Instituto de Computación - Facultad de Ingeniería

Universidad de la República

Proyecto de Grado:
Modelado de la variabilidad de Procesos de Negocio

Informe final

Ignacio Betancurt

Alejandro Brusco

Nicolás Dinetti

TUTORES:

Andrea Delgado

Daniel Calegari

Octubre 2016

Montevideo, Uruguay

Resumen

Un Proceso de Negocio (BP, del inglés *Business Process*) es un conjunto de actividades coordinadas para alcanzar un objetivo de negocio. La variabilidad en un BP es la posibilidad de elegir entre diferentes opciones al momento de tomar una decisión en el diseño o particularización de un BP. Entre los lenguajes utilizados para el modelado de un BP se encuentra *Business Process Model and Notation 2.0* (BPMN 2.0) lenguaje estándar definido por la OMG (*Object Management Group*) en el cual se centra este proyecto. BPMN 2.0 define un metamodelo y notación gráfica asociada. Dicha especificación provee un amplio set de elementos para representar el común de las situaciones, facilita la comunicación entre involucrados, y promueve la interoperabilidad con formato estándar XML.

Como parte de este proyecto se estudió el estado del arte de la variabilidad en BP, analizando varias propuestas para modelar variabilidad en procesos, clasificándolas y evaluando sus ventajas y desventajas. Con el objetivo de visualizar cada una de ellas, se aplican las mismas a un ejemplo basado en un proceso de *Check-in* de un vuelo presente en uno de los artículos estudiados.

Con base en el análisis realizado se propone un nuevo enfoque denominado BPMNext que extiende BPMN 2.0, adaptando la propuesta vSPEM para procesos software a la notación BPMN 2.0. BPMNext permite definir actividades (tareas, sub-procesos) y roles como puntos de variación y asociar distintas variantes posibles. Para dar soporte al lenguaje BPMN 2.0 extendido para variabilidad, se construye una herramienta a modo de plugin de *Eclipse* extendiendo el plug-in de BPMN2 Modeler, la cual permite generar nuevos procesos a partir de un proceso base con puntos de variación junto a las variantes de los mismos.

Finalmente, para demostrar la factibilidad de la propuesta se aplica el enfoque en un caso de estudio real elaborado por el grupo COAL para la Dirección General de Relacionamiento y Cooperación Internacional (DGRC) de UdelaR, que representa el proceso de adjudicación de movilidades de estudiantes de grado. En este caso de estudio se visualiza el proceso base, y las posibles derivaciones del mismo teniendo en cuenta las variantes.

Palabras clave: *Procesos de negocio, BPMN, variabilidad, flexibilidad, Eclipse plug-in, Provop, CVL, vBPMN, PESOA, C-EPC, C-BPMN, BPMNt, BPMN**

Contenido

1 Introducción

Un Proceso de Negocio (BP, del inglés *Business Process*) es un conjunto de actividades realizadas en coordinación en un entorno organizacional y técnico, para alcanzar un objetivo del negocio [1]. El modelado de estos procesos es una actividad clave en el ciclo de vida de las organizaciones y debe realizarse con atención ya que repercutirá luego en los resultados obtenidos. Los procesos de negocio presentan, en algunos casos, puntos de variabilidad que determinan variantes del proceso base. Algunos ejemplos son los procesos de ventas para productos diferentes, o los procesos de contabilidad con variantes según el país.

En la última década han surgido diversas propuestas para tratar la variabilidad de procesos focalizándose en el modelado de familias de procesos con variantes. Una familia de procesos es un conjunto de procesos de negocio con sus variantes dependientes del contexto. Con esta representación se evita modelar cada variante en forma separada lo que implica duplicación y dificultad en el mantenimiento de las partes comunes. Los enfoques considerados pueden categorizarse como variables por restricción o extensión, según consideren en el proceso base todas las variantes o no. Éstas proponen definir una familia de procesos mediante un proceso base común (denominado proceso configurable o variable) más las variantes que cada proceso específico requiere. Entre las propuestas analizadas encontramos PESOA [2], C-EPC [2], Provop [3], y otras específicas de BPMN.

De esta forma, cada variante es derivada desde el proceso variable al proceso particular, eliminando o agregando fragmentos del modelo, según parámetros de configuración. En general los lenguajes de modelado de procesos de negocio no soportan en forma explícita la representación de variabilidad, ni existe soporte metodológico ni de herramientas para tratar con familias de procesos de negocio.

Para representar procesos de negocio similares, con pequeñas variaciones, es necesario generar un proceso independiente para cada uno. Esto es un problema ya que al compartir en gran medida parte del proceso, la modificación de alguna sección del mismo implica reflejar el cambio en cada uno de los procesos que lo contienen. Por otro lado, esto genera que tengamos una gran cantidad de procesos prácticamente iguales con diferencias menores.

Los objetivos específicos de este proyecto son [4]:

- OE1: Investigar enfoques existentes para el modelado de la variabilidad en procesos de negocio en particular utilizando el lenguaje *Business Process Modeling Notation* 2.0 (BPMN 2.0, [5])
- OE2: Analizar ventajas y desventajas de cada enfoque, así como el soporte tecnológico que ofrecen
- OE3: Seleccionar/adaptar/extender el enfoque más adecuado según el contexto definido
- OE4: Desarrollar un prototipo de herramienta de soporte a la propuesta como plug-in de Eclipse

El resto del documento se organiza de la siguiente manera:

- En el Capítulo 2 introduciremos conceptos claves y necesarios para el entendimiento del informe. A su vez, será el encargado de detallar las propuestas que plantean soluciones al problema de la variabilidad en procesos de negocio. Aquí se profundizará en cada una y permitirá construir una tabla resumiendo las características de cada enfoque.
- El Capítulo 3 ahondará en la extensión necesaria al lenguaje BPMN2 para permitir el modelado de variabilidad de elementos y justificará el porqué de la elección de dichos elementos como variables.
- Los requerimientos de la solución, así como el diseño e implementación del misma serán el eje central del Capítulo 4.
- Para ver el funcionamiento de la solución en la práctica, se decidió ejemplificar con un proceso de negocio real elaborado en el Instituto de Computación de la Facultad de Ingeniería. El Capítulo 5 abarca este punto.
- Finalizando, encontraremos en el Capítulo 6 un conjunto de conclusiones y sugerencias de trabajo a futuro teniendo en cuenta los objetivos del proyecto y el problema planteado.

2 Estado del arte

Para la comprensión de este proyecto es fundamental tener conocimientos de ciertos conceptos claves además de realizar un relevamiento y análisis de propuestas de variabilidad existentes. Con este objetivo dividiremos este capítulo en 2 secciones que definiremos a continuación. En la primer sección de conceptos y definiciones, estaremos planteando los conceptos relacionados con Procesos de Negocio siguiendo con los conceptos propios de variabilidad; mientras que la segunda sección se enfocará en los distintos enfoques analizados.

2.1 Conceptos y definiciones

2.1.1 Procesos de Negocio

En primer lugar, definimos a un Proceso de Negocio (BP) como un conjunto de actividades realizadas en coordinación en un entorno organizacional y técnico, para alcanzar un objetivo del negocio [1].

La gestión de los BPs (BPM, del inglés *Business Process Management*) incluye conceptos, métodos y técnicas para apoyar el diseño, gestión, configuración, ejecución y análisis de los BPs.
La base de BPM es la representación explícita de los BPs con sus actividades y restricciones de ejecución entre ellas. Luego de que un BP haya sido definido, pasa a ser candidato de análisis, mejora y publicación. Las empresas se pueden beneficiar del uso de sistemas informáticos para la coordinación de las actividades involucradas en los BPs. Estos sistemas son conocidos como *BPM systems* [1].
En la Figura 2.1 se presenta un ejemplo de un proceso de negocio modelado en la notación estándar *Business Process Model and Notation* (BPMN 2.0) [5] de la OMG.

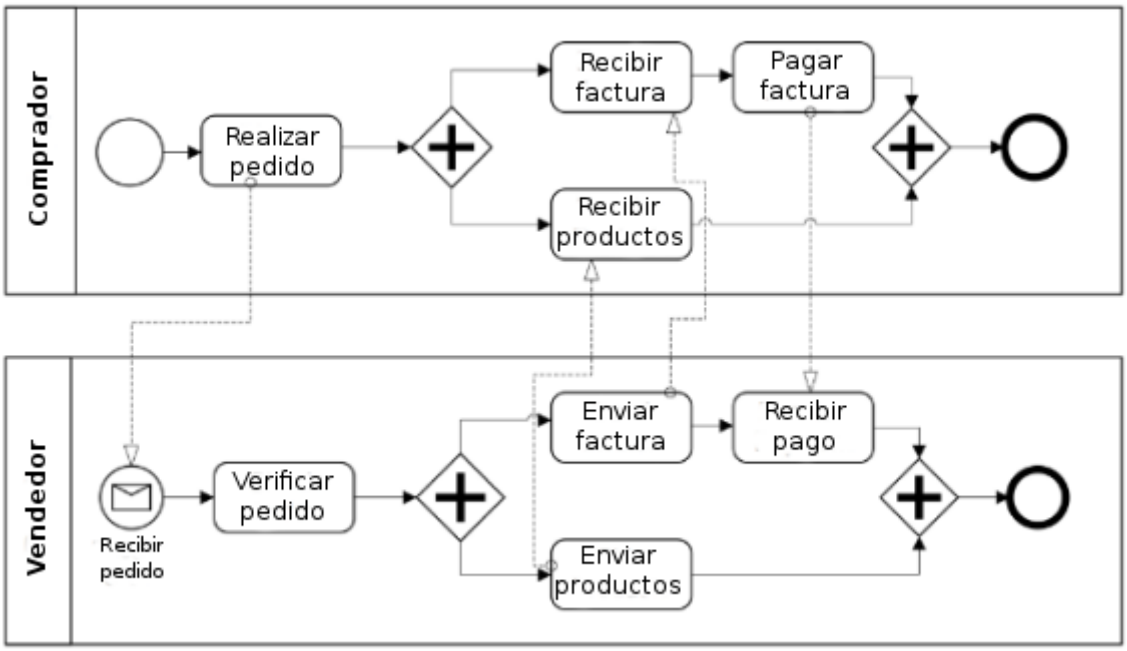


Figura 2.1: Proceso BPMN de la compra de un producto de [1]

Los modelos de proceso de negocio especifican el orden en el que las distintas actividades serán ejecutadas.

Por otro lado, también detallan qué actividades pueden ejecutarse de forma concurrente. Por ejemplo, tomando la figura 2.1 vemos que luego de recibir una orden (Recibir pedido), se ejecuta concurrentemente el envío de la factura (Enviar factura) y el envío de los productos (Enviar productos).

La figura 2.1 representa las actividades que un vendedor y comprador ejecutan para procesar una compra. El mismo está compuesto por el proceso llevado adelante por el vendedor, por el proceso del comprador y la interacción entre ambos. La interacción se da de la siguiente manera:

1. El comprador envía un mensaje con la orden al vendedor (Realizar pedido).
2. El vendedor recibe el mensaje (Recibir pedido). Se analiza la información del mensaje y continúa el procesamiento.
3. El vendedor envía la factura (Enviar factura) y los productos solicitados (Enviar productos).
4. El comprador recibe la factura (Recibir factura).
5. El vendedor envía el pago (Pagar factura).
6. Finalmente, el comprador recibe los productos (Recibir productos).

El flujo de mensajes puede representar, además de mensajes electrónicos, el transporte de objetos físicos como en este caso, los productos solicitados.

Las interacciones de un conjunto de participantes en un BP colaborativo son especificadas en una coreografía de procesos. El término coreografía indica la ausencia de un agente central que controla las actividades en los BPs involucrados. Cada participante es dueño de su proceso el cual ocurre como una orquestación bajo el control de dicho participante.

La única forma de interacción entre participantes es a través del envío y recepción de mensajes. Para alcanzar una correcta interacción, los procesos de los participantes involucrados en un BP colaborativo deberán acordar una coreografía antes de empezar a interactuar.

A modo de entender un poco más los conceptos y tecnologías que son relevantes para BPM, Weske plantea el ciclo de vida de un BP el cual se puede ver en la figura 2.2.

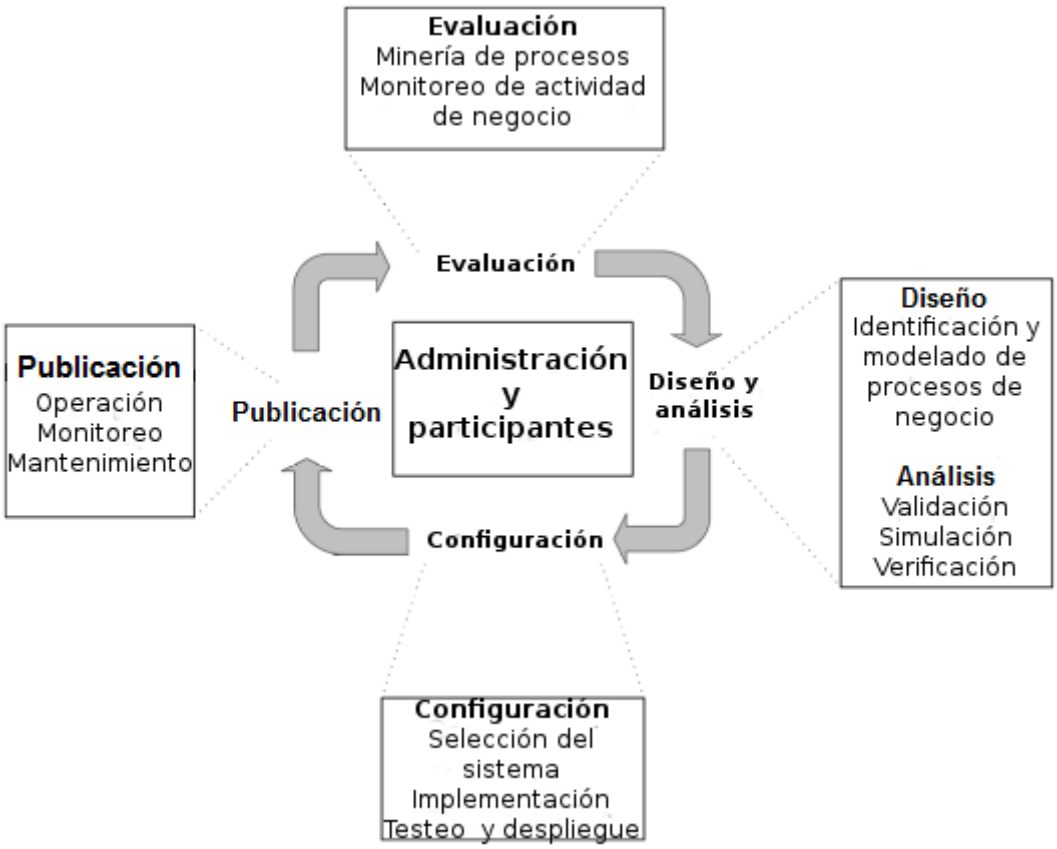


Figura 2.2: ciclo de vida de un BP

El mismo, consiste en 4 fases relacionadas entre sí y organizadas en forma cíclica, donde muchas actividades de diseño y desarrollo se llevan a cabo y no es raro ver actividades concurrentes en múltiples de estas fases.

El ciclo comienza en la fase de diseño y análisis, en la que se llevan a cabo estudios sobre el BP y su entorno organizacional y técnico. Basados en esto, se llega a una representación del modelo del mismo, la cual expresada en una notación gráfica facilita la comunicación entre las diferentes partes involucradas en el ciclo. Una vez que se llega a un modelo inicial se debe validar, por ejemplo, verificando que todos los escenarios del BP se encuentren reflejados en el modelo del mismo.

Una vez que el modelo del BP esté diseñado y verificado se debe implementar. Para ello existen diferentes maneras. En el caso de ser implementado por un conjunto de políticas y procedimientos que los empleados de la empresa tienen que cumplir, puede llevarse a cabo sin ningún apoyo por un

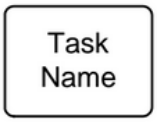
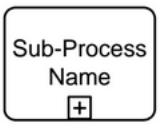
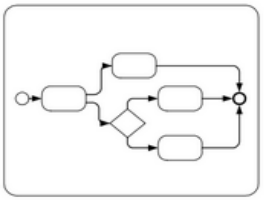
sistema BPM dedicado. En este caso, durante la fase de configuración, es elegida la plataforma donde se va a implementar el BP. El sistema BPM necesita ser configurado acorde al entorno organizacional de la empresa y el BP que se debe gestionar. Estas configuraciones incluyen tanto las interacciones entre empleados con el sistema así como la integración del sistema con los sistemas de software existentes.

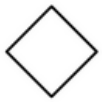
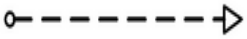
Una vez que el sistema fue configurado, la implementación del BP necesita ser testeada. Para esto, son usadas técnicas tradicionales de testing a nivel de actividades de proceso para verificar si el sistema se comporta como era esperado. A nivel de proceso, es importante realizar pruebas de integración y performance para detectar posibles errores en tiempo de ejecución. Cuando concluyen las pruebas, el sistema es instalado en el ambiente correspondiente.

En tercer lugar, luego de la fase de configuración, el BP puede ser instanciado. Esta fase abarca el tiempo de ejecución real del BP siendo el sistema BPM el encargado de controlar activamente la misma según como fue definido en el modelo del BP. El sistema BPM debe brindar un componente de monitoreo que muestre el estado de estas instancias. Esta información es muy importante, por ejemplo, para proveer información acertada a un usuario que consulta sobre el estado de su proceso. Muchos sistemas BPM proveen un monitoreo basado en colores sobre las distintas instancias del BP y muchos otros se basan en el estado del BP activo. Además, durante la ejecución existe información típicamente almacenada en forma de un archivo de log, indicando eventos que ocurrieron durante la misma. Esta información es sumamente útil para la evaluación del proceso que se realiza en la siguiente fase del ciclo.

Esta última fase, utiliza la información disponible para evaluar y mejorar los modelos de los BPs y sus implementaciones. Utilizando técnicas de proceso, estos logs son analizados con el objetivo de determinar la calidad de los modelos y la correctitud del entorno de ejecución. Por ejemplo, la monitorización de actividades puede identificar que una de las mismas consume demasiado tiempo debido a la escasez de recursos.

Como fue mencionado, el lenguaje BPMN 2.0, tiene entre sus características que: provee de un amplio set de elementos para representar el común de las situaciones en los BP; facilita la interpretación humana definiendo una representación visual para la mayoría de los elementos del proceso; y mejora el mapeo con lenguajes ejecutables [7]. A continuación se presentan en general los elementos básicos que propone BPMN:

Event	
Start  Intermediate  End 	Representa algo que ocurre durante un proceso o coreografía. Afectan el flujo del modelo y generalmente tienen una causa o impacto. Se representan como círculos con centros en blanco y existen 3 tipos de ellos según cómo afectan al flujo. Los distintos tipos de eventos son: Evento de inicio: indican el comienzo de un proceso o coreografía. Eventos intermedios: ocurrente entre un evento de inicio y un evento de fin. Afectan al flujo pero no permiten comenzar o finalizar un proceso directamente. Eventos de fin: indican el final de un proceso o coreografía.
Activity	
	Término genérico para representar a las acciones que un actor debe realizar en un proceso. Se representan como un rectángulo con bordes redondeados y pueden ser del tipo atómicas o no atómicas. A las primeras les llamaremos *Tasks* (o Tareas) y a las segundas *Sub-Process* (o Subprocesos).
Task	
	Es una *Activity* atómica incluida en un proceso. Se utilizan para representar acciones que no precisan estar detalladas a mayor nivel.
Collapsed Sub-Process	
	Subproceso cuyos detalles no se visualizan en el diagrama. Se representan con un signo de suma en el centro del borde inferior de la figura indicando que se trata de un Subproceso y que tiene un mayor nivel de detalle.
Expanded Sub-Process	
	Subproceso en el cual es posible visualizar su contenido con detalle, y contenido dentro de su figura.

Gateway	
	Figura utilizada para controlar la divergencia y convergencia de Sequence Flows en un proceso o coreografía. Permite crear ramificación, bifurcaciones, fusiones y uniones de caminos.
Sequence Flow	
	Figura utilizada para indicar el orden en el que se ejecutarán las actividades en un proceso o coreografía.
Message Flow	
	Muestra el flujo de mensajes entre dos participantes que están preparados para enviar y recibirlos. Para representar a estos participantes, utilizaremos dos Pools separados dentro de un diagrama de collaboration.
Pool	
	Representación gráfica de un participante en una collaboration. Puede oficiar de *swimlane* y de contenedor gráfico para particionar un conjunto de *Activities* de otros Pools.
Lane	
	*Lane* es una sub-partición dentro de un proceso (que puede estar incluido en un *Pool*). Se utilizan para organizar y categorizar *Activities*.
Data Object	
	Figura utilizada para representar información necesaria para ejecutar *Activities* o generadas por éstas. Pueden representar a un único objeto o a una colección de ellos.[5]

2.1.2 Variabilidad de Procesos de Negocio

Otros conceptos importantes a introducir en el contexto del proyecto son el de flexibilidad y variabilidad. Para poder entender estos conceptos, según [2], es necesario introducir las distintas fases de un proceso: *design-time* (tiempo de diseño), *customization-time* (tiempo de configuración) y *run-time* (tiempo de ejecución).

Llamaremos *design-time* a la fase en la cual es creado el modelo de procesos personalizable. Todas las decisiones de diseño tomadas en esta etapa impactarán en la familia de procesos del modelo.

Por otro lado se encuentra la fase *customization-time*. En esta etapa se considera el modelo de procesos personalizable creado en la *design-time*, y se personaliza para obtener variantes de procesos particulares. El modelo de procesos personalizado identifica a una variante de procesos lista para ser activada. Las decisiones de personalización tomadas en esta etapa afectarán únicamente a dicha variante.

Finalmente, la última fase es la llamada *run-time*. Aquí el proceso de modelos personalizado es ejecutado como si hubiese sido modelado independientemente. Para cada una de estos proceso, se toman decisiones en tiempo de ejecución (por ejemplo: aceptar o rechazar una petición en particular, basándose en la evidencia obtenida). Ese tipo de decisiones sólo afectarán a un instancia específica.

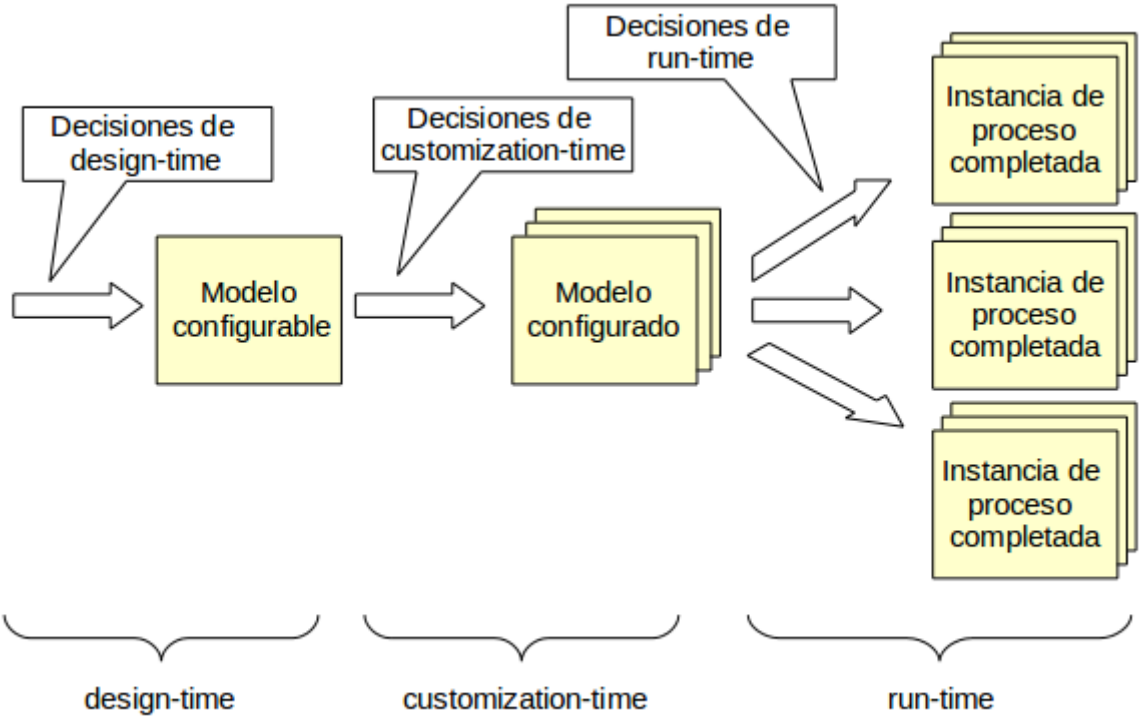


Figura 2.3: Etapas de un proceso de [2]

La diferencia entre la flexibilidad y la variabilidad en los BP es que la primera está asociada a decisiones en el *run-time,* mientras que la variabilidad concierne a decisiones durante las fases de *design-time* y *customization-time*.

La variabilidad es la posibilidad de optar por diferentes variantes de opciones al momento de tomar una decisión. Existen 2 formas distintas de modelar la variabilidad en un proceso [2]:

- Variabilidad por restricción

  Comienza considerando todos los comportamientos de todas las posibles variantes de procesos de un modelo de procesos personalizable. La personalización se alcanza restringiendo el comportamiento del modelo de procesos personalizable.
- Variabilidad por extensión

  Comienza de forma opuesta a la variabilidad por restricción. El modelo de procesos personalizable, no contiene todos los posibles comportamientos, pero en cambio, representa los comportamientos más comunes o los comportamientos más compartidos por la mayoría de las variantes del proceso.

Definiremos como puntos de variación (VP, del inglés *Variation Point*) a las actividades abstractas (tareas o subprocesos) que precisan ser realizadas a través de una variante concreta. Le llamaremos variantes a las distintas opciones que pueden sustituir a un punto de variación determinado.

En algunos de los enfoques analizados se utiliza también el concepto de modelo de características. En desarrollo de software, un modelo de características (o *feature model*) es una representación compacta de todos los productos de la Línea de Productos de Software (SPL) en términos de características. Los *feature model* son representados visualmente por medio de diagramas de características. Los modelos de características son utilizados durante todo el proceso de desarrollo de productos y se utiliza para generar otros bienes, como documentos, definición de la arquitectura, o piezas de código.

Un *feature model* es un modelo que define características y sus dependencias, comúnmente representado como un diagrama de características + restricciones, aunque también podría representarse como una tabla de combinaciones posibles [7]. A modo de ejemplo, en la figura 2.4, podemos ver el *feature model* asociado al ejemplo que se utiliza en el capítulo 2 para presentar los distintos enfoques analizados.

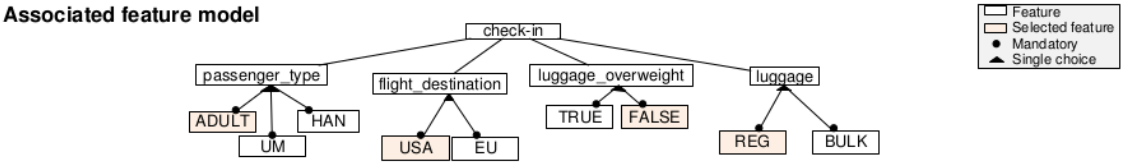


Figura 2.4: Feature model check-in de un vuelo de [7]

Por otro lado, también debemos definir lo que se conoce como *process tailoring*. Su traducción al español sería proceso a medida o adaptación de un proceso, el cual en este contexto, consiste en reutilizar un proceso estándar de software y adaptarlo a las necesidades específicas de una organización o determinado proyecto. Según [8], este proceso puede darse en dos diferentes niveles: a nivel organizacional y a nivel de proyecto. Es recomendable que la organización tenga su propio set de procesos estándar (previamente adaptados a sus necesidades organizacionales) para luego utilizarlos como base para procesos de proyectos específicos [7].

Para poder construir una solución al problema de la variabilidad en BPMN, analizamos en forma previa distintas propuestas o enfoques ya existentes. Estos enfoques no están todos basados en el lenguaje BPMN. Algunos son propuestas independientes del lenguaje, y otros son basados en otros lenguajes como *Event-driven Process Chain* (EPC) [9], *Software Process Engineering Metamodel* (SPEM) [10] y *Common Variability Language* (CVL) [11]. En la siguiente sección presentamos el análisis y evaluación realizados sobre dichas propuestas.

2.2 Análisis de propuestas de variabilidad existentes

A continuación detallaremos cada uno de los enfoques analizados, destacando sus características y mostrando los mismos en un ejemplo de aplicación.

2.2.1 Ejemplo de aplicación – Check-in de un vuelo

Para poder visualizar las características de cada uno de los enfoques analizados, consideramos un proceso de *Check-in* de un vuelo y aplicamos cada propuesta al mismo. Este proceso de ejemplo fue presentado en [7], y por su sencillez es excelente para poder visualizar las propuestas.
El proceso consiste en el *Check-in* realizado en el aeropuerto por una persona que desea viajar en avión. Esta persona puede ser: un menor de edad no acompañado (UM, del inglés *Unaccompanied minor*), una persona discapacitada (HAN, del inglés *Handicapped*), un adulto (ADULT). Por otro lado, el destino del viaje es Estados Unidos (USA) o la Unión Europea (EU) y además otras variaciones respecto a características del equipaje. De acuerdo a la configuración seleccionada, existen distintas tareas que se deben llevar a cabo. En cada enfoque se ilustra cómo se visualizan en el diagrama estos puntos de variación.
Las imágenes con la aplicación de cada enfoque al Check-in están en inglés dado que se utilizaron diagramas en este idioma, y en las imágenes propias se mantuvo buscando conservar los nombres de los VPs y variantes, y poder comparar los enfoques entre sí.

2.2.2 Análisis detallado de propuestas

2.2.2.1 Provop

En primer lugar, analizamos la propuesta de variabilidad que provee *PROcess Variants by OPtions* (Provop) [3]. La misma se presenta como una variabilidad por extensión, únicamente sobre Actividades, mediante una secuencia de cambios en el proceso base ajustando así al contexto, derivando en un proceso particular a partir de un proceso configurable. Provop no está especificado para un lenguaje particular sino que se aplica a elementos de cada notación, por ejemplo UML, BPMN; lo cual es una gran ventaja.
Dentro de un proceso configurable existen regiones configurables las cuales están delimitadas por 2 puntos de ajuste (diamantes negros). Estas regiones no pueden ser especificadas en tiempo de resolución, debiendo ser definidas previamente, y es donde se aplican adaptaciones estructurales (opciones de cambio). Se definen a su vez, alternativas de configuración en términos de opciones de cambio que incluyen:

1. una lista de operaciones de cambios atómicos modificando una región del proceso configurable.
2. una regla de contexto que define las condiciones de contexto bajo las cuáles las operaciones de cambios se deben aplicar.

En cuanto a las herramientas, existen editores gráficos que soportan la creación de modelos de procesos configurables, permitiendo crear tanto el modelo base como especificar sus opciones disponibles para su configuración.

El prototipo de Provop mencionado en el artículo [7], el cual introduce facilidades de configuración y administración de variantes, chequea si las opciones definidas violan alguna restricción definida en el conjunto de opciones. Si se detecta alguna violación de restricciones, el prototipo notifica al ingeniero de procesos.

Provop soporta 4 tipos de relaciones:

1. *Dependencia*

   La Dependencia significa que las respectivas opciones son siempre: aplicadas de forma conjunta con el proceso base; o que ninguna de ellas se utiliza cuando se configura una variante de procesos.
2. *Exclusión mutua*

   Permite reducir las posibles combinaciones de opciones que pueden ser aplicadas al modelo de procesos base. Dos opciones son mutuamente excluyentes si no pueden ser nunca utilizadas en conjunto porque constituyen variaciones entre ellas (por ejemplo, ambas opciones añaden la misma actividad al modelo de proceso base pero en diferentes posiciones, lo que genera diferentes variantes).
3. *Limitación en el orden de ejecución*

   Dado que una opción puede insertar una actividad cuyos atributos sean modificados por una segunda, determinar un orden de ejecución para estas opciones es fundamental (en el ejemplo a continuación, se puede ver como no es posible ejecutar la segunda opción sin haber ejecutado previamente la primera). Por lo tanto, Provop permite la especificación del orden en el cual se deben aplicar las opciones al proceso base.
4. *Jerarquía*

   La jerarquía de opciones es una combinación entre la relación de dependencia y el orden de ejecución. Más precisamente, si una opción “hija” debe aplicarse a un modelo de procesos base, la correspondiente opción “padre” también debe ser aplicada. Para prevenir inconsistencias debido a opciones “padre” no determinísticas, éstas siempre son aplicadas antes que sus opciones “hijas” [12].

En la Figura 2.5 se presenta el caso de estudio modelado con PROVOP.

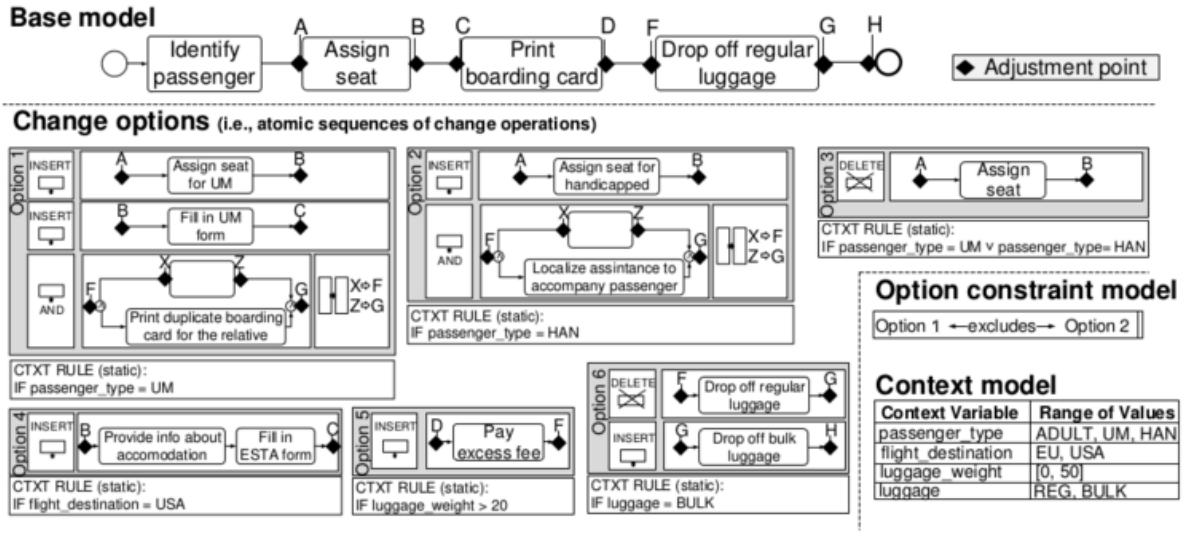


Figura 2.5: Modelo del proceso base, variantes, contexto y restricciones de Check-in en Provop de [7]

En la figura 2.5 se puede ver el proceso base y distintos puntos de ajuste junto con 6 variantes que se pueden seleccionar. También en dicha figura, se observa el *"Option constraint model"* donde se muestran opciones excluyentes entre sí. Por ejemplo si se considerara que el Check-in es realizado por un **ADULT**, que viaja a **USA** y que su equipaje es de tipo **BULK** que no excede los **20kg**, se deberían considerar las opciones 4 y 6 lo que agregaría las tareas *"Provide info about accommodation"* y *"Fill in ESTA form"* entre los puntos B y C, quitaría la tarea "*Drop off regular luggage"* entre los puntos F y G, y agregaría la tarea *"Drop off bulk luggage"* entre los puntos G y H generando así un modelo particular a partir de las variantes seleccionadas.

En cuanto a ventajas y desventajas en la utilización del mismo notamos, lo ya mencionado, en cuanto a la no dependencia de un lenguaje específico, pero además, la expresividad, modularización, la fácil adopción de los elementos gráficos al ser una cantidad bastante acotada y la posibilidad de expresar restricciones sobre las combinaciones de opciones. Por el contrario como desventajas encontramos que tanto el resultado como el modelo base no son muy expresivos por no contar con una variabilidad sobre los gateways y por la restricción lineal como punto inicial. También notamos que el entendimiento del modelo y el armado de todas las posibilidades se complejiza debido a la utilización de varios componentes. Destacamos también que el prototipo mencionado en el artículo basado en ARIS Business Architect ya no se encuentra disponible.

2.2.2.2 CVL

Common Variability Language (CVL) [13] provee los mecanismos para representar variaciones por extensión, únicamente sobre Actividades, en cualquier *Domain Specific Language* (DSL) no extendidos o sobrecargados con información de variabilidad. Por ejemplo CVL + BPMN ya que la especificación actual de BPMN no soporta el modelado de la variabilidad de proceso, así como tampoco los motores de ejecución.

Como veremos más desarrollado luego, PESOA y C-EPC integran todas las posibilidades en un único modelo sobrecargado con las especificaciones de variabilidad de modo que dificultan la comprensión de los mismos. Sin embargo, CVL provee mecanismos para representar las variaciones en forma separada aliviando el impacto que los problemas de variabilidad tienen en los modelos BP, consiguiendo una mejor legibilidad, mayor comprensión y escalabilidad

CVL está basado en el enfoque Base-Variación-Resolución (BVR), es decir que es representado por un Modelo Base (Base Model) el cual contiene las partes comunes de todos los procesos y *placements fragments* (básicamente puntos de variación); un Modelo de Variación (Variation Model) que contiene los *replacements fragments* (en esencia, alternativas) que se colocarán en los *placements fragments;* y un Modelo de Resolución (Resolution Model) que especifica un conjunto de condiciones de contexto el cual determina bajo qué condiciones los *replacements* pueden ser instanciados.

Por último, el Modelo de Contexto el cual provee el razonamiento formal de la información de contexto. Basado en ontologías, provee un fuerte vocabulario semántico para la representación del conocimiento y para describir situaciones específicas de contexto.
Posibilita el análisis del conocimiento del dominio utilizando lógica de primer orden.

En la Figura 2.6, 2.7 y 2.8 se presenta el caso de estudio modelado con CVL

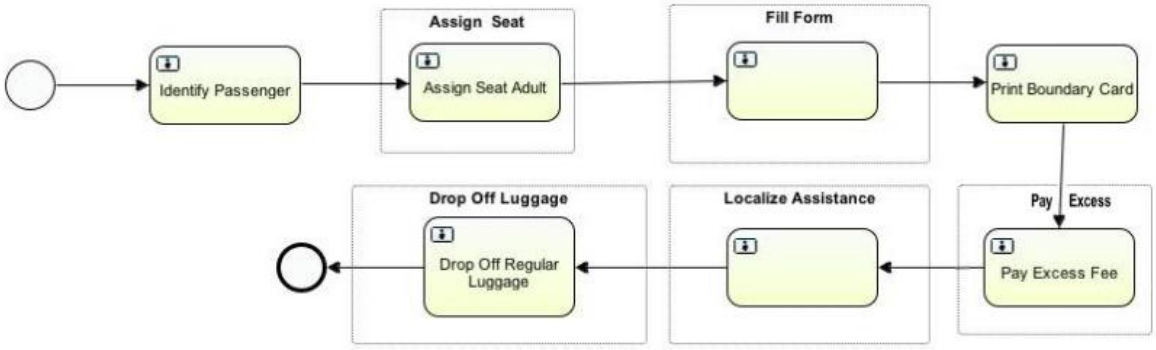


Figura 2.6: Modelo del proceso base de Check-in en CVL

Al igual que para el enfoque anterior, en la figura 2.6 se tiene el proceso base de Check-in modelado en CVL. En este caso, en el proceso base los puntos de variación están dados por las tareas encuadradas en gris y con el nombre como identificador de los mismos.

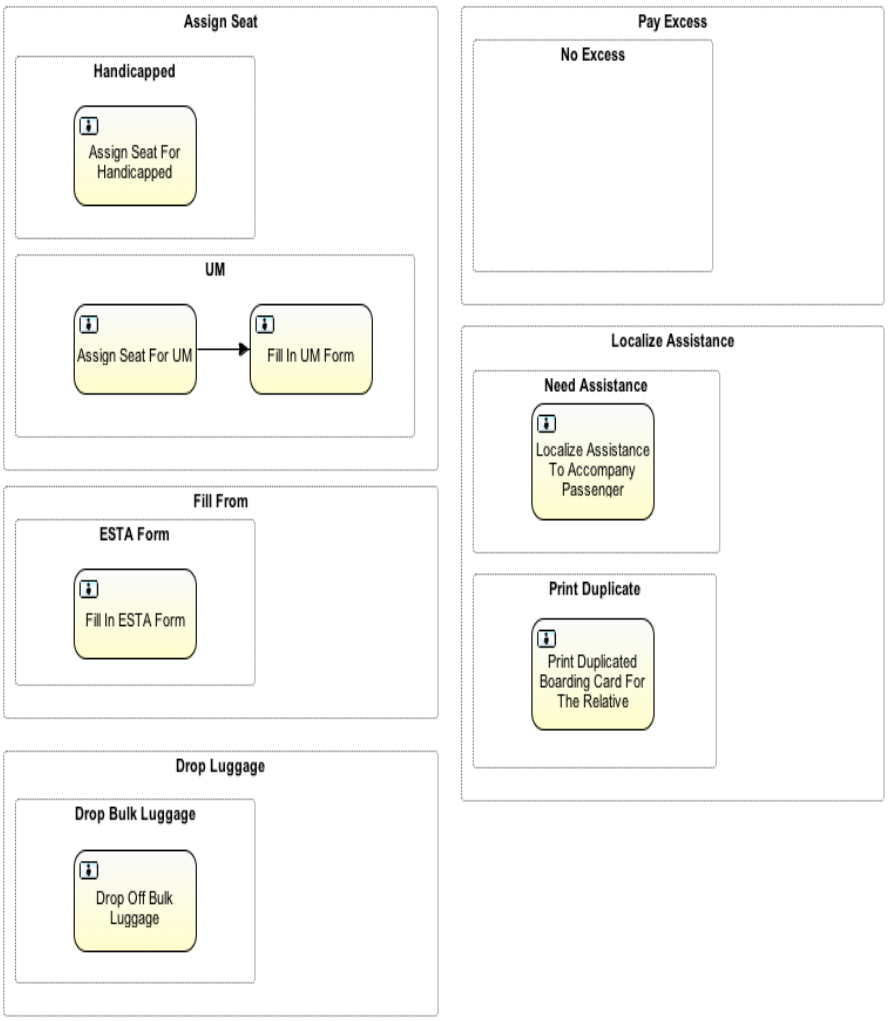


Figura 2.7: Modelo de variación de Check-in en CVL

Las variaciones se presentan en la figura 2.7 dentro de un recuadro con el mismo nombre del punto de variación al que pertenecen. Dentro de ese recuadro cada variante se encuentra en otro recuadro que contiene la construcción de esa variación.

Figura 2.8: Modelo de resolución de Check-in en CVL

Por otro lado, en la figura 2.8 se muestra un ejemplo de un modelo de resolución para el caso de *"Handicapped"* en el cual se determina por ejemplo que la variación considerada para *"Assign Seat"* es la variación *"Handicapped"*. Estas 3 figuras mencionadas, componen el BVR.

En cuanto a ventajas y desventajas nuevamente notamos la no dependencia de un lenguaje específico, la modularización y la posibilidad de expresar restricciones sobre las combinaciones de opciones. A su vez se destaca que las herramientas que se utilizan siguen siendo vigentes, estando disponibles. Por el contrario al igual que Provop como desventajas encontramos que el resultado no es muy expresivo por no contar con una variabilidad sobre los gateways y por la restricción lineal como punto inicial. También notamos que el entendimiento del modelo y el armado de todas las posibilidades se complejiza debido a la utilización de varios componentes.

2.2.2.3 vBPMN

Para hacer frente a los desafíos de manejar el modelado de la complejidad en tiempo de diseño, así como la creación de variantes en tiempo de ejecución de manera integrada, vBPMN propone una combinación de flujos, reglas y reutilización de patrones en el modelado de casos [14]. Además, propone algo llamado *event-awareness* que se refiere a contar con la capacidad de reaccionar de forma diferente ante los cambios de estado externos en diferentes contextos de datos. Pero al ser en tiempo de ejecución, no aplica a este estudio.
Como solución, vBPMN promueve un marco de apoyo a las variantes de flujo basado en BPMN 2.0 en combinación con el lenguaje de reglas R2ML. El mismo utiliza un enfoque por extensión y soporta variabilidad de actividades (Tareas y subprocesos), *gateways* y eventos. No así roles y datos.
Para la construcción del marco, define 3 grandes conceptos:

- *Adaptive Workflow Segments*
  El modelo base se construye con un contexto de datos y segmentos adaptables (*Adaptive Workflow Segments*) que pueden ser objeto de adaptaciones estructurales dependientes de este contexto o no. Un *Adaptive Workflow Segment* puede ser cualquier bloque parcial estructurado del flujo de trabajo, es decir, con una sola entrada y una sola salida. Existen dos

convenciones diferentes para las anotaciones. Los segmentos adaptables se pueden marcar con nodos intermedios identificados con paréntesis rectos. O, se puede poner un diamante negro en la esquina superior izquierda de una sola tarea, lo que implica misma semántica pero con un mayor ahorro de espacio.

La semántica de ejecución define que: En cada momento de entrada a un segmento adaptable, se crea un segmento aislado que no cambia a pesar de que una variable externa puede cambiar mientras se ejecuta el segmento variante. Sin embargo, si el mismo segmento adaptable se ejecuta varias veces, por ejemplo a través de un ciclo, la semántica de ejecución permite la creación de múltiples variantes diferentes del mismo segmento.

- *Event-Aware Adaptation Patterns*

  Las patrones de adaptación (*Event-Aware Adaptation Patterns*) tienen lugar al entrar en un segmento adaptable y se definen en un catálogo de patrones de adaptación BPMN 2.0. El mismo contiene las pautas básicas para la realización de las variantes de comportamiento del flujo base, como la inserción y el salto de tareas, así como los patrones más sofisticados para la conducta reactiva dependiente del contexto y los eventos. Cada patrón se compone de un parámetro implícito <*AdaptationSegment*> relativo al elemento al cual se le realiza la adaptación. Para algunos modelos, se especifican parámetros adicionales relativos a modelar elementos del metamodelo.

  La combinación convenientemente de patrones de adaptación es una característica vista como un potencial clave para un eficiente manejo del modelado de la variabilidad y el mantenimiento.

  Actualmente los autores están trabajando en una descripción completa de los patrones y la formalización de sus limitaciones.

- *Rule-Based Application of Patterns at Runtime with R2ML*

  La conexión entre los datos de contexto y la aplicación de los patrones de adaptación se puede establecer mediante la formulación de reglas. Las condiciones constituyen restricciones de valor sobre las variables de contexto y las acciones contienen patrones de adaptación según los parámetros del catálogo.

En cuanto a la derivación, se modela todo en base a los segmentos adaptables y allí en conjunto con las reglas definidas, se deriva en un nuevo flujo. Los segmentos adaptables pueden estar modelados en concordancia con los patrones de adaptación definidos o no. Pero la utilización de los mismos genera un modelado bastante claro, con pocas diferencias al lenguaje BPMN ya que la forma de identificar las regiones configurables es poco invasiva. Esto hace que se entienda el modelo muy fácilmente.

En lo que respecta a herramientas para aplicar el enfoque, vimos que no existe una todavía. Existe un prototipo en construcción e ideas para el mismo pero no hay referencias al respecto. Si se menciona un prototipo de extensión de *JBoss Drools* para la parte de reglas del que tampoco existen referencias al respecto.

En la figura 2.9 se observan algunas estructuras definidas en este enfoque como los diamantes negros en la esquina superior izquierda de algunas tareas que representan a segmentos adaptables, y las actividades entre eventos intermedios con corchetes que representan lo mismo. A su vez, mediante las reglas definidas con R2ML, se presentan las variantes a aplicar de acuerdo al contexto.

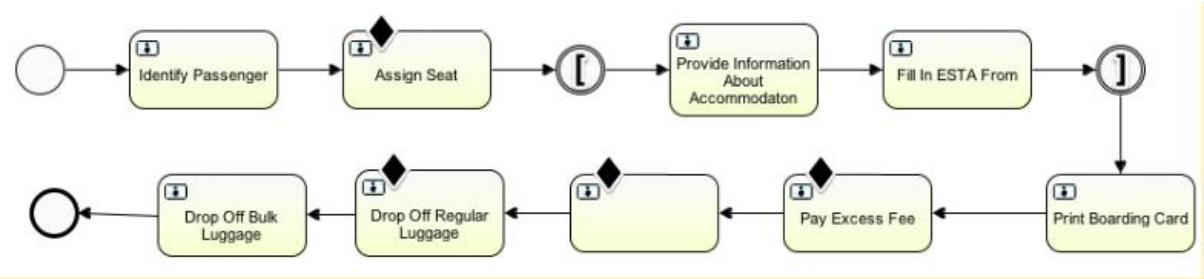


Figura 2.9: Modelo del proceso base de Check-in en vBPMN

Las reglas son:

Rule #1: ON IdentifyPassenger_entry IF passenger_type = UM THEN APPLY replace(task="IdentifyPassenger", "IdentifyPassengerUM");
Rule #2: ON IdentifyPassenger_entry IF passenger_type = HAN THEN APPLY replace(task="IdentifyPassenger", "IdentifyPassengerHAN");
Rule #3: IF flight_destination != USA DELETE ProvideInformationAboutAccomodation;
Rule #4: IF flight_destination != USA DELETE FillESTAForm;
Rule #5: IF luggage_overweight = FALSE DELETE PayExcessFee;
Rule #6: IF passenger_type = UM INSERT PrintDuplicatedBoardingCardForTheRelative WITHIN PayExcessFee;
Rule #7: IF passenger_type = HAN INSERT LocalizeAssistantAccompanyPassenger WITHIN PayExcessFee;
Rule #8: IF luggage != BULK DELETE DropOffBulkLuggage;

Si, por ejemplo, nos encontramos con un pasajero menor de edad (**UM**), que viaje a **Europa** con un equipaje del tipo ***BULK***, pero **con exceso de equipaje**; debemos aplicar las reglas 1, 3, 4 y 6, para obtener el proceso deseado.

Cabe destacar, que Intentar definir un estándar de modelado hace del enfoque algo con mucho potencial, y los patrones adaptables tienen muy desarrollado el tema de eventos, los cuales son dejados de lado por varios enfoques. Todo esto le da al enfoque una mejor posición de partida para posteriores comprobaciones de coherencia del modelo resultado y sobre todo una mejor mantenibilidad y extensibilidad de los propios patrones.
En contrapartida, observamos que los patrones de diseño no son sencillos de entender y que la introducción de otro lenguaje como R2ML para definir las transformaciones le agrega un poco de complejidad. También pudimos observar algunos detalles menores como que la representación de la variante de eliminar una actividad, debemos realizarla colocando una actividad vacía, lo cual no

resulta muy amigable. Y que no queda claro dónde se representan las tareas que no están en el modelo base, sobre todo cuando se desea insertar algo.

2.2.2.4 PESOA

PESOA es una propuesta que se enfoca en el desarrollo y personalización de software, y en las familias de procesos.
Con este objetivo, se introdujo el concepto de *Variant-Rich Process Models* que consiste en construir un único modelo anotado conteniendo todas las posibles variantes del BP. Por lo tanto, esta es una propuesta que plantea la variación por restricción.
Además, esta propuesta es independiente del lenguaje sobre el que se aplique (sirve para BPMN, UML, etc) y se aplica a los elementos de cada notación.
Propone indicar los lugares en donde puede haber variabilidad como puntos de variación, los que también pueden marcar variantes por defecto, o si son exclusivas, opcionales, etc [2].

Este enfoque representa una familia a través de un modelo de procesos configurables que incluye un conjunto de anotaciones. Las anotaciones están asociadas a las actividades de los procesos que pueden estar sujetas a variación. Sin embargo, la semántica de estas anotaciones es compleja y dificulta el entendimiento del modelo [15]. Entre las anotaciones (o estereotipos) encontramos los siguientes:

- <<*Abstract*>> - Actividad que es reemplazada por alguna variante <<*Variant*>> específica
- <<*Variant*>> - Actividad específica. Se selecciona cuando la condición especificada se cumple
- <<*Default*>> - Actividad específica que se utiliza cuando ninguna de las demás condiciones especificadas se cumple
- <<*Null*>> - Actividad parecida al <<*Abstract*>> pero que si la condición especificada no se cumple o alguna de las variantes <<*Variant*>> tampoco, la actividad se elimina
- <<*Optional*>> - Actividad no obligatoria

Por otro lado, se utilizan técnicas independientes del lenguaje como encapsulación, herencia, patrones de diseño, y puntos de extensión.

PESOA opta por un modelado multi-artifact. Uno de ellos es el contexto de aplicación de las actividades variables que se especifica utilizando un modelo de características.
Algo a destacar, es que contrario a PROVOP por ejemplo, las relaciones que puedan existir entre las alternativas de diferentes puntos de variación no se consideran. De esta forma, la aplicación de las mismas es independiente y no hay restricciones entre ellas ni variantes que necesariamente se utilicen en conjunto.

El modelo se compone de una representación gráfica con anotaciones, y un modelo de características. Este modelo de características se agrega adjunto a la definición de cada alternativa como condiciones de contexto que definen cuándo la alternativa será seleccionada [7].

Para realizar una derivación, es necesario realizar la configuración del contexto en base al feature model para remover partes configurables del modelo.

Por el lado de las herramientas que trabajen con PESOA, encontramos que fue realizado un Plug-in de Eclipse que soporta la creación de procesos de modelos configurables. Es un editor gráfico que permite representar regiones configurables incluyendo la configuración de alternativas.

Siguiendo con el ejemplo presentado para los enfoques analizados, en la Figura 2.10 podemos apreciar el caso de Check-in de vuelo modelado en PESOA el cual está representado por un Proceso Base y un Modelo de Características. En el Proceso Base se identifican únicamente tres actividades en color gris las cuales no permiten una variación. El resto se encuentran anotadas según los estereotipos vistos anteriormente. En este caso se considera en el modelo de características que el pasajero es un **adulto**, cuyo destino es **USA** y que su equipaje de tipo **regular no excede** el peso permitido. Por tanto, la actividad *"Seat assignment"* será reemplazada por la variante *"Assign seat"* y no se deberá realizar la actividad opcional *"Fill in UM form"* debido al tipo de pasajero. Además, debido al destino, se requiere realizar las actividades *"Provide information about accommodation"* y *"Fill in ESTA form"*. A su vez, al no llevar sobrepeso en el equipaje, no se deberá realizar *"Pay excess fee"* y nuevamente debido al tipo de pasajero y a que dicha actividad está anotada como <<Null>> se elimina la actividad *"Print duplicated boarding card for the relative".* Por último, al ser un equipaje regular (REG) no se deberá realizar *"Drop off bulk luggage".*

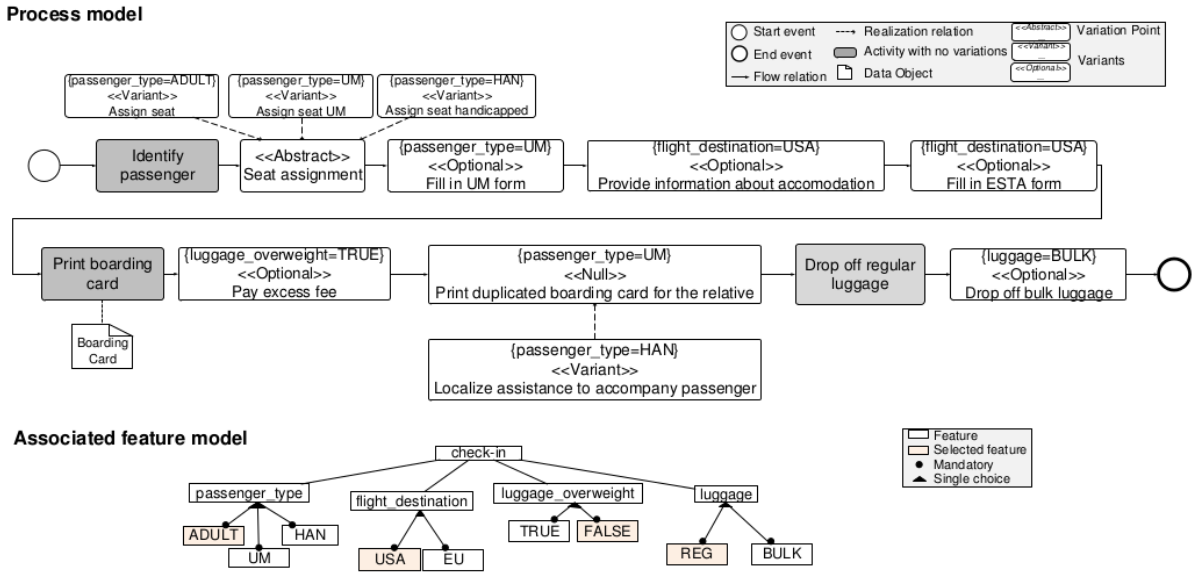


Figura 2.10: Modelo del proceso base de Check-in en PESOA de [7]

En cuanto a los aspectos positivos de este enfoque, cabe destacar la independencia del lenguaje utilizado, y la fácil adopción del enfoque gracias a la acotada cantidad de elementos con la que cuenta.
Si nos centramos en los aspectos negativos, la sobrecarga de anotaciones en el modelo sumado a que todas las variantes deben estar expresadas, conllevan a una alta complejidad para entender el mismo. El hecho de no manejar variabilidad sobre eventos también puede ser un problema.

2.2.2.5 C-EPC

Una de las formas de realizar un modelo de proceso configurable es enriquecer un modelo de procesos con nodos configurables. Un lenguaje de modelado que soporta esta característica es *Configurable-EPC* (C-EPC). C-EPC extiende el lenguaje de modelado de procesos EPC introduciendo elementos configurables.
Este permite modelar las múltiples variantes de un proceso integradas mediante nodos configurables en un único modelo, en un mismo artefacto (*single-artifact*). Para cierto contexto, el modelador es el encargado de configurar la variante del proceso que más se ajuste al mismo.

Los nodos configurables se corresponden con fragmentos con una sola entrada y una sola salida (*SESE fragments*). Existen 2 tipos distintos. En primer lugar se encuentran los *SESE fragments* que consisten en un conector configurable de división seguido por un conjunto de ramas representando alternativas de configuración y un conector configurable de unión (Ver en el ejemplo figura 2.11: *Configurable region 1*). Por otro lado están los *SESE fragments* como funciones configurables que pueden seleccionarse como: ON (será mantenida en el proceso), OFF (será excluida del proceso), OPT (será incluída condicionalmente de acuerdo a la decisión de ejecutarla o no, en tiempo de aprobación). (Ver en el ejemplo figura 2.11: *Configurable region 2*)

Las variantes son representadas como *SESE fragments* insertas entre dos conectores configurables.
Por otro lado, la configuración de contexto se representa separado del modelo de procesos configurable, a través de un cuestionario. La configuración de restricciones semánticas de funciones y conectores configurables, se especifica en términos de requerimientos de configuración asociados a los nodos configurables.

A la hora de querer derivar un proceso, debemos primero derivar los conectores configurables, luego las funciones y por último aplicar una reducción al gráfico.

En conclusión, estamos ante una propuesta de variabilidad por restricción que cuenta con conectores y funciones configurables, que es una extensión del lenguaje EPC y lo hace dependiente al mismo, el cual soporta variabilidad de Actividades, Datos, *Gateways* y Roles.

Las herramientas existentes que se utilizan con este enfoque son: el conjunto de herramientas *Synergia* ([16]) que soporta la creación de un modelo de procesos configurable a través de un editor gráfico, y el validador *C-EPC Validator* ([17]) que chequea el cumplimiento de los requerimientos y guías de configuración.

En la Figura 2.11 se presenta el caso de estudio aplicado a C-EPC.

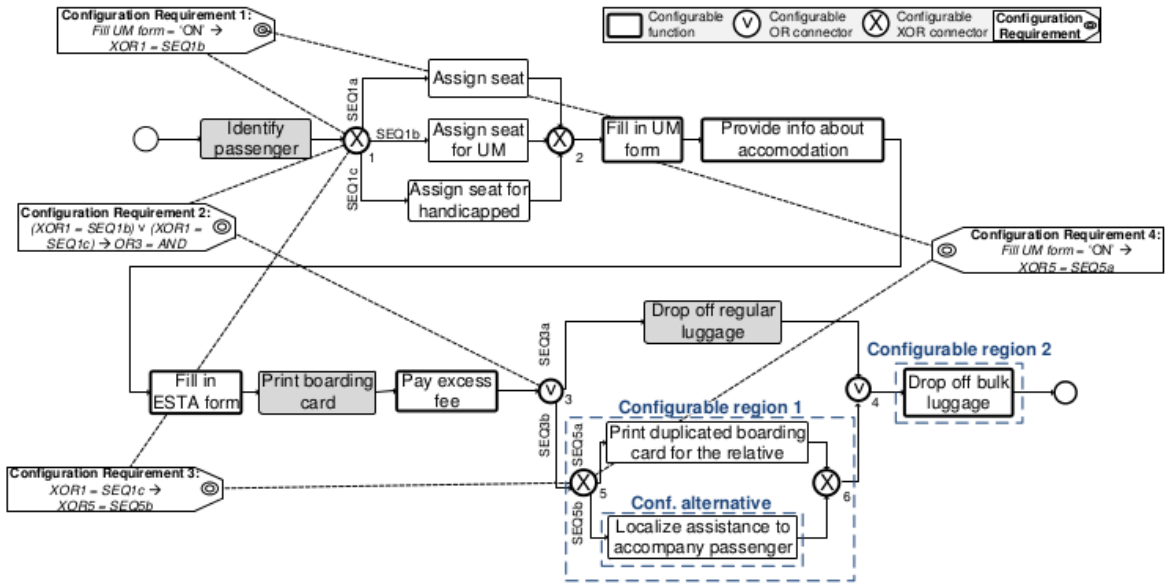


Figura 2.11: Modelo del proceso base de Check-in en C-EPC de [7]

La figura 2.11 ilustra el uso de C-EPC en el contexto de un proceso de *Check-in*. En ésta, se pueden ver los nodos configurables (representados con una línea más gruesa) así como conectores configurables y los requerimientos de configuración. Considerando el contexto de pasajero menor de edad (**UM**) que viaja a **USA** con equipaje **REGULAR** que no excede los **20kg**, las actividades "*Fill in UM form"* (pasajero UM), "*Provide info about accommodation"* y "*Fill in ESTA form"* (destino USA), estarán habilitadas. Por otro lado, las actividades "*Pay excess fee"* y *"Drop off bulk luggage"* (regular que no excede los 20kg) estarán deshabilitadas. En cuanto a los conectores configurables, según las etiquetas de configuración y el contexto definido, podemos decir que: XOR1 deberá tomar el camino SEQ1b, OR3 deberá ser un AND y XOR5 deberá tomar el camino SEQ5a. De esta forma, queda definido un proceso particular a partir del proceso base en C-EPC.

Dentro de las características positivas encontramos la acotada cantidad de elementos la cual facilita la adopción de este enfoque.

El problema de esta propuesta surge cuando la realidad del proceso es realmente compleja, ya que el tamaño de los modelos crece rápidamente así como el etiquetado, convirtiéndose en artefactos muy difíciles de manejar e interpretar.

2.2.2.6 C-BPMN

Siguiendo la línea del enfoque C-EPC, encontramos una variante del mismo para el lenguaje BPMN llamada *Configurable-BPMN* (C-BPMN).
C-BPMN es una extensión a BPMN para soportar el modelado de procesos configurables enfocados en la perspectiva de control de flujo. Define formalmente la semántica de BPMN, la correcta preservación de condiciones y su configuración semántica.

A raíz de que este enfoque se centra en configuraciones semánticas en la dimensión del tiempo, el pasaje entre BPMN y C-BPMN se produce a través de la implementación del ocultamiento y bloqueo. Estas operaciones fundamentales en los procesos de configuración semántica definen la correctitud y preservación de condiciones en la sintaxis C-BPMN; y la correctitud y preservación de restricciones en los modelos C-BPMN de ejecución semántica. Además provee un algoritmo de individualización basado en la definición de modelos C-BPMN configurados y configuraciones semánticas.

Los elementos variables (en este caso, tareas y gateways) son identificados por tener doble línea de borde. Las tareas pueden tomar los mismos valores que en C-EPC: ON, OFF, OPT. Los gateways pueden mantenerse, restringirse a una secuencia o eliminarse [18].

C-BPMN también entra dentro de las propuestas que presentan variabilidad por restricción, y no soporta variabilidad de roles ni datos. Para construir una derivación, es necesario configurar el contexto en base al *feature model* para eliminar partes configurables del modelo.

En la figura 2.12 vemos aplicado el enfoque C-BPMN al caso de estudio del *check-in*. En este caso, observamos los puntos de variación identificados con un borde negro más grueso que los elementos sin variaciones posibles.
Si planteamos la situación de un pasajero minusválido (**HAN**), que viaja a **Europa**, con **exceso de equipaje del tipo REGULAR**; debemos: eliminar las tareas "*Fill in UM Form*", "*Provide information about accommodation*", "*Fill in ESTA Form*", "*Print duplicated boarding card for the relative*"; mantener las tareas "*Pay Excess Fee*" y "*Localize Assistance to Accompany Passenger*".

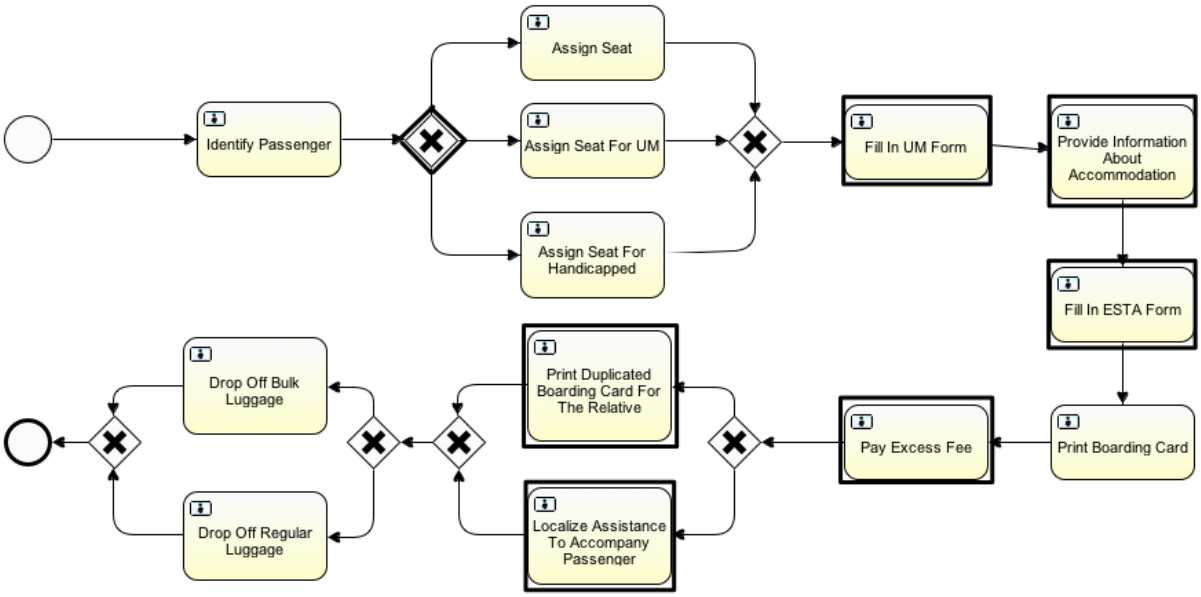


Figura 2.12: Modelo del proceso base de Check-in en C-BPMN

Analizando esta propuesta, encontramos como característica positiva que dada la reducida cantidad de elementos que introduce, su adopción es sencilla. El no contar con variabilidad sobre eventos y roles acota el poder de este enfoque. A su vez, al contar con un diagrama con muchos componentes y sus variantes, nuevamente, produce un diagrama sobrecargado y difícil de interpretar y mantener. En cuanto a herramientas que se ajusten a esta opción, no se ha desarrollado ninguna.

2.2.2.7 BPMNt

BPMNt [6] una extensión conservadora de BPMN 2.0 que tiene como objetivo la creación de un mecanismo de representación de *process tailoring*. Por conservadora, se entiende que es una extensión que no modifica la semántica original de la especificación BPMN 2.0. Dicha extensión introduce elementos en BPMN que capturan la sintaxis y la semántica para apoyar la capacidad de eliminar, reemplazar y agregar elementos del proceso, con base en un mapeo a elementos de SPEM [10] para procesos de software.

Utilizando el mecanismo de extensibilidad de BPMN 2.0, BPMNt incluye una representación específica para el *process tailoring: suppression, local contribution* y *local replacement,* las cuales establecen vínculos entre los elementos del proceso.
La extensión BPMNt está representada por un nuevo concepto, llamado *tailoring*, y sus atributos asociados. Los tres nuevos atributos: *useBaseElement*, *useKind (extension, localReplacement, localContribution)* y *supressedBaseElement* serán utilizados por subclases de

FlowElementsContainer y *FlowElement* ya que permiten la especificación de procesos y subprocesos en una estructura jerárquica, similar a la utilizada en *SPEM*.

- *usedBaseElement:* Un elemento de proceso representado por *FlowElementsContainer* o *FlowElement* debe contener un atributo que represente la asociación con un elemento similar. Por esto, se agrega el atributo que estará relacionado con la clase (elemento de proceso) que será reutilizado.
- *useKind:* Este atributo define el tipo de adaptación que se aplica entre elementos relacionados por el atributo *usedBaseElement.* El dominio del mismo, es representado por el enumerado *ElementUseKind* el cual tiene los mismos valores a los utilizados por SPEM:
  - ***NA***: valor por defecto cuando la relación con *usedBaseElement* no está definida.
  - ***Extension***: define el reuso de la estructura jerárquica basada en una instancia de la clase *FlowElementsContainer.*
  - ***LocalContribution***: define una adición al elemento de proceso relacionado.
  - ***LocalReplacement***: reemplaza el elemento de proceso relacionado junto con su jerarquía asociada.
- supressedBaseElement: Atributo que permite excluir cualquier *BaseElement* de la estructura del proceso. Es usado en el contexto de un proceso reutilizado por el atributo *usedBaseElement*.

Gracias a esto, el enfoque soporta variabilidad casi para todo: Actividades (Tareas y subprocesos), Datos, Gateways, Roles y Eventos. A partir del Proceso Base, y utilizando los diferentes tipos de operaciones de *tailoring* (*NA*, *Extension*, *LocalContribution*, *LocalReplacement*) junto a reglas de interpretación, se logra la derivación para obtener los Procesos Hijos.

La figura 2.13 muestra la representación en BPMNt del modelo base del proceso de *Check-in* junto con la representación del mismo en BPMN. En BPMNt se representan todas las construcciones en una especie de lista asociada al *Check-in,* que es el nombre del proceso principal.

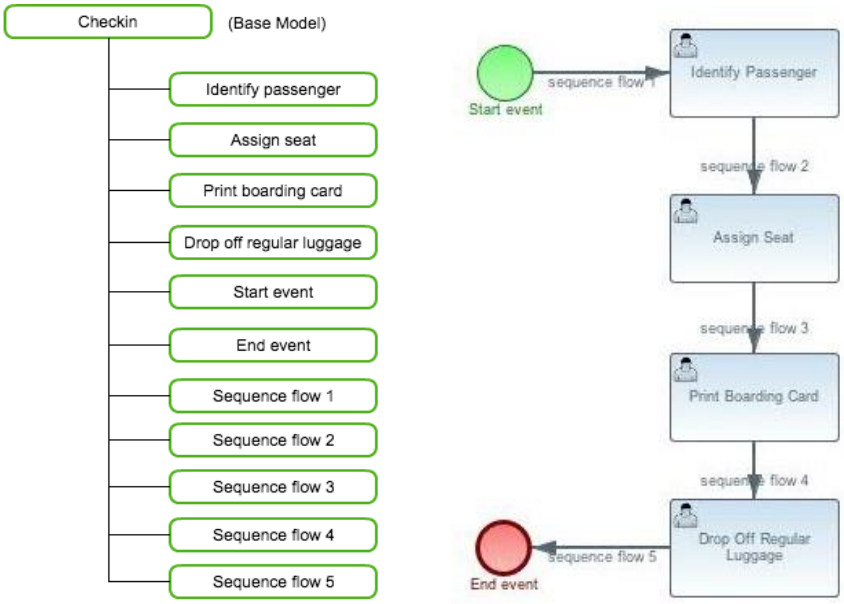


Figura 2.13: Modelo del proceso base de Check-in en BPMNt y su correspondiente en BPMN

Dado que el enfoque presenta un tipo de variabilidad por extensión podemos ver las operaciones necesarias que se deben realizar sobre el modelo base cuando existe un exceso en el equipaje en la figura 2.14 (a). En este caso, se agrega al proceso base una actividad *Pay excess fee* y se reemplaza *Drop off bulk luggage* por *Drop off regular luggage*, además de incluir y quitar los *sequence flows* correspondientes

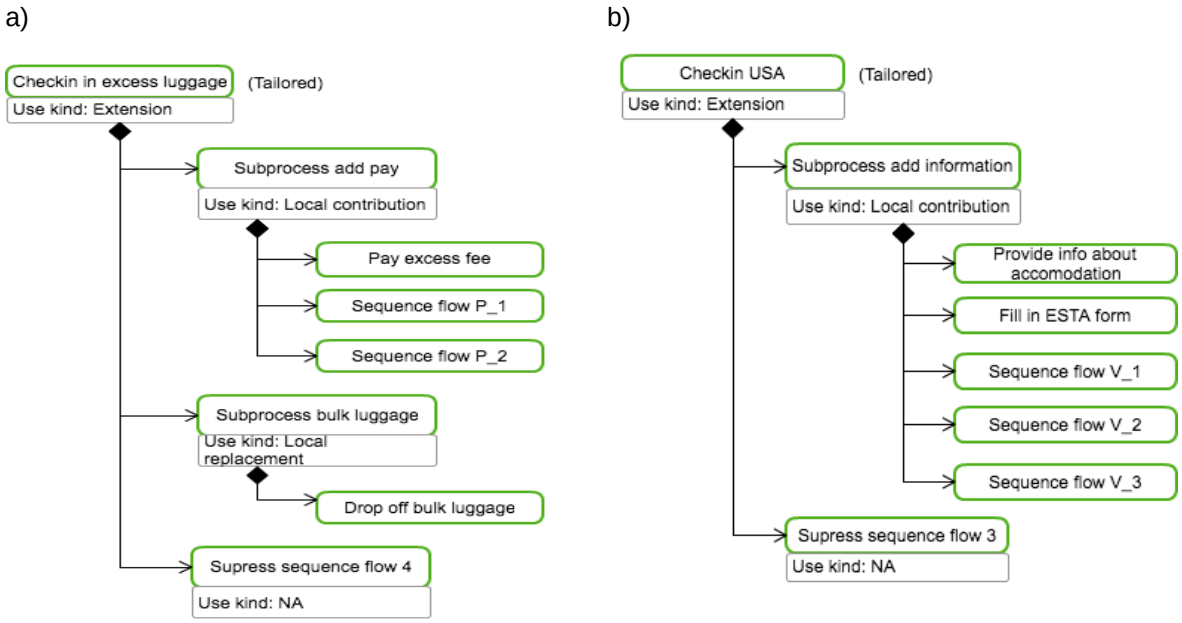


Figura 2.14: Modelo de variación de Check-in en BPMNt para: a) exceso de equipaje, b) USA

Análogamente en la figura 2.14 (b) se pueden ver las operaciones que deben realizarse cuando el destino es USA, lo que implica agregar al proceso base las actividades *Provide info about accommodation* y *Fill in ESTA form* junto con las modificaciones en los *sequence flows.*
Siguiendo con los ejemplos se pueden ver en la figura 2.15 (a) las operaciones necesarias cuando el pasajero es menor de edad, mientras que en la figura 2.15 (b) cuando el pasajero sea minusválido.

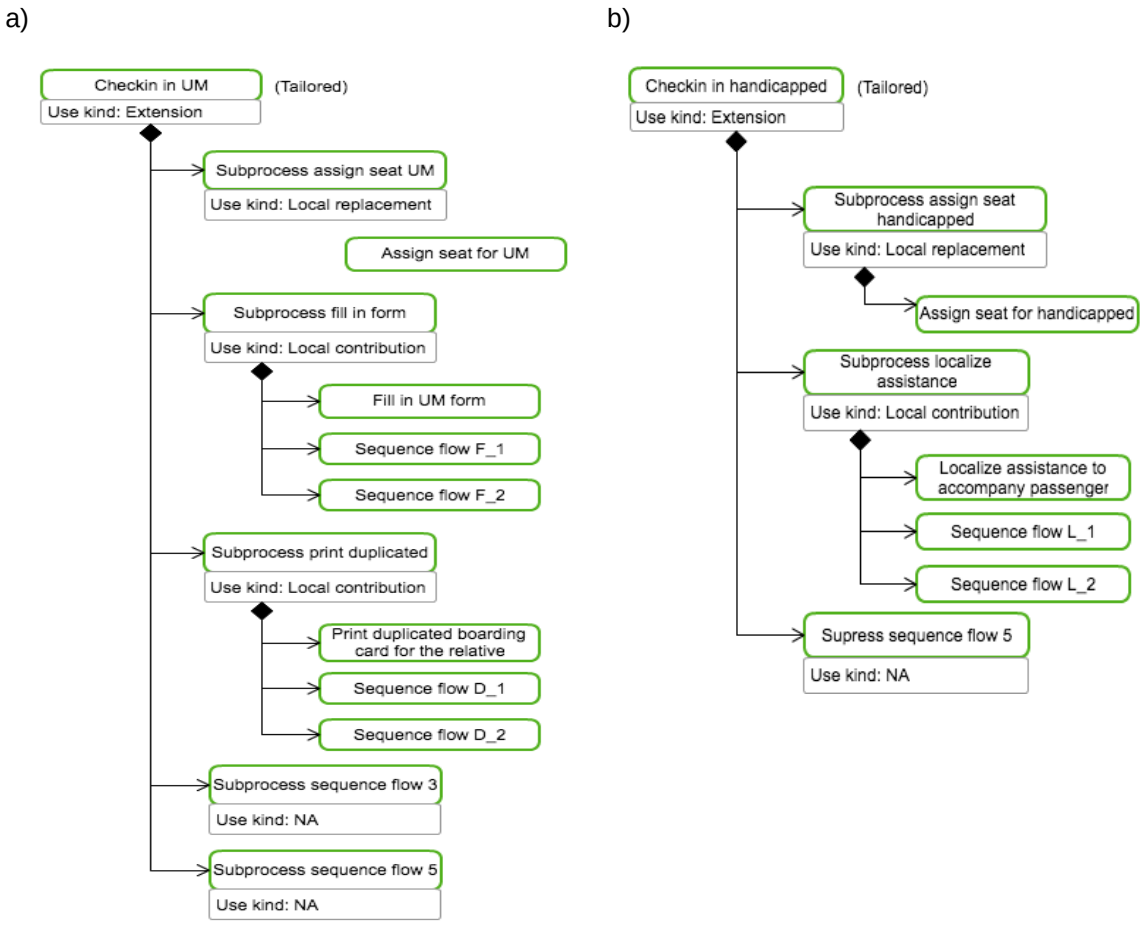


Figura 2.15: Modelo de variación de Check-in en BPMNt para: a) UM, b) HAN

Una gran ventaja del enfoque es que está dirigido a extender BPMN 2.0, lo cual lo hace específico en este lenguaje. Está pensado como una extensión tan genérica que soporta muchos tipos de variabilidad, lo cual lo hace muy potente. También se pudo comprobar que existe una herramienta desarrollada, la misma extiende al proyecto MDT/BPMN2 usando *Eclipse Modeling Framework* (EMF) para producir clases *Java* a partir del modelo BPMN y proveer un editor básico para los mismos. No existe una referencia de descarga, lo único que se hace es extender la definición del *XML schema*.

Algo desfavorable que se pudo notar es la pobre representación gráfica del tailoring, no es del todo clara a simple vista. También, como soporta muchos tipos de variabilidades inevitablemente cuenta con muchas operaciones disponibles; Eso lo complejiza un poco. El enfoque exige siempre partir de un proceso existente (aunque se podría crear uno vacío y hacer todas inserciones), y al no estar todo en principio representado, no se ven las tareas que no están en el modelo original (ni las que se pueden utilizar). No está tan claro de antemano las posibilidades de extensión o las tareas reemplazantes de otra.

*2.2.2.8 BPMN**

BPMN* [19] es una extensión de BPMN 2.0 que modela la variabilidad por restricción. La misma consiste en agregar estereotipos y etiquetas a los elementos de BPMN, que se relacionan directamente con un *Feature Model* asociado. También define un nuevo elemento en el metamodelo para representar la relación entre los puntos de variación y las respectivas variantes.
Observamos que con esta técnica, a partir de un estudio basado en [20][21], que la variabilidad puede darse en los siguientes elementos BPMN 2.0: Actividades (Tareas y subprocesos), Datos, *Gateways, Pools* y Eventos. Luego de determinadas las anotaciones y estereotipos, los mismos se combinan con el *Feature Model* y un contexto de configuración que determina las restricciones sobre él. De allí se puede derivar un nuevo flujo.
A la hora de buscar herramientas, no encontramos referencias de la existencia de alguna.

La figura 2.16 ilustra un proceso de negocio base del proceso de Check-in, aplicando el enfoque BPMN*. Se puede observar cómo siendo un enfoque con variabilidad por restricción, se presentan todas las variantes en el diagrama del proceso base. Contamos con elementos del tipo *Variant* asociados a *VarPoints* en los cuales es necesario escoger al menos una variante para cada uno, y elementos del tipo *Optional* que pueden permanecer en el proceso derivado o no.
A su vez, en la figura 2.17 se encuentra representado el *Feature Model* como un árbol en el cual se encuentran cada una de las combinaciones posibles.

Base Model

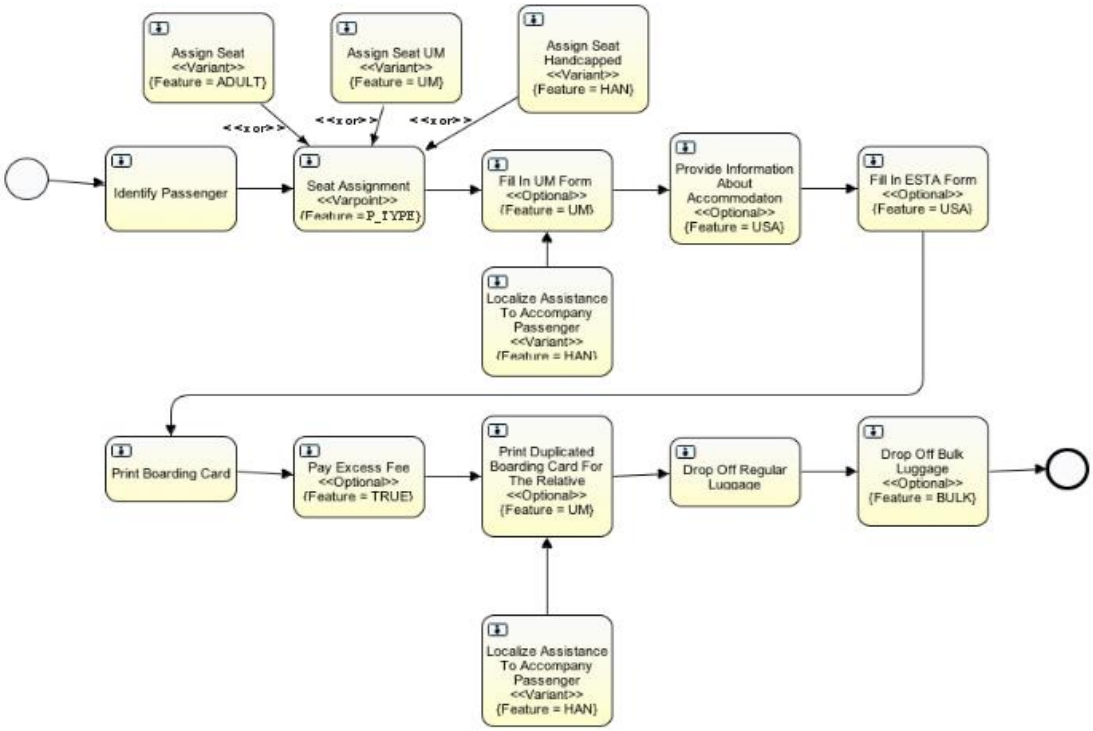


Figura 2.16: Modelo del proceso base de Check-in en BPMN*

Feature Model

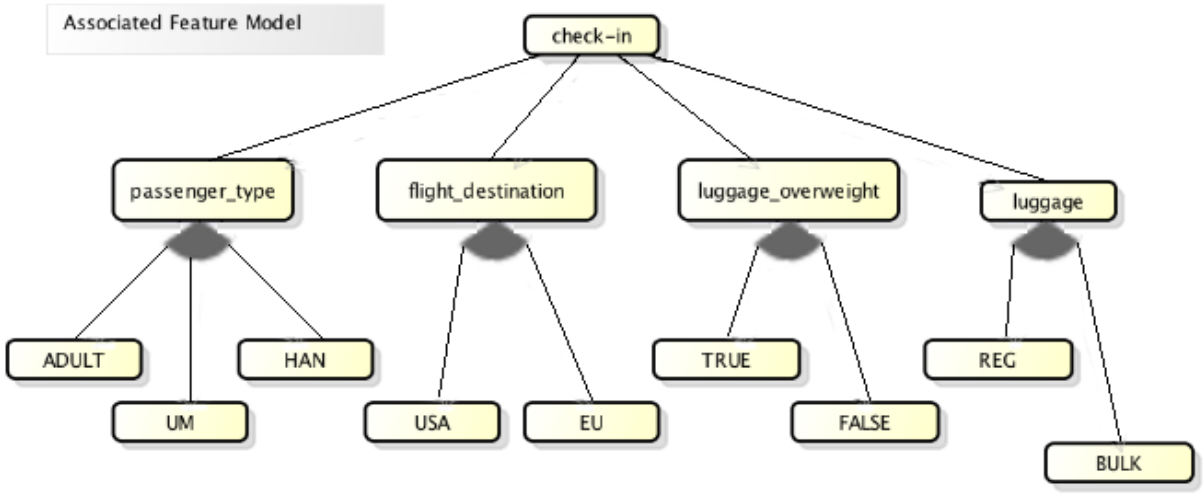


Figura 2.17: Modelo de características de Check-in en BPMN*

En la figura 2.17 puede observarse cómo se representan las posibilidades para cada punto de variación. Este árbol contiene los puntos de variación como nodos intermedios y las variantes como hojas del mismo. Por ejemplo, el VP *"passenger_type"* solo admite los valores ADULT, UM y HAN como posibles variantes. Análogamente funciona para los otros tres VPs y para cada uno de ellos debe seleccionarse una sola de las posibilidades. No está representado aquí, si hubiese restricciones o dependencias entre opciones de diferentes puntos de variación.

La fortaleza más grande de este enfoque radica en su sencillez y su fácil comprensión. Se realizó un estudio empírico para demostrar y comprobar que los errores cometidos son menores a otro enfoque, no analizado en este proyecto, llamado variant-rich BPMN (vrBPMN). Al ser una solución con una cantidad de elementos acotada, se hace fácil su adopción. También es bastante abarcativo con respecto a la variabilidad, y cuenta con varias etiquetas para representar todas las variabilidades de actividades.
Como puntos desfavorables, y atentando un poco contra la facilidad de comprensión, al ser un enfoque por restricción implica que todas las posibilidades tengan que estar expresadas. Esto ante flujos de complejidad alta (o con muchos elementos) puede ser un problema. En el mismo sentido, la utilización de varios componentes de forma separada para la definición puede dificultar su lectura.

2.2.2.9 vSPEM

Dentro de los enfoques por extensión, se analizó otro de esta clase llamado vSPEM. Dado que el mismo extiende el lenguaje SPEM utilizado para el desarrollo de software, no fue considerado en primera instancia en la investigación para la construcción del estado del arte. Para que vSPEM brinde la capacidad de modelar la variabilidad, la extensión enriqueció el metamodelo de SPEM con nuevos elementos que lo permitan. En este sentido, agrega un nuevo paquete llamado *ProcessLineComponents* al metamodelo. El mecanismo de variabilidad en vSPEM está basado en VPs y variantes. Los puntos de variación son agregados al proceso base en donde ocurren las mismas siendo las variantes una implementación específica para ellas, generando por cada una, un proceso final único.

Si realizamos una recorrida desde un nivel más abstracto hacia un nivel más concreto en el metamodelo, encontramos en primer lugar al elemento *ProcLElement*, una abstracción de todos los elementos relacionados a variaciones. Dos especificaciones de *ProcLElement* que son *VarPoint* y *Variant*. Estas dos son clases abstractas, las cuales son heredadas por VPs y variantes concretas. A su vez, estas últimas heredan de elementos de SPEM otorgándoles la capacidad de variar.

La figura 2.18 muestra un ejemplo de como una variante y punto de variación de una actividad son creados (*VActivity* y *VPActivity)*.

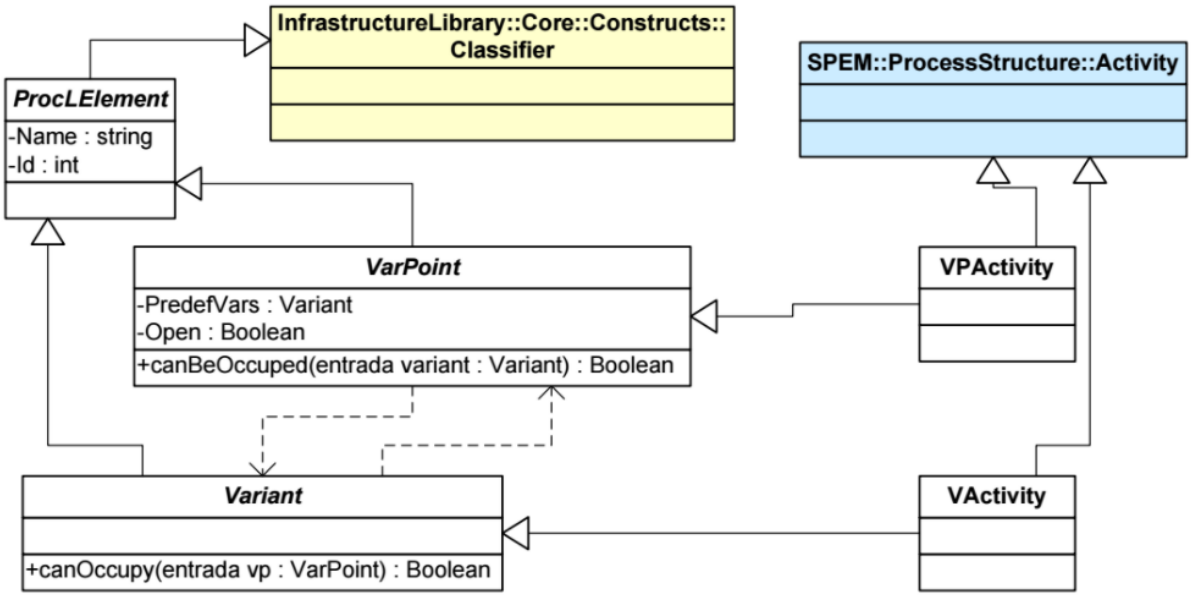


Figura 2.18: Modelo abstracto vSPEM para *Activity* de [22]

Para distinguir diagramas de procesos con variantes de los que no presentan las mismas, y que sean más fáciles de comprender, en vSPEM se definen nuevos iconos basados en los originales de SPEM. En la figura 2.19 se pueden ver los mismos.

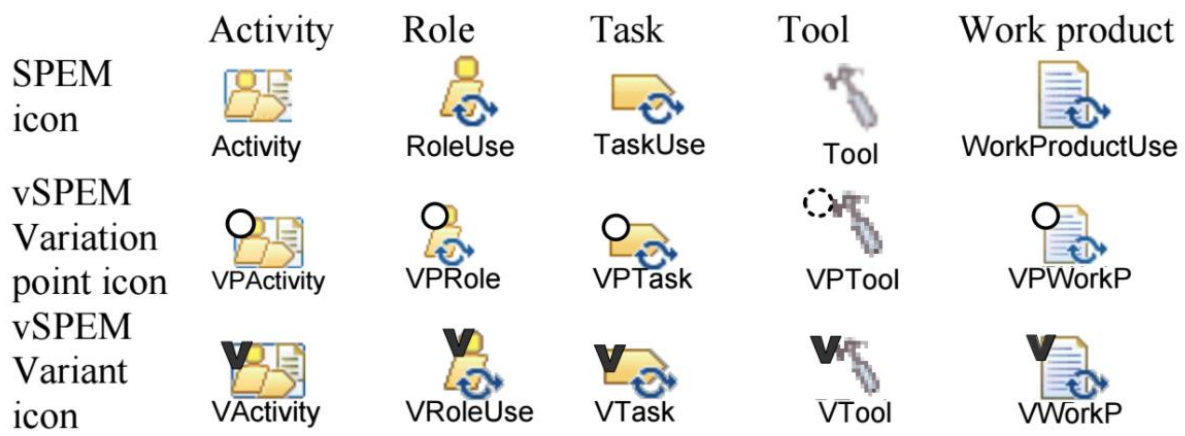


Figura 2.19: Íconos utilizados en vSPEM de [22]

Tiene nuevas representaciones para las cinco construcciones más importantes, a saber: *Activity, Role, Task, Tool* y *Work product.* Las mismas se distinguen claramente por círculo o una “v” en la parte superior izquierda que denota el punto de variación y la variante respectivamente. [22]

2.2.3 Resumen de los enfoques

En la Tabla 1, Tabla 2 y Tabla 3 se presenta un resumen de las propuestas analizadas en esta sección. En la Tabla 1 se muestra el tipo de variabilidad contemplada (por extensión o restricción), los elementos que introduce para el modelado de la variabilidad, los elementos que permite variar y el lenguaje al cual aplica, o no, dicho enfoque. Por otro lado en la Tabla 2 y Tabla 3 se puede ver el proceso de derivación para llegar a un modelo particular, las fortalezas y debilidades, y las herramientas que posee cada enfoque para su utilización.

Tabla 1: Resumen de enfoques - Parte 1

Enfoque	Tipo	Elementos	Variaciones	Lenguaje
Provop	Por extensión	Base model con Adjustment points	Actividades (Tareas y subprocesos	Cualquiera. Es independiente del lenguaje
		Change options con change operations par	NO Datos	
		Option constraint model	NO Gateways	
		Context model	NO Roles	
			NO Eventos	
CVL	Por extensión	Base Model (placements fragments)	Actividades (Tareas y subprocesos	Cualquiera. Es independiente del lenguaje
		Variation Model (replacements fragments)	NO Datos	
		Resolution Model (context conditions)	NO Gateways	
		Context Model	NO Roles	
			NO Eventos	
vBPMN	Por extension	Adaptive Workflow Segments	Actividades	Extensión de BPMN2
		Data-contexts	NO datos	
		Rule-Based Application of Patterns at Runtime with R2ML	NO Gateways	
			NO Roles	
			Eventos	
PESOA	Por restricción	Estereotipos (<<VarPoint>>, <<Variant>>,	Actividades (Tareas y subprocesos	Cualquiera. Es independiente del lenguaje
		<<Default>>, <<Abstract>>,	Datos (artefactos)	
		<<Null>>, <<Optional>>)	NO Gateways	
		Feature Model (alternativas configuración)	NO Roles	
			NO Eventos	
C-EPC	Por restricción	Configuration Requirement (actividades y c	Actividades (Tareas y subprocesos	Dependiente de EPC.
		Configurable task (ON, OFF, OPT)	Datos	
		Configurable OR connector	Gateways	
		Configurable XOR connector	Roles	
			NO Eventos	
C-BPMN	Por restricción	Configuration Requirement (actividades y c	Actividades	Extensión de BPMN2
		Configurable task (ON, OFF, OPT)	NO datos	
		Configurable OR connector	Gateways	
		Configurable XOR connector	NO Roles	
			NO Eventos	
BPMNt	Por extension	Tailoring (useBaseElement, useKind,	Actividades (Tareas y subprocesos	Extensión de BPMN2
		supressedBaseElement)	Datos	
			Gateways	
			Roles	
			Eventos	
BPMN*	Por restricción	Estereotipos (<<VarPoint>>, <<Variant>>,	Actividades (Tareas y subprocesos	Extensión de BPMN2
		<<Mandatory>>, <<Optional>>,	Datos	
		<<Or>>, <<Xor>>)	Gateways	
		Feature Model	Roles	
			Eventos	

Tabla 2: Resumen de enfoques - Parte 2

Enfoque	Derivación	Fortalezas	Debilidades	Herramienta
Provop	A partir del Proceso Base, y utilizando las Change Operations (INSERT, MODIFY, DELETE y MOVE) de acuerdo al Context Model definido, se logra determinar cuáles Change Options se van a utilizar.	No depende del lenguaje utilizado La cantidad de elementos gráficos es acotada, lo cual es de fácil adopción. Es super expresivo Muy modularizado. Se pueden expresar restricciones sobre las combinaciones de opciones.	Al no tener variabilidad sobre los gateways el resultado no es tan expresivo. La utilización de varios componentes puede hacer complejo de entender el modelo y el armado de todas las posibilidades.	Existe un prototipo basado en la herramienta de modelado ARIS Business Architect. El prototipo introduce las facilidades de configuración y administración de variantes de procesos. Este prototipo no se encuentra disponible para ser descargado
CVL	A partir del Base Model con sus placements fragments, el Variation Model es utilizado para reemplazar los placements por replacements. Luego, se utiliza el Resolution Model para determinar bajo qué condiciones se pueden instanciar a los replacements. Y al final, se utiliza el Context Model para apoyar el razonamiento formal de información de contexto.	No depende del lenguaje utilizado Muy modularizado lo cual lo hace ordenado. Se pueden expresar restricciones sobre las combinaciones de opciones.	Al no tener variabilidad sobre los gateways el resultado no es tan expresivo. La utilización de varios componentes puede hacer complejo de entender el modelo y el armado de todas las posibilidades.	Bpmn 2 modeler - http://projects.eclipse.org/projects/soa.bpmn2-modeler Eclipse Plug-in instalación alternativa: http://www.omgwiki.org/variability/doku.php?id=cvl_tool_from_sintef
vBPMN	Enfoca todo desde algo que llama "event-aware workflow variants" y lo modela en base a "Adaptive Workflow Segments" identificados con construcciones del lenguaje, y aplicando lo que define como "Event-Aware Adaptation Patterns", algunas mencionadas son "task insertion" y "task skipping"	El modelo base es bastante claro, con pocas diferencias al BPMN que hace que se entienda más fácilmente. Está muy desarrollado el tema del manejo de eventos Se enfoca mucho en definir un estándar de modelado y por ello cuenta con patrones de diseño. Fácilmente se pueden definir regiones configurables con los corchetes.	Los patrones de diseño no son sencillos de entender, pero tampoco fueron necesarios para la representación de ejemplo. La representación de una tarea opcional con una actividad vacía, para representar que en vez de ser reemplazada la misma puede no existir, no es muy amigable. Hay que aprender un nuevo lenguaje de reglas R2ML para definir las transformaciones de forma correcta. No queda claro dónde se representan las tareas que no están en el modelo base elegido. Por ejemplo las PrintDuplicatedBoeardingCardForTheRelative y LocalizeAssintantAccompanyPassenger que se agregan con INSERT dependiendo la condición	No existe una herramienta desarrollada. Se habla de un prototipo en construcción e ideas para el mismo pero no hay referencias. Existe un prototipo de extensión de JBoss Drool para la parte de reglas que tampoco existen referencias.
PESOA	Se necesita realizar la configuración del contexto en base al feature model para remover partes configurables del modelo.	No depende del lenguaje utilizado La cantidad de elementos es acotada, lo cual es de fácil adopción. El Feature Model es de muy fácil interpretación. Cuenta con varias etiquetas para representar todas las variabilidades de actividades.	No manejar variabilidad sobre eventos puede ser un problema. La utilización de varios componentes puede hacer complejo de entender el modelo. Al ser de tipo "por restricción" implica que todas las posibilidades tengan que estar expresadas y complejiza el diagrama.	PESOA fue realizado como un plug-in de eclipse que soporta la creación de proceso de modelos configurables. Es un editor gráfico que permite representar regiones configurables incluyendo la configuración de alternativas. Tampoco existen referencias

Tabla 3: Resumen de enfoques - Parte 3

Enfoque	Derivación	Fortalezas	Debilidades	Herramienta
C-EPC	Derivar los conectores configurables, después derivar las funciones y aplicar una reducción al gráfico.	La cantidad de elementos es acotada, lo cual es de fácil adopción.	No manejar variabilidad sobre eventos puede ser un problema. La utilización de varios componentes puede hacer complejo de entender el modelo. Al ser de tipo "por restricción" implica que todas las posibilidades tengan que estar expresadas y complejiza el diagrama. La notación con etiquetas es un poco entreverada y difícil de seguir.	Existe un toolset Synergia http://www.processconfiguration.com/download.html que soporta la creacion de un modelo de proceso configurable usando un editor gráfico. También existe un validador (C-EPC Validator) que chequea el cumplimiento de los requerimientos y guías de configuración. http://www.mendling.com/EPML/C-EPC-Validator.xsl
C-BPMN	Derivar los conectores configurables, después derivar las funciones y aplicar una reducción al gráfico.	Aunque está aplicado a BPMN la idea general no depende del lenguaje utilizado La cantidad de elementos es acotada, lo cual es de fácil adopción.	No manejar variabilidad sobre eventos puede ser un problema. La utilización de varios componentes puede hacer complejo de entender el modelo. Al ser de tipo "por restricción" implica que todas las posibilidades tengan que estar expresadas y complejiza el diagrama. La notación con etiquetas es un poco entreverada y difícil de seguir.	No hay referencias.
BPMNt	A partir del Proceso Base, y utilizando los diferentes tipos de operaciones de tailoring (NA, Extension, LocalContribution, LocalReplacement) junto a reglas de interpretacion, se logra obtener los Procesos Hijos	Está dirigido a BPMN 2.0 lo cual lo hace específico. Soporta muchos tipos de variabilidad Existe una herramienta desarrollada que se muestra en el artículo	La representación del "tailoring" no es del todo clara a simple vista. Quizás no es la mejor representación gráfica. Como soporta muchos tipos de variabilidades, cuenta con muchas operaciones disponibles; Lo cual lo complejiza un poco. Siempre se parte de un proceso existente (Aunque podría ser uno vació y hacer todos insert) Al no estar todo en principio representado, no se ven las tareas que no están en el modelo original (Ni las que se pueden utilizar). Es como que ingresan de la nada. No está tan claro de antemano las posibilidades de extensión o las tareas reemplazantes de otra.	Existe una herramienta que extiende al proyecto MDT/BPMN2 usando Eclipse Modeling Framework (EMF) para producir clases Java a partir del modelo BPMN y proveer un editor básico para los mismos. No existe una referencia de descarga, lo único que se hace es extender la definición del XML schema
BPMN*	Combina anotaciones con un feature model que con un contexto que determina las reestricciones en el FM, se puede derivar en un nuevo flujo. No queda claro si la derivación es en	Extensión sencilla de BPMN2 La cantidad de elementos es acotada, lo cual es de fácil adopción. El Feature Model es de muy fácil interpretación.	No manejar variabilidad sobre eventos puede ser un problema. La utilización de varios componentes puede hacer complejo de entender el modelo. Al ser de tipo "por restricción"	No existen referencias a ninguna herramienta

2.2.4 Clasificación de enfoques

Luego de analizar los distintos enfoques, hemos decidido agruparlos en distintas categorías según sus características comunes:

1) Primer categoría:
    - Provop
    - CVL
    - vBPMN

Comparten que son por extensión, por ello tienen en común que cuentan con un modelo base. Pero además, cuentan con un *Context Model* y un tipo de representación de reglas a cumplir (*Option Constraint Model, Context Conditions* ó *Rule-Based Application* dependiendo el caso).
Se centran en la variabilidad de las actividades, aunque vBPMN también suma poder configurar una región y tiene soporte para los eventos.
Por lo general se los ve gráficamente sencillos, aunque al contar con varios componentes diferentes, puede ser complejo entender todas las posibles variantes.
Cuentan con alguna forma de definir restricciones y operaciones para la derivación del modelo base.

2) Segunda categoría:
    - PESOA
    - BPMN*

Si bien uno es una restricción de BPMN y el otro puede ser aplicado en principio a cualquier lenguaje; los dos comparten la utilización de etiquetas de tipo estereotipos para identificar las variabilidades en las actividades.
Además, comparten que la definición de restricciones y las derivaciones se realizan con la configuración de un *Feature Model*.
Los dos enfoques son por restricción, lo cual hace que esté todo representado en el modelo, que puede llegar a ser confuso y difícil si es muy complejo, pero a la vez se pueden observar todas las posibilidades en un solo lugar. Para flujos de actividades sencillas quizás sea lo mejor.

3) Tercer categoría:
    - C-EPC
    - C-BPMN

También estas dos son por restricción, pero no se utilizan estereotipos sino que se basa la representación gráfica de la variabilidad en etiquetas. Son un poco invasivas en el modelado en el sentido que se hace difícil de seguir lo que representan, pero las mismas determinan las restricciones y sobre qué componentes causan efecto.

4) Cuarta categoría:
    - BPMNt
    - vSPEM

Los enfoques presentados en la cuarta categoría son enfoques totalmente diferentes al resto. Basados en proceso de software modelados con SPEM. El *tailoring* de BPMNt se realiza con operaciones definidas pero la representación no es con reglas o sentencias sino que gráficamente pero de una forma no muy amigable. También, por el paradigma que utiliza, en el cual a partir de un modelo definido se lo transforma en otro, obliga a definir un proceso base pero no de forma genérica sino un proceso base que sea un caso particular ejecutable.
Por otro lado, en vSPEM se definen nuevos iconos basados en los originales de SPEM lo que hace que sea notoriamente más claro respecto a BPMNt.

Todos los enfoques permitieron representar a su forma el ejemplo del Check-in de un vuelo. Para algunos casos el resultado final fue muy similar y aunque en ocasiones tuvimos que asumir ciertos criterios por no estar bien definidos en el enfoque, en general las representaciones de la variabilidad quedan claras.
Para los modelos por extensión, como era de esperarse, obtuvimos muchos más componentes para poder hacer la representación. Viendo que el caso no era de una complejidad alta, parecería que los enfoques por restricción para este caso son más sencillos de interpretar.

En la Tabla 4 se presenta a modo de resumen para los distintos enfoques investigados el tipo de variabilidad y su forma de representación

Tabla 4: Clasificación de grupos

Clasificación de grupos	**Tipo de variabilidad**	**Representación**
1 (Provop, CVL, vBPMN)	Extensión	Modelo Base + Modelo de Contexto
2 (PESOA, BPMN*)	Restricción	Modelo Base + Feature Model
3 (C-EPC, C-BPMN)	Restricción	Modelo único
4 (BPMNt, vSPEM)	Extensión	BPMNt no tiene rep. gráfica. vSPEM utiliza Modelo Base con VPs + Variantes

3 Extensión de BPMN2 para modelado de la variabilidad

A continuación se presenta la extensión de BPMN propuesta para el manejo de la variabilidad. Se analizan los pros y contras del enfoque elegido como también se explica por qué se considera BPMNext una extensión que se basa en vSPEM. Se muestran los aspectos principales que se permiten modelar así como también los requerimientos que se decidieron abarcar y la representación gráfica elegida junto con la solución integral. En un apartado se puede ver también en más detalle la extensión del metamodelo que se realizó.

3.1 Enfoque seleccionado como base

Luego del análisis realizado de los diferentes enfoques existentes para representar la variabilidad, y observando las ventajas y desventajas de los mismos, en términos de tipificación, nos pareció mucho más adecuado el enfoque por extensión. Si bien puede volverse complejo por dividirse en varios elementos de representación, tiene la ventaja de ser más claro y con un mayor poder de reutilización.
El enfoque vSPEM fue al que le encontramos mayores ventajas, pero a su vez se tomaron algunas características de los otros enfoques. Gráficamente nos resultó sumamente claro, y si bien sus construcciones no son las mismas ya que no se basa en BPMN, se intentó llevar la idea hacia este lenguaje.
En definitiva, el enfoque que se seleccionó como base para nuestra extensión fue vSPEM.

3.2 Modelado de la variabilidad con BPMNext

La variabilidad es un concepto tan grande que es aplicable a casi cualquier construcción de BPMN. Querer abarcar todos los casos posibles puede ser, además de una ardua tarea, algo de muchos años de trabajo. En nuestro caso, se acotó la extensión a dos de los aspectos más utilizados, buscando construir un modelo que nos permita manejar la variabilidad de actividades (tanto *Tasks* como *SubProcess*) y roles.

Para lo referente a actividades, se buscó alcanzar una implementación lo más genérica posible. Por ello, la misma da una flexibilidad de modelado muy potente y un gran abanico de posibilidades. Se

permite colocar como punto de variación una tarea o un subproceso, y los mismos pueden ser reemplazados indistintamente por cualquier tipo de tarea (*Manual*, *Service*, *Script*, etc) o subproceso. En el reemplazo por otro subproceso, optamos por colocar siempre el subproceso de forma embebida. Como trabajo a futuro, podría ofrecerse elegir de qué forma se quiere hacer este reemplazo e implementar las diferentes variantes: como está hecho,un subproceso embebido, o con un llamado a una actividad aparte (*Call Activity*).

Otra aspecto modelado, es el de contar con la capacidad de cambiar el rol de una actividad marcada como punto de variación, cambiarla por un rol nuevo o en su defecto dejarla donde ya se encuentra. Para este caso, lo que se modeló es en realidad la variabilidad de la relación entre un punto de variación y el *Lane* al que pertenece. Esto da la capacidad de redefinir quién es el responsable de realizar determinada actividad o conjunto de actividades. En medio del proceso de modelado de este punto, llegamos a la conclusión que definir como punto de variación un *Lane* por sí solo carece de sentido y por sí mismo no aporta una funcionalidad relevante, más allá de la posibilidad de poder renombrar el *Lane* involucrado. Por ello, se dejó de lado esto y se modeló la variabilidad de una actividad en un *Lane* dado. Eso otorga un aporte mucho más significativo y permite también -aunque quizás de una forma un poco más complicada- renombrar un *Lane* si se requiere.

Modelar también la posibilidad de eliminar un punto de variación nos pareció fundamental. Normalmente en un proceso base a partir del cual se quiere obtener una variante, como el mismo representa una familia de procesos, existen construcciones sobrantes o que no aplican. Por este motivo el modelado de la eliminación es otro de los aspectos que fue necesario llevar adelante. Como se acotó a modelar principalmente actividades, se llevó adelante el modelado de la eliminación de las mismas. Para ello, el trabajo se basó en la limitación de que los puntos de variación de actividades contasen con únicamente una transición de entrada y una de salida. De ésta forma, la desaparición del punto de variación determina que esas dos transiciones se conviertan en una sola uniendo “virtualmente” la salida de una a la entrada de la otra.

Algo no menor, es que se modeló de manera relativamente sencilla la posibilidad de definir la variabilidad de manera recursiva en términos de subprocesos. Es decir, al reemplazar una actividad del proceso base por un subproceso determinado se permite que éste también contenga puntos de variación. Estos nuevos puntos de variación deberán ser definidos y podrán contener nuevos subprocesos con más puntos de variación. Esto ocurre de manera recursiva hasta contar las definiciones de todos los puntos involucrados.

Una vez realizado el proceso de generación de la variante, se obtiene un único proceso BPMN con todas las construcciones elegidas previamente, junto con una definición de las mismas.

3.3 Extensión del metamodelo

Para lograr la extensión del metamodelo de BPMN, lo primero que se realizó fue un análisis de la definición del lenguaje. El mismo, si lo observamos desde el punto de vista de componentes y lo tomamos como un componente solo, fue extendido por dos módulos más: El módulo de puntos de variación y el de variantes. A su vez, éstos dos están directamente relacionados con clases genéricas que representan la variabilidad: Classifier y VElement.

Las mismas las utilizamos como interfaces de marca, con ellas marcamos todos los objetos que son puntos de variabilidad y variantes. Además, cada uno de estos nuevos módulos dependen directamente de BPMN, ya que cada nueva construcción está relacionada con la construcción de BPMN que está extendiendo. Un módulo agrupa las extensiones de puntos de variación y otro agrupa las variantes.

El modelo de la extensión realizada se presenta en la figura 3.1.

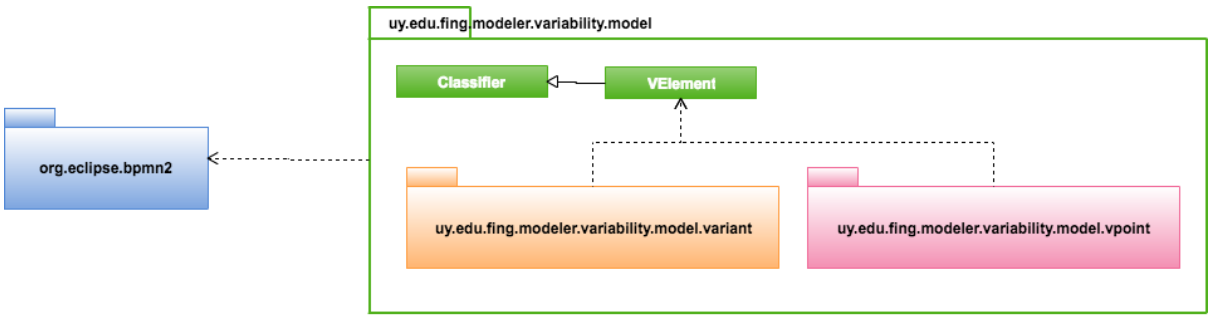


Figura 3.1: Modelo de la extensión desarrollada

El modelo de BPMN se puede ver en la figura 3.2 de forma simplificada. Están representadas solamente las construcciones involucradas en la extensión y algunas de las relaciones entre ellas. En BPMN, de las construcciones que se extendieron, todas parten de *BaseElement* y de allí se desprenden dos implementaciones: *Lane* y *FlowElement*. La primera no es un punto de variación directamente pero se muestra la interrelación con las actividades y las variantes que pueden llegar a participar en un proceso de generación de un nuevo proceso.

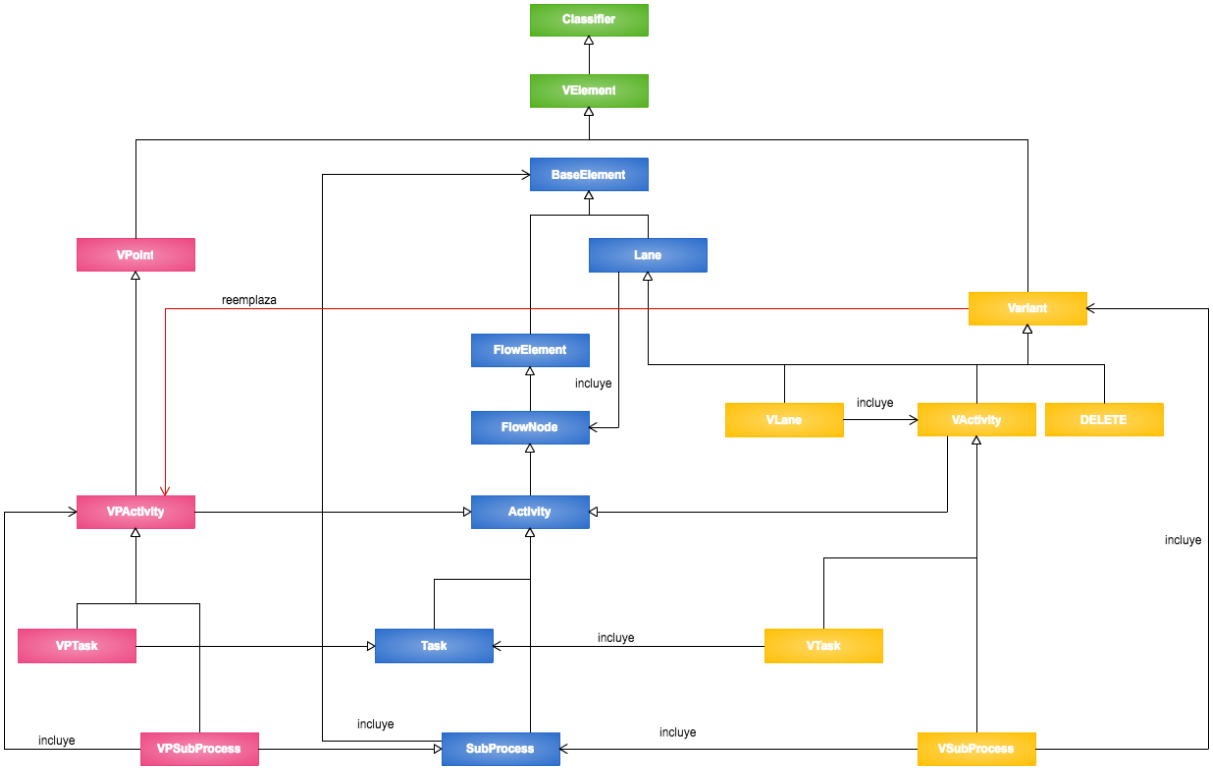


Figura 3.2: Modelo BPMN simplificado

El módulo de puntos de variación (Color rosado en la figura 3.2) está representado por las clases *VPoint*, *VPActivity*, *VPTask* y *VPSubProcess*. La primera es la representación general de un punto de variación, no tiene relación directa con ninguna construcción de BPMN en particular por lo cual esa asociación queda dada en niveles más específicos, pero de todas formas extiende directamente de *VElement* que representa a todas las clases involucradas en la extensión que se construyó. *VPTask* y *VPSubProcess* son las representaciones de puntos de variabilidad de *Task* y *SubProcess* respectivamente, y *VPActivity* es el concepto general y abstracto que agrupa las dos. Se aplicó el patrón *Composite* para poder reflejar la recursividad, donde un *VPSubProcess* puede a su vez contener más puntos de variación y los mismos son *VPActivity*.

Del lado de las variantes (Color amarillo en la figura 3.2) se puede ver que en parte ese módulo es un espejo del módulo de puntos de variación. *Variant* es el análogo a *VPoint* y lo mismo sucede con *VLane*, *VActivity*, *VSubProcess* y *VTask*. Las diferencias fundamentales radican en que cualquier variante (*Variant*) puede ser reemplazo de un punto de variación. Aquí se expresa la flexibilidad del modelo, por el cual por ejemplo, una *VPTask* puede asociarse a una variante de tipo tarea (*VTask*), ser reemplazada por un subproceso (*VSubProcess*), ser cambiado de *Lane* (*VLane*) o en su defecto ser borrada (*DELETE*). Esto aplica para *VPSubProcess* también, por ello se modeló la relación con *VPActivity*. Si la variante es un *VLane*, el mismo puede contener algún *VActivity* que también será reemplazo del punto de variación correspondiente. En caso de no tener, solo se haría el reemplazo al Lane correspondiente o a un nuevo *Lane* si el mismo no existe previamente. Otra diferencia que se

puede observar, es la opción de borrado como un *DELETE*. Ésta variante puede aplicarse a cualquier punto de variación también

Si a futuro se quisiera continuar extendiendo este modelo, simplemente se necesitará seguir añadiendo más construcciones de forma análoga a lo que se hizo. Parecería relativamente sencillo, agregar lo mismo para *Gateways* por ejemplo.

En el capítulo 5 se profundiza aún más en la utilización del modelado y en la notación específica con la realización de un caso de estudio.

3.4 Selección de requerimientos

Para comenzar a desarrollar la solución, lo primero que se hizo fue analizar los posibles puntos de variación que podían existir, y a la vez seleccionar un subconjunto de ellos para definirlos como alcance. Los elegidos fueron aquellos que representan y permiten realizar la mayoría de las funcionalidades, y en definitiva generar a partir de una familia de procesos la mayor cantidad de variantes posibles. Por ello, es que el trabajo se basó en los casos de uso donde dado un proceso base con al menos un punto de variabilidad, para cada uno de ellos se deberían poder realizar las siguientes operaciones:

- **Sustituir el punto de variación *VPActivity* por una variante *VActivity* (**Poder realizar esto resulta ser algo básico que se debió soportar). Dado que *Activity* es abstracta, buscamos sus especificaciones y se definió que fundamentalmente se deberían soportar las sustituciones de la siguiente forma:
    - Sustituir el punto de variación *VPTask* por una variante *VTask* (donde *VTask* puede ser cualquier tipo de *Task* de BPMN (*Task*, *Service*, *Manual*, etc)).
    - Sustituir el punto de variación *VPTask* por una variante *VSubProcess (que no contenga puntos de variación).*
    - Sustituir el punto de variación *VPSubProcess* por una variante *VSubProcess (que no contenga puntos de variación).*
    - Sustituir el punto de variación *VPSubProcess* por una variante *VTask.*
- **Sustituir el punto de variación *VPActivity por una variante VLane***. Más específicamente los casos a considerados fueron:
    - Sustituir el punto de variación *VPTask* por una variante *VLane* formada por un Lane ya existente, y que contenga una *VTask* (donde *VTask* puede ser cualquier tipo de *Task* de BPMN (*Task*, *Manual*, *Service*, etc)).
    - Sustituir el punto de variación *VPTask* por una variante *VLane* formada por un *Lane* ya existente, y que contenga un *VSubProcess (que no contenga puntos de variación).*
    - Sustituir el punto de variación *VPTask* por una variante *VLane* formada por un *Lane* nuevo, y que contenga una *VTask* dentro (donde *VTask* puede ser cualquier tipo de *Task* de BPMN (*Task*, *Manual*, *Service*, etc)).

    - Sustituir el punto de variación *VPTask* por una variante *VLane* formada por un *Lane* nuevo, y que contenga un *VSubProcess (que no contenga puntos de variación).*
    - Aplicar los cuatro puntos anteriores pero partiendo de un *VPSubProcess* en vez de un *VTask.*
- **Eliminar una *VPActivity.***
    - Eliminar una *VTask*
    - Eliminar un *VSubProcess*
- **Recursividad sobre *VPSubProcess:*** Se trata de permitir la posibilidad de reemplazar una *VPActivity* por un *VSubProcess* el cual contenga dentro puntos de variación. Estos puntos tendrán que ser especificados seleccionando las variantes correspondientes para poder realizar la deseada derivación. Este escenario recursivo se puede dar en la cantidad de niveles que sean necesarios de acuerdo a lo que se necesite para modelar correctamente el problema en cuestión.

3.5 Solución propuesta

Luego de definidos los requerimientos, se necesitó plantearse de qué forma era posible construir una solución que abarcara todos los casos, que pudiese ser de fácil entendimiento y que resultara usable. Para ello, se separó la problemática en dos frentes principales como abordaje de una solución integral.

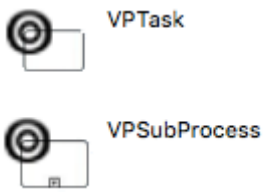


Figura 3.3: Representación de *Variation Points* de *Activities*

En primera instancia, se vio sumamente necesario poder expresar los puntos de variación de forma sencilla, comprensible, expresiva y esquemática. Basados en los trabajos estudiados, las fortalezas más grandes en éste aspecto surgían al contar con la posibilidad de tener una representación gráfica dentro del mismo BPMN, una extensión del lenguaje con construcciones específicas para graficar la variabilidad, que representen claramente lo que se quiere expresar.

Así surgió la necesidad de extender BPMN con nuevas construcciones, y la opción en la que preferimos basarnos fue la de *vSPEM*, debido a que nos resultó ser la que cumplía mejor con los aspectos que se buscaron. La solución de *vSPEM* coloca un símbolo que diferencia la construcción en cuestión como punto de variación. Nuestra solución se basa en lo mismo, pero se adaptó la propuesta hacia el lenguaje BPMN. Como se muestra en la figura 3.3, se necesitó expresar la

variabilidad en *VPTask* y *VPSubProcess* y se realizó con un doble círculo en la esquina superior izquierda de cada construcción.

En segunda instancia, se vio necesaria la creación de un mecanismo que permitiese dado un proceso base con puntos de variación y un conjunto de variantes para cada caso, seleccionar las mismas y permitir realizar la derivación correspondiente dando como resultado final un nuevo BPMN estandar que represente la variante elegida. Ese mecanismo debería ser a partir de un flujo con la representación presentada anteriormente, y con un enfoque sencillo y práctico. Que permitiese crear diferentes derivaciones de forma rápida y repetir el proceso cuantas veces se desee.

Otro apunte que surgió durante el proceso de elaboración de la propuesta, fue explorar la posibilidad de brindar un entorno integrado que cumpla con los dos frentes. Un solo lugar, una sola herramienta, donde se pueda realizar el modelado de la variabilidad y a su vez, generar las derivaciones.

4 Diseño e implementación de solución

Este capítulo intenta mostrar parte del trabajo de investigación realizado para llegar a las conclusiones de diseño e implementación que también se presentan aquí. Se cuenta el proceso de descarte de las diferentes opciones de implementación encontradas hasta llegar a la elegida. Se abarca en un apartado el diseño de la solución y en otro cómo este fue implementado. También se habla de las limitaciones conocidas con las que se terminó el trabajo y se muestra un conjunto de pruebas realizadas de diferentes escenarios, que funcionaron de forma satisfactoria.

4.1 Investigación y diseño

Se comenzó buscando herramientas de software en el mercado que sirvieran para modelar procesos en el lenguaje BPMN. Además, se buscó que en lo posible fuesen de código abierto para facilitar su comprensión y extensión, y que implemente el lenguaje BPMN estándar sin agregados o extensiones propias. En el medio de la investigación notamos que en la mayoría de las alternativas eran o herramientas puramente gráficas -de diseño- o plugins del *IDE Eclipse*. La segunda opción resultó mucho más interesante debido que extendiendo alguna de ellas se podría dar una herramienta integral, mucho más armoniosa que aportase a los procesos de desarrollo.

Se indagó entonces en los plugins de *Eclipse* del mercado, y se encontró que muchas opciones eran intentos de creación de herramientas de modelado pero que quedaron por el camino, desactualizadas. Las dos mejores propuestas a nuestro entender, debido a que están aún en proceso de mejora y mantenimiento con cierto soporte y respaldo, fueron las conocidas como *BPMN2 Modeler* [23] y *Activiti Designer* [24].

Se comenzó con la investigación de *Activiti*, estudiando sus posibilidades y módulos: La misma pareció en un principio bastante completa y flexible. Avanzando en un análisis más profundo nos encontramos con dos problemas: El primero fue que si bien se cuenta mucha información y foros sobre *Activiti*, el código fuente está demasiado desordenado. Cuenta con una cantidad enorme de módulos, y la relación entre los mismos está manejada de forma bastante desprolija de manera interna. Esto nos imposibilitó encontrar las versiones de los módulos que se debían extender para lograr nuestro cometido. El segundo problema que encontramos, por motivos poco claros los desarrolladores de *Activiti* decidieron no utilizar más internamente la implementación estándar de

BPMN 2.0, el metamodelo *Eclipse BPMN 2.0 EMF* [25] el cual es parte del proyecto *Model Development Tools* [26] *(MDT)*, quien fue originalmente desarrollado por miembros del equipo encargado de definir *OMG BPMN 2.0* [5], sustituyendo la misma por una implementación propia con modificaciones. Principalmente por estos dos motivos, la posibilidad de utilizar *Activiti* comenzó a perder fuerza.

Luego, se comenzó a evaluar *BPMN2 Modeler* [27]. Este plugin desde hace un tiempo comenzó a ser parte del mismo Eclipse y no un plugin complementario. Indagando más sobre el mismo, vimos que respeta totalmente el estándar BPMN 2.0 porque utiliza el el metamodelo *Eclipse BPMN 2.0 EMF.* Otra ventaja no menor, es que cuenta con un mecanismo de extensión sumamente sencillo y flexible, para extender las representaciones gráficas, para extender el modelo estándar o para partir de un modelo propio totalmente diferente.

Por estos motivos se seleccionó *BPMN2 Modeler* como herramienta para trabajar en este proyecto y cumplir con el objetivo de extender el lenguaje BPMN soportando variabilidad.

Por otra parte, se buscó la creación de una herramienta integral y en ese sentido, resultó atractiva la posibilidad que desde el mismo *Eclipse IDE* se pudiesen realizar las derivaciones de los modelos. En ese camino se investigó la factibilidad de desarrollar dentro del mismo plugin de extensión de *BPMN2 Modeler*, acciones que permitieran generar algunas pantallas de asistencia (*Wizard*) para trabajar sobre un modelo base y sus variantes, pudiendo configurar una determinada derivación y ejecutando la misma con el objetivo de obtener un nuevo proceso como resultado final. En este sentido, se modeló una capa "*Core*" con la lógica de transformación para cada caso, y la misma es utilizada por la capa "*UI*" que es la que implementa el *Wizard*.

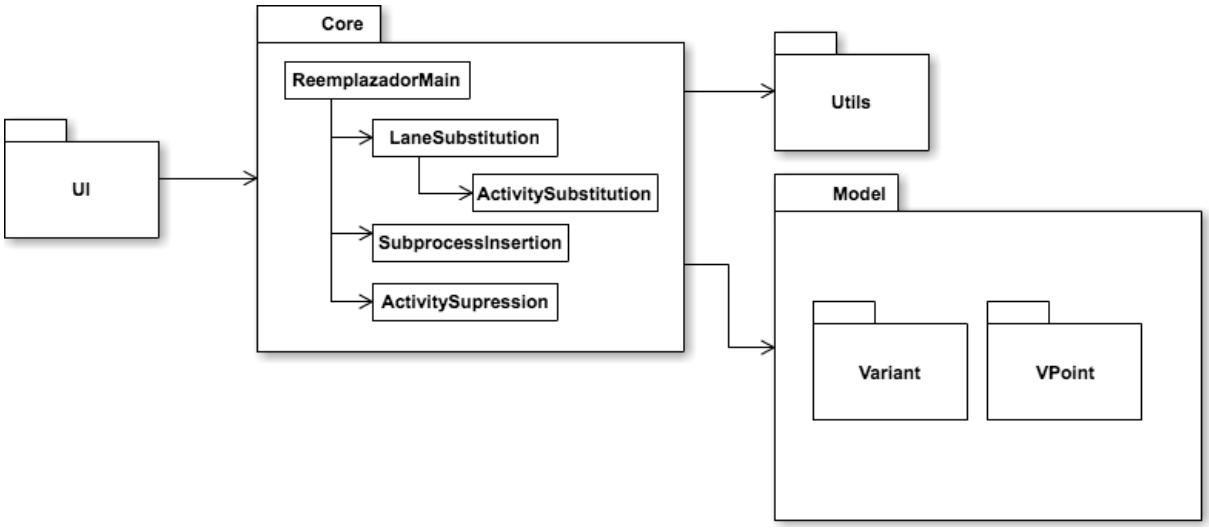


Figura 4.1: Diseño de la solución

También se cuenta con un "*Model*" que contiene todas las clases necesarias para la extensión de BPMN, un "*Utils*" con algunas utilidades como la implementación con *Yaoqiang* [28] para la

regeneración de las coordenadas visuales luego de la derivación y otras clases propias -y necesarias- de todo plugin de *Eclipse*.

4.2 Implementación

Se extendió *BPMN2 Modeler* llevando adelante los pasos proporcionados por la guía de extensión [29] propuesta por los mismos desarrolladores de la herramienta. La misma indica el procedimiento a seguir para extender el plugin de *Eclipse* desde los puntos de extensión ya definidos, pudiendo extender el modelo BPMN y la representación gráfica. Lo principal que vale la pena mencionar, es que se creó un proyecto de tipo *Plugin project* y la mayor parte de la configuración se encuentra en el archivo de nombre *plugin.xml* como se ve en la figura 4.2.

```
<plugin>
    <extension point="org.eclipse.bpmn2.modeler.runtime">

        <runtime
            class="uy.edu.fing.modeler.variability.VariabilityRuntimeExtension"
            id="uy.edu.fing.modeler.variability"
            name="VariabilityRuntimeExtension">
        </runtime>

        <!-- Variation Points -->
        <customTask
            category="VarPoints"
            featureContainer="uy.edu.fing.modeler.variability.container.VariabilityFeatureContainer"
            icon="VPTask.png"
            id="VPTask"
            name="VPTask"
            runtimeId="uy.edu.fing.modeler.variability"
            type="Task">
                <property name="variability" value="VPTask" />
        </customTask>
        <customTask
            category="VarPoints"
            featureContainer="uy.edu.fing.modeler.variability.container.VariabilityFeatureContainer"
            icon="VPSubProcess.png"
            id="VPSubProcess"
            name="VPSubProcess"
            runtimeId="uy.edu.fing.modeler.variability"
            type="SubProcess">
                <property name="variability" value="VPSubProcess" />
        </customTask>
```

Figura 4.2: Configuración en plugin.xml

En este se definen los aspectos más importantes como ser:

extension point: Determina que se está extendiendo BPMN2 Modeler

customTask: Es la etiqueta para definir las extensiones propias de las construcciones BPMN. Lo importante aquí es el atributo *type* que determina que artefacto de BPMN se está extendiendo. Para este trabajo, se extendieron *Task* y *SubProcess*. A los mismos se le agregaron una nueva propiedad *variability* y se les definió un nuevo ícono para ser utilizado en la interfaz gráfica de la herramienta.

Con esto, y sumado a unas clases que utilizan los valores definidos en el archivo de configuración, la extensión gráfica quedó resuelta.

Por otra parte, se amplió la extensión construyendo la lógica para tomar los diagramas realizados con puntos de variabilidad definidos y realizar la derivación. Se agregó en el mismo XML de configuración, una sección de extensión del menú contextual de *Eclipse* con una opción para iniciar el *Wizard*. Para lograr esto se configuró como se puede observar en la figura 4.3 y figura 4.4.

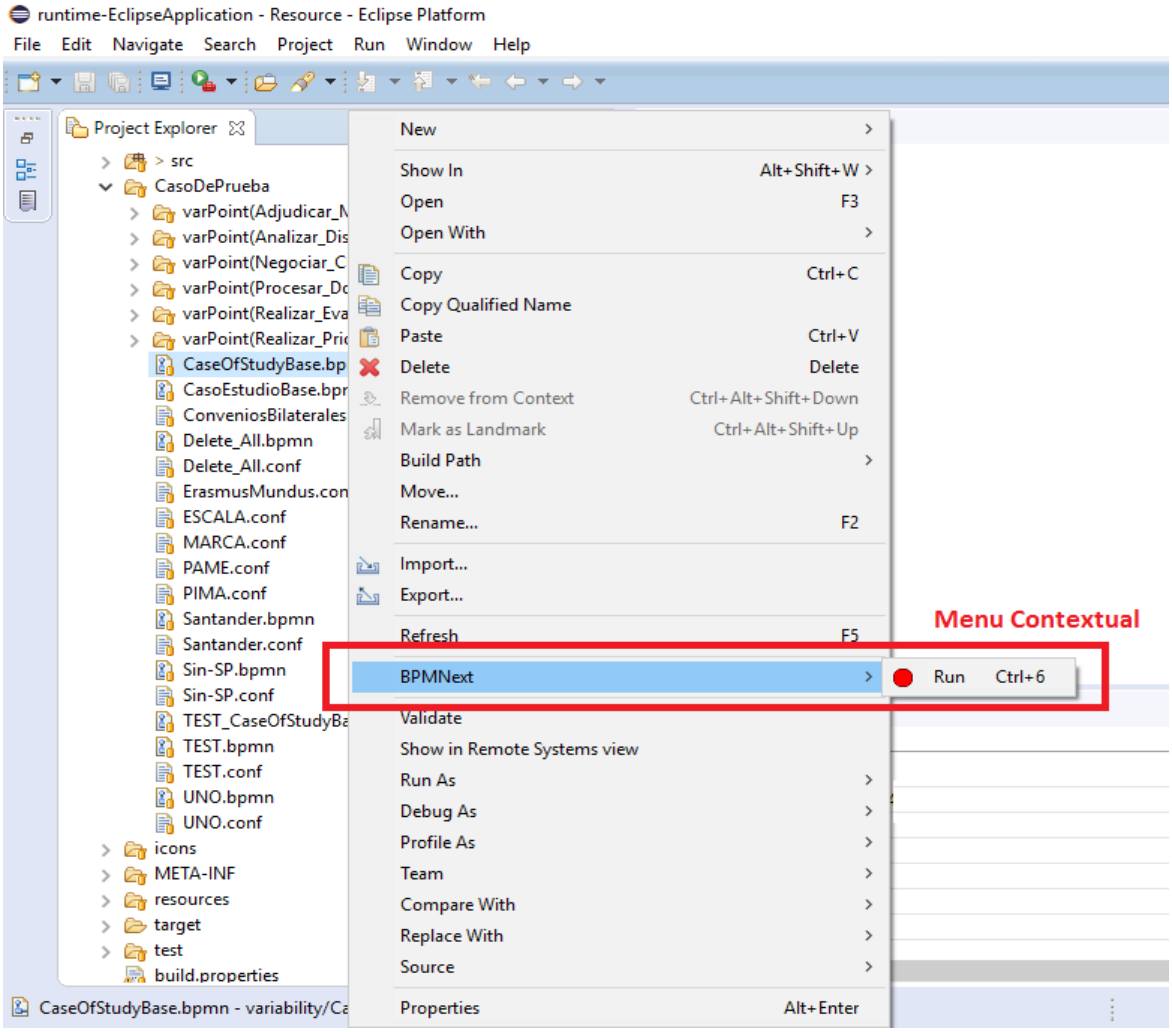


Figura 4.3: Configuración del menú contextual

```
<extension point="org.eclipse.ui.commands">
    <category id="variability.commands.category" name="Substitution Category" />
    <command categoryId="variability.commands.category" id="variability.commands.substitutionCommand" name="Substitution Command" />
</extension>

<extension point="org.eclipse.ui.handlers">
    <handler class="uy.edu.fing.modeler.variability.ui.VariabilityPlugIn" commandId="variability.commands.substitutionCommand" />
</extension>


<extension point="org.eclipse.core.expressions.definitions">
    <definition id="uy.edu.fing.modeler.variability.bpmnFile">
        <iterate ifEmpty="true">
            <adapt type="org.eclipse.core.resources.IFile">
                <test property="org.eclipse.core.resources.name" value="*.bpmn"/>
            </adapt>
        </iterate>
    </definition>
</extension>

<extension point="org.eclipse.ui.menus">
    <menuContribution locationURI="popup:org.eclipse.ui.popup.any?before=additions">
        <menu id="bpmnSubMenu" label="BPMNext">
            <command
                commandId="variability.commands.substitutionCommand"
                label="Run"
                icon="icons/red_dot.gif"
                style="push">
            </command>
            <visibleWhen checkEnabled="false">
                <or>
                    <with variable="activeMenuSelection">
                        <reference definitionId="uy.edu.fing.modeler.variability.bpmnFile"/>
                    </with>
                    <with variable="activeMenuEditorInput">
                        <reference definitionId="uy.edu.fing.modeler.variability.bpmnFile"/>
                    </with>
                </or>
            </visibleWhen>
        </menu>
    </menuContribution>
</extension>
```

Figura 4.4: Configuración del menú contextual

Esta configuración genera que desde un archivo BPMN, realizando un click derecho se despliegue un menú para ejecutar el inicio del *Wizard*.

El *Wizard* se implementó como parte del plugin también, y el mismo proporciona las opciones para definir de qué forma se van a reemplazar los puntos. Éste, luego de proporcionar las opciones y que el usuario las selecciona, toma esos datos y ejecuta la lógica *Core* de la aplicación donde se realizan todas las operaciones necesarias para llegar al resultado final. El *Core* manipula los archivos *XML* detectando las partes clave del mismo y reemplazando esa sección por la sección del archivo *XML* de la variante que corresponda.

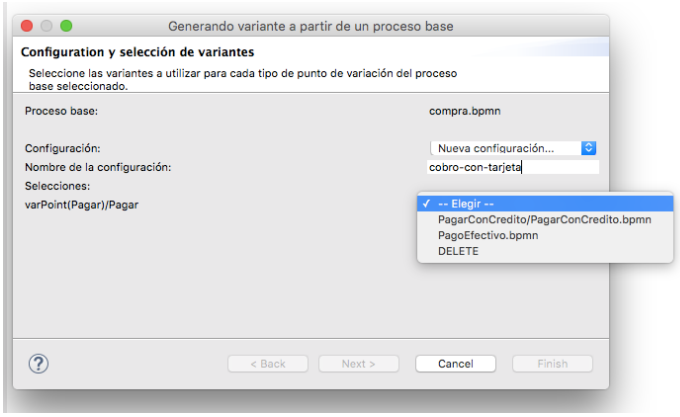


Figura 4.5: Wizard - Pantalla de configuración

En el caso del reemplazo de una tarea por otra, solamente se realiza la sustitución de las propiedades que definen el tipo de tarea y el nombre. Para la sustitución por un subproceso ya no fue tan sencillo; aquí se tiene que remover la sección que define el punto de variabilidad, generar la construcción del lenguaje que determina el subproceso extendido e insertar todas las construcciones (flujo entero) que existen en el archivo que contiene la variante. Luego, conectar a este nuevo subproceso las flechas correspondientes que salían y entraban al punto de variabilidad.

Para la eliminación de un punto de variabilidad la conexión de las flechas del flujo es un tanto similar pero al quitar el punto de variabilidad hay que reemplazar cada flecha saliente y cada flecha entrante, por una nueva que recorra desde donde sale la entrante hasta donde llega la saliente y después eliminar las que tomamos como base.

También se implementó el caso de modificar el *Lane* de un punto de variabilidad y para ello fue necesario manipular el *XML* de tal forma de extraer la información dentro del *Lane* origen y eliminarla, para luego buscar e insertar esta misma dentro del *Lane* destino. Para el caso de que el *Lane* destino no existiese, se implementó su creación.

Toda esta lógica mencionada perteneciente al *Core*, está distribuida entre las clases *LaneSubstitution, ActivitySupression, ActivitySubstitution* y *SubprocessInsertion* orquestada por el *ReemplazadorMain* que se encarga de tomar todos los archivos, la configuración seleccionada e ir armando el *XML* final luego de cada proceso realizado de manera consecutiva y de forma recursiva cuando un subproceso tiene puntos de variabilidad. Al final, con el *XML* resultado obtenido se implementó un paso más: Para lograr una mejor visualización del mismo y que no se superpongan gráficamente las diferentes construcciones, se continúa manipulando el *XML* eliminando toda la sección del archivo correspondiente a la representación gráfica y regenerándola con el uso de una librería externa llamada *Yaoqiang*. Con esto se logró que la derivación fuese mucho más agradable a la vista.

El resultado final es un archivo *XML* con el *BPMN* generado y un archivo de configuración de extensión *.conf* que contiene las variantes seleccionadas para cada punto de variación. Esto permite que desde el *Wizard* se puedan retomar estas configuraciones, modificarlas de ser necesario y generar nuevas derivaciones a partir de algo configurado previamente.

4.3 Limitaciones

Se intentó abarcar la mayor cantidad de funcionalidad, pero a la hora de dejar algo de lado se optó por centrarse en lo que se consideró de mayor importancia. De todas formas, quedaron situaciones no contempladas y oportunidades de mejora.

Una de las mismas es la posibilidad de elegir que al seleccionar como variante un subproceso, el mismo pueda ser considerado e insertado en el proceso base como un proceso colapsado, como un proceso expandido o un *Call Activity*. Se llegó a implementar solamente la inserción como un proceso expandido y por ello no se da la opción de variar esto.

Otro mejora posible, es que al mover construcciones del lenguaje entre *Lanes*, detectar si existen *Lanes* vacíos y eliminarlos ya que pierden sentido.
Cuando intervienen *Lanes* en el proceso de derivación, la librería utilizada *Yaoqiang* no funciona correctamente dejando colapsado cosas de más. Eso es otro punto de mejora a considerar.
Se podría dar más potencial al *Wizard*, de forma que permita seleccionar los puntos de variación o las variabilidades dejándolas en el directorio correspondiente. Hoy toda esta gestión se realiza por fuera y podría hacerse desde allí dentro.
La lógica y el *Wizard* se podrían mejorar internamente para dar mayor información cuando por algún motivo el procesamiento falla. Hoy el error es muy genérico. Esto permitiría al usuario poder corregir con mayor facilidad los problemas.

4.4 Pruebas realizadas

Con la intención de verificar el correcto funcionamiento del plugin desarrollado, se definió un conjunto de **19** tests intentando representar todas las posibles combinaciones de acciones a realizar sobre los distintos elementos de un proceso base con puntos de variación. Para cada uno, se crearon archivos de configuración con las distintas elecciones que deseamos verificar y el resultado esperado.

En la tabla 5 se detalla un conjunto de tests realizados con sus respectivas configuraciones. Por otro lado, también se realizaron tests que verificaron el correcto funcionamiento de la solución en procesos con recurrencia en su variabilidad. Esto quiere decir, procesos que contienen VP cuyas variantes también cuentan con sus propios VP; requiriendo de esta forma, la elección de variantes un nivel más abajo.

Tabla 5: Tests realizados

ID Test	*Proceso Base*	*Configuración*	*Proceso Correcto*	*VP1*	*VP2*	*VP3*
				Task_1	**Task_2**	**SubProcess_1**
1	p1.bpmn	c1.conf	r1.bpmn	*DELETE*	-	-
2		c2.conf	r2.bpmn	*A*	-	-
3		c3.conf	r3.bpmn	*Sub_A*	-	-
4	p2.bpmn	c4.conf	r4.bpmn	*DELETE*	*DELETE*	-
5		c5.conf	r5.bpmn	*DELETE*	*B*	-
6		c6.conf	r6.bpmn	*DELETE*	*Sub_B*	-
7		c7.conf	r7.bpmn	*A*	*B*	-
8		c8.conf	r8.bpmn	*A*	*Sub_B*	-
9	p3.bpmn	c9.conf	r9.bpmn	-	-	*DELETE*
10		c10.conf	r10.bpmn	-	-	*A*
11		c11.conf	r11.bpmn	-	-	*Sub_A*
12	p4.bpmn	c12.conf	r12.bpmn	*Lane_2*	-	-
13		c13.conf	r13.bpmn	*Lane_3*	-	-
14	p5.bpmn	c14.conf	r14.bpmn	*DELETE*	*Lane_1*	-
15		c15.conf	r15.bpmn	*A*	*Lane_1*	-
16		c16.conf	r16.bpmn	*Lane_2*	*Lane_1*	-
17		c17.conf	r17.bpmn	*Lane_2*	*Sub_B*	-
18		c18.conf	r18.bpmn	*Sub_A*	*Sub_B*	-
19	p6.bpmn	c19.conf	r19.bpmn	-	-	*Lane_2*

5 Caso de estudio

Para plasmar mejor el estudio se utilizó un caso de la realidad en el cual nos basamos para aplicar los conocimientos adquiridos y así se verificó que el enfoque de la variabilidad propuesto es totalmente viable. En este capítulo comienza explicando el caso de estudio, luego mostrando el modelado utilizando BPMN estandar, luego identificando las posibles variantes y representando las mismas en BPMNext. Al final, se presentan algunas derivaciones para comparar el resultado final obtenido.

5.1 Caso de estudio

El caso de estudio se refiere al modelado del proceso “Adjudicación de movilidades de estudiantes de grado”, el cual abarca los siguientes programas de intercambio académico: Becas Santander, Convenios bilaterales, Erasmus Mundus, ESCALA Estudiantil, MARCA, PAME y PIMA. Este modelo fue realizado por el grupo COAL para la Dirección General de Relaciones y Cooperación Internacional (DGRC) de UdelaR. A los efectos de producir un resultado tendiente a la mejora de la gestión relacionada con dichos programas, en lugar de modelar cada programa individualmente, se elaboró un modelo de proceso genérico que pudiese ser tomado como base, que constituye el proceso configurable asociado a la familia de variantes de programas de movilidades de grado. Este modelo, que se muestra en la figura 5.1, destaca los aspectos comunes que se encontraron.

La especificación de un modelo común posee varias ventajas. En particular, este modelo genérico aporta una mejora cualitativa en la gestión de los procesos al hacer disponible una perspectiva horizontal de los programas de intercambio académico (de estudiantes en este caso), que trasciende la visión compartimentada por programa. En este sentido, permite realizar un análisis más global tendiente a la definición de medidas de ejecución del proceso como un todo y a la evaluación de potenciales mejoras que impacten en todos los programas. Además, la independencia en relación a las particularidades de cada programa podría permitir utilizar el modelo como base para la especificación de otros programas de intercambio académico, por ejemplo de docentes o estudiantes de postgrado, utilizando los bloques de actividades opcionales definidos y/o agregando nuevos bloques de actividades posibles en las variantes que permitan llegar a un proceso genérico que modele todas las variantes de movilidad, sean de grado, postgrado o docentes, en el marco o no de proyectos de investigación y/o redes temáticas.

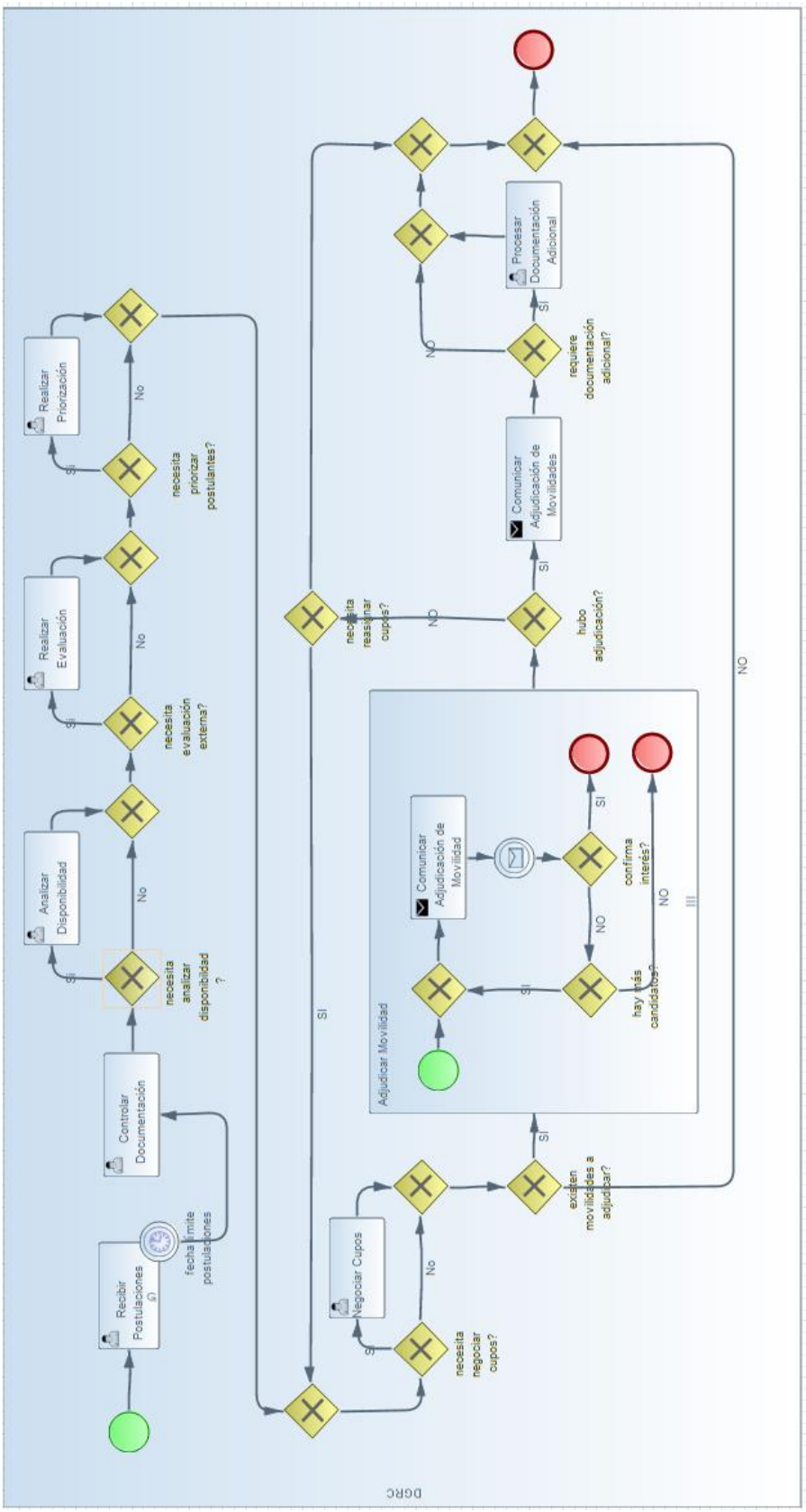


Figura 5.1: Modelo genérico con BPMN

5.2 Modelo genérico con BPMN

El proceso de “Adjudicación de movilidades de estudiantes de grado”, que se observa en la figura 5.1, comprende la recepción de la solicitud de postulaciones a programas de intercambio académico para estudiantes de grado de la Udelar, la evaluación de los postulantes y la adjudicación de movilidades individuales relacionadas con los programas. El proceso finaliza con la presentación de documentación académica para realizar la movilidad, dando comienzo al proceso de ejecución de cada movilidad.

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, 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 realizó cada servicio cuando envía las postulaciones. Esta evaluación determina qué postulantes cumplen con las condiciones mínimas para realizar una movilidad y cuáles no, así como puede establecer un orden inicial entre los postulantes.
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 reevaluación académica, por ejemplo de escolaridad mínima necesaria o de avance en la carrera para aplicar a determinadas universidades. La priorización puede verse afectada por la disponibilidad analizada anteriormente. 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 ver afectada por el análisis de disponibilidad realizado inicialmente. Luego, se

pasa a adjudicar las movilidades definitivas (**subproceso Adjudicar Movilidad** en figura 5.2) lo que implica la comunicación con cada postulante aceptado para recibir su aceptación o rechazo.

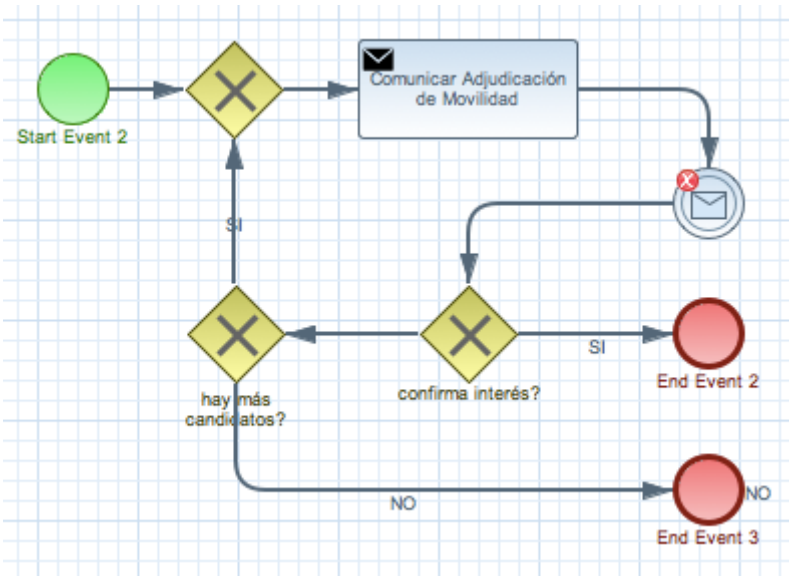


Figura 5.2: Subproceso Adjudicar Movilidad

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.

5.3 Modelo con BPMNext

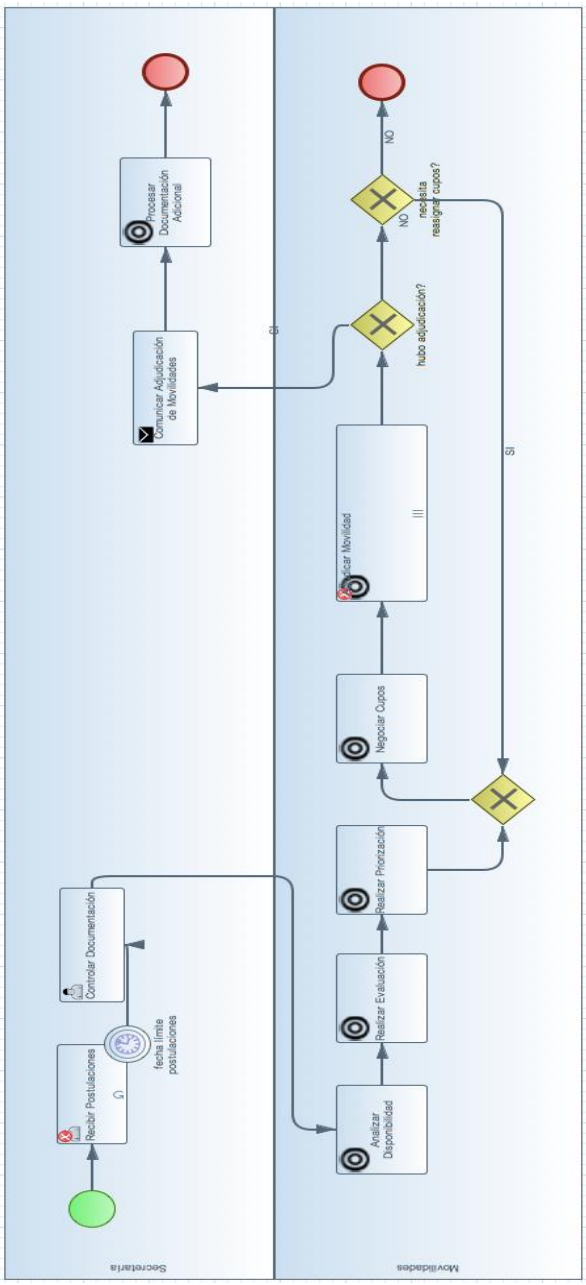


Figura 5.3: Modelo genérico con BPMNext

Con BPMNext se utilizaron las construcciones que permiten manejar la variabilidad. De esta forma, se pudo representar todo el proceso de forma mucho más clara. Las tareas opcionales fueron representadas como puntos de variación y de forma separada se modelaron las variabilidades. Los mismos son: “Analizar disponibilidad”, “Realizar Evaluación”, “Realizar Priorización”, “Negociar Cupos”, “Adjudicar Movilidad” y “Procesar Documentación Adicional”. Así puede verse un modelado mucho más sencillo que antes, donde todo está representado de forma genérica en una familia de procesos con unos cuantos puntos de variación de definir para derivar. Visualmente, es mucho más

fácil detectar aquellos puntos que tienen que definirse en cada variante por una implementación particular o eliminarse en caso de no requerirse. También se puede apreciar que se reduce notablemente la cantidad de compuertas, mejorando la legibilidad del modelo.

Salvo para el caso de “Adjudicar Movilidad” donde el mismo es un subproceso, los demás casos pudieron ser representados sólo con tareas particulares, es decir, un caso sencillo en el cual se reemplaza un punto de variación por una variante que es una tarea concreta. El caso del subproceso es un poco más complejo pero en el caso de estudio existe solo ese o sino queda la opción de borrado del punto de variación. Dada su sencillez creemos que no aporta valor mostrar las variantes en este caso, pero sí por ejemplo el resultado de una derivación particular como puede ser el caso “Santander” donde la configuración es:

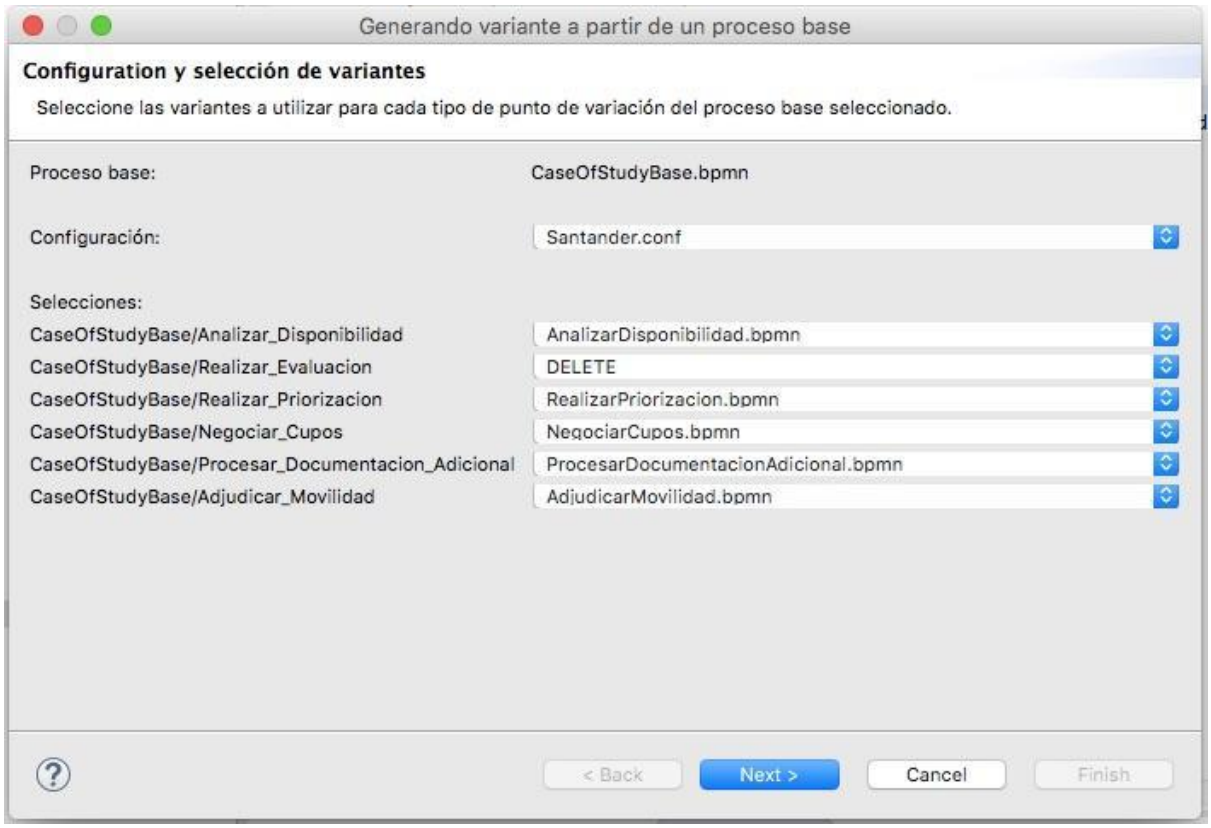


Figura 5.4: Configuración derivación Santander

Y ante esta configuración su derivación obtenida es:

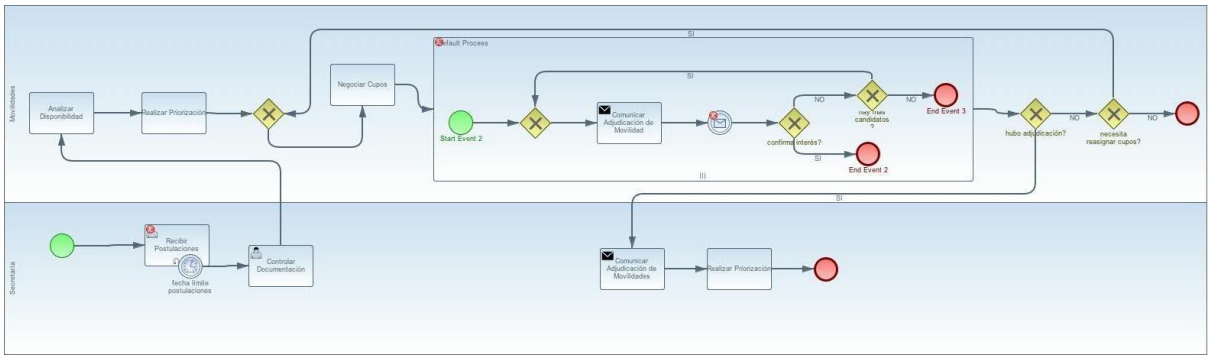


Figura 5.5: Proceso resultado derivación Santander

Y para el caso más sencillo de PIMA, en el cual se eliminan todos los pasos menos la solicitud de documentación adicional , podemos ver lo sencilla y clara que queda la derivación:

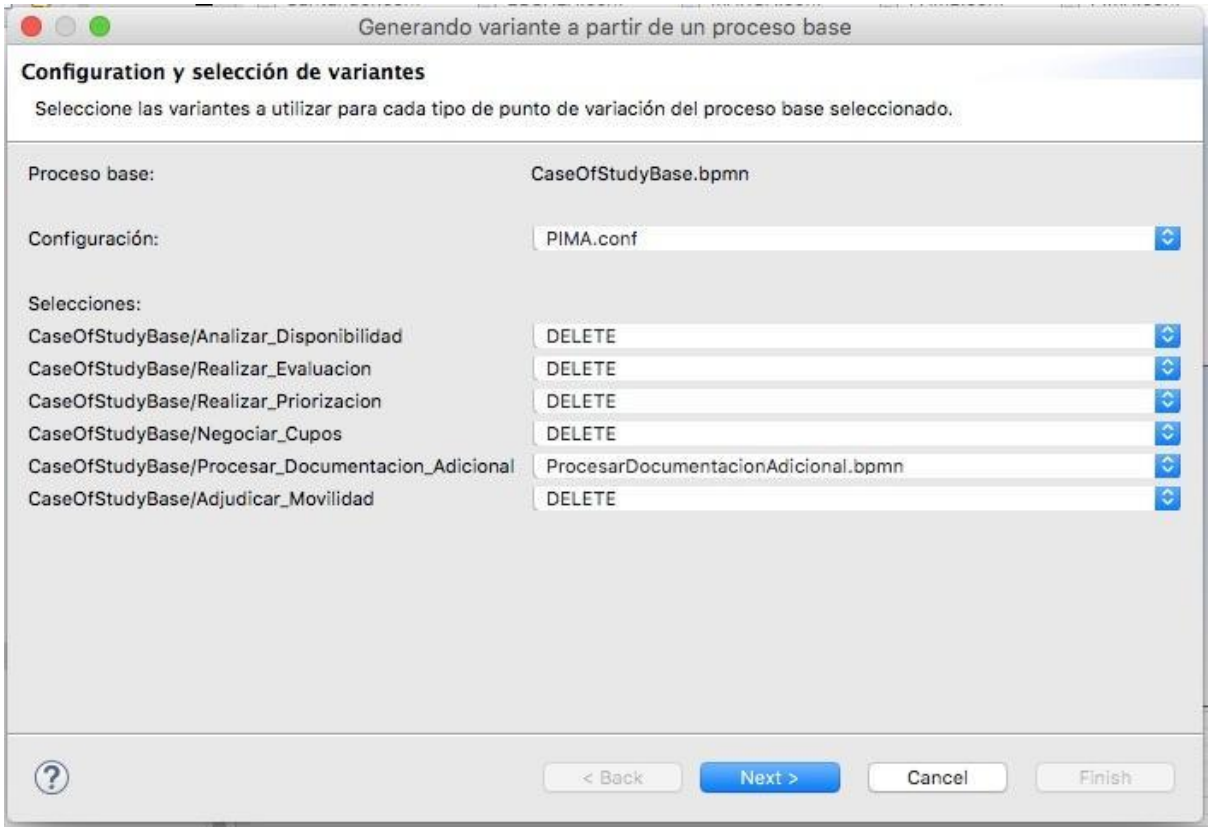


Figura 5.6: Configuración derivación PIMA

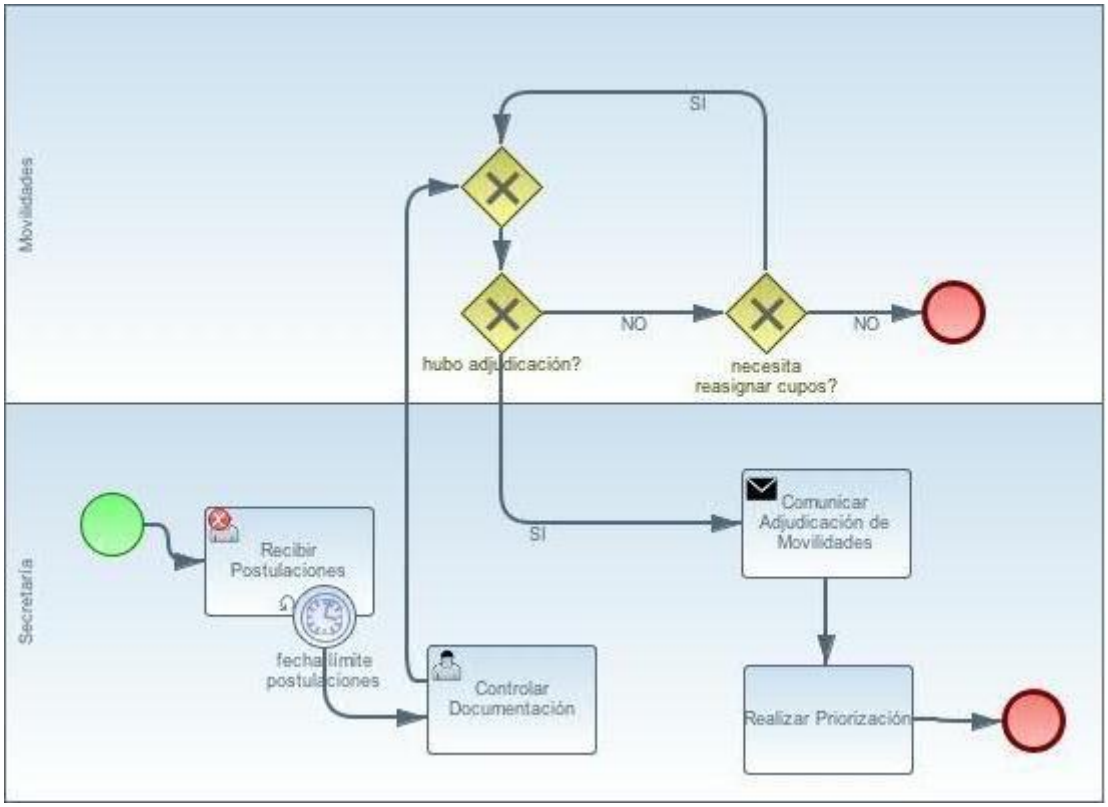


Figura 5.7: Proceso resultado de la derivación PIMA

6 Conclusiones

En este capítulo se analizan las conclusiones del proyecto, y se plantean posibles mejoras a ser realizadas sobre la solución planteada en el proyecto.

6.1 Conclusiones

El principal objetivo de este proyecto era investigar los distintos enfoques existentes para el modelado de la variabilidad en procesos de negocio BPMN 2.0 (OE1). Luego del análisis de cada una (OE2), se debía realizar una adaptación (OE3) y desarrollar un prototipo de herramienta como plugin de Eclipse (OE4).

Es por ello que previo a construir una solución al problema de la variabilidad en BPMN, se analizaron distintas propuestas o enfoques ya existentes.

Dada la complejidad de los diagramas de los enfoques por restricción, decidimos que era más adecuado elaborar una herramienta capaz de modelar la variabilidad por extensión.

Para llegar a resolver esta problemática se investigó distintos plugins de Eclipse que permitieran modelar un proceso utilizando BPMN destacándose: *BPMN2 Modeler* [23] y *Activiti Designer* [24]. *BPMN2 Modeler* [27] vimos que respeta totalmente el estándar BPMN 2.0 y presenta una ventaja no menor, el cual permite extender las representaciones gráficas, para extender el modelo estándar o para partir de un modelo propio totalmente diferente.

Teniendo en cuenta estas consideraciones, y basándonos fuertemente en vSPEM, se elaboró BPMNext haciendo hincapié en las ventajas y desventajas de cada propuesta, extendiendo el modelo de BPMN. Entre los puntos a destacar, se encuentra la independencia entre el proceso base y las variantes asociadas a sus puntos de variación. De esta forma, la incorporación de variantes se puede realizar sin afectar en nada al proceso base ya definido y facilita enormemente el mantenimiento del mismo. A su vez se amplió la extensión construyendo la lógica para tomar los diagramas realizados con puntos de variabilidad definidos y realizar la derivación. Se logró que BPMNext provea de un asistente gráfico en Eclipse que permite la derivación de procesos de forma intuitiva, generando el proceso derivado junto a su configuración para un posterior uso. A su vez, BPMNext permite la variabilidad multinivel, esto quiere decir que una variante puede tener a su vez puntos de variación con sus respectivas variantes, resolviendo su implementación a través de métodos recursivos.

De la misma manera, aplicando BPMNext a un proceso real como el de la “Adjudicación de movilidades de estudiantes de grado” en la UdelaR, demostró la correcta adecuación de la herramienta al problema.

En cuanto a las ventajas que podemos resaltar se encuentran las siguientes:

- Representación de forma sencilla de puntos de variabilidad dentro del modelo base y las variantes en archivos independientes.
- Mejora notoria en la legibilidad, ya que reduce la cantidad de compuertas necesarias para “modelar la variabilidad”

Por otro lado, existen algunas limitaciones que pueden ser mejoradas fácilmente:

- No poder elegir la forma de cómo reemplazar el VP de tipo subproceso (expandido, colapsado o *call activity)*, hoy en día solo se permite insertar la variante como un proceso expandido.
- No detectar Lanes vacíos para su eliminación, luego de mover actividades entre los mismos.
- La librería utilizada Yaoqiang no funciona correctamente cuando intervienen Lanes en el proceso de derivación.
- El manejo de la estructura de archivos necesarios para realizar una derivación debe hacerse por fuera del *Wizard.*
- Manejo de errores presenta poca información al usuario, hoy el error es muy genérico

6.2 Trabajo a futuro

Teniendo en cuenta las limitaciones ya presentadas, mencionaremos una serie de puntos a mejorar en un futuro.

Uno de los aspectos a mejorar es el relacionado al diagrama generado utilizando *Yaoqiang*. Si el proceso generado cuenta con *Lanes*, el proceso derivado y diagrama generado por esta herramienta, mostrará los *Lanes* como rectángulos angostos con todos sus elementos apilados imposibilitando su interpretación. Se debería trabajar en conseguir que el rectángulo sea lo suficientemente ancho como para poder visualizar correctamente el diagrama. A su vez, en el caso que un *Lane* quede sin *Activities* dentro en el proceso generado, sería deseable que el mismo se eliminara.

Por otro lado, sería conveniente posibilitar substituir VP por variantes de subprocesos que puedan modelarse como *Call Activities* o de forma colapsada.

Tal vez uno de los puntos más importantes a mejorar sea la forma de creación de variantes para un VP. Aconsejamos permitir la creación de variantes desde el mismo *Wizard*, ubicando las mismas en la ubicación correcta según la estructura definida en esta propuesta.

7 Referencias

[1] M. Weske. *Business process management: concepts, languages, architectures*. 2da ed. Berlin: Springer-Verlag Berlin Heidelberg, 2012. ISBN 978-3-642-28615-5.

[2] M. La Rosa, W.M.P. van der Aalst, M. Dumas, F.P. Milani. *Business Process Variability Modeling: A Survey*. BPM Center Report BPM-13-16, BPMcenter.org, 2013. http://bpmcenter.org/wp-content/uploads/reports/2013/BPM-13-16.pdf

[3] M. Reichert, S.Rechtenbach, A. Hallerbach, T. Bauer. *Extending a business process modeling tool with process configuration facilities: The Provop demonstrator*. Proceedings of the Business Process Management Demonstration Track (BPMDemos 2009), Ulm, Alemania, 8 de setiembre del 2009.

[4] Grupo COAL. *Modelado de la variabilidad en Procesos de Negocio. Propuesta de proyecto de grado*. Abril 2015.

[5] Object Management Group. *Business Process Model and Notation (BPMN).* Versión 2.0. OMG Specification. Enero 2011. http://www.omg.org/spec/BPMN/2.0/

[6] R. M. Pillat, T. C. Oliveira, P. S. Alencar, D. D. Cowan. *BPMNt: A BPMN extension for specifying software process tailoring.* Information and Software Technology, vol. 57 (enero 2015), pg. 95-115.

[7] C. Ayora, V. Torres, B. Weber, M. Reichert, V. Pelechano. VIVACE: *A framework for the systematic evaluation of variability support in process-aware information systems*. Information and Software Technology, vol. 57 (enero 2015), pg. 248-276.

[8] O. Pedreira, M. Piattini, M. R. Luaces, N. R. Brisaboa. *A systematic review of software process tailoring.* ACM SIGSOFT Software Engineering Notes, vol. 32 (mayo 2007), número 3, pg. 1-6.

[9] W. M. Van der Aalst. *Formalization and verification of event-driven process chains*. Information and Software technology, vol. 41 (1999), número 10, pg. 639-650.

[10] Object Management Group. *Software & Systems Process Engineering Metamodel (SPEM).* Versión 2.0. OMG Specification. Abril 2008. http://www.omg.org/spec/SPEM/2.0/

[11] CVL Submission Team. *Common variability language (CVL)*. OMG revised submission, 2012. http://omgwiki.org/variability/lib/exe/fetch.php?media=cvl-revised-submission.pdf

[12] A. Hallerbach, T. Bauer, M. Reichert. *Managing Process Variants in the Process Lifecycle.* Paper presentado en: 10th International Conference on Enterprise Information Systems (ICEIS'08), Barcelona, España, del 12 al 16 de junio del 2008, pg. 154–161.

[13] C. Ayora, V. Torres, V. Pelechano, G. H. Alférez. *Applying CVL to business process variability management*. Proceedings of the VARiability for You Workshop: Variability Modeling Made Useful for Everyone. Nueva York: ACM, 2012. ISBN: 978-1-4503-1809-9.

[14] M. Döhring, B. Zimmermann. *vBPMN: event-aware workflow variants by weaving BPMN2 and business rules.* 10th International Workshop, BPMDS 2009, and 14th International Conference, EMMSAD 2009, celebrada en CAiSE 2009, Amsterdam, Holanda, del 8 al 9 de junio del 2009, Proceedings.

[15] C. Ayora, G. H. Alférez, V. Torres, V. Pelechano. *Procesos de Negocio Auto-Adaptables al contexto.* VII Jornadas de Ciencias e Ingeniería de Servicios. A Coruña, España, del 5 al 7 de setiembre del 2007.

[16] Synergia. Versión 1.0. http://www.processconfiguration.com/download.html.

[17] *C-EPC Validator*. Versión 1.2. http://www.mendling.com/EPML/C-EPC-Validator.xsl.

[18] H. Zhang, W. Han, C. Ouyang. *Extending BPMN for Configurable Process Modeling.* Moving Integrated Product Development to Service Clouds in the Global Economy. Holanda: IOS Press, 2014, pg. 317-330. ISBN: 978-1-61499-440-4.

[19] M. F. Terenciani, D. M. Paiva, G. Landre, M. I. Cagnin. *BPMN*-A Notation for Representation of Variability in Business Process Towards Supporting Business Process Line Modeling.* Universidad Federal de Mato Grosso del Sur (UFMS), 2015.

[20] G. Gröner, M. BošKović, F. S. Parreiras, D. GašEvić. *Modeling and validation of business process families.* Information Systems, vol. 38 (julio 2013), número 5, pg. 709–726.

[21] A. Schnieders, F. Puhlmann. *Variability Mechanisms in E-Business Process Families*. International Conference on Business Information Systems (BIS 2006). Klagenfurt, Austria, del 25 de agosto al 27 de agosto del 2006.

[22] T. Martínez-Ruiz, F. García, M. Piattini, J. Munch. *Modelling software process variability: an empirical study*. IET software, vol. 5 (2011), número 2, pg. 172-187.

[23] *BPMN2 Modeler Documentation*. 2014. https://www.eclipse.org/bpmn2-modeler/documentation.php.

[24] Activiti. 2005. http://activiti.org/.

[25] Eclipse Modeling Framework (EMF). 2011. https://eclipse.org/modeling/emf/.

[26] Eclipse Model Development Tools (MDT). 2011. https://www.eclipse.org/modeling/mdt/.

[27] Eclipse BPMN2 Modeler. 2011. http://eclipse.org/bpmn2-modeler/.

[28] Yaoqiang BPMN Editor, an Open Source BPMN 2.0 Modeler. 2010. http://bpmn.sourceforge.net/.

[29] Extension Point API, Eclipse Wiki. 2014. https://wiki.eclipse.org/BPMN2-Modeler/ExtensionPoints.