UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Especificación y configuración automática de familias de procesos

Informe de Proyecto de Grado presentado por

Felipe Gustavo Castellanos Alvarez y Nicolás Eduardo Navascués Soto

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Daniel Calegari
Andrea Delgado

Montevideo, 19 de Diciembre de 2023

Agradecimientos

Agradecemos especialmente a nuestros tutores Dr. Ing. Daniel Calegari y Dra. Ing. Andrea Delgado, quienes cumplieron un rol fundamental en la realización del presente proyecto. Por otro lado, agradecemos infinitamente a todos quienes nos acompañaron tanto a lo largo del desarrollo del presente proyecto, como a lo largo de toda la carrera de grado.

Resumen

Los procesos de negocio se definen como una serie de actividades ejecutadas de manera coordinada con el fin de lograr un objetivo de negocio específico, y pueden ser especificados mediante lenguajes de modelado de tipo imperativo, como BPMN 2.0, o de tipo declarativo, como Declare o Case Management Model and Notation (CMMN). Estos lenguajes difieren en su enfoque y metodología; mientras que los lenguajes imperativos son más prescriptivos, delineando cada paso del proceso de manera secuencial, los lenguajes declarativos ofrecen mayor flexibilidad, permitiendo definir reglas y condiciones sin especificar un flujo exacto.

Dentro de una organización, un mismo proceso puede experimentar variaciones según aspectos inherentes al negocio, tales como los procesos de venta que pueden diferir según el tipo de producto. En general, esto conduce a la formulación de un proceso base compartido, acompañado de las variantes necesarias para adaptarse a las particularidades de cada proceso, conformando así lo que se conoce como una "familia de procesos".

Este proyecto busca, en primer lugar, estudiar las propuestas existentes para el modelado de familias de procesos utilizando lenguajes declarativos, dado que el relevamiento de familias con BPMN ya está realizado. Se realizará una comparación entre estas propuestas, analizando sus mecanismos asociados para asistir a los usuarios en la configuración de variantes. A partir de estos resultados, se propone un nuevo enfoque para representar las mismas implementando una extensión a un lenguaje de modelado existente como lo es CMMN.

Esta extensión, la cual se denomina CMMNext, permite a los usuarios definir puntos de variación en los procesos base para así definir distintas instancias del proceso dependiendo de las necesidades particulares de cada configuración elegida. Se proveen finalmente ejemplos donde se expone el potencial que la misma representa.

Palabras clave: Proceso de negocio, Configurabilidad, Variabilidad, Familia de Procesos, CMMN, Procesos Declarativos

Índice general

Capítulo 1

Introducción

Los procesos de negocio (Weske, 2019) son un conjunto de actividades realizadas en coordinación en un entorno organizacional y técnico, para alcanzar un objetivo del negocio. Los mismos abarcan un amplio espectro, desde procesos completamente predecibles y altamente repetitivos, como los de venta de productos, hasta procesos impredecibles y poco repetitivos como, por ejemplo, los procesos de innovación.

Para los casos de procesos de negocio en los que se desea definir todos los posibles caminos de ejecución y el paso a paso del flujo, se opta por procesos imperativos (Goedertier et al., 2015). Como ejemplo de los lenguajes imperativos se puede mencionar BPMN (*bpmn-estándar*, s.f.), que es una notación gráfica estandarizada diseñada para representar la secuencia de actividades que conforman los procesos de negocio de una organización y los mensajes que fluyen entre los participantes y cada una de las actividades.

Para procesos más impredecibles se suele optar por declarar las restricciones de ejecución que existen entre las actividades, que se puede ver en los procesos declarativos (Goedertier et al., 2015). Un ejemplo de lenguaje declarativo es CMMN (*Case Management Model and Notation*, s.f.), que es una notación gráfica que captura métodos de trabajo basados en manejo de casos que requieren muchas actividades que pueden ser realizadas en un orden impredecible en respuesta a situaciones en constante evolución.

Por otro lado, se define la flexibilidad (Reichert et al., 2012) en el contexto empresarial como la habilidad para adaptarse y reaccionar de forma eficaz a los cambios, sean estos esperados o inesperados. Procesos de negocio que son flexibles tienen la capacidad de adaptarse rápidamente y de manera eficiente a nuevas situaciones, demandas o condiciones, manteniendo al mismo tiempo la calidad, eficiencia y efectividad de sus operaciones.

En relación con esto, la variabilidad se refiere a la capacidad de un proceso de negocio de adaptarse y cambiar para satisfacer diferentes requisitos o situaciones (La Rosa et al., 2013). Se puede abordar de dos formas: con un enfoque basado en restricciones, que modela todas las variantes potenciales en un proceso personalizable; y con uno basado en extensiones, que comienza con un modelo general y agrega extensiones específicas para crear diferentes variantes del proceso.

Extendiendo estos conceptos, una familia de procesos (Reichert et al., 2012) (co-

nocidos también como procesos configurables) se refiere a un grupo de procesos de negocio y sus variantes, las cuales varían según el contexto específico. Esta forma de representación es beneficiosa ya que evita la necesidad de modelar cada variante de manera individual, lo cual a menudo conduce a duplicación y complica el mantenimiento de elementos.

Las propuestas que han surgido para el modelado de familias de procesos se han enfocado principalmente en procesos imperativos (lo presentado en (Delgado et al., 2022) es un ejemplo de esto), y es por esta razón que surge la motivación por modelar la configurabilidad y poder representar familias de procesos en lenguajes declarativos.

Por otro lado, la ingeniería dirigida por modelos (MDE) destaca el modelado como actividad central en el ciclo de vida del software, proponiendo la generación de modelos y su transformación automática (o semi automática). Es especialmente útil en la representación de las mencionadas familias de procesos, asi como también en la generación de variantes a través de definiciones de configuraciones y transformaciones que utilicen los modelos como parámetros.

El objetivo general de este proyecto es definir y evaluar modelos para la especificación de familias de procesos declarativos y asistir a la generación automática de sus variantes, a través de técnicas de Ingeniería Dirigida por Modelos.
Los objetivos de este proyecto son:

- Estudiar propuestas sobre variabilidad de procesos declarativos y mecanismos de asistencia a usuarios para la configuración de variantes.
- Definir una estrategia para asistir en la configuración de variantes de una familia.
- Experimentar con transformaciones de modelos para la generación automática de variantes a partir de la especificación de una familia y la definición asistida de sus configuraciones.
- Evaluar la aplicabilidad de la propuesta a través de casos de estudio.

Estos aspectos se abordan en lo que resta del informe. En el capítulo 2, se hace énfasis en el marco teórico del proyecto, detallando los conceptos importantes para el entendimiento de la temática. Luego, en el capítulo 3, se muestra la revisión sistemática realizada, mostrando las investigaciones realizadas y las conclusiones sacadas. El próximo paso, en el capítulo 4 se detalla la implementación realizada, mostrando las distintas etapas y tecnologías usadas para completar la extensión. La etapa siguiente se puede ver con el capítulo 5 que muestra los distintos casos de aplicación realizados, permitiendo mostrar el potencial de la solución desarrollada en comparación con los ejemplos de artículos publicados de otras familias de procesos declarativos.

Finalmente, se puede ver en Conclusiones y Trabajo Futuro un conjunto de conclusiones luego de terminado el proyecto y sugerencias de trabajo a futuro para posibles mejoras teniendo en cuenta los objetivos globales de la problemática.

Capítulo 2

Marco teórico

En este capítulo se presenta la base teórica necesaria para la comprensión del proyecto. Se hace en primer lugar una presentación de MDE (Model-driven engineering - Ingeniería dirigida por modelos), posteriormente se introducen conceptos como procesos de negocio y sus tres distintos tipos, y se presentan BPMN y CMMN como ejemplos de lenguajes existentes para la representación de los mismos. A continuación, se desarrolla sobre variabilidad, flexibilidad y familias de procesos de negocio. Se presenta BPMNext como extensión de BPMN para permitir modelar variabilidad, y se plantea una interrogante que determinará el curso del presente informe.

2.1. Ingeniería dirigida por modelos

Se desarrollan a continuación definiciones y ejemplos sobre el concepto de "Ingeniería Dirigida por Modelos", lo cual será fundamental para comprender el desarrollo del presente proyecto.

2.1.1. Definición

La Ingeniería Dirigida por Modelos (Model Driven Engineering, MDE) se presenta como un paradigma de Ingeniería de Software que destaca el modelado como la actividad central en el ciclo de vida de un sistema de software, abordando desde su construcción hasta el mantenimiento y la ingeniería inversa, entre otras fases. Este enfoque propone la generación de modelos (abstracciones) que representan diversos aspectos del sistema, seguida por la transformación (semi)automática de dichos modelos. (Kent, 2002)

2.1.2. Modelado

Un lenguaje de modelado es una herramienta que permite a los diseñadores especificar los modelos para sus sistemas (Brambilla et al., 2012). Estas herramientas posibilitan la definición de una representación concreta de un modelo conceptual, pudiendo consistir en representaciones gráficas, especificaciones textuales o ambas. En cualquier caso, están formalmente definidos y exigen a los diseñadores cumplir con su sintaxis durante el modelado. Se pueden identificar dos grandes clases de lenguajes:

Lenguajes Específicos del Dominio Los DSL (por sus siglas en inglés) son lenguajes diseñados específicamente para un dominio, contexto o empresa en particular, con el fin de facilitar la tarea de aquellas personas que necesitan describir cosas en ese dominio (Brambilla et al., 2012). Si el lenguaje está dirigido al modelado, también puede denominarse Lenguaje de Modelado Específico del Dominio (DSML). Los DSLs han sido ampliamente utilizados en informática incluso antes de que existiera el acrónimo; ejemplos de lenguajes específicos del dominio incluyen HTML para el desarrollo de páginas web, y SQL para consultas a bases de datos, entre otros.

Lenguajes de Modelado de Propósito General Representan herramientas que pueden aplicarse a cualquier sector o dominio con fines de modelado. Un ejemplo típico de este tipo de lenguajes es la suite de lenguajes UML, o lenguajes como redes de Petri o máquinas de estados.

2.1.3. Metamodelado

Un paso natural posterior a la definición de modelos es representar los modelos mismos como "instancias" de modelos más abstractos. Por lo tanto, de la misma manera en que se define un modelo como una abstracción de fenómenos en el mundo real, puede definirse un metamodelo como otra abstracción que destaca propiedades del modelo en sí (Brambilla et al., 2012). En un sentido práctico, los metamodelos constituyen básicamente la definición de un lenguaje de modelado, ya que proporcionan una manera de describir toda la clase de modelos que pueden representarse mediante ese lenguaje. Por lo tanto, se pueden definir modelos de la realidad, luego modelos que describen modelos (llamados metamodelos) y recursivamente modelos que describen metamodelos (llamados meta-metamodelos). Se presenta un ejemplo de lo mencionado en 2.1, donde se aprecia que los meta-metamodelos pueden definirse basándose en sí mismos, y por lo tanto, generalmente no tiene sentido ir más allá de este nivel de abstracción.

2.1.4. Transformaciones

Las transformaciones son una entidad fundamental en lo que respecta a la Ingeniería Dirigida por Modelos. Se definen a nivel de metamodelo y luego se aplican a nivel de modelo, sobre modelos que se ajustan a esos metamodelos. Por lo tanto, la transformación se realiza entre uno (o más) modelos fuente, y uno (o más) modelos objetivo, pero en realidad se define en los respectivos metamodelos (Brambilla et al., 2012) (Molina et al., 2013).

Para comprender estos conceptos, se presenta el siguiente ejemplo: Supongamos que queremos transformar datos de una hoja de cálculo (Modelo A) en un gráfico interactivo (Modelo B). En este caso, se tendrían dos metamodelos: uno para describir la estructura y datos de la hoja de cálculo, y otro para definir la representación gráfica interactiva.
La transformación se realizaría entonces a nivel del metamodelo, especificando cómo los datos en la hoja de cálculo se traducen en elementos interactivos en el gráfico. Por ejemplo, podría definirse que las columnas y filas de la hoja de cálculo se representen como ejes en el gráfico, y que los datos numéricos se muestren como puntos o barras. Al aplicar la transformación a una instancia específica de la hoja de cálculo (Modelo

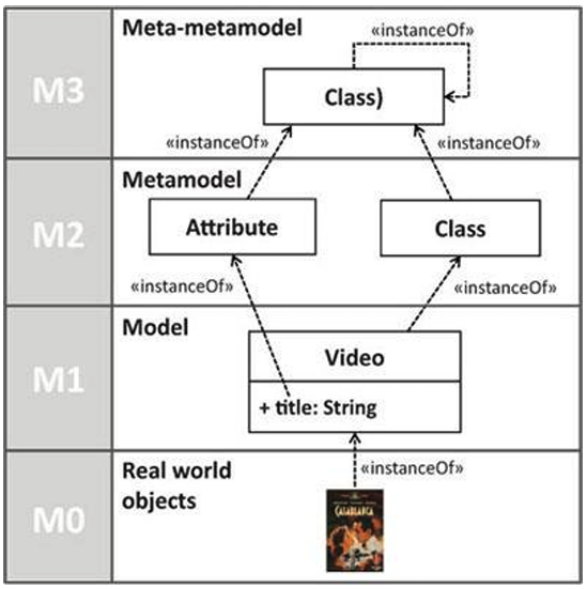


Figura 2.1: Modelos, metamodelos y meta-metamodelos - extraído de (Brambilla et al., 2012)

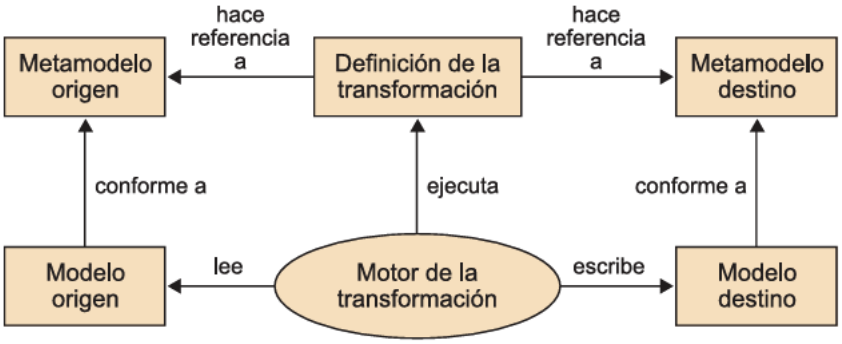


Figura 2.2: Participantes de una transformación de modelos - extraído de (Molina et al., 2013)

A), se obtendría una representación gráfica interactiva correspondiente (Modelo B). Para hacerlo de manera más general, se podría crear una transformación que considere diversas estructuras y tipos de datos en la hoja de cálculo, traduciéndose consistentemente a la representación gráfica interactiva. Esta definición a nivel de metamodelo permitiría aplicar la transformación a diferentes conjuntos de datos en hojas de cálculo para generar visualizaciones interactivas correspondientes.

Las reglas de transformación pueden definirse mediante diferentes enfoques; puede ser escrita manualmente desde cero por un desarrollador, o puede ser especificada a partir de una existente. Alternativamente, las transformaciones mismas pueden generarse automáticamente a partir de reglas de mapeo de nivel superior entre modelos. Esta técnica se basa en dos fases:

- Definir un mapeo entre elementos de un modelo y elementos de otro modelo (mapeo de modelos o entrelazado de modelos).
- Automatizar la generación de las reglas de transformación reales mediante un sistema que recibe como entrada las definiciones de los dos modelos y el mapeo entre ellos, y produce las transformaciones.

Esto permite que los desarrolladores se centren en los aspectos conceptuales de las relaciones entre modelos y luego deleguen la producción de las reglas de transformación.

Por otro lado, existen muchas formas de clasificar las transformaciones entre modelos. Particularmente para el presente proyecto es de interés desarrollar en la clasificación respecto al tipo de modelo de destino de dicha transformación, existiendo dos tipos denominados M2M (del inglés Model to Model, es decir modelo a modelo) las cuales generan nuevos modelos a partir de modelos de entrada (que pueden conformar con un mismo metamodelo, o no), y M2T (del inglés Model to Text, modelo a texto), las cuales generan texto o archivos de texto a partir de modelos de entrada (Molina et al., 2013).

Existen varias herramientas para trabajar en ambos tipos de transformaciones. En el caso del presente proyecto, se trabajó con la herramienta ATL (*ATL*, s.f.) para las transformaciones M2M, y la herramienta Acceleo (*Acceleo*, s.f.) para las M2T. Ambas forman parte de las extensiones disponibles para el IDE Eclipse Modelling Tools (*Eclipse Modelling Tools*, s.f.), sobre el cual se trabajó en esta oportunidad.

2.2. Procesos de Negocio

Un proceso de negocio (también conocido como BP por su traducción al inglés "Business Process") consiste en un conjunto de actividades que se realizan de manera coordinada en un entorno organizacional y técnico (Weske, 2019). Estas actividades realizan conjuntamente un objetivo empresarial. Cada proceso de negocio es llevado a cabo por una única organización, pero puede interactuar con procesos de negocio realizados por otras organizaciones.

La gestión de procesos de negocio incluye conceptos, métodos y técnicas para el diseño, administración, configuración, ejecución y análisis de los BPs (Weske, 2019). La base de la misma es la representación explícita de los procesos, con sus actividades y las restricciones de ejecución entre ellas; una vez que se definen los procesos de negocio, estos pueden estar sujetos a análisis y mejoras.

Cada proceso de negocio cuenta con su ciclo de vida, el cual está compuesto por cuatro fases relacionadas entre sí y dispuestas de forma cíclica definiendo dependencias lógicas, como puede observarse en la figura 2.3. Las mismas no necesariamente implican un orden temporal, ya que algunas actividades pueden ocurrir concurrentemente, o en todas las fases.

El ciclo se inicia en la etapa de diseño y análisis, donde se realizan investigaciones acerca del proceso de negocio y su contexto organizacional y tecnológico. Con base en esto, se crea una representación visual del modelo, lo que facilita la comunicación entre las partes involucradas. Una vez que se ha desarrollado un modelo preliminar del proceso, el enfoque se centra en validar su exactitud y eficacia. Esta validación se

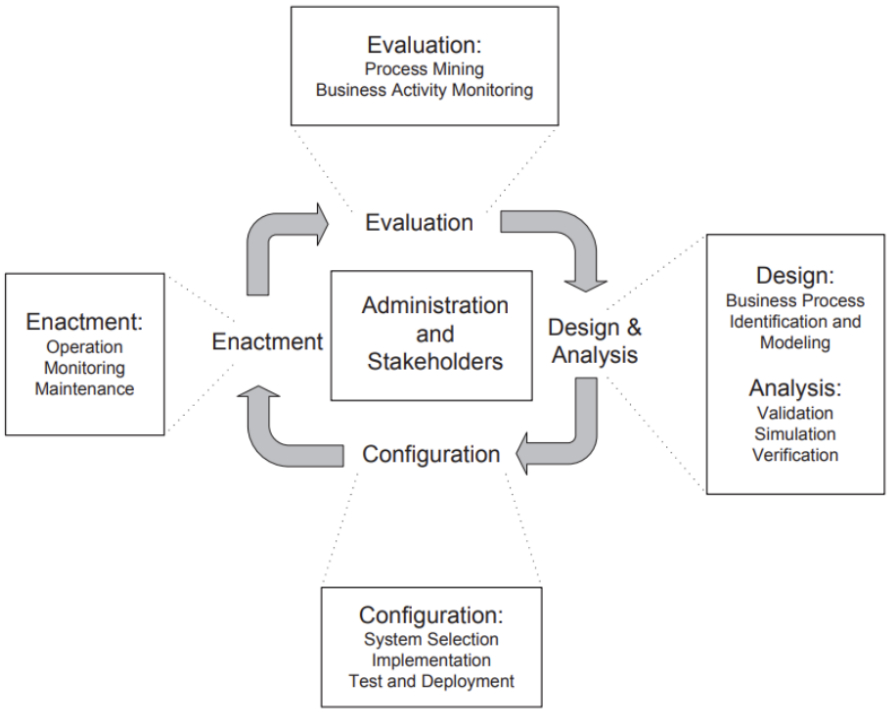


Figura 2.3: Ciclo de vida de los procesos de negocio - extraído de (Weske, 2019)

logra al cotejar detalladamente el modelo con diversos escenarios del BP, asegurando que el modelo sea una representación fiel de los diversos aspectos y posibles situaciones que el proceso podría enfrentar en la práctica.

Una vez que se diseña y valida el modelo del BP, se procede con su implementación, la cual puede realizarse de diversas formas. Durante la fase de configuración, se elige la plataforma para la implementación y se ajusta el sistema BPM conforme al entorno organizacional y al BP a gestionar. Estas configuraciones incluyen interacciones entre empleados y el sistema, así como la integración con sistemas de software existentes.

Tras la configuración, se efectúa una fase de pruebas para la implementación del BP. Se emplean métodos tradicionales de pruebas a nivel de actividades para asegurarse de que el sistema se comporte según lo previsto. Pruebas de integración y rendimiento también son cruciales a nivel de proceso para detectar posibles fallos durante la ejecución. Cuando las pruebas concluyen con éxito, el sistema se instala en el entorno correspondiente.

En la tercera etapa, el BP puede ser activado. En esta fase, el sistema BPM controla activamente la ejecución del BP conforme a lo definido en su modelo. El sistema BPM debe proporcionar una función de supervisión que muestre el estado de las instancias en ejecución, lo cual es esencial para brindar información precisa a los usuarios que consulten sobre el progreso de sus procesos. Los sistemas BPM suelen ofrecer un sistema de monitoreo para indicar estados de las instancias del BP y poder supervisar

fácilmente la correcta ejecución de los mismos.

Durante la ejecución, se acumula información en un archivo de registro que documenta los eventos ocurridos. Esta información resulta valiosa para la evaluación del proceso, que se lleva a cabo en la última fase del ciclo. En esta fase, se utiliza la información recopilada para evaluar y mejorar los modelos y las implementaciones de los BPs. Mediante técnicas de análisis de procesos, los registros se examinan para determinar la calidad de los modelos y la adecuación del entorno de ejecución. Por ejemplo, el seguimiento de actividades podría identificar retrasos significativos debido a limitaciones de recursos.

Un lenguaje de modelado de procesos ampliamente utilizado es BPMN, siglas para "Business Process Model Notation", el cual es un lenguaje estándar definido por la OMG (Object Management Group). BPMN permite representar procesos de naturaleza imperativa, es decir describir explícitamente la secuencia de pasos y/o acciones que componen la ejecución de un BP, lo que permite controlar el flujo del proceso en función de eventos y condiciones específicas. Para ello, utiliza distintos elementos que permiten conformar una representación visual del proceso, tal como puede apreciarse en el ejemplo de la figura 2.4.

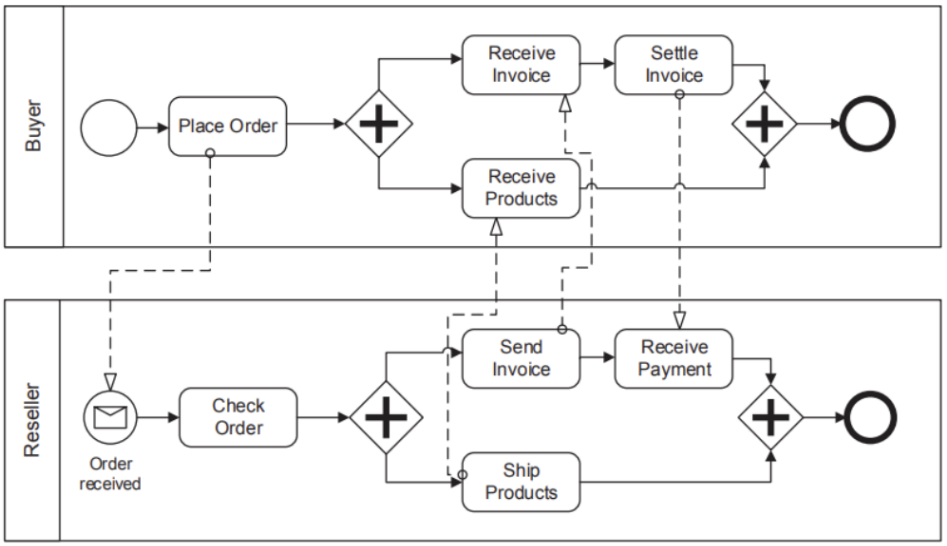


Figura 2.4: Ejemplo de procesos de negocio representados en BPMN - extraído de (Weske, 2019)

Se presenta a continuación un breve resumen de los elementos que se utilizan en BPMN según lo explicado en (Weske, 2019) y en el estándar del lenguaje definido por la OMG, los cuales se representan en la figura 2.5.

Los objetos de flujo (flow objects) son los elementos básicos de los procesos de negocio e incluyen eventos, actividades y gateways.

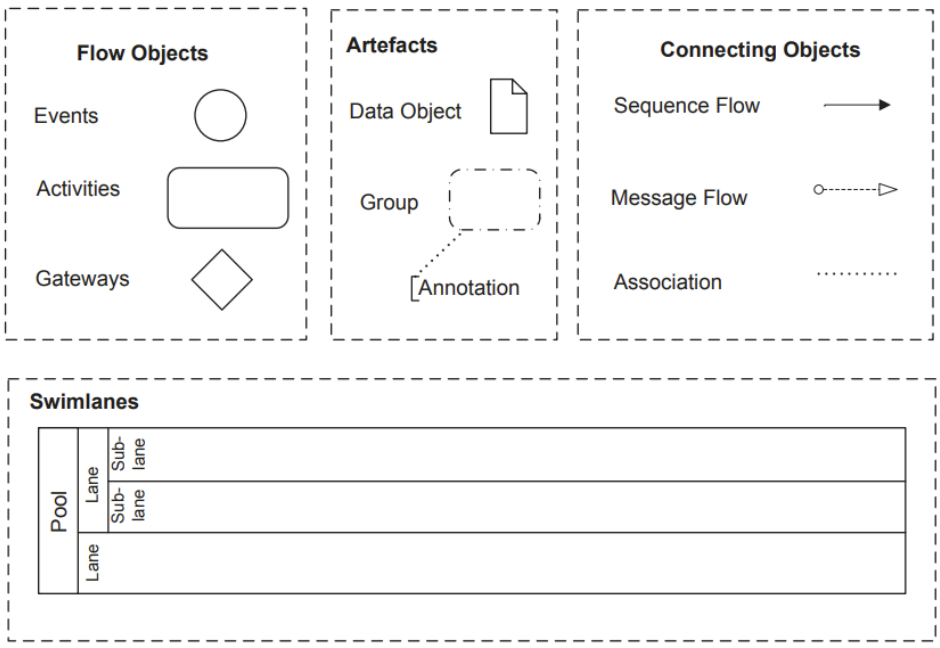


Figura 2.5: Elementos de BPMN, clasificados por categoría - extraído de (Weske, 2019)

Eventos Los eventos representan la ocurrencia de estados relevantes en el mundo real para los procesos de negocio y, en general, cualquier evento significativo que suceda. Se identifican como círculos con centros en blanco, y existen tres tipos según su influencia en el flujo. Los eventos se clasifican en: eventos de inicio, que señalan el inicio de un proceso o coreografía; eventos intermedios, que ocurren entre un evento de inicio y un evento de fin y afectan el flujo sin iniciar o finalizar directamente un proceso; y eventos de fin, que indican la conclusión de un proceso.

Actividad Es el término genérico utilizado para representar las acciones que un actor debe llevar a cabo en un proceso. Estas actividades se visualizan mediante rectángulos con bordes redondeados y pueden clasificarse como atómicas o no atómicas. Aquellas actividades que son atómicas se denominan “Tasks” (o Tareas), mientras que las no atómicas son conocidas como “Sub-Process” (o Subprocesos).

Gateway Es una figura empleada para gestionar la divergencia y convergencia de flujos de secuencia en un proceso. Su función principal es facilitar la creación de ramificaciones, bifurcaciones, fusiones y uniones de caminos dentro del flujo del proceso.

En conjunto, estos elementos constituyen los componentes esenciales para modelar y comprender los procesos de negocio.

Por otro lado se encuentran los artefactos (artefacts). Estos se utilizan para mostrar información adicional sobre un proceso de negocio que no es directamente relevante para el flujo de secuencia o el flujo de mensajes del proceso. Los artefactos admi-

tidos incluyen objetos de datos, grupos y anotaciones. Cada artefacto puede asociarse con elementos de flujo. Los artefactos tienen únicamente propósitos informativos, de manera que no influyen en la semántica de ejecución del proceso.

Objetos de datos Los "data objects" se representan simplemente por un nombre, y su estructura interna no puede definirse en BPMN. El propósito principal de los artefactos de objetos de datos es documentar la información utilizada en el proceso. A través de aristas de asociación dirigida, el modelador puede representar el hecho de que un objeto de datos es leído o escrito por una actividad del proceso. Los objetos de datos pueden representar documentos en papel, información electrónica y artefactos físicos como productos enviados..

Anotación Las "Annotations" documentan aspectos específicos del proceso de negocio en forma textual. El texto se asocia gráficamente con el objeto en el diagrama del proceso de negocio que la anotación explica.

Grupo Los "groups" son artefactos utilizados para agrupar elementos de un proceso. Estos no tienen un significado formal, sino que simplemente cumplen propósitos de documentación. Los grupos pueden abarcar lanes e incluso pools.

Los "objetos de conexión" (connecting objects) conectan objetos de flujo, swimlanes o artefactos.

Flujo de secuencia Un "Sequence Flow" se utiliza para indicar el órden en el que se ejecutarán las actividades en un proceso.

Flujo de mensaje Un "Message Flow" representa el flujo de mensajes entre dos participantes que están listos para enviar y recibir mensajes entre sí. Para visualizar estos participantes, se utilizan dos "Pools" separados dentro de un diagrama de colaboración.

Asociación La "Association" es un tipo específico de objeto de conexión que se utiliza para vincular artefactos a elementos en los diagramas de procesos de negocio.

Por último, se presentan los objetos de tipo "swimlane":

Pool Un "pool" es la representación gráfica de un participante en una colaboración. Cumple la función tanto de swimlane como de contenedor gráfico para segmentar un conjunto de actividades respecto a otros "pools" dentro del mismo contexto visual.

Lane Una "lane" es una sub-partición dentro de un proceso (que puede estar incluido en un "pool"). Estas se emplean para organizar y categorizar actividades, proporcionando una estructura visual que ayuda a clarificar la responsabilidad y agrupación de las distintas acciones dentro de un contexto específico.

Existen otros lenguajes de modelado más allá de BPMN, como ser CMMN (Case Management Model and Notation) (*What is Case Management Model and Notation (CMMN)*, s.f.) (Zensen et al., 2018) (*Case Management Model and Notation*, s.f.). CMMN es una notación gráfica que captura métodos de trabajo basados en manejo

de casos que requieren muchas actividades que pueden ser realizadas en un orden impredecible en respuesta a situaciones en constante evolución. Expande lo que puede ser modelado por otros enfoques tradicionales, y está presente en procesos de negocio que son difíciles de modelar y planificar.

Es por esto que CMMN se utiliza para representar procesos de naturaleza declarativa, que a diferencia de los procesos imperativos, se enfocan en especificar las restricciones y objetivos del proceso, sin detallar los pasos específicos necesarios para lograrlos. (Goedertier et al., 2015) Utilizando un enfoque centrado en eventos y el concepto de un archivo de caso, CMMN amplía los límites de lo que se puede modelar con BPMN, incluyendo esfuerzos de trabajo menos estructurados y aquellos impulsados por trabajadores del conocimiento. Al combinar BPMN y CMMN, los usuarios pueden abarcar un espectro mucho más amplio de métodos de trabajo. Se presenta un ejemplo en la figura 2.7

Se presentan a continuación los principales elementos de CMMN, los cuales se aprecian en la figura 2.6.

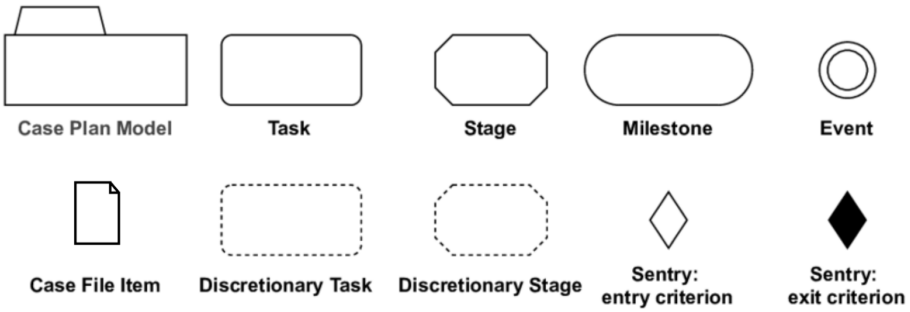


Figura 2.6: Elementos de CMMN - extraído de (Jalali, 2021)

Modelo de plan de caso Un caso en CMMN representa una instancia específica de un caso en el mundo real. Puede tener varias instancias de "Case Plan Model" asociadas, las cuales son el contenedor principal para organizar y estructurar el modelo de caso. Contienen todos los elementos que se definirán a continuación, y representan la lógica del flujo de trabajo del caso.

Tarea Las "tasks" representan unidades de trabajo que deben completarse dentro del caso. Pueden incluir actividades manuales, automatizadas o decisiones específicas.

Etapa Los "stages" son unidades organizativas dentro del modelo de plan de caso. Pueden contener tareas, eventos y otras etapas. Las etapas ayudan a estructurar y organizar las actividades dentro de un caso.

Hito Los "milestones" son un hito o punto de referencia significativo en la ejecución de un caso. Se utilizan para marcar eventos cruciales que afectan el progreso del caso y pueden tener condiciones asociadas que determinan cuándo se considera que se ha alcanzado ese hito.

Evento Los "events" representan eventos que pueden ocurrir durante la ejecución del caso. Pueden ser eventos de inicio, intermedios o de fin, y capturan cambios de estado o condiciones relevantes.

Archivo de caso Los "case file items" pueden representar información de cualquier naturaleza, desde no estructurada hasta estructurada, y desde simple hasta compleja. Esta información puede definirse en base a cualquier "lenguaje" de modelado de información, y pueden ser de cualquier tipo (por ejemplo archivos XML o bases de datos).

Elementos discrecionales Los "discretionary items" tienen la particularidad de que las instancias de los mismos pueden ser planificados a la discreción de un trabajador durante el proceso de planificación. Es decir, que pueden ser o no tenidos en cuenta en un proceso CMMN por el trabajador que lleve a cabo el mismo.

Centinelas Las "sentries", tanto de entrada como de salida, son condiciones que se evalúan para determinar si se debe activar o desactivar una transición de flujo. Ayudan a modelar la lógica de control dentro del caso, permitiendo modelar restricciones en el flujo del proceso.

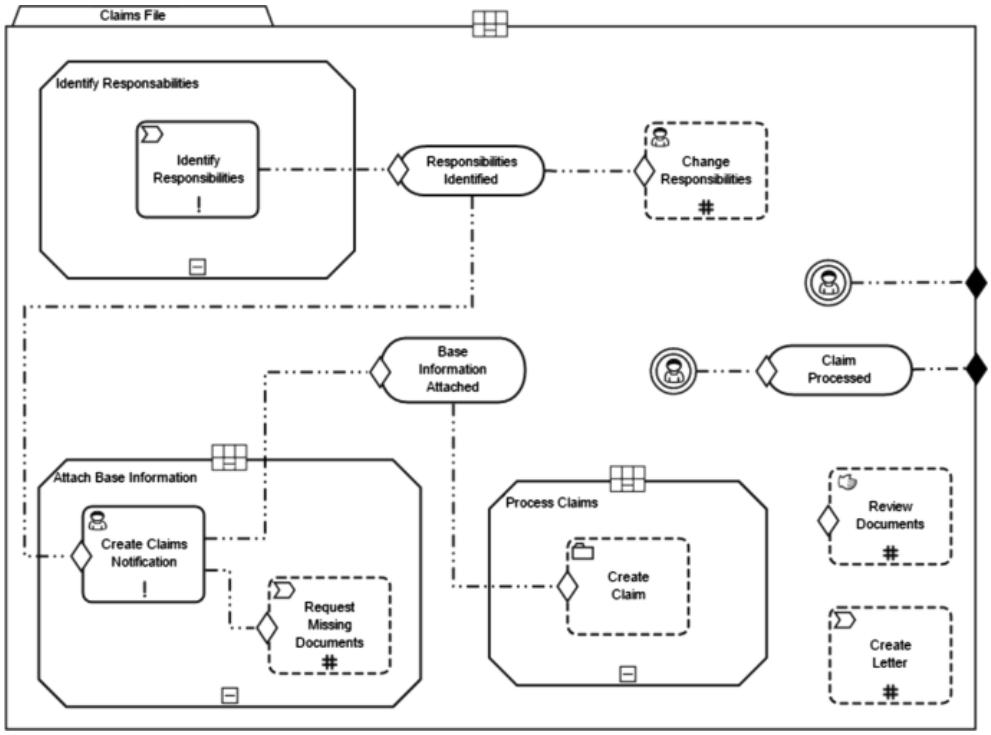


Figura 2.7: Ejemplo de proceso de negocio representado en CMMN - extraído de (Hasić et al., 2017)

Un ejemplo más de lenguajes que representan procesos de negocio declarativos es Declare. Este es un lenguaje con una notación gráfica fácil de usar, el cual describe los procesos a través de restricciones que no deben ser violadas durante la ejecución del

proceso (un ejemplo de esto puede apreciarse en 2.8. Estas restricciones se expresan mediante fórmulas lógicas temporales y representan patrones de comportamiento. Un modelo Declare se compone de un conjunto de restricciones, que son fórmulas FLTL (Lógica Temporal de Rastreo Finito) sobre eventos. Cada modelo Declare tiene un lenguaje, que es el conjunto de todas las secuencias de eventos que satisfacen todas las restricciones del modelo.

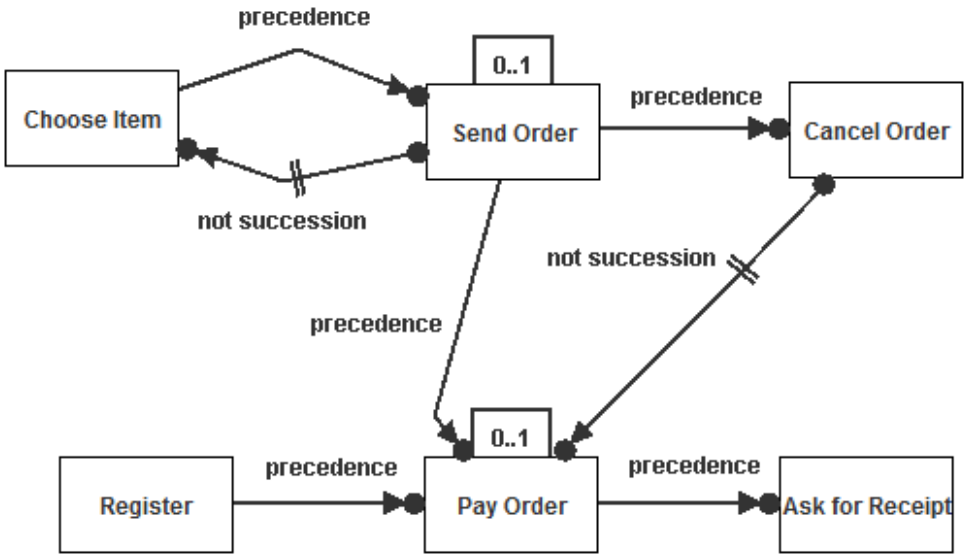


Figura 2.8: Ejemplo modelado en Declare - extraído de (Bernardi et al., 2012)

Finalmente, si bien no es parte del presente proyecto estudiar los mismos, cabe mencionar que existe otro tipo de procesos denominado "procesos híbridos", los cuales combinan elementos de los modelos imperativos y declarativos. En este enfoque, se especifican las restricciones y objetivos del proceso, así como una secuencia de acciones específicas para lograrlos.

2.3. Familias de Procesos de Negocio

Para comprender lo que se presentará en esta sección, resulta fundamental introducir las diferentes fases que conforman un proceso: el tiempo de diseño (design-time), el tiempo de configuración (customization-time) y el tiempo de ejecución (run-time). Se denomina "tiempo de diseño" a la etapa en la cual se da forma al modelo de procesos personalizable. Cada elección de diseño realizada en esta fase impactará en toda la familia de procesos que conforman el modelo, incidiendo en las distintas instancias que se generen a partir del mismo.

Por otro lado, se encuentra la fase de "tiempo de configuración". En esta etapa se toma en consideración el modelo de procesos personalizable creado durante el tiempo de diseño, y se procede a personalizarlo para obtener variantes específicas de procesos. El modelo de procesos personalizado define una variante de proceso que está lista para ser activada. Las elecciones realizadas en esta etapa de personalización solo influyen en dicha variante particular.

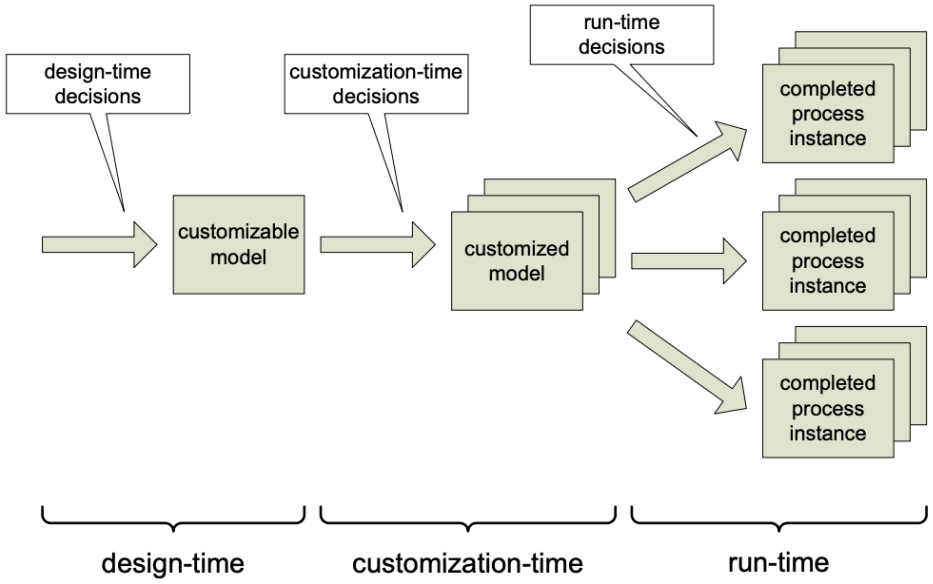


Figura 2.9: Fases principales de toma de decisiones relacionadas con BPs personalizables - extraído de (La Rosa et al., 2013)

Finalmente, se llega a la fase denominada "tiempo de ejecución"; aquí es cuando el modelo de proceso personalizado se pone en marcha. Para cada uno de estos procesos se toman decisiones durante la ejecución, las cuales afectarán únicamente a una instancia específica del proceso en cuestión. En esta etapa, el modelo es ejecutado como si hubiese sido modelado independientemente.

La flexibilidad (La Rosa et al., 2013) se refiere a la capacidad de adaptarse y responder de manera eficiente a cambios, tanto planificados como imprevistos, en el entorno empresarial. Los procesos de negocio flexibles pueden ajustarse de manera ágil y efectiva a nuevas circunstancias, requisitos o condiciones sin comprometer la calidad, la eficiencia o la efectividad de la operación. Esta flexibilidad puede manifestarse de diversas formas, como la capacidad de reconfigurar la secuencia de tareas, incorporar nuevas reglas o procedimientos, cambiar los roles y responsabilidades de los participantes y ajustar los flujos de trabajo para abordar necesidades cambiantes. La flexibilidad en los procesos de negocio es por ejemplo valiosa en un entorno empresarial dinámico, donde las demandas del mercado, las regulaciones y las expectativas de los clientes pueden cambiar de manera rápida y constante.

La variabilidad (La Rosa et al., 2013) por otro lado, tiene la posibilidad de ser representada mediante dos enfoques distintos: uno basado en restricciones, que implica contemplar todas las conductas de las diversas variantes potenciales dentro de un modelo de procesos personalizable (la personalización se logra al limitar el comportamiento del propio modelo de procesos personalizable), y otro basado en extensiones, donde se representa los comportamientos generales o más compartidos por la mayoría de las variantes del proceso, y luego se agregan extensiones en cada instancia particular para así lograr una variante del proceso original.

Existe una clara diferencia entre la flexibilidad y la variabilidad en los procesos de negocio, y esta radica en que tal como podrá entenderse a través de sus definiciones, la primera está asociada a decisiones tomadas en la fase de "tiempo de ejecución", mientras que la segunda engloba decisiones tomadas durante las fases de "tiempo de diseño" y "tiempo de configuración".

Se obtiene una familia de procesos cuando existen procesos que tienen ciertos elementos en común, a la vez que otros dependen del contexto. Si bien hay un proceso base común a todos los integrantes de una familia de procesos de negocio, el curso específico de cada proceso va a depender de qué variante se selecciona en los distintos puntos de variación definidos en la familia. Estos procesos se los denomina como variables o configurables.

Es fundamentalmente en la fase de "tiempo de configuración" donde la variabilidad entra en juego, siendo donde el usuario debe definir una configuración deseada para generar un proceso ajustado a sus necesidades partiendo de un proceso base. Un ejemplo donde se permite trabajar en ese sentido es BPMNext (Delgado et al., 2017), la cual es una extensión implementada sobre BPMN para permitir el modelado de familias de procesos de naturaleza imperativa (Goedertier et al., 2015). Esencialmente lo que permite es extender la funcionalidad original de BPMN agregando elementos que representen puntos de variación, y variantes que sean quienes tomen el lugar de estos puntos de variación siguiendo una determinada configuración elegida por el usuario para cada escenario particular generado a partir de un proceso base.

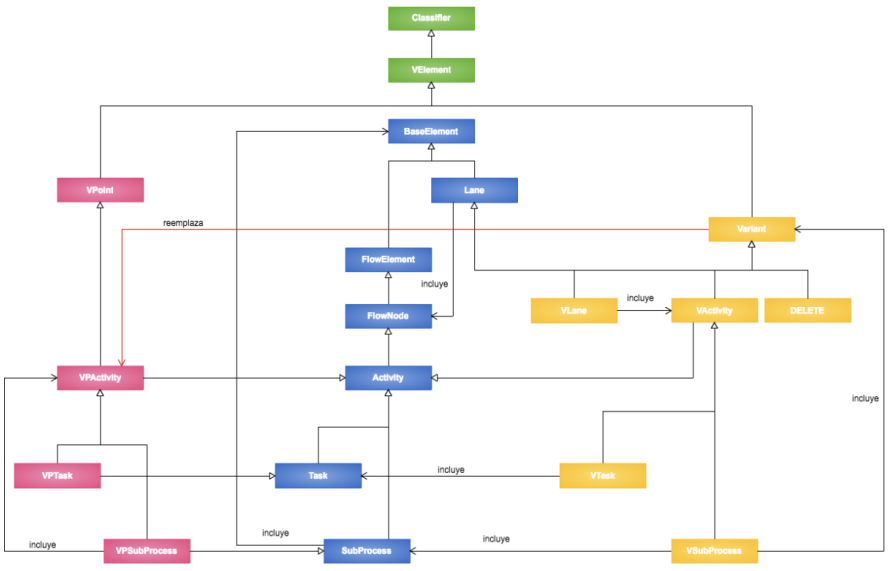


Figura 2.10: Metamodelo de BPMNext

En este lenguaje se definen los puntos de variación *VPActivity*, *VPTask* y *VPSubProcess*; VPTask y VPSubProcess son las representaciones de puntos de variabilidad de los elementos de BPMN Task y SubProcess respectivamente, y VPActivity es el concepto general que agrupa ambos. Las variantes definidas son VLane, VActivity,

VSubProcess y VTask. Es importante resaltar que cualquier variante puede ser reemplazo de un punto de variación. 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). Esta extensión fue creada a partir de la adaptación del enfoque seguido en VSPEM, el cual se encuentra en (Martínez-Ruiz et al., 2008). En 2.10 se presenta el metamodelo que fue propuesto para BPMNext, donde se destaca la adición de dos módulos - el de puntos de variación y el de las variantes, las cuales reemplazarán puntos de variación. Es importante destacar también que las variantes del proceso base serán generadas de forma automática mediante transformaciones MDE, utilizando también modelos de configuración debidamente definidos por el usuario que desea generar una variante (donde se reflejarán las decisiones tomadas respecto a cada punto de variación), tal como se muestra en (Delgado et al., 2018b).

Es así que se plantea la interrogante, ¿es posible definir algo similar a BPMNext, pero enfocado en procesos de negocio declarativos? Bajo esta premisa es que se trabajará en CMMNext.

Capítulo 3

Revisión Sistemática

Este capítulo ofrece una revisión sistemática centrada en la gestión de familias de procesos de negocio declarativos. Siguiendo las etapas y terminología establecidas en (Kitchenham et al., 2007), existen tres etapas en una revisión: fase de planificación, fase de conducción y reporte de la revisión.

Pese a haber sido realizada, la ultima etapa de la revisión sistemática no será detallada en este informe debido a que no aporta información relevante para el caso de estudio general.

Una vez finalizados los pasos formales de esta revisión, se profundiza en cada artículo seleccionado, con un enfoque especial en el análisis de Configurable Declare, un enfoque prometedor en la gestión de procesos declarativos. Este análisis se complementa con la aplicación del framework VIVACE (Ayora et al., 2015), ofreciendo una comparativa detallada con otros lenguajes de modelado de procesos de negocio.

3.1. Pasos formales de la Revisión

3.1.1. Etapa de Planificación

El primer paso es el de identificar la necesidad para la revisión. Una vez que se llegó a una base sólida de fundamentos teóricos, se identificó el objetivo de recolectar información académica relacionada a la temática que nos interpela; ver diferentes aspectos de modelado y configuración de familias de procesos declarativas. Esto conduce a buscar respuestas para la pregunta

¿Qué enfoques existen para abordar la gestión de familias de procesos de negocio declarativos?

Luego, se terminó por definir ciertos elementos faltantes del protocolo de búsqueda. Como referencia de que el string de búsqueda arroja buenos resultados se tenía el artículo (Delgado et al., 2022). Por otro lado, se definió por realizar el filtrado de forma individual en cada etapa de filtrado, teniendo una instancia donde discutir y argumentar las diferencias encontradas. Una vez finalizadas estas instancias se discutiría con los tutores para comprobar si el rumbo es el correcto. Concluida esta etapa de planificación, se avanza a la etapa principal de esta revisión.

3.1.2. Fase de conducción

Siguiendo con lo establecido por (Kitchenham et al., 2007), esta etapa comienza por definir un string de búsqueda relacionado al objetivo planteado. En lo que respecta a este proyecto, se definió lo siguiente:

```
declar* AND ("business process family" OR "business process
line" OR "configurable process model")
```

el cual fue variando levemente según el sitio en donde la búsqueda fuese llevada a cabo debido a la sintaxis específica de cada motor de búsqueda.

El siguiente paso es el del filtrado de artículos, que se puede ver resumido en 3.1, detallando los artículos proporcionados por las bases de datos en cada iteración.

Fuente	Cantidad de Artículos Encontrados	Relevantes	Candidatos	Primarios
Springer	60	18	6	3
ACM Digital Library	4	2	0	0
IEEE Digital Library	0	1	1	0
ScienceDirect	13	1	1	0
Wiley Inter Science	3	1	0	0

Figura 3.1: Tabla resultante de las iteraciones de la Revisión Sistemática

Se realizó una búsqueda en las siguientes bases de datos: Springer (donde se obtuvieron 60 documentos), ACM Digital Library (donde se obtuvieron 4 documentos), IEEE Digital Library (donde no se obtuvieron resultados), ScienceDirect (con 13 resultados), y Wiley Inter Science (con 3 documentos encontrados).

El siguiente paso es el de la recopilación de un conjunto inicial de documentos. Mediante un proceso de filtrado riguroso, se identifican y analizan los artículos más relevantes, proporcionando una visión general de las metodologías y enfoques actuales en el campo.

Se procedió a unificar en una misma lista los 80 resultados obtenidos, detallando para cada uno el título, autores y año de publicación. Tras constatar que no se contaba con artículos repetidos, se aplicó un primer filtrado donde analizando el título se realizó una primera pre selección de artículos relevantes; de esta forma se redujo el número de artículos a 22. Los criterios excluyentes tenidos en cuenta en este momento fueron la presencia de alguna de las siguientes palabras clave:

```
Declar*, Variability, Configurable, Family, Flexible
```

Se continuó el proceso con un análisis más detallado, leyendo los abstracts e introducciones (en caso de que fuese necesario) logrando así reducir nuevamente la lista, contando ahora con 7 artículos que se alineaban a la búsqueda que se pretendía. Se adicionó un artículo más a la lista final, encontrado dentro de la bibliografía citada de uno de los anteriores, finalizando este filtrado con 8 artículos candidatos.

Posteriormente fueron leídos estos artículos en su totalidad, realizando un breve resumen de cada uno; así fue que se logró destacar 3 artículos primarios que presentaban temáticas sumamente relevantes. A su vez, se llegó a la conclusión que los 5 artículos restantes o bien se referían también a estas temáticas, o realizaban aportes de índole teórica que no contribuían significativamente al objetivo de esta revisión.

Esta etapa finaliza con la síntesis de los artículos primarios, lo cual se puede observar en la siguiente sección. Esto se debe a que, por la relevancia en este informe, se decidió separarlo para darle la debida importancia.

3.2. Resultado final de la revisión

Se presenta a continuación un breve resumen de los tres artículos seleccionados.

3.2.1. The Old Therapy for the New Problem: Declarative Configurable Process Specifications for the Adaptive Case Management Support

En (Rychkova et al., 2011a) se discute el uso de especificaciones declarativas, modelado de variabilidad y semánticas basadas en lógica de primer orden para modelar procesos descriptivos (los cuales tiene como objetivo registrar y proporcionar un seguimiento de lo que ocurre durante el proceso de negocio), en particular los procesos de gestión de casos (CMPM) (OMG, 2009). Se utiliza como ejemplo el proceso de solicitud de una hipoteca, el cual se lo analiza desde el punto de vista de BPMN. A partir de esto se formulan los siguientes cinco requisitos para un enfoque de modelado de procesos descriptivo que los formalismos tradicionales difícilmente pueden cumplir:

1. Necesidad de especificar entradas/salidas distinguiendo entre datos obligatorios y opcionales, alternativas (posibles reemplazos) y sinónimos (artefactos idénticos llamados de manera diferente).
2. Necesidad de especificar la jerarquía de roles, roles alternativos y sinónimos.
3. Necesidad de especificar tareas opcionales, obligatorias y alternativas.
4. Necesidad de especificar múltiples posibilidades de flujo.
5. Necesidad de especificar el impacto de los datos en diferentes tareas y en el flujo de tareas.

Para satisfacer estos requisitos, los autores proponen un enfoque denominado **Declarative Configurable Process Modeling Notation (DeCo)**, el cual se basa en el modelado sistemático de datos relacionados con el proceso; esto permite introducir la noción de estado, donde cada tarea del proceso puede estar asociada a un conjunto de estados previos, que son los estados en los que esta tarea puede (pero no necesariamente) ejecutarse, y a un conjunto de estados posteriores, que son los estados resultantes de la ejecución de la tarea. Por otro lado, este enfoque se inspira en la notación utilizados por BPMN, pero manteniendo una naturaleza declarativa. Tomando inspiración en la notación "Configurable Integrated EPC" (C-iEPC) (La Rosa et al., 2011) que extiende la notación EPC (Event Process Chain) (La Rosa et al., 2011), se utiliza *multi-perspective configurability* para abarcar tres de los cinco requerimientos mencionados:

- Abarcar (1) a través de configurabilidad desde la perspectiva de los datos, capturando el ciclo de vida de los data objects (creación, uso, modificación, eliminación) dentro del proceso.
- Abarcar (2) a través de configurabilidad a través de la perspectiva de los recursos, la cual describe cómo el proceso es llevado a cabo dentro de la organización y trata los roles y asignaciones de recursos.

- Abarcar (3) a través de configurabilidad desde la perspectiva operacional, que aborda los aspectos técnicos de la ejecución del proceso y especifica las tareas elementales y su asignación a aplicaciones concretas o componentes de aplicación de las organizaciones.

Los dos restantes requerimientos se expresan a través de semánticas basadas en lógica de primer orden.

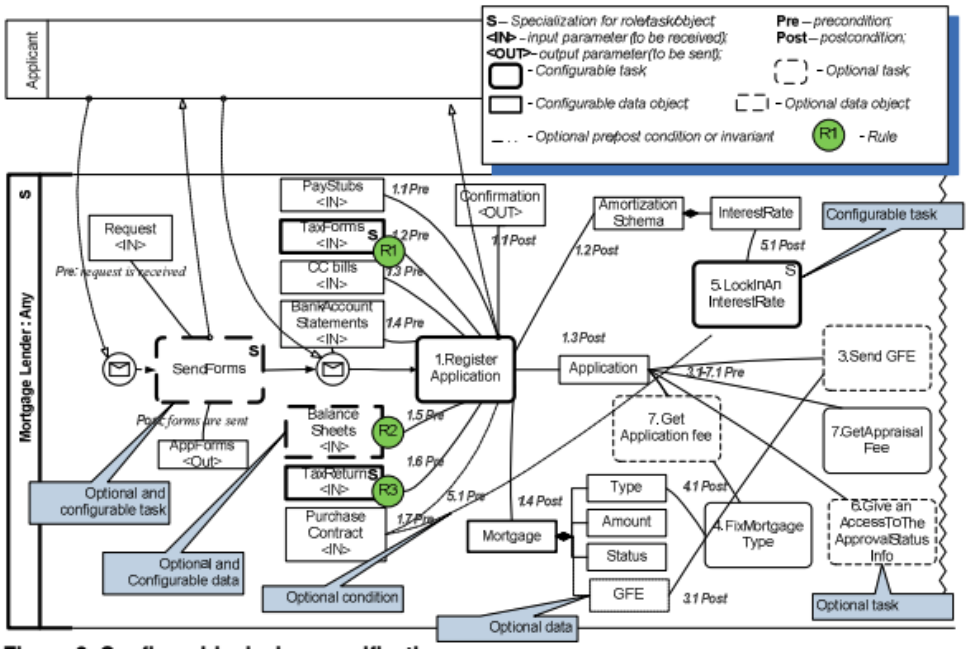


Figura 3.2: Ejemplo del artículo de DECO

Pese a no tener un caso de uso en el artículo, con el fin de poder mostrar el enfoque, se extrae el modelo representado en (Rychkova et al., 2011b), que es la segunda parte de este artículo.
En 3.2, se puede ver la manera de representar la configurabilidad de esta herramienta. Como se puede observar, en lugar de definir un flujo de trabajo estricto, el modelo DeCo utiliza restricciones para definir las reglas y condiciones que deben cumplirse en un proceso. Estas restricciones pueden ser de varios tipos, como restricciones de precedencia, restricciones temporales o restricciones basadas en recursos.
En el modelo de representación visual, la configurabilidad de un objeto se indica mediante la modificación de su contorno. Esto se logra incrementando el grosor de los bordes del objeto en cuestión.

3.2.2. A Case Modelling Language for Process Variant Management in Case-Based Reasoning

Este artículo (Cognini et al., 2016) describe la aplicación de un enfoque de CBR (Case-based reasoning, metodología que utiliza el conocimiento de situaciones previamente experimentadas y sus soluciones, para proponer una potencial solución a una nueva situación), e introduce un lenguaje de modelado de fragmentos de procesos ba-

sados en casos llamado **Business Process Feature Model (BPFM)** que soporta la generación y refinamiento manual de casos generalizados.

Para ello se presenta como ejemplo un escenario de aplicación a un máster en una universidad, y tras un breve análisis se concluye que no está del todo claro cuáles actividades son requeridas obligatoriamente, o en qué orden deben llevarse a cabo.

Mediante el análisis de varios casos del ejemplo mencionado, se derivan objetivos de investigación para el enfoque de CBR, especialmente el modelado del contenido del caso y su caracterización; este último se puede considerar como la descripción del problema (de la situación) y el contenido del caso se puede considerar como la solución o lección, que consta de al menos un elemento de conocimiento (por ejemplo, documentos, procesos, etc.).

BPFM es un Configurable Process Model ya que puede encapsular distintas variantes de un proceso de negocio. Está constituído por un árbol de actividades relacionadas:

- La raíz identifica el servicio bajo análisis, así como la familia de BPs detrás del servicio en sí.
- Cada actividad interna (no hoja) denota un subproceso que se puede refinar aún más, y
- La actividad externa (hoja) representa una tarea atómica.

Se utilizan restricciones para expresar

- Si las actividades hijas pueden o deben seleccionarse en la configuración para incluirse en la variante del BP
- Si pueden o deben incluirse en cada camino de ejecución de la variante del BP, considerando de esta manera la inclusión estática y dinámica (en tiempo de ejecución) de las actividades.

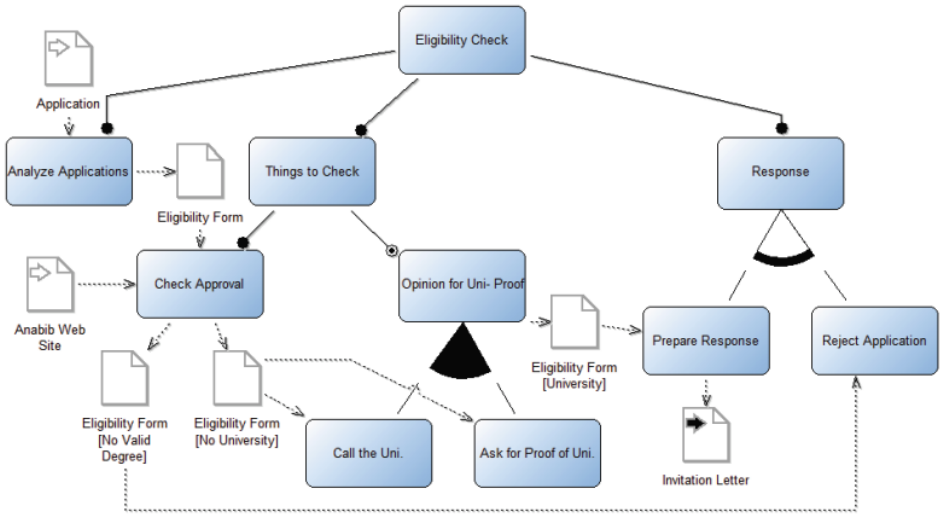


Figura 3.3: Modelo BPFM para el ejemplo de aplicación a un máster

Se presenta finalmente un modelo de BPFM para el ejemplo analizado a lo largo del artículo, el cual puede apreciarse en la Figura 3.3

3.2.3. Configurable Declare: Designing Customisable Flexible Process Models

Se analiza en este artículo (Schunselaar, Maggi, Sidorova, y van der Aalst, 2012) cómo soportar la configurabilidad de procesos declarativos; particularmente se trabaja en una variante del ya mencionado Declare, para dar lugar a un nuevo modelo llamado Configurable Declare.

En la introducción se hace mención a la utilidad y crecimiento del uso de modelos de proceso, y de la nueva tendencia a la utilización de modelos declarativos, sobre todo en escenarios donde hay un grado alto de variabilidad, o cuando es necesario especificar lo que se puede (o no) realizar, y no cómo hacerlo.

Dado que un modelo de proceso declarativo es un conjunto de restricciones sobre un conjunto de eventos (que representan la finalización de una tarea específica en un proceso de negocio), las opciones de configuración que se incluyen en Configurable Declare son (1) ocultar un evento y (2) omitir una restricción; esto se haría con booleanos. Ambas acciones se realizan mediante la inclusión de una cruz roja en el margen superior derecho de la tarea ocultable o en el medio de la restricción omisible (se puede ver en 3.4).

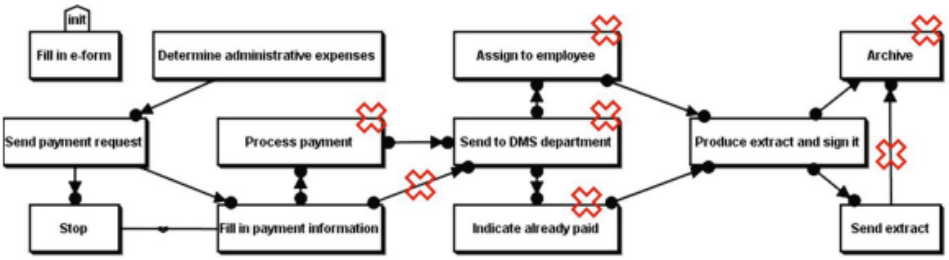


Figura 3.4: Modelo de Configurable Declare para Municipios

A continuación, se presenta un modelo de Declare para el ejemplo presentado en el artículo que se puede observar en la figura 3.4.

Lo que se destaca en esta variante es que para ocultar un evento que tiene reglas asociadas a él hay que saber elegir cuando eliminarlas o cuando editarlas, lo cual de si hay otras reglas implícitas sujetas a estas. Tomando como ejemplo el modelo presentado en 3.4, en caso de ocultar la tarea *"Send to DMS department"*, es necesaria la creación de una nueva restricción que refleje que *"Fill in payment information"* tiene que completarse previo al comienzo de las tareas *"Assign to employee"* e *"Indicate already paid"*. También existen algunas restricciones que no pueden ser eliminadas, por lo que existen meta-reglas que previenen esto para así mantener el sentido del proceso.

3.3. Profundizando en Configurable Declare

Luego de la investigación presentada en la sección anterior, se decidió profundizar con Configurable Declare debido a que era una extensión sencilla sobre el lenguaje

original Declare, y que permite comenzar a trabajar en estrategias nuevas para extender CMMN.
El objetivo principal fue complementar la documentación respecto a distintos lenguajes ya analizados (a través de la aplicación del framework VIVACE, del que se hablará a continuación), y así poder comparar este lenguaje con CMMN, para comenzar a buscar alternativas para extender el mismo.

3.3.1. Introducción al Marco VIVACE

VIVACE (Ayora et al., 2015) es un marco conceptual utilizado para la evaluación y gestión de la variabilidad en modelos de procesos de negocio. Proporciona un enfoque estructurado para identificar y manejar puntos de variación y variantes dentro de los modelos de procesos, lo que es crucial en contextos donde los procesos necesitan adaptarse a diferentes situaciones o contextos. Este marco facilita la definición y gestión de variaciones de un proceso base, permitiendo una mayor flexibilidad y personalización en la gestión de procesos de negocio.

Elementos del Marco VIVACE

LC1: Región configurable/Punto de Variación Representa puntos de variación en el modelo de proceso, como regiones configurables. Estos puntos de variación permiten la adaptación y personalización del proceso.

LC2: Configuración Alternativa/Variantes Son variantes conectadas a las partes variables del modelo. Incluyen fragmentos que pueden reemplazar a otros o definir operaciones específicas como eliminación, reemplazo o adición.

LC3: Condición de Contexto de Configuración Define las condiciones para seleccionar una variante, incluyendo condiciones de presencia para actividades y condiciones de contexto en modelos de características.

LC4: Restricción de Configuración Representa restricciones con respecto a la selección de variantes, incluyendo reglas de configuración adicionales relacionadas con los elementos de un árbol VSpec.

LC5: Tiempo de Resolución de la Región Configurable Se refiere a regiones que ofrecen flexibilidad mediante adaptación durante la ejecución del proceso. No todos los enfoques admiten este tipo de configurabilidad.

3.3.2. Documentación VIVACE - Configurable Declare

Para poder comparar Configurable Declare con otros lenguajes para el modelado de Procesos de Negocio, se aplica la adaptación del framework VIVACE presentada en (Delgado et al., 2022), el cual consiste en una herramienta para evaluar y comparar enfoques de variabilidad de procesos y su soporte del ciclo de vida del proceso de negocio.

En el mencionado artículo se profundiza en la aplicación del framework, así como también se presentan ejemplos de modelado de una instancia de una actividad, aplicando los siguientes lenguajes:

- ADOM
- BPMN*
- BPMNt
- C-BPMN
- DECO
- ABIS
- BPFM_C
- BPFM_D
- BPMNext
- ConfBPMF
- PESOA
- BPFM_N
- CVL/BVR
- PPM
- Provop
- vBPMN

Este ejemplo consiste en modelar distintas representaciones de alternativas para el punto de variación "Evaluar Candidato", dentro del escenario planteado en la sección 2 de (Delgado et al., 2022).

Partiendo de las documentaciones de los lenguajes ya analizados, y de la investigación específicamente aplicada a Configurable Declare, se aplicó el mencionado framework a este último obteniendo como resultado lo especificado en la figura 3.5 y la representación del ejemplo anterior utilizando Configurable Declare, el cual demuestra que es posible representar una actividad como ocultable agregando una cruz roja en la parte superior derecha de la misma, como puede apreciarse en la figura 3.6.

Configurable Declare cubre las siguientes perspectivas de proceso:

- (F) Funcional: Representa las actividades que son llevadas a cabo.
- (B) Conductual: Representa el flujo de control entre las actividades representadas.
- (T) Temporal: Se basa en lógica temporal y otorga la posibilidad de omitir una restricción temporal entre tareas.

En cuanto a las construcciones de lenguaje específicas de variabilidad ("Variability-specific language constructs"), se puede observar que LC1 y LC2 son soportados, observando la posibilidad de definir actividades, elementos y restricciones configurables (LC1 [+]), y la presencia de atributos configurables que permiten definir posibles variantes (LC2 [+]). Con respecto a LC3, LC4 y LC5 se observa que Configurable Declare no cuenta con soporte para ello.

Por el lado de las características de soporte de variabilidad ("Variability support features"), se entiende que no están soportados por el modelo debido a que no se cuenta con una herramienta implementada/conocida para llevar a cabo lo que conceptualmente fue definido; esto se ve reflejado en los campos "Tool implementation" y "Empirical evaluation" definidos anteriormente en la tabla. En el artículo estudiado, queda claro que esto no deja de ser una propuesta teórica tomando como referencia municipalidades holandesas, pero que no fue llevada a cabo en un ambiente productivo.

Terminando esta sección, se incluye la tabla 3.7, extraída del artículo (Cognini et al., 2016) descrito en la Subsección 3.2.2, pero incluyendo DECO, el único faltante de los lenguajes mencionados anteriormente la tabla en cuestión.

VIVACE		CONFIGURABLE DECLARE
Language		Extension of Declare
Technique		Single Artifact
Mechanism		Node Configuration
Perspectives		F, B, T
Variability-specific language constructs	LC1	+
	LC2	+
	LC3	-
	LC4	-
	LC5	-
Variability support features	F1.1	-
	F1.2	-
	F1.3	-
	F1.4	-
	F1.5	-
	F2	-
	F3.1	-
	F3.2	-
	F4	-
	F5.1	-
	F5.2	-
Tool implementation		-
Empirical evaluation		-
Domain		Government

Figura 3.5: Tabla con la aplicación de Vivace a Configurable Declare

3.4. Comparaciones entre los distintos lenguajes

En esta sección se presenta en primer lugar una comparación entre Configurable Declare y CMMN, lo cual tiene particular importancia debido a que, como se comentó con anterioridad, fue Configurable Declare el lenguaje elegido para estudiar en profundidad.
De todas formas, y por necesidad en etapas posteriores, se agrega un corresponden-

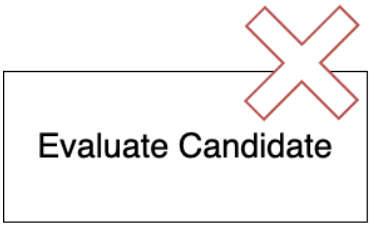


Figura 3.6: Representación gráfica de la actividad opcional “Evaluate Candidate” en Configurable Declare

	BPMN	CMMN	DECLARE	BPFM	DECO
For BP modeling	Yes	Yes	Yes	Yes	Yes
Language type	Imperative	Declarative	Declarative	Declarative	Declarative
Defined activities Flow	Full	In part	In part	In part	In part
Complex constraints	Yes	No	No	Yes	No
Data representation	Yes	In part	No	Yes	Yes
Variants representation	No	No	No	Yes	Yes

Figura 3.7: Cuadro comparativo de lenguajes de modelado

cias en líneas generales de CMMN con DECO y BPFM, lenguajes estudiados en este capítulo.

3.4.1. Comparación de Declare con CMMN

En esta etapa se busca aplicar lo visto en Configurable Declare para CMMN con el objetivo de representar la configurabilidad de los modelos en ese lenguaje. Luego de dar una breve definición introductoria de Case Management Model and Notation (CMMN), se muestra una tabla de correspondencia con los principales elementos de Declare y su notación en CMMN para luego terminar comparando el modelo estudiado en (Schunselaar et al., 2012) con su conversión realizada para CMMN, definida previamente en este artículo.

Tabla de Correspondencia entre Declare y CMMN

Se presenta a continuación, en la tabla que se presenta en la figura 3.8, la correspondencia entre las restricciones de Declare vistas en los casos de uso estudiados con la notación empleada en CMMN, siendo estas `response`, `precedence`,`succession`, `alternate response`, `exclusive 1 of 2` y `init`. En la misma se brinda una breve explicación de cada restricción y su representación en cada lenguaje, ejemplificando con elementos de estos lenguajes lo explicado en palabras. Estas correspondencias fueron definidas a partir de lo entendido sobre lo que cada restricción buscaba explicitar en Configurable Declare, así como también lo que CMMN permitía representar a través de sus elementos. Por otro lado, se presentan en la figura 3.9 ejemplos de cómo se representan las compuertas lógicas OR y AND respectivamente utilizando elementos CMMN, las cuales fueron utilizadas en la definición de la tabla anteriormente mencionada.

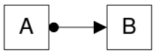
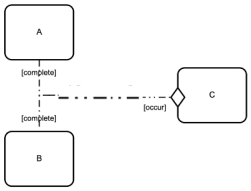
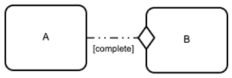
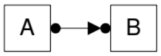
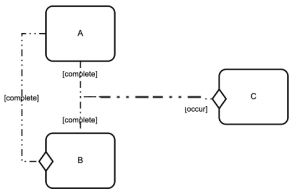
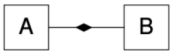
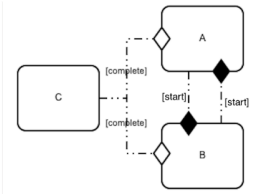
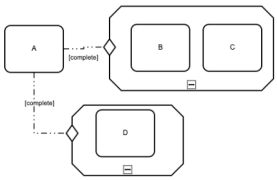
Restricción Config. Declare	Explicación Config. Declare	Rep. Gráfica Config. Declare	Explicación CMMN	Rep. Gráfica CMMN
response(A, B)	Siempre que A es ejecutada, B **debe** ser ejecutada posteriormente *(aunque no necesariamente a continuación).* *Notar que puede ejecutarse B sin obligación de haber tenido que ejecutar A anteriormente.*		Si hay una tarea C que depende de que A sea completada, entonces debe depender de que B sea completada también. Funciona igual que el AND. *Notar que puede ejecutarse B sin obligación de haber tenido que ejecutar A anteriormente.*	
precedence(A, B)	B puede ser ejecutada únicamente luego de haber sido ejecutada A. *Notar que puede ejecutarse A sin obligación de luego tener que ejecutar B.*		B es habilitada una vez que A fue completada.	
succession(A, B)	A **debe** ser seguida posteriormente por B, y B no puede completarse sin que A haya sido completada anteriormente. *Combina response y precedence.*		Si hay una tarea C que depende de que A sea completada, entonces debe depender de que B sea completada también. Si hay una tarea C que depende de que B sea completada, entonces debe depender de que A sea completada también. A su vez, B no puede ejecutarse hasta haberse ejecutado A.	
exclusive 1 of 2 (A, B)	O bien A, o bien B **debe** ser ejecutada, pero **no ambas**.		*Patrón Exclusive Choice.* Cuando una de las tareas A o B es elegida, la que no lo fue deja de estar disponible.	
init(A)	A **debe** ser la primera tarea en ejecutarse. *Notar que no hay prohibición que implique que A no pueda volver a ejecutarse en otro momento.*		Se logra un resultado análogo haciendo que todas las otras tareas dependan de que A sea completada	

Figura 3.8: Tabla de restricciones de Configurable Declare representadas con elementos de CMMN

3.4.2. Comparaciones entre CMMN y otros lenguajes

En esta subsección se provee de una lista de correspondencias entre CMMN y el restante de los lenguajes encontrados y definidos en secciones anteriores de este capítulo útiles para etapas posteriores del proyecto. Ciertas limitaciones consideradas son propias del modelador de CMMN (*cmmn-js*, s.f.) utilizado descrito posteriormente.

Correspondencias entre CMMN y DECO

Para representar CMMN en DECO se tienen que tener en cuenta los siguientes aspectos:

- Las tareas (Task) se corresponden directamente entre ambos modeladores.
- Los Data Object en DECO se corresponden con los CaseFileItem en CMMN.

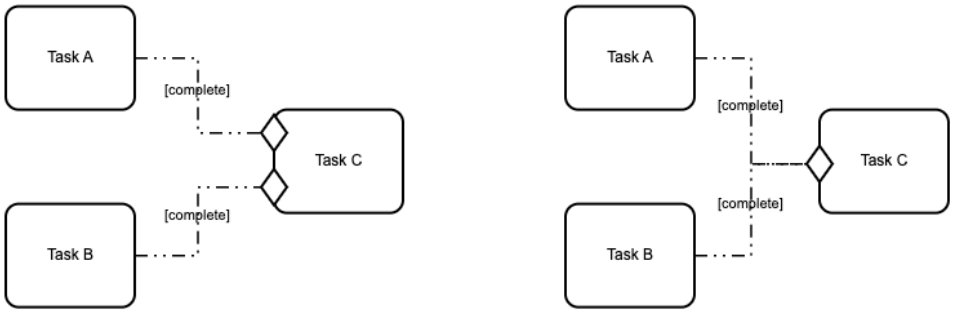


Figura 3.9: Compuertas lógicas OR y AND representadas con elementos CMMN

- Se cambian los elementos opcionales (Optional Tasks y Optional Data Object) por normales debido a limitaciones en CMMN-js que generan que los elementos opcionales (Discretionary) no se puedan conectar, de todas formas no son parte del objetivo central del proyecto.
- Al no poder conectar los CaseFileItem en CMMN debido a limitaciones de la herramienta, se opta por incluir dos caseFileItem conectados en un Stage para poder representar de alguna forma la dependencia que tienen entre sí.
- Las restricciones de BPFM se transforman en restricciones de CMMN.

Correspondencias entre CMMN y BPFM

Las correspondencias encontradas de la representación estos lenguajes son:

- Las actividades (Activities) de BPFM se corresponden con las tareas (Task) de CMMN.
- Los Data Objects en BPFM se corresponden con los CaseFileItem en CMMN.
- Las restricciones de BPFM se transforman en restricciones de CMMN.

3.5. Conclusiones de esta etapa

Luego del análisis de lo investigado en conjunto con los tutores, se llegó a la conclusión que la diferencia principal entre ambas notaciones es que Declare se enfoca fundamentalmente en las relaciones entre tareas, en tanto CMMN se enfoca en las tareas mismas.

Con Declare se tiene una concepción de 'mundo abierto' en donde es posible que suceda prácticamente cualquier cosa, mientras sean respetadas las restricciones definidas en las relaciones. En cambio, en CMMN por lo general se modela cuándo se debe hacer disponible una tarea, es decir, se enfoca en lo que debe ocurrir en el pasado para que parte del proceso esté habilitado al usuario (al definir ciertos parámetros en la etapa de configuración), pero muchas veces no obliga a que 'ese algo' suceda.

Tras algunos intentos en pequeños ejemplos, se concluyó que el trabajo de realizar una

representación formal de Declare en CMMN no sería sencillo y se alejaría el foco del objetivo principal del proyecto. Además, en caso de replicar lo hecho en Configurable Declare para extender CMMN, se entendió que se proveerían nuevas funcionalidades muy básicas a CMMN (permitiendo únicamente mantener o eliminar elementos), por lo que se tomó la decisión de no continuar con esta labor y cambiar el enfoque de la investigación a buscar la extensibilidad de CMMN a través de alguna herramienta que lo permita, como se detallará a continuación.

Capítulo 4

Definición de CMMNext

A continuación se presenta el camino recorrido para llegar a una implementación del lenguaje propuesto denominado CMMNext, para la representación de familias de procesos de negocio.

Una vez descartado el enfoque de Configurable Declare, se buscó seguir un razonamiento análogo al llevado a cabo en el desarrollo de BPMNext (Delgado et al., 2017) (lo cual fue presentado con anterioridad en 2.3), sumado también a la posibilidad de generar variantes de familias de procesos de forma automática a través de herramientas basadas en ingenería dirigida por procesos (presentado en (Calegari et al., 2019)). Es así que se trabajó en la creación de la extensión de CMMN para soportar representaciones de familias de procesos, la cual denominamos CMMNext.

Para ello, se dividió el trabajo en las siguientes etapas:

1. Definición de metamodelos de CMMN y CMMNext (Sección 4.1)
2. Extensión de la herramienta cmmn-js (*cmmn-js*, s.f.) (desarollada por los creadores de Camunda (*Camunda*, s.f.)) para permitir representar elementos de configurabilidad, dando lugar a cmmnext-js (Sección 4.2)
3. Transformación T2M de archivos `.cmmn` a `.xmi`, para permitir que los archivos de salida de cmmnext-js conformen con el metamodelo definido para CMMNext (Sección 4.3)
4. Transformación M2M de CMMNext a CMMN, utilizando el ya mencionado modelo de configuración para instanciar casos particulares de la familias de procesos (Sección 4.4)
5. Transformación M2T desde `.xmi` a `.cmmn`, para obtener un archivo CMMN con la instancia particular deseada (Subsección 4.4.1)

Se brindarán a continuación detalles de cada etapa previamente mencionada, brindando ejemplos para una mejor comprensión.

4.1. Elaboración de metamodelos de CMMN y CMMNext

Se utilizó una representación del metamodelo de CMMN en formato `XMI` (*Case Management Model and Notation*, s.f.), lo cual no se correspondía con el estándar

Ecore que se maneja en el presente proyecto.
Para solucionar este inconveniente, se realizó una pequeña transformación `.xmi` a `.ecore` para contar con el metamodelo en el formato requerido. En la figura 4.1 se puede observar una parte del mismo, extraída de la documentación de CMMN, el cual no se muestra en su totalidad debido a su extenso tamaño. Se eligió incluir particularmente esta sección del metamodelo debido a que es esencialmente sobre esta sección donde se implementa la extensión del metamodelo.

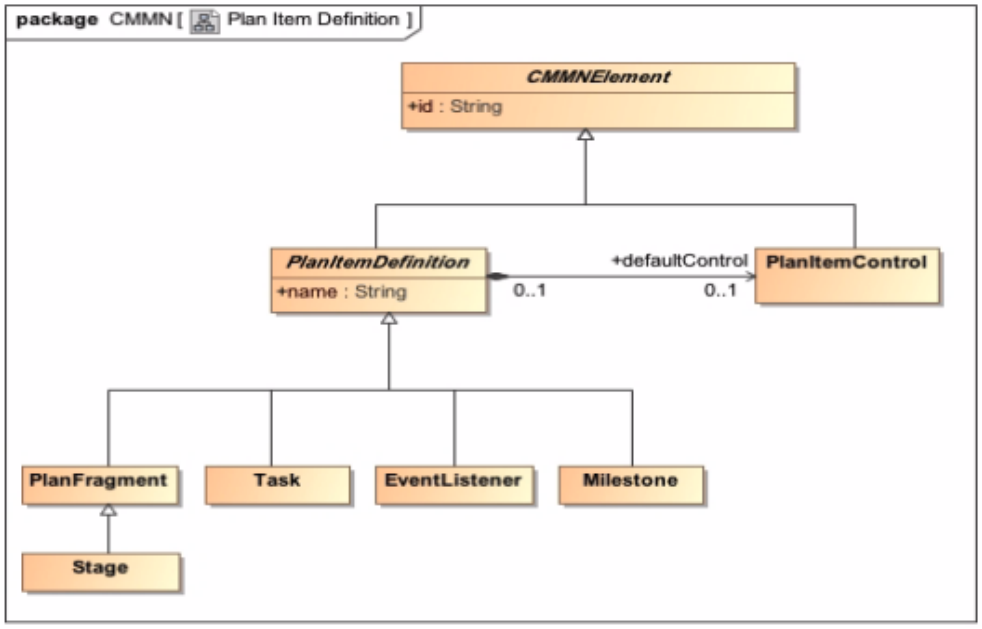


Figura 4.1: Parte del metamodelo de CMMN

Se implementó a continuación el metamodelo de CMMNext, siguiendo el mismo enfoque propuesto para el ya presentado BPMNext; en la figura 4.2 se observa parte del mismo, donde lo que se hizo para extender el metamodelo fue agregar dos módulos - el de puntos de variación y el de las variantes, las cuales reemplazarán puntos de variación. Las opciones posibles para cada punto es o bien remover el elemento, o mantenerlo. Los elementos en azul representan los elementos de CMMN que se aprecian en 4.1.

En el caso de CMMNext y a diferencia de lo presentado en Configurable Declare, no sólo se podrá eliminar un punto de variación, sino que también será posible reemplazar puntos de variación por cualquier variante disponible - es decir que, por ejemplo, un VPStage podrá ser sustituído por una variante de tipo VStage, por una variante de tipo VTask, o por una variante de tipo VMilestone.

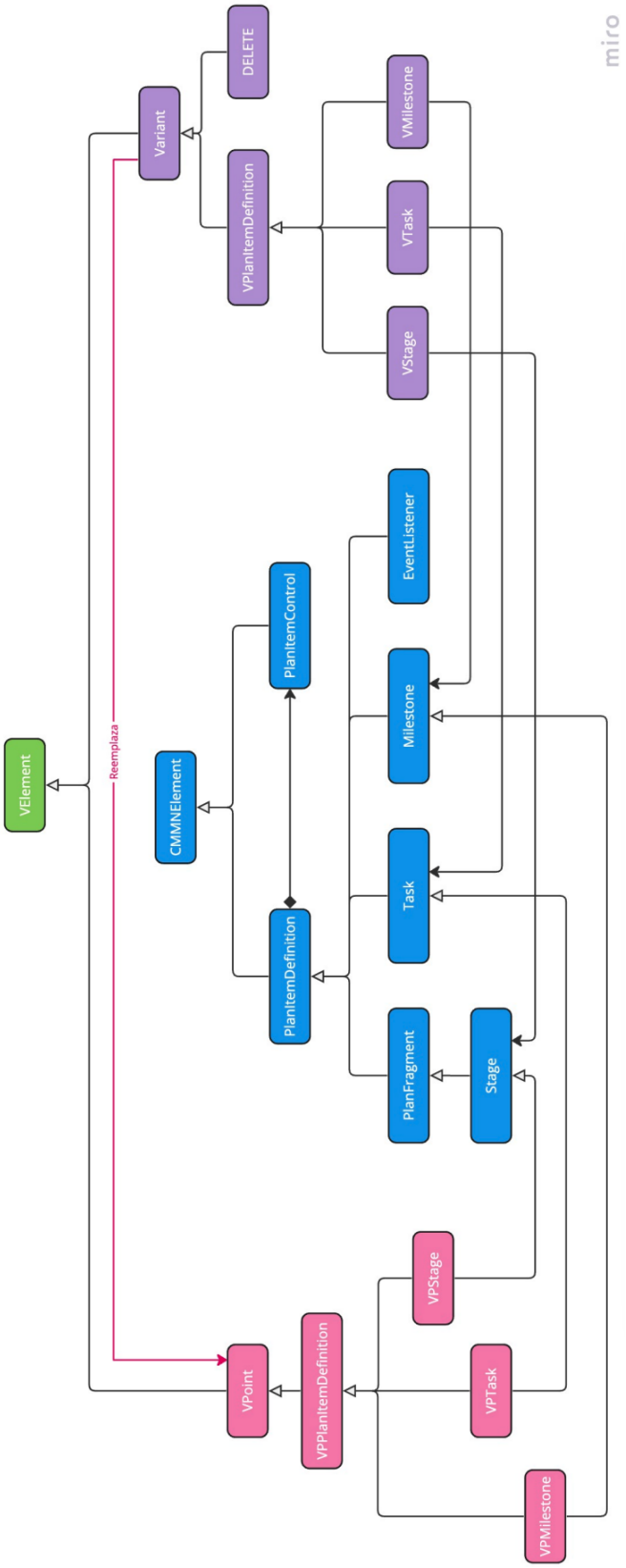


Figura 4.2: Parte del metamodelo de CMMNext

Metamodelo de Configuración Se utilizó el metamodelo de configuración presentado en el proyecto CSIC (Calegari et al., 2019) (Delgado et al., 2018a) y utilizado en BPMNext. Fue posible volver a utilizarlo en el presente trabajo gracias a que, como se comentó anteriormente, se definió el metamodelo de CMMNext siguiendo el mismo enfoque que se siguió al definir BPMNext, manteniendo los conceptos de VarPoint y Variant.

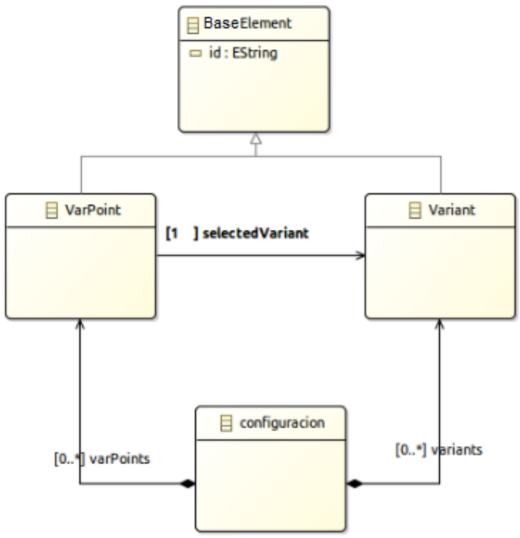


Figura 4.3: Metamodelo de configuración definido en el proyecto CSIC

En la figura 4.3 se presenta el metamodelo, donde se aprecia que tanto **VarPoint** como **Variant** son elementos que extienden **BaseElement**; además, existe un elemento **configuración** que permite referenciar las instancias de ambos elementos, de forma tal de lograr generar un modelo de configuración válido. Puede verse un ejemplo de configuración en la figura 4.4.

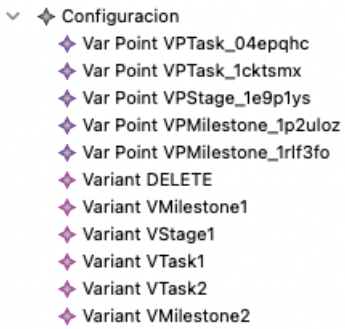


Figura 4.4: Ejemplo de modelo de configuración

Por lo tanto, teniendo en cuenta lo presentado hasta ahora, el objetivo del metamodelo de configuración es que, teniendo un modelo base que represente la familia de procesos, y un modelo de configuración específico definido para una variante, se

generará la variante correspondiente de forma automatizada.

4.2. Editando el modelador cmmn-js

En el esfuerzo por personalizar el modelador cmmn-js para ajustarse a las necesidades específicas del proyecto, se llevó a cabo la introducción de nuevos elementos en la paleta de modelado. Estos nuevos elementos, denominados `VPStage`, `VStage`, `VPMilestone`, `VMilestone`, `VPTask`, y `VTask`, fueron diseñados para replicar y extender el comportamiento de los elementos estándar `Stage`, `Milestone` y `Task`. La incorporación de estos elementos tiene como objetivo enriquecer el motor de la mencionada herramienta gráfica, proporcionando nuevas funcionalidades y adaptándose a las necesidades del proyecto que extiende CMMN.

La expansión del modelador implicó una serie de modificaciones técnicas complejas. El proceso incluyó la adaptación de las definiciones de los elementos, la personalización de los comportamientos de renderizado, del *frontend* de la herramienta para la creación de los modelos y la integración con el sistema de exportación para generar archivos `.cmmn` que reflejan fielmente las construcciones modeladas en la herramienta.

El resultado de estas modificaciones es el módulo `cmmnext-js`, una extensión del modelador cmmn-js que soporta los elementos de CMMN junto con los recién añadidos. Este módulo permite una transición suave entre el modelado y la ejecución, manteniendo la compatibilidad con las herramientas estándar de CMMN.

La integración de estas características avanzadas fue un proyecto que involucró la modificación de más de treinta archivos, lo que subraya la escala y la profundidad del trabajo realizado. Para distinguir visualmente los nuevos elementos en el diagrama, se tomó la decisión de marcar los elementos `VP` con un diamante en la esquina superior izquierda [4.5], mientras que los elementos `V` se diferencian con un círculo en la misma ubicación [4.6], proporcionando una identificación inmediata y manteniendo la coherencia estética con el estándar CMMN.

Figura 4.5: Ejemplo de representación de un VPoint (VPTask en este caso)

Figura 4.6: Ejemplo de representación de un VElement (VTask en este caso)

4.3. Transformación de salida de cmmnext-js a metamodelo CMMNext

Se implementó una herramienta de transformación de modelos de procesos que convierte archivos `.cmmn` a formato `.xmi`. Esta herramienta se adapta al metamodelo CMMNext, generando como resultado archivos que conforman con el mismo y que podrán ser utilizados como entrada para la transformación *modelo a modelo* que se desarrollará más adelante. El proceso fue llevado a cabo utilizando la biblioteca Freemarker (*Freemarker*, s.f.) para la generación de plantillas, permitiendo una adecuada construcción del archivo `XMI` resultante.

A continuación se brindan detalles de las distintas etapas recorridas persiguiendo ese objetivo.

4.3.1. Configuración de Freemarker

Freemarker es una herramienta de generación de plantillas que facilita la separación del diseño y la lógica de programación. En el proyecto, se utiliza para definir la plantilla de transformación de los elementos `.cmmn` a su representación `XMI`. Se configura la codificación `UTF-8` y se establece un manejador de excepciones que promueve una depuración efectiva de errores durante el proceso de transformación.

4.3.2. Lectura y análisis del archivo .cmmn

El proceso inicia con la lectura del archivo `.cmmn` mediante la función `readCMMNFile`. La cadena de texto resultante es procesada por `parseCMMNContent`, que construye un modelo de datos que refleja la estructura del archivo .cmmn. La función que contiene la parte más compleja de la lógica de esta transformación es `getChildElementsList` donde se itera entre los hijos de los componentes de CMMN. Es imprescindible contar con una función de este tipo debido a la estrutura arborescente de estos archivos.

4.3.3. Transformación de etiquetas y atributos

Se utiliza una interfaz funcional, `TagNameTransformer`, y el método `transformTagName` de la clase `transformation` para mapear los nombres de etiquetas CMMN a los equivalentes en el metamodelo CMMNext. La clase `CurrentStandardEventManager` gestiona los eventos estándar, crucial para la transformación adecuada de elementos específicos del modelo. Cabe destacar que en esta etapa no solo se realizaron cambios de etiqueta, si no que también se tuvo que realizar cambios en la estructura para conformar con el metamodelo de CMMNext.

4.3.4. Integración con Freemarker

La integración con Freemarker se logra a través de `TemplateMethodModelWrapper`, que permite la transformación dinámica de nombres de etiquetas dentro de las plantillas de Freemarker. En esta sección se interpreta el `DataModel` generado, para poder transformarlo en una salida que conforme con el formato de `.xmi`.

4.3.5. Inclusión de los elementos para la configurabilidad

Para soportar la lógica requerida para la configurabilidad fue necesaria la inclusión de los `VElements` en el archivo de entrada para realizar la transformación M2M.

La decisión que se tomó fue de incluir los distintos elementos en archivos distintos organizados según cada caso posible de configuración de variantes. Por esa razón se tuvo que iterar entre archivos dentro de un mismo directorio recolectando los distintos elementos y agregándolos al final del archivo `.xmi` generado. Sumado a esto, se agregó la variante `DELETE` al final de la lista.

4.3.6. Generación del archivo XMI

El contenido transformado se escribe en un archivo `.xmi` utilizando la función `writeToFile`. Este archivo conforma con el metamodelo CMMNext y con las herramientas que soportan el estándar XMI, asegurando su utilidad en diversos entornos de modelado.

La implementación de este proceso de transformación contribuye a la compatibilidad y extensibilidad de los modelos de procesos dentro del proyecto.

4.4. Transformación M2M CMMNext a CMMN

Dentro de un entorno de desarrollo de Eclipse Modelling Tools (*Eclipse Modelling Tools*, s.f.), se utilizó la herramienta ATL (*ATL*, s.f.) para realizar transformaciones entre metamodelos. Por lo tanto se trabajó en una transformación ATL para, a partir de un modelo CMMNext y uno de configuración, obtener un modelo CMMN.

El modelo de entrada CMMNext contiene el proceso base (que incluye distintos puntos de variación), y todas las variantes definidas por el usuario para reemplazar los puntos de variación definidos. Es importante destacar esto último, ya que en la transformación se deberá no solo hacer el reemplazo de los VPoints por los VElements elegidos, sino que también omitir la inclusión de estos VElements en el modelo de salida, puesto que se desea obtener un modelo CMMN válido que contemple únicamente un proceso de negocio sin variantes extras que no hayan sido tenidas en cuenta. Se ejemplifica lo mencionado en la figura 4.7.

Respecto a la transformación, en primer lugar se definieron reglas de transformación "identidad", las cuales simplemente se encargan de copiar los elementos que no representan variabilidad de un modelo a otro. Si bien son reglas simples, se deben considerar ciertos escenarios en la mayoría de ellos:

- Si el elemento depende de elementos que sí representaban variabilidad, es necesario agregar controles que determinen si el mismo debe ser omitido en el modelo resultado. Por ejemplo, si un elemento de tipo `PlanItem` referencia a un elemento de tipo VPoint, que en la configuración elegida es sustituído por la variante DELETE, entonces el mismo debe ser omitido en la transformación. Esto puede apreciarse en la figura 4.8, en las líneas 487-490.
- Si el elemento es de tipo Stage, Milestone o Task, pero se encuentra dentro de un VElement, no debe ser tenido en cuenta en estas reglas "identidad" ya que serán transformados al procesar los VPoints y sus configuraciones elegidas. Se muestra un ejemplo en la figura 4.9, líneas 724-726.
- Teniendo en cuenta el punto anterior, si el elemento pertenece a un Stage que se encuentra asociado a un VStage, y el mismo no fue seleccionado como variante

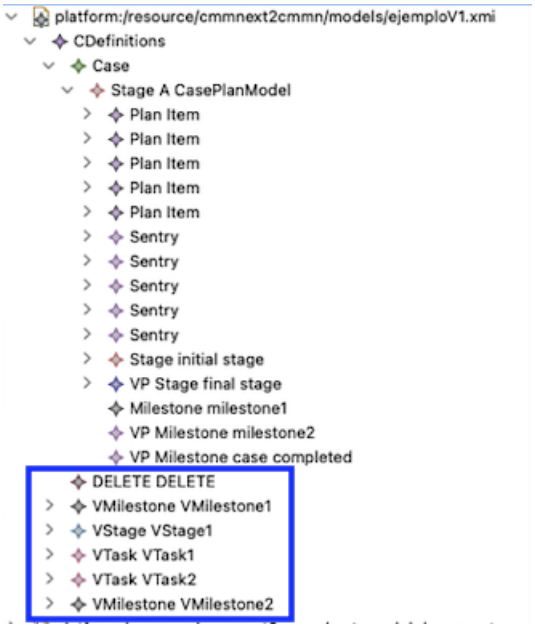


Figura 4.7: Los elementos contenidos en el recuadro deben ser omitidos en el modelo de salida de la transformación

para alguno de los VPoints, entonces no debe ser transformado. Esto se realiza de forma tal de no terminar con "elementos huérfanos", logrando como resultado un CMMN limpio y correcto. Se presenta un ejemplo de esto en la figura 4.9, líneas 727-741.

Otro aspecto a destacar es que, para garantizar compatibilidad entre los VPoints y los VElements elegidos, estos últimos deberán mantener los sentries de entrada/salida (diamantes blancos/negros) definidos para los VPoints que sustituyen, como se ejemplifica en la figura 4.10. Por lo tanto, se asume que el número de sentries asociadas a un VPoint se mantendrá, por lo que no deben definirse sentries de entrada/salida en el elemento principal de las variantes implementadas (como se muestra en 4.11).

La parte central de la transformación M2M es la que se encarga de realizar las correspondientes sustituciones de los VPoints. Como se mencionó anteriormente, es posible sustituir cualquier VPoint por una variante DELETE (eliminando el VPoint y sus items asociados, como ser los sentries de entrada/salida), sustituir un VPoint por un VElement de su mismo tipo (es decir, por ejemplo un VPTask por una VTask), y sustituir VPTask por VStage y VPStage por VTask. Sobre todo esto último provee una flexibilidad mayor, permitiendo sustituir lo que a priori sería una simple tarea por un Stage con la complejidad que se requiera para representar esa parte del proceso. Cabe destacar que no hay reemplazos de VPMilestone por elementos que no sean VMilestone ya que se entiende que dado el significado que los VMilestone tienen en CMMN, no tendría sentido modelar un punto de variación que pueda ser reemplazado por una Task o un Stage, con un VPMilestone.

Para ilustrar lo explicado, se puede apreciar lo siguiente:

- En la figura 4.12 se observa cómo se da la sustitución de una VPTask por una Task. En particular, en las líneas 1023-1025 se realiza un primer chequeo para

```
rule CopyPlanItem {
    from
        input: cmmnext!PlanItem(
            if input.oclIsTypeOf(cmmnext!PlanItem) then
                -- Borrar si el elemento al que referencian tiene como Variant asociada DELETE
                if (
                    thisModule -> hasDELETEAsSelectedVariantOfVarPoint(input.definitionRef.id)
                ) then
                    false
                else
                -- Chequeo si el Stage que lo contiene es un VStage
                    if (cmmnext!VStage -> allInstances() -> exists (VS | VS.stage.planItem.includes(input)))
                        then
                            -- y ademas, si hay al menos un Var Point que lo eligio como variante.
                            -- Entonces, en ese caso se MANTIENE el elemento. En caso contrario, se ELIMINA
                            if (confvar!VarPoint.allInstances()
                                -> exists (VPoint | VPoint.selectedVariants.first().id =
                                    cmmnext!VStage -> allInstances() ->
                                    select(VS | VS.stage.planItem.includes(input)) -> first().name)
                            ) then
                                true
                            else
                                false
                            endif
                        else
                            true
                    endif
                endif
            else
                false
            endif
        )
    to
        output: CMMN!PlanItem(
            definitionRef <- input.definitionRef,
            documentation <- input.documentation,
            entryCriteria <- input.entryCriteria,
            exitCriteria <- input.exitCriteria,
            extensionDefinitions <- input.extensionDefinitions,
            extensionValues <- input.extensionValues,
            id <- input.id,
            itemControl <- input.itemControl,
            name <- input.name
        )
}
```

Figura 4.8: Regla CopyPlanItem

verificar si la variante asociada es de tipo DELETE - en ese caso no se realiza la transformación. Este patrón se repite en cada transformación que recibe como entrada un VPElement.

- En la figura 4.13 se presenta cómo se realiza la sustitución de una VPTask por un Stage.

Se trabajó en la implementación de cuatro ejemplos con el objetivo de verificar el correcto funcionamiento de la transformación. Para ello, se definieron cuatro duplas de archivos (uno conteniendo el proceso CMMNext, y otro una configuración determinada), de forma tal que se obtuvieron cuatro archivos transformados. En los mismos se cubrieron los siguientes escenarios:

- "EjemploV1": Se tienen dos VPoints de tipo VPMilestone, dos VPoints de tipo VPTask, y un VPoint de tipo VPStage. Se decide sustituír cada VPoint por un VElement de su misma clase (es decir, los VPMilestone se sustituyen por VMilestone, VPStage por VStage, y VPTask por VTask. En este caso se prueba que las sustituciones por elementos de la misma clase funciona correctamente.

- "EjemploV2": Se cuenta con el mismo escenario inicial que en el ejemplo anterior, pero en este caso se decide eliminar todos los elementos seleccionado la variante DELETE. En este caso se prueba que los elementos se eliminan correctamente.

```
rule CopyTask {
    from
        input: cmmnext!Task(
            if (input.oclIsTypeOf(cmmnext!Task)) then
                -- Si es una task creada dentro de un VTask, no transformarla.
                if (cmmnext!VTask -> allInstances() -> exists (VT | VT.task.id = input.id)) then
                    false
                else
                    -- Chequeo si el Stage que lo contiene es un VStage
                    if (cmmnext!VStage -> allInstances()
                            -> exists (VS | VS.stage.planItemDefinitions.includes(input)))
                        then
                            -- y ademas, si hay al menos un Var Point que lo eligio como variante.
                            -- Entonces, en ese caso se MANTIENE el elemento. En caso contrario, se ELIMINA
                            if (confvar!VarPoint.allInstances()
                                -> exists (VPoint | VPoint.selectedVariants.first().id =
                                    cmmnext!VStage -> allInstances() ->
                                    select(VS | VS.stage.planItemDefinitions.includes(input)) -> first().name)
                            ) then
                                true
                            else
                                false
                            endif
                        else
                            true
                    endif
                endif
            else
                false
            endif
        )
    to
        output: CMMN!Task(
            documentation <- input.documentation,
            extensionDefinitions <- input.extensionDefinitions,
            extensionValues <- input.extensionValues,
            id <- input.id,
            inputs <- input.inputs,
            outputs <- input.outputs,
            isBlocking <- input.isBlocking,
            name <- input.name

        )
}
```

Figura 4.9: Regla CopyTask

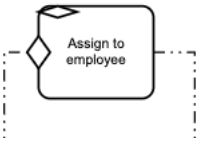


Figura 4.10: El VElement que se elija para sustituir la VTask “Assign to employee” heredará el sentry de entrada

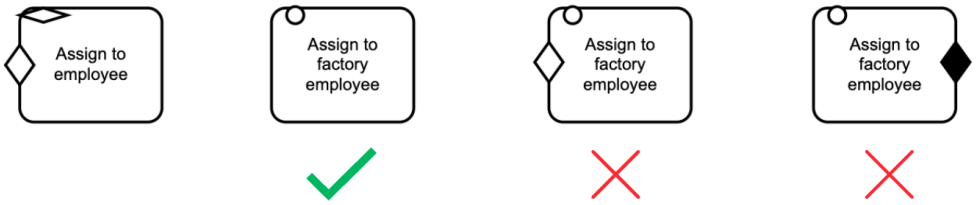


Figura 4.11: Dado un VPoint, se deberá crear un VElement sin sentries

- “EjemploV3”: Inicialmente se cuenta con el mismo escenario, pero se decide eliminar todos los elementos salvo un VPTask, que es sustituido por un VStage. En este caso se prueba la intercambiabilidad entre VPTask y VStage.

```
rule VPTask2Task {
    from
        input:
            cmmnext!VPTask(
                if thisModule -> hasDELETEAsSelectedVariantOfVarPoint(input.id) then
                    false
                else
                    thisModule -> findVariantInBaseProcess(input.id) -> oclIsKindOf(cmmnext!VTask)
                endif
            )
    using {
        variant: cmmnext!VTask = thisModule -> findVTaskInBaseProcess(input.id);
    }
    to
        output:
            CMMN!Task(
                id <- input.id, -- Toma el ID del VP para mantener referencias
                name <- variant.task.name,
                documentation <- variant.task.documentation,
                extensionDefinitions <- variant.task.extensionDefinitions,
                extensionValues <- variant.task.extensionValues,
                inputs <- variant.task.inputs,
                outputs <- variant.task.outputs,
                isBlocking <- variant.task.isBlocking
            )
}
```

Figura 4.12: Se reemplazan las VPTask por Task

```
rule VPTask2Stage {
    from
        input:
            cmmnext!VPTask(
                if thisModule -> hasDELETEAsSelectedVariantOfVarPoint(input.id) then
                    false
                else
                    thisModule -> findVariantInBaseProcess(input.id) -> oclIsKindOf(cmmnext!VStage)
                endif
            )
    using {
        variant: cmmnext!VStage = thisModule -> findVStageInBaseProcess(input.id);
    }
    to
        output:
            CMMN!Stage(
                autoComplete <- variant.stage.autoComplete,
                documentation <- variant.stage.documentation,
                exitCriteria <- variant.stage.exitCriteria,
                extensionDefinitions <- variant.stage.extensionDefinitions,
                extensionValues <- variant.stage.extensionValues,
                id <- variant.stage.id + '_' + input.id,
                name <- variant.stage.name,
                planItem <- variant.stage.planItem,
                planItemDefinitions <- variant.stage.planItemDefinitions,
                planningTable <- variant.stage.planningTable,
                sentries <- variant.stage.sentries
            )
}
```

Figura 4.13: Se reemplazan las VPTask por Stage

- "EjemploV4": Nuevamente se presenta el mismo escenario de partida, pero se decide eliminar todos los elementos salvo un VPStage, que es sustituido por una VTask. En este caso se prueba el escenario contrario al anterior, testeando la intercambiabilidad entre VPStage y VTask.

Más adelante en el presente informe se presentarán ejemplos más elaborados que permitirán explorar el potencial real del enfoque y de la herramienta.

4.4.1. Transformación de salida de CMMNext a cmmn-js

Continuando con las capacidades del metamodelo CMMNext específico del proyecto, la herramienta de transformación también admite la operación inversa, con-

virtiendo archivos `XMI` a formato `CMMN`. La clase `ReverseTransformation` en Java es el núcleo de esta funcionalidad, haciendo uso de Freemarker para la manipulación de plantillas, de forma análoga a la transformación directa. Cabe destacar que en este caso los archivos resultantes pueden ser cargados tantos en la nueva herramienta cmmnext-js como en cmmn-js (incluso en Camunda), ya que la transformación se hace a partir de un archivo que conforma con CMMN, libre de elementos que representen variabilidad. Sin embargo, debido a limitantes encontradas en los distintos modeladores de `CMMN` existentes hoy en día, no se puede levantar un modelo sin sus especificaciones gráficas por lo que de forma provisoria esta funcionalidad no puede ser cumplida.

Los modeladores consultados sin éxito fueron: CMMN-js (*cmmn-js*, s.f.), Camunda Modeler (*Camunda*, s.f.), Flowable (*Flowable*, s.f.) y CMMN Modeler publicado en Atlassian Marketplace (*CMMN Modeler from Atlassian*, s.f.).

Configuración y Lectura de Archivos XMI La herramienta inicia con la configuración de Freemarker, donde se establecen parámetros de manejo de errores y codificación de caracteres. La función `readXMIFile` lee el archivo XMI, y el contenido se procesa mediante `parseXMIContent`, construyendo un modelo de datos que refleja la estructura `XMI`.

Mapeo de Etiquetas y Atributos El mapeo inverso de etiquetas y atributos se realiza con los métodos `reverseTransformTagName` y `reverseTransformAttributes`. Estos ajustan los elementos `XMI` a sus equivalentes CMMN según el metamodelo CMMN, eliminando atributos específicos de `XMI` y convirtiendo identificadores de manera apropiada.

Integración con Plantillas Freemarker La integración con las plantillas de Freemarker se logra inyectando métodos de transformación al modelo de datos. Esto permite el uso dinámico de dichas transformaciones dentro de las plantillas definidas para la generación del contenido CMMN.

Escritura del Archivo CMMN Resultante El proceso culmina con la escritura del contenido generado en un archivo con extensión `.cmmn` a través de la función `writeToFile`, produciendo un documento listo para uso en herramientas de modelado compatibles con CMMN.

Este enfoque completa la explicación del flujo bidireccional de transformación entre los formatos de modelado CMMN y XMI, ampliando las posibilidades de interoperabilidad dentro del proyecto.

Capítulo 5

Aplicación Práctica

De forma tal de verificar el correcto funcionamiento de las herramientas implementadas, asi como también lograr mostrar el potencial que las mismas alcanzan, se trabajó en el modelado de tres ejemplos presentados en artículos que desarrollan distintos enfoques para representar la variabilidad de procesos. A continuación se brindarán detalles de cada ejemplo, mostrando los resultados obtenidos.

5.1. Ejemplo del artículo de Configurable Declare

Se optó por representar en CMMNext el ejemplo presentado para municipalidades en Configurable Declare (tal cual se observa en 3.4, y se buscará a través de una configuración determinada llegar a representar en CMMN el ejemplo de proceso para "Municipality B" (figura 5.1) presentado en (Schunselaar et al., 2012).

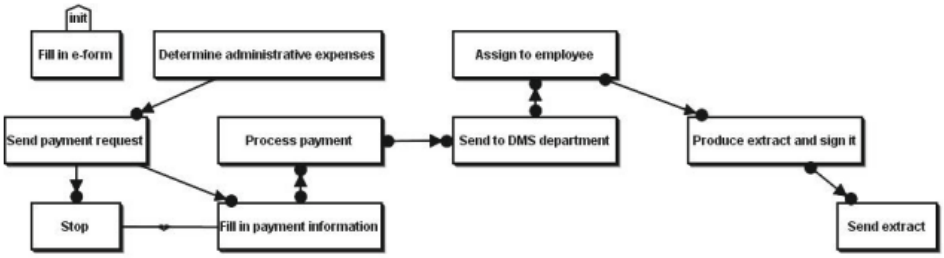


Figura 5.1: Diagrama de la Municipalidad B en Declare

Para ello, se definió a través de la herramienta cmmnext-js la familia presentada en el mismo artículo (siendo posible apreciarla en 5.2), lo cual fue realizado teniendo en cuenta el mapeo entre las restricciones de Configurable Declare y los elementos de CMMN detallados previamente [ver tabla 3.8] en el presente informe. También se generaron las correspondientes variantes, las cuales se muestran en la figura 5.3.
Debido a la naturaleza de CMMN, no fue posible representar la variablidad de las relaciones entre tareas, por lo que únicamente se modelaron VPTasks, las cuales podrán o bien ser mantenidas (siendo reemplazadas por una VTask), o ser eliminadas (al ele-

gir DELETE como la variante asociada a ese VPTask).

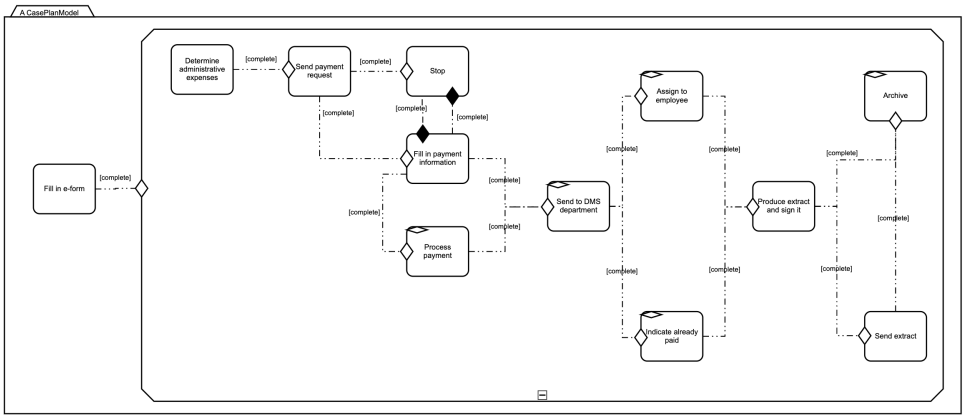


Figura 5.2: Representación de familia de procesos del artículo de Configurable Declare

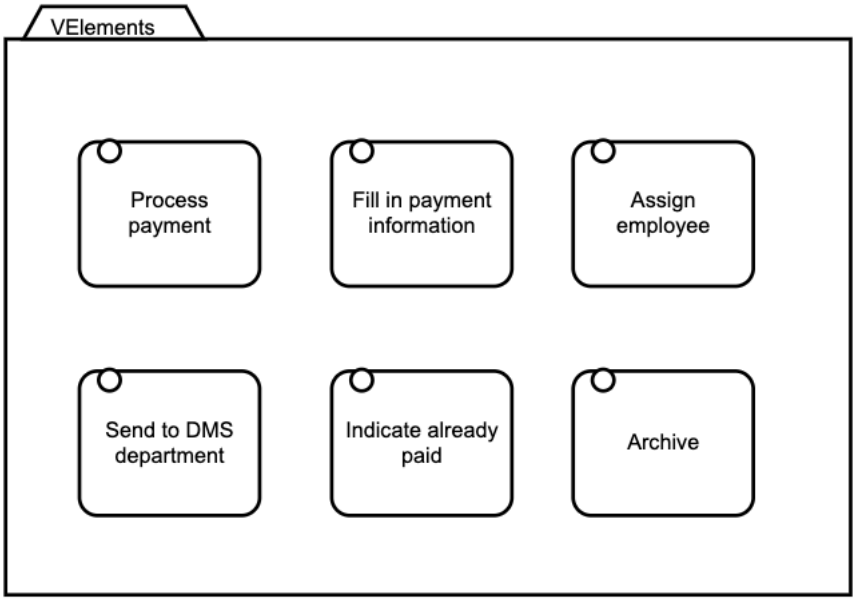


Figura 5.3: Variantes asociados al ejemplo de Configurable Declare

Para poder trabajar con este modelo, se ejecutó la transformación que convierte un archivo `.cmmn` (salida de la herramienta cmmnext-js) a `.xmi`, para así obtener un modelo que conforme con el metamodelo de CMMNext. En la figura 5.4 se puede aprecial el mismo.

Se definió a continuación un modelo de configuración buscando representar la instancia particular para “Municipality B”, la cual se aprecia en la figura 5.5. En el mismo se representan todos los VPElements presentes en el modelo de CMMNext

Figura 5.4: Modelo CMMNext de la representación de familia de procesos del artículo de Configurable Declare

como elementos de tipo `Var Point` (utilizando como id los ids de cada VElement de CMMNext), y las variantes como elementos de tipo `Variant` (utilizando como id el atributo "name" de cada variante de CMMNext).

Para representar el caso particular de "Municipality B", se tomaron las siguientes decisiones de configuración:

- VPTask **Process payment** se sustituye por la variante **Process_payment_variant** (figura 5.6)
- VPTask **Send to DMS department** se sustituye por la variante **Send_to_DMS_department_variant**

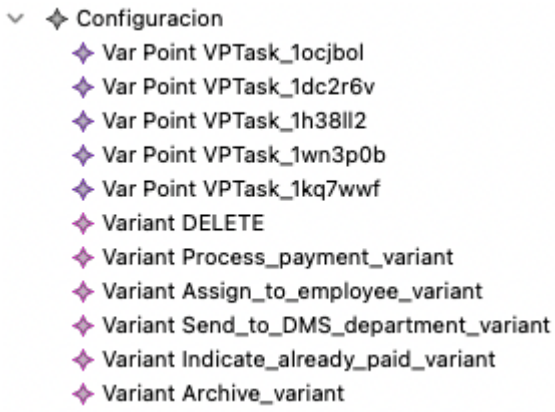


Figura 5.5: Modelo de configuración para el ejemplo de Configurable Declare

- VPTask **Assing_to_employee** se sustituye por la variante **Assign_to_employee_variant**
- VPTask **Indicate already paid** se sustituye por la variante **DELETE**
- VPTask **Archive** se sustituye por la variante **DELETE**

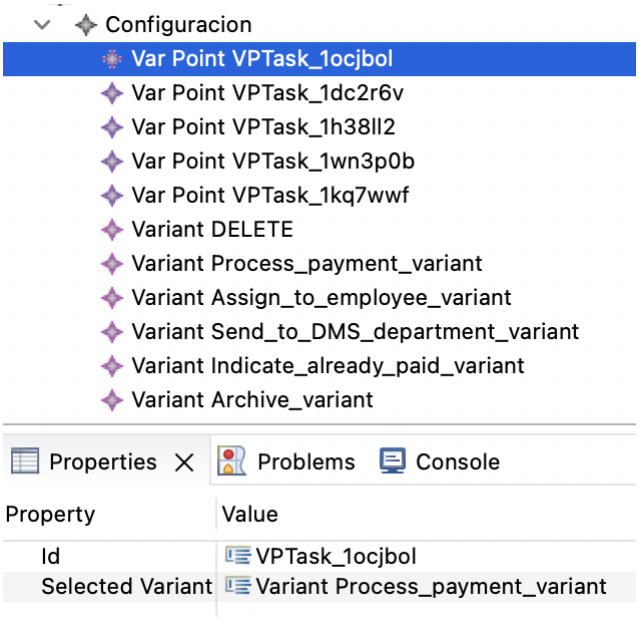


Figura 5.6: Ejemplo de una configuración elegida para determinado Var Point

Con ambos modelos definidos, se procedió a ejecutar la transformación M2M obteniendo como resultado un modelo CMMN como se muestra en la figura 5.7, el conforma con el metamodelo de CMMN.

El último paso llevado a cabo fue convertir este archivo de salida `.xmi` a `.cmmn`, con la transformación que se presentó anteriormente. Tal cual fue mencionado, el resultado

de estas transformaciones no cuenta con los elementos CMMNDI necesarios para que motores gráficos de CMMN como cmmn-js puedan dibujar el diagrama generado. Con el objetivo de cerrar el ciclo para los ejemplos que se desarrollan en el presente informe, se optó por generar manualmente los modelos resultado en la herramienta cmmn-js para permitir visualizar el resultado final de las transformaciones, y para este ejemplo se presenta el diagrama resultado en la figura 5.8.

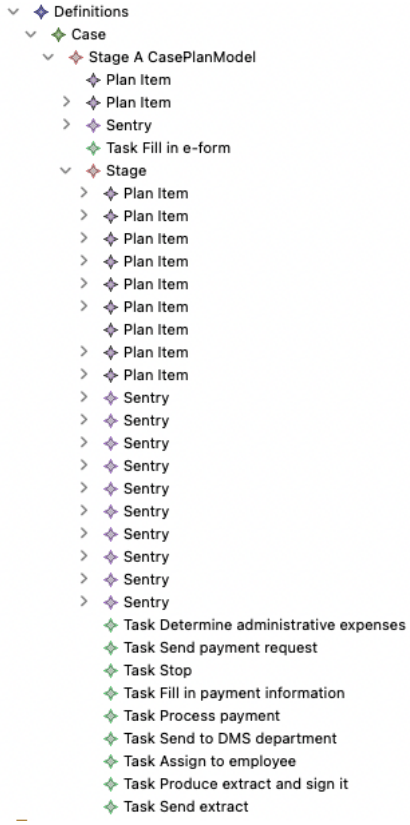


Figura 5.7: Modelo CMMN del ejemplo “Municipality B” del artículo de Configurable Declare

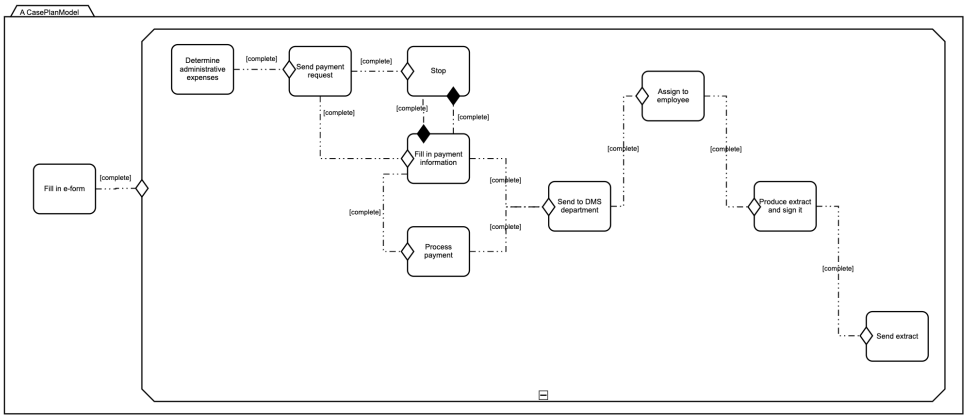


Figura 5.8: Representación gráfica del modelo resultado CMMN del ejemplo "Municipality B"

5.2. Artículo DECO

Se trabajó en el ejemplo presentado en (Rychkova et al., 2011b), el cual representa la fase de solicitud formal del proceso de aprobación de una hipoteca. En la figura 5.9 se representa dicho ejemplo en DECO, donde se observa la manera en que se representa la variabilidad en dicho lenguaje.

Se puede observar que se pueden encontrar tareas y Objetos de Datos (Data Objects)

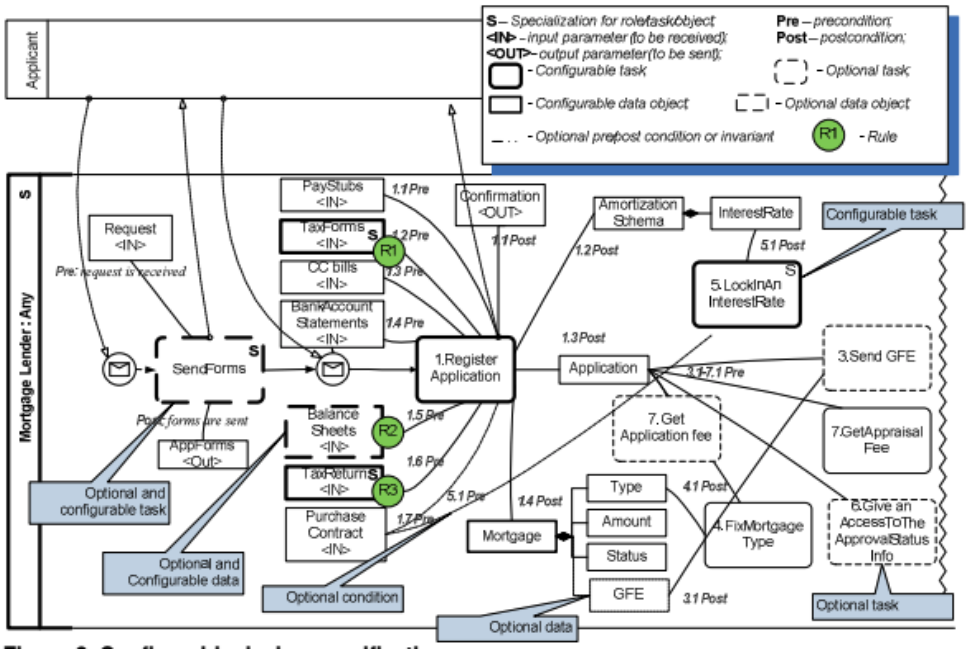


Figura 5.9: Ejemplo del artículo de DECO

configurables. Las tareas configurables presentes son 'SendForms', 'Register Application' y LockInAnInterestRate'. Los objetos de datos configurables (únicamente con la posibilidad de 'KEEP' o 'DELETE' como es el caso de Configurable Declare explicado anteriormente) son: 'TaxForms', 'BalanceSheets', 'TaxReturns' y 'Mortgage'.

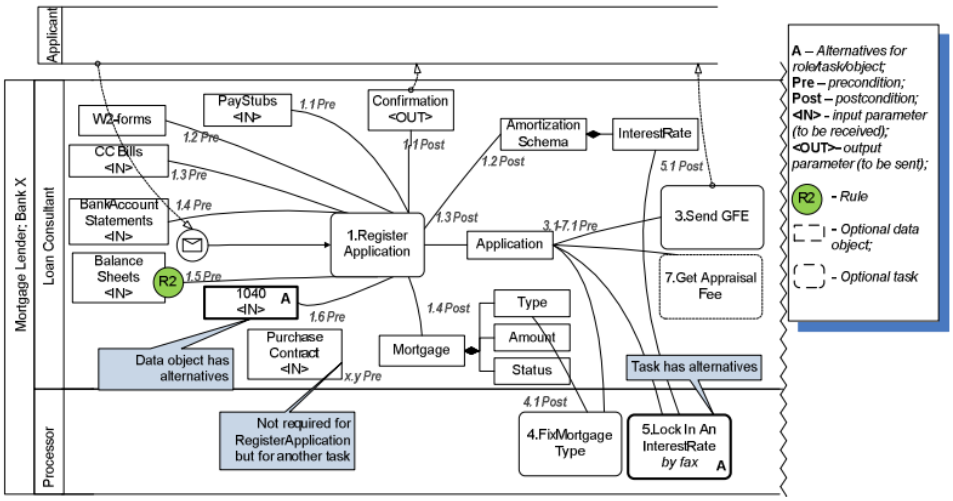


Figura 5.10: Configuración posible descrita en el artículo de DECO

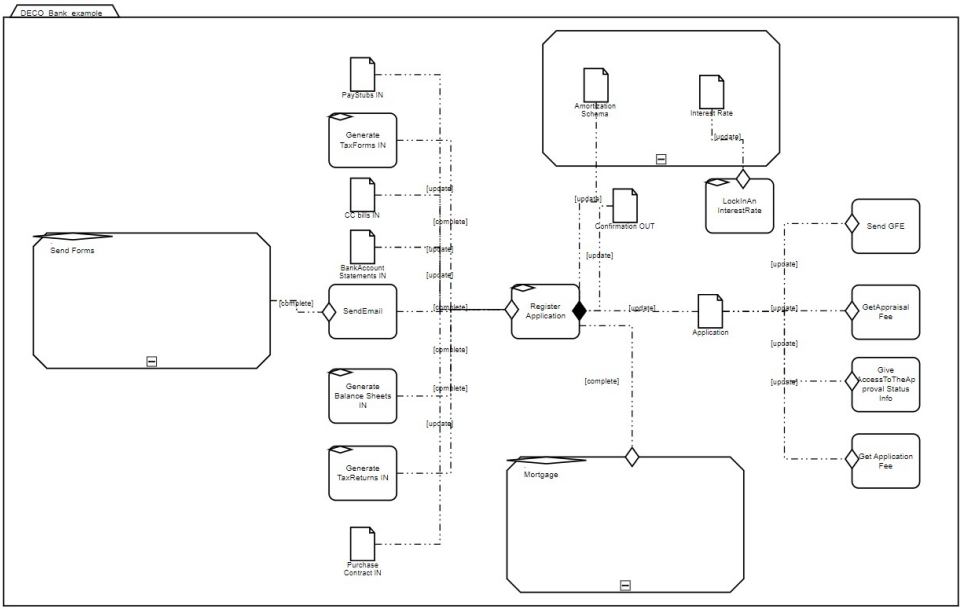


Figura 5.11: Ejemplo del artículo de DECO, representado en CMMNext

Al igual que en el ejemplo de Configurable Declare, el primer paso fue definir el modelo CMMNext utilizando cmmn-js para ello. En la figura 5.11 se puede obser-

var el resultado del mismo, el cual a través de la transformación correspondiente fue transformado de `.cmmn` a `.xmi` para poder utilizarlo como modelo de entrada en la transformación M2M. También se definieron dos archivos distintos con VElements, para representar dos escenarios diferentes que podrán ser generados a partir de la familia definida - los mismos se aprecian en las figuras 5.12 y 5.13.
El caso A de este ejemplo es el que está presente en el artículo, que se puede ver en 5.10. Se puede ver como se opta por borrar la tarea variable 'SendForms', se mantiene intacta 'Register Application' y se opta por la variable 'LockInAnInterestRate by fax'. Por otro lado, de los Data Objects variables se mantiene únicamente 'Balance Sheets' y 'Mortgage'.

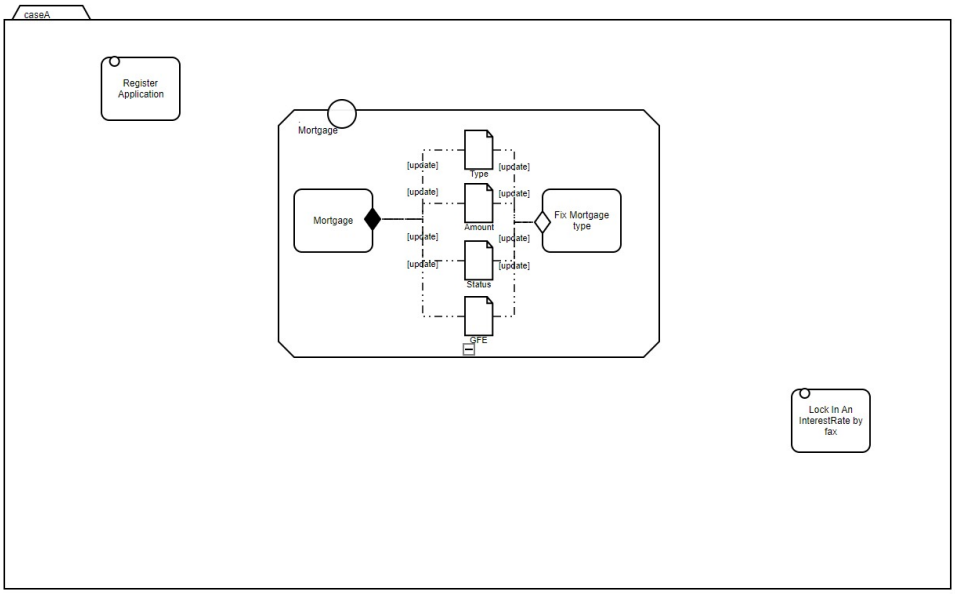


Figura 5.12: VElements que representan el escenario A

El caso B fue generado de forma manual para agregar contenido teniendo en cuenta que se menciona que una posible variable de la tarea 'LockInAnInterestRate', ademas de la tarea presente en el caso A 'LockInAnInterestRate by phone' puede ser 'LockInAnInterestRate by fax' lo cual permite mostrar la funcionalidad implementada de poder sustituir una tarea variable por más de una tarea y no sólo la funcionalidad de borrar o simplemente sustituir un VPoint por un VElement idéntico. Sumado a eso se optó por mostrar elementos que en el artículo habían quedado ocultos.

Para implementar este ejemplo en CMMNext fueron consideradas las correspondencias especificadas en Sección 3.4.2, permitiendo tener un mapeo entre los lenguajes de modelado. Sumado a esto y debido a que en esas correspondencias no se tiene en cuenta la configurabilidad se tienen en cuenta los siguientes aspectos:

- Al no soportar la configurabilidad de los caseFileItems se optó por corresponder los Data Object configurables con VPTasks que tuvieran como acción generar el objeto respectivo.

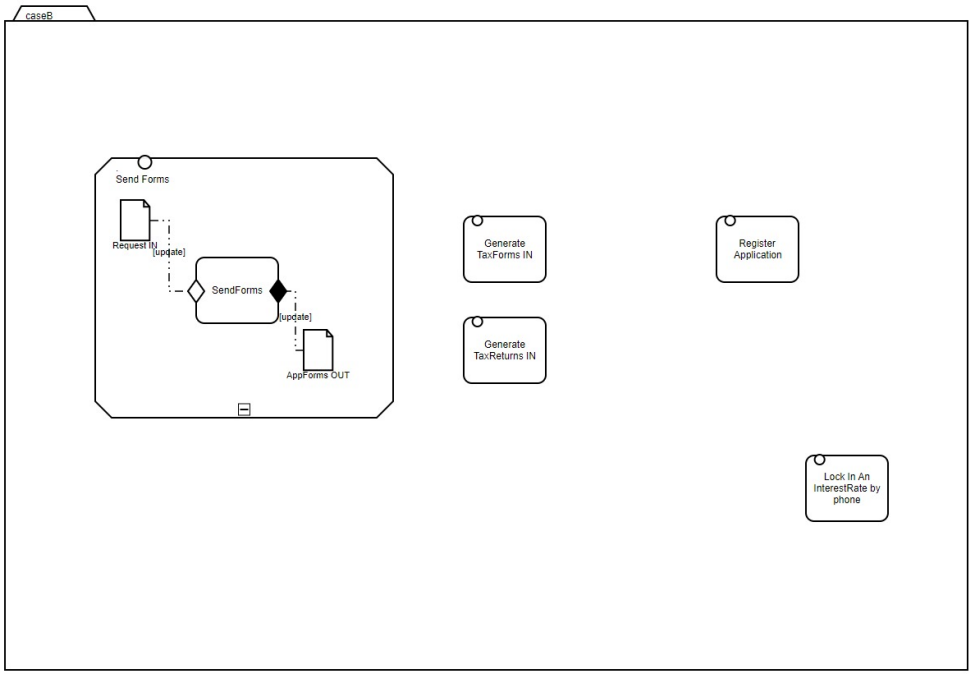


Figura 5.13: VElements que representan el escenario B

- Al no tener la posibilidad de conectar elementos que quedan sueltos luego de la eliminación de un elemento configurable, se eligió crear un VPStage que contuviera todos los elementos relacionados con Mortgage ya que si no al borrarlo quedarían elementos huérfanos.

Tras instanciar un modelo de configuración para cada escenario con las decisiones correspondientes, se obtuvieron dos modelos CMMN que conforman con el metamodelo correspondiente.
Las etapas seguidas durante la transformación son análogas al ejemplo de la sección Sección 5.1.

Finalmente, los `.xmi` resultantes fueron transformados a `.cmmn` para finalizar así el proceso de transformación. Nuevamente fue necesario generar el código CMMNDI manualmente para poder presentar los diagramas finales, presentados en las figuras 5.14 (Caso A) y 5.15 (Caso B). En ambos casos se pueden observar representados los elementos variables seleccionados.
Por un lado para el caso presentado en el artículo, podemos ver la representación en CMMN en 5.14 donde se puede ver que se representa el mismo escenario mostrado en 5.10, donde, como fue mencionado anteriormente, se puede notar la ausencia de las tarea 'SendForms', la presencia de 'Register Application' y de 'LockInAnInterestRate by fax'.
Sumado a eso, podemos encontrar la presencia de los mismos objetos de datos. Esto demuestra el éxito a la hora de replicar el ejemplo deseado.

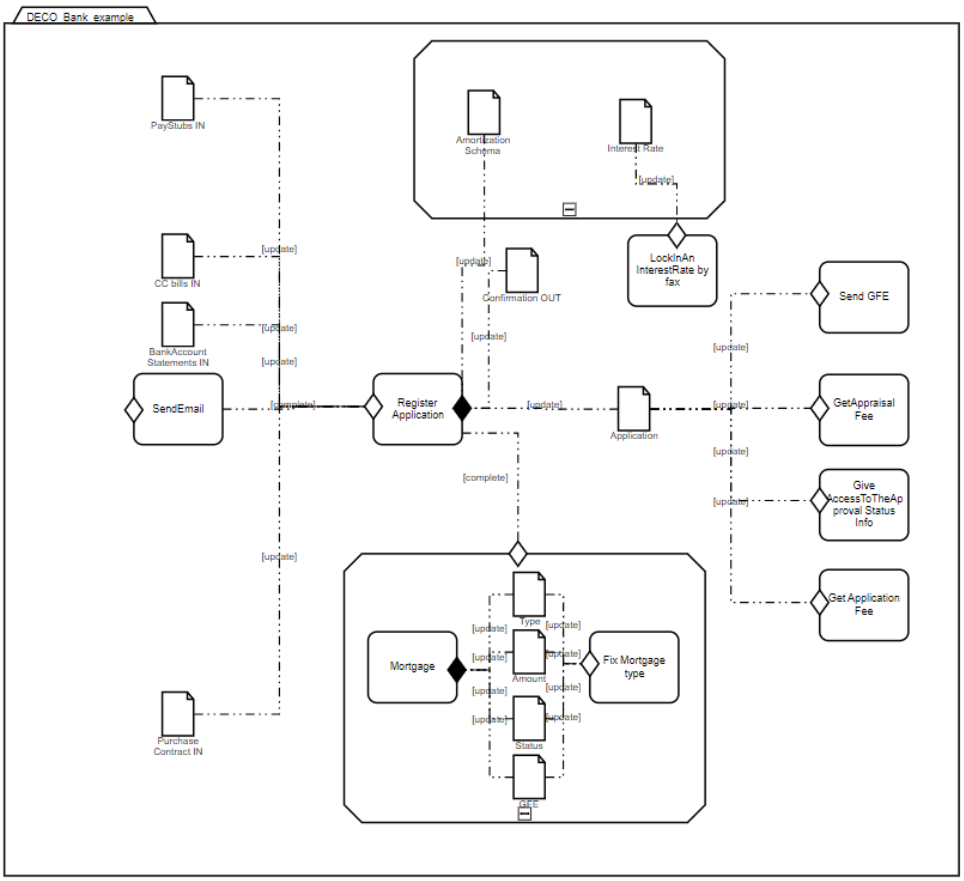


Figura 5.14: Salida final escenario A DECO

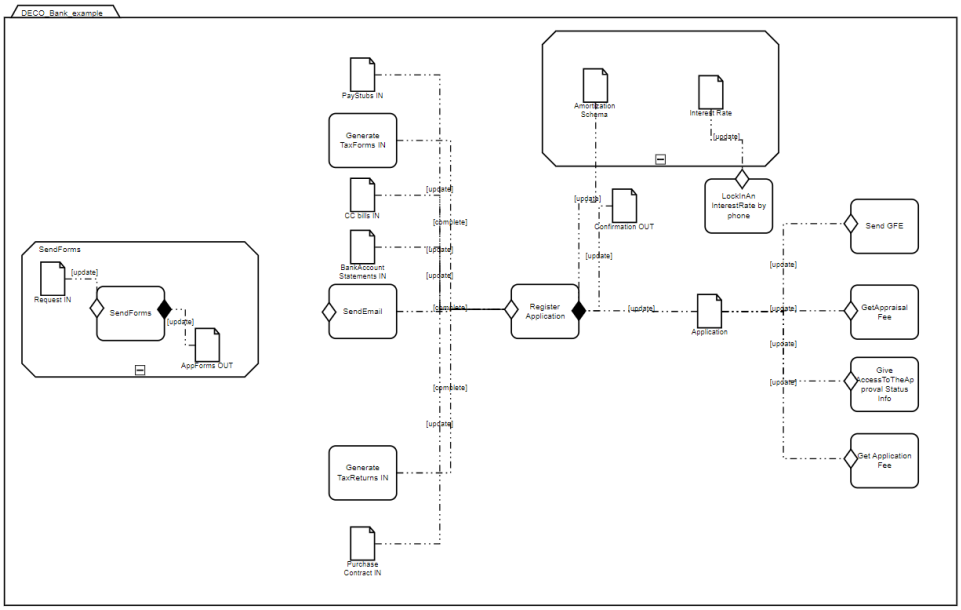


Figura 5.15: Salida final escenario B DECO

5.3. Artículo BPFM

Por otro lado, podemos ver en 5.15, como se representan las configuraciones planteadas anteriormente, destacando la tarea 'LockInAnInterestRate by phone', que muestra el potencial de CMMNext al poder sustituir una variable por otra diferente.

Se trabajó en el ejemplo presentado en (Cognini et al., 2016), el cual representa la fase de solicitud formal de estudiantes para acceder a una universidad y las verificaciones para su aprobación. En la figura 5.16 se representa dicho ejemplo en BPFM, donde se observa la manera en que se modela la variabilidad en dicho lenguaje.

Se puede identificar que las variantes de este modelo en 5.16 varían segun la elegibilidad del estudiante. Los elementos variables se componen por, en primer lugar

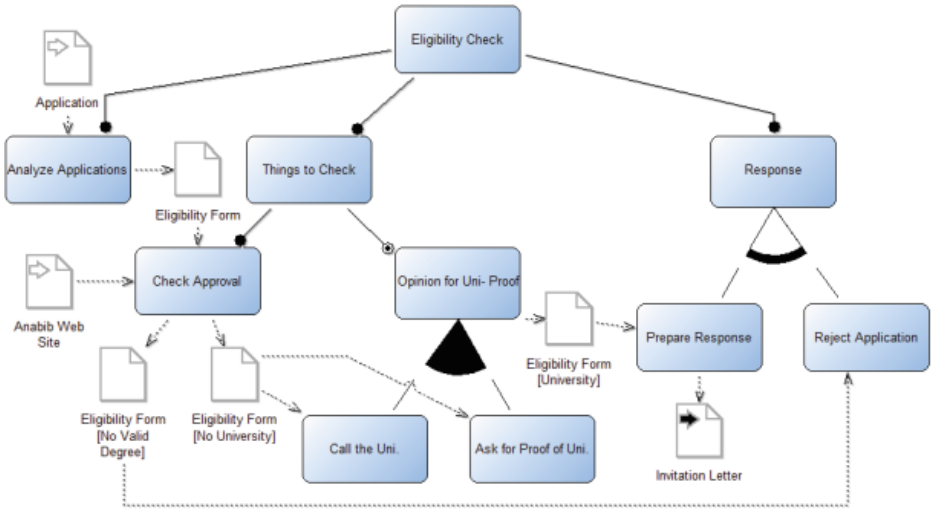


Figura 5.16: Ejemplo del artículo de BPFM, representado en BPFM

'Opinion for Uni-Proof' que aparece sólo en caso de que el candidato cumpla con los requerimientos establecidos, y luego 'Response', que refiere a la respuesta por parte de la universidad, que según el caso en el que se encuentre el estudiante se va a encontrar la tarea 'Prepare Response' o 'Reject Application'.

Para realizar este modelo en CMMNext se utilizaron las correspondencias entre los lenguajes vistas en Sección 3.4.2, permitiendo generar los modelos.
Sumado a eso, al no poder tener CaseFileItems variables, se puede ver como se genera el VPStage 'Prepare Response' con sus respectivos elementos adentro para el caso A. Por otro lado, se genera la VPTask 'Reject Application' para el caso B en lugar de que el elemento variable soporte ambos escenarios posibles, debido a que en el modelo de BPFM, para cada caso los elementos tienen distintas restricciones. Como CMMNext no soporta la variabilidad de las restricciones, se optó por generar dos objetos variables distintos, para poder representar los escenarios de forma completa.

Una vez evaluadas las correspondencias respectivas entre BPFM y CMMN se ilustró este ejemplo en la herramienta cmmnext-js (observable en 5.17) junto a los dos posibles escenarios descritos en el artículo, el caso A (5.18), que describe el caso

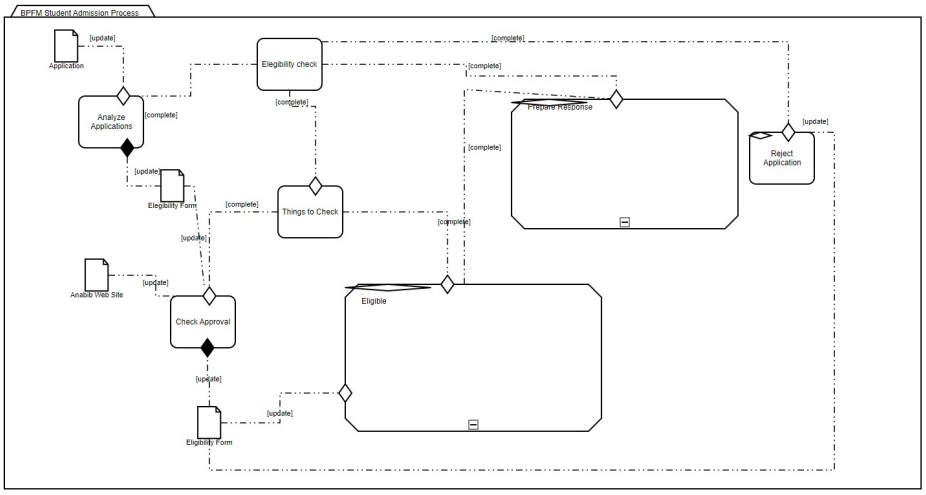


Figura 5.17: Ejemplo del artículo de BPFM, representado en CMMNext

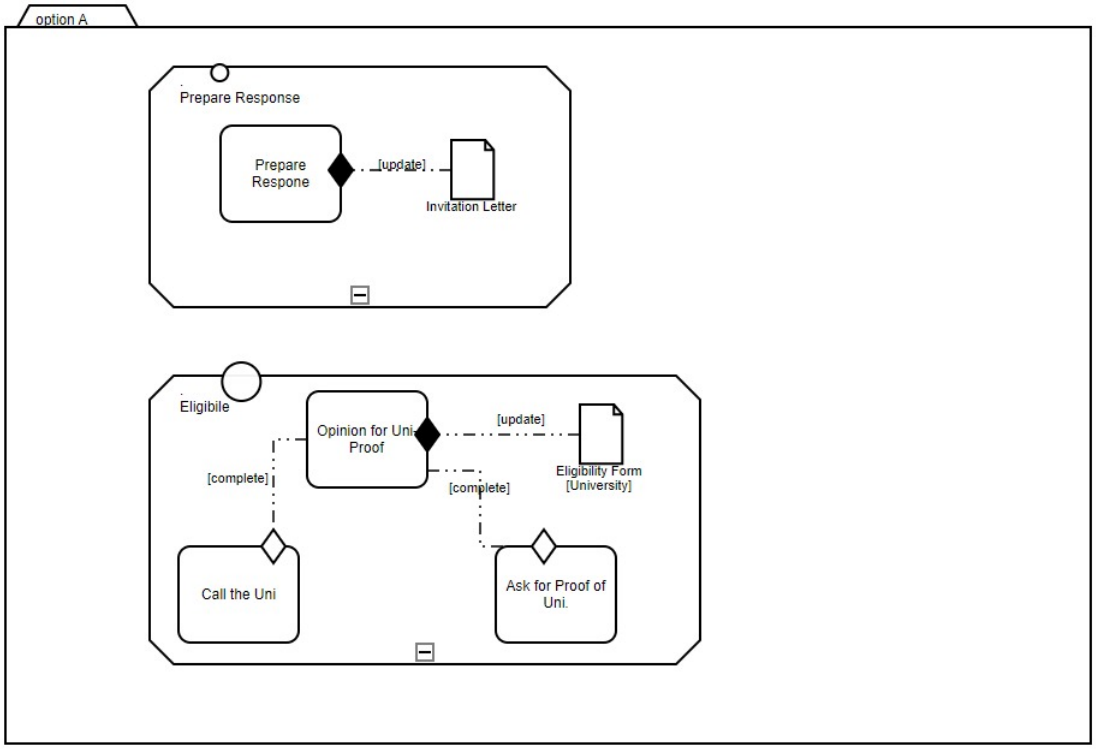


Figura 5.18: VElements que representan el escenario A

en el cual un alumno cumple con ciertas aptitudes necesarias para que su solicitud sea aprobada y el caso B (5.19) que muestra un caso en el cual la solicitud es denegada.

Luego de de generar el ejemplo en cmmnext-js, se procede a efectuar la transformación para obtener las salidas deseadas según la configuración elegida.

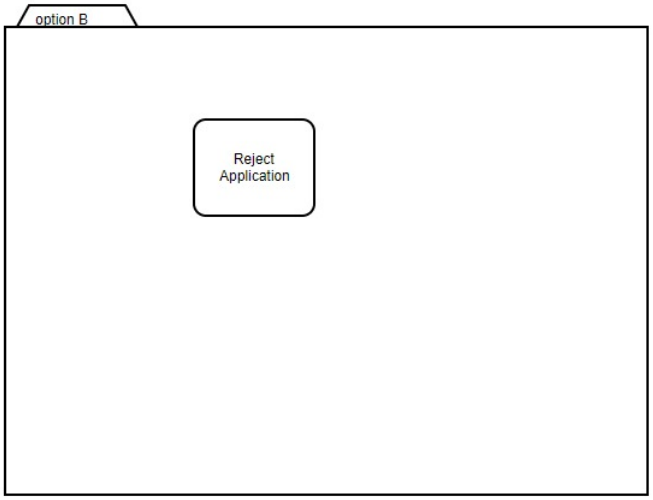


Figura 5.19: VElements que representan el escenario B

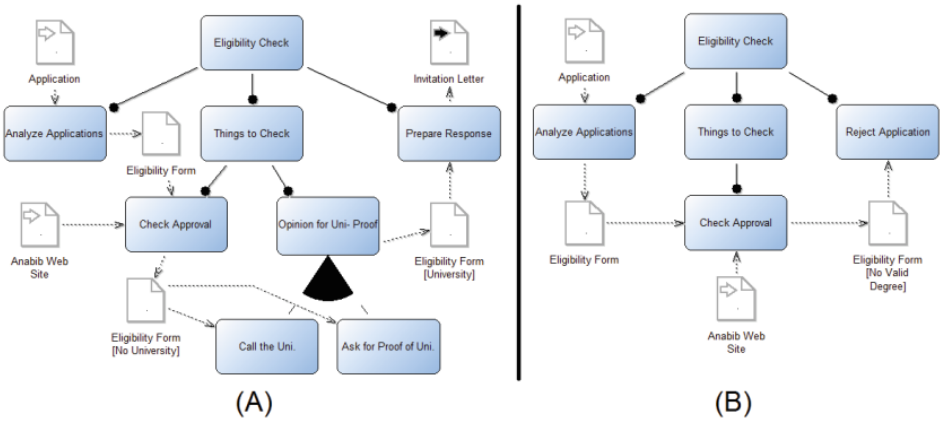


Figura 5.20: Ejemplo del artículo de BPFM, representado en BPFM, donde se muestran los dos escenarios posibles

Las etapas seguidas durante la transformación son análogas al ejemplo de la sección Sección 5.1.

Como en los casos anteriores, debido a limitaciones de la herramienta y su inca-

pacidad para levantar los archivos `.cmmn` generados, se dibujó la salida esperada para ambos casos (5.21 para el caso A) y (5.22 para el caso B), cerrando así el ciclo de este ejemplo.

Se puede ver para cada caso la similitud con sus respectivas representaciones en BPFM (5.20), logrando una mucho mayor similitud en comparación con los modelos configurables, dónde hubo que tomar decisiones de diseño para poder representar los ejemplos con la mayor exactitud posible en cuanto a los procesos de negocio representados, teniendo en cuenta las diferencias de sintaxis.

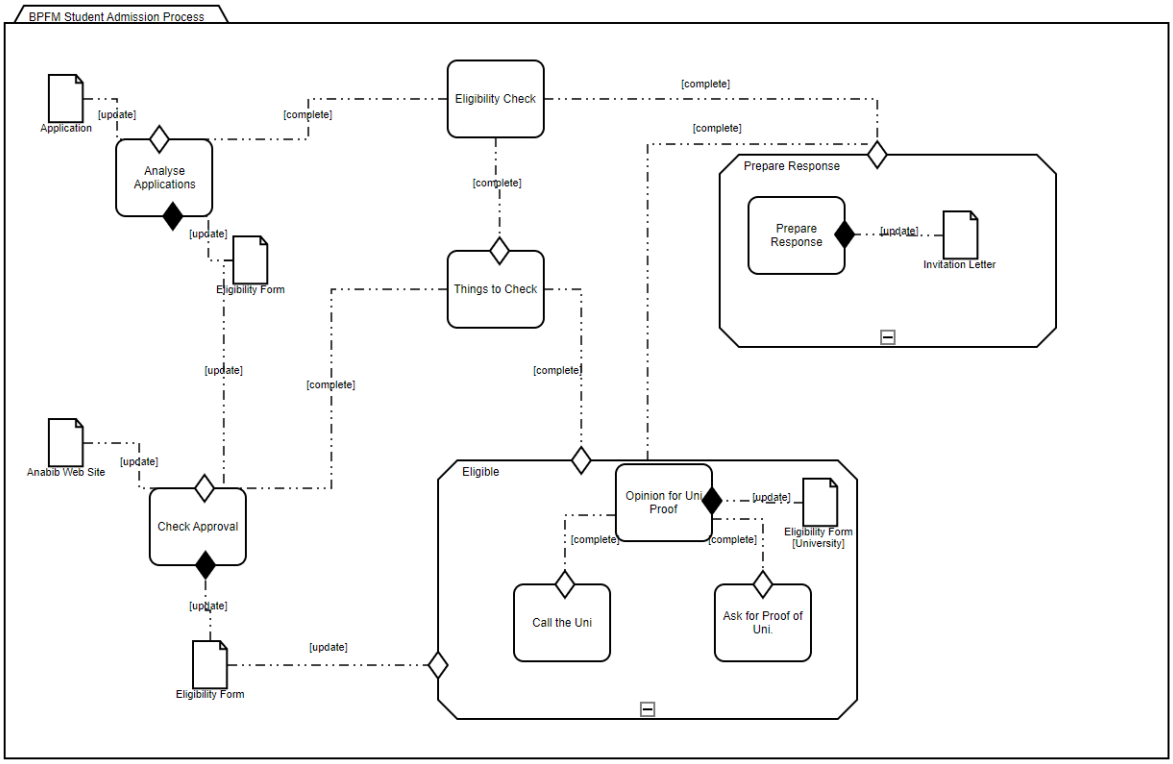


Figura 5.21: Salida final escenario A BPFM

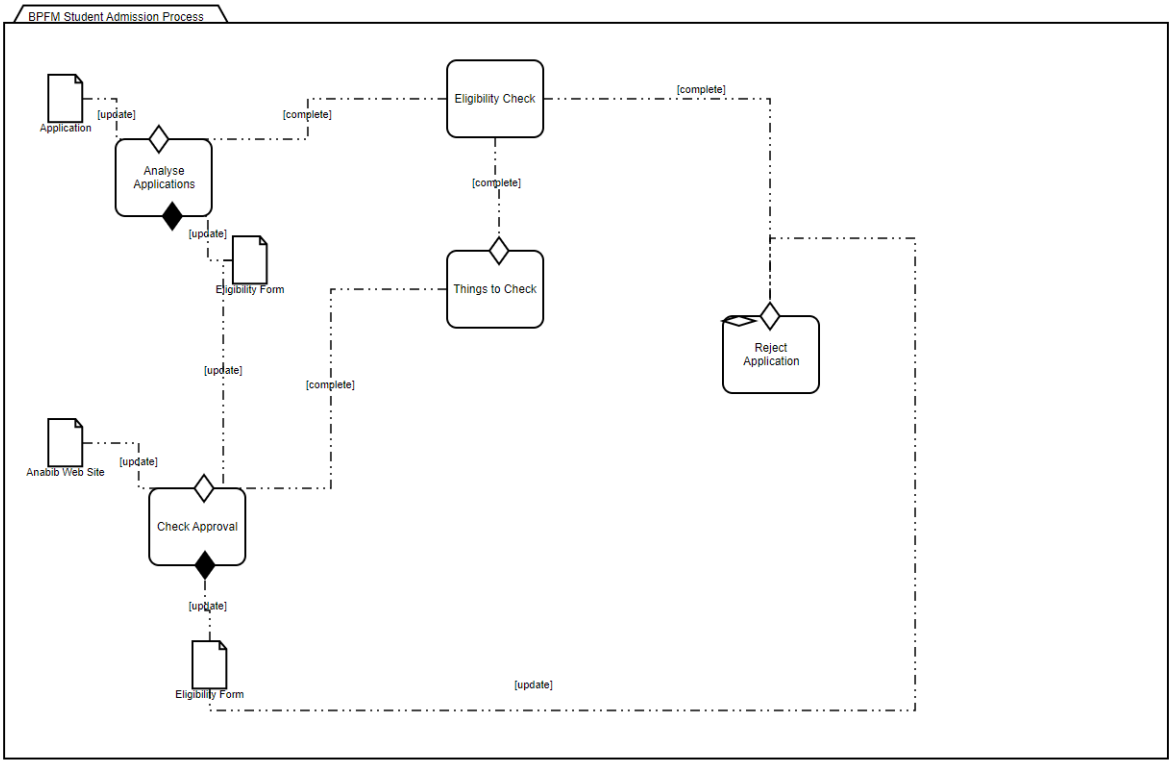


Figura 5.22: Salida final escenario B BPFM

Capítulo 6

Conclusiones y Trabajo Futuro

El objetivo de este proyecto fue definir y evaluar modelos para la especificación de familias de procesos declarativos, a través de una revisión de literatura y un estudio introductorio sobre las metodologías existentes en la gestión de procesos de negocio.

En este contexto, se logró en CMMNext extender un lenguaje declarativo como CMMN para soportar familias de procesos de negocio, describiendo detalladamente tanto las transformaciones realizadas como la aplicación del lenguaje extendido.
Ligado a lo anterior, y teniendo en cuenta que gran parte del trabajo se basó en lo aprendido en lo realizado al extender BPMN (dando lugar a BPMNext), también los usuarios que estén al tanto de esta extensión hallarán un entorno de alguna manera "familiar" al trabajar con CMMNext. Esto es de gran importancia, ya que al ser BPMNext una extensión de un lenguaje imperativo, se entiende que CMMNext se complementa de muy buena manera gracias a su naturaleza declarativa.

Para lograr estos objetivos, fue fundamental el uso de los conceptos de ingeniería dirigida por modelos. Un claro ejemplo de utilidad de MDE fue poder reutilizar el metamodelo de CMMN para crear el metamodelo de CMMNext extendiendo el anterior, logrando de esta forma imitar funcionalidades de forma eficaz. Sumado a esto, se utilizaron en M2M y M2T, dos de sus principales componentes, demostrando la importancia de este paradigma para este proyecto.

En el contexto más amplio de la transformación digital y la automatización de procesos, el proyecto también resalta la necesidad de soluciones que no solo sean poderosas desde el punto de vista técnico, sino también accesibles y comprensibles para los usuarios finales. Es por esto que se considera que sería un gran avance lograr generar una herramienta integral que abarque desde la especificación hasta el despliegue final refleja la importancia de facilitar el flujo de trabajo completo, reduciendo las barreras y complejidades asociadas con cada etapa del proceso. En particular, que integre todos los pasos llevados a cabo en el presente trabajo en un único lugar, permitiendo realizar el flujo completo (desde la especificación de la familia y sus variante en un entorno gráfico, determinar una configuración deseada, las sucesivas transformaciones y desplegar el resultado final) de forma fácil y dinámica. Esto permitiría poder integrar esta herramienta con BPFM (*BPFM*, s.f.). Sin dudas que representa un gran desafío pero sería de gran utilidad para el usuario final.

Con respecto a lo ya implementado, se considera importante implementar una herramienta que a partir de un archivo .cmmn, pueda generar un modelo CMMNDI que permita representar visualmente el proceso. Sería una excelente forma de concluir el proceso que se presenta en este informe, brindando una herramienta adicional al usuario final para poder entender y utilizar el resultado de las sucesivas transformaciones de forma útil y eficiente.

Adicionalmente, se alienta a continuar expandiendo los límites de la variablidad, permitiendo representar como VPoints y Variants más elementos de CMMN que no fueron tenidos en cuenta en esta primera aproximación. Como se pudo observar en los ejemplos de BPFM, una buena primera mejora en este sentido podría ser soportar la variabilidad de elementos de tipo “Case File”.

Por otro lado, se considera que una nueva línea de trabajo puede enfocarse en buscar confluencias entre BPMNext y CMMNext, dando lugar a un enfoque híbrido para la representación de familias de procesos de negocio. Esto podría dar lugar a representaciones de procesos que actualmente no es posible soportar en ninguna de las dos herramientas, pero que al combinar el potencial de ambas sí lo sea.

Referencias

Acceleo. (s.f.). https://wiki.eclipse.org/Acceleo/User_Guide. (Accessed: 2023-11-21)

Atl. (s.f.). https://eclipse.dev/atl/. (Accessed: 2023-11-21)

Ayora, C., y cols. (2015). Vivace: A framework for the systematic evaluation of variability support in process-aware information systems. *Information and Software Technology*, *57*, 248-276. Descargado de https://www.sciencedirect.com/science/article/pii/S0950584914001268

Bernardi, M., y cols. (2012, 09). Development of flexible process-centric web applications: An integrated model driven approach.. doi: 10.1109/WSE.2012.6320534

Bpfm. (s.f.). https://gitlab.fing.edu.uy/open-coal/bpfm. (Accessed: 2023-11-22)

bpmn-estándar. (s.f.). https://www.omg.org/spec/BPMN/2.0/. (Accessed: 2023-11-22)

Brambilla, M., y cols. (2012). Mdse principles. En *Model-driven software engineering in practice* (pp. 7–24). Cham: Springer International Publishing. Descargado de https://doi.org/10.1007/978-3-031-02546-4_2 doi: 10.1007/978-3-031-02546-4_2

Calegari, D., y cols. (2019). *Generación automática de variantes en familias de procesos de negocio.* Comisión Sectorial de Investigación Científica (CSIC). (Informe final de investigación, 2017-2019)

Camunda. (s.f.). https://camunda.com/. (Accessed: 2023-11-22)

Case management model and notation. (s.f.). https://www.omg.org/spec/CMMN. (Accessed: 2023-11-22)

cmmn-js. (s.f.). https://bpmn.io/toolkit/cmmn-js/. (Accessed: 2023-11-22)

Cmmn modeler from atlassian. (s.f.). https://marketplace.atlassian.com/apps/1216838/cmmn-modeler?tab=overview&hosting=server. (Accessed: 2023-11-22)

Cognini, R., y cols. (2016). A case modelling language for process variant management in case-based reasoning. En *Bpm 2016: Business process management workshops* (p. 30-42). Cham, Switzerland: Springer International Publishing.

Delgado, A., y cols. (2017). Bpmn 2.0 based modeling and customization of variants in business process families. En *2017 xliii latin american computer conference (clei)* (p. 1-9). doi: 10.1109/CLEI.2017.8226450

Delgado, A., y cols. (2018a). Modeling of software process families with automated generation of variants. En *The 30th international conference on software engineering & knowledge engineering (seke).* San Francisco, USA.

Delgado, A., y cols. (2018b, 07). Modeling of software process families with automated generation of variants (s). En (p. 330-371). doi: 10.18293/SEKE2018-019

Delgado, A., y cols. (2022). Model-driven management of bpmn-based business process families. *Software & Systems Modeling*, *21*, 2517–2553.

Eclipse modelling tools. (s.f.). `https://www.eclipse.org/downloads/packages/release/2022-12/r/eclipse-modeling-tools`. (Accessed: 2023-11-21)

Flowable. (s.f.). `https://www.flowable.com/`. (Accessed: 2023-11-22)

Freemarker. (s.f.). `https://freemarker.apache.org/index.html`. (Accessed: 2023-11-22)

Goedertier, S., y cols. (2015). Declarative business process modelling: principles and modelling languages. *Enterprise Information Systems*, *9*(2), 161-185. doi: 10.1080/17517575.2013.830340

Hasić, F., y cols. (2017, 11). Integrating processes, cases, and decisions for knowledge-intensive process modelling..

Jalali, A. (2021). Evaluating perceived usefulness and ease of use of cmmn and dcr. En *Enterprise, business-process and information systems modeling* (pp. 147–162). Cham: Springer International Publishing.

Kent, S. (2002). Model driven engineering. En M. Butler, L. Petre, y K. Sere (Eds.), *Integrated formal methods* (pp. 286–298). Berlin, Heidelberg: Springer Berlin Heidelberg.

Kitchenham, B., y cols. (2007). Guidelines for performing systematic literature reviews in software engineering. , *2*.

La Rosa, M., y cols. (2011). Configurable multi-perspective business process models. En *Volume 36, issue 2,* (p. 313-340).

La Rosa, M., y cols. (2013). Business process variability modeling: A survey. *BPM Center Report BPM-13-16*.

Martínez-Ruiz, T., y cols. (2008). Towards a spem v2.0 extension to define process lines variability mechanisms. En *Software engineering research, management and applications* (pp. 115–130). Berlin, Heidelberg: Springer Berlin Heidelberg. Descargado de `https://doi.org/10.1007/978-3-540-70561-1_9` doi: 10.1007/978-3-540-70561-1_9

Molina, J., y cols. (2013). *Desarrollo de software dirigido por modelos: conceptos, métodos y herramientas.*

OMG, O. M. G. (2009). *Case management process modeling (cmpm) request for proposal.* http://www.omg.org/cgi-bin/doc?bmi/2009-09-23. (OMG Document: Bmi/2009-09-23. Accessed: 2023-06-04)

Reichert, M., y cols. (2012). Enabling flexibility in process-aware information systems: Challenges, methods, technologies. En *Enabling flexibility in process-aware information systems: Challenges, methods, technologies.* Springer-Verlag, Berlin. Descargado de `https://www.researchgate.net/publication/229124837_Enabling_Flexibility_in_Process-Aware_Information_Systems_Challenges_Methods_Technologies`

Rychkova, I., y cols. (2011a). The old therapy for the new problem: Declarative configurable process specifications for the adaptive case management support. En *Bpm 2016: Business process management workshops* (p. 420-432). Cham, Switzerland: Springer International Publishing.

Rychkova, I., y cols. (2011b). Towards adaptability and control for knowledge-intensive business processes: Declarative configurable process specifications. En *2011 44th hawaii international conference on system sciences* (p. 1-10). doi: 10.1109/HICSS.2011.452

Schunselaar, D., Maggi, F., Sidorova, N., y van der Aalst, W. (2012). Configurable declare: Designing customisable flexible process models. En *On the move to meaningful internet systems: Otm 2012* (p. 20-37). Cham, Switzerland: Springer International Publishing.

Weske, M. (2019). *Business process management: Concepts, languages, architectures.*

Heidelberger Platz 3, 14197 Berlin, Germany: Springer-Verlag Berlin Heidelberg.

What is case management model and notation (cmmn). (s.f.). `https://www.visual-paradigm.com/guide/cmmn/what-is-cmmn/`. (Accessed: 2023-06-04)

Zensen, A., y cols. (2018). A comparison of flexible bpmn and cmmn in practice: A case study on component release processes. En *2018 ieee 22nd international enterprise distributed object computing conference (edoc)* (p. 105-114). IEEE.