



Generación de modelos de procesos de negocio desde texto

Proyecto de grado

Emiliano Alonzo - Rodrigo Damiano

Supervisor responsable

Andrea Delgado - Daniel Calegari

Instituto de Computación

Facultad de Ingeniería - Universidad de la República

Montevideo - Uruguay

Resumen

Los procesos de negocio son un conjunto de tareas o actividades que se realizan en un ambiente técnico y organizacional produciendo una salida del negocio definida para un cliente o un mercado particular. Un ejemplo de proceso de negocio puede ser el trámite para obtener la licencia de conducir en la intendencia o el proceso de contratación de personal de una empresa.

La notación y modelo para Procesos de Negocio (Business Process Model and notation, BPMN 2.0) es un estándar utilizado para modelar y ejecutar procesos de negocio, brindando elementos para representar actividades y tareas humanas en formularios por ej. web, y tareas automáticas como webservices, permite asociar datos a las actividades, definir el flujo que las conecta, en forma secuencial, condicional o accionar actividades en paralelo, entre otros. De esta forma permite generar modelos ejecutables los cuales, una vez configurados pueden ser directamente ejecutados en algún sistema que permita ejecutar procesos de negocio, como ser los Sistemas de Gestión de Procesos (Business Process Management Systems, BPMS).

Para realizar estos modelos, con sus respectivos elementos y luego realizar la configuración (implementación) para poder ejecutarlos, es necesario tener ciertos conocimientos técnicos, por lo que para un usuario no técnico sin asesoramiento, podría resultar en modelos con errores o en modelos que no representen fielmente lo que se desea modelar. Por esto surge la necesidad de poder generar estos modelos de forma más sencilla en base a la especificación del proceso realizada por los involucrados, de forma de proveer un apoyo importante a la hora de obtener modelos BPMN 2.0 con ciertos elementos de calidad.

Este proyecto propone definir un lenguaje textual, similar al lenguaje natural, capaz de expresar elementos de un modelo BPMN 2.0 con el objetivo de facilitar al usuario no experto la generación de modelos con elementos de calidad. Para ello, definimos un lenguaje que permite especificar en forma textual procesos con determinados elementos seleccionados, y generamos un prototipo funcional capaz de reconocer la descripción de dichos procesos expresados en forma textual, en el lenguaje definido con la gramática especificada, generando el modelo BPMN 2.0 asociado.

Entre los objetivos planteados se encuentran generar modelos sintácticamente válidos según el estándar, asistir la generación modelos semánticamente correctos, que contengan buenas prácticas y generar modelos ejecutables en algún BPMS, como ser Activiti.

Índice

1 Introducción	4
1.1 Objetivos	5
1.2 Organización del documento	5
2 Estado del arte	6
2.1 Procesos de Negocio	6
2.2 BPMN 2.0	7
2.3 Modelado con BPMN 2.0	7
2.4 Aspectos de calidad en modelos de procesos de negocio	9
2.5 Trabajos similares	9
3 Problema y solución planteada	11
3.1 Planteo del problema	11
3.2 Propuesta de la solución	12
	12
4 Definición y formalización del lenguaje	16
4.1 Especificación del lenguaje	16
4.2 Formalización del lenguaje	22
5 Desarrollo de la Solución	25
5.1 Ingresar proceso textual	26
5.2 Aplicar parser	26
5.3 Procesar el modelo intermedio	28
5.4 Generar modelo BPMN 2.0	30
5.5 Generar modelo con buenas prácticas	34
5.6 Generar representación gráfica.	35
5.7 Desplegar resultados	35
6 Prototipo funcional	37
6.1 Plataforma y herramientas	37
6.2 Presentación de la interfaz gráfica	37
7 Caso de estudio	48
7.1 Adjudicación de movilidades de estudiantes de grado	48

7.2 Construcciones no soportadas	51
7.3 Generación a partir de la descripción textual	52
7.4 Generación a partir del modelo BPMN 2.0 original	61
7.5 Ejecución del modelo BPMN 2.0 obtenido	64
8 Conclusiones y trabajo a futuro	70
8.1 Resultados obtenidos	70
8.2 Limitaciones y trabajo a futuro.	72
9 Referencias	74

1 Introducción

Un elemento clave para el desarrollo de sistemas que cumplan con las expectativas y planteos de los clientes es la captura de requerimientos, los artefactos generados para la comunicación con el cliente en dicha etapa, y los acuerdos alcanzados en base al entendimiento común de los mismos. En este sentido, existen diversas propuestas para agilizar esta etapa y alcanzar acuerdos que sirvan

Un elemento clave para el desarrollo de sistemas que cumplan con las expectativas y planteos de los clientes es la captura de requerimientos, los artefactos generados para la comunicación con el cliente en dicha etapa, y los acuerdos alcanzados en base al entendimiento común de los mismos. En este sentido, existen diversas propuestas para agilizar esta etapa y alcanzar acuerdos que sirvan

Un elemento clave para el desarrollo de sistemas que cumplan con las expectativas y planteos de los clientes es la captura de requerimientos, los artefactos generados para la comunicación con el cliente en dicha etapa, y los acuerdos alcanzados en base al entendimiento común de los mismos. En este sentido, existen diversas propuestas para agilizar esta etapa y alcanzar acuerdos que sirvan de base para el desarrollo a realizar. Por un lado la posibilidad de usar modelos gráficos y textuales con alguna posible relación entre ellos y por otro lado permitir que estos modelos sean validados rápidamente.

De esta forma es que surge la propuesta de utilizar modelos de procesos de negocio los cuales permiten modelar los procesos de una organización, para ser ejecutados en Sistemas de Gestión de Procesos de Negocio (Business Process Management Systems, BPMS)[1], siendo posible la representación lógica de la gran mayoría de las funcionalidades requeridas. En particular, un modelo de procesos de negocio expresado en una notación como la notación y modelo para Procesos de Negocio (Business Process Model and Notation, BPMN 2.0)[2], los cuales proveen las bases para acordar los elementos principales de los procesos, como el flujo de secuencia entre las actividades, los tipos de actividades a realizar (de usuario con formularios, automáticas invocando web services, de reglas de negocio ejecutando políticas de la organización, entre otras), los caminos posibles mediante nodos de decisión, y presentar rápidamente al usuario con las pantallas y tipos de ejecución posibles según el modelo.

Un elemento de motivación del proyecto es poder generar estos modelos, de forma automática, a partir de un modelo textual simple que pueda ser fácilmente entendible y utilizable por cualquier usuario sin la necesidad de tener un conocimiento exhaustivo de la notación BPMN 2.0 o el modelado de procesos de negocio. Obteniendo de esta forma una simplificación en la comunicación entre desarrolladores y clientes. De esta manera se busca, entre otros elementos, reducir el tiempo de desarrollo, dado que según establece en su trabajo Frederick [3], en proyectos de flujo de procesos, el 60% del tiempo total del proyecto es dedicado al desarrollo de modelos.

Entonces, como elemento principal de comunicación con el cliente se quiere utilizar una notación textual adecuada, a definir en base a lenguajes y propuestas existentes, para generar modelos de procesos de negocio desde dicha notación textual. Se evaluarán propuestas existentes, incluyendo un lenguaje de base que se está definiendo en el contexto del cliente, y se definirán los elementos necesarios para poder expresar no sólo el flujo de trabajo del proceso, sino también los usuarios que deben realizar las tareas y los datos que están involucrados en su realización.

1.1 Objetivos

El objetivo general de este proyecto de grado es especificar un lenguaje que permita describir procesos de negocio de forma textual así como también definir transformaciones para la generación automática de los modelos y perspectivas asociadas (flujos de control, recursos, datos). A su vez es de interés que estos modelos puedan ser validados tempranamente y que cumplan con las buenas prácticas de modelado de procesos de negocio.

Los objetivos específicos de este proyecto son:

- el relevamiento de estrategias de generación de modelos de procesos de negocio desde texto.
- una especificación de un lenguaje para descripción de procesos de negocio en forma textual y transformaciones para la generación automática de los modelos y perspectivas asociadas (flujo de control, recursos, datos).
- La construcción de un prototipo funcional para la generación automática de modelos de procesos de negocio a partir del lenguaje especificado.
- un caso de estudio de aplicación de la propuesta en un dominio específico.

Este proyecto fue realizado en el marco de la colaboración del grupo COAL en el proyecto de generación de prototipos del Instituto de Tecnologías de Información y Comunicación para Verticales (Information and Communication Technologies for Verticals, ICT4V). El objetivo de la línea de investigación es la generación de procesos de negocio ejecutables a partir de descripciones textuales, utilizando técnicas de Ingeniería Dirigida por Modelos.

1.2 Organización del documento

El resto de la documentación se encuentra organizado en capítulos como se explica a continuación. En el capítulo 2 se presenta el estado del arte, el cual contiene los conceptos técnicos del proyecto y trabajos similares en el área. En el capítulo 3 se plantea el problema afrontado de manera abstracta y se plantea una solución para el mismo. En el capítulo 4 trata sobre el lenguaje a definir junto con una formalización del mismo donde se plantea el paralelismo con los componentes del estándar BPMN 2.0. En el capítulo 5 se exponen los conceptos fundamentales de la implementación, detallando los modelos de datos utilizados y las transformaciones sobre los mismos. El capítulo 6 trata acerca del desarrollo de la solución, donde se comenta acerca del prototipo implementado, explicitando sus funcionalidades principales. En el capítulo 7 se presenta un caso de estudio, realizando distintas comparaciones contra un modelo de un proceso real de la UdelaR[4]. En el capítulo 8 se presentan las conclusiones del proyecto y las propuestas para un trabajo a futuro.

Además del informe final se pueden consultar los siguientes anexos

- Estándar BPMN 2.0
- Ejemplos generados durante el desarrollo
- Arquitectura modular de la solución

2 Estado del arte

En esta sección se mencionan los temas abordados durante el proyecto con el objetivo de establecer un marco conceptual para ayudar a entender el trabajo.

2.1 Procesos de Negocio

Un proceso de negocio [5], es un conjunto de tareas y actividades relacionadas que producen un servicio específico o producto para un cliente o mercado particular. Los elementos principales de un proceso de negocio son las entradas y salidas del mismo (requerimientos de clientes, información, servicios, productos etc.), los recursos para realizarlos (personas, materiales, etc.) y el conjunto de objetivos (organización, específicos del mismo). Como elementos específicos se pueden mencionar los participantes (internos: secciones, áreas, roles; externos: clientes, socios, proveedores), actividades, tareas, datos, información y la relación entre estas (secuencias, reglas de negocio, bifurcaciones, restricciones).

Los procesos de negocio tienen algunas particularidades como pueden ser el alcance, la complejidad del mismo, en un mismo proceso de negocio se pueden ver involucradas varias secciones o departamentos de una organización, incluso diferentes organizaciones, la duración de los procesos de negocio puede ser reducida, pero también puede ser bastante extensa (días, meses años). A su vez también existen las particularidades de cada dominio (salud, bancarios, etc). Los procesos de negocio pueden ser manuales o automatizados. Incluso difíciles de explicitar (implícitos en los sistemas).

La gestión de procesos de negocio (Business Process Management, BPM) refiere a un conjunto de actividades que realizan las organizaciones para optimizar o adaptar sus proceso de negocio a las nuevas necesidades organizacionales. Incluye, conceptos, técnicas, métodos y software, involucrando personas, organizaciones, aplicaciones, documentos y otras fuentes de información con el objetivo de optimizar, adaptar y mejorar los procesos de negocio en las organizaciones, de forma de obtener procesos más alineados con los objetivos organizacionales.

BPM se basa en el ciclo de vida de los procesos de negocio, el cual se define en [6] en cuatro etapas, la primera etapa es de diseño y análisis que consiste en modelar y validar los procesos de negocio en la organización. La segunda etapa es la de configuración, la cual consiste en implementar, testear y desplegar los procesos de negocio en la organización. Continuando, la tercer etapa es la de ejecución, que implica la ejecución de los procesos de negocio y registrar datos asociados a la ejecución. Por último, la cuarta etapa es la de evaluación que consiste en la evaluación de la ejecución de los procesos de negocio para mejorar los modelos y la implementación de los mismos. Este proyecto se enfoca en automatizar la etapa de análisis y diseño, y de forma parcial, la etapa de configuración.

Existen varias notaciones que brindan las herramientas para modelar y/o ejecutar los procesos de negocio con distintos grados de expresividad y utilidad, como ser EPC [7], YAWL [8]. Incluso, los diagramas de actividad UML [9], XPDL [10] y WS-BPEL [11] para ejecución. Pero BPMN 2.0 es desde los últimos años, el estándar de facto para la industria, adoptado también fuertemente en la academia.

2.2 BPMN 2.0

BPMN 2.0 es un estándar para el modelado de procesos de negocio que provee una notación gráfica para especificar los mismos, basado en diagramas de flujo, en forma similar (pero mucho más específica) a los diagramas de actividad de UML.

La principal meta de BPMN 2.0 es proveer una notación estándar, que sea simple de entender por todas las partes interesadas del negocio. En estos se incluyen los analistas del negocio quienes crean y refinan los procesos, los técnicos desarrolladores responsables de implementarlos, y los administradores del negocio que los monitorean y gestionan. En consecuencia, BPMN 2.0 funciona como un lenguaje en común, que reduce la brecha de comunicación que frecuentemente hay entre el diseño de los procesos de negocio y su implementación.

2.3 Modelado con BPMN 2.0

En la Figura 2.1 se muestra un ejemplo donde se modela un proceso de contratación de personal en BPMN 2.0 utilizando diversos elementos provistos por la notación, descritos a continuación.

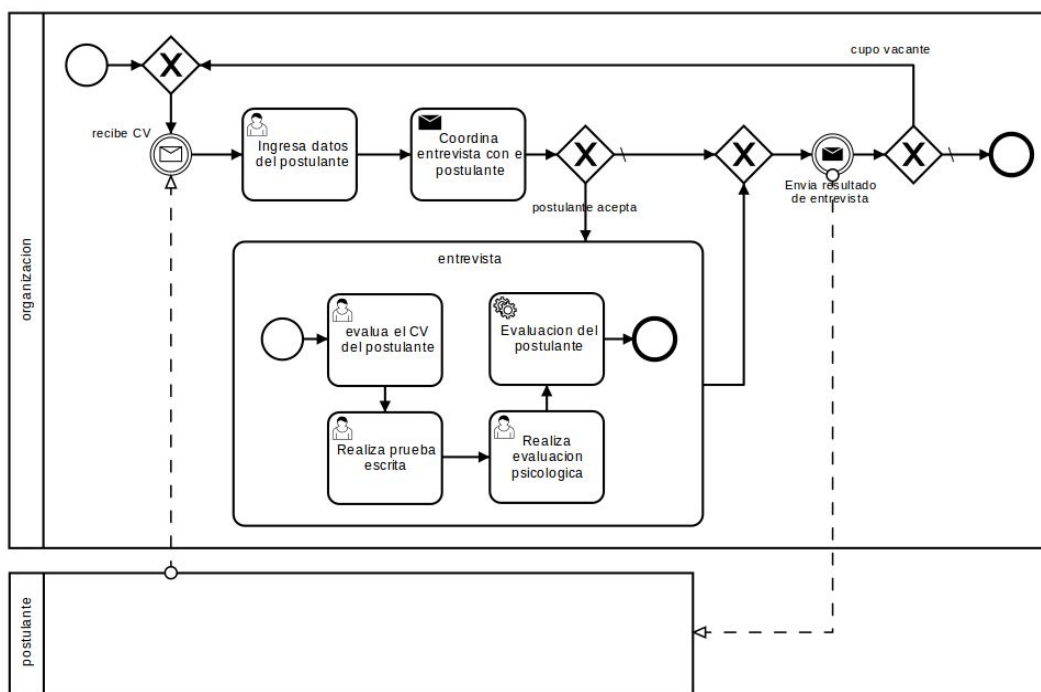


Figura 2.1. Ejemplo de modelo BPMN 2.0

Como se muestra en la Figura 2.1, el postulante envía un CV a la organización contratante. Cuando a esta le llega el CV, ingresa los datos del postulante en su sistema y coordina una entrevista con el mismo. Si el postulante acepta ir a la entrevista, se reúnen según lo coordinado. Luego de realizada la entrevista se notifica al postulante el resultado de la misma. En caso que el postulante haya sido rechazado se sigue buscando un nuevo postulante. La entrevista consta de cuatro etapas, la primera consta de una evaluación del CV, en segundo lugar se procede a realizar una prueba escrita, en tercer lugar se realiza una evaluación psicológica y por último, se realiza una evaluación del postulante donde se utiliza la información de las etapas anteriores.

Los modelos de procesos BPMN 2.0 consisten en diagramas contruidos a partir de un conjunto de elementos gráficos. Estos elementos se pueden categorizar en cuatro grupos básicos, los cuales detallamos a continuación.

Elementos de flujo: Actividades, Eventos y Compuertas.

Estos son los elementos más descriptivos de BPMN 2.0 y consisten en tres tipos de elementos, tareas, eventos y compuertas. Los eventos son utilizados para representar cosas que suceden (mensajes, timers, etc.) y son representados utilizando círculos, por ejemplo, el evento de recepción del CV de la Figura 2.1. Las actividades se utilizan para representar el tipo de cosas que deben ser hechas (tareas, subprocessos, transacciones, etc.), estas son representadas con rectángulos, por ejemplo, cuando se ingresan los datos del postulante en la Figura 2.1. Las compuertas determinan la bifurcación y unión de caminos, dependiendo de las condiciones establecidas, estas compuertas son representadas con rombos como se puede ver en la Figura 2.1.

Elementos de conexión: Flujo de secuencia, flujo de mensaje y asociación.

Los elementos de flujo están conectados entre sí utilizando los elementos de conexión que pueden ser de tres tipos, flujo de secuencia, flujo de mensaje y asociación, en todos los casos se indica un origen y destino en la conexión. Los flujos de secuencia representan la conexión entre las actividades de una misma organización y son representadas con flechas continuas como se puede ver en la Figura 2.1 conectando las actividades compuertas y eventos. Los flujos de mensaje son utilizados para representar los mensajes que fluyen a través de los límites organizacionales, estas son las flechas punteadas que conectan los eventos de mensaje del pool de la organización con el pool del postulante como se ve en la Figura 2.1. Las asociaciones son utilizadas para asociar artefactos o texto a los elementos de flujo.

Swimlanes: roles y participantes.

Los swimlanes son un mecanismo visual para organizar y categorizar las actividades. Estos pueden ser de dos tipos, lanes y pools. Los pools representan los participantes en un proceso. En la Figura 2.1 se ven dos pools, uno representando a la organización y otro representando al postulante. Típicamente los pools representan las diferentes organizaciones que participan en el proceso. Los lanes son utilizados para organizar y categorizar actividades dentro de un pool, normalmente representan los diferentes roles dentro de una organización que participan del proceso.

Artefactos: Objetos de datos, grupos y anotaciones.

Los artefactos permiten a los desarrolladores agregar más información al modelo, permitiendo un modelo más entendible. Hay tres tipos de artefactos pre definidos, objetos de datos, grupos y anotaciones. Los objetos de datos muestran que datos son requeridos o producidos por una actividad. Los grupos son utilizados para agrupar diferentes actividades pero sin afectar el flujo del diagrama. Las anotaciones son utilizadas para dar al lector del diagrama información más comprensible.

2.4 Aspectos de calidad en modelos de procesos de negocio

Como se mencionó en la introducción, es de interés que los modelos generados a partir de la notación textual cumplan con buenas prácticas para el modelado de procesos de negocio. En particular, interesan las siete guías para el modelado de procesos de negocio [12] que tienen el objetivo de mejorar la calidad de los mismos. Estas guías proveen un conjunto de sugerencias sobre cómo construir los modelos de procesos y para mejorar modelos de procesos existentes.

G1 - Usar la menor cantidad de elementos que sea posible. Los modelos de gran tamaño tienden a ser un problema a la hora de intentar entenderlos y tienen una mayor probabilidad de tener errores en comparación con modelos de menor tamaño.

G2 - Minimizar los caminos de ruteo para cada elemento. Si los elementos del modelo tienen demasiados flujos de entrada y de salida el modelo se torna más complicado de entender.

G3 - Utilizar solo un evento de inicio y un evento de fin. Permite minimizar la probabilidad de errores y hace el modelo más entendible.

G4 - Modelar de manera lo más estructurada posible. Se entiende que un modelo es estructurado si para cada compuerta que genera una bifurcación, existe otra compuerta del mismo tipo que sincronice los diferentes flujos generados en la bifurcación. Modelos no estructurados tienden a ser menos entendibles y tienden a incluir más errores.

G5 - Evitar el uso de compuertas OR. Los modelos que contienen solo compuertas AND y XOR en general tienden a tener menos errores.

G6 - Utilizar etiquetas de tipo “verbales”. Es más entendible utilizar etiquetas del tipo “enviar carta” que utilizar etiquetas del tipo “envío de carta”.

G7 - Si un modelo tiene más de 50 elementos se debería descomponer. Esta guía está relacionada con G1, la cual motiva que los modelos no sean demasiado grandes.

También el trabajo provee un ranking ordenado de las reglas más relevantes que puede apreciarse en tabla 2.1.

Posición en el ranking	1	2	3	4	5	6	7
Regla	G4	G7	G1	G6	G2	G3	G5

Tabla 2.1. ranking ordenado de las reglas

2.5 Trabajos similares

Un trabajo similar es el realizado por Fabian Friedrich en su tesis de maestría, que consiste en una herramienta capaz de realizar modelos BPMN 2.0 desde descripciones textuales en lenguaje natural. Al igual que nuestro trabajo, dado como entrada un texto, es capaz de generar el modelo en formato XML [13], pero con un dominio de valores de entrada mucho más amplio. Nuestro

enfoque acota el espacio de valores de entrada, definiendo un lenguaje con reglas precisas, evitando ambigüedades del lenguaje natural. Dada la similaridad del problema nos guiamos en algunos aspectos de la solución, en particular, los pasos que requieren el procesamiento del texto de entrada, como se describe más adelante en el capítulo 5.

En la Figura 2.2 se muestra la arquitectura de la solución implementada por friedrich donde se genera el modelo BPMN 2.0 a partir de la información almacenada en el objeto de datos World Model.

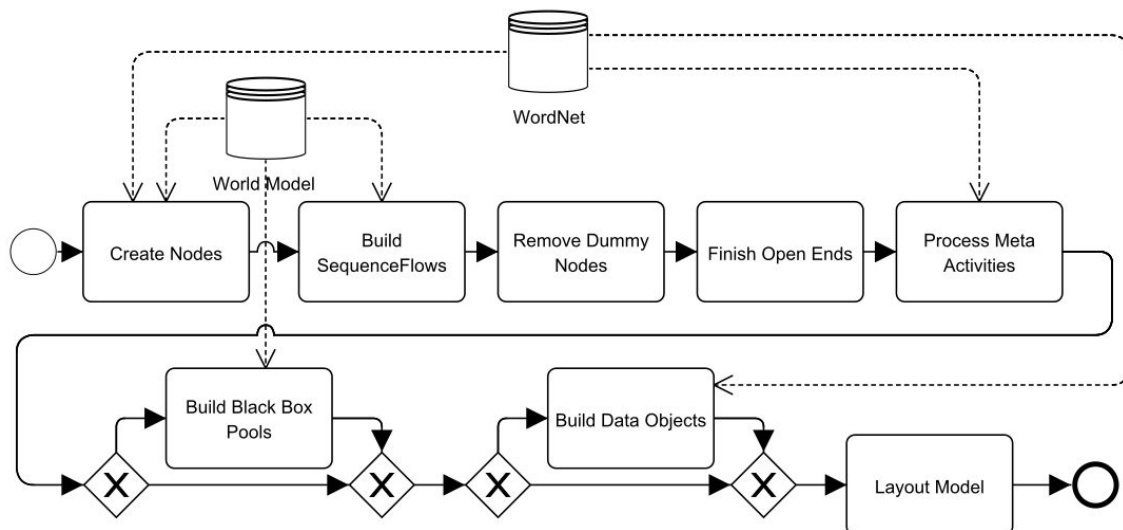


Figura 2.2. Visión general estructurada de los pasos en la fase de generación de modelos de procesos

Se realiza un procedimiento de nueve etapas, las primeras, creación de nodos, construcción de flujos de secuencia, eliminación de nodos dummy, el cierre de finales abiertos y el procesamiento de meta-actividades son utilizados para crear un modelo inicial completo. Opcionalmente, se puede mejorar el modelo, incluyendo pools externos colapsados y Data Objects. Y finalmente el modelo es dibujado para lograr una representación más simple de interpretar.

3 Problema y solución planteada

En este capítulo presentaremos el planteo del problema, analizando los objetivos del proyecto y luego pasaremos a plantear una solución para el problema planteado.

3.1 Planteo del problema

Para definir el problema que abarca este proyecto de grado analizaremos los objetivos del mismo, tomando en cuenta los que consideramos son de valor al problema en sí.

Como primer objetivo consideramos el siguiente: definir una especificación de un lenguaje para descripción de procesos de negocio en forma textual y transformaciones para la generación automática de los modelos y perspectivas asociadas (flujo de control, recursos, datos).

En cuanto a este objetivo se pueden identificar varios problemas, primero definir qué representación de procesos de negocio se va a utilizar. Una vez seleccionada, el siguiente problema consiste en analizar esta representación para identificar que elementos de esta se van a utilizar de forma de obtener un subconjunto representativo de los elementos que modelan los procesos de negocio. Luego, definir el lenguaje que describa los elementos del subconjunto seleccionado, de forma que este sea fácil de entender y por ende, simple de utilizar por cualquier usuario independientemente de su conocimiento de la representación de procesos utilizada. Una vez definido el lenguaje, queda definir las transformaciones necesarias para generar de forma automática los modelos y perspectivas asociadas, lo que implica definir qué modelos se quieren obtener.

Continuando con los objetivos tenemos el siguiente: la construcción de un prototipo funcional para la generación automática de modelos de procesos de negocio a partir del lenguaje especificado. Este objetivo implica el desarrollo de una herramienta, que dada como entrada la especificación del lenguaje mencionado, y luego de aplicar un conjunto de transformaciones, se obtenga como salida los modelos de procesos de negocio y sus perspectivas asociadas.

En cuanto al siguiente objetivo: el relevamiento de estrategias de generación de modelos de procesos de negocio desde texto, el mismo fue abarcado en el estado del arte cuando se mencionan los trabajos similares en la sección 2.6.

Y en cuanto al siguiente objetivo: realizar un caso de estudio de aplicación de la propuesta en un dominio específico, este se menciona en el capítulo 7 donde se define el caso de estudio.

En resumen, los problemas a resolver son los siguientes:

- definir la representación de procesos de negocio a utilizar.
- analizar la representación seleccionada para tomar un subconjunto representativo de sus elementos.
- definir el lenguaje que represente los elementos del subconjunto seleccionado
- definir las perspectivas que se quieren obtener.
- definir las transformaciones necesarias para generar de forma automática las perspectivas seleccionadas.

Cabe aclarar que la definición del lenguaje es realizada en el capítulo 4 donde se realiza la

especificación de las oraciones del lenguaje y la formalización del mismo. De manera similar la selección de las perspectivas asociadas y la definición de las transformaciones necesarias para generar las mismas son mencionadas en el capítulo 5 de implementación de la solución.

3.2 Propuesta de la solución

Habiendo definido el problema del proyecto, continuamos con el planteo de una solución al mismo, donde mencionaremos uno por uno los problemas mencionados y detallaremos la solución propuesta.

3.2.1 Representación de procesos de negocio

El primer problema planteado implica definir qué representación se va a utilizar para modelar los procesos de negocio. Entonces, como fue mencionado en la sección 3 del capítulo 2, hay diferentes formas de representar procesos de negocio; sin embargo la organización ISO [14] adoptó en el 2013 a BPMN 2.0 como el estándar ISO/IEC 19510:2013 [15]. Además en [16] realiza una evaluación sobre la calidad de algunos de los lenguajes de modelado de procesos y concluye que BPMN 2.0 es el que presenta mejores atributos de calidad, resaltando: límites perceptivos y cognitivos, integración cognitiva, estructurado y simplicidad. Así es que BPMN 2.0 fue adoptado como estándar de facto para la industria y la academia. Por esto es que lo utilizaremos para el modelado de los procesos de negocio en este proyecto de grado.

3.2.2 Análisis de BPMN 2.0

Continuando con el planteo de la solución, entramos en el siguiente problema planteado, el cual implica que dado que el estándar BPMN 2.0 es muy extenso, resulta fundamental tomar un subconjunto de los componentes del mismo que conformen la base del problema a resolver. Para esto realizaremos un análisis del mismo de forma de tomar un subconjunto que sea representativo, es decir, que con los elementos contemplados se puedan modelar los procesos de negocio sin inconvenientes.

En el estado del arte en el capítulo 2, cuando se mencionó el estándar BPMN 2.0 en la sección 2.2, mencionamos que los elementos del estándar se pueden categorizar en cuatro grupos básicos:

- Elementos de flujo: Actividades, Eventos y Compuertas.
- Elementos de conexión: Flujo de secuencia, flujo de mensaje y asociación.
- Swimlanes: roles y participantes.
- Artefactos: Objetos de datos, grupos y anotaciones.

Empezaremos analizando los elementos de flujo, donde tenemos tres subgrupos: actividades, eventos y compuertas. Dentro de las actividades tenemos las siguientes: *Subprocess*, *Call Activity*, *Service Task*, *Send Task*, *Receive Task*, *User Task*, *Manual Task*, *Business Rule*, *Script Task*, *Global Task*. De las cuales consideramos como elementos más representativos los siguientes: *Subprocess*, *Service Task*, *User Task* y *Manual Task*.

Siguiendo con los elementos de flujo, en el subgrupo de las compuertas tenemos las siguientes: *Exclusive*, *Event-Based*, *Parallel Event-Based*, *Inclusive*, *Complex*, *Parallel*. De las cuales consideramos como elementos más representativos los siguientes: *Exclusive* y *Parallel*.

Para terminar con los elementos de flujo, nos queda considerar el subgrupo de los eventos, el cual

a su vez puede ser clasificado de la siguiente manera: *Start event, End event, Intermediate Event, Attached Intermediate Event*. Dentro de los *Intermediate Event* encontramos los siguientes: *Message, Timer, Escalation, Compensation, Conditional, Link, Signal, Multiple, Parallel Multiple*. Dentro de los *Attached Intermediate Event*: *Message, Timer, Escalation, Error, Cancel, Compensation, Conditional, Signal, Multiple, Parallel Multiple*.

Dentro de este subgrupo, consideramos como más representativo los siguientes: *Start event, End event, Intermediate Message, Intermediate Timer, Attached intermediate event timer*.

Continuando, analizamos los elementos de conexión donde tenemos tres elementos: Flujo de Secuencia, Flujo de Mensaje y Asociación. De los cuales consideramos como más representativos los siguientes: Flujo de Secuencia y Flujo de Mensaje.

Luego, analizamos los elementos de la categoría Swimlanes, los cuales son todos considerados como representativos ya que son utilizados para modelar roles y organizaciones.

Para finalizar, nos queda analizar los artefactos: Objetos de datos, grupos y Anotaciones. Dentro de los Objetos de datos tenemos los siguientes: *Property, Data Object* y *Data Store*. De estos consideramos sólo el elemento *Property* dentro de los Objetos de datos.

En la tabla 3.1 se muestra un resumen de los elementos considerados como más representativos, los cuales serán soportados en el lenguaje a especificar soportados y los que no.

Componente				Soportado
Elementos de flujo	Actividades	Subprocess ¹		Si
		Call Activity		No
		Tasks	Service Task	Si
			Send Task	No
			Receive Task	No
			User Task	Si
			Manual Task	Si
			Business Rule	No
			Script Task	No
			Global Task	No
	Eventos	Start Event		Si
		End Event		Si
		Intermediate	Message	Si

¹ Multi-Instance y Loop son soportados

		Event ²	Timer	Si
			Escalation	No
			Compensation	No
			Conditional	No
			Link	No
			Signal	No
			Multiple	No
			Parallel Multiple	No
		Attached Intermediate Event ³	Message	No
			Timer	Si
			Escalation	No
			Error	No
			Cancel	No
			Compensation	No
			Conditional	No
			Signal	No
			Multiple	No
			Parallel Multiple	No
	Compuertas	Exclusive		Si
		Event-Based		No
		Parallel Event-Based		No
		Inclusive		No
		Complex		No
		Parallel		Si
Elementos de conexión	Sequence Flow			Si
	Message Flow			Si

² En caso de ser soportado, se refiere tanto eventos de tipo throw como eventos de tipo catch

³ En caso de ser soportado, se refiere tanto a adjuntos interrumpibles como adjuntos no interrumpibles.

	Asociación		No
Swimlanes	Lanes		Si
	Pools		Si
Artefactos	Objetos de datos	Property	Si
		Data Object	No
		Data Store	No
	Grupos		No
	Anotaciones		No

Tabla 3.1. Resumen de los elementos del estándar que serán soportados por el lenguaje.

4 Definición y formalización del lenguaje

En este capítulo definimos y justificamos el formato de las oraciones válidas del lenguaje, indicando también cuáles son los elementos del estándar asociados. Luego en la siguiente subsección especificamos la gramática que gobierna a este lenguaje.

4.1 Especificación del lenguaje

Una vez obtenido el subconjunto de elementos del estándar que vamos a representar, estamos en posición de definir el lenguaje, de forma que permita representar el subconjunto de los elementos del estándar mencionados en la sección anterior. A continuación iremos explicando, para cada uno de los elementos, como es su representación en el lenguaje.

Swimlanes: roles y participantes.

Los pools cumplen el rol de contener la definición estructural del proceso. Para poder definir un pool, necesitamos saber el nombre del pool y cual va a ser el proceso que va a contener. Típicamente, el nombre del pool es el nombre de la organización que define el proceso. Pero en este caso lo construimos a partir del nombre del proceso. Cabe aclarar que este no se obtiene a partir del lenguaje directamente sino que es generado automáticamente. Para saber cual es el proceso que el pool va a contener, no alcanza con saber el nombre del proceso, sino que es necesario tener alguna referencia a la definición del proceso. Para entender esto se puede la representación en formato XML de algunos de los ejemplos definidos en el documento Estándar BPMN 2.0 entregado como anexo.

Habiendo definido la representación de los pooles, seguimos con la representación de los lanes. Estos representan los roles internos dentro del pool, los cuales típicamente coinciden con los roles de la organización que lo define y que participan en el proceso. Para poder construir estos elementos necesitamos saber el nombre del mismo, el cual se obtiene a partir de la definición de los actores en las oraciones del lenguaje. Estos actores son los encargados de realizar las actividades o tareas que pertenecen al proceso de negocio y son definidos en las oraciones del lenguaje, donde se especifica su nombre o etiqueta. Por otro lado, para definir un actor no solo necesitamos su etiqueta, sino que también es necesario un artículo en forma previa para identificarlo. Así que en un principio consideremos la siguiente forma de definir un actor.

“Artículo” “nombre actor”

ejemplo: El administrador o La encargada

Hay que considerar además los nombres simples (una palabra) o compuestos (más de una palabra), por ejemplo, encargado de ventas. Para poder utilizar la segunda forma es necesario que luego de especificar el actor se utilice una coma (“,”). En el siguiente ejemplo se muestra cómo se utilizan las dos formas.

La administradora realiza la tarea manual quema papeles.

El encargado de ventas, autoriza la venta.

Elementos de flujo: Actividades, Eventos y Compuertas

Estos elementos tienen que ser asignados a un rol o lane, para esto utilizaremos el identificador de cada elemento para establecer la referencia. Una vez asignados a un lane queda definir la especificación particular de cada uno.

Empezaremos con las diferentes actividades, seleccionadas del total de actividades de BPMN 2.0, estas actividades pueden ser de varios tipos: *tareas de usuario*, *tareas manuales*, *tareas de servicio* o *subprocesos*. Estas tareas comparten algo en común, y es que todas necesitan de un actor para que sean asignadas a un rol o lane, y por otro lado todas tienen un nombre de tarea.

Entonces como primera instancia, se podrían definir las diferentes tareas con una oración como la siguiente, “actor” “nombre de tarea”. Pero de alguna forma tendríamos que poder diferenciar las diferentes tareas. Entonces consideramos las siguientes oraciones para definir las diferentes tareas.

- Para las tareas de usuario utilizamos una oración de la forma “actor” “nombre de tarea”.,
- Para las tareas de servicio, “actor” utiliza el servicio “nombre de tarea”.,
- Para las tareas manuales, “actor” realiza la tarea manual “nombre de tarea”.
- Para los subprocesos “actor” realiza el subproceso “nombre de tarea”..

Algo importante a aclarar, es que la definición de los subprocesos son incluidos en el proceso donde se especifica su utilización. Por esto es necesario que el subproceso especificado en la oración haya sido definido previamente.

De manera de finalizar con las actividades, es importante destacar que estas se pueden ejecutar reiteradas veces, pero cuando definimos el alcance del proyecto, entendimos que no aportaba demasiado contemplar esta construcción para todas las actividades, por lo que solo la definimos para los subprocesos, donde entendemos es más común que se utilice.

Entonces, contemplando que los subprocesos pueden ser ejecutados reiteradas veces, sería necesario incluir alguna forma de definir esta repetición, por lo que definimos la siguiente oración.

“actor” realiza el subproceso “nombre de tarea”; varias veces.

De esta forma quedan definidas las diferentes actividades del estándar que son soportadas por el lenguaje.

Continuando con los siguientes elementos de flujo, los eventos, son elementos parecidos a las actividades. Estas también requieren un actor, pero no tienen un nombre. Hay diferentes tipos de eventos con diferentes tipos de comportamiento cada uno, pero los soportados por el lenguaje definido en este proyecto, son los mencionados a continuación.

Primero que nada tenemos los *eventos de inicio* los cuales representan el comienzo de un proceso. Como todo proceso definido tiene que empezar por uno evento de inicio, estos fueron definidos de forma implícita a la hora de construir los procesos, es decir, no se definió una oración

particular para generar este tipo de eventos.

Luego, tenemos los *eventos de fin*, que de manera similar a los eventos de inicio, estos están definidos de forma implícita, ya que todo proceso tiene que finalizar eventualmente. Pero existe una diferencia en cuanto a estos eventos y es que se pueden definir de forma explícita luego de la definición de una tarea de usuario, como se muestra en la siguiente oración definida.

“actor” “nombre de tarea”. y finaliza.

En estos casos se generan eventos de fin extras y se debe a que en la mayoría de los casos, es preferible tener un evento de fin extra, que un flujo de secuencia que atravesase todo el proceso para conectarse con el único evento de fin generado implícitamente. Esto contradice en parte lo mencionado en la sección 2.3, donde se comentan algunas buenas prácticas de modelado, en particular, la guía **G3**, la cual menciona que solo se debe utilizar un solo evento de inicio y un solo evento de fin, pero también permite reducir la complejidad del modelo

Siguiendo con los eventos, tenemos los *eventos intermedios*, los cuales tienen que ser definidos explícitamente al igual que las diferentes actividades. Los eventos seleccionados de todos los tipos de eventos de BPMN 2.0 son los de tiempo y de mensaje.

Los *eventos de tiempo* son utilizados para indicar que se debe esperar un tiempo antes de continuar con el proceso. Entonces, para poder generar este tipo de eventos necesitaremos un actor y un tiempo. Entonces definimos la siguiente oración para representar este tipo de eventos.

“actor” espera por “número” “unidad de tiempo”.

Los *eventos de mensajes* además pueden ser de dos tipos: *de envío o de recepción*. De la misma forma que los anteriores necesitan un actor, pero tienen una gran diferencia con los eventos de tiempo, y es que este tipo de eventos se utiliza para la comunicación con otros pools que participan del proceso. Entonces no solamente necesitamos un actor sino que también necesitamos el nombre del pool remitente para los de recepción y el nombre del pool destinatario para los de envío. Entonces, para representar este tipo de eventos utilizamos las siguientes oraciones, la primera es para los eventos de envío de mensajes y la segunda para los eventos de recepción de mensajes.

“actor” envía mensaje a “destinatario”.

“actor” espera por respuesta “remitente”.

Otra forma de utilizar los eventos intermedios, es de forma adjunta a una tarea o subproceso, donde se puede optar por dos opciones, que el *evento interrumpa la tarea o subproceso en caso de ser disparado (interrumpible) o que no lo interrumpa (no interrumpible)*. Es importante aclarar que solo se dio soporte a los eventos intermedios de tiempo en estas construcciones. Entonces, para generar este tipo de elementos, definimos las siguientes oraciones.

Para los eventos adjuntos no interrumpibles,

alternativa de “nombre de tarea”, si transcurre “número” “unidad de tiempo” (*) fin.

De esta forma estamos indicando que la tarea “nombre de tarea”, tiene un evento intermedio de tiempo adjunto, y en caso de dispararse dicho evento, esta tarea sigue ejecutando y en paralelo se inicia lo especificado en (*).

Luego para los eventos adjuntos interrumpibles,

alternativa de “nombre de tarea”, se interrumpe si transcurre “número” “unidad de tiempo” (*) fin.

El comportamiento es similar al obtenido de la oración anterior, pero en este caso se interrumpe la ejecución de la tarea “nombre de tarea”.

Y por último, consideramos como un caso particular, cuando el único flujo de salida de una tarea es a través de un evento adjunto siendo este interrumpible. En dicho caso, definimos la siguiente oración.

unica alternativa de “nombre de tarea”, se interrumpe si transcurre “número” “unidad de tiempo” (*) fin.

Para finalizar con los elementos del flujo nos queda hablar de las compuertas de las que solo se da soporte para las compuertas AND y XOR.

Las compuertas paralelas o compuertas AND sirven para generar varios flujos de ejecución en paralelo en un mismo proceso, estas permiten que diferentes elementos se ejecuten al mismo tiempo y luego se sincronicen. Entonces, para este tipo de compuertas necesitamos saber, cuántos flujos son generados y que es lo que sucede en cada uno de estos flujos. Para entenderlo mejor, a cada uno de estos flujos le llamaremos **bloqueAND_i** donde *i* representa el número de flujo. Le llamamos bloque, porque pueden definirse estructuras más complejas que una sola actividad o evento. Entonces, para construir este tipo de compuertas definimos la siguiente oración.

al mismo tiempo, 1 bloqueAND_1 ... N bloqueAND_N fin

Como dijimos, también se da soporte a las compuertas exclusivas o compuertas XOR, las cuales a diferencia de las compuertas AND, generan diferentes flujos de ejecución, pero sólo uno de ellos será ejecutado. Para esto es necesario utilizar el concepto de condición lógica. Las diferentes condiciones lógicas serán asociadas a los diferentes flujos generados a partir de la compuerta XOR, y se ejecutará el flujo cuya condición lógica sea evaluada verdadera. Entonces, primero que nada, como definimos una condición utilizando el lenguaje? A continuación pasaremos a explicar esto.

Las condiciones lógicas que utilizaremos constan de tres componentes, una variable, una constante y un operador lógico. Los operadores lógicos pueden ser, “es mayor que” (>), “es menor que” (<), “es”(==), “no es”(!=). entonces, la forma de definir las condiciones es la siguiente,

“variable” “operador” “constante”

Cabe aclarar que todas las constantes y variables son interpretadas como texto.

Siguiendo con las compuertas XOR, un detalle importante es que este tipo de compuerta necesita tener un flujo, el cual llamamos flujo defecto, que funciona como vía de escape en caso que ninguna de las condiciones lógicas sea evaluada verdadera. Este flujo no tiene una condición lógica

asociada. Finalizando, al igual que para las compuertas AND, a cada uno de los flujo posibles, le llamaremos **bloqueXOR_i** al cual le asociaremos la condición lógica **condicion_i** y ya que el bloque defecto, el cual llamaremos **bloque_def**, no tiene asociada una condición lógica, lo identificamos con la etiqueta “**si no**” antes de la definición del bloque defecto. Entonces para construir una compuerta XOR definimos la siguiente oración,

```
si se cumple, “condicion_1” “bloqueXOR_1” ... “condicion_N” “bloqueXOR_N”  
si no “bloque_def” fin
```

Un detalle importante es que existe la posibilidad de que uno de los flujos generados por la compuerta XOR sea un flujo vacío, es decir, es un flujo en el cual no hay ninguna actividad o evento para ejecutar. Este tipo de flujo, sólo puede usarse en uno solo de los flujos, es decir, no pueden haber dos flujos vacíos para una misma compuerta XOR. Para construir esto, es necesario utilizar la etiqueta “**no hace nada**” en la definición del bloque correspondiente, por ejemplo,

```
si se cumple, “condicion_1” “bloqueXOR_1” ... “condicion_N” no hace nada  
si no “bloque_def” fin
```

Definimos también una oración particular para la construcción del elemento LOOP basado en compuerta XOR. Este no es un elemento particular del estándar, pero cómo es bastante utilizado nos pareció interesante considerarlo como si lo fuera. Este tipo de construcción no genera flujos en paralelo o flujos excluyentes según una condición lógica, si no que toma un flujo y lo repite tantas veces como sea necesario según una condición lógica. Por lo tanto para definir este tipo de elementos necesitamos una condición lógica y un flujo, al cual llamaremos **bloqueLOOP**.

Este tipo de construcción utiliza compuertas XOR por lo que la idea es un poco similar, solo que en el caso de las compuertas XOR la evaluación de las condiciones lógicas se realiza al encontrarse con la compuerta XOR que genera los diferentes flujos y en el caso del LOOP, la condición es evaluada luego de ejecutar el flujo, en la compuerta XOR que se podría decir, cierra la construcción. Entonces para esta construcción definimos la siguiente oración,

```
mientras “condicion_LOOP”, “bloqueLOOP” fin
```

Al igual que los bloques **bloqueAND_i** y **bloqueXOR_i** en las compuertas AND y XOR, **bloqueLOOP** puede ser una estructura compleja, no solo una actividad o evento.

Elementos de conexión: Flujo de secuencia, flujo de mensaje y asociación.

Los elementos de conexión que son soportados son los flujos de secuencia y los flujos de mensaje, pero para generarlos no se definió una oración del lenguaje, sino que estos son generados cuando corresponde, es decir, los flujos de mensaje son generados cuando se generan los eventos de mensaje y los flujos de secuencia son generados a medida que se definen los elementos del flujo, ya que con los flujos de secuencia se define el orden de ejecución de los mismos.

Artefactos: Objetos de datos, grupos y anotaciones.

Por último, nos queda mencionar los artefactos, de los cuales solo son soportados los objetos de datos, en particular, como formularios en las tareas de usuario. Entonces, cuando definimos tareas de usuario, es de interés que se puedan definir datos de forma que para poder realizar estas tareas sea necesario completar un formulario o simplemente que la tarea consiste en desplegar

un formulario de solo lectura.

Entonces, para poder construir este tipo de formularios precisamos saber si va a ser un formulario de lectura o escritura y para cada campo, precisamos saber el tipo y en caso que el formulario sea de escritura, si el campo es obligatorio o no. Entonces, para los formularios de escritura definimos la siguiente oración,

“actor” “nombre de tarea”. “nombre de tarea”, es un formulario con los siguientes campos: “nombre campo 1” que es un “tipo campo 1” [obligatorio],...,“nombre campo N” que es un “tipo campo N” [obligatorio].

Y para los formularios de solo lectura definimos la siguiente oración.

“actor” “nombre de tarea”. “nombre de tarea”, muestra los campos: “nombre campo 1”,...,“nombre campo N”.

Entonces, en conclusión, definimos un lenguaje simple, capaz de representar los elementos del estándar que se mencionaron en la sección 3.2.2 y dicho lenguaje consta de un conjunto de oraciones el cual se muestra en la tabla 3.2.

1	tarea de usuario	“actor” “nombre de tarea”.
2	tarea de usuario con fin implícito	“actor” “nombre de tarea”. y finaliza.
3	tarea de servicio	“actor” utiliza el servicio “nombre de tarea”.
4	tarea manual	“actor” realiza la tarea manual “nombre de tarea”.
5	subproceso	“actor” realiza el subproceso “nombre de tarea”.
6	subproceso con loop	“actor” realiza el subproceso “nombre de tarea”; varias veces.
7	evento de timer	“actor” espera por “número” “unidad de tiempo”.
8	evento de envío de mensaje	“actor” envía mensaje a “destinatario”.
9	evento de recepción de mensaje	“actor” espera por respuesta “remitente”.
10	evento adjunto no interrumpible	alternativa de “nombre de tarea”, si transcurre “número” “unidad de tiempo” (*) fin.
11	evento adjunto interrumpible	alternativa de “nombre de tarea”, se interrumpe si transcurre “número” “unidad de tiempo” (*) fin.

12	evento adjunto con única alternativa	única alternativa de “nombre de tarea”, se interrumpe si transcurre “número” “unidad de tiempo” (*) fin.
13	compuerta AND	al mismo tiempo, 1 bloqueAND_1 ... N bloqueAND_N fin
14	compuerta XOR	si se cumple, “condicion_1” “bloqueXOR_1” ... “condicion_N” “bloqueXOR_N” si no “bloque_def” fin
15	loop	mientras “condicion_LOOP”, “bloqueLOOP” fin
16	formulario escritura	“actor” “nombre de tarea”. “nombre de tarea”, es un formulario con los siguientes campos: “nombre campo 1” que es un “tipo campo 1” [obligatorio],...,“nombre campo N” que es un “tipo campo N” [obligatorio].
17	formulario lectura	“actor” “nombre de tarea”. “nombre de tarea”, muestra los campos: “nombre campo 1”,...,“nombre campo N”.

Tabla 4.1. Oraciones definidas del lenguaje

4.2 Formalización del lenguaje

Para formalizar el lenguaje especificado en la sección anterior, utilizamos una gramática PEG[17] (parsing expression grammar) las cuales pueden ser modeladas con la notación ebnf⁴[18]. La principal ventaja de utilizar este tipo de gramáticas, es que son deterministas, se basan en un criterio de prioridad, lo que las convierte en una buena alternativa para parsear textos estructurados. En nuestro caso, como definimos un lenguaje estructurado y sin ambigüedades, este tipo de gramáticas son ideales.

Habiendo definido el lenguaje, queda definir la gramática para parsear el lenguaje y generar a partir del mismo los elementos BPMN 2.0.

Entonces, como toda gramática, esta se compone de un elemento de inicio, elementos terminales, elementos no terminales y reglas de producción. En la gramática que definimos, el elemento de inicio se identifica en la Figura 4.1 en la parte izquierda de la primer regla de la gramática como “start”. Los elementos terminales se pueden identificar como los que están entre comillas (Ej. “el”, “utiliza el servicio”, “no hace nada”, etc.). Por otro lado, los no terminales son los que no se encuentran entre comillas (Ej. *prefijo_tarea_subproceso*, *operador*, *prefijo_compuerta_AND*, etc.). Y por último las reglas de producción son todas las presentadas en la Figura 4.1 donde se agruparon las diferentes reglas que son utilizadas para generar cada componente, de la mejor manera posible y agregando un comentario que indique para que son utilizadas.

Regla de inicio de la gramática.

⁴ Extended Backus-Naur Form (Notación de Backus-Naur extendido)

```
start ← secuencia ws
```

Regla utilizada para generar uno o más elementos de forma secuencial

o para cada bloque generado en las compuertas.

```
secuencia ← (ws elementos)+
```

Regla utilizada para generar actividades, eventos o compuertas.

En esta se decide qué elemento se va a generar.

```
elementos ← construccion_evento_intermedio |  
            construccion_tareas finaliza? |  
            construccion_compuerta_AND |  
            construccion_compuerta_OR |  
            construccion_evento_adjunto |  
            construccion_loop
```

Reglas utilizadas para generar las diferentes actividades.

```
construccion_tareas ← ws actor ws prefijo_tarea_manual ws accion ws "." |  
                    ws actor ws prefijo_tarea_servicio ws accion ws "." |  
                    ws actor ws prefijo_tarea_subproceso ws accion (ws ";" ws  
                    subproceso_loop)? ws "." ws |  
                    ws actor ws accion ws "." ws (construccion_formulario?)  
accion ← ([a-z_]i+ ws)*  
prefijo_tarea_servicio ← "utiliza el servicio"  
prefijo_tarea_manual ← "realiza la tarea manual"  
prefijo_tarea_subproceso ← "realiza el subproceso"  
subproceso_loop ← "varias veces" | "muchas veces"  
finaliza ← "y finaliza"
```

Reglas utilizada para generar los formularios de lectura/escritura asociados a

las tareas de usuario.

```
construccion_formulario ← (palabras "," | palabra) ws formulario_id ws palabras  
                        ws ":" ws formulario "." ws | (palabras "," | palabra)  
                        ws ver_form_id ws palabras ws ":" ws  
                        formulario_solo_lectura "." ws  
formulario ← campo+  
campo ← ws palabra ws auxCampo ws tipo ws obligatorio? ws ","? ws  
auxCampo ← "que es un" | "que es una"  
formulario_solo_lectura ← campo_solo_lectura+  
campo_solo_lectura ← ws palabra ws ","? ws  
obligatorio ← "obligatorio"  
formulario_id ← "es un formulario"  
ver_form_id ← "muestra" | "despliega"
```

Reglas utilizadas para generar los eventos intermedios.

```
construccion_evento_intermedio ← ws actor ws evento_catch ws construccion_evento  
                                ws "." | ws actor ws evento_throw ws pool ws "."  
evento_throw ← "envia mensaje a" | "envia respuesta a"  
evento_catch ← "espera por"  
mensaje ← "mensaje" | "mail" | "respuesta"
```

Reglas utilizadas para generar un evento adjunto.

```
construccion_evento_adjunto ← ws unica?  
                             ws prefijo_evento_adjunto ws palabras ws ","  
                             ws adjuntoInterrumpible?  
                             ws auxiliar_evento_adjunto ws construccion_evento  
                             ws secuencia ws
```



```

"fin"
prefijo_evento_adjunto ← "alternativa de"
auxiliar_evento_adjunto ← "si transcurre" | "si llega" ws mensaje
adjuntoInterrumpible ← "se interrumpe"
unica ← "unica"

# Regla utilizada para generar un evento de tiempo o de mensaje.
# Es utilizada por las reglas que generan los eventos adjuntos o eventos
# intermedios.
construccion_evento ← digito ws tiempo | mensaje ws articulo ws palabras

# Regla utilizada para generar una compuerta AND.
construccion_compuerta_AND ← ws prefijo_compuerta_AND ","
ws Integer secuencia
(ws Integer secuencia)+ ws
"fin"
prefijo_compuerta_AND ← "al mismo tiempo" | "a la vez" | "en paralelo"

# Reglas utilizadas para generar una compuerta XOR.
construccion_compuerta_OR ← ws prefijo_compuerta_OR ","
ws (condicion entonces ws secuencia_or)
(ws condicion entonces ws secuencia_or)*
ws defecto ws secuencia_or ws
"fin"
secuencia_or ← noHaceNada | secuencia
condicion ← termino (operador termino)?
termino ← ws [a-zA-Z0-9]+ ws
prefijo_compuerta_OR ← "si se cumple"
entonces ← "entonces"
defecto ← "si no"
noHaceNada ← "no hace nada"

# Reglas utilizadas para generar un LOOP.
construccion_loop ← ws prefijo_loop ws condicion "," secuencia ws "fin"
prefijo_loop ← "mientras"

# Reglas utilizadas para los actores.
articulo ← "el" | "la" | "los" | "las" | "del" | "de" | "una" | "un"
actor ← articulo ws ([a-z ]i+ ws) "," ws | articulo ws [a-z]i+

# Reglas auxiliares.
tipo ← "texto" | "numero" | "fecha" | "booleano"
operador ← "no es" | "es mayor que" | "es menor que" | "es"
tiempo ← "segundos" | "minutos" | "horas" | "días" | "semanas" | "meses" |
"años" | "segundo" | "minuto" | "hora" | "día" | "semana" | "mes" | "año"
Integer ← [0-9]+
digito ← [0-9]+
palabra ← [a-z]i+
palabras ← ([a-z]i+ ws?)+
ws ← [ \t\n\r]*

```

Figura 4.1. Gramática del lenguaje definido.

5 Desarrollo de la Solución

En este capítulo describimos los aspectos involucrados en la implementación de la solución, esto implica una serie de transformaciones aplicadas sobre el lenguaje, al cual a partir de ahora lo llamaremos modelo textual, hasta obtener el modelo BPMN 2.0. En la Figura 5.1 presentamos el proceso de generación como un modelo BPMN 2.0.

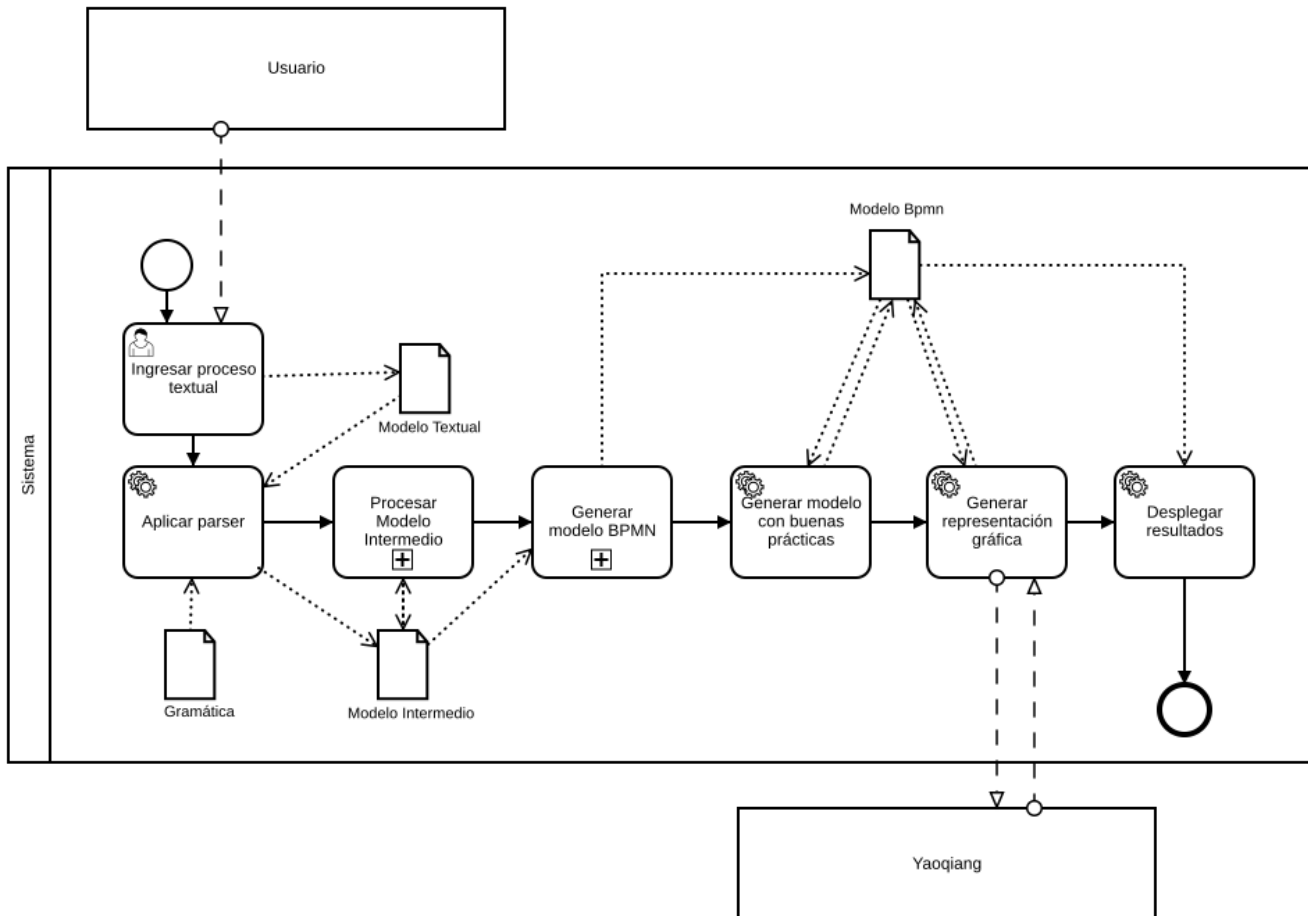


Figura 5.1. Modelo BPMN 2.0 del proceso de la solución

Este modelo presenta una tarea de usuario para la recolección del modelo textual del proceso. A partir de allí se aplican una serie de tareas automáticas para lograr el objetivo de obtener el modelo final. Por lo tanto, una vez obtenido el modelo textual se pasa a ejecutar la tarea “Aplicar Parser”, que se utiliza para obtener una estructura formal a partir del modelo textual. Siguiendo el flujo de secuencia la siguiente etapa es el subproceso “Procesar modelo intermedio” que consiste en una serie de tareas automáticas para trabajar sobre la estructura formal obtenida en la tarea previa. El paso siguiente es “Generar el modelo BPMN 2.0”, que consiste en otra serie de transformaciones automáticas para obtener el modelo en el formato correcto. Una vez obtenido el modelo BPMN 2.0, la siguiente actividad en la secuencia es una tarea de servicio “Generar modelo con buenas prácticas” que permite mejorar la calidad del modelo BPMN 2.0 generado. Finalmente, la última tarea se encarga de presentar todos los elementos necesarios para desplegar el modelo BPMN 2.0 en la aplicación.

5.1 Ingresar proceso textual

Lo primero que se realiza en el proceso de la figura 5.1 es ingresar el modelo textual del proceso, el cual debe respetar las reglas del lenguaje mencionadas en el capítulo 4. En caso de no respetar estas reglas, se producirá un error al momento de intentar generar el modelo BPMN 2.0 donde el proceso finaliza de manera abrupta. Esto no se ve reflejado en el proceso ya que se modelo suponiendo el caso en que el modelo textual es correcto. A su vez, como se verá más adelante en el capítulo 6, cuando se ingresa el modelo textual, también se ingresa el nombre del proceso.

5.2 Aplicar parser

Con el lenguaje definido y formalizado utilizando una gramática, estamos en posición de generar un parser. Un parser es un analizador sintáctico capaz de determinar si una tira de caracteres es válida para un lenguaje definido por una gramática, además de retornar un *abstract syntax tree grammar*, que es una estructura que contiene los componentes sintácticos del texto analizado. A partir de esta estructura sintáctica buscamos mediante una serie de transformaciones generar el modelo BPMN 2.0.

Para ello utilizamos `peg.js`[19], una herramienta capaz de generar un parser a partir de una gramática en formato similar al `ebnf` presentado en la Figura 4.1 del capítulo 4, donde a cada regla es posible asignarle un bloque de código a ejecutar al momento de ser parseado pasándole como parámetros las reglas reconocidas; de esta forma es posible asignar a cada regla un valor de retorno para generar una estructura sintáctica en caso que la tira sea válida. En la Figura 5.2 se puede ver la construcción sintáctica genérica de la gramática `peg.js`. A la estructura retornada por el parser le llamamos modelo intermedio y contendrá elementos sintácticos del texto parseado.

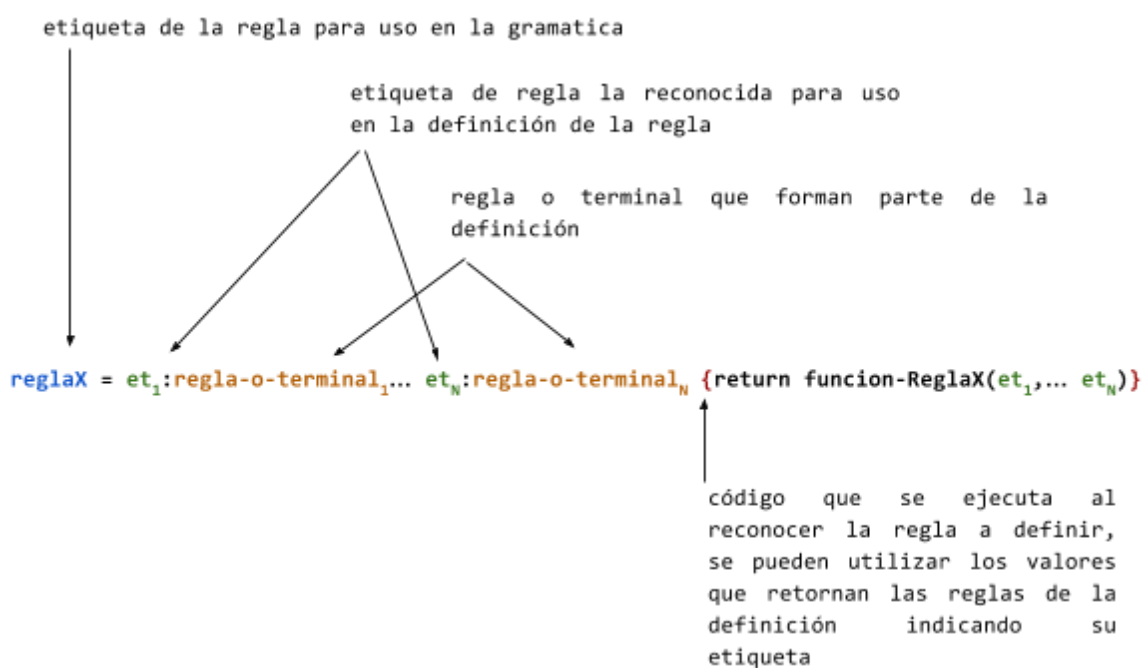


Figura 5.2. Estructura sintáctica genérica de la gramática `peg.js`

5.2.1 Modelo Intermedio

Una vez aplicado el parser sobre un texto a procesar obtenemos el **modelo intermedio**. Esta estructura es un objeto JavaScript [20] con los siguientes elementos:

{tipo:secuencia, sentencia: [lista de nodos]} Se retorna cuando una secuencia es reconocida en el texto. Puede ser en la secuencia principal o dentro construcciones compuestas como en el reconocimiento de compuertas, loop o eventos adjuntos.

{tipo:task, sentencia: {accion: descripcionAccion, actor: nombreActor} } Se obtiene cuando es reconocida una tarea, puede tener información adicional sobre datos.

{tipo:evento, sentencia: {evento: {evento}, actor: nombreActor, tipo: tipoDeEvento} } Surge cuando un evento es reconocido. El tipo puede ser mensaje o tiempo.

{tipo:and, sentencia: [lista de secuencias]} Se retorna al reconocerse una compuerta paralela.

{tipo:xor, sentencia: [lista de secuencias]} Se retorna al reconocerse una compuerta exclusiva.

{tipo:adjunto, sentencia: {adjunto_a: descripcionAccion, secuencia:[lista de nodos]}} Es retornado si se reconoce un evento adjunto,

{tipo:mientras, sentencia: { condicion:condicion, secuencia:[lista de nodos]} } Identifica sentencias.

A modo ilustrativo, el modelo intermedio podría ser instancia del siguiente diagrama de clases uml representado en la Figura 5.3 (se incluyeron solo relaciones relevantes):

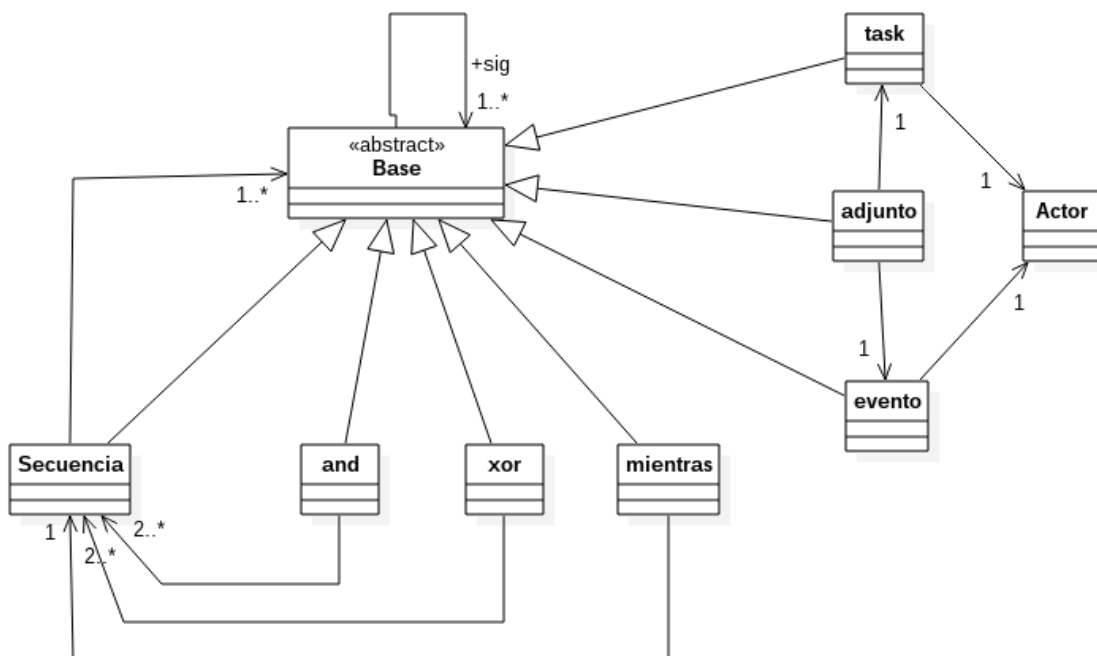


Figura 5.3. Diagrama de clases del modelo intermedio

5.3 Procesar el modelo intermedio

En el modelo intermedio es sencillo asociar cada uno de los elementos con una construcción del estándar BPMN 2.0, como los son actividades y sus artefactos de datos asociados, eventos, lanes, participantes y flujo de mensajes. Sin embargo uno de los componentes más complejos que restan por determinar es el flujo de secuencia, para los cuales hay que tener en cuenta algunos aspectos claves como son las secuencias, las compuertas y las tareas adjuntas.

Sin embargo la estructura básica del modelo intermedio es la secuencia, lo cual puede ser contradictorio puesto que la secuencia no es una estructura nativa del estándar BPMN 2.0, es decir, no hay una etiqueta secuencia.

Estas secuencias tiene una lista con el orden en que se acceden a los elementos. Esto se debe a que al momento de obtener el modelo ninguno de los elementos tiene un id único asociado, por ello es necesario ubicarlos dentro de listas para mantener el orden.

- Modelo intermedio: El flujo viene determinado por el orden en que aparecen en la secuencia (lista) o anidando elementos para el caso de flujo complejo.
- Modelo BPMN 2.0: El flujo viene determinado por el id y las etiquetas de flujo de secuencia donde se indica el id de entrada y el de salida.

Una vez obtenido el modelo intermedio es necesario aplicarle una serie de transformaciones hasta lograr el modelo XML. Para ello en primer lugar aplicaremos una serie de transformaciones que le agregaran metainformación al modelo, agregando ids a los nodos, agregando el id del siguiente elemento en ejecutarse. Y por otro lado tenemos la transformación que toma este modelo JSON [21] transformado y lo transforma en un XML, es decir que modifica la estructura básica del modelo.

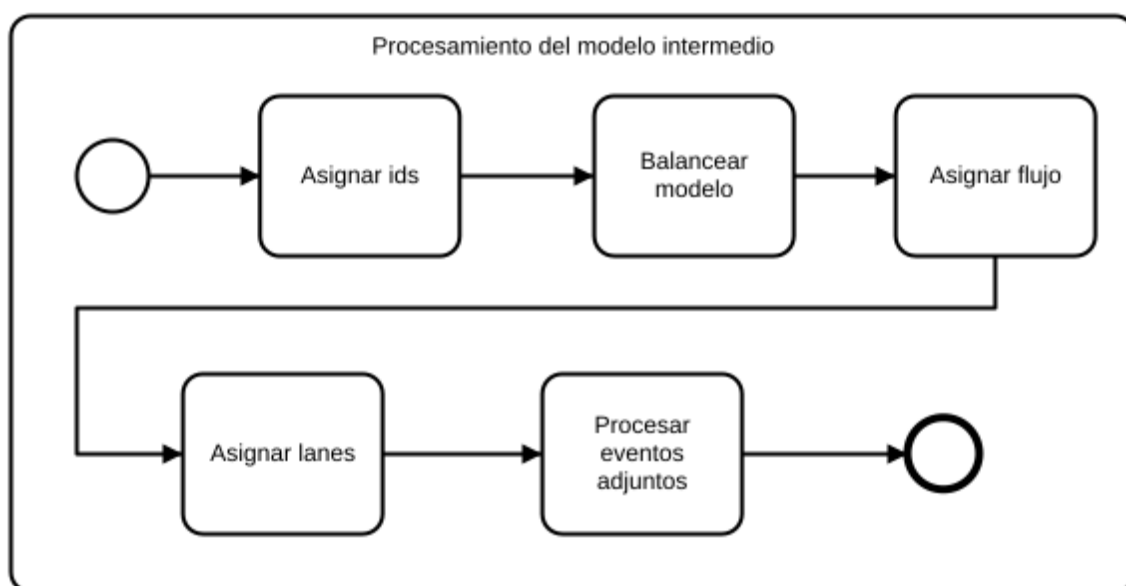


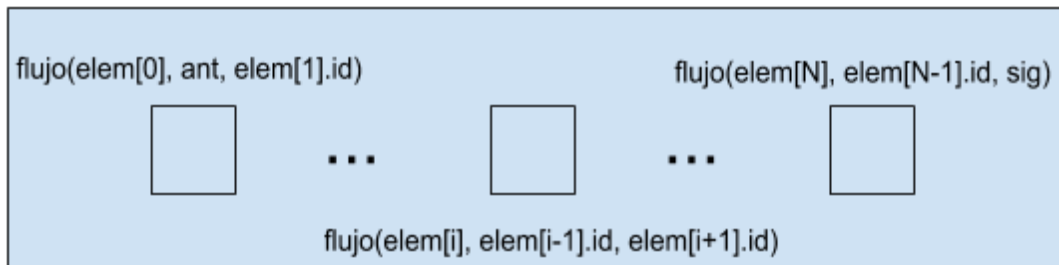
Figura 5.4. Subproceso 'Procesamiento del modelo intermedio'

A continuación se detallan las transformaciones descritas en el subproceso, que agregan información al modelo pero no modifican su estructura, en el orden que fueron aplicadas. Ver Figura 5.4.(el orden en la mayoría de los casos importa, por ejemplo, no podríamos determinar el id del siguiente si todas las tareas no tuvieran un id)

1. **asignarId**: Dado que el modelo BPMN 2.0 utiliza identificadores en cada una de sus construcciones, es menester agregar un identificador a cada elemento definido. Es la primera acción que debemos realizar puesto que necesitaremos los identificadores para definir flujo, asignar eventos adjuntos, asociar lanes y definir datos.
2. **balancerarModelo** Esta operación se encarga de balancear las compuertas del modelo generado. Para cada elemento del modelo intermedio del tipo “xor”, “and”, “mientras” o “adjunto e interrumpible”, se genera un elemento del tipo “cierro” y se le coloca como siguiente elemento dentro de la secuencia. De esta forma asegurar aspectos de calidad, dado que asegura que el modelo esté estructurado acorde a la regla G4 la cual es la mejor posicionada en el ranking.
3. **asignarFlujo**: Es la transformación más compleja, debe iterar sobre el modelo intermedio, con sus definiciones anidadas, asignándole a cada elemento definido el id de la siguiente actividad a ejecutar. Esta función se ejecuta de manera recursiva, tomando como parámetro un elemento del modelo intermedio, el anterior elemento y el siguiente. La primera ejecución debe ser ejecutada sobre la secuencia inicial, con los nodos virtuales Inicio y Fin, como anterior y siguiente respectivamente, asegurando que haya un único evento de inicio como sugiere la regla G3 de la sección 2.3. En el caso de compuertas y evento, simplemente setea los valores de siguiente y anterior a los pasados por parámetros. Para el flujo de una secuencia o una compuerta se determinan asignando el flujo de cada uno de sus elementos internos como puede verse en la Figura 5.5
4. **asignarLanes**: Esta transformación extrae los datos sobre el actor que realiza la tarea o evento y le agrega un atributo lane. Itera sobre todo el modelo, y en caso de tener propiedad actor, le asigna su valor a la propiedad lane y actualiza la variable laneActual; en caso de no tener la propiedad actor le asigna el valor laneActual.
5. **corregirFlujoSecuencia**: La transformación se encarga de corregir los flujos de las secuencias. La estructura secuencia del modelo intermedio no tiene una correspondencia directa con una construcción de BPMN 2.0, para indicar flujo secuencial en BPMN 2.0, simplemente hay que indicar el siguiente de cada elemento en la secuencia. Por ello se realiza una corrección en cada elemento del modelo intermedio y si algún elemento tiene como siguiente el id de una secuencia, entonces, cambia ese id por el id del primer elemento en la secuencia.
6. **procesarEventosAdjuntos**: Esta función se encarga de asociar los eventos adjuntos con su respectiva tarea según su nombre. Busca una tarea en el modelo intermedio con el nombre indicado y le asocia el id de la misma. Si no hay tarea con ese nombre, finaliza el procesamiento indicando un error.

Entonces, una vez aplicadas las transformaciones horizontales sobre el modelo intermedio, este contará con toda la metainformación para poder construir el modelo BPMN 2.0 en formato XML.

flujo(secuencia, ant, sig)



flujo(compuerta, ant, sig)

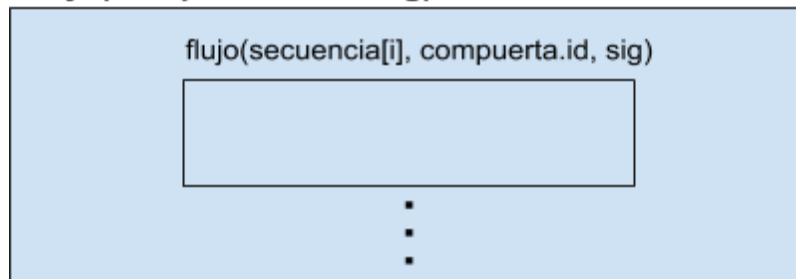


Figura 5.5 Llamadas recursivas sobre estructuras complejas del modelo abstracto

5.4 Generar modelo BPMN 2.0

Para generar los modelos BPMN 2.0 en formato XML, utilizamos la librería x2js [22], que permite pasar de una estructura en formato JSON a su equivalente en formato XML y viceversa. Para esto fue necesario aplicar transformaciones al modelo intermedio, de forma de poder obtener la estructura necesaria para generar el formato XML según el estándar BPMN 2.0.

Para esto se realiza una transformación en tres pasos como se muestra en la figura 5.6.

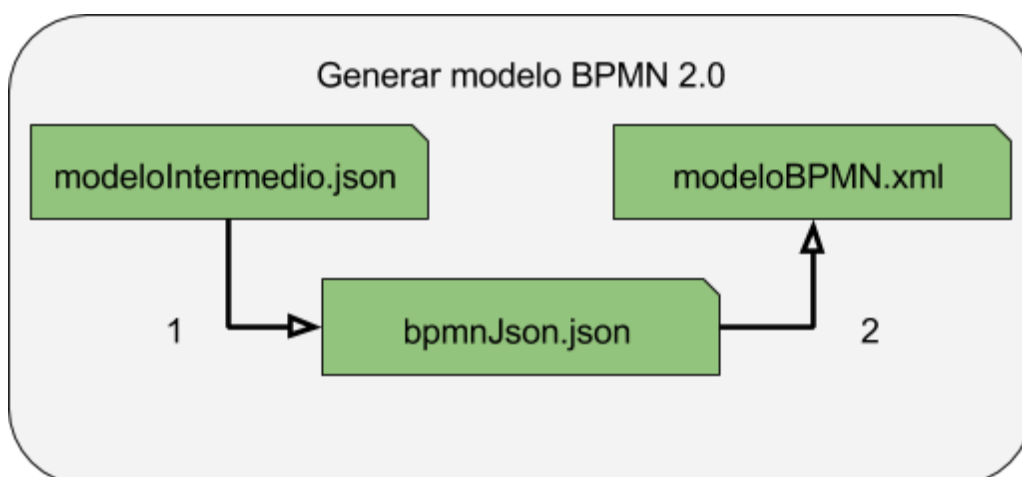


Figura 5.6. Generación de los modelos BPMN 2.0 en formato XML.

La primer transformación, parte del artefacto `modeloIntermedio.json` y genera el artefacto `bpmnJson.json` el cual tiene la estructura que se ve en la figura 5.7, que es una estructura similar a la estructura que tienen los modelos BPMN 2.0 en formato XML pero no es exactamente igual ya que es un artefacto en formato JSON, es decir, sería la representación en formato JSON de la estructura que tienen los modelos BPMN 2.0 en formato XML. Para finalizar, aplicamos la segunda transformación utilizando la librería `x2js`, de forma de hacer la conversión de formato JSON a formato XML obteniendo así el modelo BPMN 2.0 en formato XML.

```
collaboration : {
  participant : [],
  messageFlow : []
},
process : {
  laneSet : {
    lane : []
  },
  startEvent : [],
  userTask : [],
  manualTask : [],
  serviceTask : [],
  exclusiveGateway : [],
  parallelGateway : [],
  intermediateCatchEvent : [],
  intermediateThrowEvent : [],
  boundaryEvent : [],
  endEvent : [],
  subProcess : [],
  sequenceFlow : []
}
```

Figura 5.7. Estructura del artefacto `bpmnJson.json`

Las transformaciones necesarias para obtener el modelo con la estructura mencionada en la figura 5.7 consisten en varias etapas. Primero que nada, necesitamos saber cuales son los lanes del proceso, es decir, los actores que participan del proceso. Para esto iteramos sobre el modelo intermedio buscando los elementos que tienen asociado un actor. Estos elementos son los diferentes tipos de tareas, subprocessos y eventos.

Luego de haber obtenido todos los lanes, es momento de construir cada elemento. Por ejemplo, si en el modelo intermedio encontramos un elemento como el que se muestra en la figura 5.8, se debería generar una tarea de usuario con el identificador (*id*) y nombre (*acción*) correspondiente y agregar a la colección de tareas de usuario (*userTask : []*).


```

{
  "tipo": "task",
  "sentencia": {
    "actor": "personal",
    "accion": "evalua el cv del postulante",
    "task": "human",
    "campos": null
  },
  "doc": "el personal evalua el cv del postulante.\n",
  "finaliza": false,
  "id": 3,
  "sig": [
    4
  ],
  "ant": [
    "S"
  ],
  "lane": "personal"
}

```

Figura 5.8. Ejemplo de tarea de usuario en el modelo intermedio.

Continuando, luego de haber obtenido todos los elementos, asociamos estos elementos a los lanes correspondientes, esto es, agregar en el lane correspondiente, una referencia a dicho elemento utilizando el id correspondiente. En el caso de la tarea de usuario de la figura 5.8, se debería agregar una referencia a la misma utilizando su id, en el lane asociado al actor personal.

En el caso de los elementos que no tienen un actor asociado, es decir, no tienen un lane al cual asociarlo, estos fueron asociados a un lane de forma que el modelo gráfico quedará lo más claro posible. Típicamente, el mismo lane que el elemento anterior. De esta forma se va logrando una distribución de los elementos en los lanes.

Luego, se generan los elementos correspondientes a los flujos de secuencia, consiguiendo el grafo dirigido que representa el orden de ejecución del proceso. Para generar estos flujos de secuencia se recorre la estructura del modelo intermedio y para cada elemento se ve quienes son los componentes siguientes (indicado en el campo *sig* como una lista de identificadores). Entonces, para cada elemento que se encuentra en esta lista, se genera un flujo de secuencia, donde el id del componente origen es el id del elemento que se está procesando y el id del componente destino es uno de los que aparecen en esta lista.

Siguiendo con el procesamiento, el siguiente paso es conectar el evento de inicio y asignarlo al mismo lane que al primer elemento del modelo intermedio ya procesado.

En caso de haber un subproceso, es necesario importar el modelo de este, transformarlo en un modelo JSON, descartar lo que sobra e incluirlo en la definición del subproceso. Esto se debe a que en el modelo intermedio solo se tiene el nombre del subproceso y en que lane debe ser incluido,

pero no la información interna del mismo, esta información está almacenada en su correspondiente modelo BPMN 2.0.

Fue necesario realizar un pequeño ajuste a los identificadores de los elementos del subproceso incluido, agregándoles un prefijo. Para los elementos internos de los subprocesos, el prefijo se corresponde con el nombre del subproceso seguido de un guión bajo (“_”) y el identificador original del mismo y en el caso de los elementos del proceso el prefijo es solamente el guión bajo.

5.4.1 Perspectivas Asociadas

Dado que no solamente es de interés generar artefactos gráficos, es decir, el modelo del proceso de negocio en formato BPMN 2.0, sino que también uno de los objetivos es generar artefactos ejecutables, decidimos presentar el modelo BPMN 2.0 en tres formatos distintos.

El primero, un formato básico, el cual contiene los artefactos básicos del estándar BPMN 2.0, como definición de actividades, eventos, participantes y flujo. De esta forma se puede obtener un modelo gráfico simple y fácil de entender. Hay que mencionar que en este formato no se incluyen los artefactos relacionados con la representación de datos.

Por otro lado, se genera un formato estándar el cual tiene los mismos artefactos que el básico y agrega artefactos de datos y de asignación de participantes como se especifica en el estándar BPMN 2.0, de manera de introducir elementos de ejecución propios del estándar. Esta perspectiva puede ser ejecutada en el motor incluido en el editor de procesos de negocio Yaoqiang [23].

Y por último, un formato específico listo para ejecutar en Activiti. Para cumplir con esto fue necesario realizar una serie de cambios sobre el modelo estándar con el objetivo de evitar algunas construcciones, agregar otras y utilizar extensiones del estándar propias de Activiti [24]; detallamos estos cambios en los siguientes párrafos.

En el sistema, los eventos de mensaje y los pooles externos vienen de la mano, es decir, por cómo construimos el modelo no va a haber un pool externo si no hay un evento de mensaje y dado que la configuración necesaria para ejecutar estos en Activiti es muy compleja, y en si no aporta contenido al objetivo del proyecto. Tomamos la decisión de cambiar estos por otro tipo de componentes de forma de simular este tipo de comportamiento. Para poder ejecutar estos procesos en Activiti, los pooles externos se eliminan arbitrariamente y los eventos de mensaje son cambiados según si es un evento de recepción de mensaje o un evento de envío de mensaje, de la siguiente manera: en caso de ser un evento de recepción de mensaje, este se cambia por una tarea de usuario. Lo que se intenta con este cambio es representar la recepción de un mensaje con una interacción por parte del usuario. En cambio, si se trata de un evento de envío de mensaje, este se cambia por una tarea de servicio de tipo mail como se muestra en la figura 5.9, la cual simula el envío de un correo.

```
<serviceTask id="sendMail" activiti:type="mail">
```

Figura 5.9. Tarea de servicio de tipo mail.

Para definir formularios sobre tareas de usuario se le asoció una entrada y salida, sin embargo, Activiti no da soporte para este tipo de construcciones. Por ello se cambió la especificación de la

entrada y salida (ioEspecificacion) por una extensión del estándar, realizada por Activiti, donde se define los campos del formulario, obteniendo así el formato que se puede ver en la figura 5.10 para una tarea de usuario.

```
<userTask id="task">
  <extensionElements>
    <activiti:formProperty id="room" />
    <activiti:formProperty id="duration" type="long"/>
  </extensionElements>
</userTask>
```

Figura 5.10. Definición de un formulario en una tarea de usuario en Activiti

Un detalle menor surge a la hora de establecer condiciones sobre las expresiones de las compuertas exclusivas, y es que Activiti reconoce las expresiones si se encuentran envueltas en el siguiente formato ***$\{expresión\}$*** . Por lo tanto a cada expresión de un flujo de secuencia asociado a una compuerta exclusiva se le envuelve en con la notación indicada.

Dado que Activiti retorna un error si se llega a ejecutar un flujo de secuencia con una condición que contenga variable no definida, a los eventos de inicio se les agrega información sobre todas las variables involucradas en el proceso. Para esto, al igual que como se define un formulario sobre una tarea de usuario, se le agrega a los eventos de inicio, por cada variable del proceso, un campo para inicializar la variable.

En última instancia, para que el proceso sea ejecutable hay que agregar información sobre que rol está autorizado a tomar cada tarea de usuario. Para esto a cada tarea de usuario se le agrega se le agrega el atributo *activiti:candidateGroups* a partir del nombre del lane en que se encuentre ubicada. Luego para que el modelo pueda ser ejecutado por usuarios, en el BPMS se deberán asociar usuarios a los roles definidos.

5.5 Generar modelo con buenas prácticas

Una vez obtenido el modelo BPMN 2.0 se realiza un procesamiento para mejorar su calidad, esto implica realizar una transformación que implica remover las compuertas exclusivas innecesarias como se explica en el siguiente párrafo.

Si bien el objetivo es generar modelos con compuertas balanceadas hay ciertas construcciones del lenguaje que permite alterar el flujo dentro de compuertas exclusivas y eventos adjuntos para introducir eventos de finalización. De esta manera, dado una compuerta exclusiva con N flujos, si N-1 de estos finalizan de manera prematura sería innecesario el balanceo de dicha compuerta dado que una compuerta xor quedaría con un único flujo de entrada y un solo flujo de salida; esto se puede ver en la Figura 5.11. Como se puede apreciar, la compuerta en rojo carece de sentido puesto que es un punto de sincronización con una sola entrada. Por lo tanto, esta etapa del procesamiento se centra en iterar sobre todas las compuertas exclusivas y para cada una de ellas, si tiene un solo flujo de entrada y uno solo de salida entonces se elimina dicha compuerta corrigiendo el flujo asociado a los elementos afectados. De esta forma minimizamos la cantidad de

elementos utilizados, como sugiere la regla **G1** indicado en la sección 2.3.

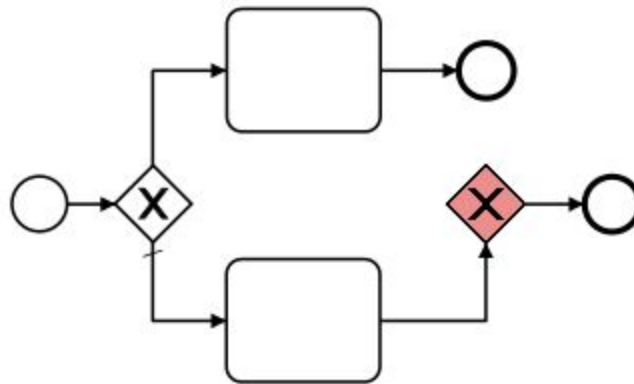


Figura 5.11 Identificación de compuertas innecesarias

5.6 Generar representación gráfica.

Hasta el momento el modelo BPMN 2.0 generado únicamente define los componentes lógicos, es decir, define la estructura de los procesos modelados pero sin asociar los componentes gráficos específicos de cada elemento.

Uno de los inconvenientes para generar los componentes gráficos es la disposición de estos en el plano, dado que hay que especificar coordenadas, altura y ancho para cada uno; es lo que llamamos el **layout** del modelo. Dada la complejidad de la generación del layout, luego de analizar distintas opciones decidimos utilizar Yaoqiang como herramienta externa para generar el modelo.

Yaoqiang es un editor gráfico BPMN 2.0 escrito en java. Entre sus variadas características hay dos que lo hacen fundamental para esta etapa del proceso: una de ellas es que es capaz de generar los elementos gráficos de un modelo BPMN 2.0 (agregando la etiqueta BPMNDI y los datos asociados) y su capacidad para ejecutarse modularmente. Otra característica interesante es que realiza el validado XSD de los modelos BPMN 2.0.

Utilizando el módulo *org.yaoqiang.asaf.bpmn-graph.jar*, que a partir de un archivo BPMN 2.0 con la etiqueta *process* bien definida, con sus componentes internos que definen la estructura del proceso, es capaz de generar un archivo con BPMN 2.0 con la misma estructura y su componente gráfico asociado. Este módulo contiene algunos bugs críticos con respecto a los lanes, y algunos menores en cuanto a compuertas con más de un flujo de entrada y tareas adjuntas. Es por ello que antes de utilizar el módulo le quitamos los componentes asociados al lane para generar el modelo BPMN 2.0 con el componente BPMNDI.

Finalmente, de este modelo obtenido por el módulo de yaoqiang utilizaremos únicamente el componente gráfico BPMNDiagram que agregaremos al modelo BPMN 2.0 que se estaba construyendo, es decir, antes de quitar el componente asociado a los lanes.

5.7 Desplegar resultados

Una vez obtenido el modelo BPMN 2.0 realizamos dos presentaciones gráficas; por un lado se representa el modelo mediante un modelador de BPMN 2.0 utilizando los componentes gráficos

estándar, por otro lado realizamos una representación gráfica del flujo de actividades, eventos y compuertas con una herramienta de visualización de grafos.

5.7.1 Modelador embebido

Una vez obtenido el modelo BPMN 2.0 con su componente gráfico lo integraremos con la herramienta bpmn.io [25], que es un herramienta de modelado BPMN 2.0 escrito en javascript capaz de ser embebido en cualquier documento html para generar e importar este tipo de modelos. Para importar un documento BPMN 2.0, este debe tener su componente gráfico bien definido.

El modelador fue configurado para ajustar los elementos ya existentes, imposibilitando agregar nuevos elementos, el objetivo del mismo es ajustar los componentes visuales generados automáticamente.

5.7.2 Visualización gráfica de nodos mediante DOT

Para realizar una primera aproximación gráfica del trabajo realizado con el fin de mejorar el depurado utilizamos inicialmente DOT [26], que es un lenguaje para la construcción gráfica de grafos incluido en el framework Graphviz [27] desarrollado en c++. Viz.js es una adaptación para javascript de la herramienta Graphviz.

El objetivo de este módulo es generar una simple visualización de los elementos básicos de BPMN 2.0 y el flujo que los une. Este módulo ignora construcciones como lanes y pools, el contenido de los subprocessos y datos asociados a las actividades.

Cada elemento renderizado tiene indicado su id y se colorea dependiendo del tipo de elemento al que se encuentre asociado (los colores fueron elegidos de manera arbitraria):

- eventos en tonos de rojo
- compuertas en tonos de azul
- acciones en tonos de verde
- flujo de eventos adjuntos con flechas punteadas

6 Prototipo funcional

En este capítulo se presenta el prototipo funcional con sus principales características. Comenzamos planteando la plataforma y las bibliotecas gráficas utilizadas, luego presentamos la interfaz con sus principales componentes y finalmente detallaremos las funcionalidades principales y cómo utilizarlas.

6.1 Plataforma y herramientas

El prototipo funcional es una aplicación escrita en javascript que corre en la plataforma nodejs [28]. Para la construcción de la interfaz gráfica, se utilizó el framework Electron [29], una herramienta capaz de construir aplicaciones de escritorio utilizando tecnologías web como html, css y javascript.

Finalmente para el desarrollo de la aplicación se utilizaron los siguientes frameworks html: para manipulación de dom utilizamos jquery [30], para mejorar aspectos visuales bootstrap [31], para desplegar datos XML y JSON con indentación anidada pretty-data [32] y para manejar ayuda interactiva bootstrap-tour[33].

6.2 Presentación de la interfaz gráfica

En la Figura 6.1 se puede ver la interfaz gráfica del prototipo funcional referenciando sus principales componentes. Se puede ver que es una interfaz simple, la cual tiene un listado de los ejemplos con los que hemos ido trabajando, dos campos de texto para ingresar un proceso nuevo y una serie de opciones las cuales presentan el resultado obtenido luego de procesar una descripción de un proceso. Estos componentes se ven numerados en la Figura 6.1 y a continuación pasaremos a explicar en detalle cada uno de estos.

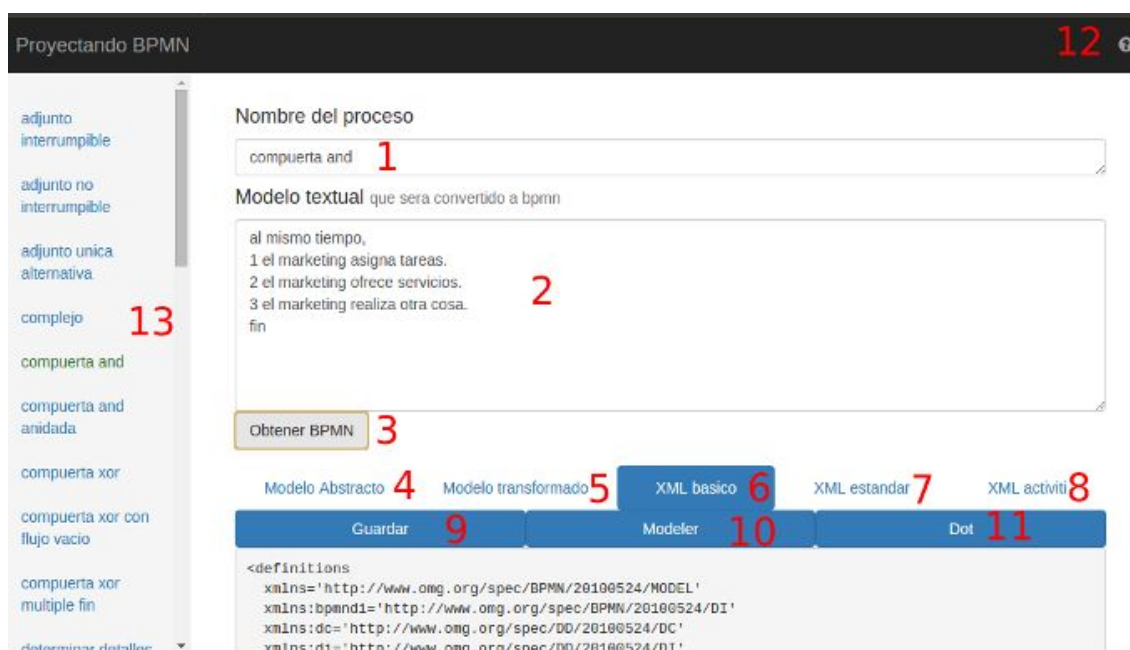


Figura 6.1 Interfaz gráfica del prototipo con referencias numéricas

6.2.1 Componentes del prototipo

6.2.1.1 Caja de Texto “Nombre del proceso”

Este campo es el número 1 en la Figura 6.1, la finalidad de este campo es especificar el nombre del proceso que se quiere generar. En la Figura 6.2 se ve más claro el componente donde se muestra un ejemplo de un nombre de proceso. En este campo solo se aceptan caracteres alfanuméricos.

6.2.1.2 Área de Texto “Modelo Textual”

Este campo es el número 2 en la Figura 6.1. Como indica, es donde se ingresa el modelo textual del proceso. Este modelo textual especifica la estructura del proceso. Al igual que el nombre del proceso acepta solo caracteres alfanuméricos y debe cumplir con las reglas del lenguaje especificadas en el capítulo 4. Agregamos una pequeña ayuda a la hora de definir subprocesos, de forma que haciendo click derecho sobre una selección permite agregar un subproceso. Esto se puede ver en la Figura 6.2

6.2.1.3 Botón “Obtener BPMN”

Este botón está etiquetado con el número 3 en la Figura 6.1. Una vez ingresado el nombre del proceso y el modelo textual, haciendo click en este botón es la forma de realizar el procesamiento del modelo textual para obtener los diferentes modelos gráficos. En caso de haber algún error en el modelo textual, este se indica con un mensaje de error e indicando en el modelo textual donde se identificó el error. Esto se puede ver en la Figura 6.3.

Nombre del proceso

ejemplo del prototipo

Modelo textual que sera convertido a bpmn

al mismo tiempo,
1 el marketing asigna tareas.
2 el marketing ofrece servicios.
3 el marketing realiza otra cosa.

Obtener BPMN

Figura 6.2

En caso de ingresar un modelo textual incorrecto o vacío, cuando se da click en el botón Obtener BPMN, se despliega un mensaje de error indicando cuál fue el error y se indica en el modelo donde pudo haber estado el error. En las Figuras 6.3 se pueden ver algunos de los errores más

comunes. En la Figura 6.3.a se da el caso de no ingresar el modelo textual. En la Figura 6.3.b es el caso donde no se indica el fin de la construcción, en particular, se está definiendo una compuerta AND, pero no se especificó el fin de la misma.

Modelo textual que sera convertido a bpmn

Obtener BPMN

Error de sintaxis. Se esperaba por: "a la vez", "al mismo tiempo", "alternativa de", "de", "del", "el", "en paralelo", "la", "las", "los", "mientras", "si se cumple", "un", "una", "unica" o un salto de línea pero se encontró fin del texto .

Figura 6.3.a. Error desplegado cuando el modelo textual es vacío.

Modelo textual que sera convertido a bpmn

al mismo tiempo,
1 el marketing asigna tareas.
2 el marketing ofrece servicios.
3 el marketing realiza otra cosa.

Obtener BPMN

Error de sintaxis. Se esperaba por: "a la vez", "al mismo tiempo", "alternativa de", "de", "del", "el", "en paralelo", "fin", "la", "las", "los", "mientras", "si se cumple", "un", "una", "unica", "y finaliza", un salto de línea, entero, palabra o palabras pero se encontró fin del texto .

Figura 6.3.b. Error desplegado cuando el modelo textual es incorrecto.

En la Figura 6.3.c se da el error cuando la palabra clave utilizada para definir una construcción no es utilizada correctamente, en este caso particular, se está queriendo definir una compuerta XOR, pero en lugar de utilizar el texto “Si se cumple,”, se utilizó “Si se,”.

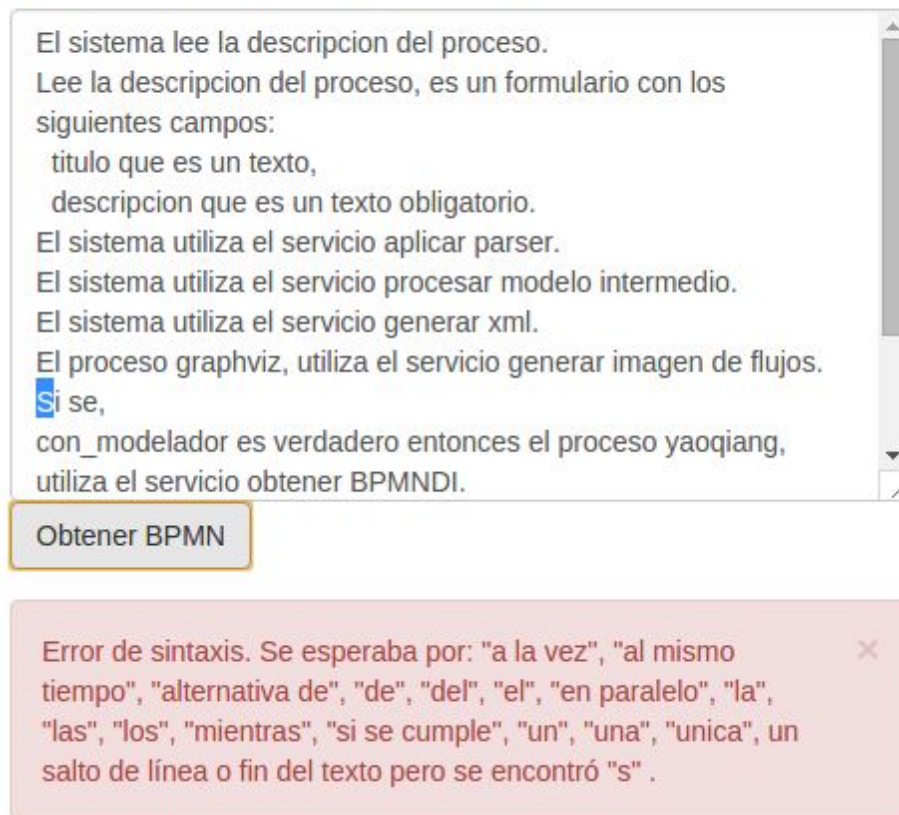


Figura 6.3.c. Error de sintaxis por error en palabra clave.

Otro tipo de error fácil de introducir es la omisión del punto final de una oración, el cual indica el fin de una actividad. Esto se puede ver en la Figura 6.3.d, cuando se quiere definir la tarea de usuario “el marketing ofrece servicios.” donde se omite el punto final.

Modelo textual que sera convertido a bpmn

al mismo tiempo,
1 el marketing asigna tareas.
2 el marketing ofrece servicios
3 el marketing realiza otra cosa.
fin

Obtener BPMN

Error de sintaxis. Se esperaba por: ",", ".", un salto de línea o [a-z_]i pero se encontró "3" .

Figura 6.3.d. Error de sintaxis por omisión del punto final.

Una vez ingresado correctamente el modelo textual del proceso el prototipo retorna su representación en los 3 formatos BPMN 2.0, en las pestañas “XML básico”, “XML estándar” y “XML Activiti”. Cada una de las pestañas tiene la posibilidad de desplegar un modelador para acomodar los componentes gráficos de BPMN 2.0 y desplegar una imagen con los componentes. A continuación seguimos describiendo estos componentes del prototipo.

6.2.1.4 Pestaña “Modelo Abstracto”

Luego de haber procesado el modelo textual, se despliegan varios resultados, los cuales se pueden ver en las pestañas que se encuentran debajo del botón “Obtener BPMN”. Entre estas se encuentra el modelo abstracto, con el número 4 en la Figura 6.1. Este modelo abstracto representa el primer modelo en formato JSON, obtenido a partir del procesamiento realizado por la gramática especificada en el capítulo 4 sobre el modelo textual. Luego este modelo es procesado en varias iteraciones hasta conseguir los modelos BPMN 2.0 finales. En la Figura 6.4 se puede ver parte del modelo abstracto obtenido para un ejemplo simple.

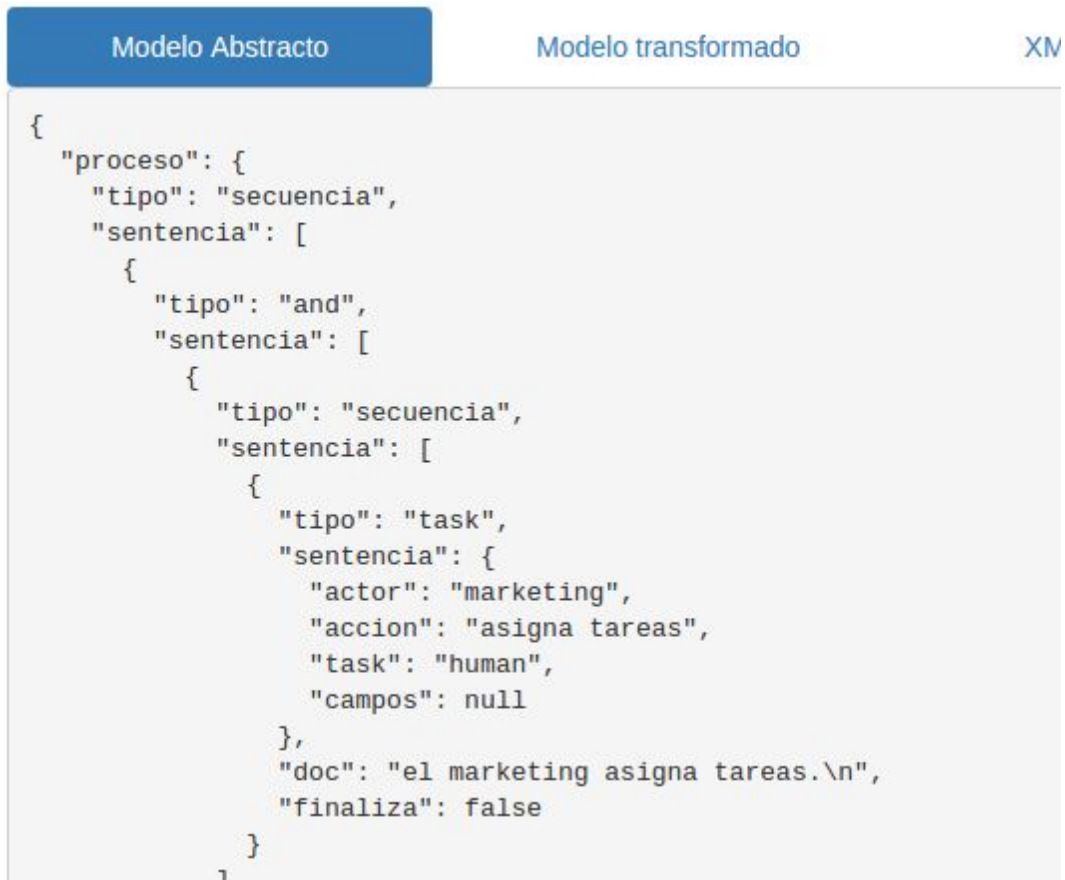


Figura 6.4. Ejemplo de modelo abstracto.

6.2.1.5 Pestaña “Modelo Transformado”

Continuando, etiquetado con el número 5, luego de haber realizado la primer etapa de procesamiento sobre el modelo abstracto se tiene el modelo intermedio el cual contiene algunas diferencias sustanciales con respecto al modelo abstracto, las cuales fueron explicadas en el capítulo 5. En la Figura 6.5 se puede ver el modelo intermedio correspondiente al ejemplo anterior.

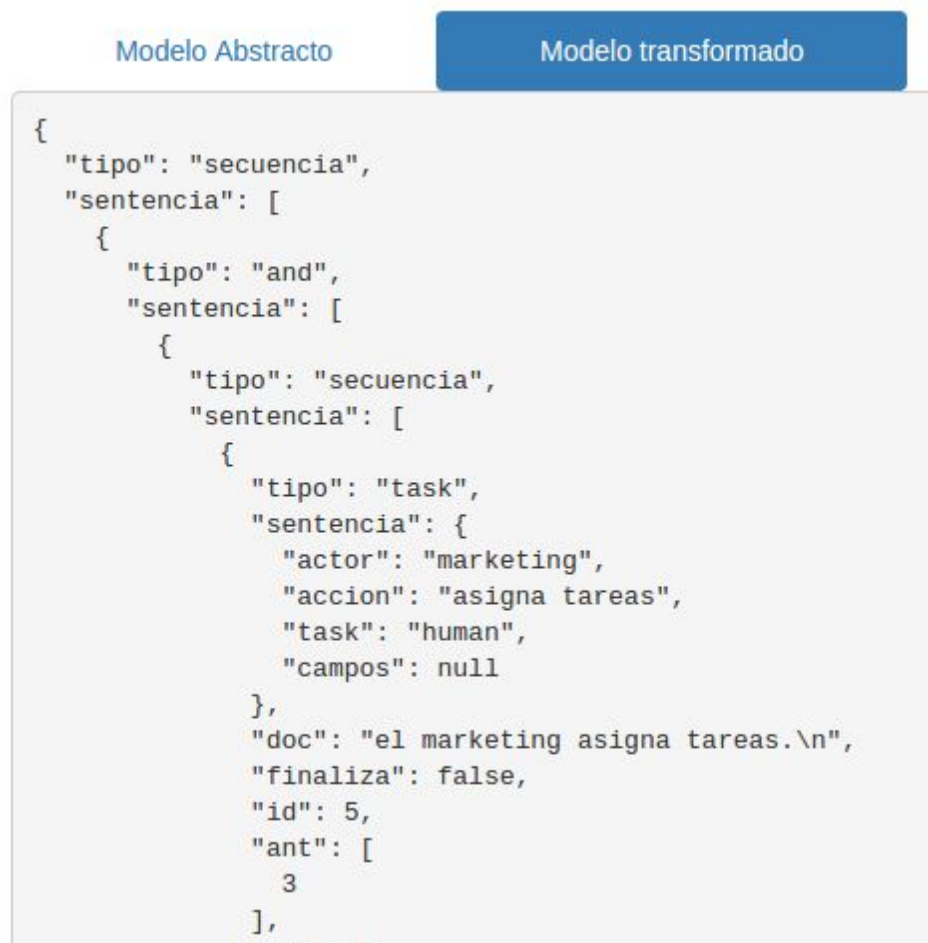


Figura 6.5. Ejemplo de modelo intermedio

6.2.1.6 Pestaña “XML básico”

Etiquetado con el número 6, se puede ver en la Figura 6.1 la pestaña donde se despliega el modelo BPMN 2.0 básico en formato XML. Este modelo es uno de los resultados finales luego del procesamiento del modelo textual. Este modelo solo contiene la representación de los componentes básicos, elementos específicos de la etapa de análisis.

6.2.1.7 Pestaña “XML estándar”

Continuando, con el número 7 en la Figura 6.1, se encuentra la pestaña del modelo BPMN 2.0 estándar en formato XML y en el mismo sentido que el modelo BPMN 2.0 básico, este solo contiene la representación de los componentes estrictamente en formato estándar, incluyendo también los componentes de ejecución.

6.2.1.8 Pestaña “XML Activiti”

Siguiendo, en la pestaña etiquetada con el número 8 en la Figura 6.1, se encuentra el modelo BPMN 2.0 asociado al BPMS Activiti, el cual contiene los mismos componentes que el modelo BPMN 2.0 estándar, pero utilizando las extensiones particulares de Activiti como se explicó previamente en el capítulo 5 .

6.2.1.9 Botón “Guardar modelo”

Como se puede ver en la Figura 6.6, para cada modelo obtenido se permite guardar su modelo BPMN 2.0 haciendo click en el botón Guardar etiquetado con el número 9 en la Figura 6.1.



Figura 6.6. Guardar modelo BPMN 2.0 seleccionado

Con este botón se abre un diálogo que permite guardar el modelo BPMN 2.0 seleccionado en el sistema de archivos como se ve en la Figura 6.7.

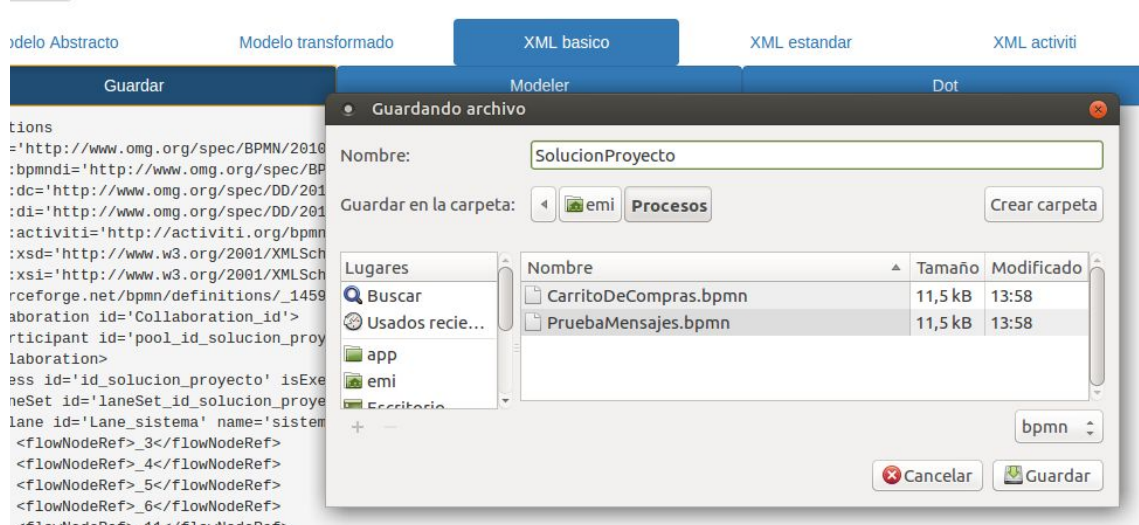


Figura 6.7. Diálogo para guardar el modelo BPMN 2.0 seleccionado

6.2.1.10 Botón “Modelador”

Al igual que con el botón de guardar, también se ofrece la opción de visualizar el modelo BPMN 2.0 seleccionado en la herramienta de modelado bpmn.io como se ve en la Figura 6.8. Esto se hace haciendo click en el botón Modeler, etiquetado con el número 10 en la Figura 6.1. Esta herramienta permite mover los componentes y guardar el mismo como imagen png.

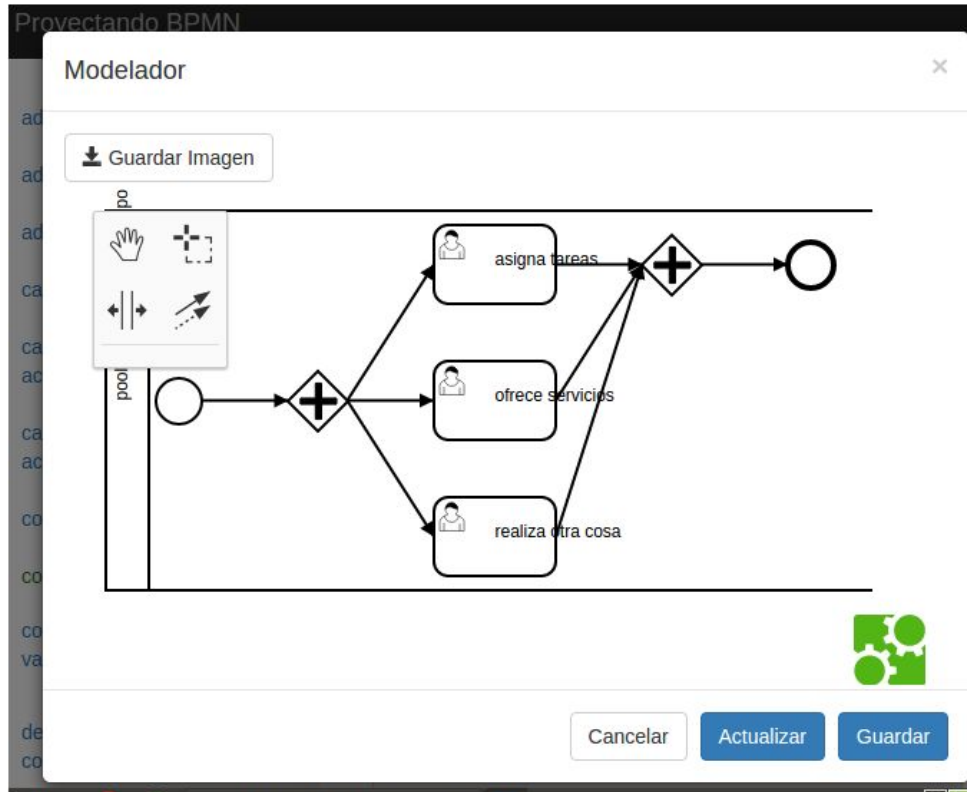


Figura 6.8. Visualización del modelo en la herramienta de modelado integrada

Como se explicó en el capítulo 5, el layout se realiza de manera automática y contiene algunos pequeños errores, como por ejemplo, las etiquetas de las tareas no están bien alineadas con las mismas al igual que el nombre del pool. De todas formas, se puede corregir de manera simple con el modelador. En la Figura 6.9 se ve el modelo anterior luego de haber sido ajustado.

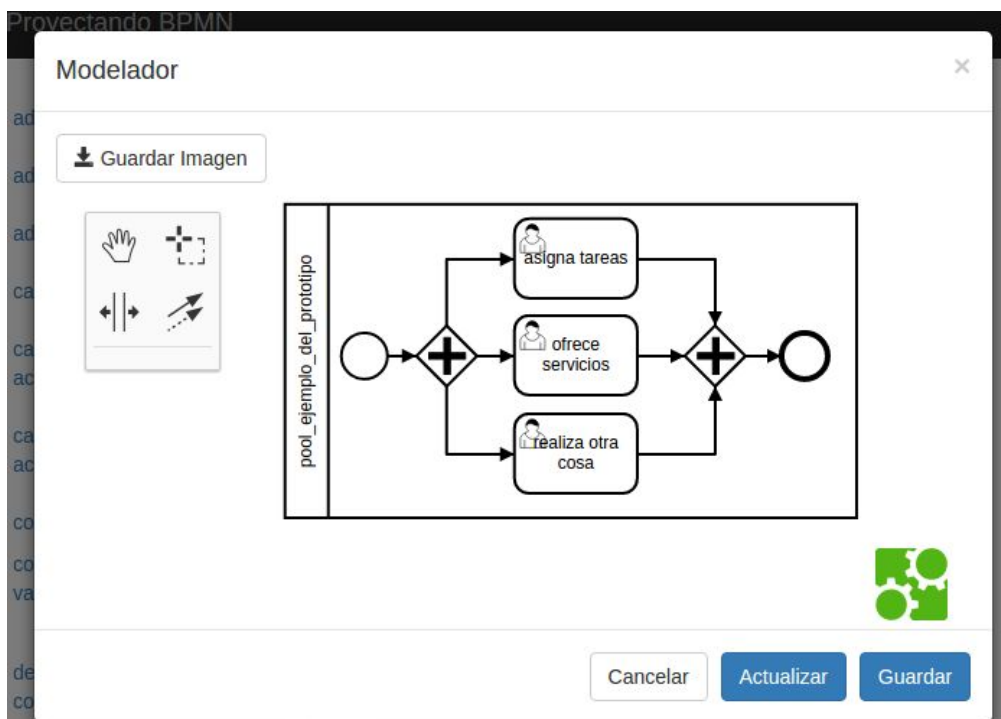


Figura 6.9. Visualización del modelo luego de ser ajustado

6.2.1.11 Botón “Dot”

Finalizando, en lo que respecta a los componentes que se aplican a un modelo seleccionado, se agregó una visualización gráfica que no es del estándar BPMN 2.0, pero que nos ayudó durante el desarrollo de la solución. Esta fue muy útil a la hora de corregir errores en el flujo principalmente.

Esta herramienta es la visualización del modelo utilizando DOT, etiquetada con el número 11 en la Figura 6.1, haciendo click en el botón DOT se despliega un visualizador del modelo en formato DOT como se ve en la Figura 6.10.a . Si se realiza click sobre la imagen, esta se expande, permitiendo tener una visibilidad más legible del modelo como se ve en la Figura 6.10.b .

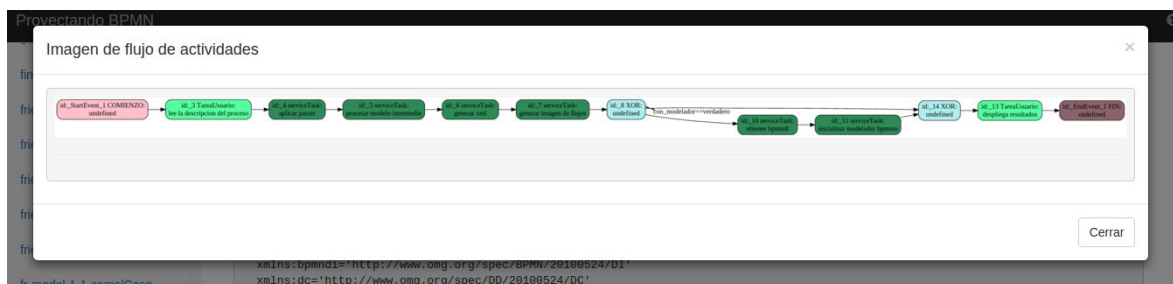


Figura 6.10.a. Visualización del modelo en formato DOT

Imagen de flujo de actividades

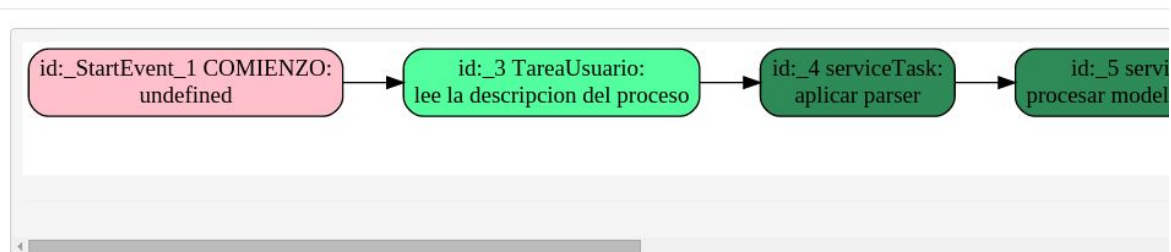


Figura 6.10.b. Imagen expandida del modelo en formato DOT

6.2.1.12 Botón de Ayuda Interactiva

Etiquetado con el número 12 en la Figura 6.1, en la esquina superior derecha se encuentra disponible una ayuda interactiva mediante tooltips donde se realiza una demostración de la aplicación. En la Figura 6.11 se puede ver una captura del primer mensaje que se despliega en el tour interactivo por la aplicación. Con los botones Next y Prev se navega por la ayuda interactiva y haciendo click en el botón End Tour se finaliza la misma.



Figura 6.11. Ayuda interactiva sobre la aplicación

6.2.1.13 Barra de ejemplos

Para finalizar, en el costado izquierdo de la aplicación, etiquetado con el número 13 en la Figura 6.1, se encuentra un listado de los procesos que fueron correctamente procesados, donde se puede seleccionar cualquiera de ellos para volver a procesar o utilizar como ejemplo para construir otros. Si se ingresa un proceso nuevo, el modelo textual del mismo será guardado en el sistema de archivo con el nombre ingresado, y la siguiente vez que se ejecute la aplicación este ejemplo será cargado en la lista. El objetivo de esto es, principalmente, tener a mano una variedad de ejemplos simples.

7 Caso de estudio

Realizamos un caso de estudio de la aplicación en base al proyecto de “Mejora de gestión por procesos Universidad de la República” realizado por el grupo COAL entre los años 2013-15. El proceso elegido es uno de los modelos de procesos realizados de la Dirección General de Relaciones y Cooperación (DGRC) Udelar, que corresponde al proceso de adjudicación de movilidades a estudiantes de grado de la Udelar a distintos programas de intercambio académico.

Esta documentación fue provista por los tutores y contiene una descripción textual en lenguaje natural del proceso y el diagrama asociado a la descripción. Dado que la aplicación desarrollada genera el modelo gráfico BPMN 2.0 a partir de una descripción textual surge natural hacer la comparación tanto de la descripción textual del proceso como también de los modelos gráficos generados.

Por esto, a partir del texto que describe el proceso, realizamos una traducción al lenguaje estructurado que recibe como entrada la aplicación y generamos el modelo a partir de este texto. Luego comparamos el modelo obtenido originalmente contra el modelo obtenido con el prototipo (en conjunto con el XML, ya que, por ejemplo, en el dibujo no se muestran los lanes pero en el XML si están). Por otro lado, generamos una descripción a partir del modelo que ya está hecho, y lo comparamos con el original.

7.1 Adjudicación de movilidades de estudiantes de grado

En esta sección presentamos la descripción textual del proceso en la figura 7.1, junto con el modelo BPMN 2.0 original en las Figuras 7.2.a y 7.2.b los cuales serán además entregados como anexos.

El proceso comienza con la recepción de las postulaciones (tarea Recibir Postulaciones), las cuales se pueden recibir de los postulantes o de los servicios universitarios. Además, estas postulaciones pueden realizarse de forma electrónica (por mail, sitio web, etc.), a través de un expediente, personalmente en la DGRC, etc. Todas estas variantes dependen de cada programa en particular.

Todo programa plantea una fecha límite de recepción de postulaciones, tras lo cual se pasa a controlar la documentación entregada (tarea Controlar Documentación). Esta tarea implica la consolidación de la información suministrada por los postulantes o servicios, y el control de la información proporcionada, con la eventual solicitud de documentación adicional en caso de que lo entregado no satisfaga los requerimientos. Como resultado de esta tarea se obtiene una lista consolidada de postulantes habilitados, la cual puede estar ya con una evaluación previa realizada por un servicio y una ordenación de los postulantes, dependiendo del programa.

A continuación se puede analizar la disponibilidad (tarea Analizar Disponibilidad), tanto presupuestal como de cupos existentes (si existiese un número predefinido), del llamado. Esta tarea se realiza o no dependiendo del programa particular y en caso de realizarse su resultado puede afectar tareas posteriores. Por ejemplo, en caso de no existir suficiente disponibilidad, se podrá requerir una priorización de los postulantes o la negociación de cupos adicionales.

Tanto habiendo analizado disponibilidad como si no, se procede a realizar la evaluación de los postulantes (tarea Realizar Evaluación). Esta tarea se encuentra condicionada a la existencia o no de una evaluación previa, la cual en algunos casos ya lo realiza cada servicio cuando envía las postulaciones. Esta evaluación determina que postulantes cumplen con las condiciones mínimas para realizar una movilidad y cuáles no, así como puede establecer un orden inicial entre los postulantes. La misma es realizada por el rol Evaluador, el cual puede ser asumido tanto por participantes internos a la Udelar, como la CRICRI, como por externos y/o las universidades de destino.

Una vez se cuenta con una lista de postulantes evaluada, se puede asignar una prioridad a los postulantes (tarea Realizar Priorización), más allá de que la lista tenga un orden inicial. La priorización en este caso es un mecanismo de depuramiento de una evaluación inicial, pudiendo incluso eliminar postulantes. Esto se debe a que se pueden adoptar criterios de evaluación académica, por ejemplo de escolaridad mínima necesaria de avance en la carrera para aplicar a determinadas universidades. La priorización puede verse afectada por la disponibilidad analizada anteriormente. La misma es realizada por el rol Decisor, el cual al igual que el Evaluador, puede ser asumido tanto por participantes internos como externos a la Udelar.

Una vez que se constituye la lista preliminar de movilidades a adjudicar, es posible una ronda de negociación de cupos (tarea Negociar Cupos) a los efectos de satisfacer la demanda. Esta tarea se puede realizar en conjunto con la Contraparte Extranjera y el Organismo Coordinador y se puede ver afectada por el análisis de disponibilidad realizado inicialmente.

Luego, se pasa a adjudicar las movilidades definitivas (subproceso Adjudicar Movilidad en Figura 2) lo que implica la comunicación con cada postulante aceptado para recibir su aceptación o rechazo. Es posible realizar este subproceso de forma paralela para cada servicio en particular, cada cupo determinado, etc. En algunos casos, cuando la adjudicación de una movilidad está determinada por un orden de prelación, existe la posibilidad de contactar a más de un postulante en caso de rechazo de la propuesta por parte del postulante priorizado. Incluso, en caso de que no se pueda realizar la adjudicación (por ejemplo porque el estudiante no acepta la propuesta de movilidad pero existen otras opciones), es posible regresar a negociar cupos para plantear nuevas alternativas.

La adjudicación de movilidades finaliza con la lista definitiva de movilidades adjudicadas, tras lo cual se pasa a comunicar la adjudicación (tarea Comunicar Adjudicación de Movilidades) a los servicios, postulantes y las contrapartes extranjeras. Finalmente, en caso de que se requiera algún tipo de documentación adicional, se solicita y controla la misma (tarea Procesar Documentación Adicional). Esto no implica la gestión de documentos como pasaporte, Visa, etc., sino documentación de carácter académico, como los contratos de estudio (firmados por los coordinadores académicos de ambas partes, el Decano del servicio de Udelar y el coordinador institucional) y la firma de los compromisos de reconocimiento de actividades entre el postulante, la Udelar y la contraparte extranjera.

Figura 7.1. Descripción textual del proceso

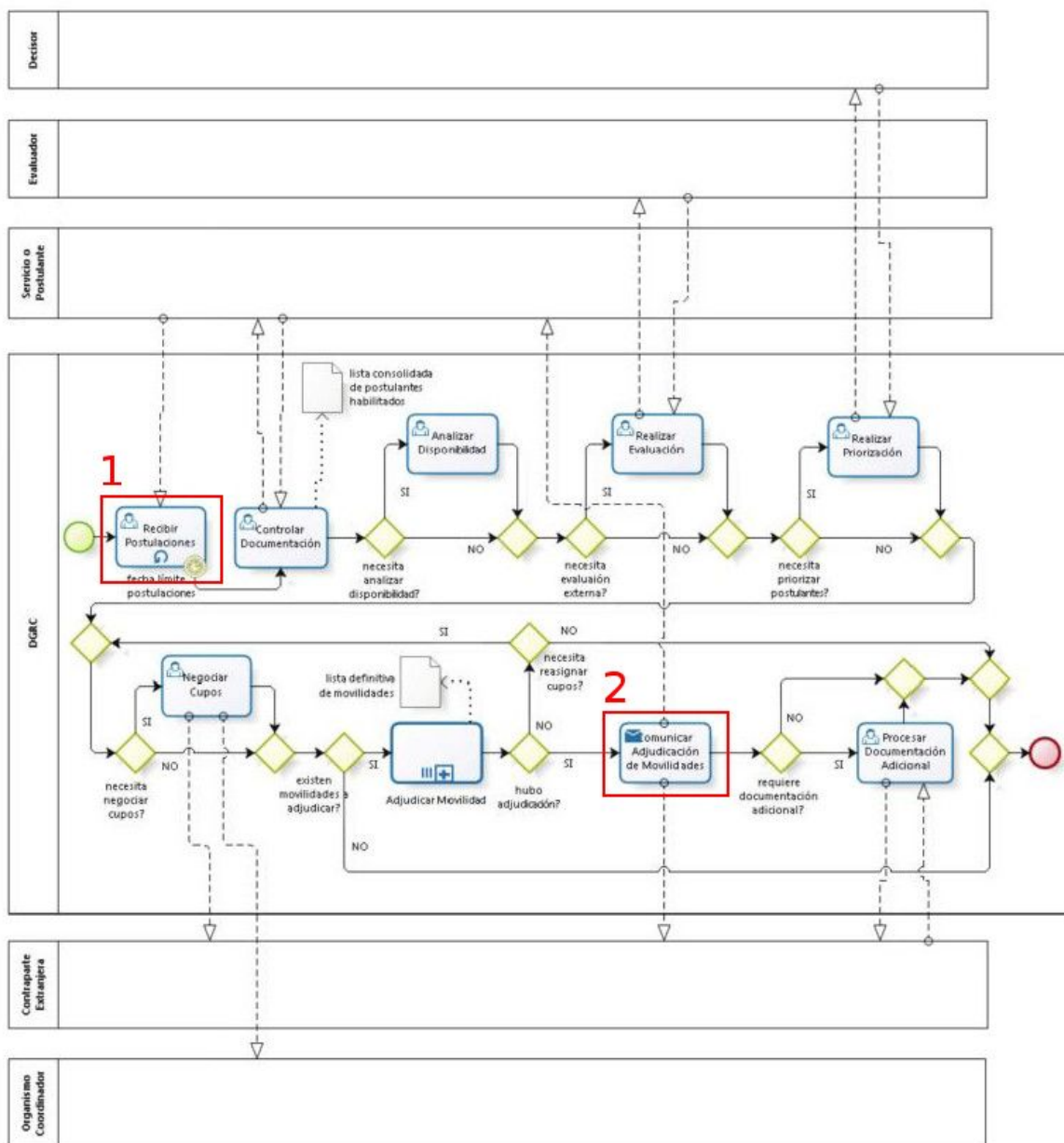


Figura 7.2.a. Modelo BPMN 2.0 del proceso

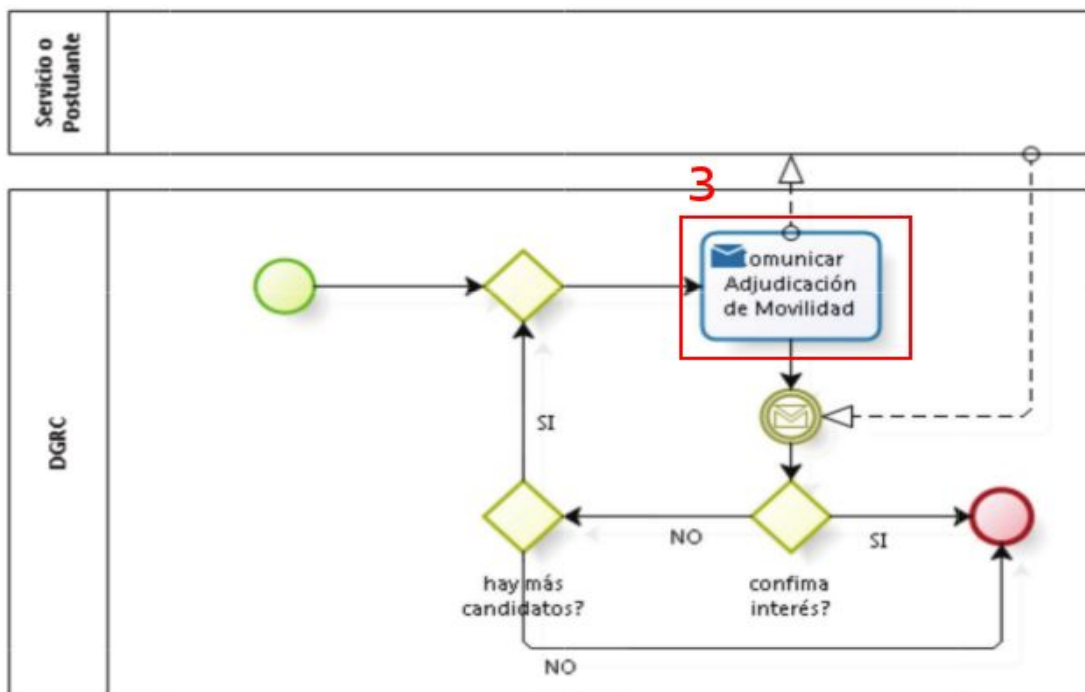


Figura 7.2.b Modelo BPMN 2.0 del subprocesso Adjudicar Movilidad

7.2 Construcciones no soportadas

En el modelo original se pueden ver algunos componentes que no son soportados por el lenguaje, los cuales están marcados en las figuras 7.2.a y 7.2.b con recuadros rojos y un número, por lo que es necesario realizar algunos cambios en el modelo original antes de continuar con el análisis del mismo.

En primera instancia se puede ver en el recuadro rojo número 1 de la Figura 7.2.a que la primera tarea de usuario, “Recibir postulaciones”, es realizada varias veces. Como no es posible a partir del lenguaje generar tareas de usuario que se realicen varias veces, vemos la necesidad de transformarla en un subprocesso el cual solo tenga la tarea de usuario en cuestión, y de forma que el subprocesso sea realizado varias veces. En la Figura 7.3.a se ve el modelo original de la tarea de usuario en cuestión y en la Figura 7.3.b el modelo generado para sustituirla.



Figura 7.3.a. Modelo original

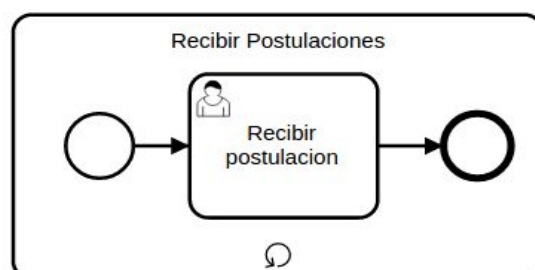


Figura 7.3.b. Modelo generado

Luego, en el recuadro rojo número 2 de la Figura 7.2.a, nos encontramos con la tarea de envío de mensaje “Comunicar adjudicación de movilidades” la cual se comunica con dos pools diferentes. El lenguaje no da soporte para este tipo de tareas por lo que decidimos cambiarla en un principio por un evento de envío de mensajes. Cómo se puede ver la tarea de envío de mensaje se comunica con dos pools diferentes, por un lado el pool “contraparte extranjera” y por otro lado el pool “servicio o postulante”, ya que con un solo evento de envío de mensaje no podríamos comunicarnos con dos pools diferentes a la misma vez manejamos dos opciones para resolver esto, una primera es utilizar dos eventos de envío de mensajes que se ejecuten de forma secuencial y la segunda opción es utilizar una compuerta AND de forma que se puedan ejecutar en paralelo. En la descripción del modelo no especifica que la comunicación tenga que ser en paralelo o secuencial, pero del modelo entendemos que debería ser en paralelo, por esto es que optamos por la segunda opción. Entonces en la Figura 7.4.a vemos el modelo original del componente y en la Figura 7.4.b el modelo generado para sustituirlo.

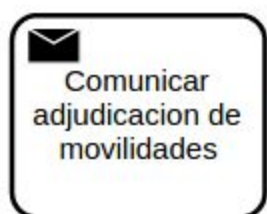


Figura 7.4.a. Modelo original

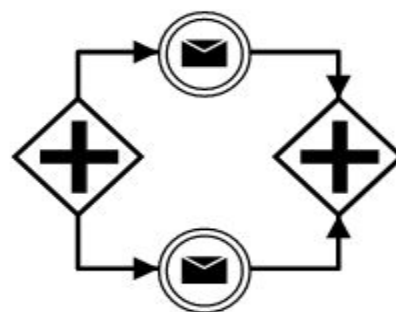


Figura 7.4.b. Modelo generado

Para finalizar, nos queda analizar el modelo del subproceso “Adjudicar Movilidad”, en el cual sólo encontramos una tarea de envío de mensaje, la cual está enmarcada en rojo en la Figura 7.2.b con número 3. De la misma forma que se mencionó para la tarea de envío de mensaje del proceso principal, esta tarea de envío de mensaje será sustituida por un evento de envío de mensaje y solo por uno ya que la comunicación es solo con el pool “servicio o postulante”.

Por otro lado, el bindeo de datos como puede ser la lista consolidada de postulantes vinculada a la tarea de usuario “controlar documentación”. Este tipo de bindeo de datos no es soportado por el lenguaje. De la misma forma que los flujos de mensaje que comunican tareas de usuario con otros pools. Este tipo de construcciones no serán sustituidas por otras, sino que no serán consideradas a la hora de generar el modelo textual, ya que en el primer caso, ese tipo de bindeo de datos creemos que se refiere más a una referencia a un documento adjunto que tiene el proceso (por ejemplo en formato pdf) y en el segundo caso porque entendemos que no es la forma más común de representar este tipo de comunicaciones.

7.3 Generación a partir de la descripción textual

En este enfoque, buscamos un acercamiento similar al uso normal, en el sentido que inicialmente no vamos a tener un modelo para interpretar, sino que vamos a tener una descripción textual del proceso y en base a este se tendrá que generar el modelo textual que luego será utilizado para

generar el modelo BPMN 2.0. A continuación se detalla la interpretación de la descripción textual del proceso, junto con los elementos del modelo textual que se van construyendo.

El proceso comienza con la **recepción de las postulaciones**, las cuales se pueden recibir de los postulantes o de los servicios universitarios. Además, estas postulaciones pueden realizarse de forma electrónica (por mail, sitio web, etc.), a través de un expediente, personalmente en la DGRC, etc. Todas estas variantes dependen de cada programa en particular. Todo programa plantea una **fecha límite** de recepción de postulaciones,

Figura 7.5

En la figura 7.5, destacamos la tarea de usuario recepción de postulación, pero como es una tarea que se realiza varias veces, se construirá un subproceso que incluya esta tarea, y este subproceso será el que se realizará varias veces. A su vez, se puede ver que este subproceso, tiene un evento adjunto, indicando que ya no se recibirán más postulaciones, en caso que se llegue a la fecha límite, el subproceso debe terminar su ejecución y solamente en caso que se de este evento adjunto se podrá continuar con la ejecución del proceso, entonces, será un evento adjunto interrumpible con única alternativa. El modelo textual es el que se puede apreciar en la figura 7.6.

la DGRC realiza el subproceso DGRC_recibir_postulaciones; varias veces.
única alternativa de DGRC_recibir_postulaciones, se interrumpe si
transcurre 1 mes

Figura 7.6. Modelo textual de la figura 7.5.

tras lo cual se pasa a controlar la documentación entregada. Esta tarea implica la **consolidación de la información** suministrada por los postulantes o servicios, y el **control de la información** proporcionada, con la eventual **solicitud de documentación adicional** en caso de que **lo entregado no satisfaga los requerimientos**. Como resultado de esta tarea se obtiene una lista consolidada de postulantes habilitados, la cual puede estar ya con una evaluación previa realizada por un servicio y una ordenación de los postulantes, dependiendo del programa.

Figura 7.7

En la figura 7.7, encontramos las tareas de usuario, consolida la información y controla la información. Seguido de esto, de forma condicionada, se debe realizar la tarea de usuario solicitar información adicional, por lo que necesitamos utilizar un compuerta exclusiva, donde en un caso se ejecuta la tarea de usuario y en otro no se hace nada. El modelo textual correspondiente se ve en la figura 7.8.

```
la DGRC consolida la informacion.  
la DGRC controla la informacion.  
si se cumple,  
noCumpleRequerimientos entonces la DGRC solicita informacion adicional.  
si no no hace nada  
fin
```

Figura 7.8. Modelo textual de la figura 7.7.

A continuación se puede **analizar la disponibilidad** (tarea Analizar Disponibilidad), tanto presupuestal como de cupos existentes (si existiese un número predefinido), del llamado. Esta tarea **se realiza o no dependiendo del programa particular** y en caso de realizarse su resultado puede afectar tareas posteriores. Por ejemplo, en caso de no existir suficiente disponibilidad, se podrá requerir una priorización de los postulantes o la negociación de cupos adicionales.

Figura 7.9

En la figura 7.9, también de forma condicionada, se encuentra la tarea de usuario analizar la disponibilidad. También utilizaremos una compuerta exclusiva, donde en un caso se ejecuta la tarea de usuario y en el otro no se hace nada. Entonces, partir de esto obtenemos el modelo textual de la figura 7.10.

```
si se cumple,  
analizarDisp entonces La DGRC analiza la disponibilidad.  
si no no hace nada.  
fin.
```

Figura 7.10. Modelo textual de la figura 7.9.

Tanto habiendo analizado disponibilidad como si no, se procede a **realizar la evaluación de los postulantes** (tarea Realizar Evaluación). Esta tarea se encuentra **condicionada a la existencia o no de una evaluación previa**, la cual en algunos casos ya lo realiza cada servicio cuando envía las postulaciones. Esta evaluación determina que postulantes cumplen con las condiciones mínimas para realizar una movilidad y cuáles no, así como puede establecer un orden inicial entre los postulantes. La misma es realizada por el rol Evaluador, el cual puede ser asumido tanto por participantes internos a la Udelar, como la CRICRI, como por externos y/o las universidades de destino.

Figura 7.11

Continuando con la figura 7.11, nuevamente tenemos una tarea de usuario que se ejecuta de forma condicionada, esta es la tarea de usuario evaluar los participantes. Por lo que también

utilizaremos una compuerta exclusiva, obteniendo el modelo textual de la figura 7.12.

```
si se cumple,  
evaluacionPrevia entonces no hace nada  
si no la DGRC evalua los participantes.  
fin
```

Figura 7.12. Modelo textual de la figura 7.11.

Una vez se cuenta con una lista de postulantes evaluada, se puede **asignar una prioridad** a los postulantes (tarea Realizar Priorización), más allá de que la lista tenga un orden inicial. La priorización en este caso es un mecanismo de depuramiento de una evaluación inicial, pudiendo incluso eliminar postulantes. Esto se debe a que se pueden adoptar criterios de evaluación académica, por ejemplo de escolaridad mínima necesaria de avance en la carrera para aplicar a determinadas universidades. La priorización **puede verse afectada por la disponibilidad analizada anteriormente**. La misma es realizada por el rol Decisor, el cual al igual que el Evaluador, puede ser asumido tanto por participantes internos como externos a la Udelar.

Figura 7.13

En la figura 7.13, identificamos nuevamente una tarea de usuario que se ejecuta condicionalmente, ya que esta puede verse afectada por el resultado de la ejecución de una tarea previa. Esta tarea es la realización de priorizaciones, y obtenemos el modelo textual de la figura 7.14.

```
si se cumple,  
realizarPriorizacion entonces La DGRC realiza priorizacion.  
si no no hace nada  
fin
```

Figura 7.14. Modelo textual de la figura 7.13.

Una vez que se constituye la lista preliminar de movilidades a adjudicar, es posible una ronda de **negociación de cupos** (tarea Negociar Cupos) a los efectos de satisfacer la demanda. Esta tarea se puede realizar en conjunto con la Contraparte Extranjera y el Organismo Coordinador y se puede ver afectada por el análisis de disponibilidad realizado inicialmente. Luego, se pasa a **adjudicar las movilidades** definitivas (subproceso Adjudicar Movilidad en Figura 2) lo que implica la comunicación con cada postulante aceptado para recibir su aceptación o rechazo. Es posible realizar este subproceso de forma paralela para cada servicio en particular, cada cupo determinado, etc. En algunos casos, cuando la adjudicación de una movilidad está determinada por un orden de prelación, existe la posibilidad de contactar a más de un postulante en caso de rechazo de la propuesta por parte del postulante priorizado. Incluso, en caso de que no se pueda realizar la adjudicación (por ejemplo porque el estudiante

no acepta la propuesta de movilidad pero existen otras opciones), **es posible regresar a negociar cupos** para plantear nuevas alternativas.

Figura 7.15

En la figura 7.15, encontramos la tarea de usuario negociación de cupos, seguida de el subproceso adjudicar movilidades. Y como se destaca, luego de la ejecución del subproceso es posible volver a tener que negociar cupos, por lo que la tarea y el subproceso van a estar incluidas en un loop. De esta forma conseguimos el modelo textual de la figura 7.16.

```
mientras negociarMasCupos,  
la DGRC realiza una negociacion de cupos.  
la DGRC realiza el subproceso adjudicar movilidad.  
fin
```

Figura 7.16. Modelo textual de la figura 7.15.

Luego detallaremos la generación del modelo textual para el subproceso.

La adjudicación de movilidades finaliza con la lista definitiva de movilidades adjudicadas, tras lo cual se pasa a **comunicar la adjudicación** (tarea Comunicar Adjudicación de Movilidades) a los servicios, postulantes y las contrapartes extranjeras. Finalmente, **en caso de que se requiera** algún tipo de **documentación adicional**, se solicita y controla la misma (tarea Procesar Documentación Adicional). Esto no implica la gestión de documentos como pasaporte, Visa, etc., sino documentación de carácter académico, como los contratos de estudio (firmados por los coordinadores académicos de ambas partes, el Decano del servicio de Udelar y el coordinador institucional) y la firma de los compromisos de reconocimiento de actividades entre el postulante, la Udelar y la contraparte extranjera.

Figura 7.17

En la figura 7.17, se identifica una comunicación con lo que serían participantes externos en el proceso, que como el lenguaje no soporta tarea de mensajes, esta comunicación será representada utilizando eventos intermedios de envío de mensajes. Como son dos los participantes externos a los que se les comunican las adjudicaciones, utilizamos una compuerta paralela de forma de ejecutar en paralelo esta comunicación. Finalizando, nuevamente otra tarea de usuario que se ejecuta condicionalmente, esta tarea se ejecuta en caso de ser necesaria documentación adicional y se encarga de procesar la misma. Entonces el modelo textual obtenido es el de la figura 7.18.

```

al mismo tiempo,
1 la DGRC envia mensaje a contraparte extranjera.
2 la DGRC envia mensaje a servicio o postulante.
fin
si se cumple,
docAdicional entonces la DGRC procesa documentacion adicional.
si no no hace nada
fin

```

Figura 7.18. Modelo textual de la figura 7.17.

En conclusión, para la descripción textual de la figura 7.1 construimos el modelo textual de la figura 7.19.

```

la DGRC realiza el subproceso DGRC_recibir_postulaciones; varias veces.
unica alternativa de DGRC_recibir_postulaciones, se interrumpe si
transcurre 1 mes
la DGRC consolida la informacion.
la DGRC controla la informacion.
si se cumple,
noCumpleRequerimientos entonces la DGRC solicita informacion
adicional.
si no no hace nada fin
si se cumple,
analizarDisp entonces la DGRC analiza la disponibilidad.
si no no hace nada fin
si se cumple,
evaluacionPrevia entonces no hace nada
Si no la DGRC evalua los participantes. fin
si se cumple,
realizarPriorizacion entonces La DGRC realiza priorizacion.
si no no hace nada fin
mientras negociarMasCupos,
la DGRC realiza una negociacion de cupos.
la DGRC realiza el subproceso DGRC_adjudicar_movilidad. fin
al mismo tiempo,
1 la DGRC envia mensaje a contraparte extranjera.
2 la DGRC envia mensaje a servicio o postulante.
fin
si se cumple,
docAdicional entonces la DGRC procesa documentacion adicional.
si no no hace nada fin
fin

```

Figura 7.19. Modelo textual de la figura 7.1.

Solo nos queda por analizar la descripción del subproceso de forma de poder generar el modelo textual del mismo, para esto consideremos la figura 7.20 a continuación.

Luego, se pasa a adjudicar las movilidades definitivas (subproceso Adjudicar Movilidad en Figura 2) lo que implica la **comunicación con cada postulante** aceptado para **recibir su aceptación o rechazo**. Es posible realizar este subproceso de forma paralela para cada servicio en particular, cada cupo determinado, etc. En algunos casos, cuando la adjudicación de una movilidad está determinada por un orden de prelación, existe la posibilidad de **contactar a más de un postulante en caso de rechazo** de la propuesta por parte del postulante priorizado.

Figura 7.20. Descripción textual del subproceso.

Como se puede ver, lo primero a realizar es la comunicación de la adjudicación al postulante, lo que sigue es esperar por la respuesta del mismo para saber si acepta o rechaza la propuesta y en caso de rechazarla y en caso de haber más postulantes, se puede contactar con estos. Entonces el modelo textual definido para este subproceso es el de la figura 7.21.

```
mientras hayMasCandidatos,  
  el DGRC envia mensaje a Servicio o Postulante.  
  el DGRC espera por mensaje de Servicio o Postulante.  
  si se cumple,  
    confirmaInteres entonces el DGRC procesa confirmacion. y finaliza  
  si no no hace nada  
fin  
fin
```

Figura 7.21. Modelo textual del subproceso.

Se puede ver que se agrega una tarea de usuario “procesa confirmación”, que no se encuentra en la descripción del proceso original. Esto se debe a que el prototipo no permite que haya más de un flujo vacío en una compuerta exclusiva.

7.3.1 Modelos BPMN 2.0 obtenidos

Antes de continuar con el modelo BPMN 2.0 obtenido a partir del modelo textual es importante destacar que esto fue una primera interpretación a partir de la descripción textual del proceso. No hubo ninguna instancia de corrección del mismo con un cliente. Es posible que el modelo BPMN 2.0 obtenido no realice un cubrimiento exhaustivo del proceso original.

Luego de realizado el análisis del texto, queda ingresar el modelo textual obtenido en la aplicación y ver los modelos generados a partir de estos. Entonces, tanto el modelo textual del subproceso como el del proceso, son ingresados en el prototipo y a partir de este obtenemos los modelos que se pueden ver en las Figuras 7.22 y 7.23 para el subproceso y el proceso respectivamente.

Si bien el autolayout no funciona de la mejor manera, la herramienta permite la edición del

modelo de forma de acomodar los componentes para que se puedan ver mejor. Los modelos presentados en las Figuras 7.22 y 7.23 ya fueron ajustados para que se vean mejor los componentes.

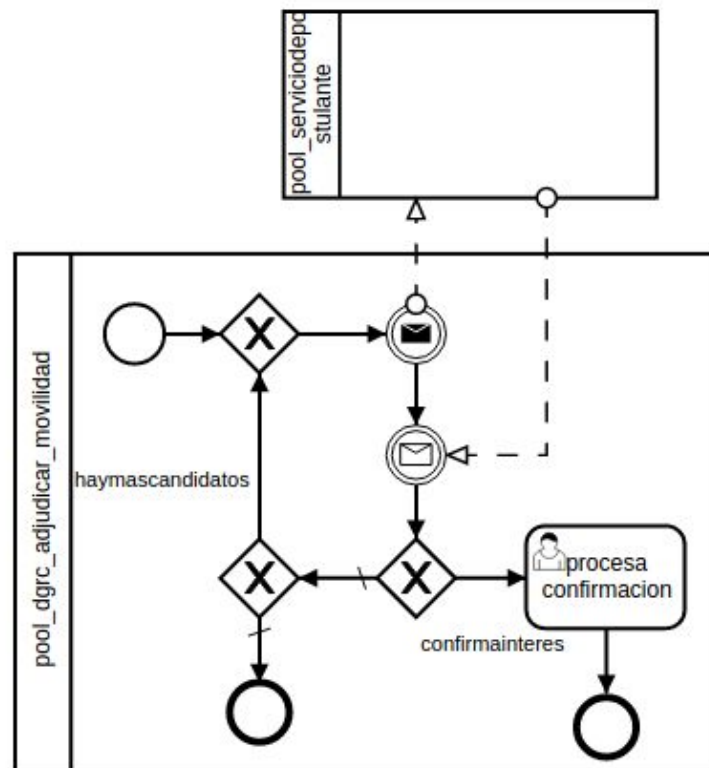


Figura 7.22. Modelo BPMN 2.0 del subprocesso Adjudicar Movilidades

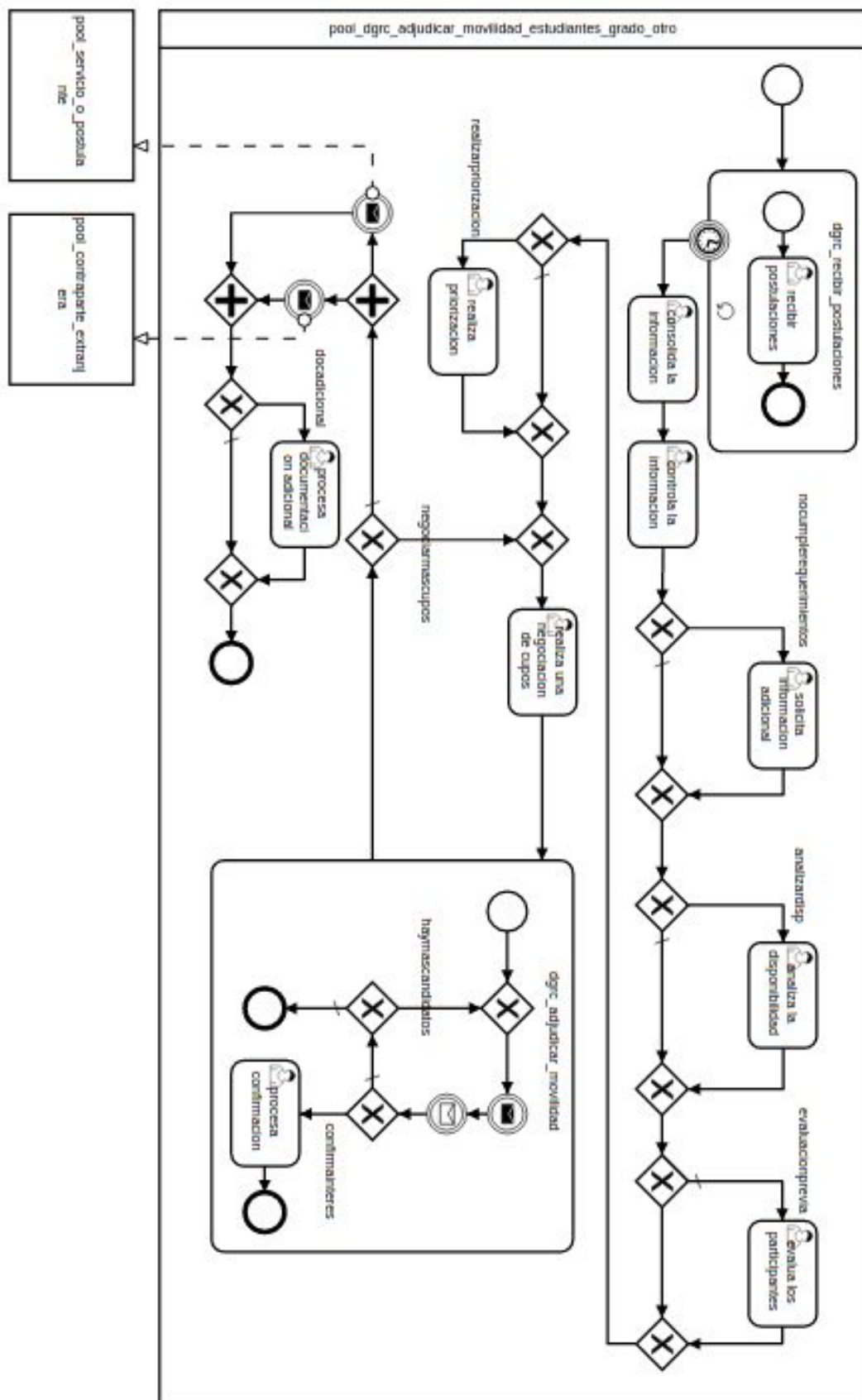


Figura 7.23. Modelo BPMN 2.0 obtenido para el proceso Adjudicacion de movilidades de estudiantes de grado

Siguiendo, compararemos el modelo BPMN 2.0 obtenido con el modelo BPMN 2.0 original detallando sus diferencias. Entonces, salvo por los componentes que fue necesario cambiar ya que no eran soportados, las diferencias no son sustanciales. Al principio, en el proceso original se realiza la tarea de usuario “controlar información” y en el obtenido a partir de la descripción se realizan dos tareas, “consolida la información” y “controla la información”. Luego, en caso de ser necesario se ejecuta la tarea “solicitar información adicional” la cual no se encuentra en el modelo original. Creemos que estas diferencias se deben a la falta de correcciones que se pueden haber tenido en una instancia de discusión con un cliente.

La siguiente diferencia que encontramos, es que la tarea de usuario “negociar cupos” y el subproceso “adjudicar movilidad” se ejecutan en una especie de loop en el modelo original dadas ciertas condiciones. Esto difiere un poco del modelo obtenido en el cual depende de una condición para que se vuelvan a ejecutar las dos actividades. Por otro lado, en el modelo original, se permite volver a ejecutar el subproceso sin la necesidad de volver a ejecutar la tarea de usuario “negociar cupos” que en el caso del modelo obtenido no se permite.

Para finalizar, en el modelo original se permite la finalización del proceso sin que se haya adjudicado una movilidad, lo cual en el modelo obtenido no se permite.

Por último nos queda ejecutar el proceso obtenido en el BPMS de Activiti, pero antes, veremos otro enfoque que se puede dar al analizar un proceso de negocio, que entendemos no sería el caso más común pero podría darse.

7.4 Generación a partir del modelo BPMN 2.0 original

Este enfoque consiste en tomar el modelo BPMN 2.0 hecho a mano y a partir de este generar el modelo textual que luego será utilizado para generar el modelo BPMN 2.0. El objetivo de este enfoque es mostrar que si bien se soporta una gran variedad de construcciones del estándar, los modelos generados por el prototipo pueden perder algunas de las características artesanales que tienen los modelos elaborados manualmente.

Primero que nada, analizamos el modelo del subproceso Adjudicar Movilidades, donde no encontramos ninguna diferencia respecto a lo obtenido en la sección anterior cuando analizamos la descripción del mismo. Por esto es que el modelo obtenido es igual al obtenido en la sección anterior el cual se puede ver en la Figura 7.22.

El modelo textual construido a partir del modelo BPMN 2.0 de la Figura 7.2.a es el siguiente.

```
la DGRC realiza el subproceso DGRC_recibir_postulaciones; varias veces.
unica alternativa de DGRC_recibir_postulaciones, se interrumpe si
transcurre 1 mes
  la DGRC controla documentacion.
  si se cumple,
    analizarDisponibilidad entonces la DGRC analiza disponibilidad.
    si no no hace nada fin
  si se cumple,
    evaluacionExterna entonces la DGRC realiza evaluacion.
    si no no hace nada fin
  si se cumple,
```

```

    realizarPriorizacion entonces la DGRC realiza priorizacion.
    si no no hace nada fin
    mientras necesitaReasignarCupo,
        si se cumple,
            necesitaNegociarCupos entonces la DGRC negocia cupos.
            si no no hace nada fin
        si se cumple,
            existenMovilidadesAAsignar entonces la DGRC realiza el
subproceso
    DGRC_adjudicar_movilidad.
    si no no hace nada fin
    fin
    si se cumple,
        huboadjudicacion entonces
            al mismo tiempo,
                1 la DGRC envia mensaje a contraparte extranjera.
                2 la DGRC envia mensaje a servicio o postulante.
            fin
        si se cumple,
            requiereDocumentacion entonces la DGRC procesa documentacion
                adicional.
            si no no hace nada fin
        si no no hace nada fin
    fin

```

Figura 7.24. Modelo textual construido a partir del modelo BPMN 2.0 de la figura 7.2.a

Entonces para finalizar con el capítulo, veremos la ejecución del proceso obtenido en la sección 7.3.1 donde se muestran los pasos para desplegar el mismo en el BPMS de Activiti y algunos de los pasos más importantes de la ejecución.

7.5 Ejecución del modelo BPMN 2.0 obtenido

Por último nos queda ejecutar el proceso. Pero antes mencionaremos algunos detalles que tuvimos que ajustar para poder realizar el despliegue del proceso en el BPMS de Activiti. Por defecto, en el motor de Activiti viene configurado un conjunto de usuarios y roles, los cuales no coinciden con los del proceso, por lo que construimos un modelo textual idéntico al obtenido, pero cambiando los actores de forma de poder ejecutarlo. Otra opción hubiera sido configurar un conjunto de usuarios que se adaptara a las necesidades del proceso original pero consideramos que no aportaba al objetivo de esta sección.

7.5.1 La ejecución paso a paso

Entonces, primero que nada es necesario desplegar el proceso en el motor, esto se puede hacer en la pestaña de gestión, en el menú Despliegues, seleccionamos la opción Cargar nueva. De esta forma aparece el diálogo que vemos en la Figura 7.26.a donde seleccionamos Elegir un archivo y buscamos en el sistema de archivos el proceso a desplegar. Una vez desplegado, podemos ver una imagen del modelo gráfico como se ve en la Figura 7.26.b.

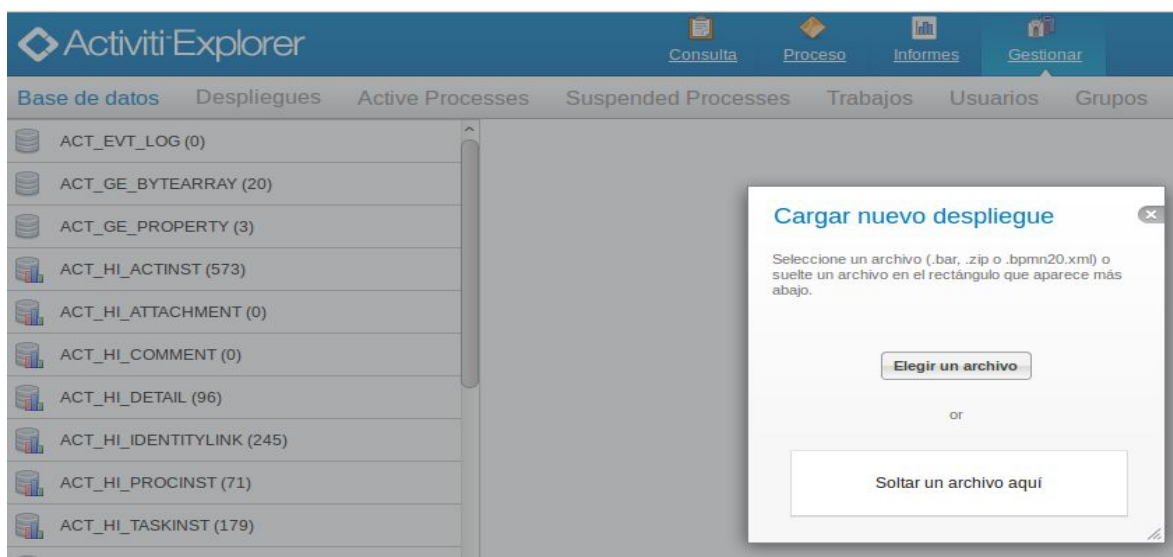


Figura 7.26.a. Desplegando el proceso

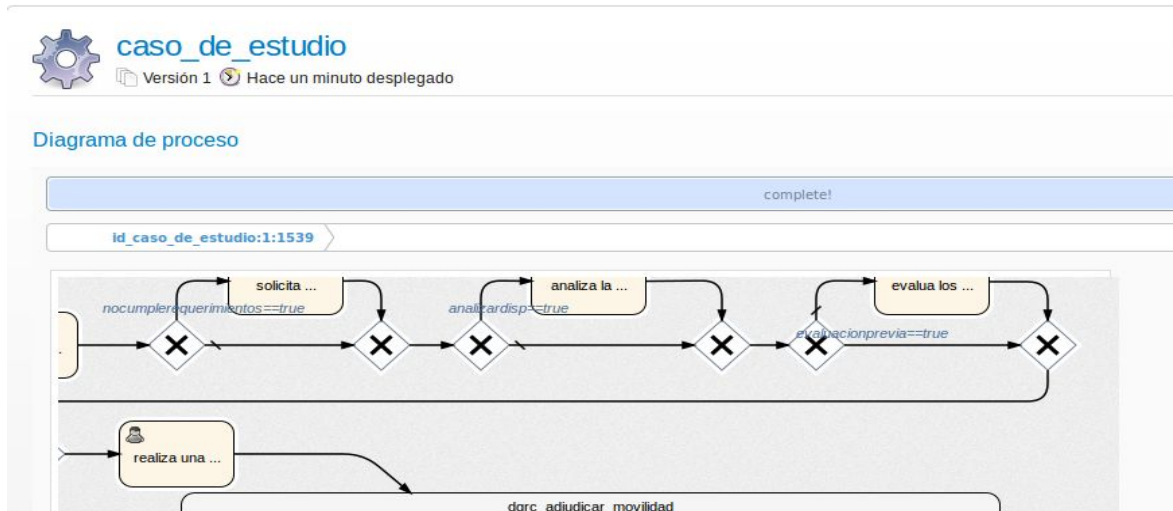


Figura 7.26.b. Proceso desplegado

Luego de desplegado es momento de iniciar el proceso, para lo cual hacemos click en el botón de Iniciar proceso que se ve en la Figura 7.27.

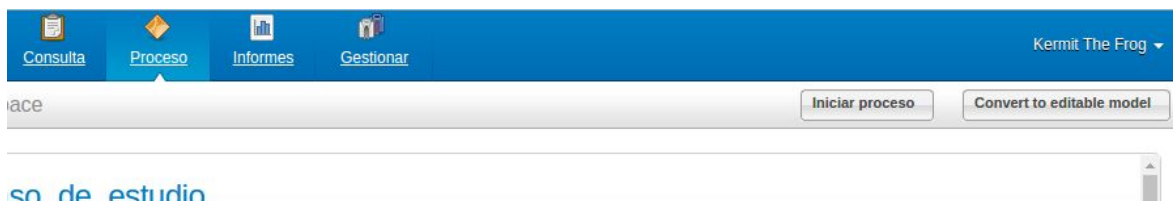


Figura 7.27. Iniciando el proceso desplegado

Luego de hacer click en el botón, nos aparece una ventana donde debemos completar un formulario como se ve en la Figura 7.28. Este formulario sirve para ingresar los valores que van a tomar las variables de las condiciones definidas.



caso_de_estudio


Versión 1  Hace 16 minutos desplegado

nocumplerequimientos
 analizardisp
 evaluacionprevia
 realizarpriorizacion
 confirmainteres
 haymascandidatos
 negociarmascupos
 docadicional

Figura 7.28. formulario de inicio del proceso

Luego de completar el formulario, vamos a la pestaña Proceso, seleccionamos el menú Mis instancias y seleccionamos el proceso que fue iniciado recientemente del listado de instancias. Si nos movemos para abajo, podremos ver la información del proceso como se muestra en la Figura 7.29. Esta información irá cambiando a medida que se vaya avanzando en la ejecución del proceso.

Tareas

NOMBRE	PRIORIDAD	ASIGNADO A	FECHA DE VENCIMIENTO	CREADO	COMPLETADAS
 recibir postulaciones	50			Hace momentos	

Variables

NOMBRE	VALOR
analizardisp	false
confirmainteres	true
docadicional	true
evaluacionprevia	true
haymascandidatos	false
negociarmascupos	false
nocumplerequimientos	false
realizarpriorizacion	false

Figura 7.29. información del proceso una vez iniciado.

Vayamos a ejecutar una de las tareas. Para esto vamos a la pestaña Consulta. Seleccionamos el menú En cola y el rol al cual está asignada la tarea que queremos ejecutar, en este ejemplo, las tareas están todas asignadas al rol Marketing, por lo que este será el que seleccionemos. Una vez seleccionado el rol, nos aparece un listado de las tareas que este puede ejecutar. Entre estas tareas se encuentra la primera del proceso, la cual se da por completada luego de pasado 1 mes (para simplificar el ejemplo, lo cambiamos por 1 minuto). Si seleccionamos una tarea nos aparece la información de la misma como se ve en la Figura 7.30.a, y para ejecutarla, primero es necesario “pedir” la tarea, esto implica hacer click en el botón Pedir tarea donde luego nos aparece la opción de completarla como se ve en la Figura 7.30.b

Figura 7.30.a. pedido de tarea

Figura 7.30.b. completado de tarea

Repetimos los pasos para las siguientes tareas como se ve en las Figuras 7.31.a y 7.31.b

Figura 7.31.a

Figura 7.31.b

Luego llegamos a lo que sería un evento de mensaje, pero como Activiti no soporta este tipo de elementos, nosotros realizamos una transformación de este tipo de componentes por tareas de servicio o tareas de usuario según corresponda, como fue explicado en capítulos previos. En este caso, es un evento de envío de mensaje, por lo que es transformado a una tarea de servicio la cual se encarga de enviar un mail. Para esto usamos una herramienta que nos permite ver si el mensaje es enviado correctamente. Esta herramienta es conocida como Fake SMTP Server, la cual simula un servidor de correos, con el cual se comunica Activiti para enviar los correos. Al ser una tarea de servicio, no hay interacción con el usuario, por lo que la captura que mostramos en la Figura 7.32 se corresponde con el envío del mensaje en el servidor de correos donde se puede ver el correo enviado.

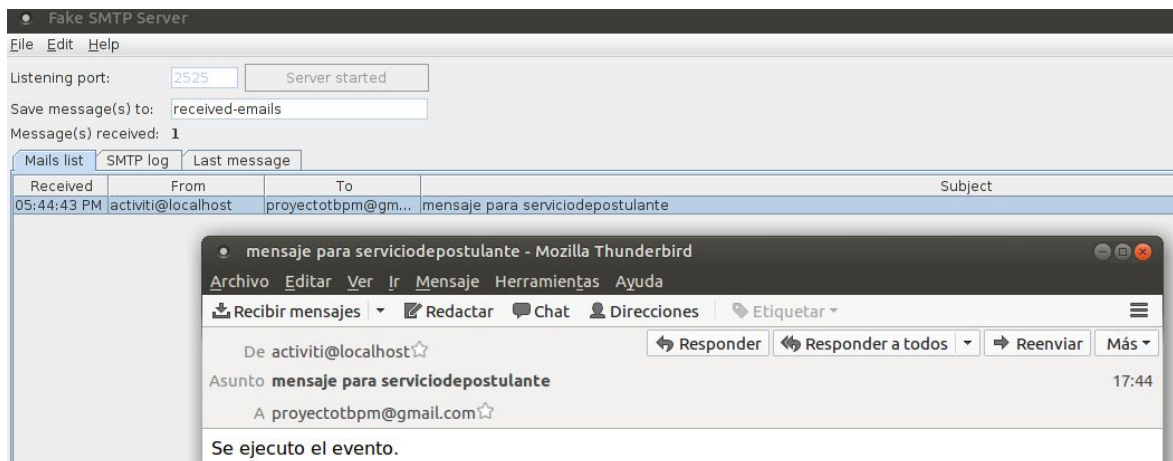


Figura 7.32. Vista del correo enviado en Fake SMTP Server y Mozilla Thunderbird

Luego de ejecutar la tarea de servicio que envía el mail, el proceso queda esperando por la respuesta, la cual es una tarea de usuario y una vez completada la siguiente tarea es la que está marcada en rojo en la Figura 7.33.

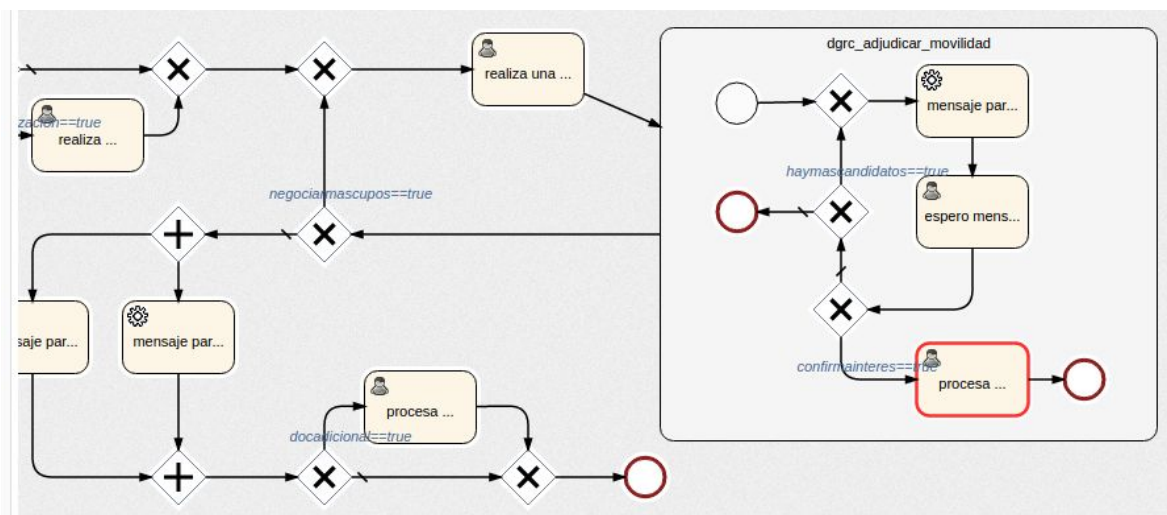


Figura 7.33. Siguiete tarea a ser ejecutada

Luego de ejecutar esta tarea se enviaran los dos mensajes para comunicar la adjudicación a la contraparte extranjera y al servicio o postulante. Se puede ver en la Figura 7.34 una captura del

servidor de correos fakeSMTP donde aparecen los correos enviados.

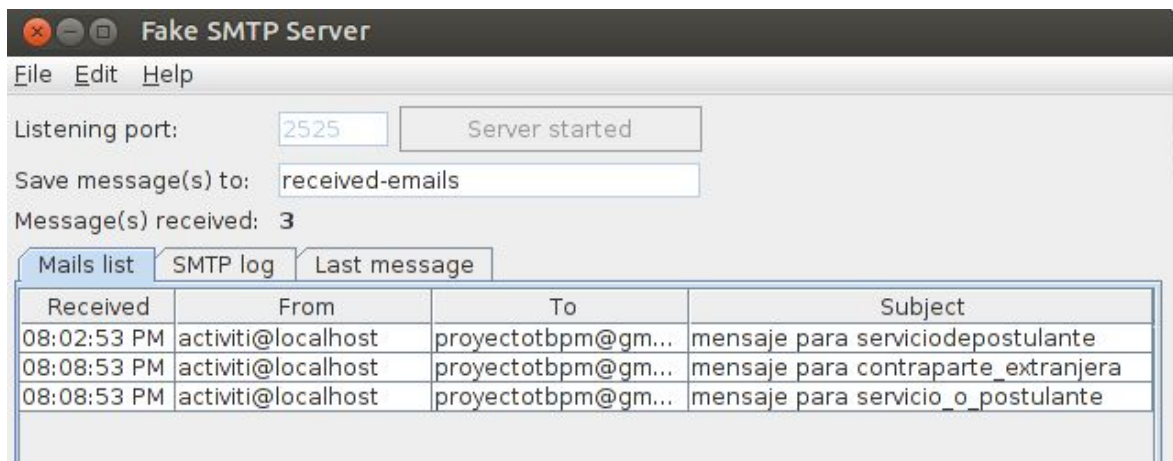


Figura 7.34. Captura de fakeSMTP donde se envían los mensajes correspondientes

Una vez ejecutadas estas tareas, queda la última tarea a ejecutar ya que la condición “docadicional” fue seteada en true. En la Figura 7.35 se puede ver que se está esperando por la ejecución de esta tarea.

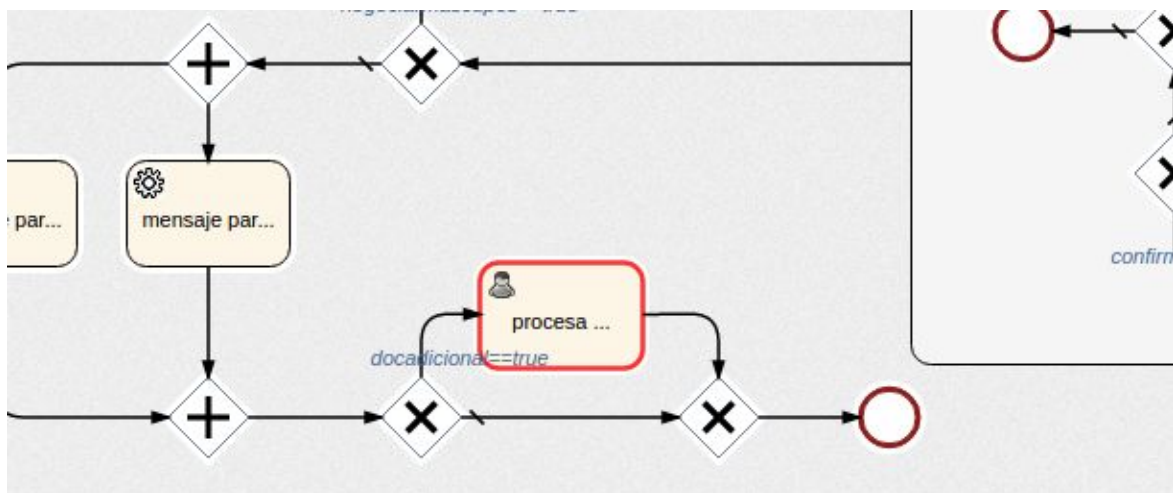


Figura 7.35. Última tarea esperando por ser ejecutada

Luego de ejecutada esta tarea, damos por terminado la ejecución del proceso. Para ver otros caminos de ejecución posibles, sería cuestión de setear las variables del proceso en los valores correspondientes, pero hay que tener en cuenta que según los valores seteados se podría tener un problema de loop infinito ya que las variables no son modificadas en tiempo de ejecución.

8 Conclusiones y trabajo a futuro

En este capítulo retomaremos los problemas planteados en el capítulo 3 y analizaremos las soluciones implementadas para los mismos. Para cada uno, haremos una comparación entre los resultados que se obtuvieron y los resultados esperados, luego mencionaremos algunas limitantes que fueron surgiendo y para finalizar comentaremos sobre los posibles trabajos a futuro que podrían realizarse.

8.1 Resultados obtenidos

Específicamente, los problemas a resolver fueron los siguientes:

- definir la representación de procesos de negocio a utilizar.
- analizar la representación seleccionada para tomar un subconjunto representativo de sus elementos.
- definir el lenguaje que represente los elementos del subconjunto seleccionado
- definir las perspectivas que se quieren obtener.
- definir las transformaciones necesarias para generar de forma automática las perspectivas seleccionadas.

La representación de procesos de negocio a utilizar fue definida en la propuesta del proyecto, donde se especifica que se utilizara BPMN 2.0. Entendemos que esta fue la mejor opción ya que es un lenguaje conocida por nosotros y además es un estándar ISO que, como se mencionó en el capítulo 3, está fuertemente adoptado por la industria y por la academia. Esto simplificó mucho la curva de aprendizaje inicial en cuanto al estándar.

Al analizar BPMN 2.0 para tomar un subconjunto representativo de sus elementos, entendemos que el conjunto obtenido es lo suficientemente expresivo para modelar procesos de forma consistente.

Conseguimos formalizar un lenguaje capaz de expresar procesos de negocio. El mismo es simple y estructurado, donde queda bien definido el conjunto de oraciones válidas y evita las ambigüedades típicas del lenguaje natural. En particular construimos una oración para cada componente del subconjunto seleccionado de BPMN 2.0, de forma de poder generar cada uno de estos de forma automática. El lenguaje definido tiene una cierta estructura pero la misma no dificulta su entendimiento. Consideramos que es expresivo, y puede ser entendido por cualquier usuario dado que utiliza frases simples de la lengua española.

Analizamos las tres perspectivas asociadas de los modelos BPMN 2.0. Una primer perspectiva de análisis, la cual tiene los elementos definidos como atributos de análisis según el estándar, de esta forma generamos modelos gráficos acordes a las necesidades de la etapa de diseño, con todos los componentes gráficos del proceso y sin entrar en detalles de ejecución, los cuales son abordados en las restantes perspectivas. Esta perspectiva es la que mencionamos como formato básico en el capítulo 5.

Una segunda perspectiva asociada a la fase de diseño y parcialmente a la fase de ejecución, siguiendo estrictamente el estándar BPMN 2.0, agrega los atributos a cada componente apuntando principalmente a la fase de ejecución. Es decir, maneja la asociación de datos para las

tareas de usuario o condiciones sobre los flujos asociados a las compuertas excluyentes. En este caso, hay atributos de ejecución que no manejamos de manera estándar dada su complejidad como en las tareas de servicio y los eventos de mensaje. De todas formas logramos ejecutar en el motor de ejecución yaoqiang sin necesidad de realizar cambios sobre el proceso. Esta perspectiva es la que mencionamos en el capítulo 5 como formato estándar.

Finalmente manejamos una tercer perspectiva asociada al BPMS de Activiti, también relacionada con la fase de diseño y parcialmente con la fase de ejecución. que es capaz de ejecutarse en la versión 5.16 del mismo [34]. Es importante recordar que, como fue mencionados en el capítulo 5, fue necesario realizar algunos cambios en algunos componentes para poder ejecutar los procesos de negocio asociados a esta perspectiva. Si bien creemos que esto son limitaciones, no interfieren en la ejecución de manera significativa y permite simular estos elementos.

Las tres perspectivas son generadas automáticamente a partir del modelo textual ingresado. Las últimas dos pueden ser ejecutadas sin la necesidad de realizar modificaciones. Para cada una de ellas se ofrece un modelo gráfico y un modelo en formato XML.

En cuanto a las transformaciones necesarias para generar las diferentes perspectivas, las mismas fueron mencionadas en el capítulo 5 y el resultado de estas se puede ver en el prototipo, donde a partir del modelo textual se generan de forma automática estas perspectivas. Entendemos que el prototipo funcional implementado, brinda una interfaz de usuario simple, con mecanismos de ayuda para el usuario, de forma que éste pueda utilizarla sin problemas. A su vez, utilizando este prototipo podemos obtener las diferentes perspectivas que mencionamos previamente, de forma de poder exportarlas tanto como imagen como en formato XML.

Finalmente realizamos una evaluación en base al caso de estudio donde realizamos dos metodologías, una primera que sería el enfoque más común el cual, a partir de la descripción textual del caso de estudio se construye un modelo textual y a partir de este se genera el modelo BPMN 2.0. La segunda metodología parte del modelo BPMN 2.0 original del caso de uso a partir del cual se construye el modelo textual y a partir de este se obtiene el modelo BPMN 2.0. Si bien el modelo del caso de estudio planteado es válido, no es estrictamente correcto en lo que respecta a los aspectos de calidad planteados para el proyecto de grado, un ejemplo serían las tareas de usuario que se vinculan con participantes externos o las compuertas que no están bien balanceadas, pero creemos que en ambos casos, los modelos BPMN 2.0 obtenidos son cercanos al modelo original del caso de estudio, salvo algunas diferencias en cuanto a elementos no soportados y algunos detalles de estructuras de las bifurcaciones que aparecen en el modelo original. Diferencias que podrían ser abordadas y mitigadas en una instancia de discusión con un cliente.

En resumen, consideramos que las principales contribuciones de este proyecto son:

- Un lenguaje simple y estructurado capaz de expresar los componentes más comunes del estándar BPMN 2.0.
- Una herramienta simple para la generación de modelos BPMN 2.0 desde texto expresado en el lenguaje definido, incluyendo aspectos de calidad.
- Una herramienta capaz de generar modelos en dos niveles:
 - i) básico sin información extra en formato XML para ser importado en cualquier herramienta de modelado;

ii) completo con información extra de datos, variables y demás elementos listo para ejecutar en el motor de Activiti.

8.2 Limitaciones y trabajo a futuro.

Una clara limitante de nuestro trabajo es que no soporta todos los componentes BPMN 2.0. Esto deja una línea de trabajo abierta para extender el lenguaje hasta ser capaz de expresar los elementos no soportados del estándar especificados en la sección 3.1.

En cuanto al lenguaje, si bien cada regla es simple y está bien definida, sin permitir ambigüedades, cualidad que aporta a la comprensión del texto, este puede volverse complicado de leer debido a la anidación que surge a la hora de definir compuertas. Creemos que para paliar esta limitación sería interesante trabajar el feedback visual por parte del prototipo, resaltando las oraciones construidas, nivel de profundidad (o anidación) o generación parcial de diagramas al vuelo.

La flexibilidad del lenguaje puede verse afectada por su simpleza donde cada construcción viene dada por una palabra que lo identifica, como por ejemplo, para definir una tarea de servicio hay que emplear la frase “utiliza el servicio”. Para atacar este tipo de problemas en la sección 2.4 citamos a Friedrich que establece una lista categorizada por comportamiento de “stop words”. De esta manera para identificar el comportamiento de iteraciones, hay una lista de palabras como mientras tanto, mientras, etc. Un enfoque similar se puede utilizar para lograr mayor flexibilidad en el lenguaje, definiendo una lista de sinónimos para los identificadores de construcciones y preprocesar la gramática para agregar estos sinónimos como alternativas.

Otra limitante del proyecto es la cantidad de motores soportados. La solución plantea una versión para ejecutar en el BPMS de Activiti, que como especificamos, requiere algunas extensiones para simplificar el modelado de procesos. La misma línea de desarrollo sigue Camunda [35] que especifica extensiones para mejorar el soporte de ejecución. Otros motores requieren de algún artefacto extra para poder ser ejecutado; un ejemplo de esto son los motores desarrollados en javascript como bpmn [36] y bpmn-engine [37], que necesitan piezas de código para ejecutar los distintos tipos de tarea. Un trabajo a futuro sería incluir soporte para otros BPMS.

Para mejorar la usabilidad de la aplicación se podría desarrollar una herramienta de completado automático integrada en la aplicación que ofrezca una serie de opciones a escribir en base al texto ingresado. Dado que el desarrollo se basa en peg.js, se podría utilizar el buen reporte de errores para construir dicha herramienta. Basta con introducir un carácter no esperado (por ejemplo un carácter de control unicode) para que el parser reporte un error retornando una lista con las opciones esperadas, que a su vez se podrían utilizar como entrada de datos de la herramienta de sugerencias.

El módulo de generación de auto-layout actualmente utiliza la herramienta yaoquiang, dado que es un BPMS completo escrito en java, se podría migrar a alguna otra herramienta de auto generado para mejorar la performance y evitar algunos bugs persistentes. El grupo bpmn.io está desarrollando una herramienta de auto layout [38] que podría integrarse de manera más homogénea al prototipo dado que está escrita en javascript aunque de momento soporta una cantidad muy reducida del estándar BPMN.2.0.

Para mejorar la calidad de los modelos obtenidos sería interesante definir un mecanismo inverso a la generación de modelos, es decir, en base a un modelo BPMN 2.0 generar texto, de forma de

poder iterar a la hora de generar los procesos. Tal como concluye el trabajo de H. Leopold [39] es posible generar un modelo textual en lenguaje natural en base a un modelo BPMN 2.0 basándose en el enfoque de “generación del lenguaje natural real”, utilizando herramientas lingüísticas. Este enfoque no suena apropiado para nuestra aplicación dado que el lenguaje determina un modelo, y un modelo (si está bien formado y cumple con buenas prácticas) determina el mismo conjunto de oraciones del lenguaje. Tal como se menciona en [40] sería apropiado el enfoque “llenado de template”, donde se tiene un template de todas las oraciones válidas y hay que cambiar con los parámetros extraídos del modelo BPMN 2.0.

9 Referencias

- [1] BPMS - <http://dl.acm.org/citation.cfm?id=209894> [Último Acceso - 12/2016]
- [2] Estandar BPMN 2.0 - <http://www.omg.org/spec/BPMN/2.0/> [Último acceso - 09/2016]
- [3] Friedrich, F. (2010). *Automated generation of business process models from natural language input* (Doctoral dissertation, HUMBOLDT-UNIVERSITÄT ZU BERLIN) - <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.228.2293&rep=rep1&type=pdf> [Último acceso - 09/2016]
- [4] Udelar - Universidad de la República de Uruguay
- [5] The Complete Business Process Handbook Volume 1: Body of Knowledge from Process Modeling to BPM. August-Wilhelm Scheer, Henrik von Scheel, Mark von Rosing,
- [6] Business Process Management. Capítulo 1. Mathias Weske
- [7] Overview Event-driven process chain notation - <http://www.ariscommunity.com/event-driven-process-chain> [Último acceso - 02/2017]
- [8] YAWL: yet another workflow language - <http://www.yawlfoundation.org/> [Último acceso - 02/2017]
- [9] Lenguaje unificado de modelado - <http://www.omg.org/spec/UML/2.5/> [Último acceso - 02/2017]
- [10] XPD L - <http://www.wfmc.org/XPD L.htm> [Último acceso - 02/2017]
- [11] BPEL - <http://bpel.xml.org/> [Último acceso - 02/2017]
- [12] J. Mendling, H.A. Reijers, W.M.P. van der Aalst, Seven process modeling guidelines (7PMG), Information and Software Technology, Volume 52, Issue 2, February 2010, Pages 127-136, ISSN 0950-5849 [Último acceso - 09/2016]
- [13] XML - <https://www.w3.org/TR/REC-xml/> [Último acceso - 09/2016]
- [14] International Organization for Standardization - <http://www.iso.org/iso/home.html> [Último acceso - 02/2017]
- [15] ISO/IEC 19510:2013 - http://www.iso.org/iso/catalogue_detail.htm?csnumber=62652 [Último acceso - 02/2017]
- [16] LO Johansson, M Wärja, H Kjellin, S Carlsson - 2008 "An evaluation of business process model techniques, using Moody's quality criterion for a good diagram"
- [17] Ford, Bryan p (2004). "Parsing Expression Grammars: A Recognition Based Syntactic Foundation". Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - <https://ece.uwaterloo.ca/~vganesh/TEACHING/W2014/lectures/peg.pdf> [Último acceso -

02/2017]

[18] Estandar EBNF - <https://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf> [Último acceso - 09/2016]

[19] PEG.js - <http://pegjs.org/> [Último acceso - 09/2016]

[20] ECMAScript 5 - <http://www.ecma-international.org/ecma-262/5.1/> [Último acceso - 09/2016]

[21] Definición de JSON - <https://tools.ietf.org/html/rfc7159> [Último acceso - 09/2016]

[22] x2js - <https://github.com/abdmob/x2js> [Último acceso - 09/2016]

[23] Editor BPMN Yaoqiang - <http://bpmn.sourceforge.net/> [Último acceso - 09/2016]

[24] Activiti BPM Platform - <http://activiti.org/> [Último acceso - 09/2016]

[25] Modelador bpmn.io - <https://github.com/bpmn-io/bpmn-js> [Último acceso - 09/2016]

[26] DOT Language - <http://www.graphviz.org/content/dot-language> [Último acceso - 09/2016]

[27] Graphviz - Graph Visualization Software - <http://www.graphviz.org/> [Último acceso - 09/2016]

[28] Node.js - <https://nodejs.org/en/> [Último acceso - 09/2016]

[29] Electron - <http://electron.atom.io/> [Último acceso - 09/2016]

[30] JQuery - <https://jquery.com/> [Último acceso - 09/2016]

[31] Bootstrap - <http://getbootstrap.com/> [Último acceso - 09/2016]

[32] pretty-data - <https://github.com/vkiryukhin/pretty-data> [Último acceso - 09/2016]

[33] bootstrap-tour - <http://bootstraptour.com/> [Último acceso - 09/2016]

[34] Activiti 5.XX - <https://github.com/Activiti/Activiti/releases> [Último Acceso - 03/2017]

[35] Camunda BPMN - <https://camunda.org/> [Último acceso - 02/2017]

[36] paed01 bpmn engine - <https://github.com/paed01/bpmn-engine> [Último acceso - 02/2017]

[37] e2ebridge bpmn engine - <https://github.com/e2ebridge/bpmn> [Último acceso - 02/2017]

[38] autolayout - <https://github.com/bpmn-io/bpmn-moddle-auto-layout> [Último acceso - 02/2017]

[39] Generating Natural Language Texts from Business Process Models - http://link.springer.com/chapter/10.1007%2F978-3-642-31095-9_5 [Último acceso - 10/2016]

[40] Presentación *Natural Language Generation from Process Models* H. Leopold - https://www.fing.edu.uy/inco/eventos/bpmuy/presentaciones/presLeopold_BPMuy ICT4V.pdf [Último acceso - 10/2016]