contienen el sistema software a implementar. Esta capacidad podría hacer que fuera más sencilla la integración de los modelos de negocio en un entorno de desarrollo de software dirigido por modelos.

A pesar de que algunos autores lo consideran fácil de aprender, en las conclusiones de este documento, se puede afirmar que desde este punto de vista, esto no es así. El estudio de UML ha sido el más complejo debido a la gran documentación de la última versión de UML 2.2 [44], la semántica y la gran variedad de posibilidades que ofrece en su notación, haciéndolo complejo y tedioso a la hora de entender, relacionar y obtener conclusiones. Contiene diagramas y estructuras que son redundantes o muy

poco usadas. Además, como ya se mencionaba anteriormente, la semántica es ambigua lo que hace que UML no sea un lenguaje formal, lo que puede llevar a distintas interpretaciones del modelo, con lo que ello supondría.

Hasta aquí las ventajas y desventajas de los dos lenguajes no formales que se han expuesto en este documento. No hay que olvidar el lenguaje formal YAWL basado en redes de Petri, también muy empleado en la definición de procesos de negocio pero que no nos permiten la integración con otras herramientas.

Gracias a las bases matemáticas en las que descansa la teoría de redes de Petri proveen definiciones precisas y una semántica clara.

La representación gráfica de las redes de Petri es bastante sencilla e intuitiva, fácil de aprender, y existen muchas aplicaciones de software que permiten de manera sencilla y genérica diseñar y modelar sistemas. Pueden soportar primitivas funcionales necesarias para modelar sistemas de procesos de negocio de documentos sin tener detalle de qué contienen dichos documentos. Sin embargo se puede agregar expresividad con las extensiones de datos y de tiempo a los modelos, por medio de las diferentes extensiones de las redes de Petri.

Algunos autores como en [6] exponen las carencias y las desventajas de modelar con redes de Petri los procesos de negocio. En [9] también se presenta uno de los argumentos más simples que pueden ser encontrados. Se menciona que las redes de Petri fueron inventadas antes de que lo fuesen los sistemas de administradores de flujo de trabajo. Por lo tanto no poseen las características necesarias. Sin embargo se puede pensar que por esa misma razón han surgido diversas modificaciones o extensiones para modelar flujos de trabajo de una manera más natural. Otro argumento es que las redes de Petri sólo sirven para modelar sistemas cerrados y no sistemas reactivos, ya que se considera un flujo de trabajo como un sistema reactivo [6], esto es por la naturaleza de disparo de las transiciones de las redes de Petri. En [9] se argumenta el porqué las redes de Petri de alto nivel no proporcionan los elementos suficientes para modelar ciertas situaciones que pueden aparecer en un proceso de flujo de trabajo.

Aun que todas las anotaciones tienen problemas a la hora de modelar estados, estructuras del sistema y el historial, como se ha visto anteriormente, en el análisis comparativo, UML apoya la construcción básica del estado, que es totalmente omitido tanto en BPMN y YAWL. Uno de los problemas de implementación de YAWL son algunas limitaciones que ha sido necesario realizar en el modelo de interpretación para la implementación de procesos de negocio.

Además, el exceso de construcción y constructores, proporcionan que la sobrecarga este presente tanto en BPMN como en UML. Este problema de sobrecarga que tienen BPMN y UML se refiere a qué: *pool* (BPMN), *lane* (BPMN), *pool* (UML), pueden representar muchas cosas en el modelo. Cosa, clase, tipo, sistema, subsistema, todos pueden ser representado ya sea en la *pool*, como en la *lane*.

El resultado de utilizar los patrones de flujos de trabajo para comparar la notación de BPMN, UML y YAWL, para su utilización en el modelado de procesos de negocio puede verse resumido en la Tabla 9. La primera columna indica el lenguaje, la segunda (Sí) indica el número de patrones que soporta el lenguaje, la segunda el número de patrones que no soporta y la tercera indica el número de patrones a los que da soporte parcial.

Lenguaje	Sí	No	Sí/No
BPMN	30	7	6
YAWL	36	6	1
UML	33	9	1

Tabla 9. Patrones soportados por los lenguajes

Viendo la tabla 9, se puede decir que YAWL es más expresivo debido a que da soporte a más patrones que BPMN y UML 2.2, por las razones anteriormente expuestas, no nos interesa para el modelado de procesos de negocio. En este caso UML2.2 es más expresivo que BPMN, aparte de otras características que ya se han citado y que interesan más para el modelado de procesos de negocio.

Al final, las diferencias del poder de representación de las notaciones pueden ser bastante estrecha: desde un punto de vista, BPMN y UML se pueden encontrar similares en muchos aspectos. Hay algunas diferencias que hacen que se prefiera UML, pero no hay una investigación suficiente sobre este asunto para llegar a estas conclusiones. Mientras que unos encuentran que UML tiene mayor poder de representación debido a la falta de la redundancia, otros pueden encontrar que este tipo de detalles no tienen ninguna relevancia.

Ahora bien, para poder utilizar correctamente los modelos resulta fundamental construir metamodelos que permitan sostener y formalizar los lenguajes de modelado en los que están basados los modelos. En este estudio se puede decir, que UML sería mejor para representar modelos que BPMN y YAWL, debido a que proporciona un metamodelo que permite integrar los procesos de negocio en un proceso de desarrollo de software dirigido por modelos.

Lo que resultaría interesante sería poder llegar a un estudio más amplio sobre las herramientas que integran los procesos de negocio y los lenguajes que facilitan esta integración. En este aspecto se podría ampliar este estudio, pero no se ha llevado a cabo ya que alargaría mucho más tiempo este estudio, debido a lo amplio que es el campo. Se puede emplear este estudio para la motivación de estos trabajos.

Con lo que UML además de todas las ventajas que ya se han citado, frente a los otros lenguajes, es el que facilita integrar, gracias al metamodelo que define, los procesos de negocio en desarrollos software, centrándose en mejorar estos, para no tener que centrarse en procesos de más bajo nivel y relegar a un segundo plano la programación del software, gracias a la definición clara de los procesos de negocio.

4. LINEAS FUTURAS DE TRABAJO

Una vez finalizado este estudio, se pueden proponer algunas líneas de trabajo que pueden tomar como base este documento.

Se puede llevar a cabo un estudio más concreto sobre la integración de aplicaciones de empresa, conocidas como EAI (del inglés *Enterprise Application Integration*), que se define como el uso de software y principios de arquitectura de sistemas para integrar un conjunto de aplicaciones. Sería interesante estudiar el proceso principal de estas aplicaciones y cómo intercambian información operativa o financiera, para mejorar el rendimiento, mejorar los tiempos de desarrollo y disminuir los costes.

La *Integración de Aplicaciones de Empresa* (EAI, del inglés *Enterprise Application Integration*) se define como el uso de software y principios de arquitectura de sistemas para integrar un conjunto de aplicaciones.

Es el proceso de conectar las aplicaciones con otras para intercambiar información operativa o financiera. Cuando dichos sistemas no pueden compartir su información efectivamente se crean cuellos de botella que requieren de la intervención humana en la forma de toma de decisiones o en el ingreso mismo de la información. Con una arquitectura EAI correctamente implementada, las organizaciones pueden enfocar la mayoría de sus esfuerzos en la creación de competencias que generen valor en lugar de enfocarse en la coordinación de labores operativas.

Uno de los retos que encaran las organizaciones modernas es darles a sus empleados información completa en tiempo real. Muchas de las aplicaciones en uso actualmente se apoyan en tecnologías antiguas, por lo cual esos sistemas enfrentan dificultades a la hora de mover esta información entre las aplicaciones.

EAI, como una disciplina, busca solventar muchos de esos problemas, así como crear nuevos paradigmas para ciertamente mejorar las organizaciones, tratando de trascender el objetivo de conectar las aplicaciones individuales para buscar ser un mecanismo de incrementar el conocimiento de la organización y crear ventajas competitivas futuras a la empresa.

A pesar de que en este documento se han citado brevemente algunas de las herramientas que existen para el modelado de procesos de negocio, sería interesante analizar las nuevas herramientas de software y realizar una comparación con lo que hay en el mercado actualmente, cuales son los avances que mejoran las herramientas ya existentes o si por el contrario no lo hacen y su implantación en las empresas.

5. BIBLIOGRAFÍA

- [1]. *The Application of Petri Nets to Workflow Management*. W.M.P. Vander Aalst.
- [2]. Hollingsworth. David. *The Workflow Reference Model. Issue 1.1. Document Number TC-1003*, Workflow Management Coalition, 19-Jan- 1995.
- [3]. *The Workflow Management Coalition Specification. Issue 3.0. Document Number TC-1011*, Workflow Management Coalition, Feb-1999.
- [4]. *Web The Workflow Management Coalition*: www.wfmc.org.
- [5]. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. PhD thesis, University of Twente, Enschede, The Netherlands, 2002. Eshuis. Hendrik.
- [6]. *Fundamentals of Control Flow in Workflows*. Kiepuszewski I, A.H.M. ter Hofstede I, and W.M.P. Vander Aalst I (Se puede encontrar en <http://eprints.qut.edu.au/archive/00009948/>).
- [7]. *Expressiveness and Suitability of Languages for Control Flow Modeling in Workflows*. Bartosz Kiepuszewski. M.Sc.
- [8]. *Donde se pueden encontrar patrones para el modelado de Workflows*: <http://www.workflowpatterns.com>.
- [9]. *YAWL: Yet Another Workflow Language*. W.M.P. Vander Aalst and A.H.M. ter Hofstede (<http://eprints.qut.edu.au/archive/00010244/>)
- [10]. *Petri nets: Properties, analysis and applications*. *Proceedings of the IEEE*, 77(4):541—580, Abril 1989. Tadao Murata.
- [11]. *Business Process Management Initiative* (<http://www.bpmi.org/>)
- [12]. *Business Process Management Notation v1.2, 2009* (www.bpmn.org)
- [13]. *The Association of Business Process Management Professionals*. (www.abpmp.org)
- [14]. *Business Process Modeling Notation (BPMN) Specification Final Adopted Specification dtc/06-02-01*
- [15]. *Business Process Definition MetaModel (BPDM) (Final submission) March 5, 2007*.
- [16]. *Object Management Group* (www.omg.org).
- [17]. *Business Modeling & Integration DTF* (<http://bmi.omg.org/>).

-
- [18]. *Business Process Model and Notation (BPMN) 1.2. Request For Proposal OMG Document: BMI/2007-06-05.*
- [19]. *OMG Specifications for Business Modeling – BPDM. By Jon Siegel, Ph.D. Vice President, Technology Transfer Object Management Group.*
- [20]. *Business Process Management: A Survey. W.M.P. Vander Aalst, A.H.M. ter Hofstede, and M. Weske.*
- [21]. http://en.wikipedia.org/wiki/Meta_model
- [22]. <http://www.omg.org/mof/>
- [23]. *Workflow Data Patterns. Nick Russell1, Arthur H.M. ter Hofstede, David Edmond.*
- [24]. http://es.wikipedia.org/wiki/Lenguaje_de_especificaci%C3%B3n_OCL2.0
- [25]. *Workflow Resource Patterns Nick Russell1, Arthur H.M. ter Hofstede1, David Edmond.*
- [26]. *Workflow Control-Flow Patterns. A Revised View. Nick Russell1, Arthur H.M. ter Hofstede.*
- [27]. *Exception Handling Patterns in Process-Aware Information Systems. Nick Russell1, Wil M.P. Vander Aalst2,1, and Arthur H.M. ter Hofstede.*
- [28]. <http://www.yawl-system.com/>
- [29]. <http://es.wikipedia.org/wiki/YAWL>
- [30]. *YAWL: Yet Another Workflow Language W.M.P. van der Aalst1, and A.H.M. ter Hofstede.*
- [31]. *K.M. Van Hee. Information System Engineering: a Formal Approach. Cambridge University Press, 1994.*
- [32]. *R.J. Van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. Journal of the ACM, 43(3):555–600, 1996.*
- [33]. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2003. B. Kiepuszewski. Encontrado en <http://www.workflowpatterns.com>.*
- [34]. *The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, 8(1):21–66, 1998. W.M.P. Vander Aalst. Encontrado en <http://www.workflowpatterns.com>.*
- [35]. *Workflow Management: Models, Methods, and Systems. MIT press, Cambridge, MA, 2002. W.M.P. Vander Aalst and K.M. Van Hee.*

- [36]. *Modelado de Workflow con Redes de Petri Coloreadas Condicionales. Tesis Doctoral de Samuel Garrido.*
- [37]. *Integration Definition For Function Modeling (IDEF0).* dmi.uib.es/~burguera/download/IDEF0trabajo.doc.
- [38]. *MDA: From Hype to Hope, and Reality, Conferenciante Invitado en UML'03 (2003).* Bézivin.J.
- [39]. *Model-Driven Engineerin, IEEE Computer Society, Febrero (2006), pag. 25-31.* Shmidt, DC.
- [40]. *The Emergence of Business Process Management, CSC's Research Services, Enero (2002).* Smith, H., Neal, D., Ferrara, L., Hayden, F.
- [41]. *The Rise of Model-Driven Enterprise Systems, Business Process Trends, Junio (2003), pag. 1-15.* Frankel, D.
- [42]. *What a BPMS Is, Business Process Trends, Febrero (2005).* Smith, H.
- [43]. *Introduction to BPMN.* Stephen A. White, IBM Corporation.
- [44]. *OMG Unified Modeling Language™ (OMG UML), Superstructure. Version™ 2.2.*
- [45]. "Model Driven Engineering: An Emerging Technical Space". In *Proceedings of the International Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE 2005).* Jean Bézivin.
- [46]. "Model Driven Engineering". In *Proceedings of the Third International Conference on Integrated Formal Methods, pp. 286-298, 2002.* Stuart Kent.
- [47]. *OMG. "Model Driven Architecture". Document ormsc/2001-07-01, July 2001.* Accesible en <http://www.omg.org/>.
- [48]. *MDA Distilled. Principles of Model-Driven Architecture.* Addison-Wesley, 2004 Stephen J. Mellor, Kendall Scott, Axel Uhl, Dirk Weise.
- [49]. *OMG. "MDA Guide Revision Draft", Version 00.03. Document ormsc/05-11-03, November 2005.* Accesible en <http://www.omg.org/>.
- [50]. "Towards a Precise Definition of the OMG/MDA Framework". In *Proceedings of 16th IEEE International Conference on Automated Software Engineering (ASE'01), November 2001* Jean Bézivin, Olivier Gerbé.
- [51]. *Workflow Modeling. Tools for Process Improvement and Application Development.* Artech House, 2001. Alec Sharp, Patrick McDermott.

[52]. *Enterprise Modeling with UML. Designing Successful Software through Business Analysis*. Addison-Wesley, 2000. Chris Marshall.

[53]. *The Object Advantage. Business Process Reengineering With Object Technology*. Addison-Wesley, 1995. Ivar Jacobson, Maria Ericsson, Agneta Jacobson.

[54]. *Business Modeling with UML. Business Patterns at Work*. Wiley, 2000. Hans-Erik Eriksson, Magnus Penker.

[55]. *Página Oficial de YAWL: <http://www.yawl-system.com/>*.

6. SOFTWARE

En este apartado se va a enumerar el software o las herramientas empleadas para dibujar las figuras expuestas en este trabajo. Generalmente, se han utilizado herramientas específicas del lenguaje que es estaba describiendo, en los casos en los que no había que seguir ningún estándar se han empleado herramientas de dibujo más accesibles y gratuitas, ya que en los otros casos se ha tenido que emplear una versión de prueba, ya que para este estudio no se ha visto necesaria la adquisición de la licencia para el uso de la herramienta durante un periodo de tiempo más largo.

{1} *Microsoft Office PowerPoint 2007*. Para dibujar diagramas sin estructura ni denominación concreta.

{2} *BizAgi Process Modeler Versión 14.0.0.162*. Para modelar los distintos elementos de BPMN.

{3} *Visual Paradigm for UML 7.1 Modeler Edition*. Para modelar los distintos conceptos de UML.

ANEXOS

A. ANEXO: PRESUPUESTO

Después de haber finalizado este estudio, se va a calcular el presupuesto, que supondría realizar este estudio. Como bien puede comprenderse en la materialización de este tipo de proyectos, no es necesaria la utilización de material alguno, únicamente se utiliza los recursos encontrados en la web (documentación), recursos humanos (persona que realiza el estudio) y además de las herramientas necesarias para poder llevar a cabo los objetivos del proyecto (herramientas para el modelado de procesos de negocio).

Lo primero que se va a realizar es calcular el tiempo que se ha empleado en el desarrollo de este trabajo. Debido a que se ha alargado el tiempo, sobre el planificado para finalizar este trabajo, se va a realizar una estimación. El esfuerzo empleado no ha sido a jornada completa (estimando por jornada completa: 8 horas/día), si no a tiempo parcial e intentando aprovechar tiempos muertos y fines de semana, con lo que no se puede hacer un cálculo exacto del tiempo, y hacer un cálculo general. Aunque no se haya realizado (6 meses a jornada completa, en horario laboral) en el plazo de tiempo que se indica, sí se compacta los tiempos empleados, saldría un total de los meses indicados en la Tabla 10.

Periodo de Realización	Días	Horas/Día	Total Horas (días*hora)
01/04/2009 – 30/09/2009	131 días	8 horas/día	1048 horas

Tabla 10. Total de horas empleadas

Todos los costes que se van a mostrar, no se les ha aplicado el 16% de IVA correspondiente. Este se añadirá en el cálculo final del coste del estudio.

En el caso de la documentación empleada (ver apartado de Bibliografía), se ha encontrado gratuitamente en la web, sin necesidad de pago de licencias, ni derechos de autor. Con lo que este recurso ha resultado gratuito. A continuación se muestra en la Tabla 11.

Recurso	Coste (€)	Horas (h)	Coste Total (€)
Bibliografía	0 €	1048 h	0 €

Tabla 11. Total euros requeridos en la bibliografía

Además del recurso humano o de personal, se puede decir, qué no sólo debe contarse con el coste/horas, por persona (en este caso, únicamente se va contabilizar al autor de este documento), sino que también con el recurso de tiempo disponible de esta persona, ya que, en este caso el recurso de tiempo ha sido limitado, debido a la

necesidad de su presencia en el puesto de trabajo. Para el desarrollo de este trabajo, el perfil mínimo al que se puede referir del autor de este trabajo es de, analista de procesos de negocio. En la Tabla 12.

Recurso	Coste/Hora (€)	Horas (h)	Total Coste (€)
Analista	25 €	1048 h	26200 (€)

Tabla 12. Total euros personal

En el caso de las herramientas utilizadas, no ha hecho falta obtener las licencias para este estudio. Se han empleado para ello, en algunos casos, software de prueba, que permitían un periodo de tiempo necesario para poder analizarlas y utilizarlas. En otros casos se ha utilizado software gratuito. Se muestra en la Tabla 13 el total de Euros requeridos para las herramientas.

Recurso	Coste/Año
<i>Visual Paradigm for UML 7.1 Modeler Edition</i>	0(€)
<i>BizAgi Process Modeler Versión 14.0.0.162</i>	0 (€)
<i>Microsoft Office PowerPoint 2007</i>	0(€)
Total Herramientas	0 (€)

Tabla 13. Total Euros requeridos en herramientas

Con lo que después de enumerar los recursos disponibles, se muestra una tabla del presupuesto total que se necesitaría para llevar a cabo este proyecto. Hay que decir, que el gasto en materiales ha sido mínimo, con lo que no se contabiliza el gasto en equipos (equipo portátil del analista), que es uno de los recursos que podría ser contabilizado. En cambio si se va a contabilizar la conexión mensual a internet ADSL con Telefónica. Se muestra en la Tabla 14 el importe total sin IVA.

Recurso	Coste
Bibliografía	0 (€)
Diseñador/Modelador	26.200 (€)
Herramientas de Software (Licencias)	0 (€)

Recurso	Coste
Conexión ADSL Internet (6 meses)	294 (€)
Total sin IVA	26494 (€)

Tabla 14. Total euros sin IVA

A continuación se muestra en la Tabla 14 el presupuesto total con IVA:

Gastos	Coste Total
Total Recursos	26.494 (€)
IVA (16%)	4.240(€)
Total + IVA (16%)	30.734(€)

Tabla 15. Total Presupuesto con IVA

Con lo que el presupuesto total asciende a **treinta mil setecientos treinta y cuatro (30.734 €) euros**.

