Universidad de la República

Facultad de Ingeniería
Instituto de Computación

Generación automática de variantes para procesos de negocio

Informe de Proyecto de Grado

Estudiantes
Darwin Fernandez
Gonzalo López
Leonel Peña

Tutores
Dr. Ing. Daniel Calegari
Dra. Ing. Andrea Delgado

Año 2018

Resumen

Un Proceso de Negocio (PN) es un conjunto de actividades realizadas en coordinación en un entorno organizacional y técnico, para alcanzar un objetivo del negocio. Los procesos de negocio presentan, en algunos casos, variantes de un proceso general base. Ejemplos son procesos de ventas para productos diferentes, o de contabilidad para distintos países. En la última década han surgido diversas propuestas para tratar la variabilidad de procesos con foco en el modelado de familias de procesos con variantes, evitando modelar cada variante en forma separada lo que implica duplicación y mantenimiento de las partes comunes, una de estas propuestas es Common Variability Language (CVL) la cual propone la definición de variantes mediante un lenguaje independiente de dominio capaz de contener la abstracción de variablidad. Adicionalmente a las dificultades anteriores existe la complejidad de realizar la configuración de estas variantes por lo que son útiles técnicas de Ingeniería Basada en Modelos, del inglés Model Driven Engeneering (MDE), tales como las transformaciones de modelos, las cuales simplifican esta tarea para el usuario.

El objetivo principal de este proyecto es explorar la generación automática de variantes en familias de procesos de negocio utilizando técnicas de MDE, evitando así el proceso manual de configuración de las variantes específicas. Este proyecto continúa proyectos de grado de años anteriores que hicieron un relevamiento extenso sobre propuestas para el modelado de familias de procesos.

Para cumplir con los objetivos del proyecto se analizaron diferentes propuestas de variabilidad, y la posibilidad de aplicar esas propuestas a la variabilidad en PN implementados en Business Process Model and Notation (BPMN). Finalmente se desarrolló una herramienta gráfica que permite realizar la configuración de variantes en procesos BPMN basándose en el modelado de variantes propuesto por una adaptación de CVL, y transformaciones MDE que permiten la generación de la familia de variantes.

Índice

Capítulo 1

Introducción

Un Proceso de Negocio (PN) [1] es un conjunto de actividades realizadas en coordinación en un entorno organizacional y técnico, para alcanzar un objetivo del negocio. Los procesos de negocio presentan, en algunos casos, variantes de un proceso general base. Ejemplos son procesos de ventas para productos diferentes, o de contabilidad para distintos países. En la última década han surgido diversas propuestas para tratar la variabilidad de procesos con foco en el modelado de familias de procesos con variantes, evitando modelar cada variante en forma separada lo que implica duplicación y mantenimiento de las partes comunes.

Estas propuestas pretenden definir una familia de procesos mediante un proceso base común (denominado proceso configurable) más las variantes que cada proceso específico requiere. Para la particularización específica de variantes en general se define algún tipo de regla de operación con parámetros de configuración [2], modelos de características [3], cuestionarios o tablas de decisión [4] para apoyar la derivación de variantes desde el modelo genérico, en base a los parámetros de configuración elegidos en cada caso. Algunos enfoques proveen guías y feedback al usuario para asegurar que las opciones de particularización elegidas resultan en modelos con comportamiento y sintaxis correctas. Sin embargo, no se han encontrado propuestas que realicen la construcción automática de variantes.

El objetivo principal de este proyecto es explorar la generación automática de variantes en familias de procesos de negocio utilizando la técnica de transformación propuesta por Model Driven Engeneering (MDE), evitando así el proceso manual de configuración de las variantes específicas. Continúa el trabajo realizado en proyectos de grado de años anteriores [5,6] que hicieron un relevamiento extenso sobre propuestas para el modelado de familias de procesos.

De los proyectos anteriores se decidió analizar y profundizar su aplicabilidad

para proceso de negocio de los modelos de variantes de (CVL, [7]), leguaje estandar de *Object Management Group* para la especificación y resolución de variabilidad y su integración a herramientas gráficas de modelado.

Los objetivos específicos de este proyecto son:

- Generar un relevamiento del estado del arte en la temática de generación automática de variantes en procesos de negocio
- Definir estrategias para la generación automática de variantes a partir de la aplicación de técnicas MDE
- Integrar las estrategias a un herramienta gráfica de modelado de variantes
- Aplicar las estrategias en casos de estudio para validación de la propuesta

El presente documento se encuentra organizado de la siguiente manera:

- Capítulo 2 - Estado del arte: Conceptos más relevantes para el entendimiento del proyecto, así como las diferentes propuestas consideradas en el desarrollo del mismo.
- Capítulo 3 - Problema planteado: Descripción del problema y desafíos que se intentan resolver.
- Capítulo 4 - Solución propuesta: Especificación detallada de la solución desarrollada así como tecnologías y enfoque utilizado.
- Capítulo 5 - Caso de estudio: Aplicación de la herramienta desarrollada a un caso de particular.
- Capítulo 6 - Conclusiones: Desarrollo de las conclusión y propuestas de trabajos futuros.

Capítulo 2

Estado del arte

2.1. Business Process Management

Business Process Management (BPM) incluye conceptos, métodos y técnicas para soportar el diseño, administración, configuración, ejecución y análisis de los PN en las organizaciones [1]. BPM ofrece un marco para apoyar la definición, el control y la mejora continua de la operación de los PN [8].

2.1.1. Procesos de negocio

Un Proceso de negocio (PN) [1] es un conjunto de actividades realizadas en coordinación en un entorno organizacional y técnico, para alcanzar un objetivo del negocio.

Los procesos de negocio presentan, en algunos casos, variantes de un proceso general base. Ejemplos son procesos de ventas para productos diferentes, o de contabilidad para distintos países.

Existen diversos lenguajes y estándares para el modelado de procesos de negocio, como ser: Diagramas de Actividad de UML, XML Process Definition Language(XPDL) y BPMN, entre otros.

2.1.2. Business Process Model and Notation (BPMN) 2.0

BPMN [9] provee una notación gráfica para la especificación de procesos de negocio independientemente del entorno de implementación.

El objetivo principal es establecer una notación simple capaz de ser comprendida por los usuarios del negocio, así como compleja y precisa para los técnicos que deberán traducir los mismos en componentes de software para su ejecución y monitoreo.

Es un lenguaje estándar de OMG compuesto de elementos que permitirán representar una variedad de situaciones así como también realizar una identificación de los elementos del lenguaje mediante un componente de software.

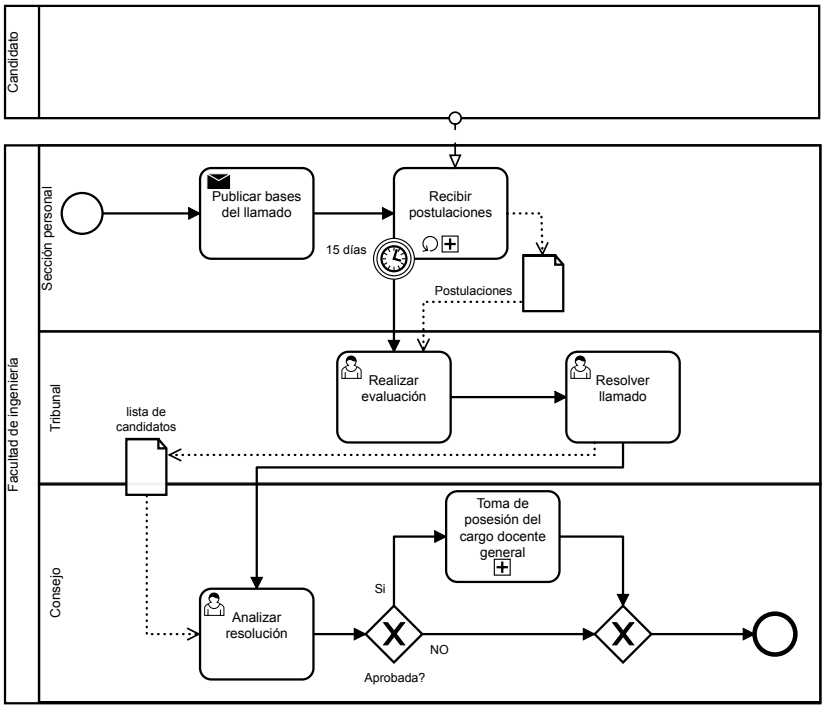
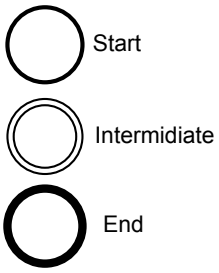
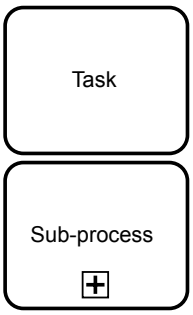
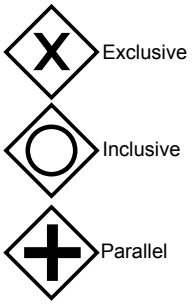


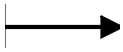
Figura 1: Ejemplo de proceso BPMN

En la figura 1 se representa el proceso de publicación y selección de un cargo docente, aquí se puede visualizar la expresividad de la notación ya

que por ejemplo permite representar la distribución de tareas entre roles del proceso, cambiar el flujo de ejecución con compuertas de decisión, así como la reutilización de procesos ya definidos mediantes sub-proceso.

En el cuadro 1 se muestra algunos ejemplos de la amplia variedad de elementos que posee el lenguaje para permitir una mayor expresividad a la hora de diseñar e implementar un proceso de negocio.

Event	
	Determinan un suceso dentro de la ejecución de un proceso. Afectan en el flujo del proceso y por lo general tienen un disparador o resultado. Existen diferentes tipos, los de *inicio* que permiten marcar el inicio de un proceso, intermedio representan sucesos dentro del curso de un proceso, y los de *fin* que indican la finalización de un proceso
Activity	
	Representan la unidad de trabajo en un proceso. Esta pueden ser atómicas (*Task*) o compuesta (*Sub-process*). En el caso de las tareas se podrá indicar el tipo permitiendo así una mayor expresividad en el diagrama.
Gateway	
	Permiten representar como el flujo del proceso converge o diverge en su ejecución. Algunos tipos a destacar son: *Exclusive* permite dividir un flujo en dos o más caminos pero solo será tomado uno, *Inclusive* similar al anterior pero permitiendo más de un camino posible, *Parrallel* indica la ejecución en paralelo de flujos

Sequence Flow	
	Utilizados para indicar el orden de precedencia de las actividades

Cuadro 1: Elementos BPMN 2.0

2.1.3. Herramientas de soporte

El ciclo de vida de un proceso, se puede describir como un proceso iterativo que involucra el modelado de procesos de negocio, el desarrollo de software para soportar esto, su ejecución, y la evaluación de su ejecución. *Business Process Management System (BPMS)* surge como la tecnología para soportar este ciclo de vida [8].

Actualmente hay una gran variedad de BPMS disponibles, tanto propietarios como de código abierto. Ejemplo de estos son Bonita, Activity, Camunda, Bizagi, por mencionar algunos.

2.2. Variabilidad en procesos de negocio

Habitualmente para un proceso de negocio, existen diferentes variantes. Cada una de estas constituyen un ajuste de un proceso de referencia a requerimientos específicos del contexto. Las herramientas actuales de administración de procesos de negocio, no soportan adecuadamente el modelado de cada variante del proceso.

Utilizando las herramientas de BPMN, las variantes del proceso son usualmente definidas y mantenidas en modelos de procesos diferentes. A esta solución se la llama "*multi model approach*". Esto, por lo general resulta en una cantidad de modelos altamente redundantes, porque los modelos de variantes son idénticos o similares en la mayoría de sus partes.

Otra alternativa frecuentemente aplicada en la práctica, es capturar las múltiples variantes en un solo modelo de proceso utilizando diferentes ramas, con bifurcaciones que se corresponden a las diferentes alternativas de contexto. A esta solución se le llama"*single-model approach*".

Ambos métodos a menudo conducen a redundancias que hacen que las modificaciones de los modelos sea una tarea lenta y propensa a errores.

En resumen, ni el uso de modelos separados para cada variante, ni la definición de todas las variantes en un mismo modelo constituye una solución viable para muchos casos.

Configurar las variantes de un proceso, es un desafío no trivial cuando se considera la alta variabilidad de procesos de negocio en la práctica, así como las muchas restricciones semánticas y sintácticas que las configuraciones de las variantes del proceso tienen que obedecer dependiendo del contexto dado.

2.2.1. Modelado de variabilidad

Han surgido un gran número de enfoques para modelar familias de variantes de procesos, con la característica común de admitir la representación de una familia de variantes de procesos de negocios a través de un modelo único, a partir del cual cada variante puede derivarse a través de ciertas modificaciones al modelo.

Utilizamos el término modelo de proceso personalizable ("*customizable process model*") para referirnos a dicho modelo consolidado de variantes de proceso y el término punto de variación ("*variation point*") para indicar que un elemento del modelo de proceso personalizable se puede personalizar a través de transformaciones [10]. A las distintas opciones que pueden sustituir un determinado punto de variación se la denomina variante concreta [5].

Las decisiones de configuración de un proceso pueden resultar en agregar o remover determinados comportamientos de determinado modelo de proceso personalizable. En este sentido se distinguen dos enfoques para el manejo de la variabilidad [10]:

- Variabilidad por restricción: comienza con un modelo de proceso personalizable que contiene todos los comportamientos de todas las variantes del proceso. La personalización se logra, restringiendo el comportamiento de este modelo de proceso personalizable.
- Variabilidad por extensión: es la alternativa opuesta a la anterior. Comienza con un modelo de proceso personalizable que representa el comportamiento compartido por la mayoría de las variantes. Para la

personalización el comportamiento del modelo necesita ser extendido para representar una situación particular. Por ejemplo, puede ser necesario insertar nuevas actividades en un modelo para poder crear la variante del proceso.

2.3. Model Driven Engineering (MDE)

Model Driven Engineering (MDE) [11], es un paradigma de Ingeniería de Software que jerarquiza el modelado como principal actividad del ciclo de vida de un sistema de software (construcción, mantenimiento, ingeniería inversa, etc.).

Propone la construcción de modelos (abstracciones) de diferentes aspectos de un sistema y la transformación de dichos modelos de forma (semi)automática. Pretendiendo reducir errores en el proceso de ingeniería de software al aumentar el nivel de abstracción en la especificación del sistema, posibilitando la verificación y reutilización de los modelos y las transformaciones. Asimismo, se enfoca en aumentar la productividad reduciendo tiempos de desarrollo a través de mecanismos automáticos de construcción.

2.3.1. Conceptos principales

Modelo

Un modelo es una descripción o especificación de un sistema y entorno para un cierto propósito. Usualmente es presentado como una combinación de imágenes y texto.

Metamodelo

Es el modelo de un lenguaje de modelado. ”Meta” enfatiza el hecho de que describe un lenguaje a un nivel de abstracción mayor. Debe ser capaz de describir la sintaxis concreta, abstracta y semántica de los modelos que lo conformen. Debe pertenecer a una arquitectura de metamodelado, permitiendo así ver un metamodelo como un modelo.

Los beneficios que ofrece un metamodelo son:

- Permite relacionar lenguajes descritos usando el mismo lenguaje de metamodelado, pudiendo ser manipulados y gestionados de forma unificada.
- Definir lenguajes abstractos de una tecnología de implementación.
- Definir y combinar diferentes abstracciones para crear nuevos lenguajes más aptos para un dominio de aplicación (aumento de productividad)

Transformaciones de modelos

Proveen una forma de producir modelos a partir de un número de modelos de origen, por lo que será necesario definir formas que le permitan definir a los desarolladores, como los elementos de los modelos de origen son seleccionados y navegados para inicializar los elementos del modelo de destino.

Uno de los principales conceptos en MDE es considerar todos los elementos, dentro de lo posible, como modelos. Las transformaciones se definirían como modelos conforme a un metamodelo que defina la semántica de la misma.

Una transformación define correspondencias entre diferentes metamodelos. Generalmente hay un modelo de origen y un modelo de destino que conforman diferentes metamodelos, al ejecutar la transformación se obtiene como resultado un modelo que conforma al metamodelo destino.

Formalmente, como vemos en la figura 2, una transformación de modelos Mt simple, que conforma el metamodelo MMt , debe proveer la forma de generar un modelo Mb, conforme a el metamodelo MMb, a partir del modelo Ma conforme a el metamodelo MMa

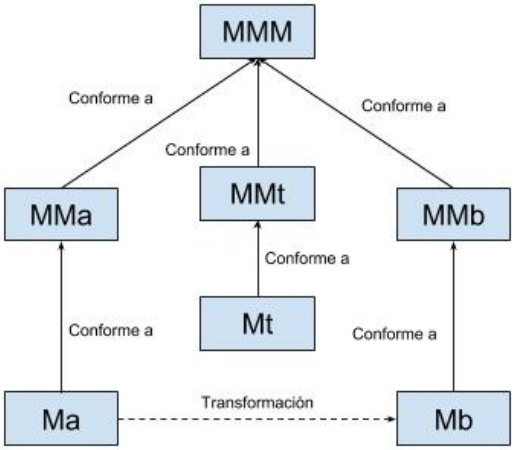


Figura 2: Transformación de modelos

2.3.2. Lenguajes de transformación

Los lenguajes de transformación definen reglas que especifican como elementos del modelo de origen deben ser tratados para generar el modelo de destino.

The Atlas Transformation Language (ATL) [12] es un lenguaje de transformaciones especificado como metamodelo y sintaxis textual concreta. Híbrido entre declarativo e imperativo. El modo predeterminado de escritura de transformaciones es declarativo pero posee constructores imperativos en el caso de que resulte muy compleja la definición en forma declarativa.

Una herramienta que da soporte a estas tecnologías es Eclipse Modeling Framework(EMF) [13], un framework de modelado y generación de código para la construcción de aplicaciones basadas en un modelo de datos estructurado.

2.4. Análisis de propuestas de variabilidad

Como punto de partida se consideran las propuestas analizadas en el proyecto [5], se toma de allí un resumen cuadro 2 que clasifica las propuestas según aspectos relevantes como son: la forma de representar la variabilidad, sobre que artefactos permite variabilidad, que lenguaje utiliza, y si posee o no herramienta de soporte.

Enfoque	Tipo	Representación	Variaciones	Lenguaje	Herramienta
Provop	Extensión	Modelo base Operaciones de cambio Modelo de contexto	Actividades	Independiente	No
CVL	Extensión	Modelo base Modelo de variantes Modelo de Resolución Modelo de contexto	Actividades	Independiente	No
vBPMN	Extensión	Modelo base con segmentos adaptativos Contexto basado en datos	Actividades	Extensión BPMN 2.0	No
PESOA	Restricción	Estereotipos Alternativas configurables	Actividades, datos	Independiente	No
C-BPMN	Restricción	Configuración de requerimientos	Actividades, compuertas	Extensión BPMN 2.0	No
BPMNt	Extensión	Tailoring	Actividades, datos, roles, compuertas, eventos	Extensión BPMN 2.0	No
BPMN*	Restricción	Estereotipos	Actividades, datos, roles, compuertas, eventos	Extensión BPMN 2.0	No

Cuadro 2: Comparativa de enfoques

De la lectura de la tabla se observa que ninguno de los enfoques tiene una herramienta que la implemente. Los enfoques de vBPMN, C-BPMN, BPMNt y BPMN* proponen extensiones al metamodelo BPMN, en cambio PROVOP, PESOA Y CVL son independientes del lenguaje sobre el que se aplique la variabilidad.

2.4.1. CVL

CVL [7, 14] (Common Variability Language) es un lenguaje independiente de dominio para la especificación y resolución de variabilidad. Facilita la especificación y resolución de variabilidad sobre cualquier instancia de un lenguaje definido a partir de un metamodelo basado en MOF [15].

Tal como se muestra en la figura 3 el modelado de la variabilidad queda independiente del modelo base y como resultado de aplicar una variante se obtiene un modelo en el mismo lenguaje permitiendo así utilizar cualquier herramienta del modelo base en este nuevo modelo.

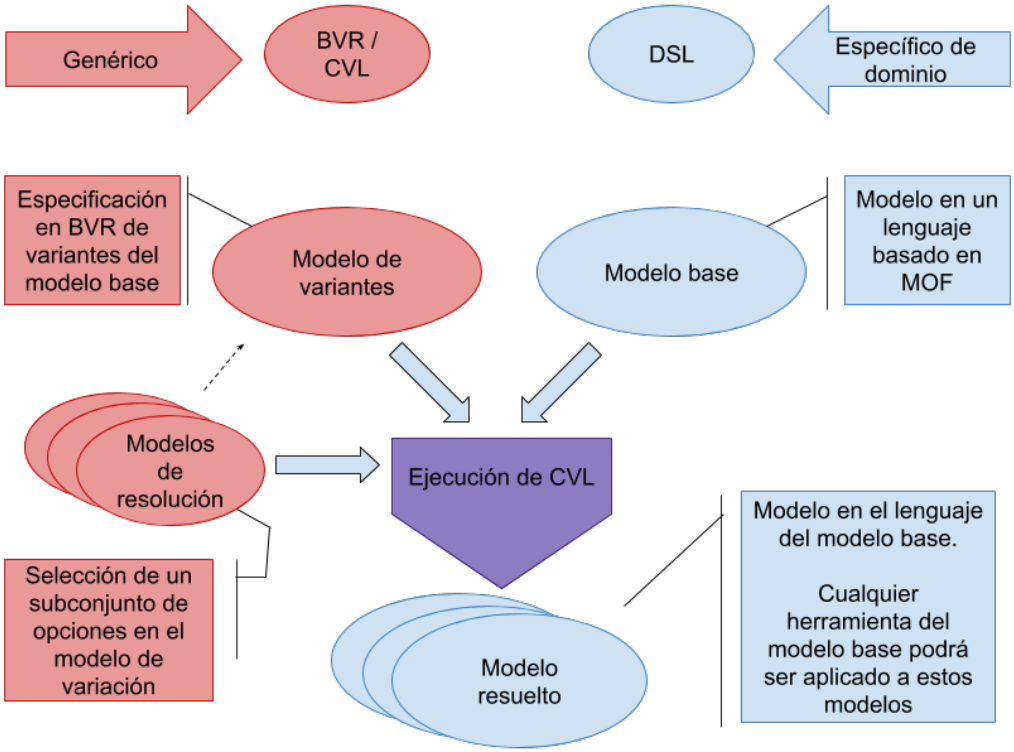


Figura 3: Principios de CVL

La arquitectura de CVL consta de diferentes modelos interrelacionados. Algunos de estos modelos (VSpec, *resolution*) sirven para definir la variabilidad de forma abstracta, esto es, no se definen consecuencias concretas

en el modelo base. Otros modelos (*realization*) contienen los puntos de variación, representando la asociación entre la abstracción de la variabilidad y el modelo base.

2.4.2. BVR

BVR (Base Variability Resolution models) [16] construido sobre CVL (Common variability Language) [17], pero CVL no es un subconjunto de BVR ya que se removieron algunas clases del metamodelo de CVL en favor de la simplicidad y se realizaron mejoras.

Una herramienta que da soporte a esta propuesta es BVR Tool [18], es un conjunto de plug-ins de Eclipse bajo la licencia "Eclipse Public License", que permiten realizar gráficamente los modelos definidos por BVR: feature model, resolution, realization y derviación de productos y mediante el desarrollo de un plugin permite la integración con diferentes editores de modelos a través de las operaciones que provee el ambiente de Eclipse.

Enfoque	Tipo	Representación	Variaciones	Lenguaje	Herramienta
BVR	Extensión	Modelo base Modelo de variantes Modelo de Resolución Modelo de contexto	Actividades, compuertas, eventos	Independiente	Si

Cuadro 3: Enfoque BVR

2.4.2.1 Modelos

Siguiendo la misma idea que CVL, está compuesto por tres modelos, a saber: realización, resolución y VSpec, que definen las variaciones sobre un modelo base.

Observamos en figura 4 las inter-relaciones de estos modelos. La abstracción de variabilidad consiste en el modelo de VSpec suplementado con restricciones, y un correspondiente modelo de resolución el cual definirá la selección del contexto. El modelo de *VSpec* es una evolución FODA [19], pero el principal propósito es proveer una definición completa tal que el modelo resultado pueda ser construido de forma automática a parir de los modelos

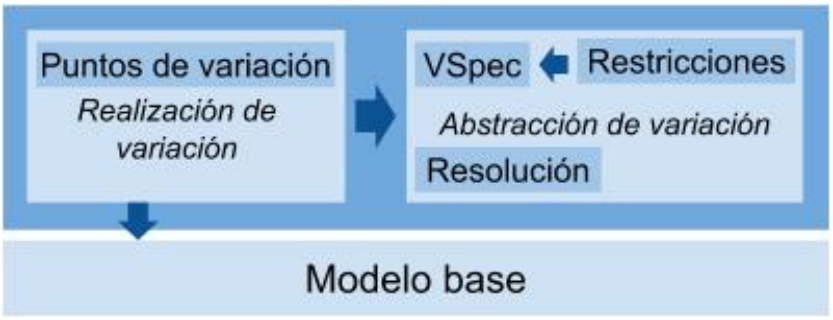


Figura 4: Arquitectura BVR

de VSpec, resolución y realización.

Modelo base

Es descrito por un Domain Specific Language (DSL) en este caso BPMN, y contiene la definición de los puntos de variación ("Placement") lugares donde serán sustituidas las variantes según el contexto, así como todos los elementos comunes a las diferentes variantes.

Modelo de realización

Contiene la definición de la correspondencia entre los elementos del modelo base con los puntos de variación, las restricciones, y el contexto en el que los puntos de variación serán válidos. A continuación vemos en detalle los puntos de variación y en particular el tipo *fragment substitution*.

2.4.2.2 VSpec

Son especificaciones abstractas de variabilidad, es decir no definen consecuencias concretas en el modelo base. Son parte del modelo de realización, su representación es mediante una estructura de árbol permitiendo reflejar restricciones lógicas al momento de la materialización:

Existen diferentes tipos de VSpec figura 5 los cuales se describen a continuación.

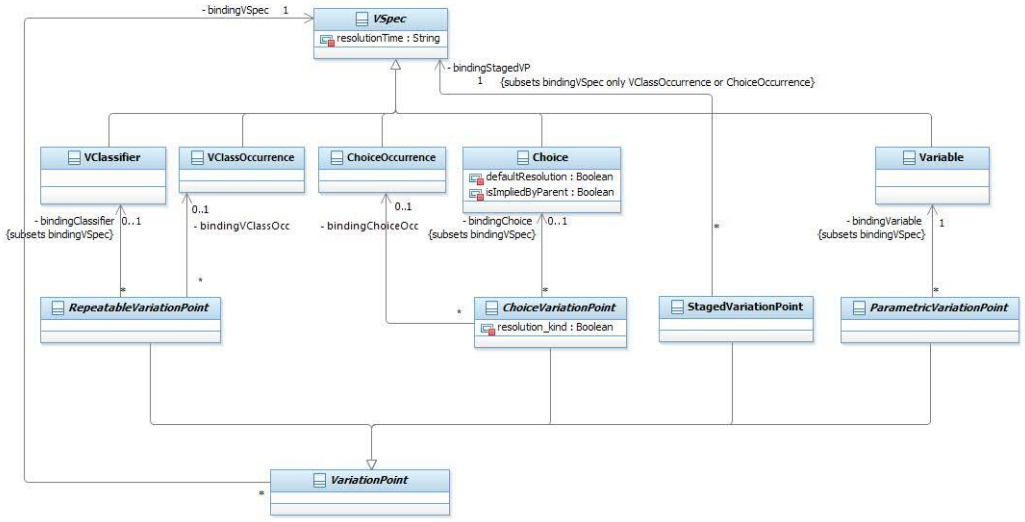


Figura 5: VSpec

- *Choice*: La resolución requiere decisión (Verdadero / Falso). Cuando un punto de variación se liga a un Choice, la decisión determina si se materializará el punto de variación.
- *ChoiceOcurrance*: Es similar a un Choice pero hace referencia a un VType
- *VClassifier*: VClassifier permite crear cero o más instancias de un Choice que luego permitirá resolver de forma independiente cada subárbol de cada instancia.
- *VClassOcurrance*: Similar a un VClassifier pero para un VType y no contiene la definición del subárbol.
- *Variable*: Variable es un VSpec cuya resolución requiere proveer un valor. Cuando un punto de variación paramétrico está ligado a un VSpec de este tipo, el valor será pasado como parámetro para la materialización del punto de variación.

2.4.2.3 Puntos de variación

Un punto de variación es una especificación concreta de variabilidad en el modelo base, define las modificaciones a realizarse al modelo base al momento de la materialización. Están relacionados al modelo base mediante *base model handles* y ligados a *VSpecs*, a partir de esta relación se desprende la dependencia de la resolución del *VSpec* al momento de la materialización.

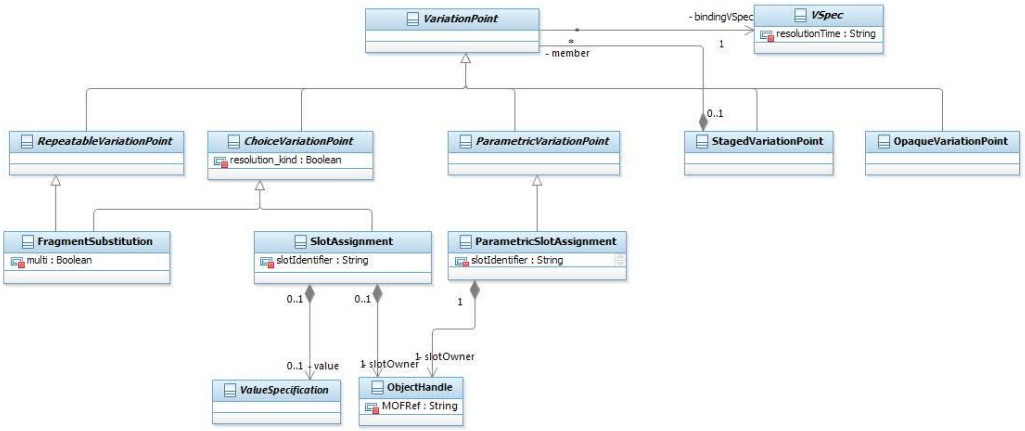


Figura 6: Metamodelo Punto de variación

Se observa en figura 6 la taxonomia de tipos de variation points posibles, y como estos se relacionan con *VSpecs* del modelo base.

Sustitución de fragmento (FragmentSubstitution)

El punto de variación *FragmentSubstitution* que hereda de *ChoiceVariationPoint*, permite definir un fragmento del modelo base y sustituirlo por uno o más fragmentos del modelo de variantes. En la figura 7 se visualiza el fragmento del metamodelo de BVR correspondiente a este tipo de punto de variación.

FragmentSubstitution, presenta dos asociaciones de suma importancia, placement que es PlacementFragment y replacemnt que es ReplacementFragmentType. Estas representan el fragmento de modelo base que va a ser sustituido, y el fragmento del modelo base

por el que se sustituye respectivamente. Tanto PlacementFragment como ReplacementFragmentType, tienen asociados ObjectHandle que contendran las referencias a los objetos en el DSL del modelo base, es decir las referencias a los objetos del modelo sobre el que se aplica la variabilidad. *FragmentSubstitution* tiene también asociado un BoundaryElementBinding que define la correspondencia entre los PlacementBoundaryElemnt (ToPlacement, FromPlacement) y los ReplacementBoundaryElement(ToReplacement,FromReplacement)

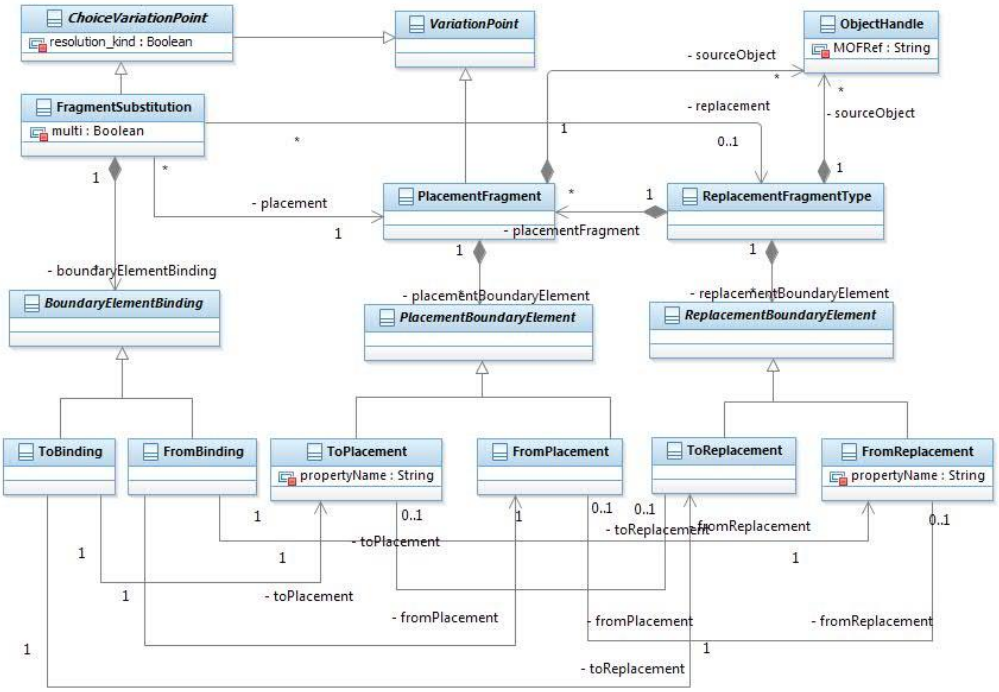


Figura 7: Metamodelo FragmentSubstitution

Cada fragmento podrá poseer un elemento de salida y un elemento de entrada, para los cuales se definirá la correspondencia mediante los "boundary elements", siendo esta la única definición de los límites de los fragmentos, se observa en detalle en figura 8.

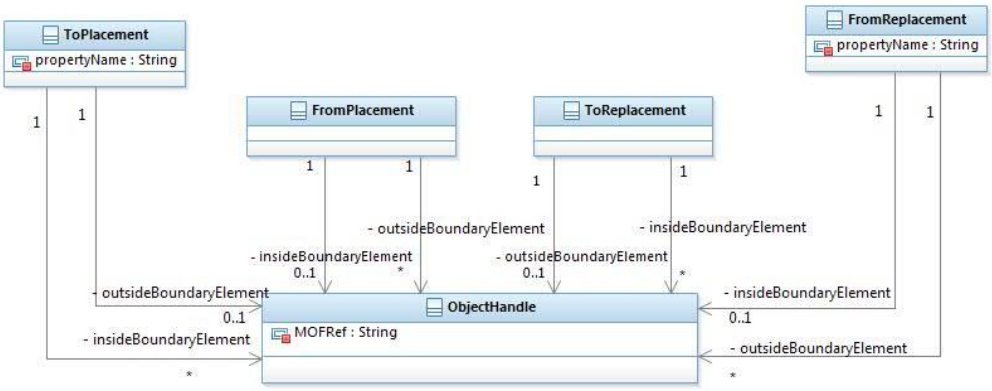


Figura 8: Definición de boundary element

2.4.2.4 Metamodelo

El metamodelo (figura 9) de BVR posee como elemento root a BVRModel el cual contiene la información de todo el modelado de variantes es decir contiene los elementos del modelo base (*baseModel*) mediante referencias a ObjectHandle, modelo de variantes (*variabiltyModel*) a través de la definición de los árboles de *VSpecs*, modelo de realización (*realizationModel*) conteniendo la relación entre los objetos del modelo base y VSpecs permitiendo así luego su materialización, y por último el modelo de resolución (*resolutionModel*) el cual dado un modelo de *VSpecs* contiene un valor de resolución, por ejemplo en el caso de un VSpec del tipo choice contendrá el valor de verdad del mismo.

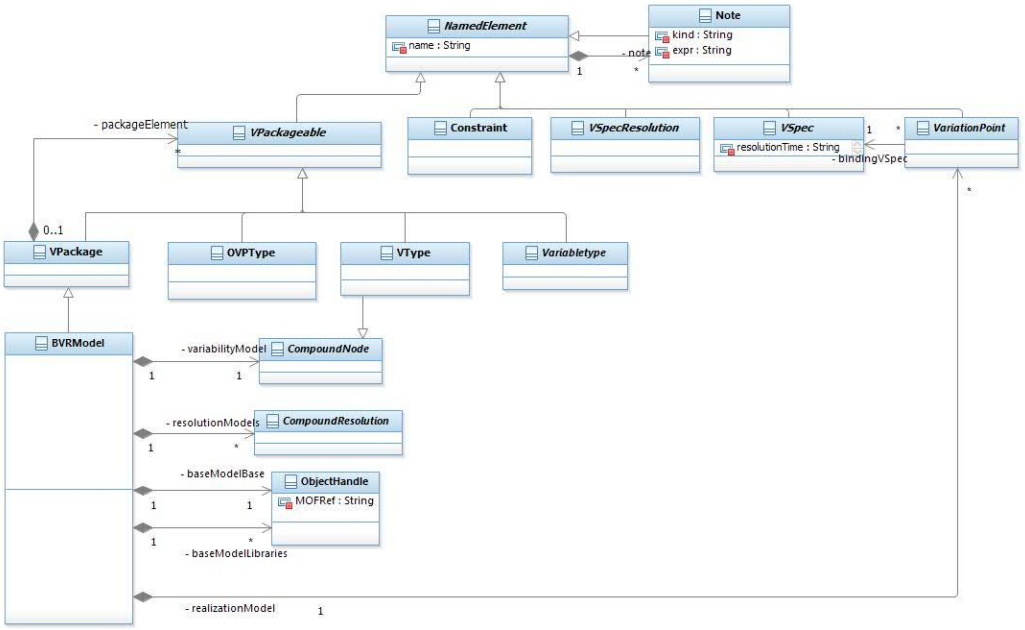


Figura 9: Metamodelo BVR

2.4.3. Modeling and managing variability in process-based service compositions

Esta propuesta [20] se enfoca en el modelado de la variabilidad en composición de servicios basados en procesos. Propone modelar la variabilidad con puntos de variación y variantes, y luego asignar correspondencias con feature models. Para soportar los puntos de variación propone una extensión del metamodelo de BPMN.

2.4.4. vBPMN:Event-Aware Workflow Variants by Weaving BPMN2 and Business Rules

El enfoque de variabilidad propuesto en [21] está enfocada en dar soporte a la variabilidad de eventos dependientes del contexto y control de excepciones, es decir, se enfoca en aspectos en tiempo de ejecución.

Puede considerarse una extensión de PROVOP, no hace énfasis en el

modelado de la variabilidad.

Para su modelado propone una extensión de BPMN creando delimitadores para los fragmentos pertenecientes a un punto de variación ”variant segment”:

1. paréntesis [ ], que definen ”adaptive segment”, que tendrá para cada variante un ”variant segment”.
2. diamante, que se coloca en la esquina izquierda de una tarea simple, tiene la misma semántica que lo anterior.

2.4.5. Configuring the variability of business process models using non-functional requirements

Esta propuesta [4] expone una metodología para identificar las posibles variantes que puede tener un proceso de negocio, basado en el análisis de los requerimientos no funcionales del problema.

Puede ser entendido como un procedimiento ordenado para la identificación de todas las posibles variantes.

Propone realizar dos fases la primera en la que se determina en primera instancia la detección de las variaciones, para este caso utiliza preguntas tales como ¿quién realiza la tarea? , ¿con qué condiciones se debe realizar, el objetivo? Luego se describen las variaciones en términos de BPMN utilizando variación y puntos de variación.

Capítulo 3

Problema planteado

Ante la problemática actual de la variabilidad en PN y las diversas propuestas de modelado de variabilidad, se pretende construir una herramienta que utilizando MDE permita la generación automática de las variantes de un PN, evitando así el modelado y mantenimiento manual de cada variante específica.

Para lograr ese objetivo es necesario resolver:

- la definición de un modelo de proceso personalizable y el modelado de las variantes
- la configuración de las variantes
- la generación de las variantes concretas a partir del modelo de proceso personalizable, las variantes y la configuración de la variantes

A continuación se plantea un ejemplo para ilustrar los problemas en un caso concreto, y se analizan con detalle los problemas a resolver para lograr cumplir el objetivo poniendo foco en el ejemplo.

Utilizaremos como ejemplo el proceso de reparación de un vehículo en un taller mecánico figura 10 (una simplificación del ejemplo propuesto en [2]).

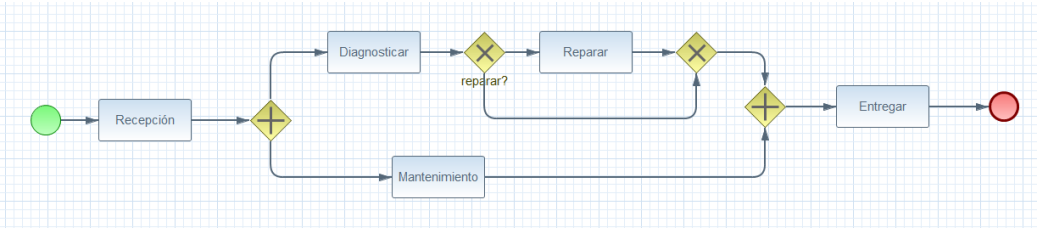


Figura 10: Proceso de reparación en taller mecánico

Este proceso comienza con la recepción de un vehículo, luego se hace un

diagnóstico y el vehículo es reparado si es necesario. También se hacen tareas de mantenimiento mientras se hace el diagnóstico y reparación, y finalmente el vehículo es entregado.

Dependiendo del contexto el proceso puede cambiar, y surge entonces la necesidad de un modelo de proceso distinto que contemple tal situación. En el ejemplo, tenemos el caso donde según el taller donde se realiza la reparación, puede ocurrir que el mantenimiento sea realizado o no. Hay talleres que hacen mantenimiento, y talleres que no hacen mantenimiento. Otra situación variante de contexto, es que algunos países tienen regulaciones específicas que exigen chequeos de seguridad antes de entregar el vehículo.

En la figura 11 se muestra la variante para el caso en que el taller no realiza el servicio de mantenimiento y tampoco se realiza el chequeo de seguridad (porque el país al que pertenece el taller no lo exige).

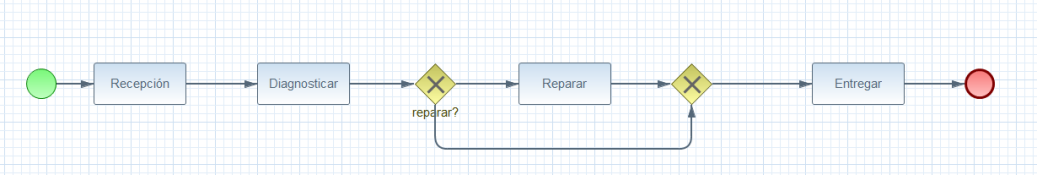


Figura 11: Variante sin mantenimiento ni chequeo de seguridad

En la figura 12 se muestra la variante de un taller donde se realiza tanto el servicio de mantenimiento como el chequeo de seguridad.

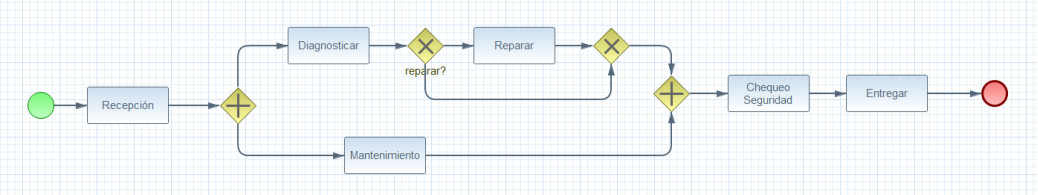


Figura 12: Variante con mantenimiento y chequeo de seguridad

Ante lo expuesto, se necesita una herramienta que sea capaz de a partir de ciertas configuraciones del modelo generar las variantes del modelo adecuadas para cada contexto (taller, país).

3.1. Modelado de variabilidad y familias de proceso

Lo primero que se debe resolver es el modelado de la variabilidad. Esto es, en primer lugar elegir entre uno de los dos enfoques de variabilidad expuestos en 2.2 (variabilidad por extensión o variabilidad por restricción), lo que dará como resultado un modelo base, en segundo lugar, algún mecanismo que permita definir puntos de variación sobre el modelo base, alguna forma de modelar las posibles variantes.

Un desafío es diseñar un modelo de proceso básico, que sirva de referencia para configurar un familia de modelos de procesos relacionados.

Dependiendo de la propuesta elegida, el mecanismo deberá poder indicar donde se pueden agregar artefactos en el modelo (extensión), o que elementos del modelo pueden ser eliminados (restricción).

Otro desafío es diseñar el modelo y ajustes de estructura que tienen que ser aplicados para configurar los diferentes procesos además del proceso básico.

Volviendo al ejemplo, sea cual sea la propuesta elegida, debe poder resolver las siguientes situaciones: ¿Cómo se puede indicar en el modelo que la tarea "Mantenimiento"puede ser borrada en el proceso de figura 10?

Para la variante con chequeo de seguridad se agregó la tarea 'Chequeo Seguridad' en figura 12 ¿Cómo y dónde se modelan los elementos que se pueden insertar en el modelo?

Algunas alternativas son:

- Extender BPMN agregando algún elemento para indicar puntos de variación.
- Buscar algún mecanismo para marcar esos puntos sin extender BPMN

En resumen, se debe resolver la forma de modelar estos 3 artefactos: modelo base, puntos de variación del modelo base y variantes.

3.2. Configuración de variantes

Con el problema del modelado de la familia resuelto, lo siguiente es buscar algún mecanismo que defina la correspondencia entre los puntos de variación y las variantes. De esta manera, definiendo diferentes configuraciones, modelar las diferentes variables concretas para los distintos contextos del proceso. Llamaremos configuración de variantes al mecanismo que resuelva este problema.

Volviendo al ejemplo, si se quiere obtener la variante figura 12, ¿de qué forma se indica si el taller ofrece el servicio de mantenimiento o no lo ofrece? ¿cómo indicar si es necesario el chequeo de seguridad?

3.3. Transformación

Resolviendo los aspectos anteriores contamos con una forma de realizar el modelo base, definir los puntos de variación y variantes y especificar una configuración concreta. En definitiva, contamos con toda la información para obtener una variante concreta. Para ello se propone definir una transformación de modelos, que tenga como entrada toda esa información y como salida el modelo de proceso de la variante deseada, como se muestra esquemáticamente en figura 13.

De esta manera, al ejecutar la transformación con diferentes configuraciones, se obtendrían las diferentes variantes concretas del proceso.

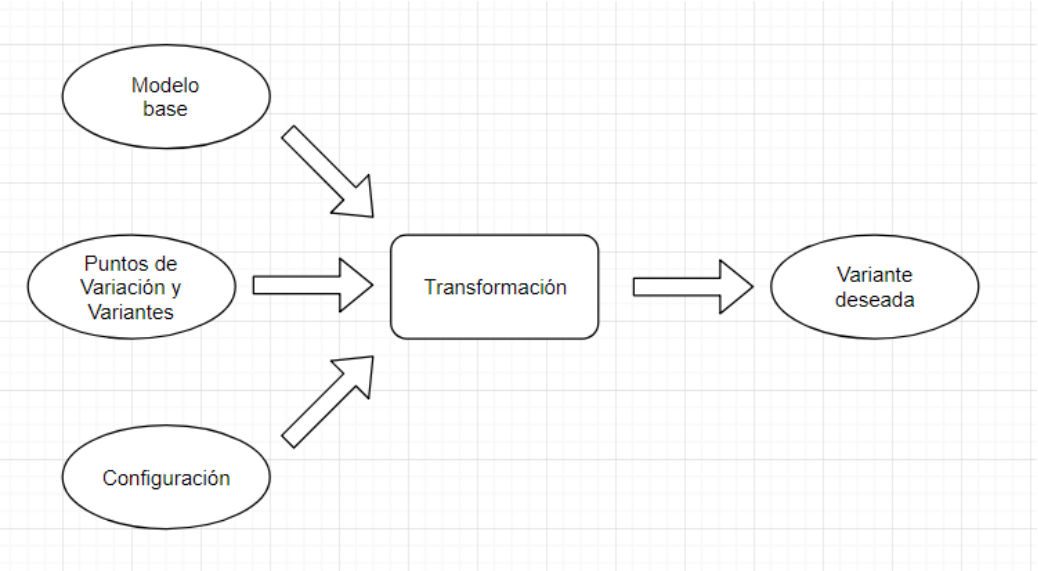


Figura 13: Transformación

Capítulo 4

Solución propuesta

En este capítulo nos enfocamos en definir estrategias para la generación automática de variantes a partir del modelado de la variabilidad y la aplicación de técnicas MDE, y describir como se hizo para integrar esas estrategias a un herramienta gráfica de modelado de variantes

Como resultado del estado del arte fueron descartadas algunas propuestas de variabilidad en BPMN que proponen la extensión del lenguaje con nuevos artefactos, es decir modificaciones al metamodelo BPMN original. Dentro de las propuestas estudiadas, vimos que en general las que no agregan artefactos a los modelos estaban basadas en PROVOP o CVL.

Luego de estudiar las alternativas se observó que CVL es un lenguaje muy completo con buena documentación y especificación, que explica los tipos de puntos de variación que soporta, en qué casos utilizarlos y cómo se aplican los cambios en el modelo luego de la resolución. Mientras que las propuestas basadas en PROVOP son más teóricas y falta bajar muchas cosas a tierra, y hay escasa documentación.

A partir de este análisis se decidió seguir por un enfoque basado en CVL.

Al investigar CVL se identificó el tipo de punto de variación "Fragment Substitution", como el más adecuado para aplicar en la mayoría de los casos de BPMN.

Además, se determinó desarrollar nuestra solución basándose en BVR ya que se presenta como una evolución de CVL, utiliza "Fragment Substitution" y cuenta con una herramienta (BVR Tool) que permite crear modelos BVR a partir de otros modelos.

Otro aspecto a resolver es la herramienta de modelado de procesos a utilizar. Se decidió utilizar la herramienta de código abierto *Eclipse BPMN2 Modeler*, dado que se integra al ambiente de desarrollo Eclipse. *Eclipse BPMN2*

Modeler es una herramienta de modelado gráfico para la creación de procesos de negocios. Proporciona un marco de edición de flujo de trabajo gráfico, que se puede personalizar fácilmente para cualquier motor de ejecución compatible con BPMN 2.0 [22].

La herramienta BVR Tool [18] provee una interfaz gráfica para la definición de las familias, es de código abierto y de forma relativamente simple permite la integración mediante interfaces a cualquier editor de modelos basado en ECore, lo que posibilita la integración con el modelador BPMN2 Modeler [22].

A lo largo del capítulo se presentan ejemplos con imágenes que refieren al ejemplo del taller mecánico planteado en capítulo 3.

4.1. Modelado de variabilidad y familias de procesos

4.1.1. Modelado de Variantes

Nuestra solución está basada en el punto de variación ”Fragment Substitution”, esto implica la sustitución de un fragmento ”Placement” por un fragmento ”Replacement”. Para esto se definió dos modelos, un modelo base que contiene los ”Placement” y un modelo de replacement que contiene los ”Replacement”.

Modelo base

El modelo base figura 14 es el punto de partida para la generación de la familia. Se utilizó la propuesta de variabilidad por extensión. Es un modelo que contiene el flujo común a todas las variantes identificadas y contiene los ”Placement” que serán sustituidos por ”Replacemnt” según la configuración, para determinar que variante concreta es generada.

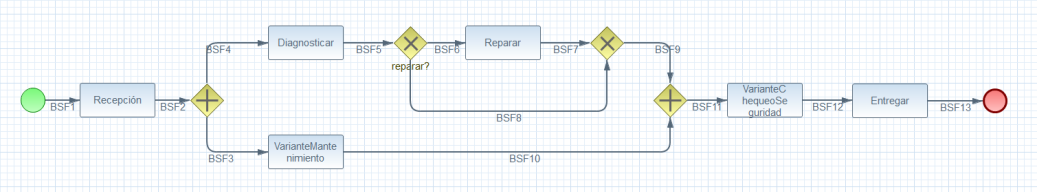


Figura 14: Modelo base

Modelo de replacements

El modelo de *replacements* figura 15 se modela como un Diagrama de Colaboraciones de BPMN. Este diagrama contiene procesos dentro de Pools, cada uno de estos procesos representa un fragmento de tipo "Replacement". Este mecanismo es bueno porque:

- Se mantienen todos los fragmentos "Replacement" en un mismo modelo.
- Es muy fácil diferenciar los fragmentos porque cada uno está en un Pool diferente.

Como se mencionó, cada fragmento es representado como un proceso, esto implica que tiene un evento de inicio y uno de fin, estos eventos solo existen para que sea posible el modelado pero no forman parte del fragmento.

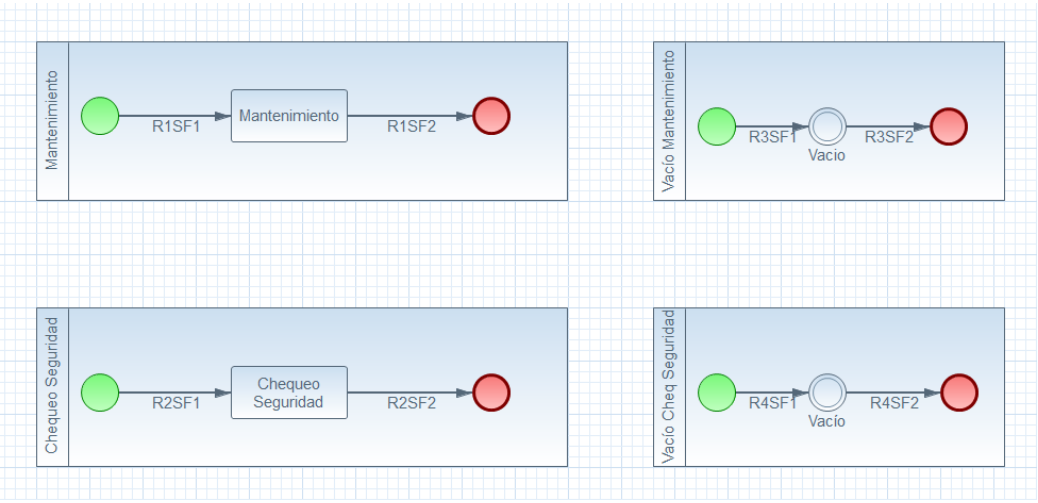


Figura 15: Modelo replacements

4.1.2. Modelado de Variabilidad con BVR Tool

Un ”Fragment Substitution” es un ChoiceVariationPoint, esto implica que la sustitución se va a aplicar si el punto de variación está ligado a una VSpec de tipo Choice que se resolvió ”true”.

4.1.2.1 Especificación de Variabilidad

Como primer paso para la generación de familias es necesario definir el árbol de VSpec que servirá para determinar que condiciones debe cumplir un punto de variación asociado al mismo para luego ser materializado en la etapa final del proceso.

El editor de VSpec permite definir de forma gráfica cuales serán las variables de contexto que se deberán evaluar en el momento de la materialización de la instancia, aquí también se verá fácilmente como es la dependencia entre las mismas.

La estructura de árbol también implica restricciones, si un elemento termina recibiendo valor ”true” implica que el padre debe ser ”true”.

La figura 16 indica la especificación de variabilidad para el ejemplo del taller mecánico. Los ”Fragment Substitutions” van a estar ligados a las choices ”ConMantenimiento”, ”SinMantenimiento”, ”ConChequeo” y ”SinChequeo”.

4.2. Configuración de variantes

La configuración de variantes se resolvió mediante la utilización del ”Resolution Editor” de BVR Tool. Este editor crea un árbol basándose en el de VSpecs, la diferencia es que en este árbol se le dan valores true/false a las hojas (se resuelven los Choices), estos valores son los que determinan si el ”Fragment Substitution” asociado al VSpec se va a aplicar.

La figura 17 muestra una posible resolución para el ejemplo del taller mecánico. A la hora de dar valores a las Choices es necesario tener mucho cuidado ya que en este caso, por ejemplo, las Choices ”ConMantenimiento” y ”SinMantenimiento” tienen que ser excluyentes.

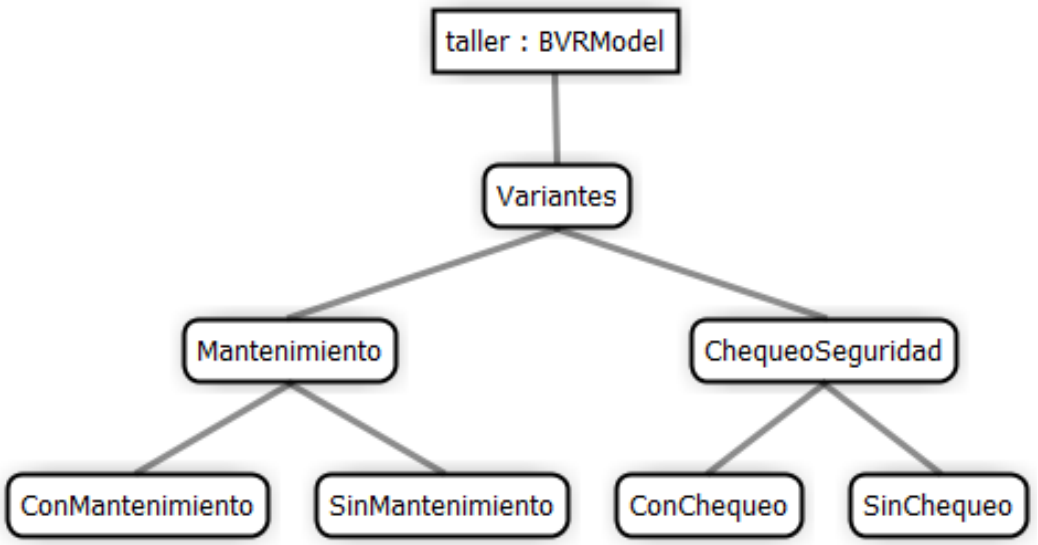


Figura 16: Especificación de Variabilidad

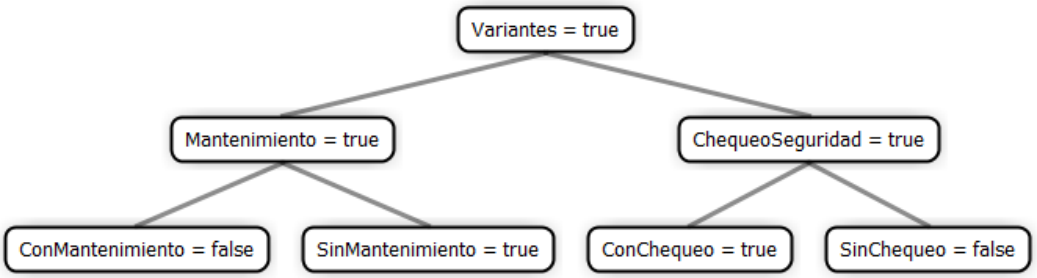


Figura 17: Resolución sin mantenimiento, con chequeo de seguridad

4.3. Transformación

Se desarrolló una transformación ATL que determina qué ”Fragment Substitutions” hay que aplicar y luego hace la sustitución.

La transformación tiene como modelos de entrada el modelo base en BPMN, el modelo de reemplazos en BPMN y el modelo BVR, que contiene toda la información sobre los cambios a realizar. Con estos tres modelos se generará un nuevo modelo BPMN según las variables de contexto seleccionadas en el modelo de BVR.

En ATL esto es:

```
create OUT: bpmn from IN: bvr, base: bpmn, replacement: bpmn;
```

A partir del modelo bvr, la transformación define un helper que indica qué ”Fragment Substitutions” se van a aplicar. Como se mencionó anteriormente, estos ”Fragment Substitutions” son los que están ligados a una VSpec que se resolvió true.

```
helper def: getResolvedFragmentSubstitutions: Sequence(bvr!FragmentSubstitution) =
        bvr!FragmentSubstitution.allInstances() -> select(fs | thisModule.
                        getVSpecsResolvedTrue.includes(fs.bindingVSpec));
```

En cada regla de la transformación se resuelve si el elemento que cumple con la regla debe ser o no copiado al modelo resultado. Un elemento será copiado si pertenece al modelo base y no está incluido en ningún fragmento de tipo ”Placement”, si pertenece al modelo base y pertenece a un ”Fragment Substitution” resuelto como false, o si pertenece al modelo replacement y pertenece a algún fragmento de tipo ”Replacement” que pertenece a un ”Fragment Substitution” que fue resuelto como true.

Los fragmentos se obtienen a partir del helper que indica los ”Fragment Substitutions” a aplicar:

```
-- Indica si hay que copiar el elemento
helper def: copyElement(element: bpmn!FlowElement): Boolean =
         thisModule.baseFlowElementsToCopy.includes(element) or thisModule.
                         replacementFlowElements.includes(element);
```

La transformación hace uso de reglas abstractas para facilitar la copia de elementos, siguiendo la estructura del metamodelo de BPMN. A continuación se presenta un ejemplo simplificado para explicar el funcionamiento.

La figura 18 muestra un extracto del metamodelo de BPMN para ilustrar la herencia entre elementos. Un elemento de tipo Event es un FlowNode, por lo que tiene las referencias incoming y outgoing. De la misma forma, un elemento de tipo StartEvent es un CatchEvent que a su vez es un Event. El mapeo de esa estructura a reglas es el siguiente

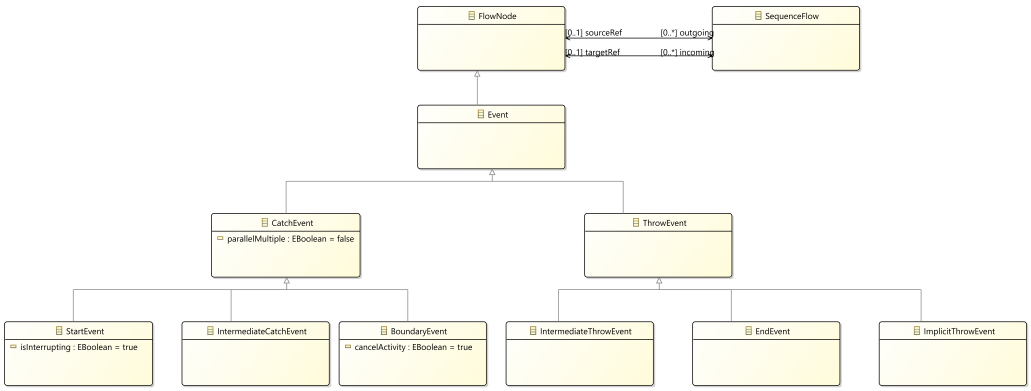


Figura 18: Extracto metamodelo BPMN

```
abstract rule copyEvents {
        from
                event: bpmn!Event (
                        thisModule.copyElement(event)
                )
        to
                eventOut: bpmn!Event (
                        id <- event.id,
                        name <- event.name,
                        incoming <- event.incoming,
                        outgoing <- event.outgoing
                )
}

abstract rule copyCatchEvents extends copyEvents {
        from
                event: bpmn!CatchEvent
        to
                eventOut: bpmn!CatchEvent (
                        parallelMultiple <- event.parallelMultiple
                )
}
```

```
rule copyStartEvent extends copyCatchEvents{
        from
                event: bpmn!StartEvent
        to
                eventOut: bpmn!StartEvent (
                        isInterrupting <- event.isInterrupting
                )
}
```

Un caso especial de sustitución, es cuando se desea borrar un fragmento. Para modelar este caso, se establece como convención de modelado, hacer una sustitución por un artefacto BPMN de tipo IntermediateThrowEvent, sin asociarle un EventDefinition, como se muestra en la figura 20 (los 2 fragmentos de la derecha). Esto implica que al hacer la sustitución, en el modelo de salida se van a agregar IntermediateThrowEvents.

Dado que estos IntermediateThrowEvents no agregan semantica al proceso BPMN, tienen que ser eliminados, para que efectivamente el resultado de la sustitución sea el borrado de un fragmento.

Para quitar esos elementos del modelo de salida se implementaron 2 transformaciones, una de ellas se encarga de separar esos IntermediateThrowEvents y unir su elemento de entrada con su elemento de salida, mientras que la otra transformación se encarga de borrar dichos IntermediateThrowEvents.

4.4. Propuesta integrada

El objetivo de esta sección es ilustrar la arquitectura final de la solución.

La solución propuesta como se muestra en la figura 19 se divide en dos plugins, uno que será el encargado de interactuar entre la herramienta BVR tool y el modelador, y otro que permitirá ejecutar la transformación realizada en ATL mediante la selección del modelo base, modelo de reemplazo y bvr.

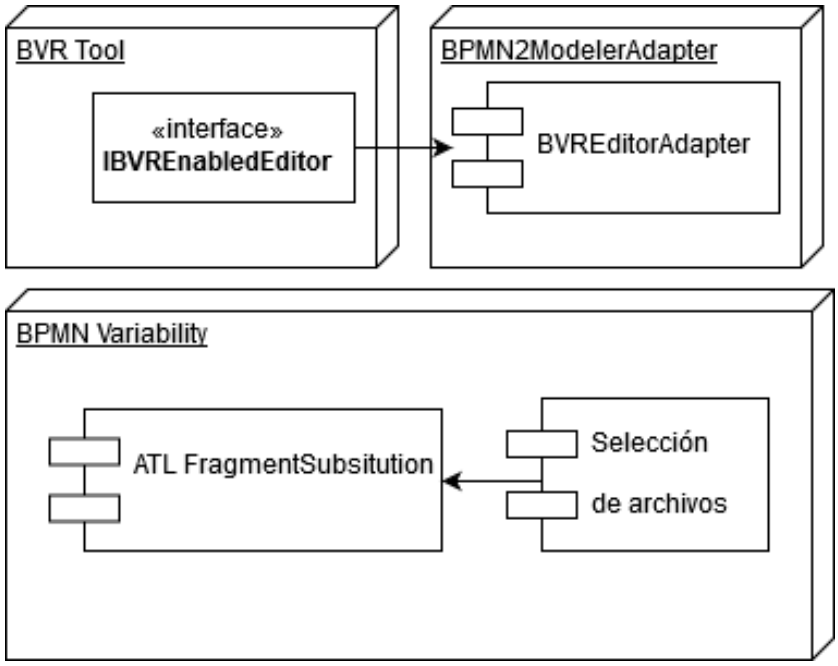


Figura 19: Arquitectura de la solución

4.4.1. Integración de BPMN con BVR Tool

BVR Tool provee una interfaz que debe ser implementada para poder integrar modeladores basados en ECore. Se desarrolló un plugin llamado BPMN2ModelerAdapter que implementa esa interfaz para integrar el BPMN2 Modeler. Esta integración permite modelar los fragmentos "Placement" y "Replacement" (BVR) que luego formarán parte de un "Fragment Substitution" (BVR), el modelado de fragmentos se crea mediante selección de elementos en un modelo BPMN.

Debido a limitantes de Eclipse respecto a que thread pude manipular por ejemplo los elementos seleccionados se decidió crear la siguiente clase, la que luego será ejecuta de forma sincrona permitiendo así obtener los elementos seleccionados de un modelador que se encuentre actualmente activo.

```
class SelectionManager implements Runnable {
    private BPMN2MultiPageEditor editor;
    private List<Object> selection;

    public SelectionManager(BPMN2MultiPageEditor editor) {
        this.editor = editor;
    }

    public List<Object> getSelection(){
        return selection;
    }

    @SuppressWarnings("unchecked")
    public void run() {
        ISelection iselection = editor.getSite().getSelectionProvider().getSelection();
        StructuredSelection structuredSelection = (StructuredSelection) iselection;
        selection = structuredSelection.toList();
    }
}
```

La operación más relevante de la interfaz es getModelObjects ya que mediante esta se obtienen cuales serán los elemento del modelo en la creación de una variante, para esto se le pasa a la operación un listado de elementos (aquellos que fueron seleccionados por el usuario) y deberá retornar cuales son los elementos del modelo para los mismos. En el caso de BPMN fue necesario descartar los elementos pertenecientes al modelo gráfico ya que solo puede existir en el marco de la herramienta un elemento Ecore para cada elemento seleccionado, para esto de los elementos seleccionados se descartan todos los que no pertenezcan al metamodelo de BPMN2

```
@Override
public List<EObject> getModelObjects(List<Object> objects) {
    List<EObject> eObjects = new BasicEList<EObject>();
    for (Object object : objects) {
        EObject eObject = null;
        if (object instanceof EObject) {
            eObject = (EObject) object;
        } else if (object instanceof IGraphicalEditPart) {
                eObject = ((IGraphicalEditPart) object).resolveSemanticElement();
                eObjects.add(eObject);
        } else if (object instanceof IPictogramElementEditPart) {
            PictogramElement pe = ((IPictogramElementEditPart) object).getPictogramElement();
            PictogramLink link = pe.getLink();
            if (link != null) {
                for (EObject bo : link.getBusinessObjects()) {
                    /*Solo me quedo con los elementos que pertenecen a bpmn2*/
                    if (bo.eClass().getEPackage().getName().equalsIgnoreCase("bpmn2"))
                        eObjects.add(bo);
                }
            }
        }
    }
    return eObjects;
}
```

4.4.1.1 Modelo de realización

Habiendo solucionado la integración entre el BPMN2 Modeler y BVR Tool, la siguiente etapa es modelar los Puntos de Variación junto con las cosas asociadas. Para esto se utilizó el "Realization Editor" de BVR Tool.

Creación de Fragmentos

Para crear un "Fragment Substitution" primero es necesario crear los fragmentos. El plugin BPMN2ModelerAdapter hizo posible crear fragmentos mediante selección de elementos en los modelos BPMN.

La figura 20 muestra los fragmentos que se crearon para el ejemplo del taller mecánico, notar que el "Replacement" seleccionado en el "Realization Editor" es el que está resaltado en el modelo BPMN, esto es otra funcionalidad que se logró gracias al BPMN2ModelerAdapter.

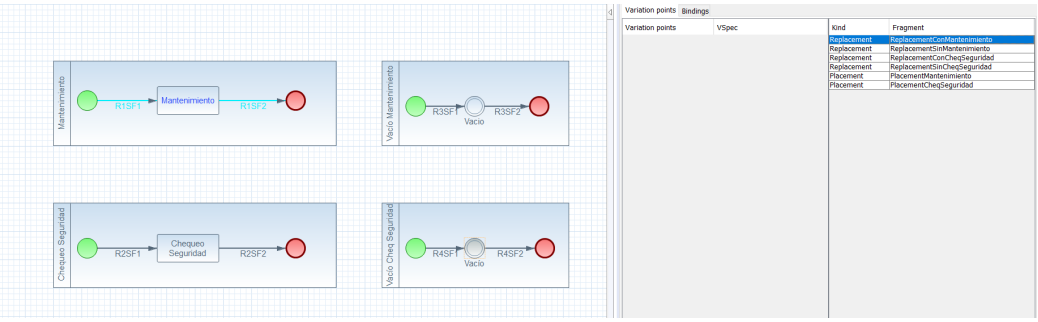


Figura 20: Fragmentos

Punto de variación - Fragment Substitution

Luego de generados los placement y replacement se debe proceder a realizar la asociación de puntos de variación del tipo "Fragment Substitution", para esto se deberá seleccionar un placement y un replacement, dar click derecho en la parte izquierda y seleccionar Create FragmentSubstitution tal como se muestra en la figura 21.

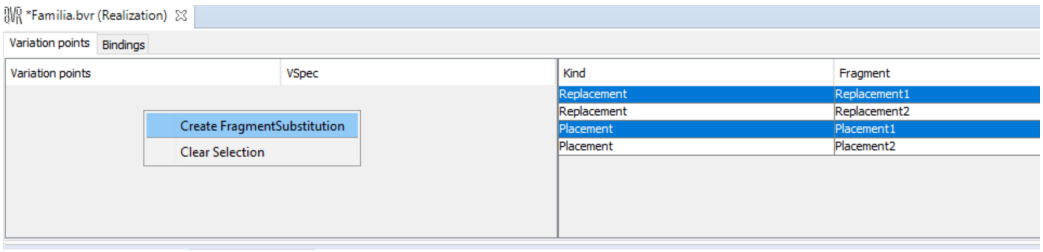


Figura 21: Creación de punto de variación

Al crearlo se desplegará por cada fragment substitution un listado de los VSpec definidos en el modelo de variantes, esto permitirá indicar cual variable de contexto afectará que se decida si se aplica o no la sustitución de fragmentos.

Por último hay que generar los bindings que determinarán cual es la correspondencia entre elementos del placement y replacement así como se deben conectar los limites de cada fragmento. Realizando Click derecho sobre un punto de variación y seleccionando generate bindings, se generará toda la información de los elementos a los que se debe realizar la correspondencia.

Al ir a la hoja de bindings en el Realization Editor se podrá ver y asignar los mismos (figura 22), teniendo en cuenta las siguientes indicaciones:

- Todos los elementos referentes a metamodelo gráfico (BPNShape, BPMNEdge) dejarlos en NULL ya que no se van a considerar en al generación.
- Los flowElements del modelo base asignarlos a los correspondientes del replacement.
- Indicar el targetRef del SequenceFlow (límite de entrada del modelo base) al targetRef del elemento de entrada del modelo del replacement.
- Al igual que en el punto anterior, será necesario asignar el sourceRef del modelo base (límite de salida del modelo base) al correspondiente límite de salida del modelo destino.
- Asignar cual será la conexión inicial desde el modelo de replacement al modelo base mediante el incoming del primer elemento del modelo de replacement .

- Tal como se indicó en el paso anterior será necesario indicar como se conectará el fin del modelo del replacement al modelo base mediante el outgoing.

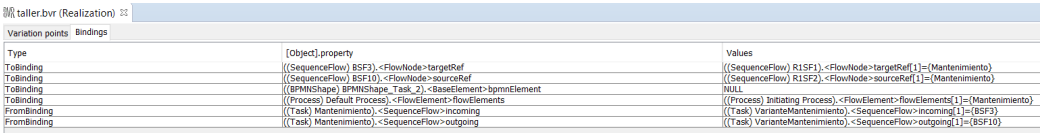


Figura 22: Definición de Bindings

4.4.2. Ejecución, Plugin de Eclipse

Para facilitar la ejecución de la transformación, se desarrolló un plug-in eclipse que permite seleccionar los archivos (modelo base, modelo de reemplazo, BVR) y generar la familia.

Se invoca el plugin desde la barra de menú desplegable "BPMN Varibilityz luego en la opción "Generate Variants"

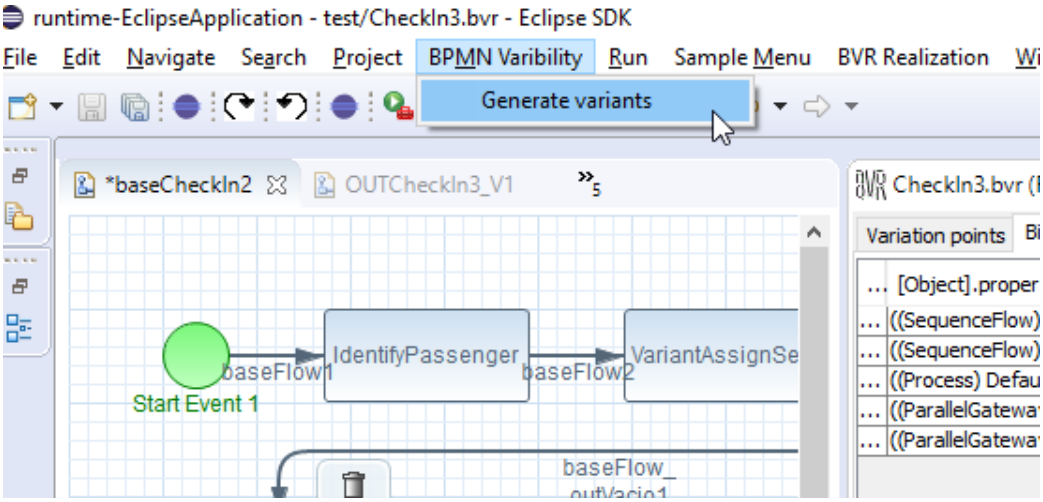


Figura 23: Invocación de plugin

Al mostrarse el cuadro para la selección de los archivos, se tiene que indicar el archivo Configuration (BVR), el modelo Base (BPMN), el modelo Replacement (BPMN) y el modelo Output (BPMN).

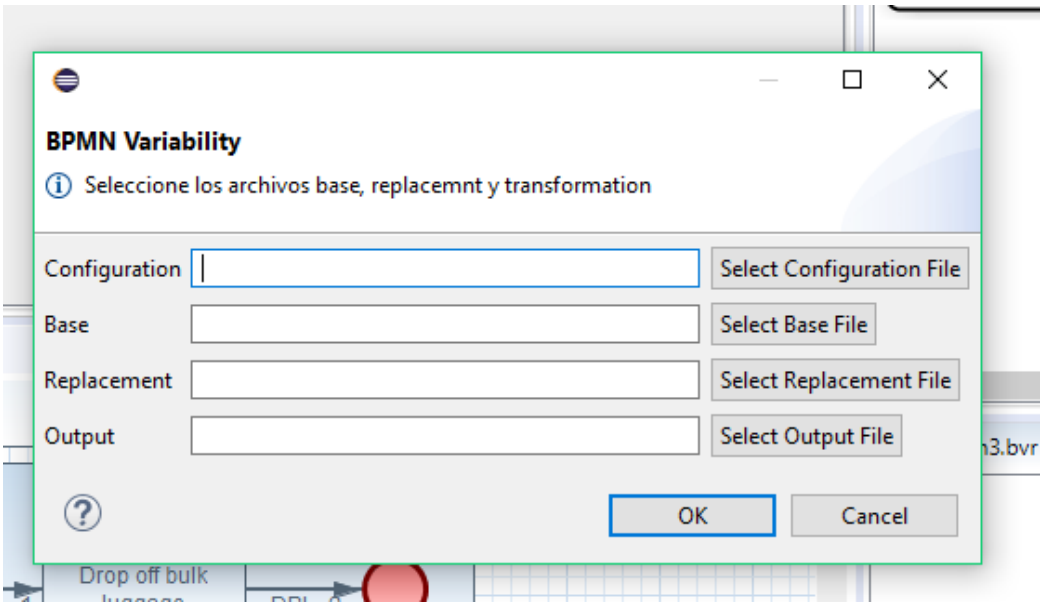


Figura 24: Selección de archivos

Luego de seleccionados todos los archivos se invoca la transformación al hacer click en botón OK.

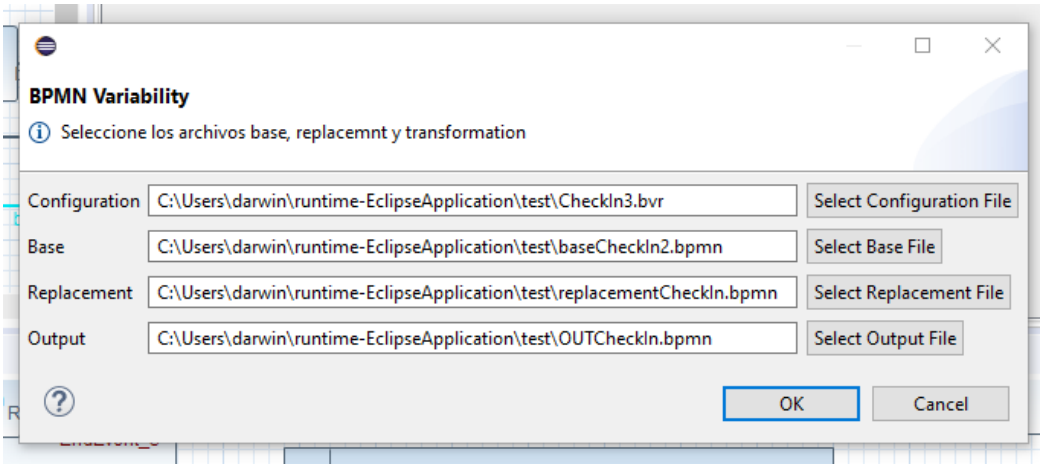


Figura 25: Invocación de transformación

4.4.2.1 Restricciones

Para la generación de las variantes se deberá tener en consideración las siguientes restricciones:

- Se debe poseer un solo archivo del modelo base.
- Los modelos de sustitución deberán estar definidos en un mismo archivo.
- Los placement (fragmentos del modelo base) y replacemente (fragmentos del modelo de sustitución) deberán tener solo un sequence flow de entrada y uno de salida ya que los mismo serán utilizado para establecer los bindings al momento de la materialización.
- un Replacement podrá ser utilizado en un solo fragment substitution.

No se establece como una restricción pero si como una recomendación el nombrar todos los elementos de los modelos ya que estos permitirán identificar fácilmente los elementos que se quieren sustituir y como es su relación con el modelo base.

4.4.2.2 Consideraciones

Debido a que al momento de la sustitución no se puede determinar si la posición de los elementos gráficos es correcta y debido a que la herramienta BVR tool no permite poseer más de una definición de un elemento de un metamodelo, ejemplo una actividad posee el elemento de actividad de BPMN y shape correspondiente a la parte gráfica, se decidió excluir los elementos gráficos en la trasformación. Esto provoca que al abrir un archivo generado como resultado de la ejecución de la herramienta se emita un mensaje de que hay elementos que que no poseen definición gráfica.

Capítulo 5

Caso de estudio

Para verificar que el prototipo propuesto es aplicable, se implementó la generación de las variantes a un ejemplo que plantea las diferentes opciones al realizar el Chek-in de vuelo de un aeropuerto, caso de referencia expuesto en VIVACE [23]. Para el modelado de la solución del caso, se tuvo en cuenta el análisis realizado en ese documento, se recomienda seguir la solución aquí propuesta teniendo a la mano el documento que expone el caso con mayor detalle.

El caso plantea 6 variantes diferentes que son el resultado de distintas situaciones que pueden darse al realizar el check in, dependiendo de características del pasajero y variantes de equipaje. Para una persona se distinguen los casos: menor de edad no acompañado (UM, Unaccompanied minor), persona discapacitada (HAN, Handicapped), adulto (ADULT). Se distingue el destino del viaje, considerando: Estados Unidos (USA), o Unión Europea (EU). Se distinguen también el tipo de pasaje (business, economy), casos de equipaje con sobrepeso (overweight luggage), check in realizado electrónicamente.

Las variantes 1 figura 26 y 2 figura 27 asumen que el check-in está hecho en línea por el pasajero. Primero, se identifica al pasajero y un asiento es asignado.

La variante 1 describe el proceso en caso de que el pasajero esté volando desde Europa a los Estados Unidos, que requiere información sobre el alojamiento, así como completar el sistema electrónico para autorización de viaje (es decir, formulario ESTA). Finalmente, se imprime una tarjeta de embarque electrónica y el pasajero se despacha el equipaje en el mostrador de clase ejecutiva.

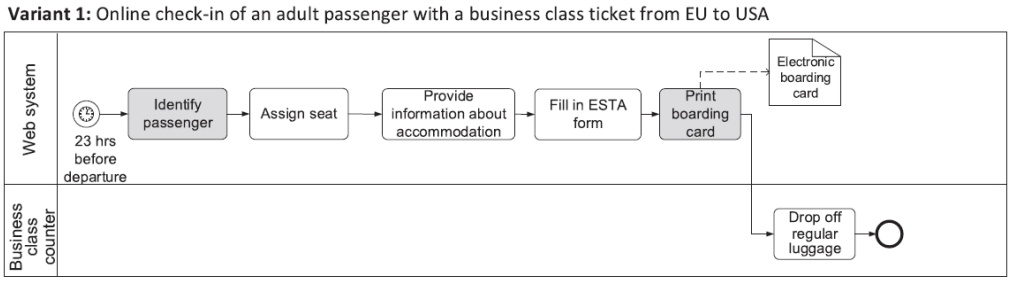


Figura 26: Variant 1: Check-in online de pasajero adulto que viaja de USA a UE en clase ejecutiva

En cuanto a la variante 2, después de imprimir la tarjeta de embarque, se requiere el pago de una tarifa adicional, debido al sobrepeso del equipaje.

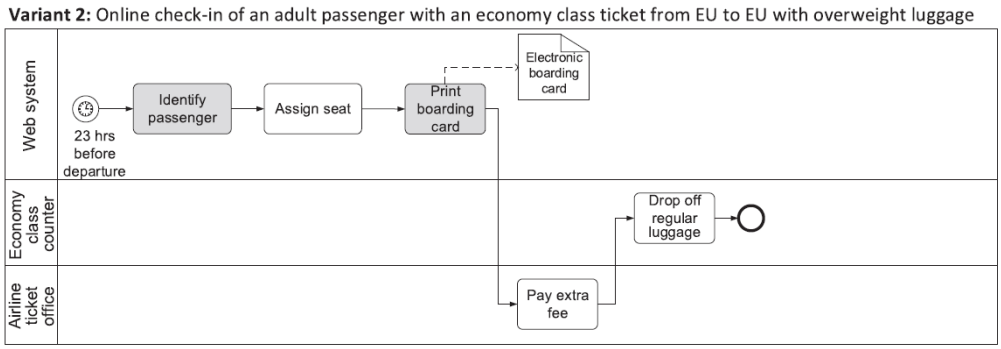


Figura 27: Variant 2: Check-in online de pasajero adulto que viaja de EU a EU en clase económica con sobrepeso de equipaje

A su vez, en la Variante 3 (figura 28) el check-in se realiza en la máquina de autoservicio y el equipaje se deja en el mostrador de entrega rápida del equipaje.

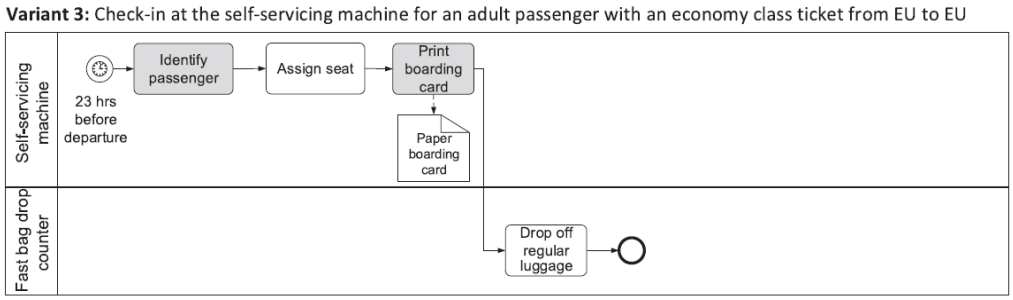


Figura 28: Variant 3: Check-in en maquina de autoservicio de pasajero adulto que viaja de EU a EU en clase económica

Finalmente, para estas tres variantes de proceso, el check-in está disponible 23 horas antes de la salida. Por el contrario, las variantes 4 figura 29 y 6 figura 31 representan el proceso de check-in realizado en el mostrador respectivo en el aeropuerto. La variante 4 describe el check-in para una persona menor no acompañada, se asigna un asiento especial y se completa un formulario adicional, además, se requiere una copia de la tarjeta de embarque para el pariente que acompaña al menor a la puerta de embarque.

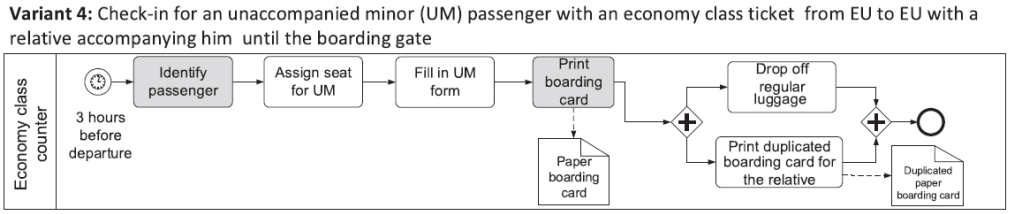


Figura 29: Variant 4: Check-in de menor no acompañanado que viaja de UE a UE en clase económica con familiar que lo acompaña hasta la puerta de embarque

Variante 5 figura 30 se refiere a un pasajero discapacitado que requiere asistencia adicional por parte de un acompañanate.

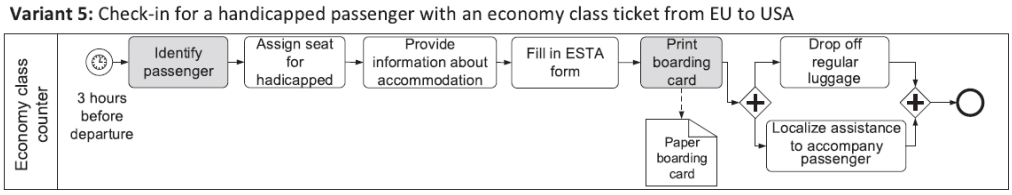


Figura 30: Variant 5: Check-in de pasajero discapacitado que viaja de UE a USA en clase economica

Para la variante 6 figura 31 corresponde al proceso de registro de un pasajero que transporta equipaje a granel.

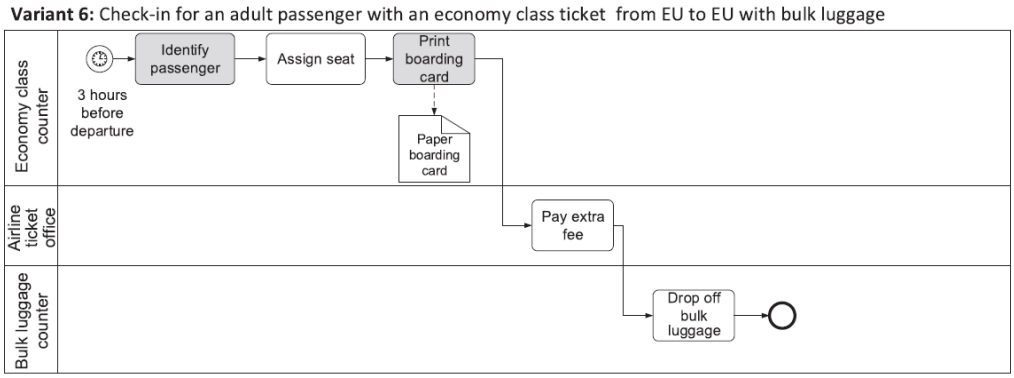


Figura 31: Variant 6: Chek-in de pasajero adulto que viaja de EU a EU con equipaje a granel

En estas tres últimas variantes(4,5,6), un check-in solo puede realizarse al máximo 3 horas antes de la partida, una vez que los mostradores se hayan abierto. Finalmente, la tarjeta de embarque está impresa en formato de papel.

figura 26 y figura 27 ejemplifican la familia de procesos del check-in e ilustran la complejidad de este último caso debido a la variabilidad de su contexto de aplicación (por ejemplo, tipo de pasajero, destino del vuelo, y tipo de equipaje).

El modelado, diseño e implementación de la familia de procesos sería un

trabajo muy engorroso. Resulta desafiante la adecuada gestión de las familias de procesos (proceso de variabilidad) y sus miembros (variantes de proceso).

Para modelar el caso, debemos crear:

1. Proceso base BPMN, donde se definen los puntos de variación
2. Proceso de reemplazo BPMN, donde se definen las posibles variantes
3. Modelo VSPEC BVR, que defina las opciones lógicas de la realidad
4. Modelo de Realización BVR, que defina las sustituciones
5. Modelo de Resolución BVR, que asigna valores a las variables definidas en VSPEC, la configuración.

Iniciamos creando el modelo BPMN base el cual define un flujo común de las variantes. Este modelo sigue la propuesta de variabilidad por extensión, representa el comportamiento compartido por la mayoría de las variantes.

El proceso base que observamos en figura 32 contiene tareas que son comunes a todas las variantes, IdentifyPassenger y PrintBoarding, y tareas que serán sustituidas según el caso que se esté generando (VariantAssingnSeat, VariantExtraInfo, VariantPayment y VariantLuggage) estas tareas serán los puntos de variación.

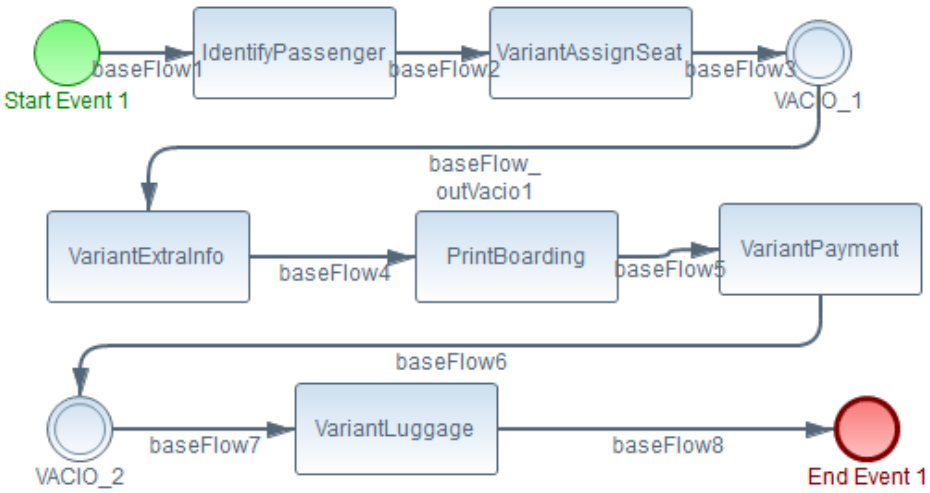


Figura 32: Proceso base

Por la limitación de modelado que posee nuestra solución, no pueden quedar contiguos en el modelo base dos artefactos que participen de diferentes fragment substitution, es por esto que entre VariantAssgingSeat y VariantExtraInfo está el evento VACIO 1. Al ejecutarse la transformación y generar la variante, este artefacto será eliminado. Lo mismo sucede entre VariantPayment y VariantLuggage donde está VACIO 2.

Modelamos como se muestra en figura 33 los posibles reemplazos identificados en un proceso BPMN, que llamamos de replacements.

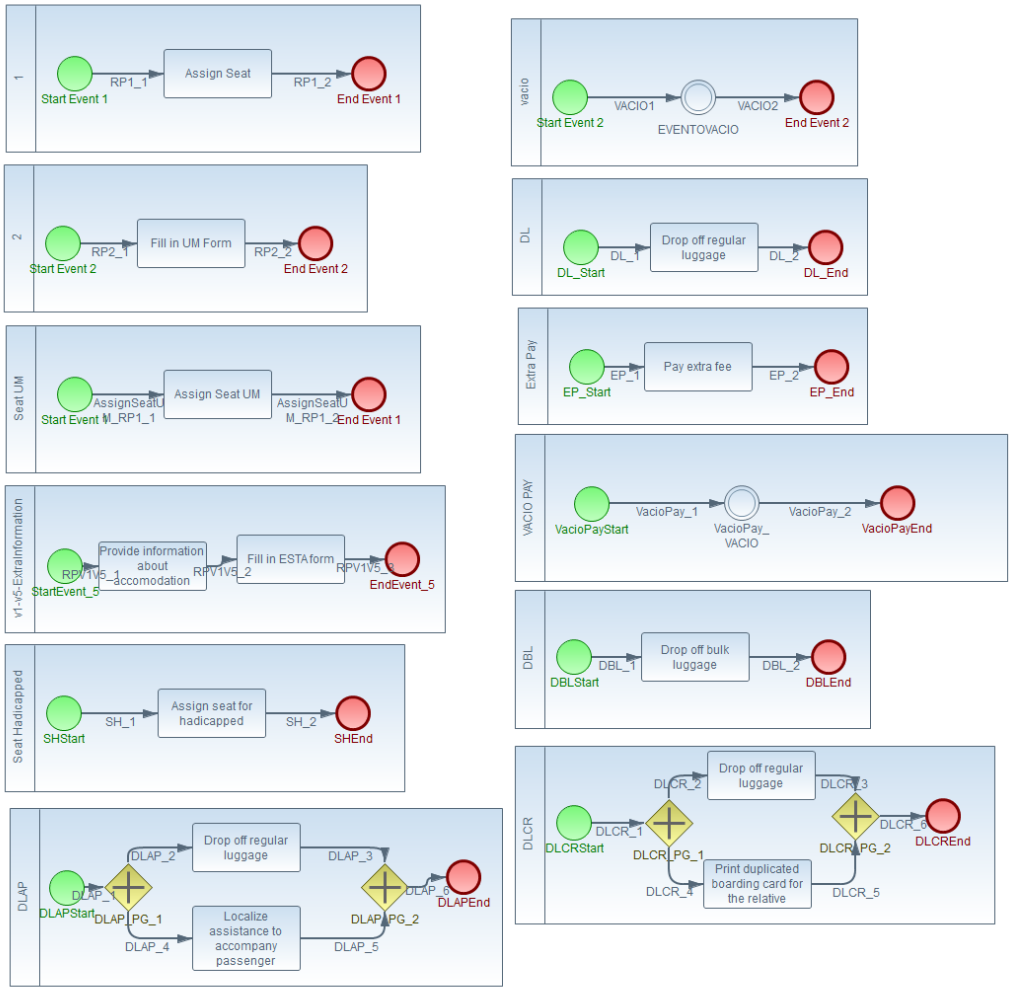


Figura 33: Replacements

Luego de creados el modelo base y el modelo de replacements, continuamos con el modelo de realización BVR, donde creamos los fragment substitution, que definen la correspondencia entre placements del modelo base y replacements del modelo de reemplazo. Cada fragment substitution se corresponde con una variable VSPEC que indica cuando debe ser ejecutado.

En figura 34 mostramos los dos archivos BPMN y el modelo de reemplado BVR con los fragment substitution definidos.

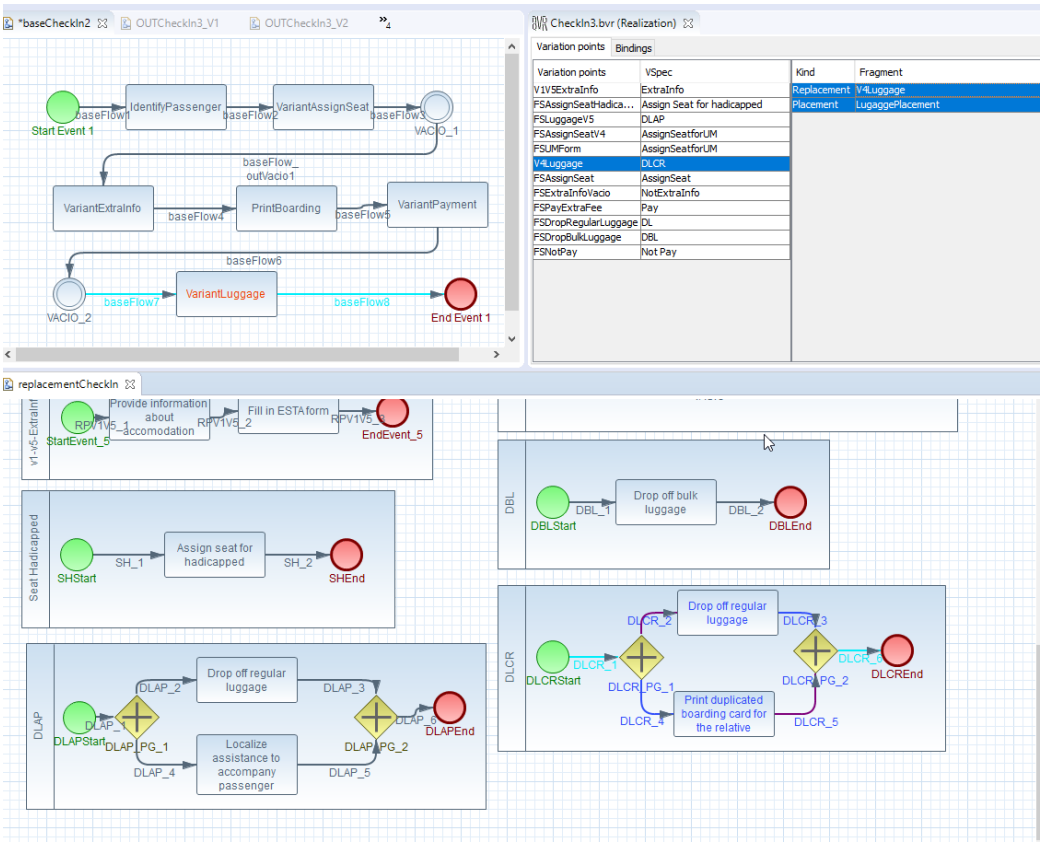


Figura 34: Ejemplo replacements

En figura 35 vemos en detalle el modelo de realización BVR, específicamente el fragment substitution V4Luggage. Este fragment substitution reemplaza el fragmento V4Luggage del modelo base por LugaggePlacement del modelo de reemplazos, cuando al ejecutar la transformación es True el valor de la

variable DLCR del modelo VSPEC BVR.

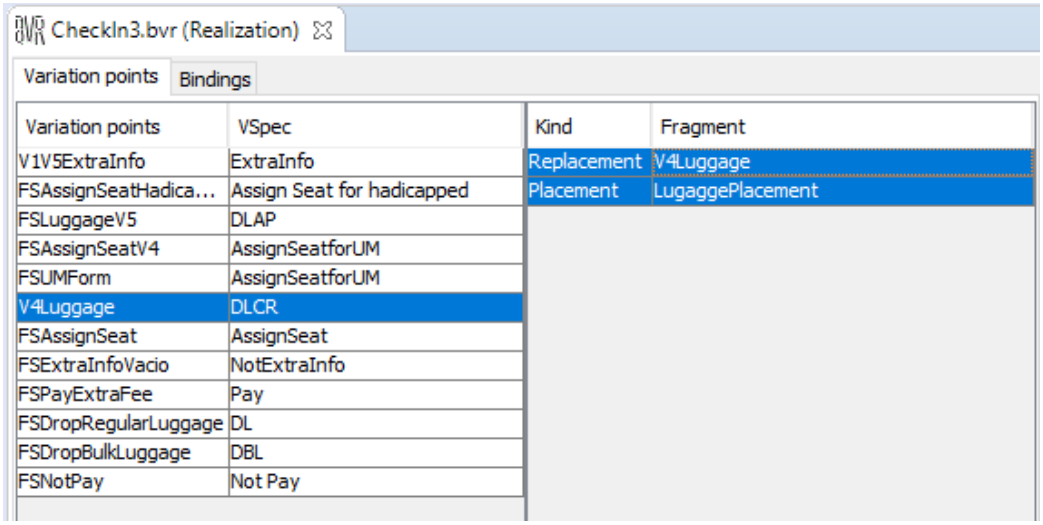


Figura 35: Detalle de realization

Además, para cada fragment substitution se deben definir los bindings entre los elementos de la sustitución, vemos esto en figura 36

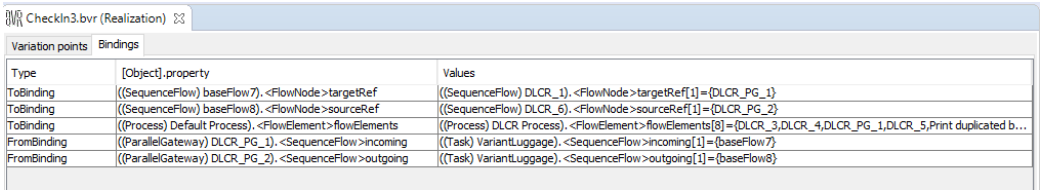


Figura 36: Ejemplo bindings

En el modelo VSPEC del BVR que vemos en figura 37 se define el árbol de las posibles variables, combinando adecuadamente los valores de estas variables en los Resolution models, se obtienen las configuraciones que permiten generar adecuadamente la familia de procesos.

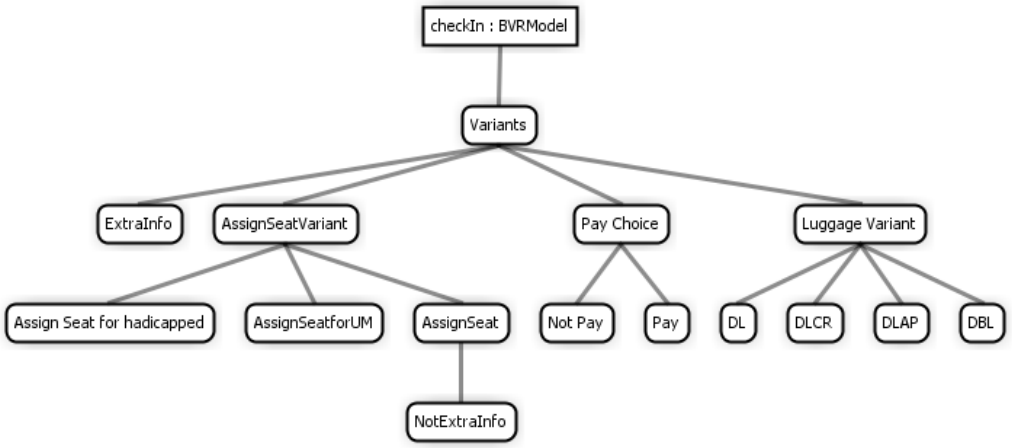


Figura 37: VSpec

Veamos en figura 38 el modelo de resolución BVR que corresponde a la variante 4.

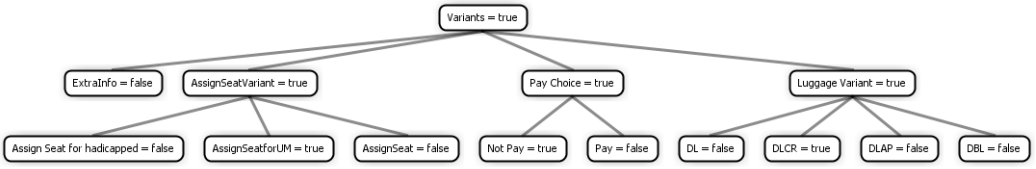


Figura 38: Resolution Variant 4

El resultado de ejecutar la transformación con los (base y replacement) y el BVR con el modelo de resolución recién visto, es el modelo que vemos en figura 39 que representa la variante 4 del caso de estudio.

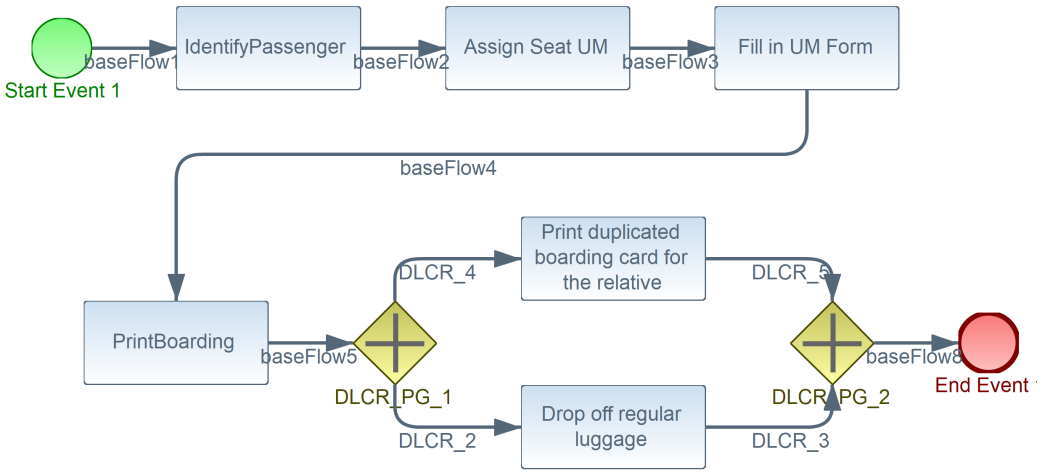


Figura 39: Out variant 4

Obtuvimos entonces la variante 4 de nuestro caso de estudio. En resumen los pasos fueron: creación de modelo base, creación de modelo de reemplazos, creación de modelo de realización BVR, creación de modelo VSPEC BVR, creación de resolution model BVR de la variante 4, ejecución de la transformación con los archivos mencionados.

Para obtener las restantes variantes, se debe ejecutar la transformación con los mismos modelos BPMN, el mismo Resolution model BVR, el mismos VSPEC BVR, y el resolution model BVR que corresponde a la configuración de la variante.

Capítulo 6

Conclusiones

6.1. Conclusiones

El objetivo principal de este proyecto era explorar la generación automática de variantes en familias de procesos de negocio utilizando técnicas de MDE. Para ello, primero se hizo un estudio de las propuestas que aplicaban variabilidad a BPMN.

Nos enfocamos en el análisis de las propuestas basadas en CVL y PROVOP, dado que no necesitan de la modificación del metamodelo del lenguaje al que se aplica la variabilidad, a partir de este análisis decidimos seguir por un enfoque basado en CVL.

Al investigar CVL identificamos el tipo de Punto de Variación "Fragment Substitution", como el más adecuado para aplicar en la mayoría de los casos de BPMN. Elegimos seguir con BVR, un lenguaje basado en CVL, simplificado y con una herramienta (BVR Tool) que permite modelar la variabilidad, mediante modelos BVR a partir de otros modelos. Decidimos utilizar BVR porque mantiene el Fragment Substitution y cuenta con esa herramienta.

Luego de elegir BVR como lenguaje para modelar la variabilidad, continuamos con la generación de las variantes a partir de un modelo BVR y modelos BPMN. Para ello, lo principal fue el desarrollo de una transformación ATL que tiene como entrada esos modelos y como salida un nuevo modelo BPMN.

Se desarrolló un plugin de Eclipse que permite incorporar el modelador de BPMN2 de Eclipse al Plugin de BVR, de esta forma se puede seleccionar gráficamente los fragmentos que van a formar parte del Fragment Substitution. Para la ejecución amigable de la transformación, se desarrolló plugin de Eclipse que permite seleccionar visualmente los modelos de la

ejecución e indicarle el modelo de salida.

Finalmente validamos la propuesta mediante la implementación de un caso de estudio de mediana complejidad.

Como resultado se desarrolló un prototipo de herramienta que permite modelar gráficamente la variabilidad, y generar una familia de variantes de procesos BPMN aplicando técnicas MDE.

Con respecto al lenguaje BVR concluimos que es muy completo, contempla la variabilidad para muchos casos (no está limitado solo a "Fragment Subsitution") y especifica claramente que es lo que hay que hacer y como hacerlo. Una de las ventajas que tiene es que no hay que agregar nuevos elementos para identificar puntos de variación en el modelo base. Si bien eso es una ventaja, también es algo complejo de resolver. Se resolvió esto mediante selección de elementos, gracias al uso de BVR Tool y la capacidad de integrar el BPMN2 Modeler. Esto es, nuestra solución está muy ligada a Eclipse, si se quisiera utilizar BVR en un contexto fuera de Eclipse, no sería un problema sencillo a resolver. También hay que mencionar que BVR Tool no está siendo mantenida, esto implica que para alguna versión de Eclipse podría dejar de funcionar.

En cuanto al uso de ATL podemos concluir que es un lenguaje potente y fue suficiente para resolver nuestro problema. Por otro lado, si bien ATL es un proyecto activo y se siguen liberando versiones, la comunidad no es activa (pueden pasar varios meses sin que nadie responda), esto hizo que algunos problemas nos llevaran mucho más tiempo del que deberían.

En virtud de lo expuesto, concluimos que se lograron todos los objetivos del proyecto.

6.2. Trabajo a futuro

Queda como trabajo a futuro la búsqueda de algún mecanismo para lograr que no sea necesario utilizar eventos intermedios cuando existen dos "Fragment Substitution" contiguos.

Otra mejora sería agregar la capacidad de obtener múltiples variantes en una única ejecución. Actualmente, si se quieren obtener múltiples variantes, es necesario generar tantos modelos BVR como variantes se quiera obtener.

Esos modelos van a ser casi idénticos, lo único que cambia es el modelo de resolución que contienen. Luego habría que ejecutar el plugin tantas veces como modelos BVR haya. Sin embargo, los modelos BVR permiten tener múltiples modelos de resolución en el mismo archivo, por lo que se podría tener un único modelo BVR y a partir de él obtener todas las variantes. Para esta mejora habría que adaptar el plugin para que ejecute la transformación tantas veces como modelos de resolución haya dentro del modelo BVR.

También se vio que es posible agregar una funcionalidad de sustituciones anidadas. El plugin de BVR no tiene restricciones sobre el modelo sobre el cual se definen los fragmentos. Actualmente se definen fragmentos de tipo "Placement" sobre el modelo base y fragmentos de tipo "Replacement" sobre el modelo replacement, pero también se podría definir un fragmento de tipo "Placement" sobre el modelo replacement.

La forma en que está definida la transformación, junto con esa capacidad del plugin, hacen que sea posible anidar sustituciones. Esto es, un fragmento del modelo replacement podría pertenecer a un "Fragment Substitution" en calidad de "Replacement" y a su vez podría pertenecer a otro "Fragment Substitution" en calidad de "Placement". En este caso, si esos "Fragment Substitutions" se resuelven true, ese fragmento no pertenecería al modelo final, también sería sustituído.

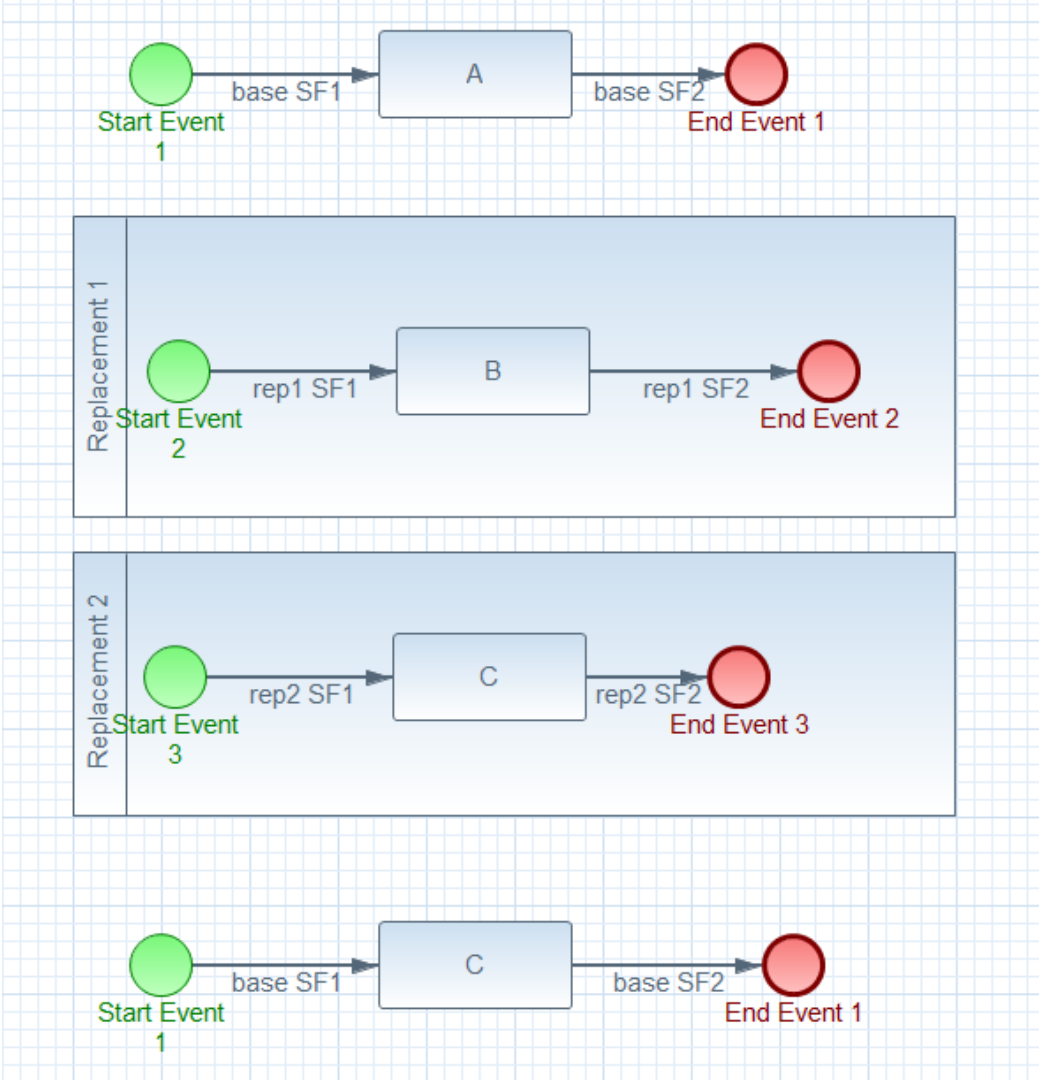


Figura 40: Sustitución anidada

La figura 40 muestra un ejemplo de sustitución anidada. En este ejemplo el primer diagrama es el modelo base, el segundo y tercer diagrama representan el modelo replacements. Se definen 2 "Fragment Substitutions":

- FS(A,B): donde A es el Placement y B el Replacement
- FS(B,C): donde B es el Placement y C el Replacement

El cuarto diagrama muestra el resultado de la sustitución anidada.

Para que esto funcione habría que hacer algunas modificaciones en la transformación:

1. Para determinar si se copia un elemento del modelo replacements habría que chequear que ese elemento no pertenezca a un fragmento de tipo ”Placement” de un ”Fragment Substitution” a aplicar. En el ejemplo, C se copia porque no pertenece a ”Placement” en FS(A,B) ni en FS(B,C), B no se copia porque es Placement en FS(B,C).
2. Habría que hacer alguna funcionalidad que determine el elemento final con el cual se conectarían los SequenceFlow del modelo base. Para el caso del ejemplo, los SequenceFlow deberían conectarse con C en lugar de B (que es lo que indica el FS que tiene a A como ”Placement”).
3. Igual que en el caso anterior, habría que desarrollar alguna funcionalidad que determine los incoming y outgoing del elemento que se copia. En el ejemplo, el incoming de C sería ”base SF1” y el outgoing ”base SF2”.

Al agregar esta funcionalidad también debería haber una restricción que impida que se formen ciclos en las sustituciones.

Referencias

[1] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag, 2007.

[2] Alena Hallerbach, Thomas Bauer, and Manfred Reichert. Capturing variability in business process models: The provop approach. 2010.

[3] Clara Ayora, Victoria Torres, Vicente Pelechano, and Germán H. Alférez. Applying cvl to business process variability management. In *Proceedings of the VARiability for You Workshop: Variability Modeling Made Useful for Everyone*. ACM, 2012.

[4] Emanuel Santos, João Pimentel, Jaelson Castro, Juan Sánchez, and Oscar Pastor. Configuring the variability of business process models using non-functional requirements. Springer Berlin Heidelberg, 2010.

[5] Ignacio Bentancur, Alejandro Brusco, and Nicolás Dinetti. Proyecto de grado de ingeniería en computación. Universidad de la República, 2016.

[6] Fabiana Roldán and Marcela Viera. Proyecto de grado de ingeniería en computación. Universidad de la República, 2016.

[7] Øystein Haugen, Andrzej Wasowski, and Krzysztof Czarnecki. Cvl: common variability language. 2013.

[8] Daniel Calegari and Andrea Delgado. Systematic evaluation of business process management systems: a comprehensive approach. *CLEI Electronic Journal*, 2018.

[9] Object Management Group (OMG). `https://www.omg.org/bpmn/`.

[10] Marcello La Rosa, Wil MP Van Der Aalst, Marlon Dumas, and Fredrik P Milani. Business process variability modeling: A survey. *ACM Computing Surveys (CSUR)*, 2017.

[11] Sami Beydeda and Volker Gruhn. *Model-Driven Software Development*. Springer-Verlag, 2005.

[12] ATLAS INRIA & LINA. Atl: the atlas transformation language. http://www.eclipse.org/atl/documentation/old/ATL_PresentationSheet.pdf.

[13] Eclipse. Eclipse modeling framework (emf). https://www.eclipse.org/modeling/emf/.

[14] F. Fleurey, Haugen, B. Mller-Pedersen, G. K. Olsen, A. Svendsen, and X. Zhang. A generic language and tool for variability modeling. *Technical Report SINTEF A13505*, 2009.

[15] Object Management Group. Metaobject facility. https://www.omg.org/mof/.

[16] Øystein Haugen and Ommund Øgård. Bvr – better variability results. 2014.

[17] OMG. Common variability language (cvl). 2012.

[18] SINTEF. Bvr tool. https://github.com/SINTEF-9012/bvr.

[19] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. 1990.

[20] Tuan Nguyen, Alan Colman, and Jun Han. Modeling and managing variability in process-based service compositions. In *Service-Oriented Computing*. Springer Berlin Heidelberg, 2011.

[21] Markus Döhring and Birgit Zimmermann. vbpmn: Event-aware workflow variants by weaving bpmn2 and business rules. In *Enterprise, Business-Process and Information Systems Modeling*. Springer Berlin Heidelberg, 2011.

[22] Eclipse. Bpmn 2 modeler. https://www.eclipse.org/bpmn2-modeler/.

[23] Clara Ayora, Victoria Torres, Barbara Weber, Manfred Reichert, and Vicente Pelechano. Vivace: A framework for the systematic evaluation of variability support in process-aware information systems. *Information and Software Technology*, 2015.