



**Facultad de Ingeniería
Universidad de la República**



Proyecto de Grado Ingeniería en Computación

Verificación de Procesos de Negocio

conceptos, métodos, lenguajes y tecnologías

Ignacio Goyetche - Nicolás Illanes

Tutores:

Daniel Calegari García

Andrea Delgado Cavaliere

DICIEMBRE 2014
MONTEVIDEO - URUGUAY

Resumen

La Gestión de Procesos de Negocio (Business Process Management, BPM) es un enfoque sistemático para identificar, documentar, diseñar, ejecutar, medir y controlar los procesos que realiza una organización, con la finalidad de lograr los objetivos de negocio que se encuentran alineados con su estrategia de negocio.

Se comienza con una primera fase de identificación de los Procesos (Análisis y Diseño), usualmente utilizando el lenguaje de modelado estándar BPMN (Business Process Management Notation). Esta descripción es utilizada en una fase posterior (Configuración) para el desarrollo de una solución informática que dé soporte a dichos procesos. En ambas se realizan actividades de verificación. Es importante poder determinar de forma anticipada si un proceso se va a comportar de forma deseable. Otras propiedades no pueden ser verificadas de forma anticipada y son evaluadas simulando la ejecución del modelo.

En este proyecto se realizó una revisión sistemática de la literatura sobre verificación de Procesos de Negocio. Como principal aporte, se brinda una clasificación de elementos existentes discriminada por distintas categorías de interés, que muestra qué se puede verificar y cómo verificar distintos aspectos. Como prueba de concepto se realizó un prototipo de un entorno integrado de verificación, que incluye herramientas de terceros. El prototipo fue utilizado en un Caso de Estudio para demostrar la factibilidad de su uso como soporte para la verificación de distintos elementos de los Procesos de Negocio.

Índice general

1. Introducción	9
2. Contexto	11
2.1. Introducción a Procesos de Negocio	11
2.2. Lenguajes de especificación de Procesos de Negocio	14
2.2.1. BPMN 2.0	15
2.2.2. Redes de Petri	17
2.2.3. UML (Diagramas de actividad)	20
2.2.4. And/Or graph	20
2.2.5. EPC	20
2.2.6. XPDL	21
2.2.7. WS-BPEL	21
2.2.8. YAWL	21
2.2.9. C-YAWL	21
2.3. Verificación de Procesos de Negocio	21
3. Revisión Sistemática	23
3.1. El proceso de la revisión	24
3.1.1. Planificación de la revisión	24
3.1.2. Selección de estudios y extracción de la información	25
3.2. Clasificación de los resultados	26
3.2.1. Tipo de proceso	26
3.2.2. Lenguajes de especificación	27
3.2.3. Etapa en la que se realiza la verificación	28
3.2.4. Herramientas utilizadas	28
3.2.5. Propiedades a verificar	28
3.2.6. Técnicas de verificación	29
3.2.7. Resumen de la clasificación	31
3.3. ¿Qué verificamos?	31
3.3.1. Propiedades de grafos	31
3.3.2. Recursos	33
3.3.3. Colaboración	33
3.3.4. Temporales	34

3.4.	¿Cómo se realiza la verificación?	35
3.4.1.	Model cheking	35
3.4.2.	Deducción Lógica	36
3.4.3.	Por construcción	36
3.4.4.	Análisis estático	37
3.4.5.	Testing-simulación	38
3.5.	Análisis de los resultados	39
3.6.	Conclusiones de la revisión sistemática	40
4.	Desarrollo de un entorno integrado de verificación	43
4.1.	Requisitos del prototipo para Eclipse	43
4.2.	Evaluación de herramientas	45
4.2.1.	Activiti	46
4.2.2.	WoPeD	46
4.2.3.	ProM	47
4.3.	Arquitectura del Plug-in	47
4.3.1.	Organización interna de componentes	48
4.3.2.	Integración con Eclipse	49
4.3.3.	Integración con Activiti	51
4.3.4.	Integración con WoPeD y ProM	53
4.3.5.	Extensibilidad	55
4.4.	Extensión del Plug-in	56
5.	Verificación del prototipo	59
5.1.	Análisis herramienta Activiti	59
5.1.1.	Caso de prueba 1: Agencia de Viajes	59
5.1.2.	Caso de prueba 2: Enviar Artículo	62
5.1.3.	Caso de prueba 3: Carrito de Compras	64
5.1.4.	Resultados de Activiti	66
5.2.	Análisis del Transformador y WoPeD	66
5.2.1.	Caso de prueba 1: Agencia de Viajes	66
5.2.2.	Caso de prueba 2: Enviar Artículo	68
5.2.3.	Caso de prueba 3: Carrito de Compras	70
5.2.4.	Resultados del transformador y WoPeD	73
5.3.	Limitaciones y errores conocidos	74
6.	Caso de Estudio	77
6.1.	Introducción	77
6.2.	Instalación y Entorno del Plug-in	81
6.3.	Verificación con BPMNValidator	83
6.3.1.	Primer paso: Modelo Inicial	83
6.3.2.	Segundo paso: Errores generados	85
6.3.3.	Tercer paso: Errores detectados con Activiti y WoPeD	87
6.3.4.	Cuarto paso: Soluciones planteadas para Activiti	88

6.3.5. Quinto paso: Soluciones planteadas para WoPeD	89
6.4. Análisis del Caso de Estudio	90
7. Conclusiones y trabajo a futuro	91
7.1. Conclusiones	91
7.2. Trabajo a futuro	92
A. Información adicional sobre Redes de Petri	103
A.1. Mapeo de conceptos de flujo de trabajo en Redes de Petri . .	103
A.2. Enrutamiento de fichas	104
A.2.1. Enrutamiento secuencial	104
A.2.2. Enrutamiento en paralelo	105
A.2.3. Ruteo Selectivo	106
B. Clasificación de Resultados de la Revisión Sistemática	107
C. Lista completa de errores detectados por Activiti	111
D. Gestión del proyecto	115
D.1. Grupo de trabajo	115
D.2. Documentación	115
D.3. Cronograma previsto	115
D.4. Etapas y cronograma real del proyecto	116
D.4.1. Contexto	116
D.4.2. Planificación y ejecución de la Revisión Sistemática . .	116
D.4.3. Análisis de resultados	116
D.4.4. Desarrollo de un entorno integrado de verificación . . .	117
D.4.5. Elaboración del informe	117
D.4.6. Cronograma real	117

Capítulo 1

Introducción

La Gestión de Procesos de Negocio (Business Process Management, BPM) es un enfoque sistemático para identificar, documentar, diseñar, ejecutar, medir y controlar los procesos que realiza una organización, con la finalidad de lograr los objetivos de negocio que se encuentran alineados con su estrategia de negocio [26]. BPM se basa en sus Procesos de Negocio y su ciclo de vida, abarcando el apoyo creciente de Tecnologías de la Información (TI) con el objetivo de mejorar, innovar y gestionar los procesos de principio a fin, que determinan los resultados de negocio, crean valor para el cliente y posibilitan el logro de los objetivos con mayor agilidad [77].

Un Proceso de Negocio consiste en un conjunto de actividades que se realizan coordinadas en un entorno organizacional y técnico con el fin de alcanzar una meta del negocio [77].

La Gestión por Procesos de Negocio comienza una primera fase (Análisis y Diseño) de identificación para poder representarlos con modelos y centralizar toda la información [77]. Esta descripción es utilizada en una fase posterior (Configuración) para el desarrollo de una solución informática que dé soporte a dichos procesos. Generalmente se utilizan Sistemas de Gestión de Procesos de Negocio (Business Process Management Systems, BPMS) que permiten modelar, ejecutar y evaluar Procesos de Negocio de forma unificada.

En ambas fases se realizan actividades de verificación de Procesos de Negocio. La verificación se realiza principalmente durante la fase de Análisis y Diseño previo a la implementación y ejecución de los procesos en la organización. Además, la verificación puede ser realizada tanto utilizando métodos formales como testing [77]. La importancia de verificar modelos es poder determinar de forma anticipada si un proceso se va a comportar de forma deseable. Otras propiedades no pueden ser verificadas de forma anticipada y son evaluadas simulando la ejecución del modelo. Existen diversas formas

para verificar propiedades deseables en los modelos dependiendo del contexto en que se encuentren.

En las últimas décadas, se ha incrementado el uso de los Procesos de Negocio, por personal técnico y del negocio, lo que llevó al desarrollo de una gran diversidad de representaciones de modelos en distintos contextos. Esto ocasionó que la verificación se realice de forma descentralizada, enfocándose en problemas y modelos específicos. Ésta problemática intentó solucionarse con herramientas de verificación, pero hasta el momento las que existen en general no son completas ya que asumen modelos muy simples que no se adaptan a los procesos de la realidad. Por otro lado, la falta de un estado del arte sobre la verificación de los procesos dificulta el entendimiento del problema y el estudio de nuevas técnicas.

Este proyecto se enmarca en la línea de investigación del grupo COAL entorno a la Gestión y Tecnologías de Procesos de Negocio, buscando generar conocimiento acerca de la verificación de Procesos de Negocio, definiendo el concepto de verificación en este contexto e identificando los métodos, técnicas, lenguajes y tecnologías utilizados para ello.

Los objetivos de este proyecto son:

- Estudio del marco teórico de Procesos de Negocio, enfocándose en la verificación de los mismos.
- Identificar métodos, técnicas, lenguajes y tecnologías para la verificación de Procesos de Negocio a través de la realización de una revisión sistemática de la literatura [34].
- Desarrollo de un entorno integrador de herramientas existentes que permita distintas verificaciones sobre los de Procesos de Negocio.
- Aplicar en un caso real las técnicas, métodos, lenguajes y tecnologías relevados, utilizando la herramienta desarrollada.

Los siguientes capítulos del documento están organizados de la siguiente manera:

En el Capítulo 2 se detalla el estado del arte y conceptos relevantes necesarios para comprender el proyecto. En el Capítulo 3 se muestra los aspectos principales del proceso de la revisión sistemática de la literatura. El Capítulo 4 especifica el desarrollo de un Plug-in en la herramienta Eclipse. En el Capítulo 5 detallamos los casos de prueba para la verificación del prototipo. El Capítulo 6 contiene el Caso de Estudio realizado. En el Capítulo 7 se brindan las conclusiones del proyecto y posibles trabajos a futuro.

Capítulo 2

Contexto

Presentaremos el contexto en el que se desarrolla el proyecto. En la Sección 2.1 se introduce el concepto de Procesos de Negocio y varias definiciones como punto de partida para comprender las ideas desarrolladas en el proyecto. En la Sección 2.2 definimos los lenguajes de especificación de los Procesos de Negocio utilizados. Por último, en la Sección 2.3 realizamos una breve presentación a la verificación de Procesos de Negocio.

2.1. Introducción a Procesos de Negocio

Un Proceso de Negocio (Business Process) consiste en un conjunto de actividades que se realizan coordinadas en entorno organizacional y técnico. Estas actividades colectivamente buscan alcanzar una meta del negocio. Cada Proceso de Negocio es ejecutado por una única organización, pero puede interactuar con Procesos de Negocio llevados a cabo por otras organizaciones (procesos colaborativos) [69]. La Gestión de Procesos de Negocio (Business Process Management, BPM) refiere a las actividades que realizan las organizaciones para optimizar o adaptar sus Procesos de Negocio a las nuevas necesidades organizacionales [7]. En las últimas décadas se ha otorgado mayor atención a BPM [65], combinando tecnologías y herramientas que soportan las etapas del ciclo de vida, guían las actividades y tecnologías relevantes para su gestión.

El ciclo de vida de los Procesos de Negocio (Figura 2.1) se puede definir en base a cuatro etapas [77]. En la etapa de Análisis y Diseño los procesos son modelados utilizando diversos lenguajes de notación como Business Process Model and Notation (BPMN 2.0) [54].

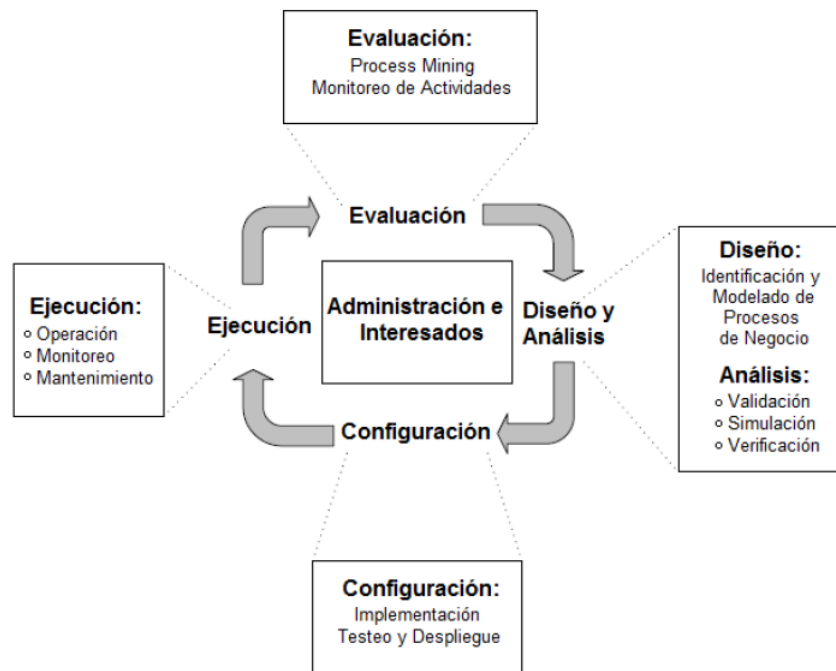


Figura 2.1: Ciclo de vida de un Proceso de Negocio

La representación explícita de los procesos permite un mejor entendimiento por todos los involucrados de la organización, así como permite guiar el desarrollo de software de soporte a la operativa. Esto ocurre en la etapa de Configuración en la cual los procesos son implementados a los efectos de ser ejecutados en Sistemas de Gestión de Procesos de Negocio (Business Process Management Systems, BPMS). Durante la etapa de Ejecución, los eventos relacionados a la ejecución de los procesos son registrados para su posterior análisis, el cual se realiza en la etapa de Evaluación utilizando diversas técnicas como Minería de Procesos (Process Mining) [71]. De esta forma, se brinda información relevante al área del negocio que identifica oportunidades de mejora que retroalimenta el ciclo de vida.

A lo largo del ciclo de vida los procesos deben ser verificados, es decir, se debe determinar si se comportan de la forma que se espera. La verificación de Procesos de Negocio se realiza en dos etapas. En la etapa de Análisis y Diseño se pueden verificar los modelos de procesos para identificar potenciales problemas antes de que sean puestos en ejecución y en consecuencia prevenirlos. Además, dado que estos modelos son utilizados como base para la ejecución en un BPMS, en la etapa de Configuración se simula la ejecución para analizar aspectos que no son identificados en la etapa anterior, aumentando así la confiabilidad de los resultados obtenidos.

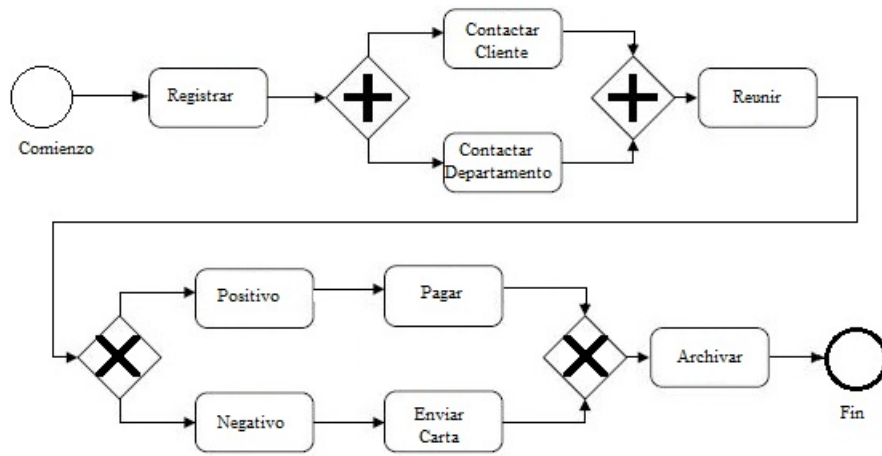


Figura 2.2: Ejemplo de un Proceso de Negocio (Manejo Reclamos) [77]

En la Figura 2.2 podemos observar una forma de modelar un proceso de manejo de reclamos, utilizando una versión simplificada de BPMN. El inicio y final del proceso se denominan *eventos* y se representan en el diagrama con círculos. Inicialmente un reclamo es registrado, luego el cliente que realizó el reclamo y el departamento afectado por el reclamo son contactados. Al cliente se le pide más información, mientras que el departamento es informado del reclamo y se le puede pedir su reacción inicial. Estas dos tareas pueden ser realizarse en paralelo, que es en simultáneo o en cualquier orden. Después de esto la información es reunida y la decisión tomada. Dependiendo de la decisión, se hará un pago de compensación o una carta será enviada. Finalmente, el reclamo es archivado. Los elementos como Archivar, Reunir, Registrar, se llaman *actividades* y son el trabajo que se realiza en la organización. Los rombos de la figura que controlan la convergencia o divergencia del flujo de actividades se llaman *compuertas de decisión*. Por último, las flechas son las que indican el flujo del proceso.

Nos basaremos en la visión de un ciclo de vida en 4 fases basado en [77]. Como se observa en la Figura 2.1, las fases son: Diseño y Análisis, Configuración, Ejecución y Evaluación. Además, en la estructura del ciclo vemos las dependencias entre las fases y las actividades que se realizan en cada una. En el modelo planteado, las fases son iterativas y buscan la evolución continua del sistema.

La primera etapa del ciclo de vida de un Proceso de Negocio es la de Diseño y Análisis, en la que se realizan estudios a los Procesos de Negocio y su entorno técnico y organizacional para poder representarlos con modelos. Lo que se busca es formalizar a partir de un modelo de Proceso de Negocio toda la información obtenida, de manera de intentar mejorar los procesos.

Luego de que se tiene un diseño inicial, debe ser validado y verificado. Las técnicas de simulación son una buena herramienta para llevar a cabo la validación. Existen varias herramientas para identificar, revisar, validar y especificar modelos de Proceso de Negocio [15].

En la fase de Configuración se elige la configuración con la cual se va a implementar el modelo diseñado. Se puede realizar de varias maneras, por ejemplo, con procedimientos y políticas que pueden ser gestionadas sin soporte de software, con algún programa de soporte para Procesos de Negocio o con sistemas integrados para BPM (BPMS). Luego se verifica con técnicas tradicionales como la verificación funcional del comportamiento o se realizan pruebas de integración y performance intentando detectar posibles problemas de ejecución.

En la fase de Ejecución se ponen en marcha las instancias de Proceso de Negocio para alcanzar las metas de la organización. Durante esta fase se toman datos sobre el funcionamiento del sistema, generalmente en archivos *Log* donde se guarda la información sobre cómo se ejecutaron los eventos en el sistema. Esto es de fundamental interés para la siguiente fase.

La fase de Evaluación parte de la información disponible en los *Log*, con técnicas como “Process Mining” y evalúa el desempeño de los Procesos de Negocio y sus implementaciones para intentar mejorarlos.

2.2. Lenguajes de especificación de Procesos de Negocio

Cada vez son más los actores involucrados en la creación y utilización de Procesos de Negocio, por lo que es necesario distintos tipos de modelos para que sean comprensibles por todos los implicados. Por otro lado, desde el punto de vista de la verificación, la notación es muy importante ya que permite el uso de técnicas más o menos directas para ello. Los modelos *no formales* son muy intuitivos de comprender pero carecen de una semántica formal definida. Los *formales* son más complejos con base matemática sólida que permite validar el modelo de forma semántica y sintáctica por lo que se necesitan conocimientos especializados. Por último, los *semi-formales* son un híbrido con semánticas parcialmente definidas.

A continuación presentaremos una introducción a BPMN 2.0 y Redes de Petri que son los lenguajes de especificación que más nos interesa conocer para la realización de este proyecto. En particular Redes de Petri porque varias de las propiedades que luego verificaremos, se especifican en este lenguaje.

2.2.1. BPMN 2.0

El estándar BPMN fue desarrollado por OMG (Object Management Group) con el objetivo de proveer una notación que sea fácilmente comprensible por todos los usuarios involucrados en el negocio. Se podría decir que BPMN es como un puente estandarizado entre el diseño de Procesos de Negocio y el proceso de implementación [54].

Como objetivo secundario, interesa asegurar que los lenguajes XML diseñados para la ejecución de Procesos de Negocio, como por ejemplo WSBPEL (Web Service Business Process Execution Language) puedan visualizarse con una notación orientada al negocio.

Esta especificación representa la fusión de las mejores prácticas del modelado de Procesos de Negocio para la definición de la notación y semántica de diagramas de colaboración, de procesos y coreografías. La idea de BPMN es estandarizar la notación y los modelos de los Procesos de Negocio. De esta forma, se proporciona un medio sencillo de comunicar información de proceso a otros usuarios del negocio, como clientes y proveedores.

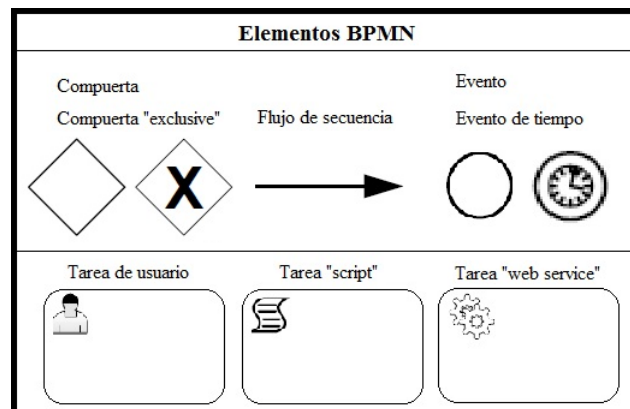


Figura 2.3: Elementos BPMN 2.0

En la Figura 2.3 mostramos los principales elementos de la notación que describimos a continuación. Un *evento* es usado para modelar algo que pasa durante el ciclo de vida del proceso. Son visualizados mediante un círculo (que varía según la definición) y existen dos categorías principales; *catching event* y *throwing event*. La definición del evento define la semántica del mismo. Algunos ejemplos son *start event*, *error event*, *timer event*.

Un *flujo de secuencia* es un conector entre dos elementos de un proceso. Luego de que un elemento es visitado en la ejecución de un proceso, todos los flujos de secuencia salientes van a ser seguidos. Esto significa que en

BPMN 2.0 si hay dos flujos de secuencia salientes se van a crear dos rutas de ejecución paralelas. La forma de representarlo es con una flecha que apunta del elemento origen al elemento objetivo.

Las *compuertas* son usadas para controlar el flujo de ejecución. Se visualizan de forma de diamante con un icono dentro que determina el tipo de compuerta. Las diferentes compuertas son: *exclusive gateway*, *parallel gateway*, *inclusive gateway*, *event-based gateway*. El *exclusive gateway* (también llamado XOR) se utiliza para modelar una decisión en el proceso. Cuando la ejecución llega a la compuerta, todos los flujos de secuencia de la salida son evaluados en el orden que fueron definidos. El flujo cuya condición se determine como verdadera, será por donde continuará el proceso. Se representa con una equis negra dentro del rombo.

Una *tarea de usuario* se utiliza para modelar el trabajo que se realiza mediante un actor humano. Cuando la ejecución del proceso llega a la tarea de usuario, una nueva tarea es creada en la lista de tareas del usuario asignado a dicha tarea. Una tarea se representa con un rectángulo redondeado y si es de usuario se le agrega un icono chico con forma de persona en la esquina superior izquierda.

El *script task* es una de las actividades que se realiza de forma automática. En el momento que la ejecución del proceso llegue a esta tarea el correspondiente *script*¹ es ejecutado. A diferencia de la tarea de usuario, esta se representa con un icono en forma de hoja en la parte superior izquierda. Otras tareas soportadas son: *web service task*, *email task*, *manual task*, entre otras [54].

El *web service task* se utiliza para sincronizar la invocación de un servicio web externo. Por otro lado, *email task* permite mandar e-mails para uno o varios destinatarios (incluyendo las opciones con copia, copia oculta entre otras funcionalidades de los mails tradicionales). Para conocer el estándar en su totalidad, ver referencia [19].

El modelado de los Procesos de Negocio son utilizados para comunicar una gran variedad de información entre diferentes actores. Los elementos estructurales de BPMN antes mencionados, permiten al usuario diferenciar las distintas secciones de un diagrama BPMN. Existen tres tipos básicos de sub-modelos; *Orquestaciones*, *coreografías* y *colaboraciones*.

Los procesos u orquestaciones, pueden ser privados o públicos. Los Procesos de Negocio privados generalmente llamados *workflow* o *BPM Processes*,

¹En programación, un script es un programa o una secuencia de instrucciones que son interpretadas o llevadas a cabo por otro programa

son internos a una organización específica mientras que los públicos representan interacciones entre Procesos de Negocio internos u otros procesos. Las colaboraciones son interacciones entre dos o más entidades del negocio. Usualmente contienen dos o más *pools*. Un *pool* se visualiza de forma rectangular y es la representación gráfica de un participante en una colaboración. También actúa como un *swimlane* y contenedor gráfico para separar un conjunto de actividades de otros pools.

Un *lane* es una sub-partición dentro de un proceso o pool y se extiende a lo largo del proceso. Se utilizan para organizar y categorizar actividades. Ver Figura 2.4

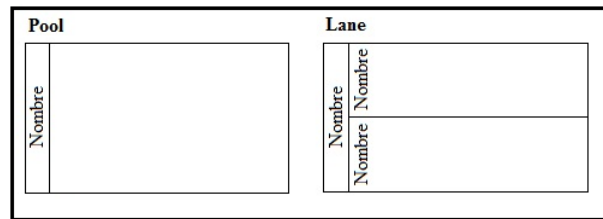


Figura 2.4: Ejemplos de Pool y Lane

2.2.2. Redes de Petri

Una Red de Petri es un modelo abstracto y formal de un flujo de información. Esta es una forma natural, simple y poderosa de describir y analizar el flujo de información y control en distintos tipos de sistemas, en particular sistemas que presentan actividades asincrónicas y concurrentes.

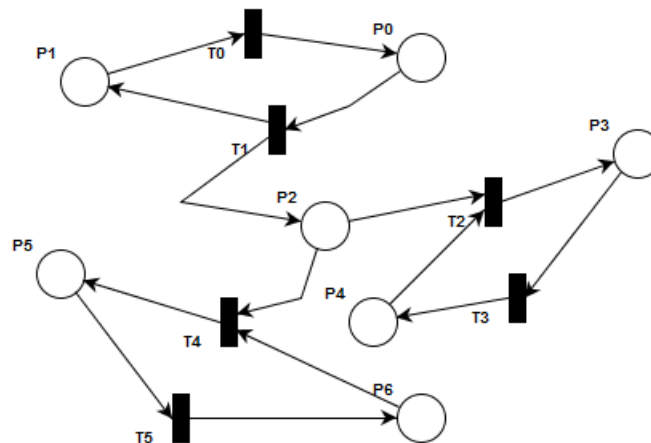


Figura 2.5: Red de Petri

La Figura 2.5 muestra una Red de Petri simple, el grafo tiene dos tipos de nodos: los círculos, llamados lugares (“places” en inglés) y las barras, llamadas transiciones (“transitions” en inglés). Estos nodos (lugares y transiciones), están conectados por arcos dirigidos desde los lugares hacia las transiciones y viceversa. Si un arco es dirigido desde el nodo i al nodo j , pudiendo ser desde un lugar a una transición o de una transición a un lugar, entonces i es una entrada de j y j es una salida de i . Para el caso particular de la Figura 2.5, por ejemplo el lugar $P1$ es una entrada de la transición $T2$, mientras que los lugares $P2$ y $P3$ son salidas de la transición $T2$.

Además de las propiedades estáticas representadas por el gráfico, las Redes de Petri tienen propiedades dinámicas que surgen en consecuencia de su ejecución. La ejecución de una Red de Petri es controlada por la posición y movimiento de marcadores, llamadas fichas (“tokens” en inglés), representados en la misma red. Las fichas están indicadas por puntos negros dentro de los círculos que representan los lugares de la red, una Red de Petri con fichas se denomina Red de Petri marcada.

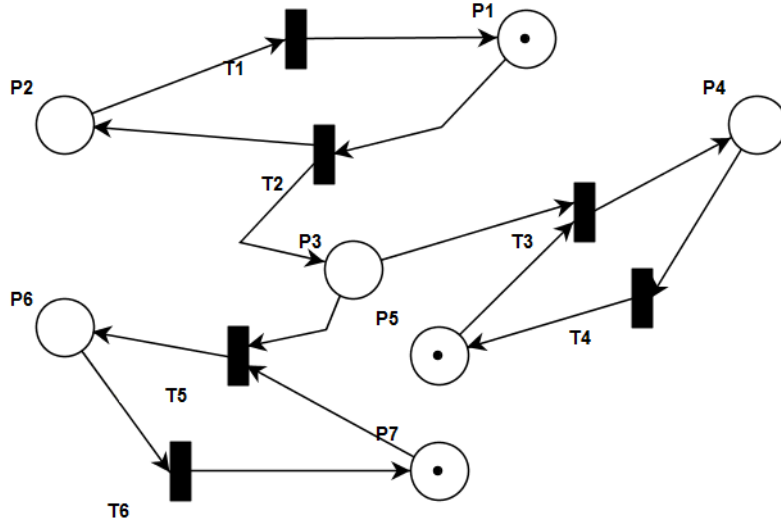


Figura 2.6: Red de Petri marcada

Las reglas de movimiento de las fichas son las siguientes: Las fichas se mueven por el disparo de las transiciones de la red. Una transición debe ser habilitada para poder dispararse (Una transición está habilitada cuando todos los lugares de entrada tienen una ficha en ellos). La transición se dispara removiendo las fichas que habilitan de sus lugares de entrada y generando nuevas fichas que son depositadas en los lugares de salida de la transición.

En la Red de Petri marcada de la Figura 2.6 por ejemplo, la transición $T2$ está habilitada desde que tiene una ficha es su lugar de entrada ($P1$). Por otro lado, la transición $T5$ no está habilitada ya que una de sus entradas ($P3$) no tiene una ficha. Si se dispara $T2$ se tiene como resultado la Red de Petri de la Figura 2.7. El disparo de la transición $T2$ remueve la ficha que habilita del lugar $P1$ y pone fichas en $P2$ y en $P3$, las dos salidas de $T2$ [58].

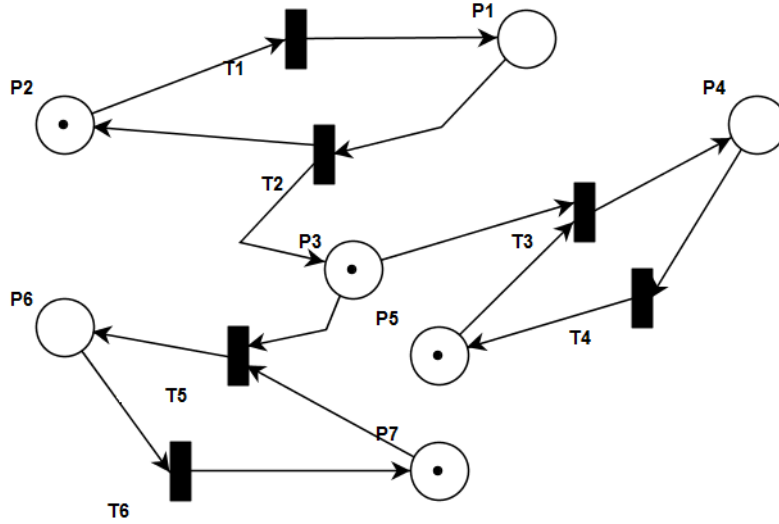


Figura 2.7: Red de Petri resultante del disparo de T2

2.2.3. UML (Diagramas de actividad)

Los diagramas de actividad UML son representaciones gráficas que muestran el flujo de control y el flujo de datos. Se basa en actividades y acciones, con soporte de iteración y concurrencia. Típicamente se usan para flujos de trabajo de Procesos de Negocio, reglas lógicas de negocio, entre otras [55].

2.2.4. And/Or graph

Los grafos AND/OR son usados como un modelo para la resolución de problemas mediante la reducción del mismo. Con la reducción del problema se crean sub-problemas que pueden ser resueltos de forma independiente. Los nodos del grafo representan estados u objetivos y sus sucesores son “ramas” llamadas “And” u “Or”. Los sucesores “And” son sub-objetivos que deben ser alcanzados para satisfacer el objetivo de sus padres, mientras que los sucesores “Or” son sub-objetivos alternativos, cualquiera podría satisfacer el objetivo de sus padres [40].

2.2.5. EPC

Una cadena de Procesos por evento es una técnica para el modelado de Procesos de Negocio. Su enfoque se relaciona más con documentación semi-formal que con la especificación formal del proceso. Se basa en eventos que describen como funciona un proceso o en el estado que resulta. Siempre tiene un estado inicial y uno final [46].

2.2.6. XPDL

XPDL (XML Process Definition Language) es un lenguaje para la definición de un Flujo de trabajo. XPDL es un formato de archivo basado en XML que puede ser usado para intercambiar modelos de Procesos de Negocio entre distintas herramientas. Es un formato de archivo que representa el “dibujo” de la definición del proceso. Los nodos y las líneas tienen atributos que pueden especificar información ejecutable tales como roles, descripción de actividades, temporizadores, llamadas a web services, entre otras [78].

2.2.7. WS-BPEL

BPEL es un lenguaje ejecutable con sus variables y operaciones. Las operaciones permiten enviar y recibir mensajes SOAP y tiene un gran soporte para XML y transformaciones XML. Además, permite de manera fácil invocar múltiples servicios web al mismo tiempo y sincronizar los resultados. BPEL no tiene soporte gráfico; es decir, no especifica cómo deben ser los diagramas interpretativos de los procesos que define. El objetivo de BPEL es ofrecer una forma de orquestar servicios web, la secuencia de interacciones subyacente y el flujo de datos punto a punto. Es la abreviación de WS-BPEL. Antes de su estandarización se denominaba BPEL4WS [53].

2.2.8. YAWL

Yet Another Workflow Language, es un lenguaje que extiende Redes de Petri (workflow nets) para dar soporte a patrones de workflow. Este lenguaje está soportado por un sistema de software que incluye un motor de ejecución y un editor gráfico. Para que la especificación del modelo esté disponible en el entorno de ejecución, es necesario especificarlo en el editor, el cual guarda el modelo en formato XML para que sea interpretado por el motor de ejecución [29].

2.2.9. C-YAWL

El editor YAWL soporta la definición de modelos configurables YAWL. Un modelo C-YAWL es un modelo YAWL donde algunas tareas son anotadas como configurables. Esta tarea configurable representa la parte variable del modelo del proceso [25].

2.3. Verificación de Procesos de Negocio

La verificación de Procesos de Negocio es el acto de determinar si un modelo de proceso es correcto con respecto a un conjunto de propiedades formales. Con frecuencia la verificación es automatizada por herramientas conocidas como *analizadores* o *model checkers*. Varios objetivos para verificar son presentados en la literatura de los Procesos de Negocio, los cuales pasaremos a desarrollar.

Un objetivo es verificar las propiedades básicas como *reachability* (alcanzabilidad) y *termination* (terminación). La propiedad *reachability* de una actividad de negocio requiere que exista un camino de ejecución que conduzca a llevar a cabo dicha actividad, comenzando desde las actividades iniciales. La propiedad de *termination*, requiere que todas las trazas de ejecución terminen. *Soundness*, una propiedad originalmente propuesta en el área de verificación Redes de Petri, en el contexto de Procesos de Negocio es conocida como la combinación de las dos propiedades anteriormente mencionadas, más una tercera: La ausencia de actividades relacionadas con el proceso que se mantengan en ejecución luego de que este ha finalizado (*por ejemplo, Proper Completion*). Evitar la puesta en producción (deployment) de Procesos de Negocio erróneos, que no cumplen con estas propiedades, es algo obviamente ventajoso. “*Procesos de Negocio erróneamente diseñados pueden llevar a flujos de trabajo fallidos, errores de ejecución y clientes desconformes*” [6].

Otro objetivo de la verificación de Procesos de Negocio es la *completitud* (Compliance). La *completitud* chequea si un modelo de proceso cumple con las especificaciones, que en este caso, puede ser otro proceso de modelo o un conjunto de reglas, como estándares y leyes. Cuando se verifica la completitud, las reglas son generalmente especificadas utilizando lógica formal, sobre las entidades del modelo del Proceso de Negocio. En otros casos son especificadas informalmente. Por ejemplo, las regulaciones pueden especificar que antes de procesar una transferencia bancaria, un banco debe identificar si existe alguna sanción contra las partes involucradas.

El objetivo final de la verificación de Procesos de Negocio está relacionado con procesos que incluyen múltiples partes, como colaboraciones de Procesos de Negocio. Incluye, por ejemplo, la compatibilidad entre procesos o Lanes [27].

Capítulo 3

Revisión Sistemática

El objetivo de esta revisión es presentar los resultados sobre verificación de Procesos de Negocio en las diferentes etapas de su ciclo de vida. Se basa en los lineamientos de revisiones sistemáticas de la literatura propuestos en [34]. De esta forma, se obtienen ciertos niveles de confianza en los resultados obtenidos y permite que la revisión pueda ser replicada. Los resultados son clasificados y analizados desde diferentes perspectivas a los efectos de favorecer la comprensión del tema. Además, se busca abarcar verificación en todas sus dimensiones, no sólo restringida a verificación formal.

Existen revisiones previas sobre verificación de Procesos de Negocio [27, 49, 80], aunque se centran en las técnicas de verificación formal, es decir, en el uso de formalismos matemáticos para abordar la verificación. Estas técnicas permiten un tratamiento más profundo del problema pero al mismo tiempo suelen resultar costosas de aplicar. En contraposición, se puede pensar en la aplicación de técnicas no formales (testing) que no permiten demostrar la corrección absoluta pero sí obtener entornos de confianza con un menor esfuerzo que con la aplicación de las técnicas formales.

El resto del capítulo está organizado de la siguiente forma. En la Sección “El proceso de la revisión” se presenta metodología utilizada para realizar la revisión de la literatura y algunos resultados estadísticos obtenidos. Luego, en la Sección “Clasificación de los resultados” se presentan los principales resultados del estudio a través de una clasificación de los artículos que permite un análisis desde diferentes perspectivas. En la Sección “¿Qué verificamos?” se profundiza el análisis de la dimensión correspondiente a los problemas que ataca la verificación y en la Sección “¿Cómo se realiza la verificación?” en la dimensión correspondiente a las técnicas utilizadas para realizar dicha verificación. En la Sección “Análisis de los resultados” se realiza un análisis de resultados más global.

3.1. El proceso de la revisión

La revisión fue realizada siguiendo los lineamientos de revisiones sistemáticas de la literatura planteados en [34]. En la etapa de planificación se define la pregunta de investigación para el estado del arte, las palabras claves y las cadenas de búsqueda que serán utilizadas para recuperar los trabajos relevantes de las fuentes seleccionadas. También los criterios de inclusión y exclusión que permitirán seleccionar los trabajos relevantes y los estudios primarios para extraer los datos asociados. En el desarrollo de la revisión las cadenas de búsqueda definidas son ejecutadas en las fuentes seleccionadas. De los trabajos obtenidos, aquellos que no cumplen el criterio de inclusión son descartados, determinando así el subconjunto de trabajos relevantes. Luego de descartar trabajos repetidos, los relevantes son leídos en profundidad y si cumplen el criterio de inclusión son seleccionados como estudios primarios. Con ellos se realiza la extracción de datos que permite realizar el análisis de los resultados.

A continuación detallamos las decisiones tomadas en cada etapa del proceso de revisión.

3.1.1. Planificación de la revisión

El objetivo de la revisión es conocer en amplitud los trabajos existentes en torno a la verificación de Procesos de Negocio. En este contexto, las preguntas de investigación planteadas fueron las siguientes:

Pregunta primaria: *¿De qué forma se ha abordado la verificación de Procesos de Negocio en las distintas fases de su ciclo de vida?*

Preguntas secundarias:

- *¿Qué propiedades interesa verificar y en qué fase?*
- *¿Qué técnicas y herramientas han sido utilizadas?*

A partir de estas preguntas identificamos las palabras clave para generar la siguiente cadena de búsqueda:

(“BPM” OR “Business Process” OR “Workflow”) AND (“Verification” OR “Testing” OR “Verify” OR “Validation” OR “Test” OR “Validate”)

La búsqueda fue realizada en inglés para maximizar los resultados obtenidos, aunque también se buscó en español. Para la definición de palabras clave, se tuvieron en cuenta ciertos términos relacionados, los cuales muchas veces se usan de forma indistinta, por ejemplo *workflow* en lugar de *business process* y *testing o validation* en lugar de *verification*.

Las fuentes confiables para la búsqueda fueron las siguientes: SCOPUS, Science Direct, ACM, IEEE, Springer. Para cada fuente se debió adaptar la cadena de búsqueda acorde al lenguaje propio que utilizan para definirlas.

3.1.2. Selección de estudios y extracción de la información

Los criterios de inclusión/exclusión para filtrado de los resultados fueron los siguientes.

- **Criterios de inclusión:** Textos publicados entre los años 2003 y julio del 2013.
- **Condición necesaria:** Tiene que estar relacionado con BPM y verificación.
- **Criterios de exclusión:**
 - No está relacionado con BPM.
 - No está relacionado con verificación
 - No se ha publicado el artículo todavía

La Tabla 3.1 muestra la evolución del proceso de selección, con las columnas de Encontrados Totales (ET), los cuales incluyen los resultados repetidos, Relevantes (R) con resultados repetidos filtrados, el total de los primarios (P) (artículos duplicados eliminados) y finalmente los porcentajes por fuente (%).

Cuadro 3.1: Cantidad de artículos según fuente

Fuente	ET	R	P	%
SCOPUS	392	33	14	35
Science Direct	155	21	13	32.5
ACM	487	10	0	0
IEEE	422	5	2	5
Springer	188	18	11	27.5
Total	1644	87	40	100

La búsqueda inicial dio como resultado 1644 artículos, de los cuales 87 fueron seleccionados como relevantes luego de leer el título, el resumen y las palabras clave. De éstos últimos, 40 fueron seleccionados como estudios primarios.

De los estudios primarios se extrajo su fuente, el año en que se publicó el documento, la clasificación del artículo, el autor(es), afiliación (organización y país) y su resumen. Durante este proceso se fueron identificando diversas características de la verificación de los Procesos de Negocio por las cuales se fueron clasificando los artículos.

La Figura 3.1 muestra los resultados de la agrupación de artículos por año de publicación.

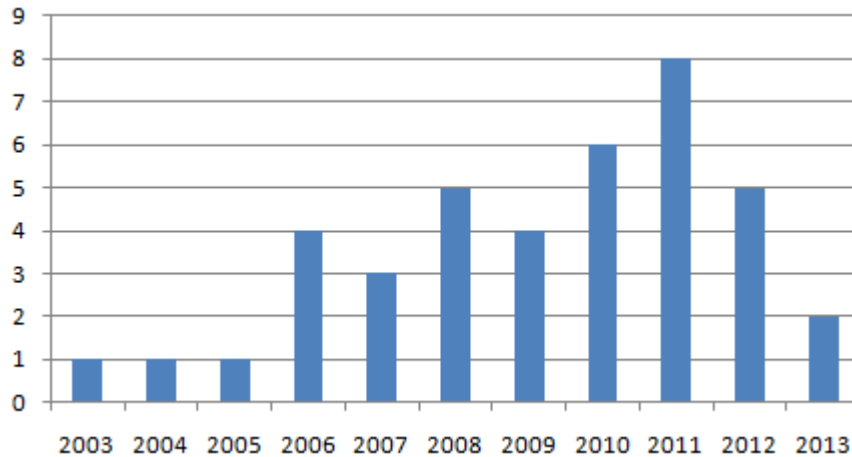


Figura 3.1: Artículos por año de publicación

Se aprecia una clara tendencia en el desarrollo e investigación en esta área hasta el año 2011. A partir del 2012 la cantidad de artículos comienza a disminuir. Cabe aclarar que los resultados de la búsqueda se extienden hasta julio de 2013.

3.2. Clasificación de los resultados

Los artículos encontrados se clasificaron acorde a los criterios que se definen a continuación. Estos criterios permite una visión del problema desde diferentes perspectivas que favorece su comprensión.

3.2.1. Tipo de proceso

Esta dimensión clasifica los trabajos según el tipo de Proceso de Negocio sobre el que se realiza la verificación: Orquestación o Colaborativo (excluyente), proceso Grid o proceso Científico. Una orquestación se refiere a un Proceso de Negocio realizado por una sola organización sin colaboración con otros Procesos de Negocio, mientras que los colaborativos son varios Procesos de Negocio de diversas organizaciones colaborando entre sí [54,77]. Todo proceso entra en una de estas dos categorías y a su vez, pueden estar orientados a recolectar, generar y analizar grandes cantidades de datos (Científico, [31]) o puede ser visto como una colección de tareas que son procesadas en recursos distribuidos, en un orden bien definido para lograr un objetivo específico (Grid, [9]).

Los resultados muestran una clara tendencia en el uso de orquestaciones ya que se utilizan en el 82.5 % de los artículos mientras que los colaborativos son un 17,5 %. Solo un artículo corresponde a un proceso de tipo Científico, tres del tipo Grid y todos ellos son a su vez orquestaciones.

3.2.2. Lenguajes de especificación

La notación utilizada para la especificación de los procesos es de gran importancia para la verificación, ya que permite el uso de técnicas más o menos directas para ello. Distinguimos tres grandes categorías de modelos: no formales que carecen de una semántica formal definida; formales con semánticas bien definidas, es decir, con base matemática que permite validar el modelo sintácticamente y semánticamente; semi-formales cuya base matemática no está totalmente definida. En la Figura 3.2 se muestra el porcentaje de trabajos acorde a los lenguajes utilizados pertenecientes a las categorías definidas anteriormente: informales (I), formales (F) y semi-formales (SF).

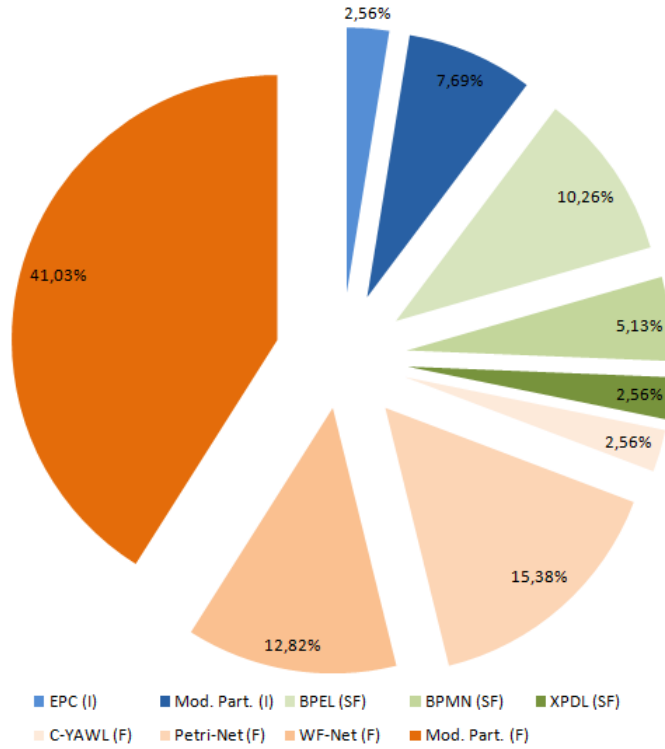


Figura 3.2: Clasificación de los modelos utilizados

Existe una clara tendencia a utilizar modelos con base formal para la verificación de Procesos de Negocio (por ejemplo, Redes de Petri [72] y YAWL [29]) ya que el 71,79 % de los artículos utilizan dichos modelos. El

uso de modelos informales (por ejemplo, EPC [46]), es decir, sin fundamentos matemáticos, es de 10,25 %. Por último, vemos que los semi-formales (por ejemplo, BPEL [53] y BPMN [54]) representan un 17,95 % del total.

3.2.3. Etapa en la que se realiza la verificación

Esta dimensión determina la etapa en la que se realiza la verificación de un Proceso de Negocio de acuerdo a su ciclo de vida [77]. Identificamos que sólo en dos de estas etapas se realiza verificación: en tiempo de Análisis y Diseño se realiza verificación estática del modelo y en tiempo de ejecución se realiza verificación dinámica de la ejecución. En la primera etapa se especifica totalmente el modelo de proceso utilizando diversos lenguajes, tanto formales como informales. En la segunda etapa el proceso se encuentra en ejecución en un BPMS, producto de la configuración del modelo generado en la primera etapa.

La gran mayoría de artículos se centra en la verificación durante la etapa de Análisis y Diseño, con un 85 % del total, mientras que en Ejecución se encuentran sólo un 5 % del total de artículos. El restante 10 % son artículos que tratan sobre la verificación en ambas etapas.

3.2.4. Herramientas utilizadas

Es importante conocer si existe disponibilidad de herramientas para llevar adelante la verificación de las propiedades de interés. Esta dimensión determina si los artículos desarrollan o especifican una herramienta para llevar adelante la verificación o no. El 72.5 % del total de artículos no especifican herramientas sino que se centran en un tratamiento abstracto del problema, en tanto que el restante 27.5 % si lo hace.

3.2.5. Propiedades a verificar

Uno de los puntos más interesantes es analizar qué propiedades son verificadas en un Proceso de Negocio. Esta dimensión contempla este aspecto identificando cinco categorías son analizadas en detalle en su propia Sección, la 3.4. Las propiedades de Grafos son las que están relacionadas con la representación de un proceso como un grafo, por otro lado definimos también las relacionadas con los recursos utilizados en los Procesos de Negocio. Además definimos una categoría que engloba las propiedades relacionadas con los procesos colaborativos, otra con las propiedades temporales que son las que están relacionadas con el avance del tiempo y por último una categoría “otros” para clasificar propuestas de propiedades diversas.

En la Figura 3.3 se muestra la distribución de propiedades de interés en los diferentes artículos acorde a las categorías definidas anteriormente: Grafos (G), Recursos (R), Temporales (T), Colaboración (C), Otros (O).

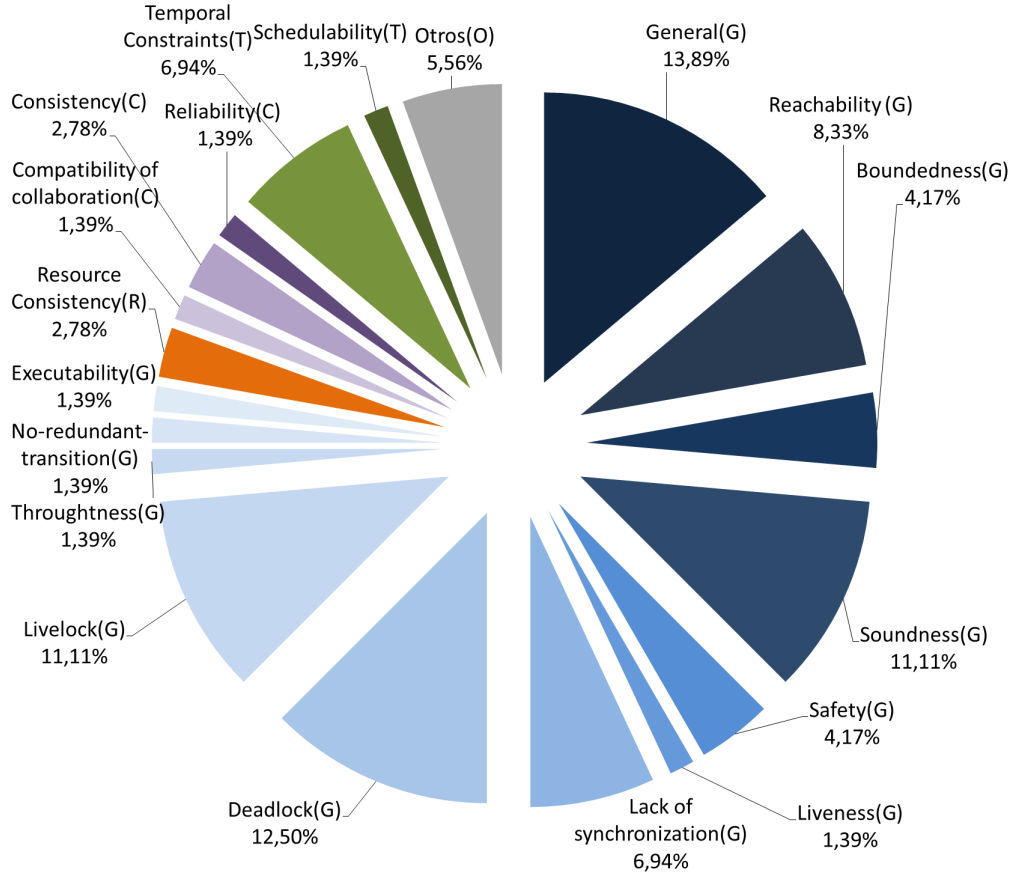


Figura 3.3: Características que se verifican

Como se puede observar, el 77,78 % de las propiedades que se verifican son de grafos, 2,78 % de recursos, 5,56 % son de colaboración, 8,33 % temporales y por último otros con un 5,56 %.

3.2.6. Técnicas de verificación

La forma en que se verifican los Procesos de Negocio tiene igual grado de importancia que las propiedades que se verifican, por lo que también será analizado en profundidad en una Sección propia, la 3.5. La clasificación fue tomada de [8] donde se aplica a enfoques de verificación de transformaciones de modelos.

Model checking consiste en el uso de una representación matemática de un sistema y de las propiedades a ser analizadas y de su verificación mediante la exploración exhaustiva del modelo matemático. Deducción lógica se basa en el uso de una representación matemática de un sistema y de las

propiedades a ser analizadas, así como del uso de una lógica en el dominio semántico de la representación, que permite el razonamiento lógico. La categoría por construcción consiste en la verificación de una propiedad al utilizar un lenguaje que satisfaga por defecto dicha propiedad, o que provea una guía para la creación de un modelo que lo haga. La verificación mediante análisis estático se realiza mediante el análisis de la estructura del modelo sin necesidad de ejecutar el mismo ni de razonar lógicamente sobre él. Por último, Testing/Simulación reside en la construcción de estrategias de pruebas para una propiedad, incluyendo la posterior ejecución del sistema (o de partes de éste) de acuerdo con dichas estrategias. Aunque la realización de pruebas se considera una estrategia de validación, pueden ser usadas con el objetivo de verificar propiedades con ciertos márgenes de error.

En la Figura 3.4 se muestra la distribución de artículos de acuerdo a las categorías definidas anteriormente. Un artículo puede utilizar varias técnicas de verificación para diferentes propiedades.

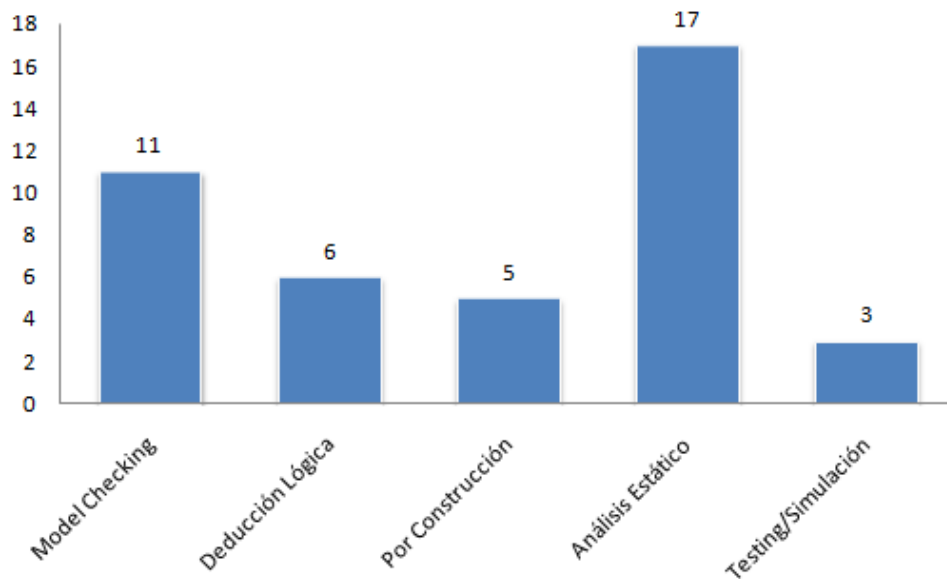


Figura 3.4: Técnicas de verificación

El Análisis estático es la técnica más usada en 17 artículos (42,5 %), luego el siguiente más usado es Model checking con un 27,5 %. Deducción lógica y Por construcción son utilizadas en 15 % y 12,5 % respectivamente. Testing/simulación no es muy usado ya que solo aparece en el 7,5 % de los artículos.

3.2.7. Resumen de la clasificación

La clasificación de los artículos desde diferentes perspectivas aporta una visión más global de la problemática, simplificando así su comprensión. Esta visión global se resume en la Figura 3.5 y Figura 3.6 donde se presenta un resumen de la clasificación de los artículos.

3.3. ¿Qué verificamos?

En esta Sección se presenta un análisis más detallado de las propiedades de interés a verificar en un Proceso de Negocio en cualquiera de las fases del mismo, independientemente de su representación y técnica de verificación utilizada.

3.3.1. Propiedades de grafos

Relacionadas a la representación de un proceso como un grafo o representación formal derivada, por ejemplo una Red de Petri.

Reachability (Alcanzabilidad) es la base fundamental para estudiar las propiedades dinámicas de cualquier sistema de Redes de Petri. Un marcado M_n (ubicación de un conjunto de fichas en determinados lugares de una Red de Petri) se dice que es alcanzable desde un marcado M_0 si existe una secuencia de disparos que tenga como consecuencia que la red pase del marcado M_0 al M_n . En algunas aplicaciones se podría estar interesado en los marcados de un subconjunto de lugares y no en el resto. Esto conduce a un problema de reachability de submarcado (Submarking reachability) que es el problema de encontrar si M_n' pertenece a los marcados alcanzables desde M_0 , donde M_n' es cualquier marcado que coincida con un subconjunto de lugares de un marcado M_n dado [51]. Artículos relacionados con *Reachability* [9, 33, 37, 45, 75, 76]. Una Red de Petri se dice que está k -bounded (k -acotada) o simplemente bounded (acotada) si el número de fichas en cada lugar no excede un número finito k para ningún marcado alcanzable desde el marcado inicial [51]. Artículos relacionados con *Boundedness* [30, 42, 59]. *Safety* (Protección), en el contexto de workflow, significa que el número de fichas en cada estado nunca va a ser mayor a uno, lo que significa que están acotados “bounded” en uno [70]. Artículos relacionados con *Safety* [37, 44, 59].

Hay ciertas características deseables que se espera que cada Proceso de Negocio presente. Primero, es importante saber que un proceso, cuando empieza, siempre pueda terminar (Propiedad “*option to complete*”). Segundo, ninguna otra tarea debería continuar ejecutándose para un proceso cuando el mismo ha finalizado (Propiedad “*Proper completion*”). Tercero, el proceso no debería tener tareas que nunca serán ejecutadas (Propiedad “*No dead transitions*”). La combinación de estas tres propiedades deseables en un Proceso

de Negocio es lo que se conoce como propiedad de *Soundness* [80]. Llevando el concepto a una red de workflow, esta es *Sound* si y solo si cumple con los siguientes tres requerimientos:

- Por cada ficha puesta en el lugar de comienzo, una y solo una ficha eventualmente llegará al lugar de fin.
- Cuando la ficha aparece en el lugar de fin, todos los otros lugares están vacíos.
- Para cada transición (tarea), es posible mover desde el estado inicial al estado en el que la transición está habilitada.

El primer requerimiento significa que cada caso será completado exitosamente en un período de tiempo. El segundo requerimiento quiere decir que una vez que el caso se completa, ninguna referencia a este quedará en el proceso. Si combinamos los primeros dos requerimientos, se puede llegar a la conclusión que existe solo un estado final: que es exactamente una ficha en el lugar de fin. El último requerimiento excluye las posibles tareas que nunca van a ser ejecutadas “tareas muertas”, lo que también quiere decir que cada tarea puede ser ejecutada [70]. Artículos relacionados con *Soundness* [33, 39, 47, 59, 64, 75, 79, 82].

Una Red de Petri se dice que es “*live*” (viva) para un marcado inicial M_0 dado, si sin importar que marcado sea alcanzado desde M_0 , es posible disparar cualquier transición de la red a partir de alguna secuencia de disparos. Esto quiere decir que una Red de Petri *live* garantiza operar libre de deadlocks, sin importar qué secuencia de disparos sea elegida. *Liveness* es una propiedad deseable para muchos sistemas, aunque es poco práctica ya que es muy costosa de verificar para algunos sistemas. Por eso en general esta condición es relajada a distintos niveles para poder verificarse [51]. Artículos relacionados con *Liveness* [59].

Un grafo de workflow presenta *Lack of synchronization* (falta de sincronización) cuando los nodos de tipo *or-join* tienen más de una entrada al mismo tiempo (cuando debería sincronizar caminos que se activan de forma excluyente) [1]. Artículos relacionados con *Lack of synchronization* [6, 11, 32, 39, 50].

Se denomina *Deadlock* cuando un proceso se bloquea antes de llegar a la condición de “terminado” [70]. Artículos relacionados con *Deadlock* [2, 6, 9, 11, 32, 33, 39, 50, 52]. Por otro lado *Livelock* es similar pero el proceso se atasca en un loop sin fin [70]. Artículos relacionados con *Livelock* [2, 6, 11, 32, 39, 45, 50, 52].

En el contexto de workflow logic, la propiedad *Throughness* indica que todas las secuencias de disparos que parten desde la marcación inicial llegan a una marcación de terminación [30]. Artículos relacionados con *Throughness* [30].

No-redundant-transition (Ausencia de transiciones redundantes) significa que partiendo desde un marcado inicial M_0 , para cualquier transición t de la red existe una secuencia de disparos que llega a un marcado M donde se puede disparar la transición t . En un modelo de workflow pueden existir transiciones que nunca van a poder ser disparadas, si estas se logran encontrar, la complejidad del “deploy” del proceso disminuye [30]. Artículos relacionados con *No-redundant-transition* [30].

En el contexto de Procesos de Negocio, una actividad cumple con la propiedad *Executable* si cumple con todas las precondiciones al momento de ser ejecutada. Si una tarea n no es ejecutable, entonces el proceso puede alcanzar un estado donde la tarea n está activa (tiene fichas en sus entradas) pero los requisitos para ejecutarse no están dados. Si el proceso está siendo ejecutado por un motor (no semántico) estándar, entonces la implementación de n va a ser ejecutada sin importar que no se satisfaga la condición, lo cual puede llevar a un comportamiento indefinido y errores [76]. Artículos relacionados con *Executability* [42, 76].

General: Artículos que tratan de verificación de propiedades particulares de la estructura y/o de afirmaciones definidas en los mismos [3, 12, 28, 36, 38, 44, 48, 57, 62, 76].

3.3.2. Recursos

Asociadas al uso de recursos en un Proceso de Negocio. En un workflow, las actividades tienen que acceder recursos durante su ejecución. Un recurso es definido como cualquier entidad requerida por una actividad para su ejecución, como un documento, una tabla de una base de datos, un dispositivo (como puede ser una impresora), una aplicación, hasta un actor. La verificación de recursos tiene como propósito verificar donde hay conflictos de recursos entre distintas actividades en un workflow, para determinar si existe o no consistencia [81]. Artículos relacionados con *Consistencia de recursos* [11, 81].

3.3.3. Colaboración

Asociadas al hecho de tener un proceso colaborativo o que consume servicios. *Compatibility of collaboration* (Compatibilidad de colaboración) es la habilidad de un conjunto de servicios interconectados de interactuar exitosamente. La compatibilidad puede dividirse en dos tipos, compatibilidad estructural y compatibilidad de comportamiento.

La compatibilidad estructural demanda que para cada mensaje que puede ser enviado, el socio “*partner*” de interacción correspondiente debe ser capaz de recibirlo. Además de esto, para cada mensaje que puede ser recibido el

socio correspondiente debe ser capaz de enviar tal mensaje. Esta es la noción de *compatibilidad estructural fuerte*, hay otras nociones más débiles para las cuales los requerimientos son menores.

Por otro lado, la compatibilidad de comportamiento considera las dependencias de comportamiento, como el flujo de control en los diferentes intercambios de mensajes dentro de una conversación [13,14]. Artículos relacionados con *Compatibility of collaboration* [13].

Servicios que interactúan son el centro de atención en los escenarios de integración de negocio a negocio “*business-to-business (B2B)*”. Modelos de interacción global especifican el comportamiento de interacción de cada servicio y sirve como base contractual para la colaboración. Esto tiene como consecuencia que la implementación de los servicios tenga que ser consistente con las especificaciones. El chequeo de *Consistency* (consistencia) asegura que el servicio implementado es compatible con otros servicios, en otras palabras, qué puede interactuar de forma exitosa con estos. Esto es importante para evitar deadlocks y garantizar la terminación correcta de la colaboración [14].

En definitiva, la consistencia puede ser definida en el contexto de la colaboración de negocios, como el estado y resultado del comportamiento de una organización en una colaboración de negocio, son lo que la organización y el colaborador esperan de la misma [66]. Artículos relacionados con *Consistency* [4,66].

Nos referimos a *Reliability* (Confiabilidad) en el contexto de negocios colaborativos cuando la colaboración se lleva a cabo como es planeada, cada participante se comporta como había acordado y la colaboración se completa en el tiempo y lugar correcto [66]. Artículos relacionados con *Reliability* [66].

3.3.4. Temporales

Relacionadas con el avance del tiempo en la ejecución de un proceso. Tratar con tiempo y restricciones de tiempo es crucial a la hora de diseñar y gestionar Procesos de Negocio. Como consecuencia de esto la gestión de tiempo debe ser una parte de la funcionalidad central proporcionada por un *workflow system* para controlar el ciclo de vida de los procesos. En el tiempo de construcción (*build-time*), cuando los esquemas de workflow son definidos y desarrollados, los modeladores de Proceso de Negocio necesitan medios para representar los aspectos relacionados con el tiempo (duración de actividades, restricciones de tiempo entre actividades) y chequear su factibilidad (por ejemplo restricciones de tiempo que no se contradigan una de otras) [24].

En tiempo de ejecución, cuando los workflows son instanciados y ejecutados, los gestores de procesos necesitan mecanismos proactivos para recibir notificaciones de posibles violaciones de restricciones. Si una restricción de tiempo es violada, el gestor debe ser capaz de disparar el manejo de la excepción para recuperar un estado consistente de la instancia del workflow. Los encargados de la reingeniería necesitan información sobre el consumo actual de tiempo de las ejecuciones de workflow para mejorar los procesos. Los controladores y administradores de calidad necesitan información sobre los tiempos de comienzo de cada actividad y duración de cada ejecución [24].

Existen distintos tipos de restricciones de tiempo, *Fixed-point constraint* (restricciones de tiempo fijo) son aquellas en las que un punto de tiempo específico del workflow debe ocurrir antes o después de un valor de tiempo absoluto predeterminado. Las restricciones de duración “*duration constraint*” son cuando se establece que la duración entre dos puntos de un workflow es menos que una constante de tiempo o un rango de duración. Una restricción interdependiente “*interdependent constraint*” es una restricción que es relativa a una restricción anterior, es una combinación de las otras dos restricciones anteriores [83]. Artículos relacionados con *Temporal Constraints* [5, 10, 11, 20, 41].

La verificación de la *Schedulability* (Planificabilidad) se refiere a que la planificación (schedule) de una tarea va a cumplir las restricciones de tiempo. Artículos relacionados con *Schedulability* [42].

Verificación de la propiedad *Correctness* en el contexto que trata el artículo [10, 43, 63, 73].

3.4. ¿Cómo se realiza la verificación?

En esta Sección se presenta un análisis más detallado de las técnicas y métodos utilizados para verificar las propiedades de interés definidas en la Sección anterior.

3.4.1. Model checking

Las técnicas de *model checking* consisten en la representación del problema como un modelo matemático y de la exploración exhaustiva de dicho modelo hasta encontrar una solución (o un contraejemplo).

En general, la forma de proceder es transformando la representación inicial del modelo (BPEL, BPNM, especificación informal) a una representación formal hecha en el lenguaje (BIR, SAL, PROMELA) del model checker utilizado (Bogor, sal-smc, SPIN, NuSMV), para que éste verifique si se cumplen

las aserciones especificadas por el usuario [4, 5, 12, 36, 38, 57, 62] o algunas ya predefinidas [37] con el objetivo de verificar algunas propiedades en particular [62]. Además de lo anterior también ataca el problema de explosión de estados (“State Explosion”) aplicando un algoritmo de recorte de workflows (Workflow Slicing) que permite generar una reducción cuasi equivalente al original de forma que se pueda verificar de manera más eficiente.

En otros casos, el procedimiento es al revés, donde se parte de un modelo formal verificable por un model checker y luego se genera la representación ejecutable como BPEL [3]. En [64] se presenta una técnica que puede formularse como un problema de model checking generalizado para verificar soundness en un modelo particular que ellos proponen. En [47] describe y valida un FCVA (Formal Compositional Verification Approach) utilizando esta técnica.

3.4.2. Deducción Lógica

Las técnicas de deducción lógica también representan el problema como un modelo matemático, pero a diferencia de las de *model checking*, éstas no realizan una recorrida exhaustiva sobre el modelo sino que se basan en el uso de una lógica que permite el razonamiento deductivo sobre el mismo.

La mayoría de los artículos que utilizan la deducción lógica como forma de verificación tienen enfoques similares. En [42] lo aplica con algoritmo de descomposición en “Workflow Nets” con restricciones de tiempo. En [6] aplican la lógica proposicional para la verificación de los flujos de trabajo y muestran que es capaz de detectar anomalías en los modelos. [44] propone un conjunto de reglas de pruebas del estilo de lógica de Hoare como un sistema de verificación axiomática para lenguajes del tipo BPEL que tienen características clave como estado de datos y manejo de errores. En [32] se realiza un algoritmo basado en los principios del Álgebra booleana. Otro punto de vista se plantea en [76] proponiendo una serie de proposiciones y teoremas mediante los cuales va demostrando las propiedades planteadas. En [10] proponen el análisis mediante “pi calculus” que permite una gestión flexible y el modelado de “Grid workflow”.

3.4.3. Por construcción

Las técnicas de esta categoría consisten en la verificación de una propiedad por el simple hecho de utilizar un lenguaje que permite satisfacer dicha propiedad usando sus construcciones.

En [42] realiza una guía para sintetizar y verificar un modelo para que cumpla con cierta propiedad. En [43] a partir de una librería de tareas, una pre condición del workflow y una postcondición de workflow, puede generar

automáticamente un workflow correcto con respecto a estas. En [20] plantea algoritmos para ir construyendo, actualizando y verificando una propiedad en el modelo y en caso de no cumplirla va modificando el modelo. [3] Verifica un modelo formal (BP-calculus) con aserciones a través de un model checker y a partir del modelo formal verificado se genera código BPEL utilizando un mapeo previamente definido. [2] Presenta un algoritmo que a partir de un modelo de procesos configurable (también llamados modelos de referencia, representado en C-YAWL) genera todas las configuraciones sin problemas de comportamiento a partir de un enfoque de “partner syntethis”.

3.4.4. Análisis estático

Las técnicas de análisis estático simplemente analizan la estructura del modelo sin necesidad de ejecutar el mismo ni de razonar lógicamente sobre él. Por ejemplo en [28] realiza un algoritmo que verifica seis restricciones del modelo definidas en el artículo. En [50] también presenta un algoritmo pero es de búsqueda para el análisis de flujo de información.

En [79] se toma un modelo representado en YAWL lo transforma a WF-net para luego verificar utilizando los algoritmos de la herramienta Woflan. Un enfoque similar se aplica en [82] donde se presenta un enfoque donde se puede transformar cualquier workflow independientemente de su representación a una Red de Petri equivalente para verificar utilizando las herramientas y teoría existente en Redes de Petri. Luego de transformado el modelo se verifica con Woflan tool.

[30] Verifica modelos de workflow logic representados con PNML con un enfoque basado en grafos de alcanzabilidad sincronizada (synchronized reachability graphs) y aplica reglas de reducción de modelos.

En [81] se presenta un análisis para la verificación y resolución de conflictos para workflows restringidos por recursos y sin duración determinada a partir de un modelado particular para este tipo de workflow calculando el momento más temprano de comienzo para cada actividad.

En [66] se presenta una plataforma (CoBTx-Net) para representar colaboraciones de Negocio y poder verificar que se cumplan las propiedades de consistencia y reliability a partir de teoremas definidos para esta plataforma que permite analizar estáticamente las colaboraciones. Otra forma de verificar la compatibilidad entre procesos colaborativos se presenta en [13] estudiando la compatibilidad semántica para evaluar la compatibilidad entre los comportamientos públicamente expuestos entre socios del negocio colaborativo.

En [33] se verifica aplicando reglas basadas en grafos para la detección de patrones erróneos de diseño, esto es beneficioso porque genera una respuesta directa durante la construcción del modelo y este puede estar incompleto al momento de la verificación. Una desventaja es que no puede verse como una validación completa de errores. También se utiliza reconocimiento de patrones en [39] donde el lenguaje visual de consultas BPMN-Q expresa patrones que están relacionados con problemas en los diagramas de BP.

En [48] se muestra un framework de verificación estática para la verificación de BP declarativos basado en la programación lógica abductiva (Abductive Logic Programming) donde se permite definir reglas y el framework las verifica. Se probó que este método escala mejor que otras técnicas como model checking.

En [73] trata de un algoritmo de verificación formal para “business process templates” que da como resultado si se cumplen las pre condiciones. En [59] se realiza una verificación paso a paso para detectar violaciones de una propiedad, inspeccionando en pequeñas porciones del modelo. Por otra parte, [9] plantea un algoritmo de cobertura realizando una recorrida sobre el grafo en “Grid workflow”.

[45] Propone un modelo de workflow basado en teoría de grafos y luego este se verifica la estructura con un algoritmo basado en el algoritmo de Warshall. Otro tipo de análisis es el que se plantea en [75] donde utiliza una herramienta para chequear que las transiciones estén bien formadas y la estructura de flujo de control esté bien estructurada.

3.4.5. Testing-simulación

Las técnicas de Testing/Simulación realizan la verificación de ciertos aspectos construyendo un conjunto de pruebas que, al ser ejecutadas, permiten obtener ciertos niveles de confianza en la satisfacción de las propiedades de interés.

En el artículo [41] se verifica dinámicamente en un punto específico de ejecución si las restricciones temporales de workflows concurrentes pueden ser satisfechas (En este caso seguir el flujo normal, sino tomar alguna medida). [63] Presenta una plataforma que permite un desarrollo basado en pruebas basadas en especificaciones de procesos de test (process test specifications) que son esqueletos para definir tests funcionales que son ejecutados en un entorno de test gestionado. Para estos test funcionales se generan configuraciones de simulación, componentes de test, aserciones y guiones de test para hacer un Proceso de Negocio testeable. En [4] el modelo en BPEL es ejecutado por un simulador y en este se verifica que se cumplan las aserciones escritas en lenguaje ALBERT.

3.5. Análisis de los resultados

Más allá de la identificación de trabajos y la clasificación de los mismos en las categorías que hemos definido en los capítulos anteriores, es conveniente realizar un estudio comparativo de los resultados.

La Figura 3.5 presenta la relación entre las categorías de técnicas de verificación y las categorías de propiedades en las que se aplican.

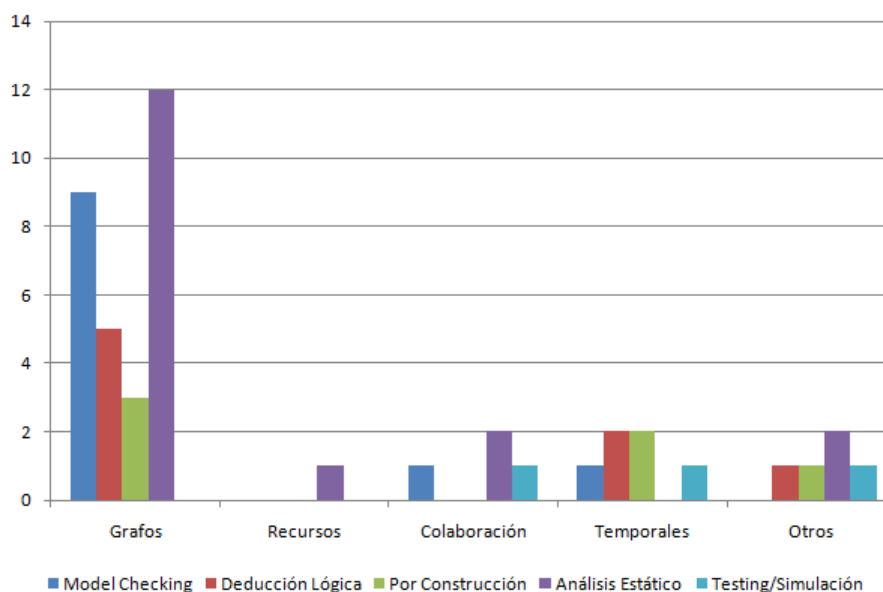


Figura 3.5: Técnicas aplicadas según propiedades

Constatamos la existencia de una gran cantidad de propiedades que interesa verificar, muchas de ellas relacionadas al hecho de que un Proceso de Negocio puede ser visto como un autómata. En este contexto surgen las propiedades de *deadlock*, *livelock*, y *soundness*, las cuales tienen un amplio tratamiento en la literatura y por ende se aplican técnicas de verificación similares.

Las propiedades de grafos, en su gran mayoría, son verificadas mediante análisis estático o model cheking, lo que resulta sensato debido a que la mayoría de las propiedades de esta categoría aplican sobre la estructura. Por otro lado, en el resto de las categorías, no existe una tendencia a la utilización de una técnica en particular.

Las técnicas de deducción lógica tiene similar grado de utilización en las distintas categorías y casi siempre con un uso menor que otras técnicas

(análisis estático y model checking) . Esto se debe a la complejidad inherente a realizar la verificación utilizando un formalismo matemático que no puede ser automatizado. De la misma forma, las técnicas por construcción son casos aislados debido a la complejidad de definir un lenguaje que satisfaga las propiedades de interés por construcción. Finalmente, las técnicas de testing/simulación no son tan utilizadas debido al hecho de que por lo general no permiten tomar decisiones concluyentes.

En la Figura 3.6 podemos analizar cuáles son los modelos más usados utilizando el mismo criterio que en la figura anterior. Existe una clara tendencia a utilizar modelos con base en fundamentos formales en todas las categorías de propiedades que se deseen verificar, variando entre un 60-100 %. El uso de modelos informales, es decir, sin fundamentos matemáticos, solo son utilizados en propiedades de grafos con apenas un 15 %. Por último vemos que los semi-formales se utilizan en todas las categorías menos en la de recursos y varían entre 15-40 %. Es lógico que para la verificación de propiedades es necesario un lenguaje formal o semi formal.

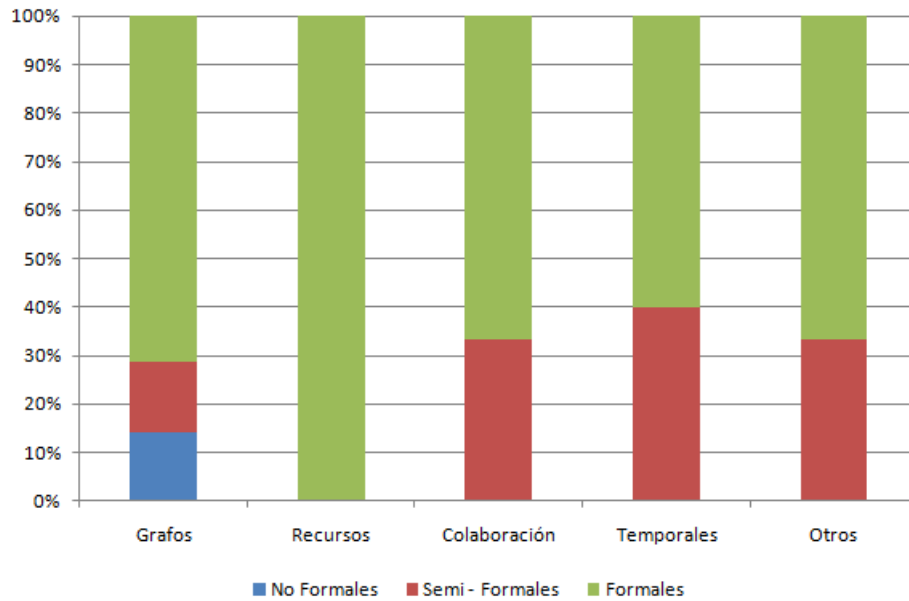


Figura 3.6: Lenguajes utilizados según propiedades

3.6. Conclusiones de la revisión sistemática

Hemos realizado una revisión exhaustiva de la literatura relacionada con la verificación de Procesos de Negocio en las diferentes etapas de su ciclo

de vida. Los resultados fueron clasificados acorde a diversas perspectivas de interés: tipo de proceso, etapa en la que se realiza la verificación, lenguaje de especificación utilizado, propiedades de interés a verificar, técnicas y herramientas utilizadas para la verificación. Cada dimensión aportan una lectura diferente de la problemática, favoreciendo así la comprensión del tema.

En los resultados obtenidos resalta el uso de lenguajes de especificación formales para la verificación, más allá de que los lenguajes informales son de gran ayuda para la integración y participación de profesionales de otras áreas para realizar procesos más adecuados al negocio. Esto se debe a que el uso de este tipo de lenguajes conlleva un trabajo extra respecto a la verificación.

Es notoria la verificación de Procesos de Negocio del tipo orquestación y está mas asociada a aspectos del modelado que de la ejecución. Además de que la mayoría de los artículos utilizan la teoría de grafos como base para la verificación de las propiedades. En cambio, la verificación de propiedades es mucho más diversificada, no hay una tendencia clara a la verificación de alguna propiedad en particular.

Resaltamos la importancia de haber realizado una revisión sistemática como método formal para la realización y organización del estado del arte de la verificación de Procesos de Negocio ya que permite mayor comprensión del tema, de una forma organizada y reproducible, y es de gran aporte para el área de conocimientos. Por el procedimiento utilizado y como nos enfocamos en el estado del arte y clasificación de resultados, el término correcto que describe lo que realizamos es *mapping study* [35]

Capítulo 4

Desarrollo de un entorno integrado de verificación

Implementamos una herramienta de verificación en el entorno del desarrollo integrado (IDE) Eclipse. Se trata de un Plug-in que integra distintas herramientas existentes de verificación de Procesos de Negocio. Al mismo tiempo, permite seguir extendiéndola, de forma que se pueda generar un entorno lo más completo posible. En la Sección *Requisitos del prototipo para Eclipse* se fundamenta por qué se optó por realizar el desarrollo de esta forma. En *Evaluación de herramientas* se desarrollan los motivos por qué se eligieron estas herramientas para integrar al entorno, mientras que en *Arquitectura del Plug-in* se explica cómo está compuesta la organización del Plug-in. En la Sección de *Extensión del Plug-in* se detalla cómo realizar una extensión de este prototipo.

4.1. Requisitos del prototipo para Eclipse

Un entorno integrado de verificación es un ambiente que centraliza diversos tipos de herramientas con el objetivo de realizar una verificación lo más completa posible de las distintas propiedades definidas. La decisión de llevar a cabo el desarrollo de este proyecto como un Plug-in de Eclipse se tomó por varios motivos relacionados con los beneficios que esta arquitectura provee.

Eclipse es una plataforma que ha sido diseñada desde cero para construir una herramienta que integre desarrollo web y aplicaciones. Por diseño, la plataforma no provee por si sola una gran cantidad de funcionalidades para el usuario. El valor de Eclipse es que fomenta el rápido desarrollo de funciones integradas basadas en un modelo de Plug-ins.

Eclipse es desarrollado por una comunidad de individuos y organizaciones que desean colaborar con software de código abierto comercialmente amigable. Sus proyectos están enfocados en construir una plataforma abierta de

desarrollo compuesta por frameworks extensibles, herramientas y tiempos de ejecución (runtimes), para construir, hacer deploy y gestionar software a través de todo el ciclo de vida.

En el núcleo de Eclipse hay una arquitectura para dinámicamente descubrir, cargar y ejecutar Plug-ins. La plataforma maneja la lógica de encontrar y ejecutar el código correcto. Cada Plug-in puede enfocarse en hacer una pequeña cantidad de tareas de forma correcta. ¿Qué tipo de tareas? Definir, testear, animar, publicar, compilar, debuggear, diagramar.

La plataforma utiliza el modelo de “workbench” (mesa de trabajo) común para integrar las herramientas desde el punto de vista del usuario final. Las herramientas que uno desarrolla puede conectarlas al workbench usando puntos de anclaje bien definidos llamados puntos de extensión (Figura 4.1). La plataforma en si misma está construida en capas de Plug-ins, cada una define extensiones para los puntos de extensión de Plug-ins de nivel más bajo, y a su vez definen sus propios puntos de extensión para una mayor personalización. Este modelo permite a los desarrolladores de Plug-ins agregar variedad de funciones a la plataforma básica de herramientas. Los artefactos para cada herramienta, como archivos y otra información, son coordinados por modelo de recursos de plataforma común. La plataforma le da a los usuarios una forma común de trabajar con las herramientas, y provee manejo integrado de los recursos que pueden crear con los Plug-ins.

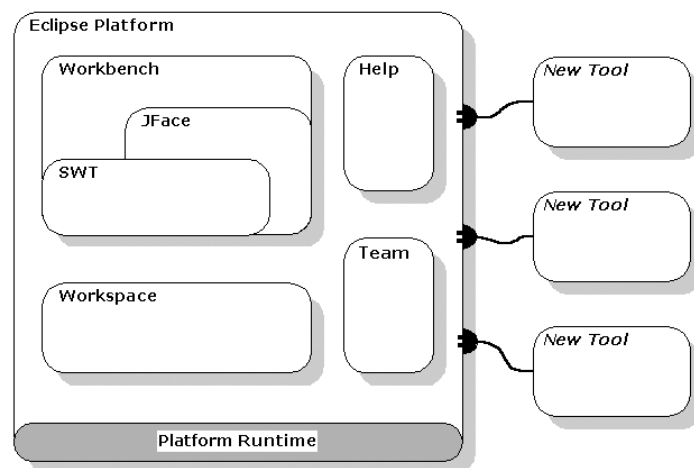


Figura 4.1: Arquitectura de la plataforma Eclipse

Los desarrolladores de Plug-ins también obtienen beneficios de esta arquitectura. La plataforma administra la complejidad de diferentes entornos

de tiempo de ejecución, como diferentes sistemas operativos o entornos de grupos de trabajo en servidores. De esta manera, los desarrolladores pueden enfocarse en sus tareas específicas en vez de preocuparse de estos problemas. [23]

El principal objetivo es integrar distintas herramientas por lo cual obtendríamos el beneficio de poder hacer uso de características de herramientas ya desarrolladas, como también permitir a terceros que continúen construyendo sobre la herramienta desarrollada en este proyecto. Dentro de los Plug-ins ya implementados que utilizamos, se encuentran Eclipse BPMN2 Modeler y Activiti Designer, porque permiten modelar un diagrama de forma gráfica y luego podemos verificar con nuestra herramienta. Por otro lado, otros desarrolladores podrían crear sus propios validadores y agregarlos como Plug-ins sobre el punto de extensión de este proyecto.

Al mismo tiempo, al ser otra de las características de los Plug-ins de Eclipse que puedan realizarse orientadas a distintos momentos del ciclo de vida del desarrollo, el Plug-in desarrollado en este proyecto orientado a la verificación puede utilizarse junto con otras herramientas enfocadas a otros momentos del ciclo, de manera de agregar valor al ciclo de vida de desarrollo. Otro de los beneficios de la plataforma Eclipse es que se ejecuta en distintos sistemas operativos.

4.2. Evaluación de herramientas

Explicaremos las decisiones tomadas a la hora de la elección de las herramientas. Hasta el momento, como ya mencionamos anteriormente, las propuestas se limitan a verificaciones de casos particulares o para Procesos de Negocio muy sencillos. Para realizar el entorno integrador de verificación nos planteamos utilizar distintos tipos de verificadores para mostrar la viabilidad del prototipo y poder extenderlo para otras verificaciones según las necesidades de terceros. Como primer paso, quisimos integrar un validador interno a Eclipse, es decir, una herramienta de código abierto, en la medida de lo posible que fuera liviano, que esté integrado con Eclipse y que no haya interacción con procesos externos. Por esto es que elegimos la herramienta Activiti, que posee diseñador gráfico integrado y detección de errores. Segundo, para ampliar el espectro de Procesos de Negocio soportados, queríamos realizar un transformador de modelos en lenguaje BPMN 2.0 a Redes de Petri. No fue fácil encontrar una herramienta completa que realice bien la transformación, pero tomamos la decisión de utilizar un Plug-in de ProM 6.0 y adaptarlo a nuestras necesidades. Este Plug-in modificado es capaz de realizar la transformación en segundo plano, sin abrir visores gráficos y como ventaja adicional, es de código abierto y desarrollada en Java. Por último nos pareció de mucho interés mostrar la forma de utilizar verificadores externos a Eclipse ya que no todos son para utilizarlos en Eclipse y sin este tipo de

herramientas el prototipo iba a quedar muy limitado. Así es que utilizamos WoPeD para la verificación de Redes de Petri.

Para la elección se tuvieron en consideración otras herramientas, pero al validar que el prototipo sirve con estos tres tipos de verificadores, agregar otros ya no tenía tanto valor. Como se explicará en una Sección más adelante, se brinda la posibilidad de una fácil integración de futuros validadores.

4.2.1. Activiti

Activiti es una plataforma de flujo de trabajo y Gestión de Procesos de Negocio (BPM) dirigido a personas de negocio, desarrolladores y administradores de sistema. Su núcleo es un rápido y sólido motor de procesos BPMN 2.0 para Java. Es de código abierto y distribuido bajo la licencia Apache. Activiti se ejecuta en cualquier aplicación Java, en un servidor, en un clúster o en la nube. Es extremadamente liviano y basado en conceptos simples [16].

La plataforma Activiti está conformada por distintos componentes, entre ellos Activiti Designer, un Plug-in de Eclipse que permite modelar procesos BPMN 2.0 desde el IDE. También tiene soporte incorporado para las extensiones específicas de Activiti para que el usuario utilice el potencial completo del proceso y del motor. El Plug-in tiene varios puntos de extensión para permitir al usuario desarrollar sobre el BPMN 2.0 regular y realizar adaptaciones para sus casos de uso específico [17].

4.2.2. WoPeD

WoPeD (Workflow Petri Net Designer, Diseñador de flujo de trabajo en Redes de Petri) es una herramienta fácil de utilizar para editar, simular y analizar flujos de trabajo como también Redes de Petri “plain place-transition”. WoPeD es desarrollado continuamente por miembros de la facultad y estudiantes de *Cooperative State University Karlsruhe* (Universidad de Educación Cooperativa, Alemania) en conjunto con un grupo de profesionales graduados de las tecnologías de la información.

WoPeD es publicado bajo la licencia de código abierto LGPL, que permite acceso gratis al código fuente y sus documentos de diseño subyacentes. Las herramientas basadas en Java pueden ejecutarse en todos los sistemas operativos que ya traen incorporado una versión de Java 5.0 o superior, en particular todas las plataformas Windows, la mayoría de las distribuciones Linux y MacOS X.

WoPeD se enfoca principalmente en propósitos educacionales y de publicación, dando soporte a profesores, investigadores y estudiantes en el área

de las Redes de Petri y Gestión de Procesos de Negocio. WoPeD está relacionado estrictamente a la bien establecida notación “van-der-Aalst” y se puede visualizar tanto la estructura como lo dinámico de las Redes de Petri, proveyendo un entendimiento más profundo de los principios subyacentes, conceptos de notación, propiedades clave y algoritmos [68].

La versión utilizada fue la 3.2.0 y el objetivo de utilizar esta herramienta es ampliar el espectro de opciones para realizar la verificación.

4.2.3. ProM

ProM (Process Mining Framework) es un framework de código abierto para algoritmos de minado de procesos. ProM provee una plataforma a los usuarios y desarrolladores de algoritmos de minado de procesos que es fácil de usar y extender [60]. Este framework soporta una amplia gama de técnicas de minado de procesos en forma de Plug-ins. Es independiente de la plataforma ya que está implementado en Java y se puede descargar gratis.

La versión 6 de ProM es distribuida en partes para brindar la máxima flexibilidad para futuros desarrollos. El núcleo de ProM 6 es distribuido como un paquete descargable usando la licencia de código abierto GNU Public License (GPL). Mientras tanto los Plug-ins son distribuidos como paquetes separados, que típicamente usan la licencia de código abierto Lesser GNU Public License (L-GPL) [61].

En la versión 6 de ProM los algoritmos de minado y la interfaz gráfica de usuario han sido cuidadosamente separadas, lo cual permitió que este proyecto pueda utilizar un Plug-in de ProM en segundo plano, sin tener que mostrar nada en la interfaz gráfica de esta herramienta [74].

4.3. Arquitectura del Plug-in

El prototipo desarrollado es un Plug-in de Eclipse que integra distintas herramientas de verificación y conversión de procesos de negocio ya existentes para lograr verificar modelos. Dentro de las herramientas integradas se encuentra un validador del proyecto Activiti que nos permite verificar si modelos representados en la notación BPMN 2.0 cumplen con determinadas reglas particulares a la notación. Otra de las herramientas integradas es ProM, particularmente utilizando Plug-ins específicos de ésta, que permiten transformar modelos representados en BPMN 2.0 a PNML con el objetivo de verificar el modelo resultante con otras herramientas. Complementariamente,

para hacer uso de la transformación de ProM, se integró WoPeD una herramienta que nos permite verificar los modelos resultantes de la transformación obteniendo un análisis estructural y verificando soundness. Finalmente, se implementó un punto de extensión para que terceros puedan agregar sus propias herramientas, de forma de agregarle valor al desarrollo.

4.3.1. Organización interna de componentes

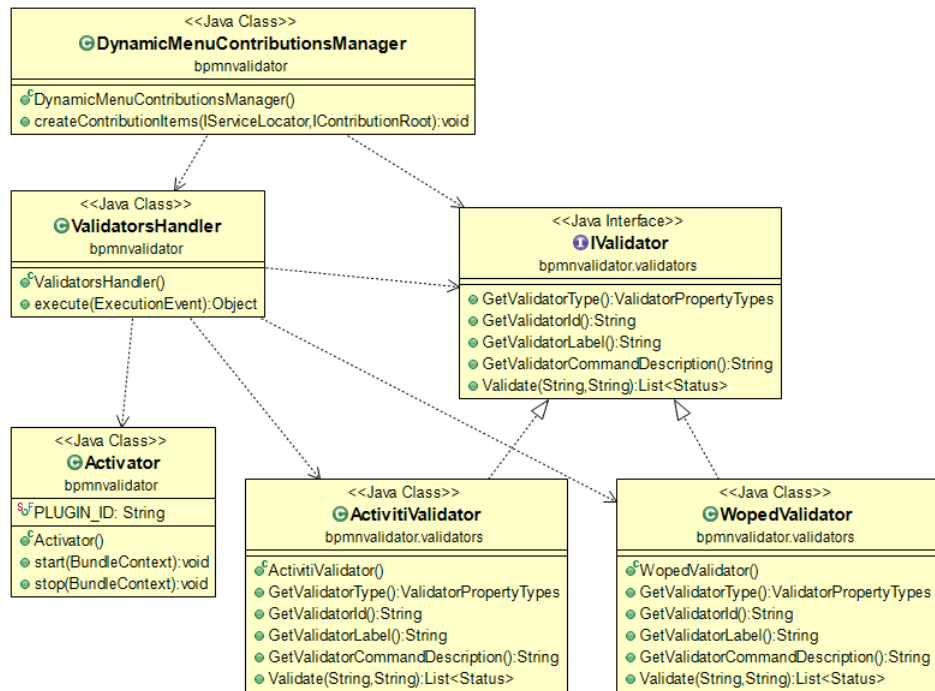


Figura 4.2: Diagrama de clases

En la Figura 4.2 se aprecia el diagrama de clases del prototipo desarrollado el cual pasaremos a explicar.

La interfaz *IValidator* es la encargada de determinar qué métodos deben implementar los validadores, ya sean internos al Plug-in, como externos agregados a través del punto de extensión. En esta interfaz se encuentra el enumerado *ValidatorPropertyTypes* que contiene las clasificaciones de propiedades vistas en este proyecto (Grafos, Recursos, Colaboración, Temporales, Otros), con el objetivo de organizar los validadores según este criterio. Los métodos a implementar por los validadores son:

- **GetValidatorType():** El validador debe retornar el tipo de propiedad que verifica para ser agregado al submenú correspondiente.

- **GetValidatorId():** Método que retorna el identificador del validador implementado.
- **GetValidatorLabel():** Método que retorna la etiqueta que se muestra en el menú para seleccionar la ejecución de este validador.
- **GetValidatorCommandDescription():** Retorna la descripción que se le asignará al comando del validador.
- **Validate(String path, String pluginId):** Método que implementa el validador del modelo. Recibe la ruta del archivo del modelo y el identificador del Plug-in, para devolver una lista de errores o un mensaje de éxito.

Dentro del proyecto hay dos validadores implementados que se encuentran en las clases *ActivitiValidator* y *WopedValidator*. Ambos implementan la interfaz mencionada y no tienen ninguna dependencia con otra clase.

Por otro lado, la clase *ValidatorsHandler* es la encargada del manejo de los validadores, tanto internos como externos. Esta se ocupa generar una lista con todos los validadores y entregárselos en una lista al *DynamicMenuContributionsManager*, como también de manejar el evento de ejecutar un validador cuando este es disparado. Por esta segunda responsabilidad es que esta clase hereda de la clase *AbstractHandler* y hace un override del método *execute*.

DynamicMenuContributionsManager es la encargada de agregar dinámicamente las contribuciones en el menú contextual del PackageExplorer de Eclipse, agrega al menú todas las herramientas integradas en este Plug-in, sean nuestras o de terceros. Esta clase hereda de *ExtensionContributionFactory* para realizar la tarea de crear los ítems del menú contextual haciendo un override del método *createContributionItems*. Es por esto que también tiene como dependencia la interfaz *IValidator*, ya que necesita obtener de los validadores sus etiquetas a mostrar, como también su clasificación para saber en qué menú agregarlos.

Finalmente, la clase *Activator* es la encargada de manejar el ciclo de vida del Plug-in, el método *getDefault()* es ejecutado por el runtime de Eclipse para que se le retorne una instancia del Plug-in. El *ValidatorsHandler* tiene como dependencia esta clase porque obtiene el identificador del Plug-in (*PLUGIN_ID*) y es necesario para mostrar los mensajes en el *Error Log*.

4.3.2. Integración con Eclipse

Para integrar nuestro Plug-in con Eclipse se extendieron dos puntos de extensión, por un lado los comandos para poder especificar el comportamiento de las acciones y por el otro los submenú para agregar al menú contextual.

```

<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
  <extension-point id="bpmnvalidator.extensions.ExternalValidator"
    name="ExternalValidator" schema="schema/bpmnvalidator.extensions.ExternalValidator.exsd"/>
  <extension
    point="org.eclipse.ui.commands">
    <category
      name="ImplementedValidators"
      id="BPMNValidator.commands.implementedValidatorsCategory">
    </category>
  </extension>
  <extension
    point="org.eclipse.ui.menus">
    <menuContribution
      allPopups="true"
      class="bpmnvalidator.DynamicMenuContributionsManager"
      locationURI="popup:org.eclipse.jdt.ui.PackageExplorer">
    </menuContribution>
  </extension>
</plugin>

```

Figura 4.3: Plugin.xml

Para ambas contribuciones se debió agregar en *plugin.xml* (Figura 4.3) la declaración que este Plug-in implementa estos dos puntos de extensión de eclipse, el punto *org.eclipse.ui.commands* y el punto *org.eclipse.ui.menus* (Figura 4.4). En el caso de los menú la clase definida como implementación de esta extensión es la llamada *DynamicMenuContributionsManager* que hereda de la clase *ExtensionContributionFactory*. Esta clase dinámicamente agrega los validadores al menú según la cantidad de validadores implementados le haya devuelto el Manejador de validadores *ValidatorsHandler* que puede ser variable en base a la cantidad de extensiones encontradas.

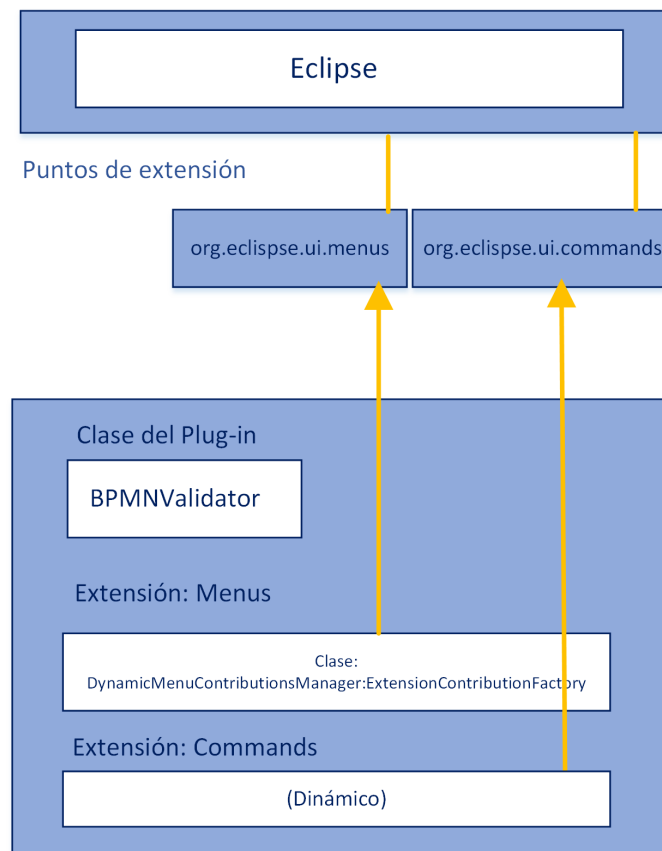


Figura 4.4: Puntos de extensión implementados

Por otro lado no hay una clase declarada como la implementación de la extensión de comandos ya que a priori no hay manera de saber cuántos comandos se deben definir, ya que como antes se mencionó, la cantidad de validadores es variable. Es por esto que los comandos se crean dinámicamente, también por la clase *DynamicMenuContributionsManager* que crea un comando por validador obtenido y lo vincula con un manejador.

4.3.3. Integración con Activiti

La integración con Activiti se realizó agregando los jars del mismo al Plug-in que se estaba desarrollando para de esta manera tener acceso a la API y los servicios para hacer uso de los mismos.

El engine API es la forma más común de interactuar con Activiti. El punto de partida central es el *ProcessEngine*, que brinda varios servicios que contienen métodos de Gestión de Procesos de Negocio y flujo de trabajo [18].

```

public List<Status> Validate(String path, String pluginId)
{
    List<Status> validatorResult;

    if (path.endsWith(this.validFileExtension1) ||
        path.endsWith(this.validFileExtension2))
    {
        try
        {
            // Inicializar variables necesarias para validacion
            ProcessEngine processengine =
                ProcessEngines.getDefaultProcessEngine();
            RepositoryService repositoryService =
                processengine.getRepositoryService();
            DeploymentBuilder db = repositoryService.createDeployment();

            // Cargar el modelo
            db.addInputStream("ProcessToValidate.bpmn20.xml",
                new FileInputStream(path));
            // Hacer deploy
            db.deploy();

            // Si termina todo ok, devolver mensaje de exito
            validatorResult = new ArrayList<Status>();
            validatorResult.add(new Status(IStatus.INFO,
                pluginId, this.mensajeDeExito));
        }
        catch (ActivitiException actEx)
        {
            // Transformar los mensajes de Activiti para mostrarselos al usuario
            validatorResult = TransformExceptionMessage(actEx.getMessage(),
                pluginId);
        }
        catch (Exception e)
        {
            // Excepcion no esperada
            validatorResult = new ArrayList<Status>();
            validatorResult.add(new Status(IStatus.INFO,
                pluginId, this.mensajeDeErroNoManejado));
        }
        finally
        {
            // Finalizar el motor de procesos
            ProcessEngines.destroy();
        }
    }
    else
    {
        // Archivo de formato no valido
        validatorResult = new ArrayList<Status>();
        validatorResult.add(new Status(IStatus.ERROR,
            pluginId, this.mensajeArchivoDeTipoNoValido));
    }

    return validatorResult;
}

```

Figura 4.5: Método Validate del *ActivitiValidator*

Para acceder a la API en el *ActivitiValidator* (Figura 4.5), lo primero que se utiliza es el método *ProcessEngines.getDefaultProcessEngine()* que inicializa y crea un motor de procesos la primera vez que es llamado (una vez inicializado, siempre devuelve el mismo motor de procesos). El motor de procesos buscará el archivo de configuración *activiti.cfg.xml* y así se inicializará según la configuración seteada en el archivo. Para este proyecto la

configuración elegida fue *StandaloneInMemProcessEngineConfiguration* que significa que se utiliza una base de datos en memoria y que Activiti se encarga de sus transacciones, se crea cuando se inicializa y se elimina cuando se cierra el motor. Se optó por esta configuración, ya que era la más sencilla posible para lo que se necesitaba, porque el motor se utiliza para validar y luego se cierra con el método *ProcessEngines.destroy()*.

Luego de inicializado el motor de procesos, se pasa a obtener el servicio de repositorio (*RepositoryService*), que ofrece operaciones para gestionar y manipular deployments y definiciones de procesos. En este caso, se utiliza para crear el deployment que internamente validará el proceso seleccionado por el usuario. Una definición de proceso (*process definition*) es una contraparte en Java de un proceso BPMN 2.0, es la representación de la estructura y el comportamiento de cada uno de los pasos del proceso. El deployment es la unidad de empaquetamiento que tiene el motor de Activiti, puede contener múltiples definiciones de archivos BPMN 2.0 xml, en este caso solo se utiliza una porque validará de a un modelo a la vez. Hacer deploy de un deployment significa cargarlo en el motor, donde todos los procesos son inspeccionados y parseados antes de ser almacenados en la base de datos. Esto es lo relevante para el Plug-in desarrollado en este proyecto, ya que se hace uso de la validación de Activiti para obtener los errores encontrados por su validador.

Cuando se valida un modelo con la herramienta en caso de que el validador encuentre algún problema, lanzará una excepción del tipo *ActivitiException* con toda la información relacionada del o de los errores detectados, en caso contrario continuará con la ejecución. Es entonces que el Plug-in devuelve en caso de error, los mensajes de error del validador de Activiti y en caso contrario un mensaje de éxito.

En la versión de Activiti utilizada en este proyecto (Activiti 5.16), el validador es capaz de encontrar sesenta y tres problemas distintos relacionados con errores de representación en BPMN 2.0 que se verifican realizando un análisis estático (en el Anexo se detalla la lista completa de errores que puede detectar).

Para ejecutar el validador simplemente hay que hacer click derecho sobre el archivo de extensión *.bpmn* y seleccionar dentro de “Validators” → “Otros” → “Activiti”. En ese momento, el Plug-in comenzará a ejecutar el validador y cuando finalice devolverá en el *Error Log* de Eclipse el mensaje que corresponda.

4.3.4. Integración con WoPeD y ProM

Como se mencionó en la Sección anterior, ProM tiene separado por un lado su interfaz gráfica y por otro sus algoritmos que pueden ser ejecutados

por línea de comando sin necesidad de hacer uso de la UI. En este caso, la invocación se hace pasando como parámetro la ruta de un script (Figura 4.6) donde se encuentran las instrucciones a ejecutar de la herramienta.

Para realizar la validación se utilizó uno de los Plug-ins preexistentes de la plataforma y se le realizaron pequeños cambios para ajustar su comportamiento, de forma de realizar la transformación como se quería. Específicamente el problema era que el Plug-in no soportaba la importación de archivos *.bpmn* por consola, acción que sí se podía realizar a través de la interfaz gráfica, por lo que se realizaron modificaciones en algunas funciones para darle soporte por línea de comando.

El prototipo adaptado fue exportado para utilizarse como un jar ejecutable de manera de desacoplarlo del resto del desarrollo. El jar también contiene otros Plug-ins, por lo tanto, haciendo el llamado adecuado también se podría hacer uso de estos para extender las funcionalidades disponibles. En la misma ubicación donde está el jar también está el archivo *ProM.ini* el cual contiene la configuración para inicializar ProM. En el proyecto se tuvo que agregar un archivo de configuración donde guardar la ruta absoluta al jar de ProM, ya que no se puede acceder de forma relativa. El archivo también contiene la ruta absoluta donde está instalado WoPeD en el sistema del usuario para abrirlo automáticamente al finalizar la conversión de ProM.

```
print ("Comenzando transformador");
# Se define la ruta absoluta del archivo que contiene el modelo bpmn a verificar
String filename = "C:\\diagram.bpmn";
# Utilizando un Plug-in se carga el modelo internamente
res = import_bpmn_model_from_bpmn_2_0_file(filename);
# Se define la ruta absoluta del archivo a generar
String outPetri = "C:\\diagram.pnml";
# Creacion de un archivo con la ruta antes definida
file = new java.io.File(outPetri);
# Llamado al Plug-in que convierte el archivo BPMN a Petri Net (PNML)
# y exporta la Petri net al archivo creado
res2 = convert_bpmn_to_petrinet_modificado(res,file);
print ("Transformador finalizado");
```

Figura 4.6: Ejemplo de script generado para que ProM ejecute

A partir de los Plug-ins de ProM que están en el jar importado, se puede lograr transformar un modelo BPMN 2.0 a un modelo PNML. Luego de transformado el archivo se lanza la ejecución de WoPeD para realizar la verificación, para esto el usuario debe tener previamente WoPeD instalado y su ruta debe estar en el archivo de configuración. Luego de abierto WoPeD el usuario puede realizar distintas verificaciones del modelo seleccionando en la solapa superior “Analyze” y luego “Semantical analysis”. En la Sección de

“Semantical analysis” se pueden apreciar distintos análisis realizados por la herramienta, divididos en dos categorías, análisis estructurales y Soundness.

4.3.5. Extensibilidad

Para lograr que sea extensible fue necesario declarar el punto de extensión en el archivo *plugin.xml* (Figura 4.3). El punto de extensión es descrito en un esquema XML donde se definen los detalles y el contrato.

En *plugin.xml* definimos su id (*bpmnvalidator.extensions.ExternalValidator*), el nombre (*ExternalValidator*) e hicimos referencia a un archivo de esquema en el cual se especifica la interfaz que la extensión debe implementar (*bpmn-validator.extensions.ExternalValidator.xsd*).

Esta extensión fue definida de forma que para agregar un Plug-in la interfaz a implementar sea *IValidator* (Figura 4.7), con los métodos a implementar explicados en la subsección anterior. De esta manera se puede implementar un verificador que a partir de una ruta de archivo verifique las propiedades que sean de interés para los desarrolladores.

```

public interface IValidator {
    /*
     * Enumerado que contiene los tipos de validadores
     * segun que tipo de propiedad verifica
     */
    public enum ValidatorPropertyTypes{
        Grafos,
        Recursos,
        Colaboracion,
        Temporales,
        Otros
    }

    /*
     * Metodo que debe devolver que tipo de validador es el implementado
     * con el objetivo de agregarlo al submenu correcto
     */
    public ValidatorPropertyTypes GetValidatorType();

    /*
     * Metodo que debe devolver un identificador para el validador implementado
     */
    public String GetValidatorId();

    /*
     * Metodo que debe retornar la etiqueta que se desea mostrar
     * en el menu para este validador especifico
     */
    public String GetValidatorLabel();

    /*
     * Metodo que debe devolver una breve descripcion para el comando
     * a implementar
     */
    public String GetValidatorCommandDescription();

    /*
     * Este metodo debe implementar el validador del modelo. A partir de la ruta
     * del modelo que recibe en el path, debe validarlo para luego devolver su
     * salida con el formato de una lista de org.eclipse.core.runtime.Status.
     * En caso de que haya errores u observaciones, se devolveran en una instancia
     * de Status cada uno/a o en caso de error un mensaje de exito dentro de una
     * instancia de Status.
     */
    public List<Status> Validate(String path, String pluginId);
}

```

Figura 4.7: Interfaz IValidator

4.4. Extensión del Plug-in

En esta Sección pasaremos a detallar cuales son los pasos a seguir para crear un nuevo validador que extienda el Plug-in desarrollado. El diagrama de la Figura 4.9 muestra cual es la relación entre el Plug-in a extender (Anfitrión) y el Plug-in que lo extiende.

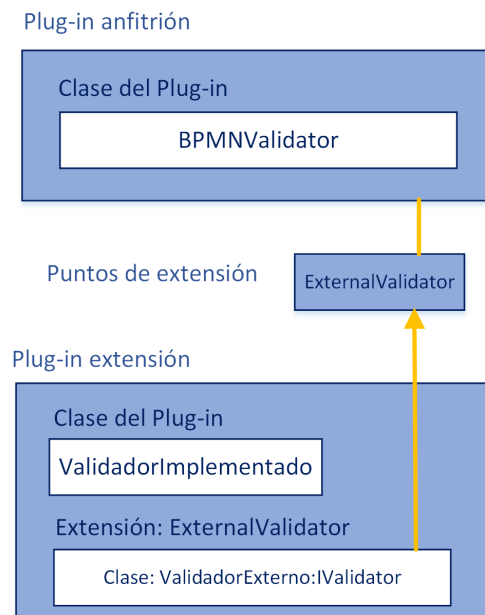


Figura 4.8: Diagrama punto de extensión

En primera instancia en Eclipse se debe crear un proyecto de Plug-in como punto de partida, luego se debe agregar como dependencia el proyecto *BPMNValidator* para tener acceso a la interfaz a implementar. Luego, en el archivo *plugin.xml* (Figura 4.9) del nuevo Plug-in agregar “*bpmnvalidator.extensions.ExternalValidator*” como punto de extensión a implementar y dentro una referencia al validador que implementa la interfaz.

```
<plugin>
  <extension
    point="bpmnvalidator.extensions.ExternalValidator">
    <externalvalidator validator=
      "bpmnvalidator.extensions.wopedvalidator.validator.NuevoValidador">
    </externalvalidator>
  </extension>
</plugin>
```

Figura 4.9: Archivo *plugin.xml* del nuevo Plug-in

Después se debe crear el validador que implemente la interfaz *IValidator* en la ruta donde se referenció en el archivo *plugin.xml*.

Los métodos a implementar de la interfaz:

ValidatorPropertyTypes GetValidatorType() : El método retorna el tipo de propiedad que el validador verifica, para esto, la interfaz tiene el enume-

rado *ValidatorPropertyTypes*. Entonces, los tipos de validador pueden ser: Grafos, Recursos, Colaboración, Temporales, Deduccion_Logica u Otros.

String GetValidatorId(): Aquí se debe devolver un identificador para el validador implementado.

String GetValidatorLabel(): El método retorna la etiqueta que se quiere mostrar en el menú contextual para hacer el llamado al validador.

String GetValidatorCommandDescription(): Devuelve una pequeña descripción de lo que hace el validador, este valor será asignado a la descripción del comando.

List<Status> Validate(String path): Este método debe implementar el validador del modelo. A partir de la ruta del modelo que recibe en el path, debe validarlo para luego devolver su salida con el formato de una lista de *org.Eclipse.core.runtime.Status*. En caso de que haya errores u observaciones, se devolverán todo en una instancia de Status y en otro caso, un mensaje de éxito dentro de una instancia de Status. La clase Status permite mostrar en el *Error Log* de Eclipse distintos mensajes, para crear una instancia lo único que se necesita son tres parámetros:

- El *IStatus* que es un enumerado que define el tipo de mensaje que se va a mostrar al usuario, varía desde información (INFO), un error (ERROR), advertencia (WARNING).
- El id del Plug-in, que para esto hay que obtener del jar del *BPMNValidator* su id de forma *Activator.PLUGIN_ID*.
- Finalmente, el mensaje que se le quiera mostrar al usuario.

Capítulo 5

Verificación del prototipo

El objetivo del Capítulo es mostrar una serie de casos de prueba, con modelos complejos de la realidad, para verificar la implementación del prototipo que fue realizado. Intentan cubrir varios elementos del lenguaje BPMN 2.0 y errores conocidos en verificación de Procesos de Negocio. En la Sección *Análisis herramienta Activiti* analizaremos la herramienta Activiti para estudiar sus ventajas, alcance y limitaciones. Luego, en la Sección *Análisis del transformador y herramienta WoPeD* utilizaremos los mismos modelos para transformarlos a Redes de Petri evaluando el alcance del mapeo entre lenguajes y ejecutaremos las pruebas sobre los modelos resultantes del mapeo en la herramienta WoPeD. Por último, en *Limitaciones y errores conocidos* explicaremos inconvenientes que encontramos en estos casos de prueba o en otros que fueron realizados.

5.1. Análisis herramienta Activiti

Para comenzar la verificación del prototipo utilizaremos primero la herramienta Activiti que chequea aspectos básicos de los elementos BPMN 2.0 y del modelo. Utilizaremos tres casos de prueba: *Agencia de Viajes*, *Enviar Artículo* y *Carrito de Compras*. Para cada uno explicaremos cual es su objetivo y se les introducirán errores para testear si es capaz de detectarlos o no. Se analizan los resultados en función de lo que se estaba probando.

5.1.1. Caso de prueba 1: Agencia de Viajes

Este caso de prueba es un proceso realizado en una agencia de viajes que se muestra en la Figura 5.1 y representa la reserva de un paquete de vacaciones. El proceso inicia registrando un nuevo pedido de viaje, verifica que se pague el paquete y recibe el resultado del pago. Posteriormente, envía el resultado de la solicitud y dependiendo del resultado existen dos caminos posibles; Si el resultado es correcto, se procede a realizar las reservas de

vuelo, hotel y automóvil (tres tareas distintas). Se esperan los resultados de las reservas para poder continuar con el proceso. Una vez que tenemos los tres resultados se registra la reserva, se envían los resultados y culmina el proceso. En el caso del resultado de la solicitud hubiera sido Incorrecto, el proceso finaliza sin realizar otra acción.

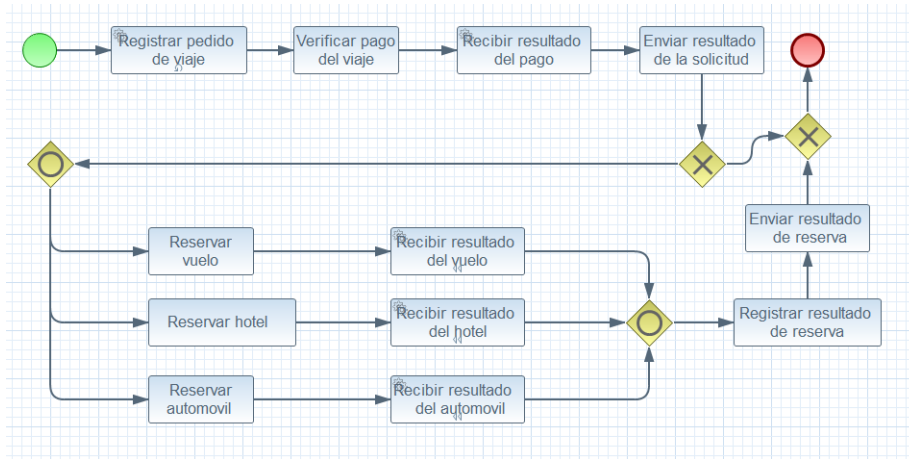


Figura 5.1: Caso de prueba 1 - Agencia de viajes

Injectaremos errores que se pueden visualizar en la Figura 5.2. Se le agregó una condición en el flujo de salida del último XOR con el que termina el proceso (en caso afirmativo, el proceso sigue por ese camino) y desde la actividad *Registrar resultado de reserva* se agregó un *gateway* sin flujo de salida. Además, ningún *Service Task* fue implementado pero no se visualiza gráficamente.

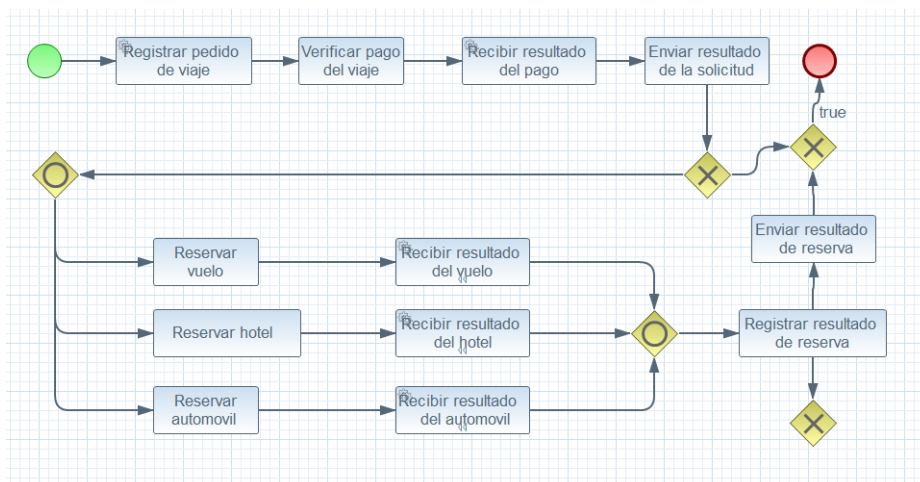


Figura 5.2: Modelo con errores

La verificación del caso de prueba con la herramienta Activiti generó una lista de errores que se puede visualizar en la Figura 5.3.

Message
Validator de Activiti finalizado
[Validation set: 'activiti-executable-process' Problem: 'activiti-exclusive-gateway-no-outgoing-seq-flow'] : Exclusive gateway has no outgoing sequence
[Validation set: 'activiti-executable-process' Problem: 'activiti-exclusive-gateway-condition-not-allowed-on-single-seq-flow'] : Exclusive gateway has on
[Validation set: 'activiti-executable-process' Problem: 'activiti-servicetask-missing-implementation'] : One of the attributes 'class', 'delegateExpressi...
[Validation set: 'activiti-executable-process' Problem: 'activiti-servicetask-missing-implementation'] : One of the attributes 'class', 'delegateExpressi...
[Validation set: 'activiti-executable-process' Problem: 'activiti-servicetask-missing-implementation'] : One of the attributes 'class', 'delegateExpressi...
[Validation set: 'activiti-executable-process' Problem: 'activiti-servicetask-missing-implementation'] : One of the attributes 'class', 'delegateExpressi...
[Validation set: 'activiti-executable-process' Problem: 'activiti-servicetask-missing-implementation'] : One of the attributes 'class', 'delegateExpressi...
Errors while parsing:
Validator de Activiti ejecutando...

Figura 5.3: Errores encontrados con Activiti

Activiti encontró que los *service task* utilizados (Registrar pedido de viaje, Recibir resultado del pago, Recibir resultado del vuelo, Recibir resultado del hotel y Recibir resultado del automovil) no están implementados. Como es el mismo error en todos los *service task*, mostraremos la descripción completa del error de la tarea *Recibir resultado del pago*.

[Validation set: 'activiti-executable-process' | Problem: 'activiti-servicetask-missing-implementation'] : One of the attributes 'class', 'delegateExpression', 'type', 'operation', or 'expression' is mandatory on serviceTask. - [Extra info : processDefinitionId = AgenciadeViajesV3 | processDefinitionName = Default Process | | id = ServiceTask_4 | | activityName = Recibir resultado del pago |] (line: 98, column: 106)

Se observa que en la salida del error nos indica cuál es el problema y la posible causa. Como información adicional para poder solucionar el error muestra el *Id* y *Nombre* del proceso y del elemento. Con estos datos podemos realizar los cambios en un editor gráfico, como por ejemplo, *Activiti Designer* o sino con el número de línea y columna lo podemos buscar en el código del archivo directamente (se puede visualizar con un editor de texto o con alguna herramienta para archivos XML).

Activiti reconoce que algún atributo de esos está vacío, pero completando al menos uno ya no indica que haya error. Es recomendable revisar la configuración de cada atributo para solucionar el problema ya que si se completa una entrada puede ocurrir que quede otra vacía pero Activiti no avisa de fallas.

Otro error encontrado fue “ Exclusive gateway has no outgoing sequence flow” en el *exclusive gateway* que se encuentra en el caso de prueba debajo de la tarea *Registrar resultado de reserva* (Uno de los errores inyectado). Esto significa que no tiene flujo saliente. A continuación vemos la salida completa del error:

[Validation set: 'activiti-executable-process' / Problem: 'activiti-exclusive-gateway-no-outgoing-seq-flow'] : Exclusive gateway has no outgoing sequence flow - [Extra info : processDefinitionId = AgenciadeViajesV3 / processDefinitionName = Default Process / / id = ExclusiveGateway_3 / /] (line: 104, column: 53)

Por último, encontró: “activiti-exclusive-gateway-condition-not-allowed-on-single-seq-flow”. El error es tener un único flujo de salida con una condición ya que en caso contrario el proceso no puede continuar y queda trancado. A continuación vemos la salida completa del error:

[Validation set: 'activiti-executable-process' / Problem: 'activiti-exclusive-gateway-condition-not-allowed-on-single-seq-flow'] : Exclusive gateway has only one outgoing sequence flow. This is not allowed to have a condition. - [Extra info : processDefinitionId = AgenciadeViajesV3 / processDefinitionName = Default Process / / id = ExclusiveGateway_2 / / activityName = / /] (line: 26, column: 91)

5.1.2. Caso de prueba 2: Enviar Artículo

El caso de prueba representa el envío de un artículo a un depósito. El proceso se inicia enviando el artículo deseado, luego de recibir la confirmación del envío, se recibe la resolución de si corresponde o no ingresarlo al depósito. En caso afirmativo se manda el código CRC y posteriormente cuando se recibe la confirmación del código, se envía la inscripción de quien lo ingresa hasta que el resultado sea positivo y se pueda solicitar un lugar para depositarlo. Se va a solicitar lugar hasta que el resultado de la solicitud sea positivo y así pueda finalizar el proceso. En la Figura 5.4 podemos ver el modelo.

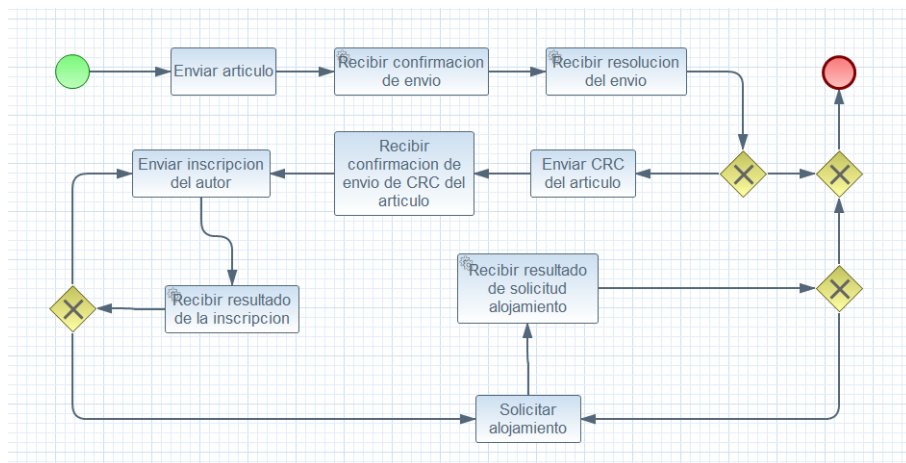


Figura 5.4: Caso de prueba 2 - Enviar Artículo

Agregamos otro evento de inicio y un *Script Task* (no implementado) que lo podemos visualizar en la Figura 5.5. Ningún *Service Task* fue implementado.

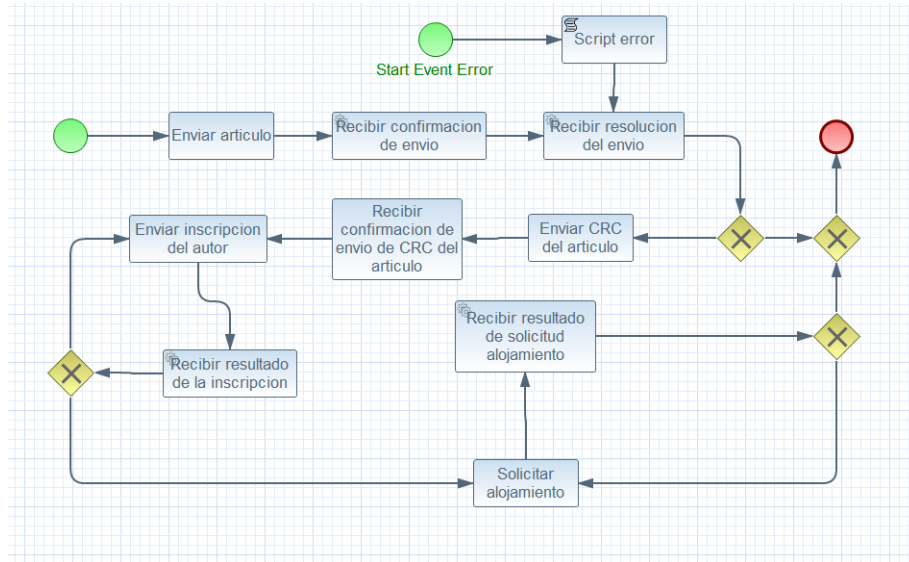


Figura 5.5: Modelo con errores

La Figura 5.6 nos muestra la lista completa de errores que se encontraron.

Message
Validator de Activiti finalizado
[Validation set: 'activiti-executable-process' Problem: 'activiti-servicetask-missing-implementation'] : One of the attributes 'class', 'delegateExpressi...
[Validation set: 'activiti-executable-process' Problem: 'activiti-servicetask-missing-implementation'] : One of the attributes 'class', 'delegateExpressi...
[Validation set: 'activiti-executable-process' Problem: 'activiti-servicetask-missing-implementation'] : One of the attributes 'class', 'delegateExpressi...
[Validation set: 'activiti-executable-process' Problem: 'activiti-servicetask-missing-implementation'] : One of the attributes 'class', 'delegateExpressi...
[Validation set: 'activiti-executable-process' Problem: 'activiti-scripttask-missing-script'] : No script provided for script task - [Extra info : proces... (Open
[Validation set: 'activiti-executable-process' Problem: 'activiti-start-event-multiple-found'] : Multiple none start events or timer start events not sup...
[Validation set: 'activiti-executable-process' Problem: 'activiti-start-event-multiple-found'] : Multiple none start events or timer start events not sup...
Errors while parsing:
Validator de Activiti ejecutando...

Figura 5.6: Errores encontrados con Activiti

Al igual que en el primer caso de prueba, se encuentra presente el error “activiti-servicetask-missing-implementation”. Aparecen dos nuevos tipos de errores que son: “ activiti-scripttask-missing-script” y “ activiti-start-event-multiple-found”. El primer error se debe a que no hay un script implementado en la actividad y el segundo se debe a que un proceso no puede tener dos eventos de inicio. El error se muestra dos veces, ya que la herramienta no puede determinar a priori cual es el evento de inicio correcto. Estos errores eran los esperados según los elementos agregados al modelo. Mostraremos el error completo de cada uno.

[Validation set: 'activiti-executable-process' / Problem: 'activiti-scripttask-missing-script'] : No script provided for script task - [Extra info : processDefinitionId = EnviarArticuloV3 / processDefinitionName = Default Process / / id = ScriptTask_1 / / activityName = Script error /] (line: 86, column: 61)

[Validation set: 'activiti-executable-process' / Problem: 'activiti-start-event-multiple-found'] : Multiple none start events or timer start events not supported on process definition - [Extra info : processDefinitionId = EnviarArticuloV3 / processDefinitionName = Default Process / / id = StartEvent_1 / / activityName = Start Event Error /] (line: 91, column: 66)

5.1.3. Caso de prueba 3: Carrito de Compras

El caso de prueba representa la compra de artículos en un supermercado. El proceso inicia con la elección de todos los artículos que se desean comprar. Una vez terminada esta etapa se envía la orden de compra y posteriormente cuando se recibe la resolución de la orden de compra, el proceso se condiciona a que la resolución sea correcta o no. En caso afirmativo, se recibe la factura en referencia a esa orden de compra y se puede consultar el estado de la orden de compra o directamente recibir los artículos. Si se consulta la orden de compra los recibe después y así termina el proceso. En caso de que la resolución de la orden de compra no sea aceptada, se termina el proceso. El modelo se observa en la Figura 5.7 y le introduciremos otro inicio con timer y un *Mail Task* para generar errores, que lo visualizamos en la Figura 5.8

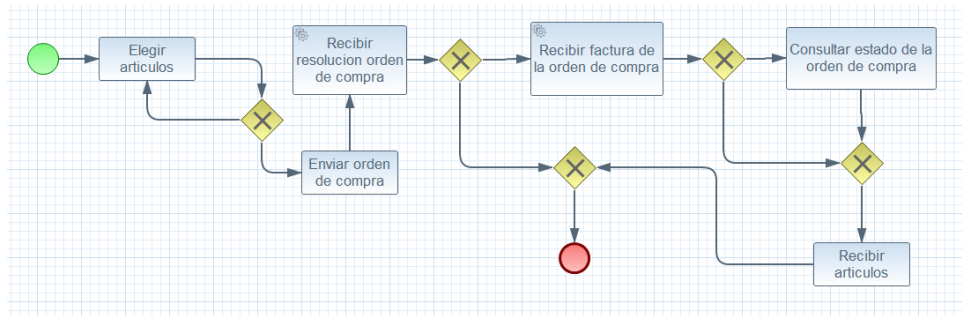


Figura 5.7: Caso de prueba 3 - Carrito de compras

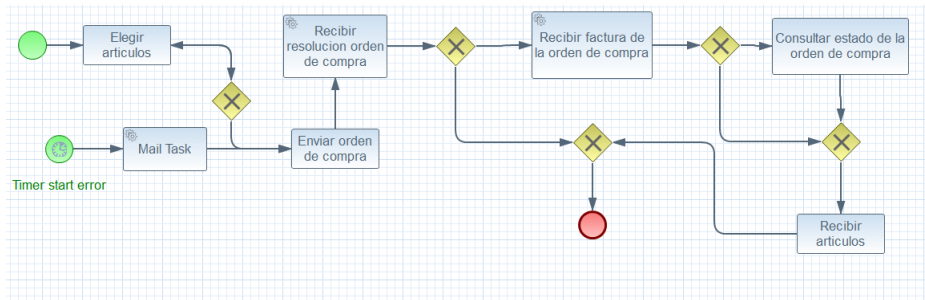


Figura 5.8: Modelo con errores

Al utilizar la herramienta Activiti obtuvimos la lista de errores que podemos ver en la Figura 5.9

Message
Validator de Activiti finalizado
[Validation set: 'activiti-executable-process' Problem: 'activiti-event-timer-missing-configuration' Timer needs configuration (either timeDate, timeCycle or timeDuration is needed) - [Extra info : processDefinitionId = CarritodeCompras2-Cliente processDefinitionName = Default Process id = StartEvent_1 activityName = Timer start error] (line: 81, column: 66)
[Validation set: 'activiti-executable-process' Problem: 'activiti-mailtask-no-content' Text, html, textVar or htmlVar field should be provided - [Extra info : processDefinitionId = CarritodeCompras2-Cliente processDefinitionName = Default Process id = ServiceTask_3 activityName = Mail Task] (line: 76, column: 108)
[Validation set: 'activiti-executable-process' Problem: 'activiti-mailtask-no-recipient' No recipient is defined on the mail activity - [Extra info : processDefinitionId = CarritodeCompras2-Cliente processDefinitionName = Default Process id = ServiceTask_3 activityName = Mail Task] (line: 76, column: 108)
[Validation set: 'activiti-executable-process' Problem: 'activiti-servicetask-missing-implementation' One of the attributes 'class', 'delegateExpression' is missing - [Extra info : processDefinitionId = CarritodeCompras2-Cliente processDefinitionName = Default Process id = ServiceTask_3 activityName = Mail Task] (line: 76, column: 108)
[Validation set: 'activiti-executable-process' Problem: 'activiti-start-event-multiple-found' Multiple none start events or timer start events not supported - [Extra info : processDefinitionId = CarritodeCompras2-Cliente processDefinitionName = Default Process id = StartEvent_1 activityName = Timer start error] (line: 81, column: 66)
[Validation set: 'activiti-executable-process' Problem: 'activiti-start-event-multiple-found' Multiple none start events or timer start events not supported - [Extra info : processDefinitionId = CarritodeCompras2-Cliente processDefinitionName = Default Process id = StartEvent_1 activityName = Timer start error] (line: 81, column: 66)
Errors while parsing:
Validator de Activiti ejecutando...

Figura 5.9: Caso de prueba 3: Errores encontrados con Activiti

Al igual que en los casos anteriores, se detectaron los errores “activiti-servicetask-missing-implementation” y “activiti-start-event-multiple-found” como estaba previsto. Otros problemas encontrados fueron que el inicio con timer no estaba configurado y que la actividad *mail task* no tiene receptor ni contenido. Esto era de esperar ya que fueron los errores introducidos junto al doble evento de inicio. Vemos a continuación la salida completa de ambos errores:

[Validation set: 'activiti-executable-process' | Problem: 'activiti-event-timer-missing-configuration' | Timer needs configuration (either timeDate, timeCycle or timeDuration is needed) - [Extra info : processDefinitionId = CarritodeCompras2-Cliente | processDefinitionName = Default Process | id = StartEvent_1 | activityName = Timer start error |] (line: 81, column: 66)

[Validation set: 'activiti-executable-process' | Problem: 'activiti-mailtask-no-content' | Text, html, textVar or htmlVar field should be provided - [Extra info : processDefinitionId = CarritodeCompras2-Cliente | processDefinitionName = Default Process | id = ServiceTask_3 | activityName = Mail Task |] (line: 76, column: 108)

[Validation set: 'activiti-executable-process' | Problem: 'activiti-mailtask-no-recipient' | No recipient is defined on the mail activity - [Extra info :

```
processDefinitionId = CarritodeCompras2-Cliente | processDefinitionName  
= Default Process | | id = ServiceTask_3 | | activityName = Mail Task | |  
( line: 76, column: 108)
```

5.1.4. Resultados de Activiti

Luego de utilizar la herramienta Activiti en los tres casos de estudio comprendimos en que se enfoca el verificador y así deducimos ventajas y desventajas. Por más que en todos los modelos encontró errores y que verifica sesenta y tres condiciones (en el Anexo se detalla la lista completa de errores que puede detectar), es muy limitado en la búsqueda de errores. Tiene la ventaja de encontrar errores no visibles directamente en el modelo (por ejemplo, un “Script Task” no implementado) pero una actividad sin un flujo de secuencia no lo marca como algo erróneo. Otro tema no menor, que limita la representación de modelos de la realidad es que *Activiti Designer* no soporta el elemento de BPMN 2.0 “message flow”. Por esta razón, representar una colaboración que los “pools” no se puedan comunicar entre sí, carece de sentido. Al verificar modelos realizados en otro editor de modelos BPMN 2.0 que si soporta “message flow”, el verificador de Activiti lo omite y no genera ningún problema. En conclusión, es una buena herramienta para usar como complemento con otras

5.2. Análisis del Transformador y WoPeD

En esta Sección tomamos los mismos tres casos de prueba, utilizamos el transformador de BPMN 2.0 a Redes de Petri y luego los visualizamos con la herramienta WoPeD. Mostraremos los casos de prueba en el mismo orden que en la Sección anterior.

5.2.1. Caso de prueba 1: Agencia de Viajes

En la Figura 5.10 vemos el proceso que vamos a transformar y ejecutar. En los modelos que vamos a visualizar en WoPeD primero tenemos que verificar que el mapeo se haya realizado bien, es decir, que el transformador pudo resolver todos los elementos de BPMN 2.0 para representarlos en una Red de Petri. Podemos ver el mapeo del caso de prueba en la Figura 5.11 y comparar los flujos y actividades para comprobar que mantiene la coherencia. Al verificar que el mapeo se hizo correcto, procedemos a realizar el análisis semántico en WoPeD. Este modelo no debería generar ningún error.

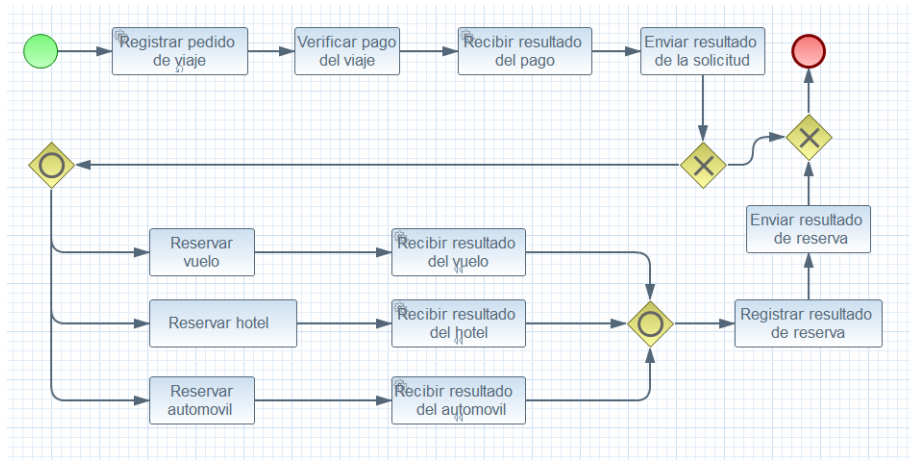


Figura 5.10: Agencia de viajes: Modelo inicial

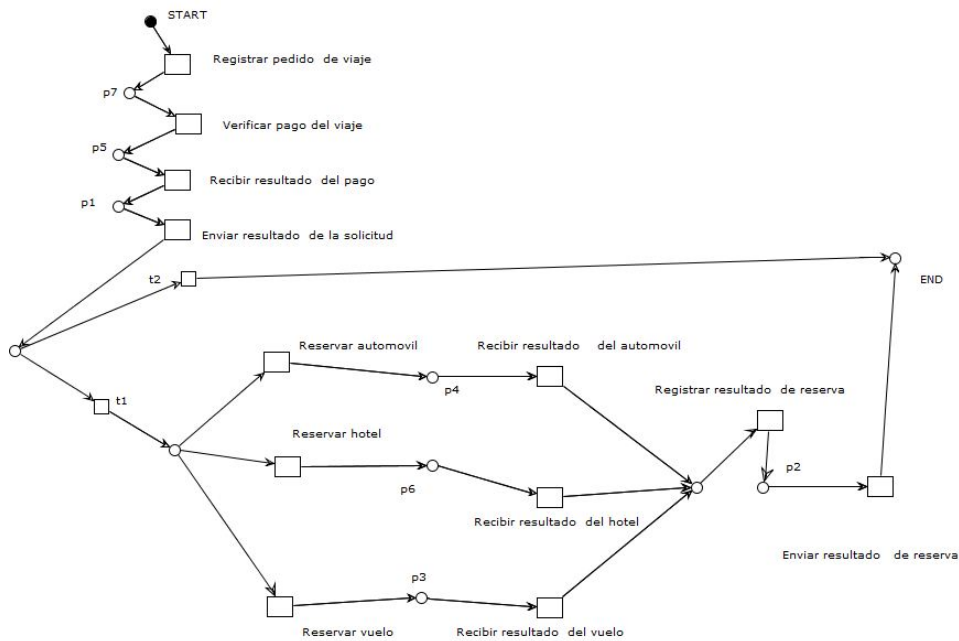


Figura 5.11: Agencia de viajes representado en Redes de Petri

En la Figura 5.12 vemos el resultado del análisis semántico que se realizó con la herramienta. El modelo está bien estructurado y cumple con la propiedad *Soundness*. De esta forma, culmina la verificación del caso de prueba 1.

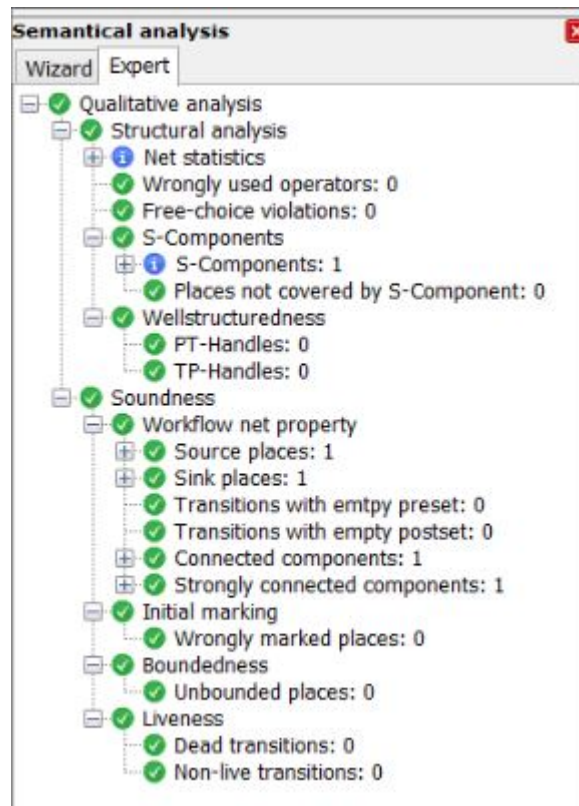


Figura 5.12: Resultado del análisis en WoPeD

5.2.2. Caso de prueba 2: Enviar Artículo

Al igual que en el caso de prueba anterior, empezamos mostrando en la Figura 5.13 el modelo inicial expresado en BPMN 2.0 y en la Figura 5.14 el mismo modelo de enviar artículo representado con Redes de Petri. Procedemos a comparar los modelos para verificar que se transformaron todos los elementos de forma correcta. Una vez comprobado que el mapeo se realizó de forma correcta, ejecutamos el analizador semántico (Figura 5.15) y observamos que en este caso estamos en presencia de errores (Sin haberle inyectado ninguno). WoPeD nos informa que el problema está en la propiedad *liveness* y viendo el modelo efectivamente podemos afirmar que hay dos grupos de transiciones y lugares que generan un loop sin fin.

El ciclo *Enviar inscripción del autor, Recibir resultado de la inscripción* genera un loop que evita que se cumpla *liveness*. Entre *Solicitar alojamiento* y *Recibir resultado de la solicitud alojamiento* tenemos el mismo problema. En estos casos hay que evaluar la realidad y pensar que pasaría, si es posible que el proceso quede en un ciclo infinito o si hay una manera clara y segura de que pueda finalizar. Claramente, si se quita la flecha de retorno al lugar

anterior, el modelo verifica *soundness* por otro lado, dejaría de representar la realidad. Para verificar que son esos los errores, vamos a quitar los loops y ejecutar el analizador semántico. Vemos una parte del modelo donde estaba la incidencia y el resultado del analizador en la Figura 5.16).

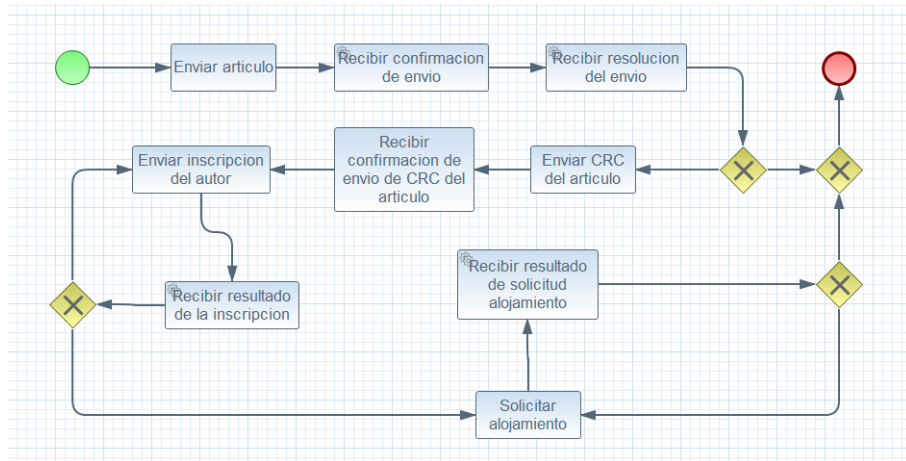


Figura 5.13: Enviar Artículo: Modelo inicial

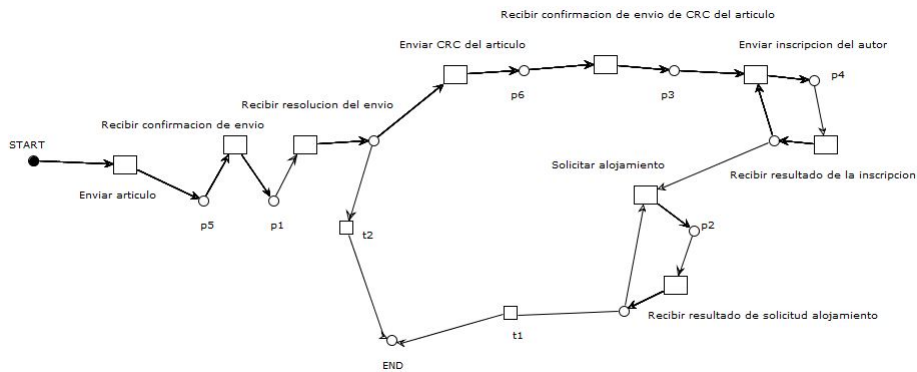


Figura 5.14: Enviar artículo representado en Redes de Petri

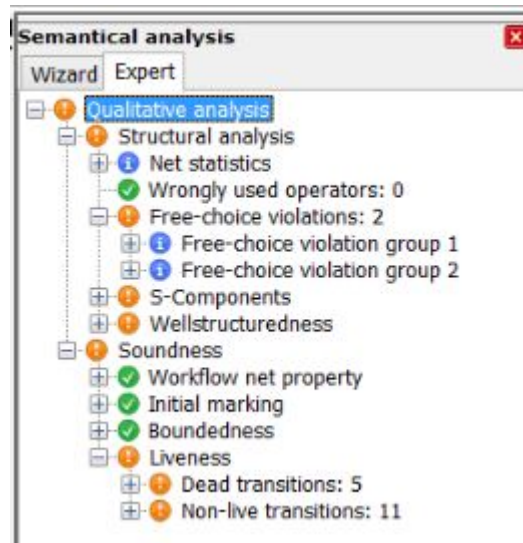


Figura 5.15: Resultado del análisis en WoPeD

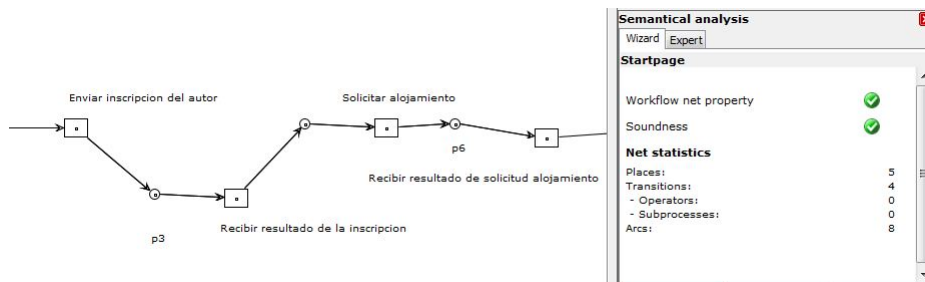


Figura 5.16: Resultado del análisis en WoPeD (sin loop)

En la Sección de resultados de la herramienta y transformador revisaremos casos como este donde el no cumplimiento de una propiedad no significa un error en el modelo.

5.2.3. Caso de prueba 3: Carrito de Compras

Retomando con este caso de prueba de la Figura 5.17, ahora le agregaremos una modificación que se observa en la Figura 5.18. Además de Recibir los artículos se le entregan cupones para un sorteo y como se tiene que realizar ambas actividades se utiliza un *parallel gateway* y para inyectarle un error cerramos el caso con un *exclusive gateway* de forma de generar un deadlock.

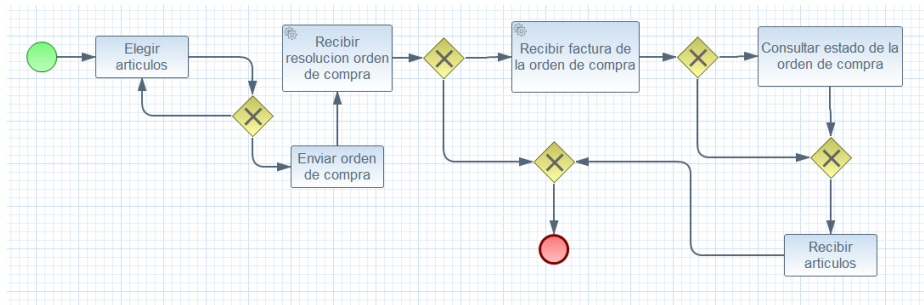


Figura 5.17: Carrito de compras: Modelo inicial

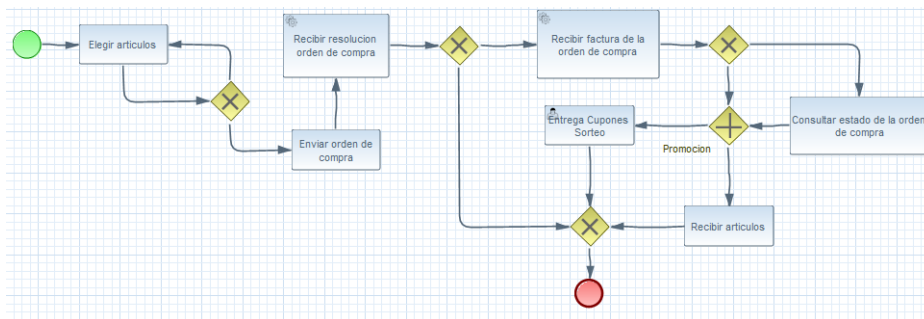


Figura 5.18: Modelo con errores

Evaluaremos el mapeo a la Red de Petri correspondiente viendo la Figura 5.19. Si comparamos las actividades y los flujos de secuencia de las representaciones del modelo podemos afirmar que la transformación se realizó de forma correcta. Posteriormente, ejecutamos el analizador de WoPeD (Figura 5.20) informándonos de la presencia de errores o no cumplimiento de alguna propiedad.

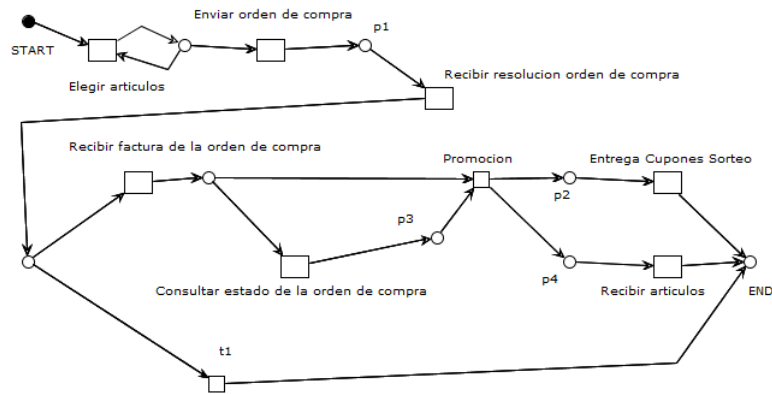


Figura 5.19: Carrito de compras (con error) representado en Redes de Petri

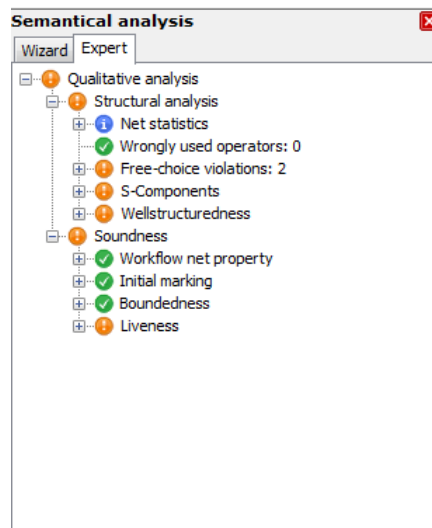


Figura 5.20: Resultado del análisis en WoPeD

Al igual que en el segundo caso de prueba, la propiedad que no se cumple es *Soundness*, pero en este caso por varias razones. Primero, se debe a que existe un loop en la transición de *Elegir artículos* ya que no existe una condición de parada explícita, por lo que no cumple *Liveness*. En la realidad podemos aceptar que se elijan todos los artículos que se desean y que no se puede realizar infinitamente, pero restringiéndonos al modelo es correcto que nos marque el no cumplimiento de la propiedad. Es un caso parecido al anterior, que si quisiéramos hacer el modelo tal que solo se elija un artículo, el analizador semántico no nos daría error, pero estamos alejándonos de la realidad que queremos representar. En segundo lugar, el *parallel gateway* de nombre “Promocion” genera un error con el último *exclusive gateway*. Esto

se debe a que *Entrega Cupones Sorteo* y *Recibir artículos* se realizan de forma paralela y por contrucción del modelo van a terminar en el mismo lugar con dos fichas entonces deja de cumplir *Boundedness*. Como en el análisis semántico de WoPeD no detalla que esa propiedad no se esté cumpliendo, arreglamos el loop de forma que solo se pueda elegir un único artículo y de esta forma comprobar si el error aparecía o no. Al realizar este cambio, vemos que WoPeD indica que hay dos propiedades que no se cumple *Boundedness* como ya se explicó y tampoco se cumple *Liveness* como era esperado por como se modificó el modelo. El error surge luego de *Recibir factura de la orden de compra* entre el *exclusive gateway* y el próximo *parallel gateway* de nombre “Promocion” ya que se genera un *Deadlock*.

5.2.4. Resultados del transformador y WoPeD

En esta Sección, repasaremos los resultados de los casos de estudio y resumiremos algunos temas importantes de esta herramientas. Evidentemente, al realizar un mapeo con otra representación y verificar propiedades sobre el nuevo modelo agrega una complejidad extra que con la herramienta Activiti no teníamos. Un mal mapeo ocurre cuando el transformador no soporta o no reconoce todos los elementos de BPMN 2.0 y será explicado en la próxima Sección con un ejemplo. La verificación de la propiedad *Soundness* es el interés central de WoPeD y en los casos de prueba dos y tres no se cumplía. Revisando el modelo y evaluando la realidad, llegamos a la conclusión de que el primer error de cada uno no es un error sino una advertencia para considerar. El fin del modelo es representar una realidad lo más precisa posible y este tipo de cosas ocurren seguido, pero si son manejadas con cuidado no deberían generar errores. Se podría mejorar agregando restricciones de tiempo o algún criterio de parada, para que sea más fiel a como ocurre el proceso en la realidad. El tercer caso de prueba agregando un error hizo que no se cumplieran dos propiedades: *Liveness* y *Boundedness*

Gracias a los casos de prueba pudimos observar que WoPeD es muy útil para identificar errores que en BPMN no podríamos haber identificado (al menos con una forma automatizada). Esto no significa que sea mejor que Activiti ya que cada una se enfoca en objetivos distintos. Una vez que el modelo se transformó, no podemos verificar ninguna propiedad o elemento relacionado con BPMN. De esta forma, podemos afirmar que el uso combinado de ambas herramientas nos brinda una verificación genérica que cubre bastantes aspectos importantes, teniendo además la posibilidad de agregar verificadores más específicos según el área de interés a trabajar.

5.3. Limitaciones y errores conocidos

En esta Sección detallaremos las limitaciones que fuimos encontrado mientras verificábamos el prototipo para dejarlas documentadas como “errores conocidos” y que puedan ser corregidas en versiones futuras.

La principal limitación que se nos presentó fue que Activiti Designer no tiene soporte gráfico para el elemento “Message flow” por lo que no es posible comunicar dos pools distintos en una colaboración. Hoy en día para que un modelo sea representativo de la realidad debería poder tener varios procesos comunicados entre sí y trabajando de forma paralela. Para realizar las pruebas, optamos por utilizar el *BPMN2 Modeler* para representar el diseño de los modelos BPMN 2.0 y que sí tiene soporte para “Message flow” [21]. Luego que representamos el modelo (ver Figura 5.21) lo mapeamos y abrimos en WoPeD (Figura 5.22). No genera ningún tipo de error pero el mapeo omite el elemento, quedando así dos procesos independientes.

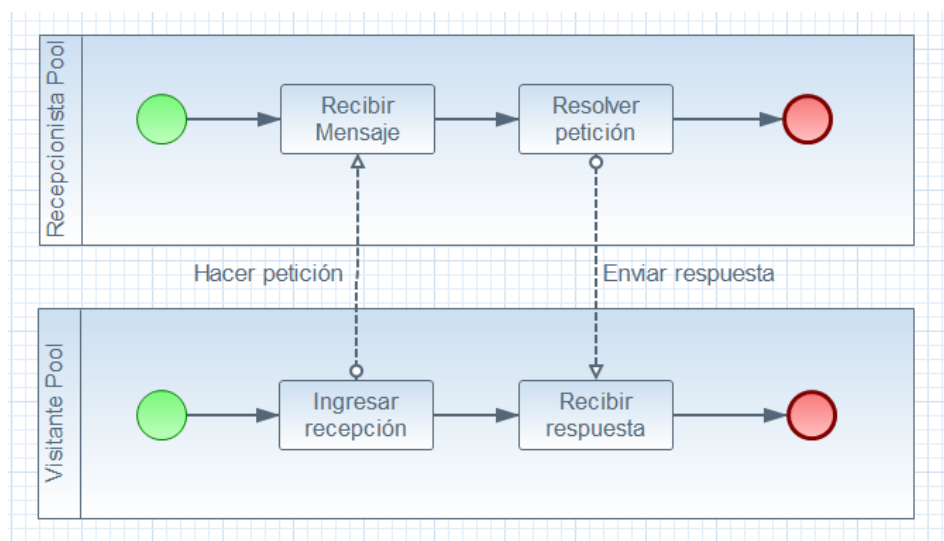


Figura 5.21: Ejemplo de colaboración representada con BPMN2 Modeler

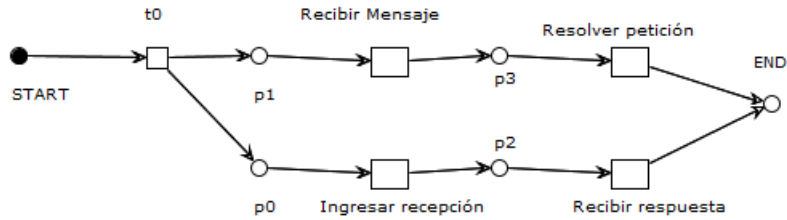


Figura 5.22: Ejemplo de Colaboración mapeada a Red de Petri

Al no tener la posibilidad de utilizar este elemento, nos vemos limitados a verificar procesos únicamente en un pool (aunque puede tener varios lanes). El Plug-in permite definir dos pools con sus procesos independientes y se mapea de forma correcta, pero al carecer de mensajes de comunicación carece de sentido, ya que agrega complejidad no necesaria para realizar la verificación.

Desarrollamos una colaboración con varios procesos independientes y realizamos varias pruebas que fueron bien mapeadas pero cuando quisimos analizarlas con WoPeD, nunca finalizaban. Los desarrolladores de la herramienta nos confirmaron que no soporta modelos de gran tamaño ya que su algoritmo no está bien optimizado. Por estas razones es que el Caso de Estudio que se implementa en el siguiente Capítulo está contenido en un pool.

Otros errores se generan al mapear algunos elementos de BPMN a Redes de Petri, por ejemplo, *Receive task*, *Manual task*, *Send task* hacen que no se pueda crear el archivo de la Red de Petri. *Call activiti* y *Timer event* se mapean sin generar error (en el mapeo) pero la Red no puede ser visualizada en WoPeD: “*Net is not workflow net and cannot be Beautified*”. La estructura visual que se realiza con el modelo en el lenguaje BPMN 2.0 se pierde en la transformación, pero de todas formas la verificación se puede realizar y WoPeD puede ordenarlo visualmente si es que la Red de Petri esta bien formada.

Encontramos un error en el analizador de WoPeD que fue explicado en el tercer caso de prueba donde tenía tres errores pero uno no se identificó hasta no solucionar otro antes.

Capítulo 6

Caso de Estudio

A continuación, presentaremos un Caso de Estudio mostrando un proceso y el procedimiento paso a paso para poder utilizar el prototipo de verificación. El objetivo de este Capítulo es demostrar como verificar errores en un modelo complejo de la realidad que abarca gran cantidad de elementos representativos de BPMN 2.0 con el prototipo de verificación que desarrollamos. En la Sección de *Introducción* explicaremos que representa el modelo en lenguaje natural y para que se utiliza. En *Instalación del Plug-in* explicamos los requerimientos de hardware y software necesarios para poder utilizarlo y el proceso completo para la instalación, de forma de estar en condiciones de empezar a probar la herramienta. En la Sección *Entorno de Verificación* mostramos la ubicación del Plug-in en Eclipse donde podemos encontrar los verificadores. Posteriormente, en *Verificación con BPMNValidator*, se introducen errores al modelo para demostrar cómo se utilizan los verificadores y se buscan soluciones para corregirlos. Por último se realiza un análisis del proceso planteado y las soluciones en *Análisis del Caso de Estudio*

6.1. Introducción

El proceso que vamos a utilizar en el Caso de Estudio se llama *Proceso Contrataciones* y su objetivo consta en la búsqueda y contratación de nuevo personal para una empresa. En la Figura 6.1 vemos la representación de su modelo en notación BPMN 2.0. Tres áreas de la empresa son las responsables de llevar a cabo este proceso; Sección RRHH, Secretaría y Supervisor.

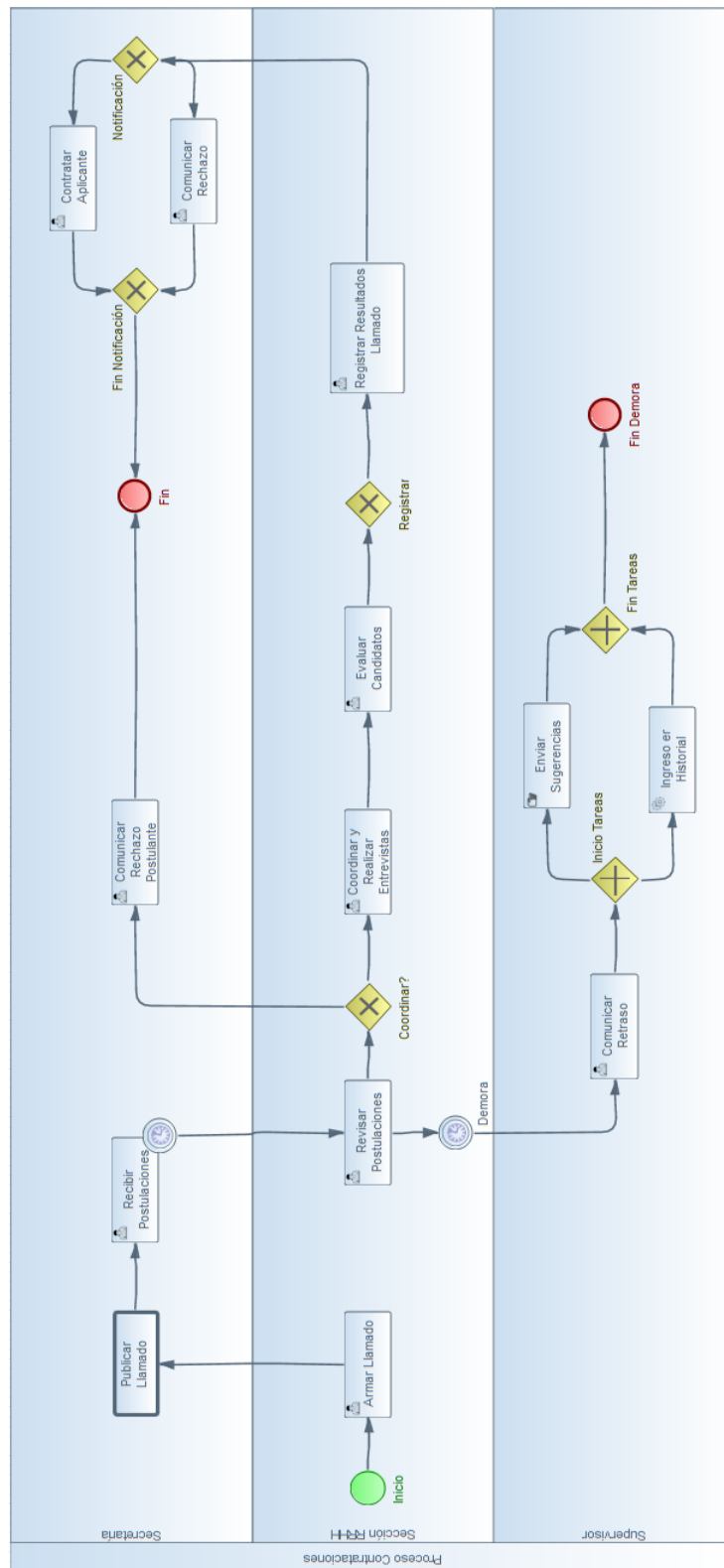


Figura 6.1: Caso de Estudio - Proceso Contrataciones

El proceso para contratación comienza en la Sección de RRHH donde una persona es la encargada de armar el llamado y pasarlo a Secretaría para su publicación. Se reciben postulaciones durante 15 días y cumplido el plazo, se manda la información a la Sección de RRHH donde se encargaran de revisarlas y evaluarlas. En caso que se demoren con la decisión se comunicaran el retraso al Supervisor (se detallará más adelante). Por otro lado, en caso de que los postulantes no cumplan con las expectativas se avisa a Secretaría para que se le comuniquen al postulante que no van a ser contratado, en otro caso, se coordina y se realizan las entrevistas en la misma Sección. Luego se evalúan los candidatos y en cualquier caso se registran los resultados del llamado. Por último, se pasan los resultados a Secretaría para que comuniquen el rechazo o realicen la contratación del aplicante. Si se produjo retraso en la revisión de postulantes, el Supervisor ingresa la demora al historial y envía las sugerencias correspondientes.

Por lo explicado en el Capítulo anterior, las herramientas tienen ciertas limitaciones que tenemos que contemplar para la realización del Caso de Estudio. Por ejemplo, las tareas y eventos no soportados, la realización del modelo en un solo pool y la limitación en el tamaño del proceso ya que WoPeD no logra analizarlos. En la Figura 6.2 vemos las adaptaciones realizadas del modelo que utilizaremos y al que le generaremos errores para mostrar el uso de las herramientas del prototipo.

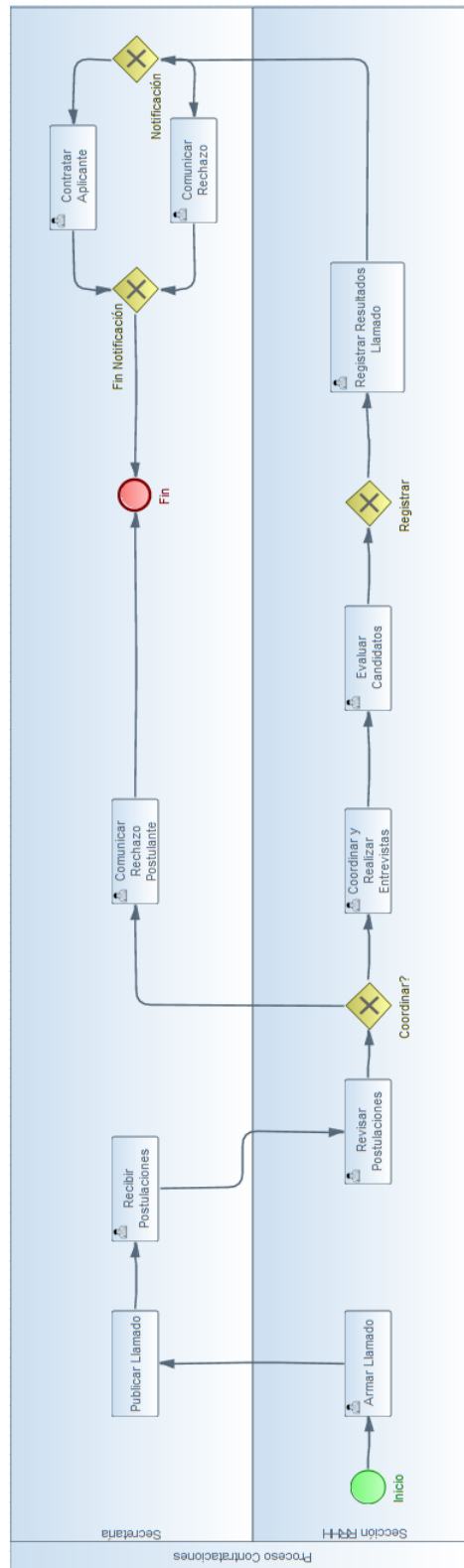


Figura 6.2: Caso de Estudio - Proceso Contrataciones Adaptado

6.2. Instalación y Entorno del Plug-in

Para utilizar la herramienta por primera vez se debe contar con una computadora con Windows 7 o posterior, en versiones de 32 o 64 bits, una carpeta con Eclipse versión Kepler [22], Java JDK versión 8u25 [56] y WoPeD versión 3.2.0 [67]. La instalación de los componentes necesarios propios de la herramienta se puede realizar con un instalador proporcionado en este desarrollo con el objetivo de facilitar este proceso. Como primer paso se debe ejecutar el instalador *BPMNValidator_Installer.exe* (Figura 6.3), que desplegará un menú donde se debe ingresar la ubicación del directorio de Eclipse donde se quiere instalar.

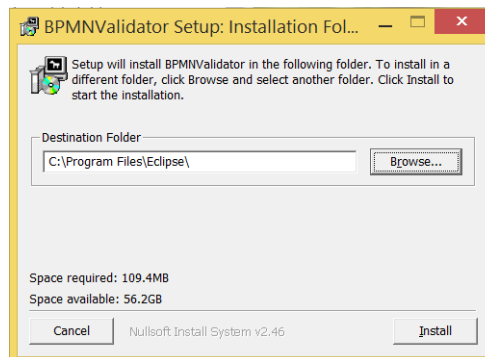


Figura 6.3: Instalador BPMNValidator Setup

Una vez ingresada la ruta (se puede utilizar el botón *Browse..* y buscar la carpeta en el sistema) seleccionando el botón *Install* comienza la instalación y cuando finaliza cerramos la ventana con *Close*. El proceso se puede visualizar en la Figura 6.4

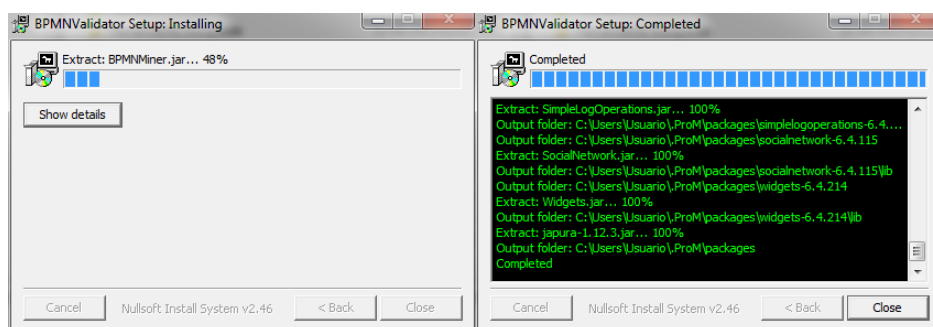


Figura 6.4: BPMNValidator - Proceso Instalación

Con el prototipo instalado y los requerimientos cumplidos, estamos en condiciones de ejecutar y Eclipse. Primero vamos a crear un proyecto Java : “*File → New → Project*” y seleccionamos “*Java → Java Project...*”, y le

asignamos de nombre “Caso de Estudio BPMNValidator” y le hacemos click a *Finish*. Importamos el archivo *CasoDeEstudioProcesoContratacion.bpmn* (se puede realizar arrastrando el archivo desde fuera del Eclipse a la carpeta del proyecto). Podemos ver el entorno del proyecto antes de comenzar con el Caso de Estudio en la Figura 6.5 y si se desea visualizar el modelo se recomienda utilizar *BPMN2 Modeler* que fue el utilizado para representar el diagrama [21].

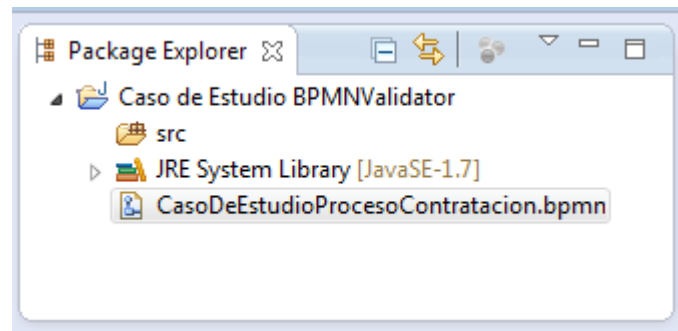


Figura 6.5: BPMNValidator - Entorno Eclipse

Al hacer click derecho sobre el archivo BPMN 2.0 (la extensión es *.bpmn) con la vista de Package Explorer, en el menú contextual hay una entrada donde dice Validators (Figura 6.6). En el submenú de Validators podemos encontrar los tipos de validadores que hay disponibles, en este caso solo contamos con los implementados en el Plug-in (Grafos y Otros), pero podría haber más si hubiera validadores externos agregados de otros tipos.

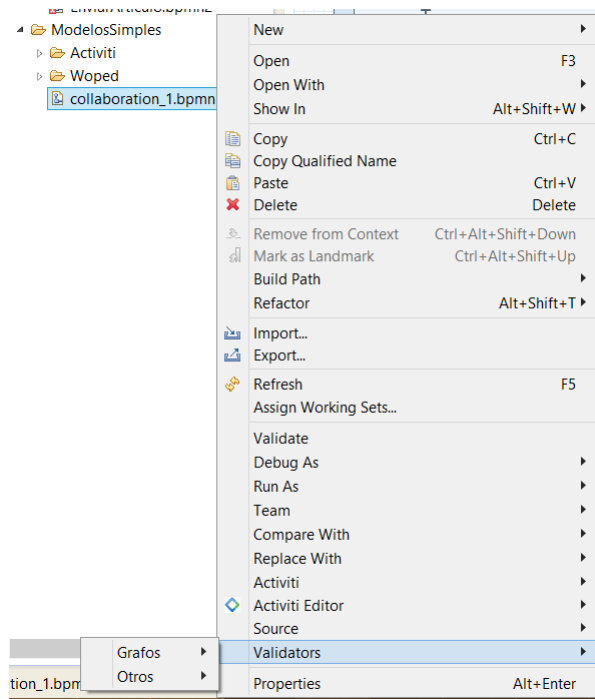


Figura 6.6: Menú contextual

6.3. Verificación con BPMNValidator

Para realizar el proceso de verificación dividimos la Sección en seis pasos de forma de plantear una metodología ordenada para el uso del prototipo.

6.3.1. Primer paso: Modelo Inicial

Iniciamos la verificación sobre el modelo de la Figura 6.2 mostrando los resultados de Activiti y WoPeD antes de generar los errores. Como mencionamos anteriormente, para utilizar Activiti hay que hacer click derecho en el archivo “*CasoDeEstudioProcesoContratacion.bpmn*”, veremos el menú contextual (Figura 6.6) y ahí seleccionamos “*Validators* → *Otros* → *Activiti*”. Como no introducimos ningún error, es de esperar que Activiti finalice sin advertencias. En la Figura 6.7 vemos la salida que se corresponde con lo que era esperado.

Message	Plug-in
Validator de Activiti finalizado	BPMNValidator
Validacion finalizada con exito	BPMNValidator
Validator de Activiti ejecutando...	BPMNValidator
Validator de Activiti finalizado	BPMNValidator

Figura 6.7: Resultado inicial de Activiti

Ejecutaremos ahora el otro validador interno del prototipo de manera similar al anterior pero seleccionando “*Validators* → *Grafos* → *Woped*”. En la Figura 6.8 vemos el modelo mapeado a Redes de Petri.

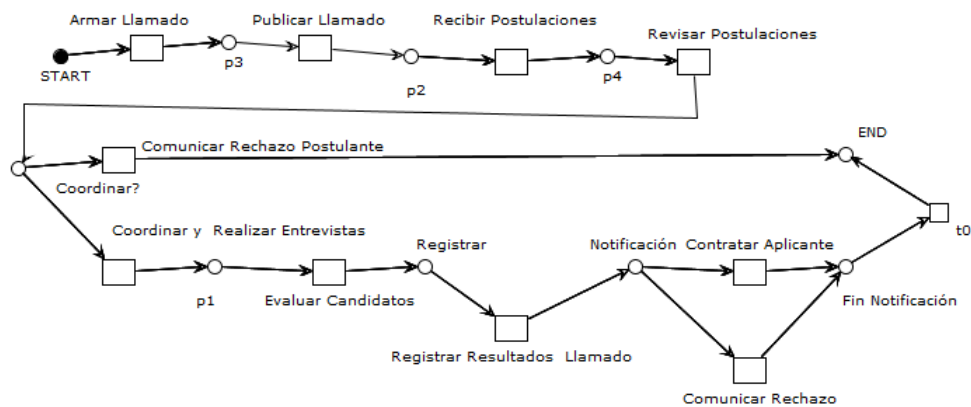


Figura 6.8: Proceso Contratación en Redes de Petri

Analizaremos el modelo seleccionando la herramienta *Semantical analysis* que se encuentra dentro del menú *Analyze* (Ver Figura 6.10) y el resultado esperado es que no contenga errores. Comprobamos que el transformador finalizó sin errores en la Figura 6.9 (en el *Error Log* de Eclipse) y que efectivamente el resultado del análisis es positivo en la Figura 6.11.

Message	Plug-in
Validator WoPeDfinalizado	BPMNValidator
Validator WoPeD ejecutando...	BPMNValidator

Figura 6.9: Error Log de Eclipse para WoPeD

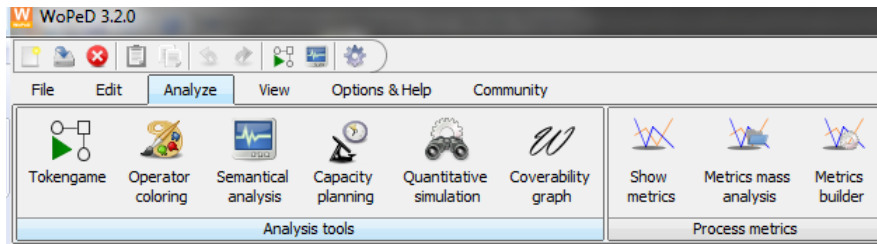


Figura 6.10: Herramientas en WoPeD

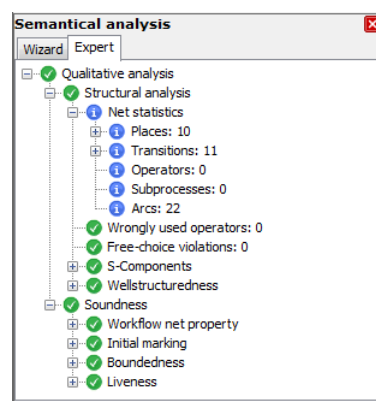


Figura 6.11: Resultado inicial de WoPeD

Quedó comprobado que comenzamos con un modelo sin errores detectados por las herramientas del prototipo.

6.3.2. Segundo paso: Errores generados

En este paso, agregaremos tres errores de distinta índole al modelo.

- **Primer Error:** Quitamos el flujo de secuencia entre el *exclusive gateway* de nombre “Fin Notificación” y el *end event* de nombre “Fin”.
- **Segundo Error:** Al flujo de secuencia de salida del *exclusive gateway* de nombre “Registrar” se le agregó un condicional de tipo *true*, es decir, que el proceso seguirá ese camino cuando evalúe de forma positiva.
- **Tercer Error:** Cambiamos el tipo de *gateway* de nombre “Notificación” de *exclusive* a *parallel*.

En la Figura 6.12 se representan de forma gráfica los errores que se agregaron al modelo.

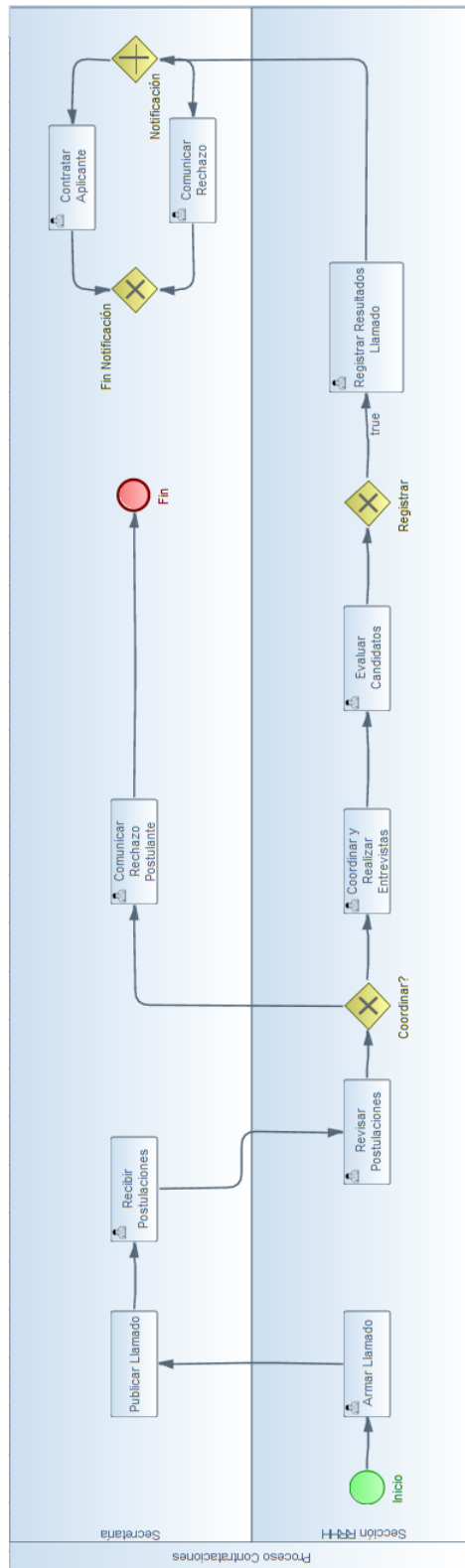


Figura 6.12: Proceso Contratación con Errores

6.3.3. Tercer paso: Errores detectados con Activiti y WoPeD

Con el mismo procedimiento realizado para utilizar los validadores en el *Primer paso*, en esta instancia lo realizaremos sobre el modelo con errores (Figura 6.12. Viendo los resultados de Activiti en la Figura 6.13, notamos que solo detecta dos de los tres errores inyectados.

Message	Plug-in
Validator de Activiti finalizado	BPMNValidator
[Validation set: 'activiti-executable-process' Problem: 'activiti-exclusive-gateway-condition-not-allowed-on-single-seq-flow']	BPMNValidator
[Validation set: 'activiti-executable-process' Problem: 'activiti-exclusive-gateway-no-outgoing-seq-flow']: Exclusive gateway	BPMNValidator
Errors while parsing:	BPMNValidator
Validator de Activiti ejecutando...	BPMNValidator

Figura 6.13: Resultado de Activiti sobre modelo con errores

Escribimos a continuación la salida completa de estos errores:

[Validation set: 'activiti-executable-process' | Problem: 'activiti-exclusive-gateway-no-outgoing-seq-flow']: Exclusive gateway has no outgoing sequence flow - [Extra info : processDefinitionId = Process_1 | processDefinitionName = Pool 1 Process | | id = ExclusiveGateway_4 | | activityName = Fin Notificación | | (line: 80, column: 71)

[Validation set: 'activiti-executable-process' | Problem: 'activiti-exclusive-gateway-condition-not-allowed-on-single-seq-flow']: Exclusive gateway has only one outgoing sequence flow. This is not allowed to have a condition. - [Extra info : processDefinitionId = Process_1 | processDefinitionName = Pool 1 Process | | id = ExclusiveGateway_2 | | activityName = Registrar | | (line: 93, column: 65)

El validador finalizó sin problemas y cuando se abrió WoPeD utilizamos la opción del menú “*View → Optimize layout*” pero no se permite realizar la tarea, informando que la Red de Petri no puede ser visualizada: “*Net is not workflow net and cannot be Beautified*”. Luego realizamos el análisis semántico como explicamos anteriormente y la salida se detalla en la Figura 6.14. El análisis nos brinda como resultado que no se cumple la propiedad *Soundness* y que la Red de Petri no está bien estructurada. En la próxima subsección se explicará cómo solucionar los errores encontrados.

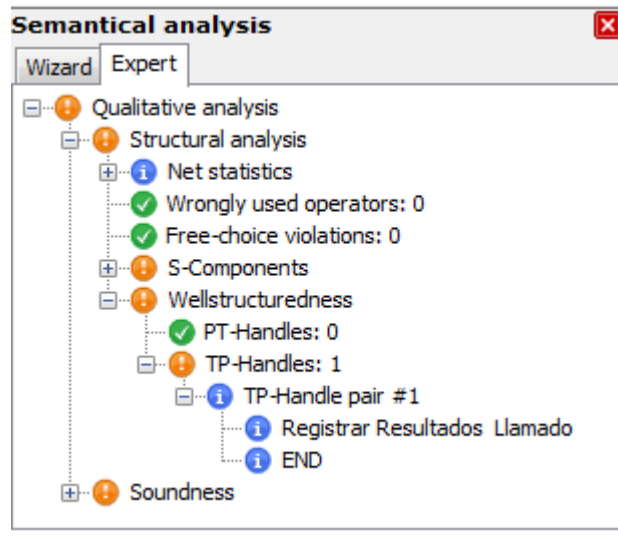


Figura 6.14: Resultado de WoPeD sobre modelo con errores

6.3.4. Cuarto paso: Soluciones planteadas para Activiti

Plantearemos como solucionar las dos correcciones justificando su veracidad. Activiti detectó el *Primer Error* que se explicó en la subsección 6.3.2, “*activiti-exclusive-gateway-no-outgoing-seq-flow*”. Es correcto que marque esta incidencia ya que al sacar la secuencia del flujo saliente de un *exclusive gateway* el modelo deja de ser consistente porque el proceso no llegó a un evento de Fin, ni tiene una secuencia por donde seguir. La manera de solucionarlo es agregarle una secuencia de flujo saliente como estaba en el modelo inicial del Caso de Estudio.

El otro error detectado es “*activiti-exclusive-gateway-condition-not-allowed-on-single-seq-flow*” que se corresponde con el *Segundo Error* explicado en la subsección 6.3.2. El único flujo de salida está condicionado y esto no puede pasar ya que en el caso del no cumplimiento de la condición el proceso no tiene otro flujo de salida para continuar. Si sabemos que el proceso siempre va a evaluar positivo en ese lugar, podemos sacar la condición ya que es el único flujo por el que puede seguir. Otra posibilidad, es que haya faltado otro flujo de salida con condicional negativo por lo que la forma de solucionarlo es volver a estudiar la realidad del Caso de Estudio. En este caso, como fue un error que se introdujo adrede, lo resolvemos quitando la condición.

La forma de solucionar estos errores puede ser mediante un editor gráfico o directamente sobre el código, como fue explicado en el Capítulo 5. Por último, la herramienta Activiti no fue capaz de detectar el *Tercer Error* explicado en la subsección 6.3.2. En la Sección 3.4 analizaremos los motivos que llevan a no detectar este tipo de errores.

6.3.5. Quinto paso: Soluciones planteadas para WoPeD

Como primer medida para resolver los errores es que cuando se transforme el modelo, realmente sea una Red de Petri. Viendo la salida del análisis semántico de WoPeD, el error se encuentra entre la tarea *Registrar Resultados Llamado* y el evento *end event*. Evidentemente este aviso se debe al *Primer Error* que se explica en la subsección 6.3.2. Por lo tanto, volvemos a abrir la herramienta Eclipse, con el editor gráfico *BPMN2 Modeler* agregamos ese flujo de secuencia y volvemos a ejecutar el validador de WoPeD como explicamos en las subsecciones anteriores. En la Figura 6.15 vemos el análisis semántico realizado luego de solucionarle el *Primer Error* y haciendo uso de la opción del menú “*View → Optimize layout*” podemos visualizar la Red de Petri de la misma forma que en el modelo sin errores de la Figura 6.8.

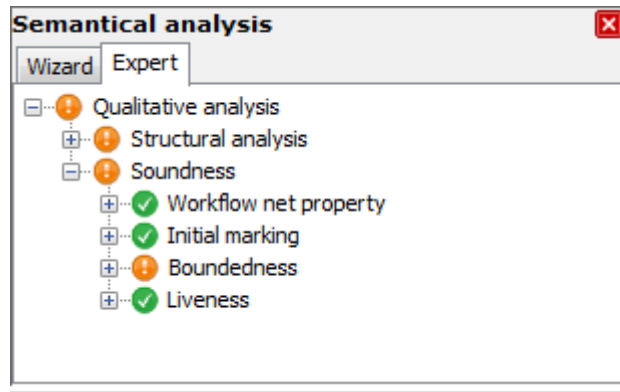


Figura 6.15: Resultado de WoPeD sobre modelo sin el Primer Error

Como podemos observar en los resultados del análisis, WoPeD todavía detecta que no se cumple la propiedad *Soundness*, en particular, *Boundedness*. Esto se debe al *Tercer Error*. Recordamos como se explicó en el Capítulo 3, que una Red de Petri se dice que está *k*-bounded (*k*-acotada) o simplemente bounded (acotada) si el número de fichas en cada lugar no excede un número finito *k* para ningún marcado alcanzable desde el marcado inicial. El caso particular de que el valor constante de *k* esté definido en uno, es la propiedad *Safety* y significa que el número de fichas en cada estado nunca va a ser mayor a uno.

Utilizando un *parallel gateway* queremos representar que se va a realizar la tarea *Contratar Aplicante* y *Comunicar Rechazo* de forma paralela, por lo que en el modelo de Red de Petri se entiende que llega una ficha al *parallel gateway* de nombre “Notificación” (representado por un estado) y sale una ficha a cada tarea (transición). Al unir ambas a un *exclusive gateway* (otro estado, de nombre “Fin Notificación”) lo que sucede es que en el estado *END* van a llegar dos fichas, violando así la propiedad *Boundedness*. Como en la

realidad no tiene sentido realizar estas dos tareas de forma simultánea, el error está en que ese estado en realidad tenía que ser un *exclusive gateway* como estaba originalmente. De esta forma cumple la propiedad *Soundness*.

Este validador no detectó el *Segundo Error* de la condición en un flujo de secuencia. En la Sección 6.4 analizaremos esta situación.

6.4. Análisis del Caso de Estudio

Luego de detectar y solucionar los errores planteados con ambos validadores podemos formular algunas conclusiones al respecto. De los tres errores agregados al modelo del Caso de Estudio, ningún validador logró detectarlos en su totalidad; Activiti no reconoció el *Tercer Error* y el transformador y WoPeD no detectaron el *Segundo Error*. Esto se debe al origen de cada tipo de error. El tercer error es sobre una propiedad de Grafos (como definimos en el Capítulo 3) y se descubre en tiempo de ejecución realizando una simulación del modelo, en cambio, el segundo error es una condición utilizada en el modelo BPMN 2.0 que al mapearlo no se considera, es por esto que Activiti la reconoce y WoPeD no.

En resumen:

- **Primer Error:** Fue detectado por ambos validadores.
- **Segundo Error:** Solo Activiti detectó este error.
- **Tercer Error:** Solo WoPeD detectó este error.

Con esto no decimos que uno validador sea mejor que otro, sino que cada uno verifica distintos aspectos de un modelo y en conjunto se complementan para realizar una verificación mucho más completa, destacando así la utilidad de un prototipo integrado de verificación.

Capítulo 7

Conclusiones y trabajo a futuro

Se presentan a continuación las conclusiones finales del proyecto en base al trabajo realizado y propuestas para posibles trabajos a futuro.

7.1. Conclusiones

El proyecto se inició mediante el estudio del contexto en el que se iba a trabajar, con el objetivo de obtener el conocimiento necesario para entender el ciclo de vida de un proceso, las etapas, la importancia del modelado y la necesidad de verificar los Procesos de Negocio. Con esta base de conocimiento, se realizó una revisión sistemática de la literatura enfocándose en la identificación de métodos, técnicas, lenguajes y tecnologías utilizadas para la verificación de los Procesos de Negocio. Como resultado de esta etapa, se obtuvo un relevamiento detallado de la verificación de procesos con una clasificación de los diferentes trabajos existentes de acuerdo a diversos aspectos de interés, realizando así un aporte a esta área de estudio y cumpliendo con dos de los objetivos planteados.

Se realizó un entorno integrador de verificación de Procesos de Negocio en Eclipse, con tres herramientas conocidas; Activiti, ProM y WoPeD. Fueron elegidas de tal forma que Activiti realiza la verificación dentro del entorno de Eclipse, ProM en un transformador de modelos (BPMN 2.0 a Redes de Petri) y por último, WoPeD como una herramienta externa de análisis de Redes de Petri. Por otro lado, como complemento del entorno, se brindan las herramientas necesarias mediante un punto de extensión para que se puedan seguir agregando validadores, de la misma forma que fueron agregados los anteriormente nombrados y verificar otro tipo de propiedades. De esta forma, quedó demostrado que el entorno integrador de verificación es viable y de fácil extensión por terceros para utilizarlo sobre cualquier Proceso de Negocio en cualquier lenguaje de modelado.

La Revisión Sistemática fue un paso previo fundamental para la realización del prototipo porque nos brindó información para comprender el contexto y sacar conclusiones de las necesidades actuales para la verificación de Procesos de Negocio. Además, nos advirtió la falta de herramientas para verificación en todas las áreas, que pudimos comprobar mientras desarrollamos el prototipo. La integración de herramientas no fue tarea fácil debido a la escasa documentación sobre los prototipos desarrollados. Fue necesaria y de mucha ayuda la comunicación con los desarrolladores de dos de las tres herramientas que utilizamos para finalizar el desarrollo. Ambos nos guiaron en el camino a la solución de los problemas encontrados.

Consideramos muy ventajoso el uso del prototipo por sus cualidades de integrar cualquier tipo de lenguaje y la posibilidad de utilizar cualquier verificador. De todas formas, falta avanzar mucho en el tema para poder verificar modelos colaborativos. Es un gran avance pero con las limitaciones ya mencionadas.

Para cumplir con el último objetivo, definimos un Caso de Estudio para evaluar el prototipo desarrollado. Se le introdujeron errores al modelo, que sirvieron para validar el desarrollo realizado además de probar las herramientas utilizadas. En el Capítulo 6 se documentó paso a paso el uso del verificador. Por más que no se probaron todos los elementos de BPMN 2.0, se tomó un modelo complejo y representativo, que sumado con los casos de prueba del Capítulo 5 nos permiten afirmar el buen funcionamiento de la herramienta. Como complemento se detallaron las limitaciones que fueron surgiendo para proponer una solución a futuro.

Podemos concluir de esta manera que se lograron cumplir los objetivos planteados anteriormente.

Los resultados obtenidos en el proyecto realizado se podrán utilizar a nivel académico para seguir profundizando en nuevas investigaciones.

7.2. Trabajo a futuro

Plantearemos algunas ideas sobre las cuales se pueden realizar futuros trabajos. Algunas refieren a mejoras de herramientas, actualización de la revisión sistemática, extensión del prototipo de verificación y actividades en general.

1. Extender el prototipo realizado agregando validadores externos con el objetivo de verificar otras propiedades o lenguajes.
2. Realización de una nueva revisión sistemática en uno o dos años con el objetivo de actualizar la lista de artículos de interés, y clasificarlos con el mismo criterio.

3. Estudio de la verificación de Procesos de Negocio en otras etapas, como por ejemplo, en Ejecución o Evaluación.
4. Realizar un análisis más profundo en la verificación de orquestaciones, en la etapa de análisis y diseño enfocándose en propiedades de grafos. Al ser el foco principal de la mayoría de los artículos estudiados (según la revisión sistemática), resultaría de suma importancia una guía de pautas de las propiedades más importantes que tenemos que verificar en dichas condiciones, como inicio de un método sistemático de verificación.
5. Profundizar en el estudio de verificación de colaboraciones de Procesos de Negocio para poder utilizar modelos mas complejos de la realidad.
6. Integrar la herramienta de Eclipse *BPMN2 Modeler* como complemento al validador de Activiti, ademas agrega indicador visual de errores.
7. Extender Activiti para que soporte visualización de errores y soporte para elementos BPMN 2.0 que aún no están desarrollados, como por ejemplo, *Message Flow*.

Bibliografía

- [1] W.M.P. van der Aalst, A. Hirnschall, and H.M.W. Verbeek. An Alternative Way to Analyze Workflow Graphs. In A. Banks-Pidduck, J. Mylopoulos, C.C. Woo, and M.T. Ozsú, editors, *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE'02)*, volume 2348 of *Lecture Notes in Computer Science*, pages 535–552. Springer-Verlag, Berlin, 2002.
- [2] W.M.P. van der Aalst, N. Lohmann, and M. La Rosa. Ensuring correctness during process configuration via partner synthesis. *Information Systems*, 37(6):574–592, 2012.
- [3] Faisal Abouzaid and John Mullins. Model-checking web services orchestrations using bp-calculus. *Electron. Notes Theor. Comput. Sci.*, 255:3–21, November 2009.
- [4] Luciano Baresi, Domenico Bianculli, Carlo Ghezzi, Sam Guinea, and Paola Spoletini. Validation of web service compositions. *IET Softw.*, 1(6):219–232, December 2007.
- [5] Cristiano Barros and Mark Song. Automatized checking of business rules for activity execution sequence in workflows. *JSW*, 7(2):374–381, 2012.
- [6] H.H. Bi and J.L. Zhao. Applying Propositional Logic to Workflow Verification. *Information Technology and Management*, 5(3-4):293–318, 2004.
- [7] BPMI. Process management initiative, <http://www.bpmi.org/>, September 2014.
- [8] Daniel Calegari and Nora Szasz. Verification of model transformations. *Electron. Notes Theor. Comput. Sci.*, 292:5–25, March 2013.
- [9] Haijun Cao, Hai Jin, Song Wu, and Shadi Ibrahim. Petri net based grid workflow verification and optimization. *The Journal of Supercomputing*, 66(3):1215–1230, 2013.

- [10] Junwei Cao, Fan Zhang, Ke Xu, Lianchen Liu, and Cheng Wu. Formal verification of temporal properties for reduced overhead in grid scientific workflows. *J. Comput. Sci. Technol.*, 26(6):1017–1030, 2011.
- [11] Jinjun Chen and Yun Yang. Key research issues in grid workflow verification and validation. In *Proceedings of the 2006 Australasian Workshops on Grid Computing and e-Research - Volume 54*, ACSW Frontiers '06, pages 97–104, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.
- [12] Vasa Curcin, Moustafa Ghanem, and Yike Guo. Analysing scientific workflows with computational tree logic. *Cluster Computing*, 12(4):399–419, 2009.
- [13] Manu De Backer, Monique Snoeck, Geert Monsieur, Wilfried Lemahieu, and Guido Dedene. A scenario-based verification technique to assess the compatibility of collaborative business processes. *Data Knowl. Eng.*, 68(6):531–551, June 2009.
- [14] Gero Decker and Mathias Weske. Behavioral consistency for b2b process integration. In *In Proceedings of the 19th International Conference on Advanced Information Systems Engineering*, pages 81–95. Springer Verlag, 2007.
- [15] Andrea Delgado, Francisco Ruiz, Ignacio García Rodríguez de Guzmán, and Mario Piattini. Minerva: Model driven and service oriented framework for the continuous business process improvement and related tools. In *ICSOC/ServiceWave Workshops*, pages 456–466, 2009.
- [16] Activiti Designer. Activiti bpm platform, <http://activiti.org/>, September 2014.
- [17] Activiti Designer. Activiti components, <http://activiti.org/components.html>, September 2014.
- [18] Activiti Designer. Chapter 4. the activiti api, <http://activiti.org/userguide/#chapterapi>, September 2014.
- [19] Activiti Designer. Chapter 8. bpmn 2.0 constructs, <http://activiti.org/userguide/index.html#bpmnconstructs>, September 2014.
- [20] Yanhua Du, PengCheng Xiong, Yushun Fan, and Xitong Li. Dynamic checking and solution to temporal violations in concurrent workflow processes. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 41(6):1166–1181, 2011.

- [21] Eclipse. Bpmn2 modeler, <http://eclipse.org/bpmn2-modeler/>, September 2014.
- [22] Eclipse. Eclipse kepler sr2, <http://www.eclipse.org/downloads/packages/release/kepler/sr2>, September 2014.
- [23] Eclipse. Welcome to Eclipse, <http://archive.eclipse.org/eclipse/downloads/drops/r-3.1-200506271435/org.eclipse.platform.doc.isv.3.1.pdf.zip>, September 2014.
- [24] Johann Eder, Euthimios Panagos, and Michael Rabinovich. Time constraints in workflow systems. In *In Proceedings of the 11th International Conference on Advanced Information Systems Engineering*, pages 286–300. Springer-Verlag, 1999.
- [25] The YAWL Foundation. Yawl- user manual version 2.2, <http://www.yawlfoundation.org/manuals/yawlusermanual2.2.pdf>, September 2014.
- [26] Rücker B y Hitpass B. Freund J. *BPMN 2.0 Manual de referencia y guía rápida*, 2011 edition, 2011.
- [27] H. Groefsema and D. Bucur. A survey of formal business process verification: From soundness to variability. In *Proceedings of the Third International Symposium on Business Modeling and Software Design*, page 198, 2013.
- [28] Wang Guangqi, Wang Juying, Wang Yan, Song Baoyan, and Yu Ge. A novel verification approach of workflow schema. *Wuhan University Journal of Natural Sciences*, 11(5):1292–1296, 2006.
- [29] Arthur Hofstede, Wil van der Aalst, Michael Adams, and Nick Russell. *Modern Business Process Automation: YAWL and Its Support Environment*. Springer Publishing Company, Incorporated, 1st edition, 2009.
- [30] Yu Huang, Hanpin Wang, Wen Zhao, and Chunxiang Xu. A practical method to analyze workflow logic models. *Concurrency and Computation: Practice and Experience*, 20(1):41–60, 2008.
- [31] G. Vossen J. Wainer, M. Weske. Scientific workflow systems. In *Proc. NSF Workshop on Workflow and Process Automation: State-of-the-art and Future Directions*, Athens, GA, May 1996.
- [32] A. A. Kalenkova. Application of if-conversion to verification and optimization of workflows. *Programming and Computer Software*, 36(5):276–288, 2010.

- [33] Stefan Kühne, Heiko Kern, Volker Gruhn, and Ralf Laue. Business process modeling with continuous validation. *Journal of Software Maintenance and Evolution: Research and Practice*, 0(22):547, 2010.
- [34] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.
- [35] Barbara A. Kitchenham, David Budgen, and O. Pearl Brereton. Using mapping studies as the basis for further research - a participant-observer case study. *Inf. Softw. Technol.*, 53(6), June 2011.
- [36] Máté Kovács and László Gönczy. Simulation and formal analysis of workflow models. *Electron. Notes Theor. Comput. Sci.*, 211:221–230, April 2008.
- [37] Máté Kovács, László Gönczy, and Dániel Varró. Formal analysis of bpmel workflows with compensation by model checking. *International Journal of Computer Systems and Engineering*, 23(5):0, 2008.
- [38] Vitus S. W. Lam. Formal analysis of bpmn models: a nusmv-based approach. *International Journal of Software Engineering and Knowledge Engineering*, 0:987–1023, 2010.
- [39] Ralf Laue and Ahmed Awad. Visual suggestions for improvements in business process diagrams. *J. Vis. Lang. Comput.*, 22(5):385–399, 2011.
- [40] Giorgio Levi and Franco Sirovich. Generalized AND/OR graphs. *Artif. Intell.*, 7(3), 1976.
- [41] Hongchen Li and Yun Yang. Dynamic checking of temporal constraints for concurrent workflows. *Electron. Commer. Rec. Appl.*, 4(2):124–142, July 2005.
- [42] JianQiang Li, Yushun Fan, and MengChu Zhou. Timing constraint workflow nets for workflow analysis. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 33(2):179–193, 2003.
- [43] Shiyong Lu, Arthur Bernstein, and Philip Lewis. Automatic workflow verification and generation. *Theor. Comput. Sci.*, 353(1):71–92, March 2006.
- [44] C. Luo, S. Qin, and Z. Qiu. Verifying bpmel-like programs with hoare logic. *Frontiers of computer science in China.*, 2(4):344–356, December 2008.

- [45] Gang Ma, Kehe Wu, Tong Zhang, and Cui Shuo. A method of workflow model structure verification based on graph theory. *JCIT: Journal of Convergence Information Technology*, 7(17):350–356, 2012.
- [46] Jan Mendling, Gustaf Neumann, and Markus Nüttgens. Yet another event-driven process chain - modelling workflow patterns with yepcs. *Enterprise Modelling and Information Systems Architectures*, 1(1):03–13, 2005.
- [47] Luis Eduardo Mendoza, Manuel I. Capel, and María A. Pérez. Conceptual framework for business processes compositional verification. *Information & Software Technology*, 54(2):149–161, 2012.
- [48] Marco Montali, Paolo Torroni, M. Alberti, Federico Chesani, Evelina Lamma, and P. Mello. Abductive logic programming as an effective technology for the static verification of declarative business processes. *Fundamenta Informaticae*, 102:325–361, 11/2010 2010.
- [49] S. Morimoto. A survey of formal verification for business process modeling. In M. Bubak et al., editor, *Proc. ICCS 2008*, volume 5102 of *LNCS*, page 514. Springer, 2008.
- [50] Arindam Mukherjee, Anup K. Sen, and Amitava Bagchi. The representation, analysis and verification of business processes: a metagraph-based approach. *Information Technology and Management*, 8(1):65–81, 2007.
- [51] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
- [52] Alex Norta and Rik Eshuis. Specification and verification of harmonized business-process collaborations. *Information Systems Frontiers*, 12(4):457–479, 2010.
- [53] OASIS. Web services business process execution language (wsbpel) v2.0, <https://www.oasis-open.org/standards#wsbpelv2.0>, September 2014.
- [54] Object Management Group. Business process model and notation (bpmn 2.0), <http://www.omg.org/spec/bpmn/2.0/pdf/>, jan 2011.
- [55] Object Management Group. Introduction to uml 2.0, http://www.omg.org/news/meetings/workshops/MDA-SOA-WS_manual/00-t4_matthews.pdf, September 2014.
- [56] Oracle. Java jdk 8u25, <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>, September 2014.

- [57] Susanne Patig and Manuela Stolz. A pattern-based approach for the verification of business process descriptions. *Information & Software Technology*, 55(1):58–87, 2013.
- [58] James L. Peterson. Petri nets. *ACM Comput. Surv.*, 9(3), September 1977.
- [59] Artem Polyvyanyy, Matthias Weidlich, and Mathias Weske. Connectivity of workflow nets: the foundations of stepwise verification. *Acta Informatica*, 48(4):213–242, 2011.
- [60] ProM. Process mining, <http://www.processmining.org/prom/start>, September 2014.
- [61] ProM. Prom tools, <http://www.promtools.org/doku.php>, September 2014.
- [62] Fazle Rabbi, Hao Wang, Wendy MacCaull, and Adrian Rutle. A model slicing method for workflow verification. *Electr. Notes Theor. Comput. Sci.*, 295:79–93, 2013.
- [63] Gerd Saurer, Josef Schiefer, and Alexander Schatten. Testing event-driven business processes. *Journal of Computers*, 1(7):69–80, 2006.
- [64] Natalia Sidorova, Christian Stahl, and Nikola Trcka. Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. *Inf. Syst.*, 36(7):1026–1043, 2011.
- [65] H. Smith and P. Fingar. *Business Process Management: The Third Wave*. Meghan Kiffer Press, 2006.
- [66] Haiyang Sun and Jian Yang. Cobtx-net: A model for reliability verification of collaborative business transaction. In Arthur ter Hofstede, Boualem Benatallah, and Hye-Young Paik, editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 220–231. Springer Berlin Heidelberg, 2008.
- [67] Andreas Eckleder Thomas Freytag, Phillipp Ohl. Woped 3.2.0, http://woped.dhbw-karlsruhe.de/woped/?page_id=22, September 2014.
- [68] Andreas Eckleder Thomas Freytag, Phillipp Ohl. Workflow petri net designer (WoPeD), <http://woped.dhbw-karlsruhe.de/woped/>, September 2014.
- [69] Wil van der Aalst, Arthur ter Hofstede, and Mathias Weske. Business process management: A survey. In Mathias Weske, editor, *Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, page 1019. Springer Berlin, Berlin, Heidelberg, May 2003.

- [70] Wil M. P. van der Aalst. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA, 2002.
- [71] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, Incorporated, 1st edition, 2011.
- [72] Wil M. P. van der Aalst and Christian Stahl. *Modeling Business Processes - A Petri Net-Oriented Approach*. Cooperative Information Systems series. MIT Press, 2011.
- [73] A.S. Varosyan. Algorithm for formal verification of business process templates. *Cybernetics and Systems Analysis*, 47(2):228–240, 2011.
- [74] H. M. W. Verbeek, J. C. A. M. Buijs, B. F. van Dongen, and Wil M. P. van der Aalst. Prom 6: The process mining toolkit. In *Proc. of BPM Demonstration Track 2010*, volume 615. CEUR-WS.org, 2010.
- [75] Jianmin Wang Wang, Zhaoxia, Xiaochen Zhu, and Lijie Wen. Verification of workflow nets with transition conditions. *Journal of Zhejiang University - Science C*, 13(7):483–509, 2012.
- [76] Ingo Weber, Jörg Hoffmann, and Jan Mendling. Beyond soundness: on the verification of semantic business process models. *Distributed and Parallel Databases*, 27(3):271–343, 2010.
- [77] Mathias Weske. *Business process management concepts, languages, architectures*. Springer, 1 edition, November 2007.
- [78] WfMC. Xpdl, <http://www.xpdl.org/documents.html>, September 2014.
- [79] M.T. Wynn, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Verifying Workflows with Cancellation Regions and OR-joins: An Approach Based on Reset Nets and Reachability Analysis. In S. Dustdar, J.L. Fiadeiro, and A. Sheth, editors, *International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 389–394. Springer-Verlag, Berlin, 2006.
- [80] M.T. Wynn, H.M.W. Verbeek, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Business process verification: Finally a reality! *Business Process Management Journal*, 15(1):74–92, 2009.
- [81] Qingtian Zeng, Huaiqing Wang, Dongming Xu, Hua Duan, and Yanbo Han. Conflict detection and resolution for workflows constrained by resources and non-determined durations. *Journal of Systems and Software*, 81(9):1491–1504, 2008.

- [82] Haiping Zha, Wil M. P. van der Aalst, Jianmin Wang 0001, Lijie Wen, and Jianguang Sun. Verifying workflow processes: a transformation-based approach. *Software and System Modeling*, 10(2):253–264, 2011.
- [83] Hai Zhuge, To yat Cheung, and Hung keng Pung. A timed workflow process model. *Journal of Systems and Software*, 55(3):231 – 243, 2001.

Apéndice A

Información adicional sobre Redes de Petri

A.1. Mapeo de conceptos de flujo de trabajo en Redes de Petri

Usando un proceso en un sistema de manejo de flujo de trabajo, podemos indicar de qué manera una categoría particular de casos debe ser tratada. El proceso define que tareas deben ser llevadas a cabo. Como también información de las tareas a ser realizadas, un proceso además tiene información sobre condiciones. De esta manera, define el orden de que tareas se necesitan realizar. También es posible usar procesos previamente definidos para crear un proceso de mayor envergadura. Un proceso puede así consistir de más de un subproceso, además de las tareas y condiciones. Una Red de Petri que ilustra un proceso debe tener una entrada (Un lugar sin arcos incidentes) y una salida (Un lugar sin arcos salientes). Se mapean las condiciones como lugares y las tareas como transiciones. Esto se desprende de que las transiciones son componentes activos en una Red de Petri y los lugares son componentes pasivos.

Para comprender con mayor profundidad como se puede representar un proceso como una Red de Petri, volveremos al ejemplo del proceso de manejo de reclamos. La Figura A.1 muestra cómo se puede representar este proceso con una Red de Petri.

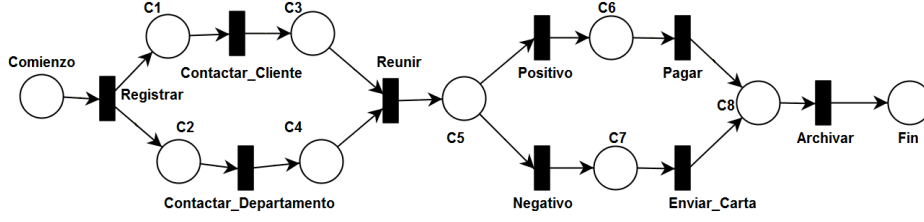


Figura A.1: Proceso reclamo

Cada una de las tareas *Registrar*, *Contactar_Cliente*, *Contactar_Departamento*, *Pagar*, *Enviar_Carta* y *Archivar* son modelados utilizando una transición. La evaluación del reclamo es modelada usando dos transiciones: *Positivo* y *Negativo*. La transición *Positivo* corresponde con una decisión positiva, mientras que la transición *Negativo* con una decisión negativa. Los lugares *Comienzo* y *Fin* se corresponden con el comienzo y el final del proceso modelado. Los otros lugares se corresponden con condiciones que no se cumplen con todos los casos en progreso. Las condiciones juegan dos roles importantes: por un lado aseguran que las tareas se desarrollen en el orden correcto, y la otra es que el estado del caso pueda ser establecido. El lugar *C8*, por ejemplo, asegura que un reclamo sea archivado solamente una vez que haya sido abordada completamente. También corresponde al estado que existe entre que el reclamo ha sido completamente manejado y su archivado.

A.2. Enrutamiento de fichas

Algunas tareas pueden ser opcionales, esto implica que puede haber tareas que deban ser llevadas a cabo solo para un subconjunto de los casos. El orden en que las tareas son ejecutadas también puede variar caso a caso. Enrutando un caso a lo largo de un numero de tareas, nosotros podemos determinar cuáles tareas necesitan ser llevadas a cabo y en qué orden. Para esto se reconocen cuatro construcciones básicas de enrutamiento.

A.2.1. Enrutamiento secuencial

Nos referimos a la ejecución secuencial de tareas cuando estas tienen que ser llevadas a cabo una tras de otra. Si dos tareas deben ser ejecutadas secuencialmente, usualmente hay una clara interdependencia entre ellas. Por ejemplo, el resultado de la primera es requerido con el fin de empezar la segunda. En Redes de Petri, esta forma de ruteo es modelada uniendo las dos tareas utilizando un lugar. La Figura A.2 muestra un ejemplo de este ruteo.

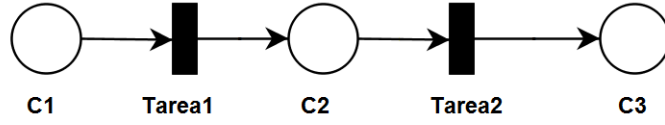


Figura A.2: Enrutamiento secuencial

La tarea que corresponde con la transición *Tarea2* es ejecutada solo cuando la tarea correspondiente con la transición *Tarea1* ha sido completada. Esto es impuesto por el lugar *C2*, que corresponde con la condición que se debe cumplir antes que la tarea *Tarea2* pueda ser llevada a cabo.

A.2.2. Enrutamiento en paralelo

Si más de una tarea puede ser llevada a cabo al mismo tiempo o en cualquier orden, entonces nos estamos refiriendo a enrutamiento en paralelo. Si nos limitamos a una situación con dos tareas, *tarea1* y *tarea2*, entonces tenemos tres posibilidades de ejecución. Ambas tareas pueden ejecutarse en simultáneo, *tarea1* puede ser llevada a cabo primero y después *tarea2*, o *tarea2* puede llevarse a cabo primero, seguida de *tarea2*. La Figura A.3 ilustra cómo puede modelarse esta situación usando Redes de Petri.

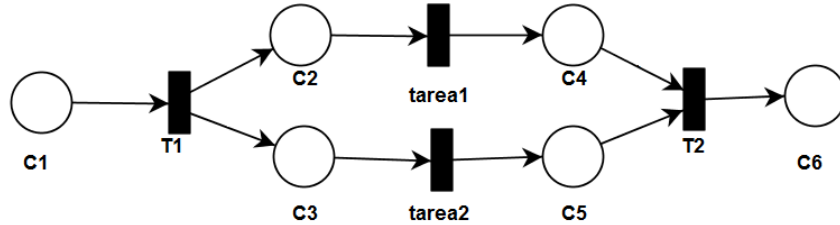


Figura A.3: Enrutamiento en paralelo

Con el fin de permitir la ejecución de *tarea1* y *tarea2* en el caso correspondiente con la ficha en *C1*, se empieza con lo que se llama *AND-split*. Esto es una tarea agregada con el fin de permitir gestionar más de una tarea a la misma vez. En la Figura A.3, la transición *T1* es equivalente a lo que se llama un *AND-split*. Se dispara cuando hay una ficha en *C1* y produce una ficha en cada uno de los lugares *C2* y *C3*. Una vez que la condición *C2* ha sido cumplida para un caso particular, *tarea1* puede ser llevada a cabo. Una vez que la condición *C3* ha sido cumplida, *tarea2* puede ser llevada a cabo. El disparo *T1* de esta manera habilita la ejecución de dos tareas. También se

puede decir que *tarea1* y *tarea2* se pueden llevar a cabo en paralelo. Solamente cuando ambas han sido ejecutadas puede ejecutarse la transición *T2*. Esto es el equivalente a un *AND-join*: una tarea agregada para sincronizar dos o más ejecuciones en paralelo. Solo cuando un caso particular ha cumplido las dos condiciones *C4* y condición *C5* esta tarea puede ejecutarse.

A.2.3. Ruteo Selectivo

Un proceso establece el ruteo para un tipo específico de casos. Puede haber diferencias en ruteo entre los casos individuales. Consideremos, por ejemplo, el caso de un proceso para tratar con procesos de reclamos de seguros. En el proceso de reclamo, dependiendo de las decisiones tomadas, se seleccionara una ruta distinta. La tarea de “enviar a un asesor”, por ejemplo, no es llevada a cabo para pequeños reclamos. Nos referimos a estos casos como enrutamiento selectivo. Esto envuelve la elección entre dos o más tareas. La Figura A.4 muestra un ejemplo modelado como Red de Petri.

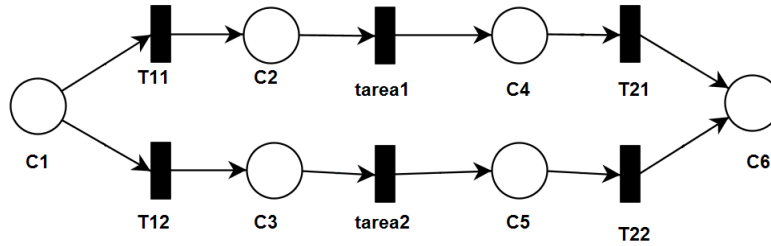


Figura A.4: Ruteo Selectivo

Apéndice B

Clasificación de Resultados de la Revisión Sistemática

En las próximas dos tablas mostraremos la clasificación de los 40 artículos primarios según las categorías definidas en el Capítulo 3.

Ref	Key	Tipo de Proceso				Cuándo Verifica		Cómo Verifica					Lenguaje de Especificación							
		Orquestación	Colaborativo	Grid	Científico	Diseño	Ejecución	Model Checking	Deducción Lógica	Por Construcción	Análisis Estático	Testing/Simulación	No Formales	Semi - Formales	Formales					
1	(JianQiang Li,2003)	x					x		x	x						x				
2	(Henry H. Bi,2004)	x				x			x							x				
3	(Hongchen Li,2005)	x					x				x					x				
4	(Wang Guangqi,2006)	x				x				x						x				
5	(Shiyong Lu,2006)	x				x				x	x					x				
6	(Chen J.,2006)	x																		
7	(Schiefer J.,2006)	x					x				x									
8	(Arindam Mukherjee,2007)		x			x				x										
9	(Baresi L.,2007)		x			x	x				x					x				
10	(Verbeek H.M.W.,2007)	x				x				x						x				
11	(Huang Y.,2008)	x				x				x										
12	(Qingtian Zeng,2008)	x				x				x					x					
13	(Kovács M.,2008)	x						x								x				
14	(Kovács M.,2008)	x				x		x								x				
15	(Luo C.,2008)	x				x			x							x				
16	(Manu De Backer,2009)		x			x				x										
17	(Curcin V.,2009)	x			x			x								x				
18	(Sun H.,2009)		x			x				x						x				
19	(Faisal Abouzaid,2009)	x				x		x								x				
20	(Montali M.,2010)	x				x				x						x				
21	(A. A. Kalenkova,2010)	x				x			x							x				
22	(Ingo Weber,2010)	x				x			x											
23	(Kuhne S.,2010)	x				x				x										
24	(Lam V.S.W.,2010)	x						x				x								
25	(Alex Norta,2010)		x			x										x				
26	(A. S. Varosyan,2011)	x				x				x						x				
27	(Antem Polyanyay,2011)	x				x				x						x				
28	(YanHua Du,2011)		x			x			x											
29	(Jun-Wei Cao,2011)	x		x				x												
30	(Haijun Cao,2011)	x		x		x				x										
31	(Sidorova N.,2011)	x				x										x				
32	(Zha H.,2011)	x				x				x										
33	(Ralf Laue,2011)	x				x				x										
34	(Ma G.,2012)	x				x				x										
35	(Barros C.,2012)	x				x		x												
36	(Luis E. Mendoza,2012)		x			x									x					
37	(Wil van der Aalst,2012)	x				x			x											
38	(Zhao-xia Wang,2012)	x													x					
39	(Fazle Rabbi,2013)	x				x		x								x				
40	(Susanne Patig,2013)	x				x		x								x				

Figura B.1: Clasificación artículos primarios (I)

Ref	Key	Clasificación Propiedades														Herramienta						
		Grafos				Recursos				Colaboración				Otros								
		General	Reachability	Boundedness	Soundness	Safety	Liveness	Lack of synchronization	Deadlock	Livelock	Throughness	No-redundant-transition	Executability		Resource Consistency		Compatibility of collaboration	Consistency	Reliability	Temporal Constraints	Schedulability	
1	(JianQiang Li, 2003)		x																x		No	Si
2	(Henry H. Bi, 2004)									x	x										x	
3	(Hongchen Li, 2005)																					x
4	(Wang Guangqi, 2006)																					x
5	(Shiyong Lu, 2006)																					x
6	(Chen J. , 2006)																					x
7	(Schiefer J. , 2006)																					x
8	(Arindam Mukherjee, 2007)																					x
9	(Baresil, 2007)																					x
10	(Verbeek H.M.W. , 2007)																					x
11	(Huang Y. , 2008)																					x
12	(Qingtian Zeng, 2008)																					x
13	(Kovacs M. , 2008)																					x
14	(Máire Kovács, 2008)																					x
15	(Luo C. , 2008)																					x
16	(Manu Backer, 2009)																					x
17	(Curcin V. , 2009)																					x
18	(Sun H. , 2009)																					x
19	(Faisal Abouzaid, 2009)																					x
20	(Montali M. , 2010)																					x
21	(A. A. Kalenkova, 2010)																					x
22	(Ingo Weber, 2010)																					x
23	(Kühne S. , 2010)																					x
24	(Lam V.S.W. , 2010)																					x
25	(Alex Norta, 2010)																					x
26	(A. S. Varosyan, 2011)																					x
27	(Artem Polywyan, 2011)																					x
28	(YanHua Du, 2011)																					x
29	(Cheng Wu, 2011)																					x
30	(Shadi Ibrahim, 2011)																					x
31	(Sidorova N. , 2011)																					x
32	(Zha H. , 2011)																					x
33	(Ralf Laue, 2011)																					x
34	(Ma G. , 2012)																					x
35	(Barros C. , 2012)																					x
36	(Luis E. Mendoza, 2012)																					x
37	(Wil van der Aalst, 2012)																					x
38	(Zhao-xia Wang, 2012)																					x
39	(Fazle Rabbi, 2013)																					x
40	(Susanne Paltg, 2013)																					x

Figura B.2: Clasificación artículos primeros (II)

Apéndice C

Lista completa de errores detectados por Activiti

A continuación mostraremos una lista con los 63 errores que detecta Activiti:

- “activiti-process-definition-not-executable”
- “activiti-association-invalid-source-reference”
- “activiti-association-invalid-target-reference”
- “activiti-execution-listener-implementation-missing”
- “activiti-event-listener-implementation-missing”
- “activiti-event-listener-invalid-implementation”
- “activiti-event-listener-invalid-throw-event-type”
- “activiti-start-event-multiple-found”
- “activiti-start-event-invalid-event-definition”
- “activiti-seq-flow-invalid-src”
- “activiti-seq-flow-invalid-target”
- “activiti-usertask-listener-implementation-missing”
- “activiti-servicetask-invalid-type”
- “activiti-servicetask-result-var-name-with-delegate”
- “activiti-servicetask-missing-implementation”
- “activiti-servicetask-webservice-invalid-operation-ref”

- “activiti-sendtask-invalid-implementation”
- “activiti-sendtask-invalid-type”
- “activiti-send-webservice-invalid-operation-ref”
- “activiti-scripttask-missing-script”
- “activiti-mailtask-no-recipient”
- “activiti-mailtask-no-content”
- “activiti-shelltask-no-command”
- “activiti-shelltask-invalid-param”
- “activiti-exclusive-gateway-no-outgoing-seq-flow”
- “activiti-exclusive-gateway-condition-not-allowed-on-single-seq-flow”
- “activiti-exclusive-gateway-condition-on-seq-flow”
- “activiti-exclusive-gateway-seq-flow-without-conditions”
- “activiti-event-gateway-only-connected-to-intermediate-events”
- “activiti-subprocess-multiple-start-event”
- “activiti-subprocess-start-event-event-definition-not-allowed”
- “activiti-event-subprocess-invalid-start-event-definition”
- “activiti-boundary-event-no-event-definition”
- “activiti-boundary-event-invalid-event-definition”
- “activiti-boundary-event-cancel-only-on-transaction”
- “activiti-boundary-event-multiple-cancel-on-transaction”
- “activiti-intermediate-catch-event-no-eventdefinition”
- “activiti-intermediate-catch-event-invalid-eventdefinition”
- “activiti-error-missing-error-code”
- “activiti-event-missing-error-code”
- “activiti-event-timer-missing-configuration”
- “activiti-throw-event-invalid-eventdefinition”
- “activiti-multi-instance-missing-collection”

- “activiti-message-missing-name”
- “activiti-message-invalid-item-ref”
- “activiti-message-event-missing-message-ref”
- “activiti-message-event-invalid-message-ref”
- “activiti-message-event-multiple-on-boundary-same-message-id”
- “activiti-operation-invalid-in-message-reference”
- “activiti-signal-event-missing-signal-ref”
- “activiti-signal-event-invalid-signal-ref”
- “activiti-compensate-event-invalid-activity-ref”
- “activiti-compensate-event-multiple-on-boundary”
- “activiti-signal-missing-id”
- “activiti-signal-missing-name”
- “activiti-signal-duplicate-name”
- “activiti-signal-invalid-scope”
- “activiti-data-association-missing-targetref”
- “activiti-data-object-missing-name”
- “activiti-end-event-cancel-only-inside-transaction”
- “activiti-di-invalid-reference”
- “activiti-di-does-not-reference-flownode”
- “activiti-di-does-not-reference-seq-flow”

Apéndice D

Gestión del proyecto

En el siguiente apéndice se pasará a desarrollar como se gestionó el proyecto y un resumen ejecutivo del mismo.

D.1. Grupo de trabajo

El grupo de trabajo de este proyecto se compuso por dos estudiantes y dos tutores a cargo. La modalidad de trabajo fue planteada de forma de mantener reuniones cuando fuera necesario para planificar objetivos, evaluar resultados y evacuar dudas, manteniendo siempre comunicación vía mail para realizar consultas puntuales y agendar reuniones. Otro medio utilizado entre los estudiantes fue la plataforma *Hangouts* de Google para comunicación más informal y espontánea.

D.2. Documentación

La documentación se fue generando desde el comienzo, ya que era una parte importante del proyecto mantener registro de los resultados. Se realizaron documentos intermedios para validar resultados y análisis, que se almacenaron en *Google Drive* con el objetivo de que estén compartidos y mantener versiones. Para las versiones finales se pasó a utilizar TeXnicCenter, que es un entorno integrado para crear archivos Latex y JabRef para el manejo de las referencias.

D.3. Cronograma previsto

El cronograma propuesto al comienzo del proyecto fue el siguiente:

- Estudio del marco teórico de BPM y de la realización de revisiones sistemáticas. (1 mes)

Producto: informe con resumen del contexto.

- Revisión Sistemática de la literatura. (3 meses)
Productos: documento del estado del arte sobre métodos, técnicas, lenguajes y tecnologías para la verificación de Procesos de Negocio.
- Realización de un caso de estudio en el que se puedan aplicar las técnicas de verificación relevadas en un ejemplo de la realidad. (3 meses)
Productos: documento con el desarrollo del caso de estudio, lecciones aprendidas, conclusiones, y en caso de desarrollo propio, también del mismo.
- Informe del proyecto de grado dividido en capítulos, correcciones de los tutores y defensa. (1 mes)
Producto: informe final del proyecto de grado por capítulos y defensa.

D.4. Etapas y cronograma real del proyecto

Este proyecto fue realizado entre Abril de 2013 y Noviembre de 2014. Pasaremos a describir las etapas llevadas a cabo en dicho período.

D.4.1. Contexto

En esta primera etapa el objetivo fundamental era tener un primer acercamiento a los Procesos de Negocio, la Gestión de Procesos de Negocio, los Sistemas de Gestión de Procesos de Negocio y la verificación de los mismos, ya que por parte de los estudiantes no se contaba con conocimientos en esta área. Al mismo tiempo, se investigó sobre cómo realizar la Revisión Sistemática que también era algo nuevo y necesario de comprender en detalle antes de realizarla.

D.4.2. Planificación y ejecución de la Revisión Sistemática

Luego de comprender los pasos para realizar la Revisión Sistemática y un conocimiento general del área de estudio, se planificaron los pasos a seguir y se ejecutaron. Esto implicó definir las preguntas de investigación, la cadena de búsqueda, definir los criterios de inclusión/exclusión, realizar la búsqueda en las distintas fuentes y filtrar los resultados para obtener los resultados primarios, entre otras tareas. En el Capítulo 3 se puede ver en profundidad todo el procedimiento realizado.

D.4.3. Análisis de resultados

Obtenidos los estudios primarios se pasa a realizar un análisis de estos, lo cual implicó una lectura en detalle para realizar distintas clasificaciones. Al

mismo tiempo, fue necesario investigar sobre todas las clasificaciones identificadas para llevarlo a cabo de la mejor manera y poder documentar formalmente todos los resultados. En la Sección 2 del Capítulo 3 se puede encontrar el análisis y los distintos criterios de clasificación.

D.4.4. Desarrollo de un entorno integrado de verificación

En esta etapa se desarrolló un entorno de verificación para el que se integraron distintas herramientas para un Plug-in del entorno integrado de desarrollo Eclipse. Fue necesario investigar y evaluar distintas herramientas y la posibilidad de integrarlas, para seleccionar las adecuadas. Luego de construída la herramienta se verificó para comprobar su correcto funcionamiento y conocer sus limitaciones. Finalmente se realizó un caso de estudio para mostrar cómo se utiliza y su efectividad. En el Capítulo 4 se encuentra toda la información del desarrollo, en el Capítulo 5 está todo lo relativo a la verificación y en el Capítulo 6 el caso de estudio.

D.4.5. Elaboración del informe

En el transcurso de cada etapa se fue generando documentación que al final se unificó y completó para el informe final. Este documento se fue mejorando de forma iterativa gracias a las correcciones y sugerencias por parte de los tutores, a medida que fueron hechas entregas de versiones parciales.

D.4.6. Cronograma real

En las figuras D.1, D.2 y D.3 se pueden apreciar la planificación propuesta y el cronograma real separado en año 2013 y año 2014 respectivamente.

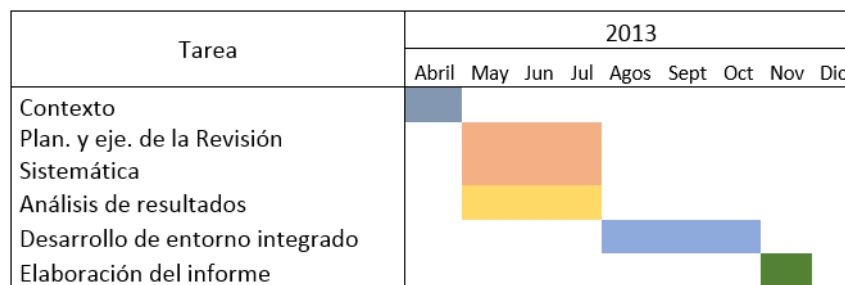


Figura D.1: Planificación propuesta

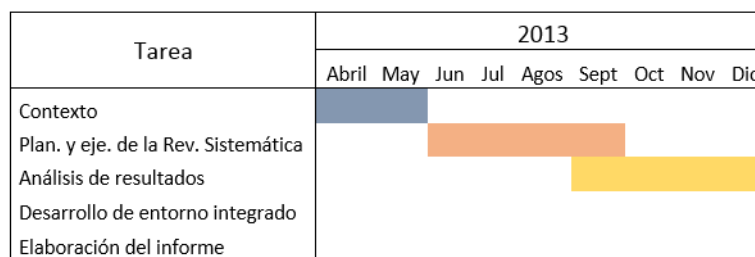


Figura D.2: Cronograma real 2013

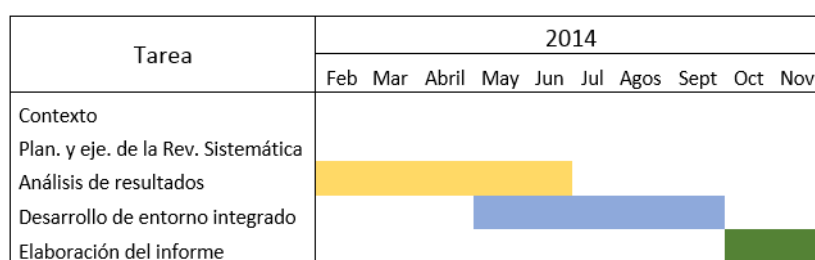


Figura D.3: Cronograma real 2014

Como se puede apreciar el proyecto se extendió más de lo planificado, detallaremos las razones.

En el comienzo la carga horaria con otras materias de los estudiantes limitó un poco la dedicación al proyecto.

Por otro lado el análisis de los resultados tuvo que realizarse tres veces ya que las primeras dos por no tener el conocimiento profundo necesario en el área por parte de los estudiantes, los resultados no fueron correctos. Finalmente la tercera vez se alcanzó comprender el tema al nivel necesario y se logró el resultado esperado.

Mientras que en la parte de desarrollo, la integración de las distintas herramientas no fue tarea sencilla. En el caso de Activiti Designer, es un proyecto Open Source que aún se encuentra en desarrollo y el módulo que permite realizar la verificación de modelos (por fuera del diseñador) es muy reciente, por lo que fue difícil encontrar documentación para utilizarlo. Integrar esta herramienta sin ayuda del foro hubiera sido muy difícil, en la Figura D.4 vemos la pregunta realizada y en la Figura D.5 se puede ver la respuesta a nuestra consulta, que nos fue de gran ayuda para lograr utilizar el verificador de Activiti fuera del diseñador. (<http://forums.activiti.org/content/trying-call-processvalidator-command>)

Trying to call a ProcessValidator from Command

Quick reply Last post

Wed, 08/06/2014 - 00:32

igoyetche

Online

Last seen: 0 sec ago

Joined: 03/30/2014 - 19:59

Posts: 5

3 posts / 0 new

#1

Trying to call a ProcessValidator from Command

Unfollow this thread

Hi,

I'm trying to call a custom ProcessValidator of Activiti designer from a toolbar command, so I started with the example of ProcessValidator in GitHub "BPMN20ProcessValidator".

I took this code and added to it a Command to call the "validateDiagram(Diagram diagram, IProgressMonitor monitor)" Method, that I'm not sure if is it OK. After trying with a lot of things, to call this method with no success, I given up with this:

```
public Object execute(ExecutionEvent event) throws ExecutionException {
    final IWorkbenchWindow window = HandlerUtil.getActiveWorkbenchWindowChecked(event);
    final IEditorPart activeEditor = window.getActivePage().getActiveEditor();
    ActivitiDiagramEditor editor = (ActivitiDiagramEditor) activeEditor;
    final IDiagramTypeProvider pro = editor.getDiagramTypeProvider();
    final Diagram diagram = pro.getDiagram();

    final ProcessValidator processValidator = (ProcessValidator)
        ExtensionPointUtil.getProcessValidator(PluginConstants.VALIDATOR_ID);

    processValidator.validateDiagram(diagram, new NullProgressMonitor());
}
```

The problem with the code above is that when I obtain the processValidator I defined, it comes with the diagramWorkerContext in null, and it throws a NullReferenceException.

I searched in all the GitHub Activiti Designer code and I didn't find how could I set the diagramWorkerContext, so I'm starting to believe that I'm using it in a way that is not expected to use it.

My doubt is: how is expected way to use the processValidator??, exist any example of it??

Please I appreciate any help!!, I been trying to made this for weeks without luck and It's for a very important project.

Thanks in advance,

igoyetche

Top

reply Unfollow this thread

Figura D.4: Pregunta al foro de Activiti

frederikherman

S



Offline

Last seen: 1 month 3 weeks ago

Joined: 08/26/2010 - 02:59

Posts: 3087

Send PM

The process validator you are referring to cannot be used outside of the Designer. However, it's your lucky day. We recently added a new module that can be used separately:
<https://github.com/Activiti/Activiti/tree/master/modules/activiti-process-validation>
Add it as a dependency to your project, get hold of an instance of
<https://github.com/Activiti/Activiti/blob/master/modules/activiti-process-validation/src/main/java/org/activiti/validation/ProcessValidatorFactory.java> and run your model through that, instead.

—

Activiti Core Developer

Top

reply

Sun, 08/17/2014 - 19:43

#3

Figura D.5: Respuesta del foro de Activiti

La integración con ProM y WoPeD no insumió tanto tiempo como Activiti pero también contaba con poca documentación para alcanzar nuestro objetivo, lo cual hizo que llevara más tiempo del estimado.