Universidad de la República

Facultad de Ingeniería

Instituto de Computación

Modelado de la variabilidad en Procesos de Software

Informe de Proyecto de Grado

Fabiana Roldán
Marcela Viera

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

Montevideo - Uruguay
Setiembre, 2016

Resumen

Un proceso de software es un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos que son necesarios para concebir, desarrollar, instalar y mantener un producto de software. La variabilidad en procesos de software es la habilidad que tiene un sistema de software construido siguiendo este proceso, de ser cambiado, personalizado o configurado para ser usado en un contexto particular. En los últimos años han surgido diversas propuestas para tratar la variabilidad de procesos, pudiendo ser modelada de diferentes formas.

El objetivo de este proyecto es investigar enfoques existentes para el modelado de la variabilidad en procesos de software, analizando ventajas y desventajas de cada uno, el soporte tecnológico existente y la posible generalización a procesos de negocio en general (más allá de procesos de software). Particularmente, interesa analizar propuestas basadas en el lenguaje SPEM 2.0, el cual permite especificar procesos de desarrollo de software mediante una notación propia.

Para cumplir con los objetivos se analizaron los diferentes enfoques propuestos en la literatura. Además, se desarrolló un prototipo que permite la adaptación de modelos de procesos de software con variabilidad, a través de una interfaz gráfica sencilla. La especificación del modelo de proceso se realizó utilizando el enfoque vSPEM a partir de la herramienta vEPF. Finalmente se realizó un caso de estudio que muestra el uso de la herramienta desarrollada.

Índice

1. Introducción 5
2. Estado del arte 8
2.1. Procesos de negocio 8
2.1.1. Modelado de procesos de negocio 9
2.2. Variabilidad 11
2.2.1. Líneas de Productos de Software (SPLs) 12
2.2.2. Líneas de Procesos de Software (SPrLs) 13
2.3. Modelado de la Variabilidad 14
2.3.1. SPEM 2.0 14
2.3.2. Modelo de Características 21
2.3.3. Orthogonal Variability Model (OVM) 22
2.3.4. SmartySPEM 24
2.3.5. BPMNt 26
2.3.6. CASPER 31
2.3.7. Common Variability Language (CVL) 37
2.3.8. vSPEM 40
2.3.9. Resúmen 44
3. Requerimientos y solución propuesta 47
3.1. Especificación de requerimientos 47
3.1.1. Requerimientos 47
3.1.2. Definición de casos de uso 48
3.2. Solución propuesta 50
3.2.1. Enfoque seleccionado 50
3.2.2. Solución propuesta 53

3.2.3. Implementación 58
4. Caso de Estudio 62
5. Conclusiones y trabajo a futuro 79
6. Referencias 82
7. Anexo 85
Anexo 1: Manual de usuario 85
Anexo 2: Gestión del Proyecto 92
A2.1. Comunicación 92
A2.2. Cronograma planificado 93
A2.3. Cronograma real 94

Capítulo 1

Introducción

Este trabajo se enmarca en el trabajo de investigación del grupo COAL del InCo, en la línea de Gestión y Tecnologías de Procesos de Negocio. Entre los objetivos se incluye el estudio del marco teórico de Procesos de Negocio, variabilidad de procesos, desarrollo de software con base en procesos y relación con otros paradigmas, estándares asociados y aplicaciones. Asimismo, este trabajo se enmarca dentro de las colaboraciones en esta línea de investigación de Procesos de Negocio con el Grupo de investigación ALARCOS de la Universidad de Castilla - La Mancha y con el cluster de BPM de la Universidad de Innsbruck, Austria.

Un Proceso de Negocio (PN) [1] es un conjunto de actividades realizadas en coordinación en un entorno organizacional y técnico, para alcanzar un objetivo del negocio. Un Modelo de Procesos de Negocio [1] consiste en la representación de un conjunto de actividades y restricciones de ejecución entre ellas, que puede ser modelado con diversos lenguajes y estándares.

Un Proceso de Software es un proceso de negocio específico para una empresa de software [2]. Como tal, puede ser modelado con diversos lenguajes también, dentro de los cuales se encuentra SPEM 2.0 [3] (estándar desarrollado por la OMG para la representación de modelos de procesos de software). Una Línea de Procesos de Software (SPrL) [4] es una familia de procesos que tiene como objetivo proporcionar las técnicas y mecanismos para modelar los aspectos comunes y variables de una familia de procesos, así como apoyar a la personalización de los procesos de software de acuerdo a las necesidades específicas del dominio del proceso de software. La mayoría de los mecanismos de variabilidad [5] para una SPrL se basan en puntos de variación (lugares en los que se produce la variabilidad) y variantes (elementos concretos que ocupan dichos puntos). Cada punto de variación se asocia con un

conjunto de variantes que corresponden a diseñar alternativas para resolver la variabilidad, evitando de esta forma modelar cada variante separada.

La Variabilidad en procesos de software [6] es la habilidad que tiene un sistema de software de ser cambiado, personalizado o configurado para ser usado en un contexto particular. Dicha variabilidad es capturada mediante la definición de características requeridas del software y las distintas variaciones que estas características podrían implicar.

En la última década han surgido diversas propuestas para tratar la variabilidad de procesos con foco en el modelado de familias de procesos con variantes, evitando modelar cada variante en forma separada, lo que implica duplicación y mantenimiento de las partes comunes [7][8]. Estas propuestas proponen definir una familia de procesos mediante un proceso base común (denominado proceso configurable) más las variantes que cada proceso específico requiere. Existen dos visiones principales para modelar el proceso base: incluir todas las posibles variantes seleccionando luego los elementos para cada variante específica (variabilidad por restricción) o incluir sólo elementos en común agregando luego para cada variante sus elementos específicos (variabilidad por extensión). De esta forma, cada variante es derivada desde el proceso configurable en la particularización, eliminando o agregando fragmentos del modelo, según parámetros de configuración. En general los lenguajes de modelado de procesos no soportan en forma explícita la representación de variabilidad, ni existe soporte metodológico ni de herramientas para tratar con familias de procesos [9].

El objetivo de este proyecto es investigar enfoques existentes para el modelado de la variabilidad en procesos de software, en particular utilizando el lenguaje SPEM 2.0, analizando ventajas y desventajas de cada uno, el soporte tecnológico existente y la posible generalización a procesos en general (de negocio).

Para lograr cumplir ese objetivo se establecieron los siguientes objetivos específicos:

- **OE1:** Realizar un informe de estado del arte sobre enfoques existentes para el modelado de la variabilidad en procesos de software.
- **OE2:** Seleccionar/adaptar/extender uno de los enfoques existentes para su estudio más detallado.
- **OE3:** Desarrollar un prototipo de herramienta de soporte a la propuesta seleccionada.
- **OE4:** Aplicar la propuesta a un caso de estudio utilizando la herramienta desarrollada.

El presente documento se encuentra organizado de la siguiente manera:

- El Capítulo 2 contiene el estado del arte con los conceptos que se entienden necesarios para la comprensión del proyecto.
- En el Capítulo 3 explicamos cuál es el enfoque seleccionado para la realización del prototipo, y lo que motivó dicha elección. Además, se realiza una descripción de los requerimientos funcionales y no funcionales, así como de la solución propuesta, tecnologías utilizadas y detalles de implementación y testeo.
- El Capítulo 4 plantea el caso de estudio utilizado.
- El Capítulo 5 describe las conclusiones y trabajo a futuro.

Capítulo 2

Estado del arte

2.1. Procesos de negocio

En [10] *Michael Hammer* y *James Champy* definen un *Proceso de Negocio (PN)* como “una colección de actividades que tomando uno o más tipos de entradas crean una salida que es de valor para un cliente”. Esta definición hace énfasis en el comportamiento de las entradas/salidas de un proceso de negocio declarando sus precondiciones (entradas) y sus postcondiciones (salidas). El proceso en sí se describe de una manera abstracta mediante un conjunto de actividades, suponiendo que este conjunto no implica un ordenamiento de las actividades ni otras restricciones de ejecución.

Las restricciones de ejecución entre las actividades son identificadas por *Davenport* [11], que define un PN como "un conjunto de tareas lógicamente relacionadas llevadas a cabo para lograr un resultado de negocio definido para un cliente o mercado en particular". *Davenport* también considera la relación de las actividades del proceso, incluyendo su orden de ejecución, definiendo un PN como "una ordenación específica de las actividades de trabajo a través del tiempo y del lugar, con un principio, un final, y entradas y salidas claramente identificadas”.

Basado en estas definiciones, *Weske* [1] define un PN como “un conjunto de actividades realizadas en coordinación en un entorno organizacional y técnico, para alcanzar un objetivo del negocio. Cada proceso de negocio es ejecutado por una sola organización, pero puede interactuar con los procesos de negocio realizados por otras organizaciones”.

Por otra parte, Fuggetta [12] define un *Proceso de Software* como un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos que son necesarios para concebir, desarrollar, instalar y mantener un

producto software. Además, se define también como un proceso de negocio específico para una empresa de software [2].

2.1.1. Modelado de procesos de negocio

Un *Modelo de Procesos de Negocio* [1] consiste en la representación de un conjunto de actividades y restricciones de ejecución, permitiendo definir entre otras cosas, un flujo entre ellas.

Una instancia de proceso de negocio representa un caso concreto en una organización, que consiste en un conjunto de instancias de actividades. Cada modelo de proceso de negocio actúa como una plantilla para un conjunto de instancias de procesos, y cada modelo de actividad actúa también como una plantilla para un conjunto de instancias de actividad.

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

BPMN [5] es un metamodelo estándar y una notación definida para la representación explícita de los procesos de negocio. Basados en la especificación de BPMN 2.0, la notación BPMN fue desarrollada para:

- Proporcionar un conjunto rico de elementos para representar situaciones comunes de los procesos de negocio.
- Facilitar la interpretación humana mediante la definición de una representación visual para la mayoría de los elementos de proceso.

Por otra parte, un *Modelo de Procesos de Software* es un caso particular de un modelo de procesos de negocio y por ello, existen también diversos lenguajes para modelarlos [13]. Entre ellos se destacan SPEM 2.0, el lenguaje estándar definido para el modelado de procesos de software establecido por la OMG, y BPMN, que en este contexto trata el proceso de software como un proceso de negocio.

SPEM 2.0 define ciertos elementos creados específicamente para lograr una mejor representatividad y así, conseguir una mejor especificación del modelo de los procesos de software. Dentro de estos elementos se pueden mencionar los *steps*, *workProducts* y *roles,* los cuales se detallan en la Sección 2.3.1.

En la Figura 2.1 y la Figura 2.2 se muestra un ejemplo de modelo de proceso de software representado en BPMN y en SPEM 2.0, respectivamente.

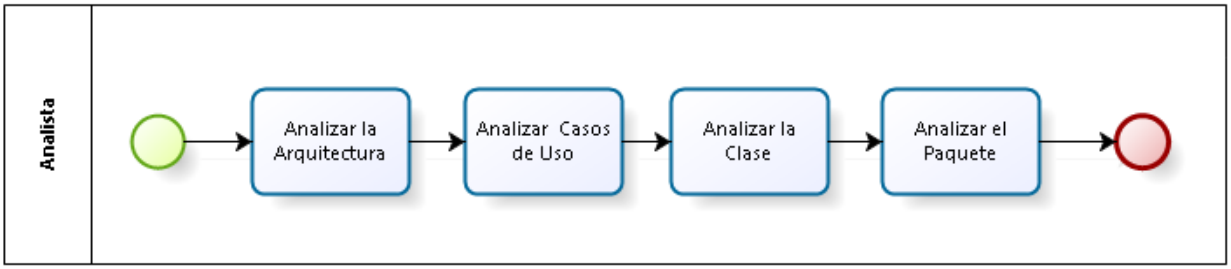


Figura 2.1. Modelo en BPMN.

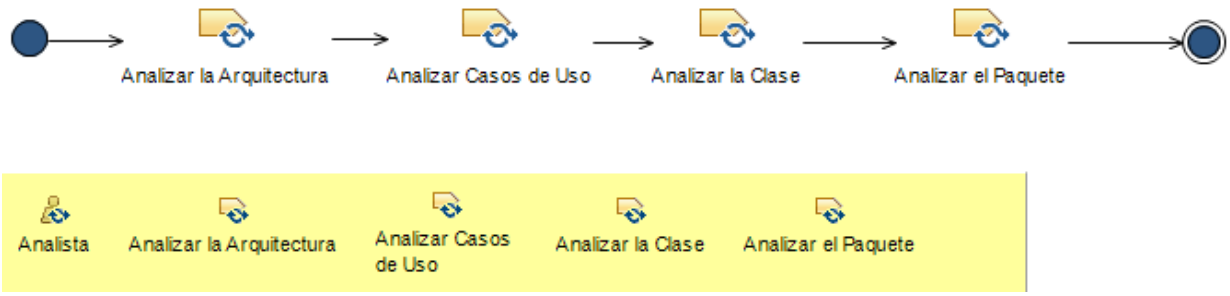


Figura 2.2. Modelo en SPEM 2.0.

La Tabla 2.1 muestra un análisis entre SPEM y BPMN considerando algunas de las características más relevantes [13], las cuales se detallan a continuación:

- Expresividad: capacidad de representar la complejidad y todos los elementos de los procesos de software.
- Reuso: capacidad para promover la reutilización de los elementos contenidos en el modelo del proceso.
- Gestión: apoyo a la gestión de las instancias del proceso (planificación, seguimiento y control).
- Evolución: facilidad para identificar las partes ineficientes del proceso y buscar la mejora y desarrollo de las mismas.
- Multinivel: capacidad de proporcionar puntos de vista de alto nivel del proceso, así como los detallados en gran medida.
- Entendimiento: capacidad para comprender el modelo por todos los implicados en el proceso.
- Integración organizacional: capacidad de establecer la integración y la interacción con los procesos de otras áreas de la organización.

Caracteristicas	SPEM	BPMN	Justificación
Expresividad	Completamente contemplado	Parcialmente contemplado	Todos los elementos básicos se pueden identificar en SPEM. BPMN sólo tiene algunos de estos.
Gestión	Completamente contemplado	Completamente contemplado	No aplica.
Reuso	Completamente contemplado	Parcialmente contemplado	No aplica.
Evolucion	Parcialmente contemplado	Parcialmente contemplado	El modelado realizado mediante el uso de los dos estándares por sí solo no es suficiente para permitir la evolución del proceso.
Multinivel	Completamente contemplado	Completamente contemplado	No aplica.
Entendimiento	Parcialmente contemplado	Completamente contemplado	Personas que no forman parte de la Ingeniería de Software tienden a tener dificultades para entender la notación de SPEM.
Integración organizacional	No contemplado	Completamente contemplado	La especificidad de los elementos del proceso provista por SPEM dificulta o impide dicha integración.

Tabla 2.1. Comparación entre SPEM y BPMN

De la Tabla 2.1 podemos concluir que BPMN tiende a ser más fácil de entender y tiene como objetivo la integración de los procesos de la organización, pero no tiene tanta expresividad en la representación de procesos de software como SPEM.

2.2. Variabilidad

La *Variabilidad en Procesos de Software* [6] es la habilidad que tiene un sistema de software de ser cambiado, personalizado o configurado para ser usado en un contexto particular. Dicha variabilidad es capturada mediante la definición de características requeridas del software y las distintas variaciones que estas características podrían implicar.

En el proceso de adaptación de un proceso de software, se obtiene un conjunto de procesos muy similares al proceso original [14]. Todos estos procesos, que son prácticamente idénticos, conforman una *Familia de Procesos* de software. En una familia de procesos, los procesos pueden hacer un buen uso de sus similitudes y explotar sus diferencias, de una manera similar a la de Líneas de Producto Software.

En la última década han surgido diversas propuestas para tratar la variabilidad de procesos con foco en el modelado de familias de procesos con variantes (con base en lo que se denomina líneas de productos), evitando modelar cada variante en forma separada lo que implica duplicación y mantenimiento de las partes comunes [7][8]. Estas propuestas proponen definir una familia de procesos mediante un proceso base común (denominado proceso configurable) más las variantes que cada proceso específico requiere.

Dichas variantes pueden ser incluidas en el proceso base utilizando, principalmente, una de las dos formas de variabilidad siguientes:

- *Variabilidad por restricción:* se incluyen todas las posibles variantes en el modelo, para luego seleccionar una variante específica.
- *Variabilidad por extensión:* solo se incluyen en el modelo los elementos que son comunes a todos los procesos de la familia, agregando luego para cada variante sus elementos específicos.

De esta forma, cada variante es derivada desde el proceso configurable en la particularización, eliminando o agregando fragmentos del modelo, según parámetros de configuración.

2.2.1 Líneas de Productos de Software (SPLs)

Una *Línea de Productos de Software (SPL)* [15] es un conjunto de sistemas en software, que comparten un conjunto de características que satisfacen las necesidades específicas de un determinado segmento del mercado, y que se desarrolla a partir de un conjunto común de activos base. Una SPL se puede ver como una familia de sistemas que se dirige a un segmento de mercado específico. La mayoría de las empresas ofrecen productos para los clientes o mercados. A pesar de algunas diferencias, la mayoría de ellos tienden a tener algunas características comunes y formar una familia de productos de esa manera, con el fin de aumentar la calidad, reducir el tiempo de desarrollo y reducir los costos.

Tradicionalmente el desarrollo de una SPL está organizado en términos de dos procesos principales:

- *Ingeniería de dominio:* en la ingeniería de dominio los desarrolladores tratan de desarrollar y mantener activos reutilizables, definir el plan de alcance y la producción. En esta etapa el desarrollador se centra en la determinación del alcance, especificación y modelado de las características comunes y diferentes de una SPL, la descripción de una arquitectura flexible que incluye la SPL, características comunes y diferentes, la producción de un conjunto de activos fundamentales como componentes, marcos, bibliotecas, los aspectos que aborda la implementación de la arquitectura de la SPL.
- *Ingeniería de aplicaciones:* es la etapa en la que se recogen los requisitos del producto en particular, y el producto se construye mediante la organización de los activos reutilizables de acuerdo con el plan de producción. En ingeniería de aplicaciones, una configuración de modelo de características se utiliza para crear e integrar los principales activos producidos durante la etapa anterior (ingeniería de dominio), por lo que una instancia (producto) de la arquitectura de la SPL se puede generar.

2.2.2 Líneas de Procesos de Software (SPrLs)

Una *Línea de Procesos de Software (SPrL)* [4] es una familia de procesos que tiene un conjunto de características gestionadas que satisfacen las necesidades específicas de una organización y se desarrollan a partir de un conjunto de procesos básicos comunes. Por lo tanto, en base a los conceptos y técnicas del enfoque de la línea de productos de software, el área de investigación SPrL tiene como objetivo proporcionar las técnicas y mecanismos para:

i) Modelar los aspectos comunes y las variabilidades existentes en una familia de procesos de software.
ii) Apoyar a la personalización de los procesos de software de acuerdo a las necesidades específicas del dominio del proceso de software.

La variabilidad es el término general que se utiliza para referirse a los aspectos variables de los productos de una SPrL. Se describe a través de puntos de variación y variantes. Un punto de variación es el lugar específico en un artefacto de la línea de productos en la que está conectada una decisión de diseño. Cada punto de variación se asocia con un conjunto de variantes que corresponden a diseñar alternativas para resolver la variabilidad.

2.3. Modelado de la Variabilidad

En esta sección se describen los diferentes enfoques estudiados para el modelado de la variabilidad, mostrando para cada uno de ellos un ejemplo de aplicación, así como las ventajas y desventajas que presentan.

2.3.1 SPEM 2.0

SPEM (*Software Process Engineering Metamodel*) [3] en su versión 2.0 es un estándar desarrollado por la OMG (*Object Management Group*) para la representación de modelos de procesos de software, que se basa en otros modelos de OMG, tales como MOF (*Meta Object Facility*) y UML (*Unified Modelling Language*).

Si bien SPEM incluye notación gráfica propia, utiliza las relaciones de asociación de UML. Esta notación nos permite representar y visualizar diagramas con los procesos y métodos.

Metamodelo SPEM

SPEM 2.0 distingue dos grupos de conceptos principales para implementar una metodología [16]:

- Primero, se implementa el *Method Content* (contenido del método) con *Content Elements* (elementos de contenido), que son los constructores básicos.
- Luego, se combinan y reutilizan para obtener *Processes* (procesos).

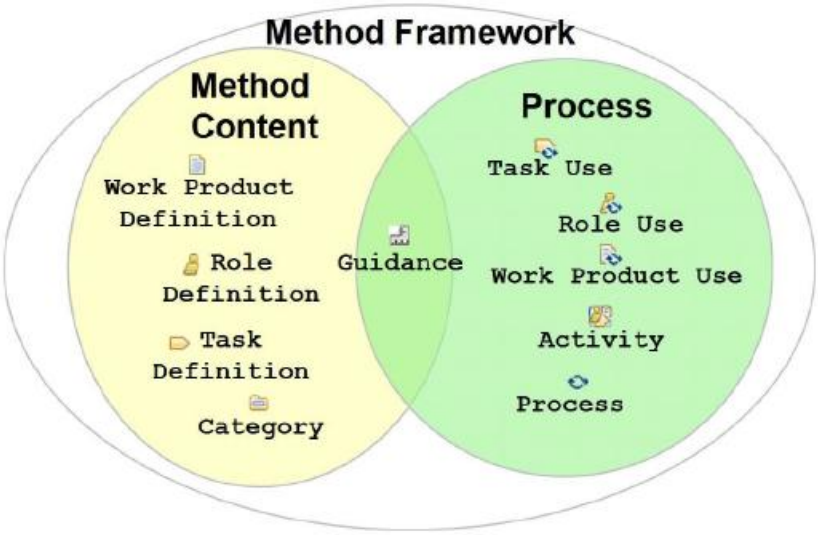


Figura 2.3. Aspectos principales para modelar con SPEM.

Existe una jerarquía de desglose del trabajo que es la siguiente, de lo más general a lo más particular:

- *Delivery Process* (proceso de despliegue): Representa procesos tan complejos como se quiera, que se utilizarán de base para realizar un determinado proyecto.
- *Capability Pattern* (patrón de capacidad): Representa un fragmento de un proceso, que puede ser reutilizado tantas veces como se quiera en un *delivery process*.
- *Activity* (actividad): Es el elemento central para definir procesos, dado que permite organizar los elementos básicos del proceso, tales como roles, productos de trabajo y tareas.
- *Task* (tarea): es la unidad mínima de trabajo que puede existir en un modelo de proceso en SPEM 2.

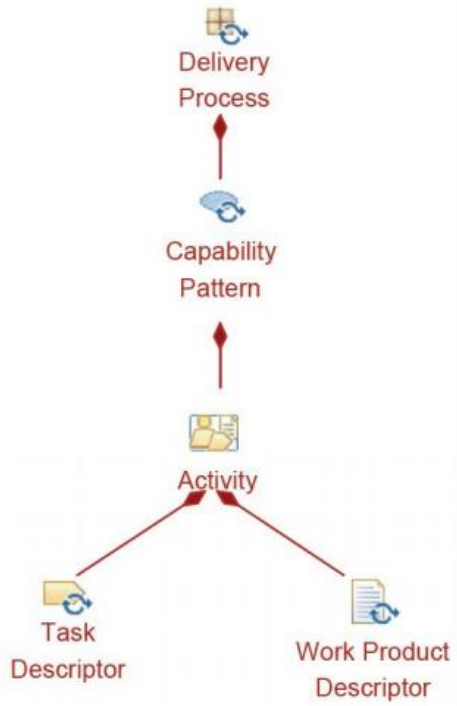


Figura 2.4. Jerarquía de desglose del trabajo

Variabilidad en SPEM 2.0

Los tipos de variabilidad entre dos elementos existentes en SPEM son los siguientes:

- *Not assigned:* No existe relación de variabilidad entre el elemento en cuestión y otros. Es el valor por defecto.
- *Contributes*: El elemento origen aporta sus propiedades al elemento destino, sin alterar directamente ninguna de las propiedades del elemento origen.
- *Replaces*: El elemento origen sustituye al elemento destino. En este caso, sólo las asociaciones entrantes del elemento destino se conservan; tanto los atributos como las asociaciones salientes del mismo serán sustituidos por los del elemento origen. Además, si un elemento es destino de múltiples relaciones *Replaces*, sólo puede ser sustituido por un elemento de origen en una configuración dada.
- *Extends*: Un elemento origen amplía la definición de su elemento destino, probablemente sobrescribiendo atributos y asociaciones del mismo. El elemento origen incluye las propiedades heredadas del elemento destino, más las propias.
- *Extends-Replaces*: Un elemento origen extiende primero su elemento destino, y luego lo reemplaza.

Las instancias de estas relaciones pueden sobrescribirse entre sí, por lo que deben ser resueltas en el orden presentado. A priori, es difícil predecir cómo interactúan entre ellas, por lo que los mecanismos de variabilidad de SPEM 2.0 rara vez se utilizan en la práctica [17]. Sin embargo, cuando se utiliza, la primitiva más usada generalmente es *Replaces*.

Ejemplo de aplicación

Se tomó como modelo base el modelo presentado en [4]. Se trata de un modelo de proceso de desarrollo de software, representado en SPEM 2.0, cuyo diagrama de actividad asociado se muestra en la Figura 2.5. El diagrama comienza con la actividad *Analizar la Arquitectura*, seguida de un nodo de decisión que permite elegir entre la actividad *Analizar Casos de Uso* o *Analizar Interfaz*. Por ejemplo, para un proceso específico, si se está interesado en las clases o los paquetes, pero no en la especificación de requerimientos, deberá seleccionarse la actividad *Analizar Casos de Uso*. Si por el contrario, para otro proceso específico, se está interesado en el análisis de interfaces y luego, en crear la especificación de componentes y arquitectura inicial, así como el descubrimiento de las operaciones de negocio de dichas interfaces, se deberá seleccionar la actividad *Analizar Interfaz*.

A lo largo de este capítulo le aplicaremos los distintos enfoques estudiados para representar variabilidad, teniendo en cuenta las siguientes consideraciones:

- La actividad *Analizar el paquete* puede ser incluida o no en el proceso adaptado, según sea el caso.
- Las actividades *Refinar componentes y arquitectura* y *Refinar interfaz y operaciones* pueden incluirse o no en el modelo final, siempre y cuando se respete la restricción de que están ambas o ninguna.
- El diseñador puede o no participar en la actividad *Refinar interfaz y operaciones*.

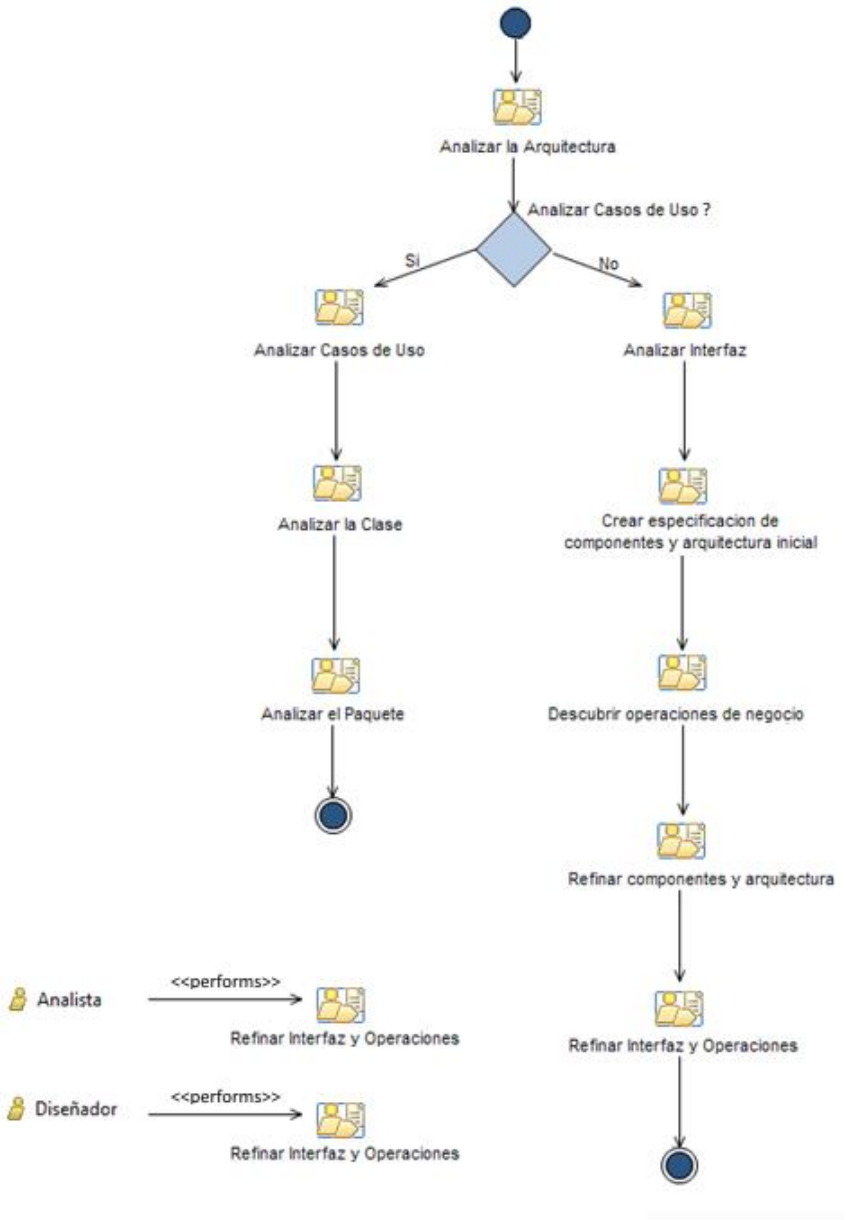


Figura 2.5. Modelo base modelado con SPEM 2.0

En base a esto, podemos adaptar el modelo utilizando los mecanismos de variabilidad provistos por SPEM 2.0 obteniendo el modelo mostrado en la Figura 2.6:

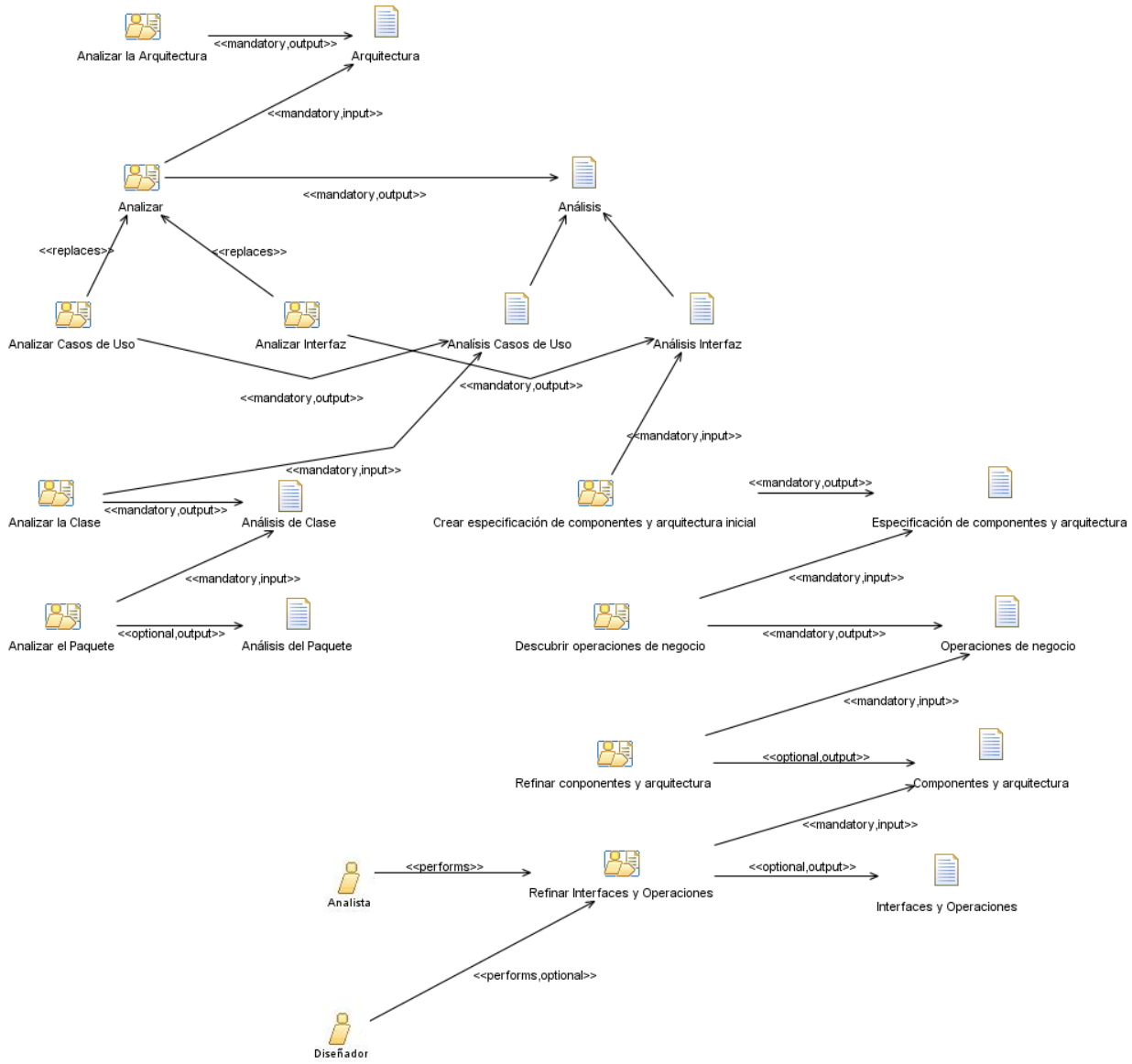


Figura 2.6. Modelo con variabilidad modelado con SPEM 2.0

Como podemos notar, la variabilidad pudo ser representada utilizando los mecanismos brindados por SPEM. Sin embargo, no podemos establecer cuál es la parte de la línea de proceso que es común a todos los procesos. Esto significa que no podemos definir el *core process* de la línea de proceso. Por otra parte, y a consecuencia de esto, no es posible incluir los objetivos del proceso en la línea de proceso y asegurar que no se hayan modificado al aplicar la variación. Para evitar esto, las acciones de los mecanismos de variabilidad deberían ser más limitadas. Es decir, en una línea de proceso debería ser posible especificar qué partes pueden variar y en qué rango de

valores. Finalmente, podemos ver que la notación utilizada para representar la variabilidad son asociaciones UML y relaciones de herencia, que muchas veces hacen que sea difícil interpretar algunas de estas relaciones.

Ventajas

- Tiene una herramienta de soporte (*EPF Composer*) que está a disposición y es robusta.
- Puede publicar procesos en *HTML*, así como exportarlos a *XML* para su posterior procesamiento.

Desventajas

- En *EPF Composer*, la opcionalidad se especifica mediante el establecimiento de una propiedad del elemento, pero las alternativas no se pueden definir directamente: el usuario debe crear manualmente dos o más modelos de proceso, uno para cada alternativa.

Limitaciones

- Este mecanismo de variabilidad no permite la delimitación de la parte central de la *Software Process Line*. Es decir, los mecanismos de variabilidad de SPEM no permiten establecer que una parte de la línea de proceso será común a todos los procesos y limitar estas variaciones.
- Los mecanismos de variabilidad de SPEM 2.0 no garantizan que las variaciones introducidas para generar un nuevo proceso no alterarán sus objetivos. Los objetivos deben ser comunes a todos los ajustes que se realizan en el proceso de utilización de los mecanismos de variabilidad.
- Puede ser difícil reutilizar modelos de SPEM 2.0 con los mecanismos de variabilidad definidos. Esto tiene la desventaja de que es necesario definir un nuevo modelo con el fin de extender el de la base que contiene la especificación del proceso y adaptarlo en el contexto en el cual se aprobó.
- SPEM 2.0 no incluye una notación específica para representar la variabilidad del proceso. La variabilidad se muestra mediante el uso de asociaciones UML y relaciones de herencia.

2.3.2 Modelo de Características

Los modelos de características (FM) [18] se utilizan con frecuencia para modelar la variabilidad en SPLs. Para especificar la variabilidad en SPrL, cada característica de un FM representa un elemento del proceso. Los elementos del proceso obligatorios/opcionales están marcados en el modelo como *mandatory/optional*, y los elementos del proceso que se pueden realizar de diferentes maneras se modelan utilizando alternativas. Las restricciones del tipo *requires* se incluyen para asegurar que los vínculos existentes entre los roles y tareas, y las tareas y productos de trabajo se conserven.

Además, cada enlace obligatorio entre una tarea y un producto de trabajo introduce una nueva restricción del tipo *requires*. La dirección de esta restricción depende de si el producto de trabajo es un artefacto de entrada o de salida. Por otra parte, los enlaces opcionales entre tareas y productos de trabajo no introducen nuevas restricciones.

Ejemplo de aplicación

En la Figura 2.7 se muestra el modelo de características para el modelo que estamos utilizando de ejemplo:

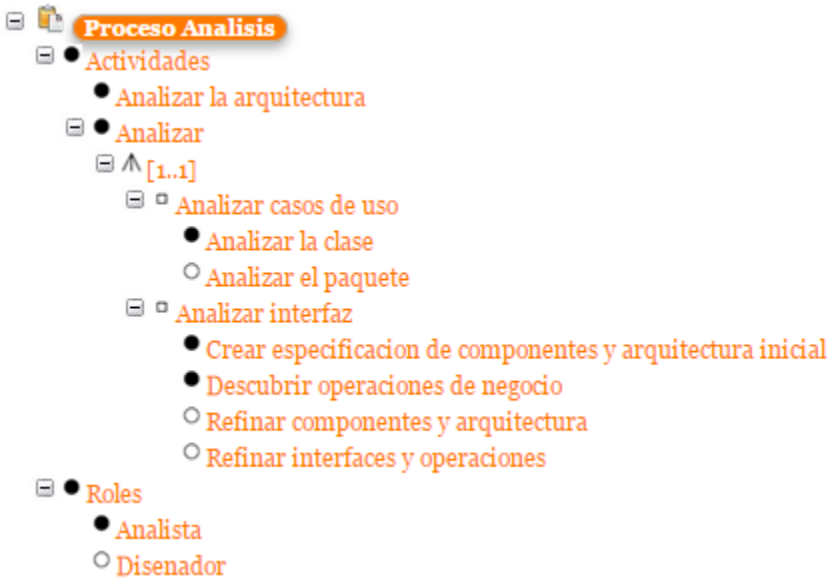


Figura 2.7. Modelo de Características

En él se puede observar la opcionalidad de las actividades *Analizar el paquete*, *Refinar componentes y arquitectura*, y *Refinar interfaces y operaciones*, así como también la del rol *Diseñador*. Faltaría agregar las restricciones necesarias para mantener la consistencia del modelo inicial, para lo cual debemos asegurar que:

- Si se incluye la actividad *Refinar componentes y arquitectura*, debe incluirse también *Refinar interfaces y operaciones*, y a la inversa, si se incluye la actividad *Refinar interfaces y operaciones*, debe incluirse también *Refinar componentes y arquitectura*.

$$(\neg \text{Refinar componentes y arquitectura} \vee \text{Refinar interfaces y operaciones})$$
$$(\neg \text{Refinar interfaces y operaciones} \vee \text{Refinar componentes y arquitectura})$$

- Si se incluye el rol de *Diseñador*, entonces, deben incluirse todas las actividades de las cuales él es responsable (en este caso solo es responsable de la actividad *Refinar interfaces y operaciones*)

$$(\neg \text{Disenador} \vee \text{Refinar interfaces y operaciones})$$

Ventajas

- Hay varias herramientas gratuitas para especificar FM, por ejemplo, *splot*, *fmp6* y *clafer7*, todas ellas capaces de exportar datos a formatos estándares *como XML* o *Alloy*.
- Permiten la especificación de FMs jerárquicos de una manera intuitiva.

Desventajas

- Las características y enlaces de diferentes tipos no pueden distinguirse; por ejemplo, los roles, tareas y productos de trabajo son todas características.
- Por otra parte, el flujo de control presente en un modelo SPEM se pierde en el FM. Por ejemplo, no se puede indicar que una tarea se debe realizar antes que otra.

2.3.3 Orthogonal Variability Model (OVM)

Los Modelos de Variabilidad Ortogonales (OVM) [19] son otra notación utilizada para modelar la variabilidad.

Es necesaria la creación de dos modelos: un modelo SPEM 2.0, donde se identifican los puntos de variación, y un modelo OVM, donde cada punto de variación está

asociado a variantes que representan diferentes realizaciones posibles de cada punto de variación. Como en el caso del FM, las variantes pueden ser opcionales u obligatorias, y pueden agregarse restricciones de tipo *requiere* y *excludes* entre los elementos.

Todas las variantes se relacionan con sus puntos de variación a través de dependencias opcionales, y en el caso de los puntos de variación obligatorios, debe elegirse exactamente una de las variantes posibles. Las restricciones *requiere* y *excludes* se utilizan para hacer cumplir las dependencias existentes entre las variantes.

Ejemplo de aplicación

Para la aplicación de este enfoque necesitamos el modelo SPEM 2.0 de la Figura 2.5, y el modelo OVM de la Figura 2.8:

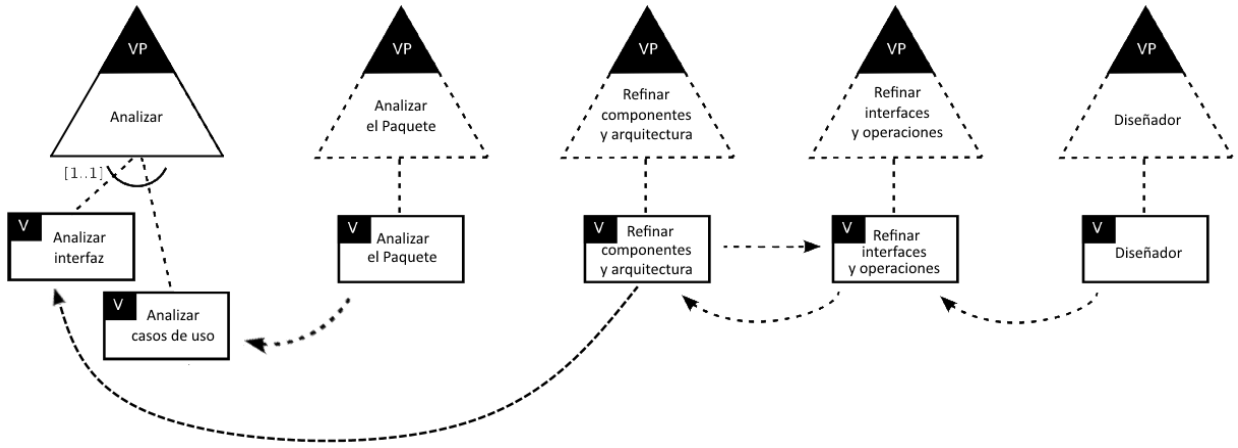


Figura 2.8. Modelo OVM

Como se puede observar, el punto de variación *Analizar* está marcado con líneas continuas, indicando la obligatoriedad del mismo, mientras que los otros están con líneas punteadas, estableciendo que son opcionales. La variante *Analizar el Paquete* no puede ser incluida si no se escogió previamente la variante *Analizar casos de uso* en el punto de variación *Analizar*, para lo cual agregamos la restricción mediante la flecha punteada. De la misma forma, *Refinar componentes y arquitectura* no puede ser incluida si no se eligió *Analizar Interfaz* previamente. Finalmente, las otras restricciones incluidas indican que las acciones *Refinar componentes y arquitectura* y *Refinar interfaces y operaciones* deben ser incluidas ambas o ninguna, y que si el rol de *Diseñador* está también debe estar *Refinar interfaces y operaciones*.

Ventajas

- Proporciona una notación concisa para especificar todas y solamente las variabilidades de un modelo determinado.
- La notación es clara y el diagrama es ordenado, ya que los elementos comunes no se incluyen.

Desventajas

- No hay herramientas maduras que soporten OVM.

2.3.4 SmartySPEM

En el enfoque *SMartySPEM* [4] se extiende el perfil de SPEM para la representación de las variabilidades en SPrLs teniendo en cuenta el enfoque de *Smarty* para la gestión de la variabilidad. SMartySPEM se compone de un perfil UML para la representación de las variabilidades y de directrices que sugieren cómo identificar las mismas.

El enfoque tiene como objetivo el apoyo a la identificación y representación de variabilidad en los elementos del proceso de software basado en SPEM. Soporta la derivación de procesos específicos mediante la resolución de variabilidades. Actualmente, no soporta la derivación automática de procesos específicos.

El enfoque *SMartySPEM* ofrece el mismo conjunto de íconos del metamodelo de SPEM. Sin embargo, cada ícono de SPEM tiene varios íconos variantes etiquetados con una de las siguientes siglas: MDT para *mandatory*, OPT para *optional*, OR para variantes inclusivas, XOR para variantes exclusivas, y VP para puntos de variación. Esto significa que cada ícono de SPEM recibe una de estas siglas para mejorar la legibilidad de un modelo basado en *SMartySPEM*.

Ejemplo de Aplicación

El modelo de la Figura 2.5 adaptado utilizando el enfoque *SmartySPEM* es mostrado a continuación en la Figura 2.9:

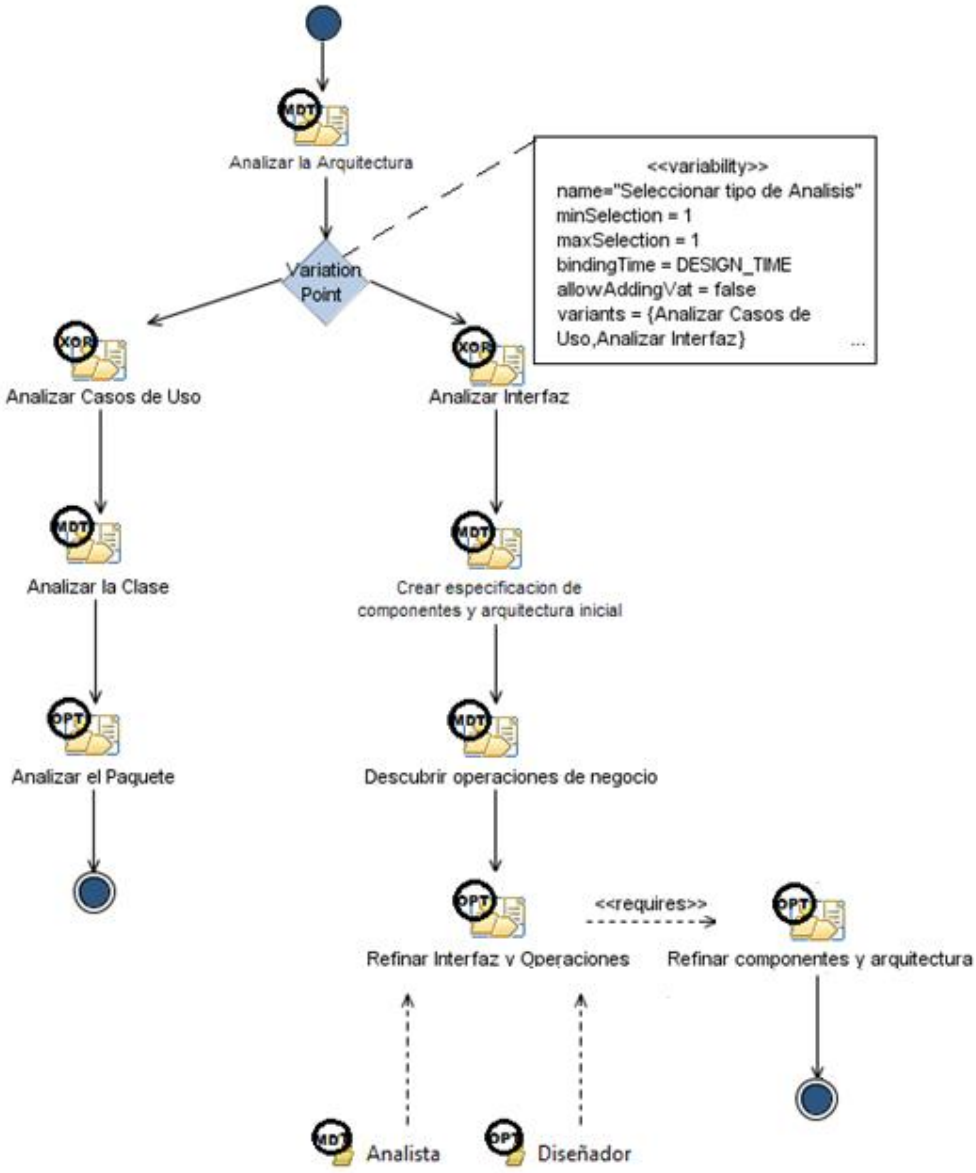


Figura 2.9. Modelo con SmartySPEM

La primera actividad es *Analizar la Arquitectura*, la cual es obligatoria y se etiqueta con el valor MDT. Esto significa que para cada proceso derivado, tal actividad debe realizarse. El primer nodo de decisión se etiqueta como un punto de variación, y tiene asociado una variabilidad denominada *Seleccionar tipo de Análisis*, con todos los detalles como *name*, *min* y *max* a seleccionar, así como *binding time*. Hay dos elementos *Activity* que son variantes exclusivas (etiquetadas con XOR) y representan las alternativas del punto de variación. Por ejemplo, para un proceso específico, si uno

está interesado en las clases y/o paquetes, pero no está interesado en las especificaciones de los componentes, *Analizar Casos de Uso* debe ser seleccionada.

La actividad *Analizar el Paquete* es opcional, lo que implica que, para otro proceso específico, luego de la actividad *Analizar la Clase* se pueda finalizar el proceso sin realizar dicha actividad.

La actividad *Refinar Interfaz y Operaciones* establece una relación <<*requires*>> con la actividad *Refinar componentes y arquitectura*, lo que significa que la selección de la primera variante implica la selección de la otra variante para un proceso específico.

Finalmente, la notación OPT sobre el rol del diseñador implica que este pueda o no participar de la actividad *Refinar Interfaz y Operaciones.*

Ventajas:

- *SMartySPEM* añade su mecanismo de variabilidad directamente a SPEM mediante la ampliación de algunos de sus estereotipos basados en el enfoque de *Smarty*. Por lo tanto, *SMartySPEM* no modifica la estructura estándar de SPEM 2.0. Por el contrario, introduce nuevos estereotipos a los elementos originales.
- Proporciona un conjunto de directrices para apoyar la aplicación de *SMartySPEM* para la identificación y representación de la variabilidad en los elementos del proceso de software basado en SPEM.

Desventajas:

- No soporta la derivación automática de procesos específicos.
- No cuenta con una herramienta de soporte.

2.3.5 BPMNt

BPMNt (BPMN + *tailoring*) [5], es una extensión conservativa de BPMN que propone un mecanismo similar al provisto por SPEM 2.0 para representar la adaptación de procesos. Por conservativa se entiende que es una extensión que no modifica la semántica original de la especificación de BPMN.

La extensión propuesta introduce elementos en BPMN que capturan la sintaxis y la semántica necesaria para soportar la habilidad de remover, reemplazar y agregar elementos de proceso. Estos elementos son relaciones de adaptación, que reciben el nombre de: *suppression*, *local replacement* y *local contribution*, respectivamente.

La Figura 2.10 muestra la correspondencia entre elementos de SPEM 2.0 y BPMN 2.0:

Meta-Clase SPEM 2.0	Equivalente Meta-Clase(s) BPMN 2.0
Activity	Process
	SubProcess
	Task
RoleUse	Lane
	HumanPerformer
WorkProductUse	DataObject
	DataInput
	DataOutput

Figura 2.10. Correspondencia entre SPEM 2.0 y BPMN 2.0

Debido a que el proceso de adaptación involucra la modificación de un proceso general para derivar uno más específico, el enfoque requiere de un modelo de proceso base, que es un modelo BPMN que contiene la especificación del proceso a ser adaptado. En la fase de especificar la adaptación, el modelo de proceso adaptado debe ser creado para representar el proceso derivado del más general. Este modelo debe contener sólo las diferencias en relación al modelo de proceso base, que son especificadas usando las relaciones de adaptación de BPMNt. Además, el modelo de proceso adaptado consiste en un proceso incompleto BPMN, más los elementos de extensión BPMNt que crean la dependencia sobre el modelo de proceso base. Finalmente, en la fase de interpretación de la adaptación, un conjunto de reglas de acción son aplicadas para interpretar las relaciones de adaptación del proceso y generar el modelo final. Este último contiene un modelo BPMN completo que es el conjunto total de elementos BPMN que componen el proceso derivado final. Así, el modelo final puede ser cargado en cualquier herramienta BPMN.

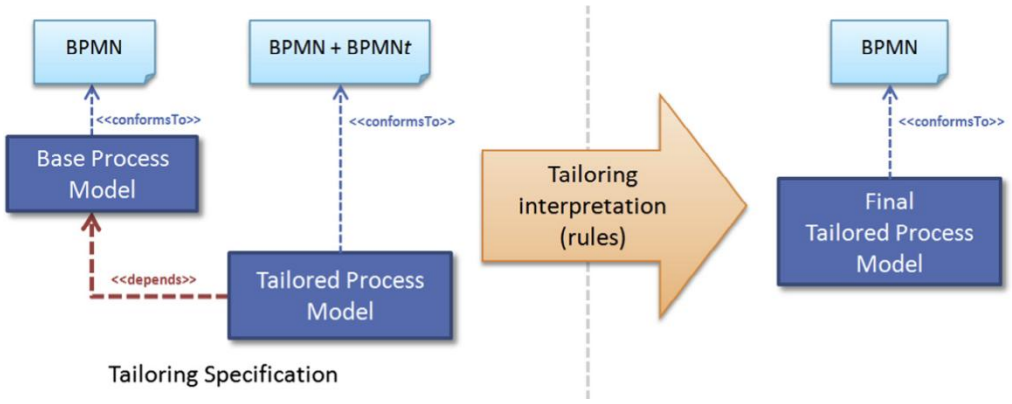


Figura 2.11. Enfoque de adaptación BPMNt

Ejemplo de aplicación

Dado que nuestro ejemplo está representado con SPEM, el primer paso es traducirlo a BPMN (Figura 2.12). Para hacer esta traducción interpretamos las actividades de SPEM como tareas (*task*) y los roles como *lanes*. Una inconveniente al cual nos enfrentamos fue que en BPMN una tarea no puede ser compartida entre *lanes*, por lo cual no pudimos representar el hecho de que la tarea *Refinar interfaz y operaciones* pueda ser realizada por el rol de *Analista* y de *Diseñador*. Dado esto, decidimos modificar el modelo de forma que sea realizada por el *Diseñador*.

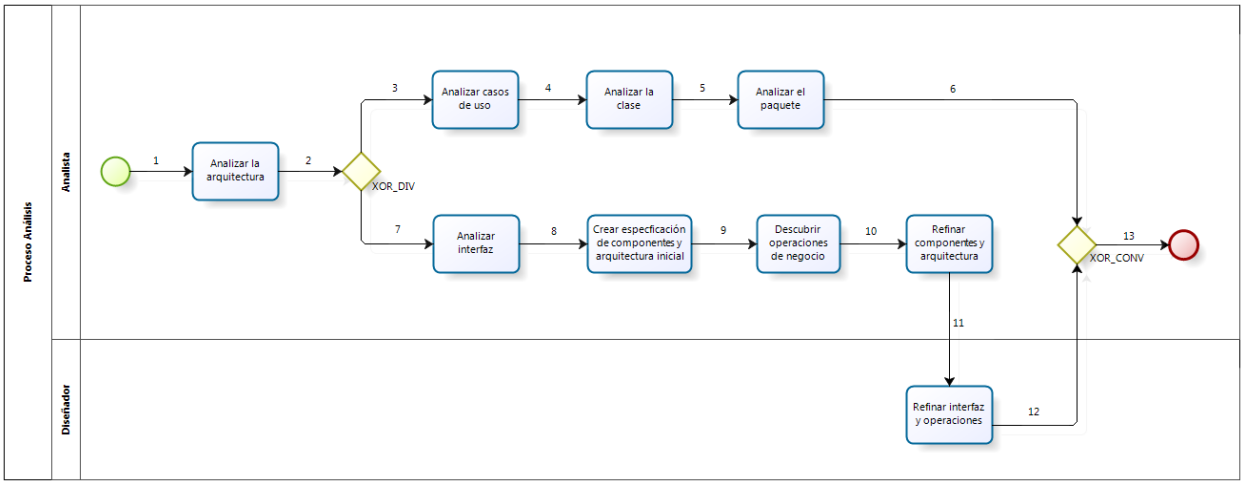


Figura 2.12. Modelo base modelado con BPMN 2.0

A continuación, con el fin de obtener una adaptación de este modelo, especificamos los cambios a realizar sobre el mismo:

- Luego de realizar la tarea *Analizar la arquitectura*, el flujo continúa por *Analizar interfaz*, descartando el otro flujo alternativo.
- Se agrega una nueva tarea con el nombre de *Realizar prototipo de la interfaz*, que será ejecutada luego de la tarea *Refinar componentes y arquitectura*.

En la Figura 2.13 se muestra cómo se relaciona el proceso derivado con el proceso base a través de las relaciones de adaptación BPMNt para representar las modificaciones recién mencionadas:

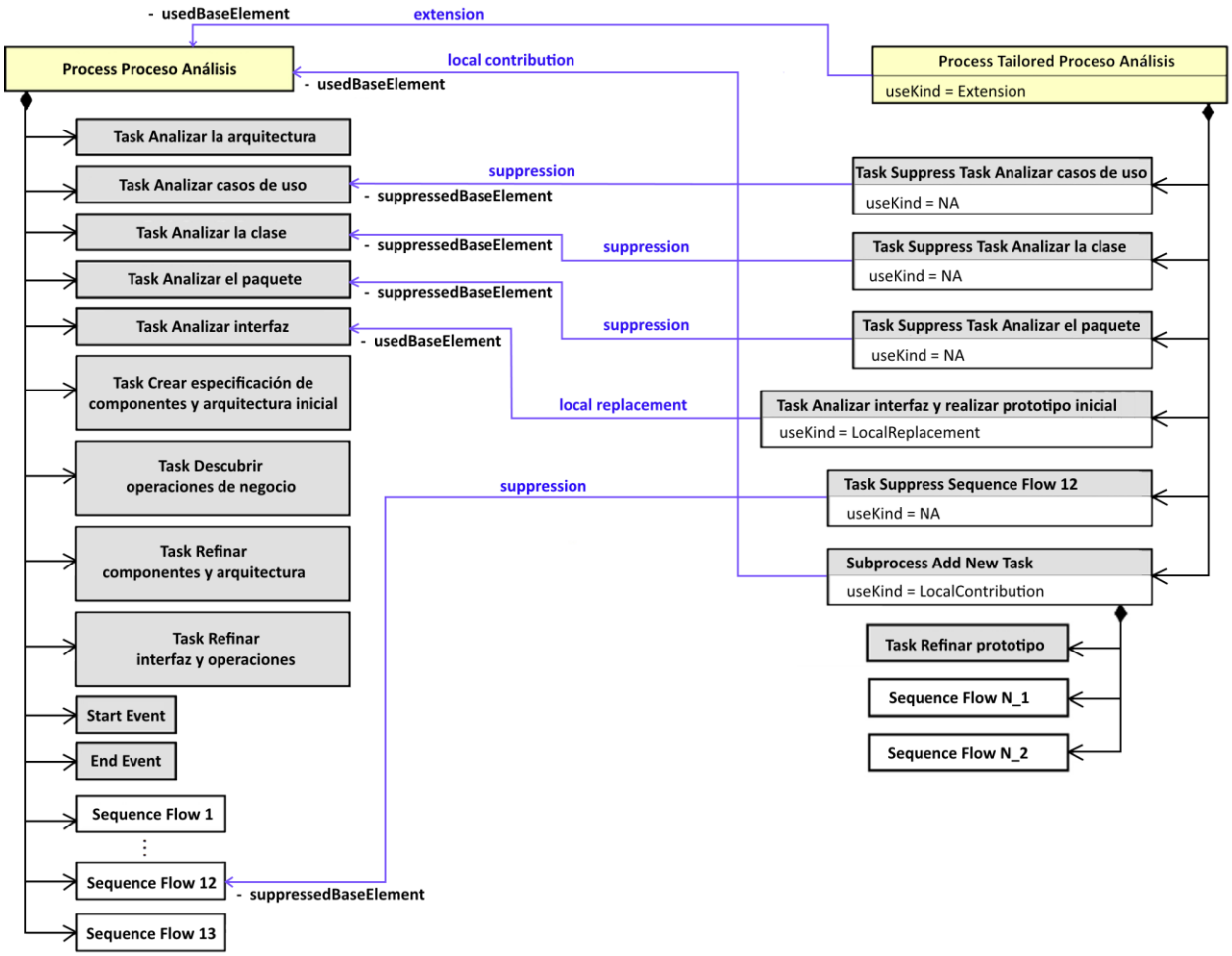


Figura 2.13. Especificación de la adaptación de BPMNt

El proceso adaptado es llamado *Tailored Proceso Análisis*. Tal proceso se relaciona con el original a través de una relación de extensión (*extension*), lo que significa que el proceso adaptado reutiliza todos los elementos del proceso especificados en el proceso original *Proceso Análisis*. Además, se agregaron otros tipos de relaciones:

- *Suppression*: La relación entre, por ejemplo, la tarea *Suppress Task Analizar casos de uso* del proceso adaptado y la tarea *Analizar casos de uso* del proceso base usa el atributo *SuppressedBaseElement* para representar la exclusión de dicha tarea.
- *Replacement*: La relación del tipo *LocalReplacement* entre la tarea *Analizar interfaz y realizar prototipo* y la tarea *Analizar interfaz* para representar el reemplazo de una por otra.

- *Suppression*: La relación desde la tarea *Suppress Sequence Flow 12* del proceso adaptado que apunta a *Sequence Flow 12* del proceso base usando el atributo *SuppressedBaseElement* para representar la exclusión de este flujo de secuencia.
- *Contribution*: La relación del tipo *LocalContribution* entre el subproceso *Add New Task* del proceso adaptado (que contiene la nueva tarea *Refinar prototipo* así como los dos nuevos flujos de secuencia que la conectan con las otras actividades) y el proceso base en sí para representar la agregación de un elemento.

Finalmente, el modelo adaptado resulta como se muestra en la Figura 2.14.

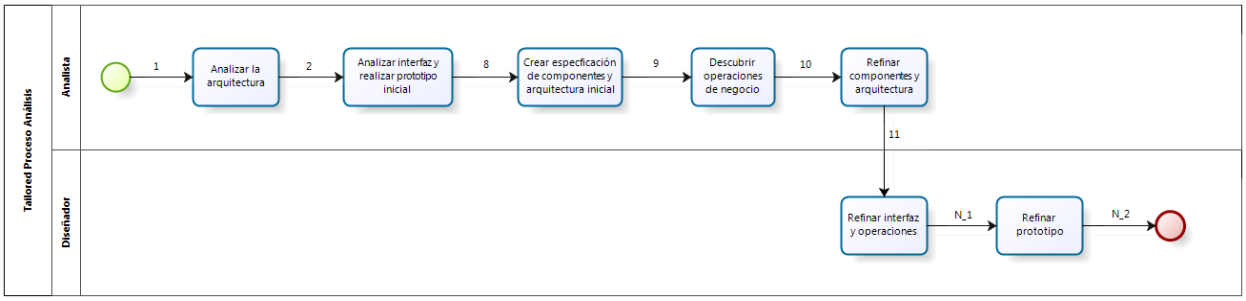


Figura 2.14. Modelo adaptado BPMN 2.0

Ventajas

- El modelo final está en BPMN, por lo que puede ser ejecutado, simulado, monitoreado y analizado, ya que existen herramientas para ello.
- Aunque solo se ha probado la capacidad de adaptación en procesos de software, también se puede aplicar para adaptar procesos de negocio, si bien habría que evaluar sus beneficios en ese caso.

Desventajas

- Las reglas de consistencia definidas no cubren operaciones de adaptación complejas como las *contributions*, que son difíciles de validar.
- El foco de la propuesta está en la representación de la adaptación del modelo, y no en su automatización.

2.3.6 CASPER

En [20], [21] y [22] se propone un enfoque para la adaptación automática de procesos de software, basado en MDE (*Model-Driven Engineering*).

Este mecanismo de adaptación toma dos modelos de entrada: el modelo del proceso de software de la organización (con las variabilidades) y un modelo de contexto de un proyecto específico. Luego, mediante un proceso de transformación, obtiene como salida el modelo de proceso adaptado a las características de dicho contexto.

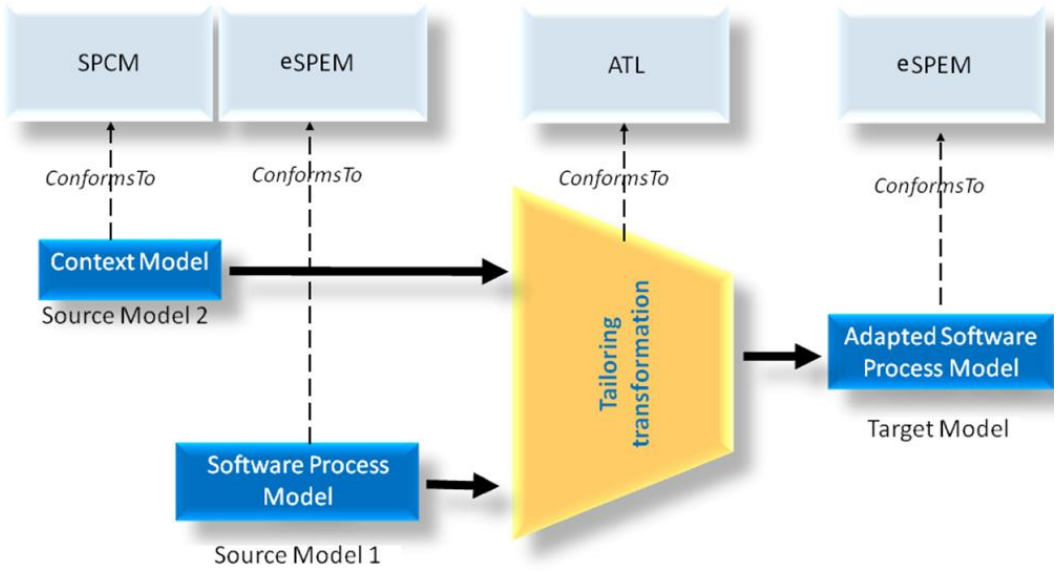


Figura 2.15. Enfoque de adaptación CASPER

El modelo de proceso de la organización es modelado con eSPEM, un metamodelo que incluye los conceptos básicos de SPEM 2.0 y las relaciones requeridas por el mecanismo de adaptación propuesto.

El contexto del proyecto puede variar según diversas variables, tales como: el tamaño del proyecto, la duración, la complejidad, el tamaño del equipo de desarrollo, etc. Para especificar el modelo de contexto, en esta propuesta se define SPCM (*Software Process Context Metamodel*), que se base en tres conceptos:

- ***ContextAttribute*** que representa una característica relevante del contexto del proceso, necesario para la adaptación, y que puede tomar uno de los valores definidos como *ContextAttributeValue*. Un ejemplo de *ContextAttribute* puede ser el tamaño del proyecto, y sus *ContextAttributeValue* posibles pueden ser {Pequeño, Mediano, Grande}.

- ***Dimension*** que representa una colección de *ContextAttributes* relacionados. Un ejemplo de *Dimension* es un Equipo, que puede ser usado para agrupar atributos del equipo como tamaño y capacidades del equipo.
- ***ContextAttributeConfiguration*** que refiere a una configuración específica que asocia un *ContextAttribute* a un único *ContextAttributeValue*. Un conjunto de *ContextAttributeConfiguration* forma el *ContextConfiguration.*

El proceso de transformación es llevado a cabo mediante la aplicación de reglas de transformación, definidas mediante ATL (*Atlas Transformation Language*), pudiendo combinarlas entre sí con el fin de configurar nuevos modelos de proceso.

Para cada elemento incluido en el proceso de la organización (proceso general), y teniendo en cuenta el modelo de contexto definido, se aplica la regla de transformación correspondiente y en caso de evaluar verdadero, el elemento se incluye en el modelo final (modelo adaptado); en caso contrario, el elemento no es incluído.

Se desarrolló *CASPER Tool3*, una herramienta implementada en Eclipse Modeling Framework (EMF 3.4) y con el plug-in ATL 2.0.

Ejemplo de aplicación

Para la adaptación con CASPER, comenzamos mostrando el modelo de características (Figura 2.16) relacionado al proceso que queremos adaptar, con el fin de establecer la variabilidad del mismo:

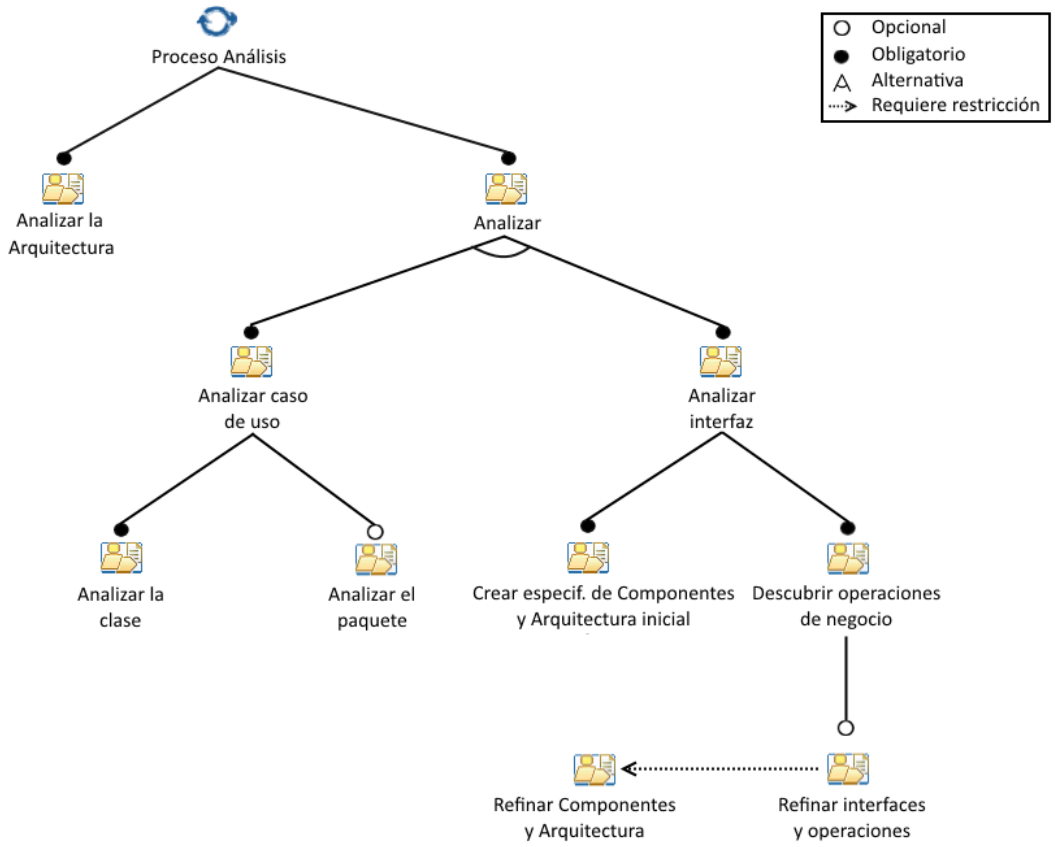


Figura 2.16. Modelo de características

Como podemos ver, luego de realizar la tarea *Analizar la Arquitectura* se debe elegir entre dos caminos posibles, ya que la tarea *Analizar* presenta dos alternativas: seguir por *Analizar caso de uso* o por *Analizar Interfaz*. Dado que ambas actividades están marcadas como obligatorias debemos sí o sí elegir uno de estos dos caminos.

Otro punto a destacar es la opcionalidad de las actividades *Analizar el paquete y Refinar Interfaces y operaciones*, que pueden ser incluidas o no en el modelo final según las circunstancias.

Finalmente cabe destacar también la flecha punteada que se muestra en el modelo, lo cual indica que si se realiza la actividad *Refinar Interfaces y operaciones* deberá realizarse primero la actividad *Refinar Componentes y Arquitectura*.

Para continuar con el proceso de adaptación, el siguiente paso sería definir el modelo de contexto correspondiente (Figura 2.17), para lo cual debemos identificar las

dimensiones, atributos y valores de atributos que consideremos pertinentes para la adaptación de este modelo.

Con el fin de ejemplificar definimos simplemente dos dimensiones: *Proyecto* y *Equipo*. Además definimos los atributos tamaño y duración del proyecto (para la dimensión *Proyecto*), así como el tamaño del equipo (para la dimensión *Equipo*), con sus respectivos valores posibles.

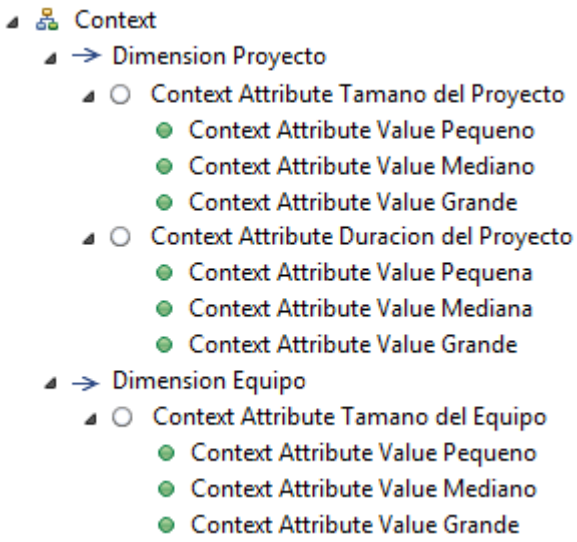


Figura 2.17. Modelo de contexto

Luego, para que la adaptación se realice de forma automática, tal como lo establece el enfoque propuesto, es necesario definir las reglas de adaptación que permiten relacionar los elementos de la configuración del modelo de contexto dada con las variantes establecidas en el proceso. En la Tabla 2.2 mostramos de forma resumida la forma en que podría realizarse la adaptación para el ejemplo propuesto, la cual debería ser traducida a reglas ATL para poder aplicar el enfoque. En las filas presentamos los atributos definidos con sus respectivos valores posibles, y en las columnas las actividades que son relevantes para la adaptación.

		Analizar	**Analizar el Paquete**	**Refinar interfaces y operaciones**
Tamaño del Proyecto	**Pequeño**	Analizar caso de uso		
	Mediano	Analizar caso de uso		
	Grande	Analizar interfaz		

Duración del Proyecto	**Pequeña**			Falso
	Mediana			Verdadero
	Grande			Verdadero
Tamaño del Equipo	**Pequeño**		Falso	
	Mediano		Falso	
	Grande		Verdadero	

Tabla 2.2. Reglas de adaptación

Para la actividad *Analizar*, si se trata de un proyecto de tamaño pequeño o mediano, se debe continuar por el camino *Analizar caso de uso*; de lo contrario se debe seguir por *Analizar interfaz*. Por otra parte, la actividad *Analizar el paquete* debe incluirse solo cuando se trate de un proyecto realizado por un equipo de tamaño grande; en cualquier otro caso esta actividad no será incluida. Finalmente, la actividad *Refinar interfaces y operaciones* será incluida siempre que se trate de un proyecto de tamaño mediano o grande.

Una vez definidas estas reglas con ATL (esto no es mostrado aquí), el siguiente paso es dar una configuración específica del modelo de contexto sobre la cual aplicar dichas reglas. Consideremos por ejemplo, un proyecto de tamaño pequeño que es llevado a cabo por un equipo también pequeño (para este caso en particular la duración del proyecto es indistinta). Entonces, el modelo adaptado sería el que se muestra en la Figura 2.18.

Si por el contrario, estuviésemos frente a un proyecto de gran tamaño y gran duración (en este caso el tamaño del equipo sería indistinto), el proceso obtenido sería el de la Figura 2.19.

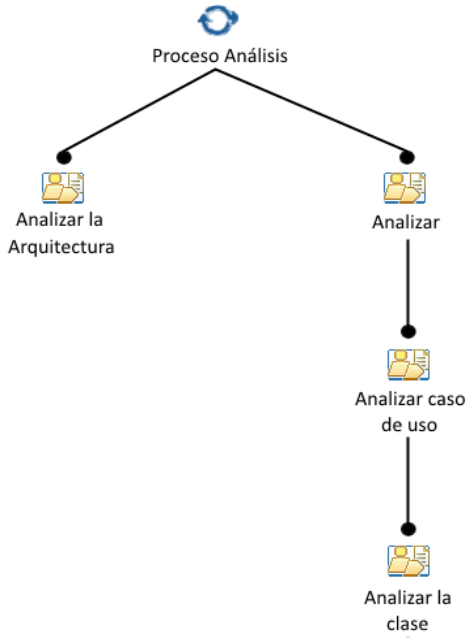


Figura 2.18. Modelo adaptado para un proyecto y equipo pequeño

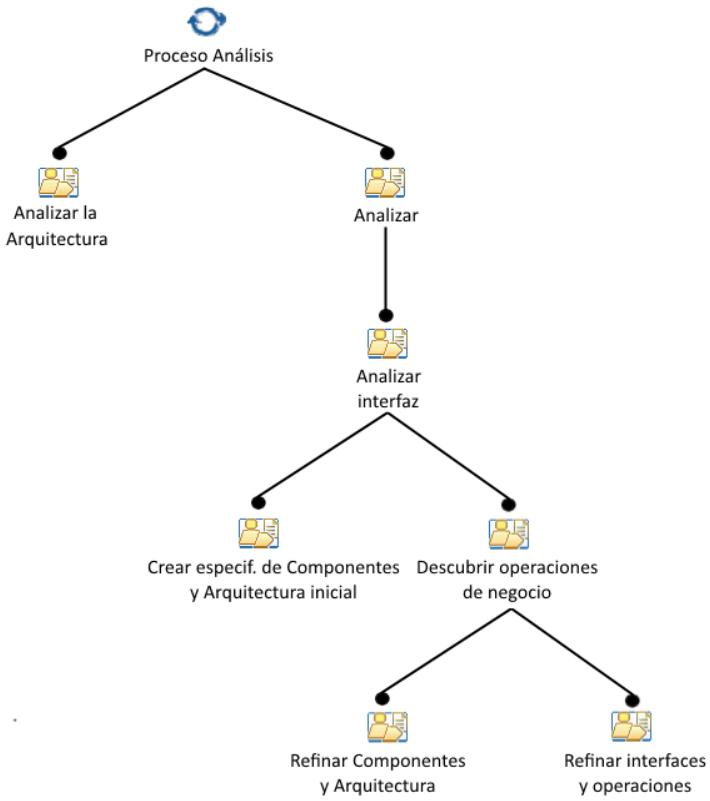


Figura 2.19. Modelo adaptado para un proyecto de gran tamaño y duración

Ventajas:

- El modelo adaptado se genera automáticamente, haciendo que el tiempo total para adaptar el proceso sea relativamente corto. Esto hace posible que al cambiar una característica del proyecto se obtenga de forma casi inmediata un nuevo modelo adaptado a este nuevo contexto.
- El proceso adaptado solo incluye los elementos que son estrictamente necesarios según la configuración del contexto dado, lo cual no podría garantizarse si la transformación se hiciera “a mano”.

Desventajas:

- Formalizar las reglas de transformación puede llegar a ser una tarea difícil para algunos proyectos.
- Este enfoque ha sido probado únicamente con empresas pequeñas, donde el número de variables de contexto y puntos de variación es pequeño.

2.3.7 Common Variability Language (CVL)

CVL [23] es un lenguaje genérico que permite la definición de un SPL y la derivación automática de un proceso de éste, de acuerdo con los requisitos de un proyecto determinado. Es un lenguaje independiente del dominio para especificar y resolver la variabilidad.

En este enfoque se habla de tres modelos:

- *Modelo base:* un modelo del proceso base.
- *Modelo de variabilidad (VAM - Variability Abstraction Model):* el modelo que define variabilidad sobre el modelo base.
- *Modelo de resolución (VRM - Variability Realization Model):* el modelo que define cómo se resuelve el modelo de variabilidad para crear un nuevo modelo.

El VRM permite la asignación entre el VAM (que se puede comparar con un modelo de características) y los activos (el modelo base). En este mapeo, el VRM define qué elementos del modelo base se ven afectados con un punto de variación específico y de qué manera. Este VRM ofrece varios tipos de puntos de variación que capturan la semántica de derivación.

Un modelo de base puede tener varios modelos de variabilidad, y cada modelo de variabilidad puede tener varios modelos de resolución. Con el modelo de la variabilidad

y el modelo de resolución definidos correctamente, es posible ejecutar CVL para generar nuevos modelos base resueltos, sin variabilidad.

El lenguaje CVL está desarrollado bajo Eclipse, y la herramienta de soporte está disponible como plug-ins de Eclipse.

Ejemplo de aplicación

En primer lugar se modela el proceso más usado, y luego, todo otro elemento del proceso que no conforma lo más usado (elementos externos del proceso). Se toma un extracto del modelo inicial para ejemplificar. El resultado se muestra en la Figura 2.20. Los elementos externos se representan en el modelo sin ningún enlace al proceso principal.

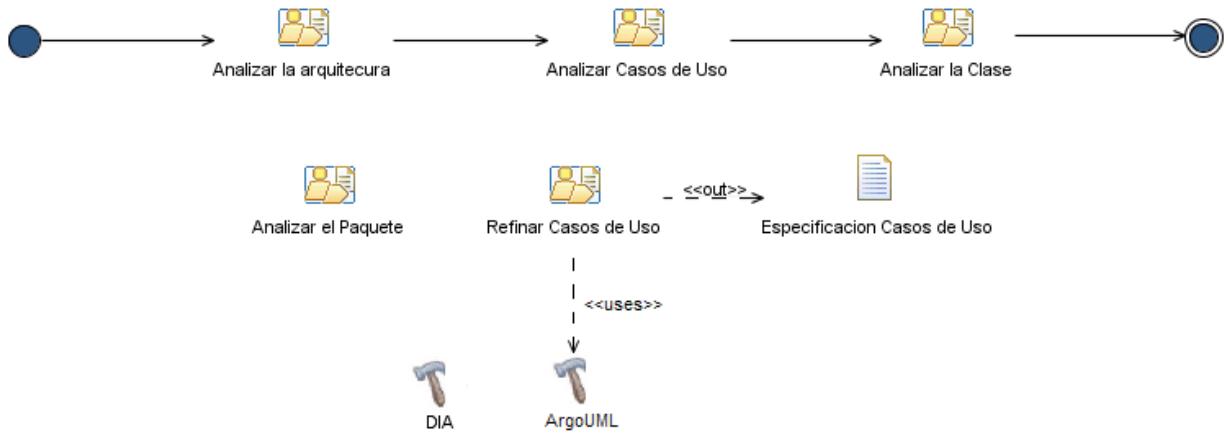


Figura 2.20. Fragmento del modelo base

A continuación, en la Figura 2.21 se presentan tres modelos:

- A la izquierda se muestra un extracto del modelo base donde aparecen los elementos variables del mismo, que en este caso son las actividades *Analizar el Paquete* y *Refinar Casos de Uso,* que se pueden realizar o no, y a su vez, la actividad *Refinar Casos de Uso* que puede utilizar la herramienta *ArgoUML* o *DIA.*
- Al centro, en el VRM se presentan las relaciones de unión entre los requerimientos variables del proyecto y el modelo base. Por ejemplo, para un

proyecto en el que se deben refinar los casos de uso, la actividad *Refinar Casos de Uso* será agregada por el VRM al proceso, así como también serán agregados su control de flujo saliente y su producto de trabajo (*Especificación Casos de Uso*). Para esto se redirige el flujo de control de salida de la actividad *Analizar Casos de Uso* hacia *Refinar Casos de Uso,* al igual que el flujo de control de salida de *Refinar Casos de Uso* se redirige hacia *Analizar la Clase*; esto se realiza utilizando la relación *Link Assignment.* Se utiliza la relación *Object substitution* para establecer que la herramienta *ArgoUML* puede ser reemplazada por la herramienta *DIA.*

- En el modelo de la derecha se presenta el RM donde se indica con ✓ y ✖ las opciones que serán seleccionadas en el modelo final y las que no.

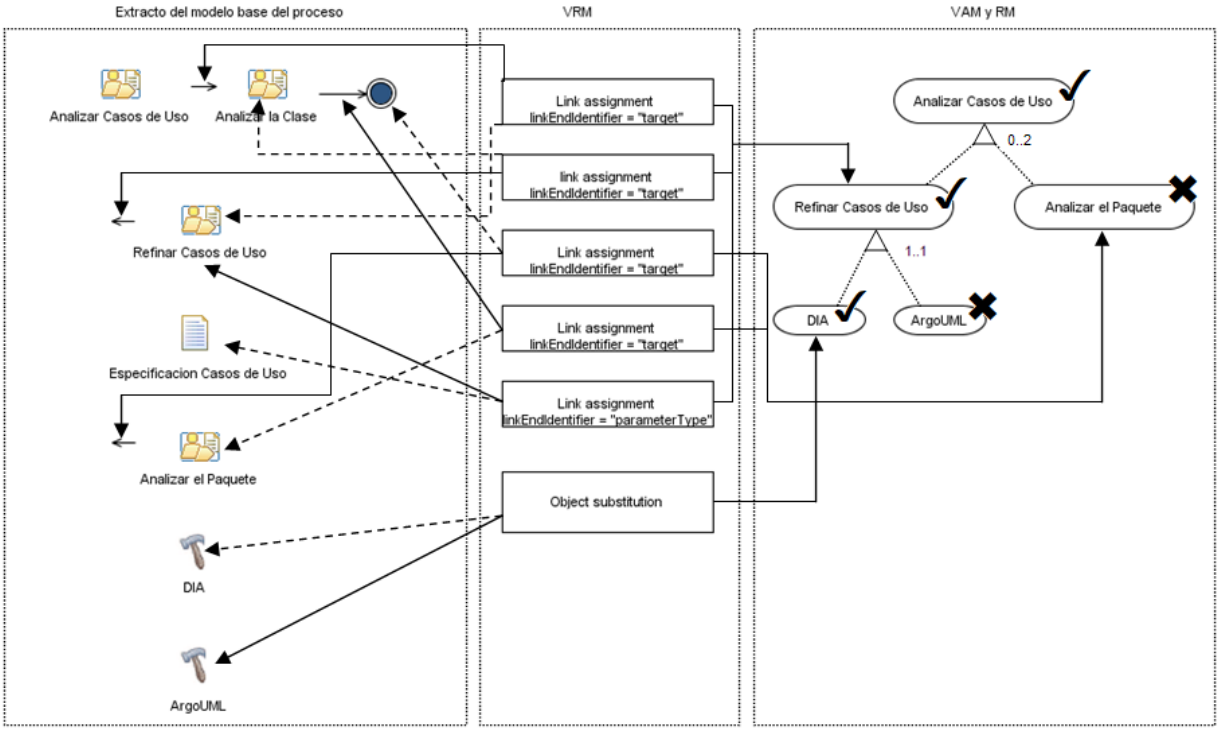


Figura 2.21. VAM, VRM y uno de los posibles RM

En el modelo de la Figura 2.22, se presenta el modelo final según las elecciones realizadas en el RM presentado anteriormente. Se puede observar que aparece la actividad *Refinar Casos de Uso* y que la misma utiliza la herramienta *DIA.*

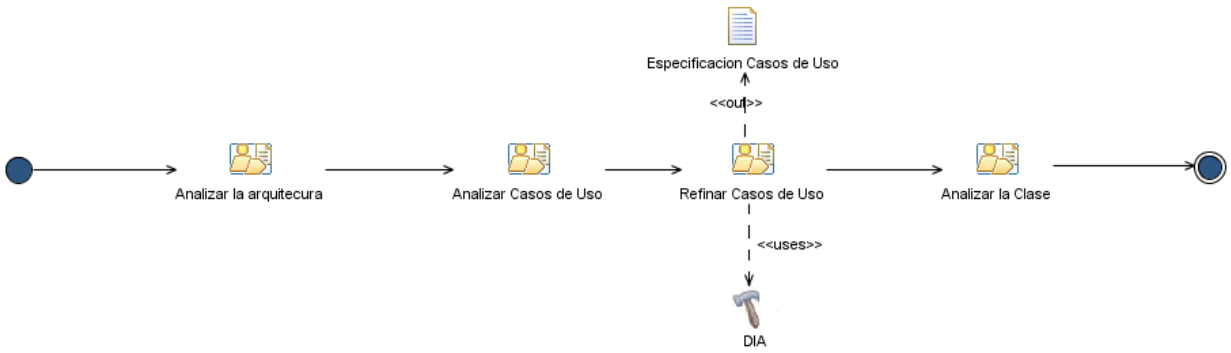


Figura 2.22. Modelo adaptado según RM

Ventajas

- Permite la derivación automática de un SPL de acuerdo a los requerimientos de un proyecto.
- Es independiente del metamodelo en el que está especificado el modelo.
- Permite vincular el modelo de variabilidad directamente con el modelo base del proceso, de forma que se puede ver explícitamente qué elementos de éste se ven afectados.

Desventajas

- Los modelos son difíciles de editar y mantener.

2.3.8. vSPEM

Teniendo en cuenta las limitaciones de los mecanismos de variabilidad en SPEM 2.0, principalmente desde el punto de vista de la comprensibilidad, surgió vSPEM [14][17], una extensión de SPEM 2.0 basada en SPrL, que permite la especificación directa de la variabilidad del proceso. En esta propuesta, el ingeniero de procesos define los puntos de variación del proceso (llamados *VarPoints*), así como las variantes que pueden ocupar dichos puntos de variación. La relación entre un punto de variación y sus variantes es una relación *replaces* de SPEM 2.0, y durante la instanciación del proceso, cada punto de variación se sustituye por exactamente una variante.

En este enfoque se define una nueva notación para representar los puntos de variabilidad y variantes, donde se proponen nuevos iconos, tal como se muestra en la Tabla 2.3.

	Activity	WorkProductUse	RoleUse	TaskUse
Base Element	Activity	WorkProductUse	RoleUse	TaskUse
VarPoint	VPActivity	VPWorkProductUse	VPRoleUse	VPTaskUse
Variant	VActivity	VWorkProductUse	VRoleUse	VTaskUse

Tabla 2.3. Iconos propuestos para el enfoque vSPEM

Ejemplo de aplicación

La Figura 2.23 representa el modelo adaptado de la Figura 2.5, utilizando la notación propuesta en vSPEM.

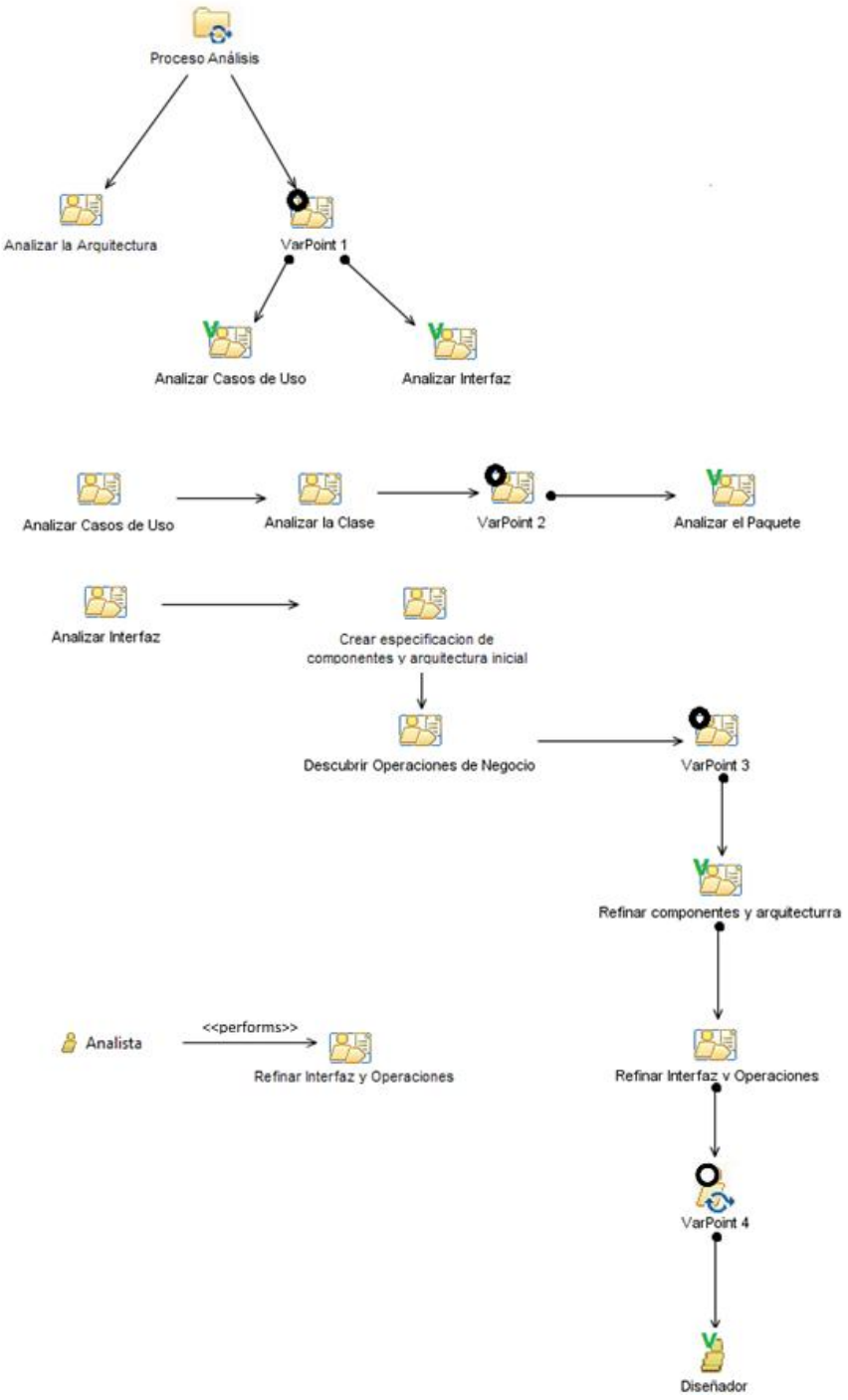


Figura 2.23. Modelo con vPEM

Las actividades obligatorias no presentan cambio en su representación en cuanto a la notación de SPEM. La actividad *Analizar la Arquitectura* es un ejemplo de una actividad

que no presenta variación y por lo tanto estará presente en todos los procesos específicos derivados del modelo .

Los puntos de variación son representados por . En dichos puntos están asociadas las actividades variantes ( ). En el ejemplo, el punto de variación *VarPoint 1* tiene dos variantes asociadas: *Analizar Casos de Uso* y *Analizar Interfaz*. En un proceso derivado estará presente una de dichas actividades.

Además, se puede dar el caso en el que un punto de variación solo tenga asociada una variante. En este caso, la actividad variante puede realizarse o no. Se puede visualizar este caso en el modelo en *VarPoint 2*, donde la actividad *Analizar el Paquete* puede estar presente o no en el proceso derivado. Algo similar sucede con el *VarPoint 3*.

Finalmente, notar que el *VarPoint 4* tiene una notación diferente ( ), ya que indica un punto de variación para un rol, en este caso, el rol diseñador.

Un ejemplo de un proceso derivado para el modelo se representa a continuación en la Figura 2.24.

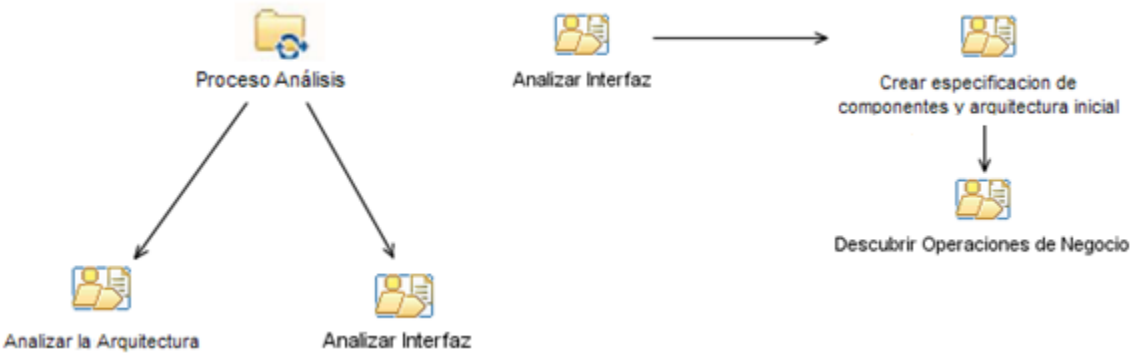


Figura 2.24. Modelo adaptado para vSPEM

Ventajas

- Este nuevo mecanismo de variación permite que la especificación esté dentro del metamodelo de SPEM.
- Debido a que las variantes son elementos muy pequeños, pueden fácilmente ser reutilizados en otros puntos de variación.

- Esta aproximación incluye una notación gráfica con la que la variabilidad puede ser más fácilmente modelada en diagramas SPEM 2.0.

Desventajas

- No soporta la derivación automática de procesos específicos.
- No cuenta con una herramienta de soporte estable. Sin embargo, existe una herramienta desarrollada como proyecto de grado, denominada vEPF.

Este enfoque fue el utilizado en este proyecto, y la motivación para ello la explicaremos en el Capítulo 3.

2.3.9 Resúmen

La Tabla 2.4 muestra de forma resumida, para cada enfoque estudiado, los elementos sobre los cuales es posible aplicar la variabilidad. Estos elementos se listan a continuación:

- *Actividades:* En el contexto de SPEM, son el elemento central para definir procesos, dado que permiten organizar los elementos básicos, tales como roles, productos de trabajo y tareas. En BPMN, es un término genérico que hace referencia al trabajo que realiza una empresa en un proceso, y existen tres tipos de actividades: Procesos, SubProcesos y Tareas.
- *Tareas:* son la unidad mínima de trabajo que puede existir en un modelo de proceso. Aplica tanto para SPEM como para BPMN.
- *Steps:* representan los pasos necesarios para realizar, de forma ordenada, una tarea (solo para SPEM).
- *Roles:* definen un conjunto de habilidades, competencias y responsabilidades relacionadas, de un individuo o de un grupo. En BPMN se representan mediante *lanes*.
- *Artefactos (work products):* en el contexto de SPEM, representan un artefacto que es consumido, producido o modificado por tareas.
- *Procesos/Subprocesos:* en el contexto de BPMN, los subprocesos son un conjunto de actividades con una determinada secuencia lógica para cumplir un propósito. Son procesos en sí mismos, cuya finalidad forma parte de un proceso más grande.
- *Control de flujo (gateways):* se utilizan para controlar la divergencia y convergencia de los flujos de secuencia en un proceso.

- *Eventos:* son algo que sucede en el curso de un proceso y afectan el flujo del modelo.
- *Flujo de secuencia:* se utilizan para mostrar el orden en que las actividades se realizarán en un proceso.

Elementos	SPEM 2.0	Modelo de Características	OVM	vSPEM	Smarty SPEM	BPMNt	CASPER	CVL
Actividades	x	x	x	x	x		x	x
Tareas	x	x	x	x	x	x	x	x
Procesos/ Subprocesos						x		
Steps	x	x	x		x			x
Roles	x	x	x	x	x		x	x
Artefactos (Work products)	x	x	x	x	x		x	x
Control de flujo (Gateways)						x		
Eventos						x		
Flujo de secuencia						x		

Tabla 2.4. Comparación a nivel de elementos

Dejando de lado BPMNt, ya que por estar basado en BPMN utiliza elementos diferentes para el modelado, podemos notar que todos los enfoques restantes admiten variabilidad sobre los mismos tipos de elemento, exceptuando vSPEM y CASPER que no admiten variabilidad sobre los *steps*. Esto nos permite afirmar que en cuanto a los elementos sobre los que es posible aplicar la variabilidad no existe una diferencia sustancial entre los distintos enfoques estudiados.

Si analizamos la Tabla 2.5, podemos ver que el único enfoque que permite agregar elementos nuevos (variantes) al proceso base es BPMNt, lo que significa que utiliza variabilidad por extensión. Los restantes enfoques manejan todos variabilidad por restricción.

Por otra parte, los únicos enfoques que tienen actualmente una herramienta de soporte para el modelado de los procesos son SPEM 2.0, Modelo de características, vSPEM (aunque no es una herramienta estable), BPMNt, CASPER y CVL.

Para finalizar, queremos resaltar que tanto CASPER como CVL permiten la derivación automática del modelo adaptado, ahorrándole tiempo al usuario a la hora de generar un nuevo modelo adaptado.

	SPEM 2.0	**Modelo de Características**	**OVM**	**vSPEM**	**Smarty SPEM**	**BPMNt**	**CASPER**	**CVL**
Lenguaje	SPEM 2.0	-	-	SPEM 2.0	SPEM 2.0	BPMN 2.0	SPEM 2.0	Independiente
Basado en		FMs	SPLs	SPrLs	SPrLs		MDE (y FMs)	SPL
¿Permite nuevos elementos?						x		
¿Tiene herramienta de modelado?	x	x		x		x	x	x
¿Deriva el modelo automáticamente?							x	x

Tabla 2.5.Comparación de enfoques

Capítulo 3

Requerimientos y solución propuesta

En esta sección se describen los requerimientos funcionales y no funcionales del prototipo desarrollado, así como el enfoque y la herramienta de base utilizada. También se describe la solución propuesta y las tecnologías que fueron empleadas, y se mencionan algunos detalles de implementación y pruebas de verificación realizadas.

3.1. Especificación de requerimientos

3.1.1. Requerimientos

Los requerimientos funcionales para el desarrollo del prototipo son principalmente tres:

> ***R1.*** *Importar proceso base:* A partir de un proceso de software con variabilidad (proceso base) modelado en vEPF, la herramienta a desarrollar deberá permitir importar desde un repositorio *GitHub* o similar dicho proceso, exportado en un archivo en formato *.xml* o *.xmi*.
>
> ***R2.*** *Configurar variante:* Una vez importado el modelo correspondiente al proceso base, la herramienta a desarrollar deberá mostrarlo gráficamente y permitir la adaptación (configuración) del mismo, mediante la selección de las variantes correspondientes en los diferentes puntos de variación que puedan existir en el modelo.
>
> ***R3.*** *Exportar variante:* La herramienta a desarrollar deberá permitir exportar el modelo adaptado por el usuario a un archivo *.xml*, de forma que pueda ser

cargado y visualizado en Eclipse EPF, además de dar la posibilidad de subirlo a un repositorio *GitHub* o similar. También, de ser posible, deberá dar la opción de publicar el modelo, del mismo modo en que lo hace Eclipse EPF.

La Figura 3.1 muestra de forma gráfica la interacción entre vEPF y la herramienta a desarrollar.

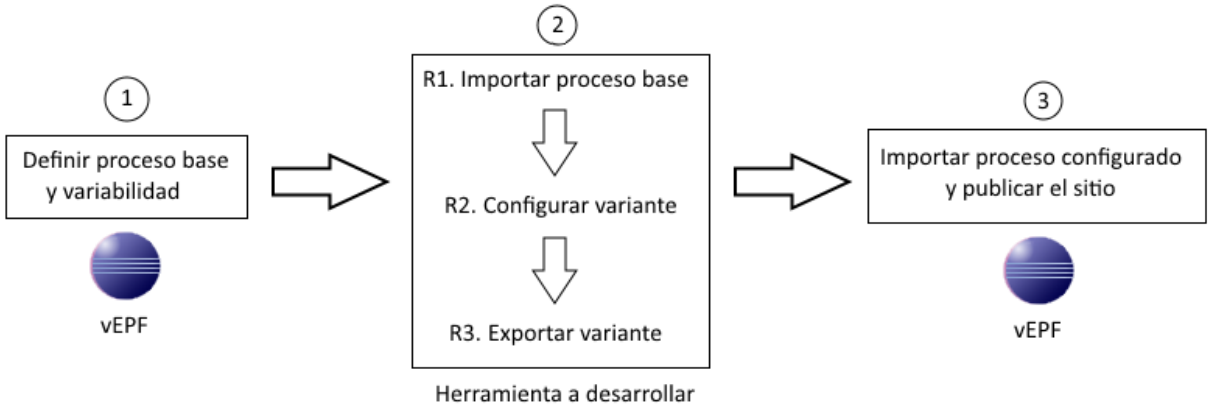


Figura 3.1. Interacción entre sistemas

En cuanto a los requerimientos no funcionales, se pidió expresamente que la herramienta fuese amigable y fácil de utilizar, ya que está orientada principalmente a personal no técnico.

3.1.2. Definición de casos de uso

Los casos de uso son los que se desprenden directamente de los requerimientos funcionales. En la Figura 3.2 mostramos el diagrama de casos de uso que los relaciona:

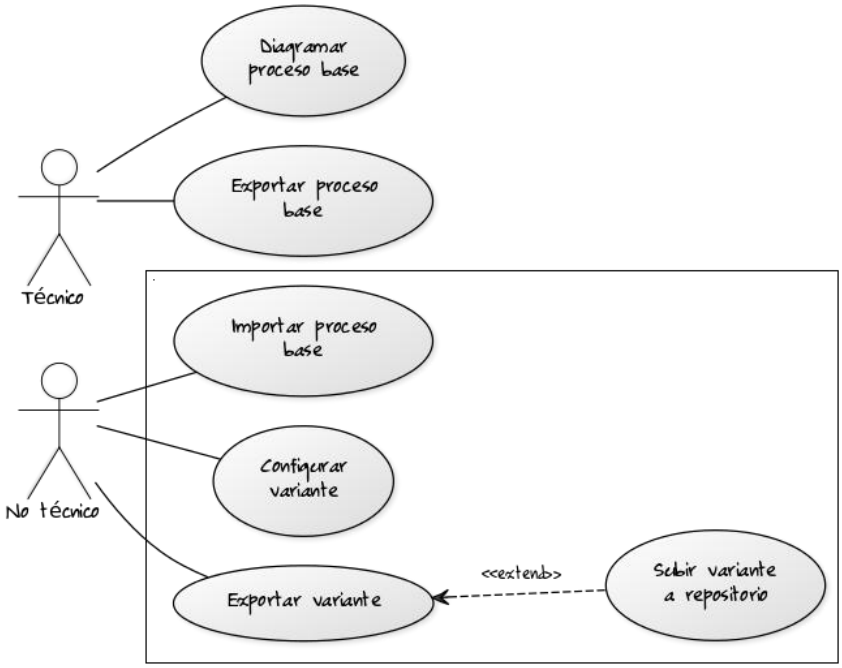


Figura 3.2. Diagrama de casos de uso

Los casos de uso que nos competen son los que quedan dentro de lo que marcamos como la frontera del sistema. Los otros fueron incluidos en la imagen simplemente con el fin de mostrar lo que se realiza por fuera del uso de nuestra aplicación.

Como se puede ver, existen dos actores:

- *Técnico:* Usuario técnico que conoce el uso de vEPF, y que tiene conocimiento sobre modelado de procesos de software, y en particular, modelado con vSPEM. Éste no interactúa con nuestra aplicación.
- *No técnico:* Gerente de la empresa o, eventualmente, cualquier otro usuario no técnico que se encarga de generar los modelos de proceso particulares requeridos para cada realidad, mediante el uso de nuestra aplicación.

En cuanto a los casos de uso, a continuación describimos brevemente su funcionamiento:

- *Importar proceso base:* El usuario no técnico decide importar un modelo con variabilidad, ya sea desde un repositorio *Github o un directorio local*. Para ello ingresa la URL del repositorio o la ruta en el directorio local, el sistema verifica que la ruta ingresada sea correcta y muestra los archivos en formato *.xmi* existentes en dicha ruta. El usuario selecciona el archivo *.xmi* que contiene la

información principal del modelo, el sistema verifica que el archivo tenga un formato válido y lo carga.

- *Configurar variante:* El usuario selecciona las variantes que desea para cada punto de variación. Una vez finalizado este proceso de selección, el usuario presiona el botón “Siguiente” y el sistema verifica que se cumplan todas las restricciones establecidas por el modelo sobre los puntos de variación. De ser así, finaliza el caso de uso; de lo contrario, el usuario deberá corregir las inconsistencias existentes para poder continuar.
- *Exportar variante:* Una vez finalizado el proceso de configuración del modelo, el usuario selecciona si desea exportar el modelo a un repositorio o no, y presiona finalizar. Si eligió no exportar a un repositorio, el sistema simplemente genera en el directorio local el archivo *.xml* correspondiente al modelo adaptado, junto con las imágenes y documentos necesarios. De lo contrario, además de esto se desprenderá el caso de uso *Subir variante a repositorio*, el cual alojará en el mismo una copia de los archivos generados localmente.

3.2. Solución propuesta

3.2.1. Enfoque seleccionado

A la hora de seleccionar cuál sería el enfoque a utilizar, tuvimos en cuenta los siguientes aspectos:

- El lenguaje en el cual está basado el enfoque, recordando que interesan particularmente los que utilizan el lenguaje SPEM 2.0.
- Sobre qué/cuántos elementos es posible aplicar la variabilidad, concluyendo, como mencionamos en la sección 2.3.9, que en este sentido todos los enfoques se encuentran en la misma posición, ya que ninguno ofrece mayores beneficios que otros.
- El soporte tecnológico existente, punto en el que podríamos considerar: Modelo de Características, BPMNt, CASPER, CVL y vSPEM.
- Qué tan comprensible resulta la metodología, para lo que podríamos considerar: OVM, SmartySPEM y vSPEM.

Esto nos lleva a decidir que el enfoque candidato para la realización del proyecto es vSPEM, dado que cumple con todos los requisitos buscados.

En la sección siguiente presentamos la herramienta base utilizada, explicando los detalles más relevantes de la misma.

3.2.1.1. Herramienta base

EPF Composer

Eclipse Process Framework Composer (EPFC) [16] es un editor de SPEM 2.0, basado en el estándar de SPEM 2.0, y más precisamente, en UMA (*Unified Method Architecture*). Es una herramienta gratuita, basada totalmente en el ambiente de desarrollo Eclipse, que permite editar fragmentos de método, procesos o metodologías y publicar de forma automática sitios web. Estos fragmentos se almacenan en formato XMI.

EPFC es una herramienta orientada a ingenieros de procesos y administradores de proyectos principalmente, dado que son los responsables de mantener e implementar los procesos de las organizaciones.

vEPF

Como ya mencionamos, el enfoque vSPEM permite especificar directamente la variabilidad en la SPrL dentro del metamodelo de SPEM, utilizando una nueva notación para representar los puntos de variabilidad y variantes. Esto permite que la variabilidad pueda ser fácilmente modelada en diagramas de SPEM 2.0, además de ser sencilla de entender. También mencionamos que el enfoque no tiene soporte tecnológico estable, aunque existe una herramienta que fue desarrollada como proyecto de fin de carrera, y es en ella en quien nos basamos para la realización del prototipo presentado en éste proyecto.

La herramienta a la que nos referimos es ***vEPF***, desarrollada, como dijimos, para un proyecto de fin de carrera, en la Universidad de Castilla - La Mancha, en 2012. La misma extiende el entorno de Eclipse EPF (Eclipse Process Framework) para dar soporte a la extensión vSPEM. Permite crear procesos ricos en variantes y hacer una gestión de estos, mediante la definición de líneas de proceso. Esto significa que permite modelar el proceso con variabilidad y luego obtener el proceso adaptado a las necesidades de la realidad en cuestión.

La Figura 3.3 muestra una captura de la herramienta.

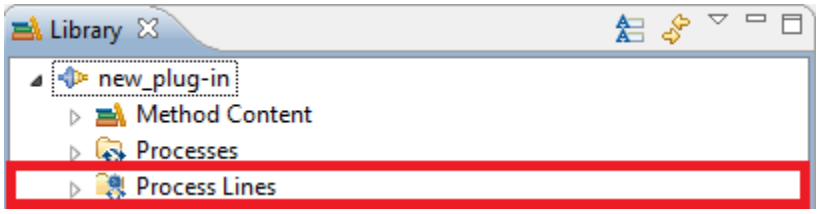


Figura 3.3. Captura vEPF

La carpeta *Processes* es la provista por EPFC, donde se definen los *Capability Pattern* y *Delivery Process* (sin variabilidad). Por otra lado, la carpeta *Process Lines* que se encuentra resaltada forma parte de la extensión desarrollada, en la cual se definen los *Capability Pattern* y *Delivery Process* que admiten variabilidad mediante puntos de variación y variantes. Las líneas de proceso definidas se crearán bajo esta carpeta, generando una estructura similar a la que se muestra en la Figura 3.4.

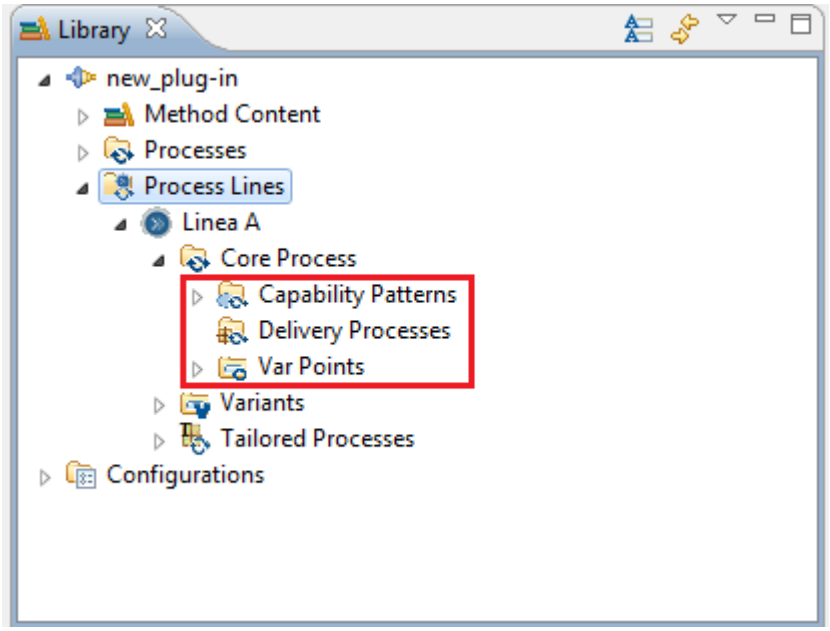


Figura 3.4. Líneas de proceso

En las carpetas *Var Points* y *Variants* aparecerán los puntos de variación y variantes definidos tanto en *Capability Patterns* como en *Delivery Processes*, permitiendo ser editados desde allí. También es posible definir relaciones de dependencia entre puntos de variación y variantes.

En *Tailored Processes* se podrán crear diferentes adaptaciones del modelo, que serán una copia idéntica de los procesos del *Core Process*, con la salvedad de que no habrán puntos de variación ni variantes dentro de los procesos.

Sin embargo, vEPF presenta algunas limitaciones o cuestiones que deberían ser mejoradas:

- La herramienta no utiliza la representación gráfica de los procesos con variabilidad según la notación planteada en vSPEM. Los diagramas de actividad del proceso no usan la notación propuesta por el enfoque.
- Los procesos definidos con variabilidad solo permiten definirla en un primer nivel, es decir, no es posible definir un punto de variación como hijo de un elemento de vSPEM.
- Solo los usuarios técnicos, que conocen de EPF, pueden realizar la adaptación del proceso, ya que el uso de la herramienta requiere el manejo de EPF.
- La publicación del proceso en la web no incluye los procesos definidos bajo línea de procesos, es decir, los procesos con variabilidad.

Estas limitaciones nos impulsaron a desarrollar una herramienta que tuviese en cuenta dichos aspectos y así permitiera que, por ejemplo, un usuario que no necesariamente es técnico, pueda adaptar de manera sencilla el proceso de acuerdo a sus necesidades. La solución propuesta para la herramienta a desarrollar se plantea en la sección siguiente.

3.2.2. Solución propuesta

En base a los requerimientos planteados, la solución propuesta se basa en el desarrollo de una aplicación web en la que queden claramente identificadas todas las funcionalidades que el usuario puede realizar, de forma clara y rápida. Para ello, diseñamos una interfaz que plantea las funcionalidades como pasos a seguir (Figura 3.5), de forma de ir guiando al usuario en el uso de la aplicación.

Figura 3.5. Menú

Eclipse EPF, y en particular, *vEPF*, permite exportar cualquier proceso a una estructura de carpetas que contiene todos los archivos *.xmi*, imágenes y documentos necesarios para levantar el proceso en otro Eclipse EPF. Es por esto que en relación al requerimiento **R1** decidimos importar, y por lo tanto parsear, archivos en formato *.xmi*. También en relación a este requerimiento, resolvimos utilizar un repositorio *GitHub* como repositorio de los archivos:

https://github.com/ModelosVEPF/Modelos

Adicionalmente, decidimos incluir la posibilidad de importar archivos desde un directorio local.

En cuanto al requerimiento **R2**, decidimos utilizar una estructura arborescente para el armado gráfico del modelo, dado que consideramos que es intuitivo y fácil de entender. Los puntos de variación del modelo estarán identificados como *links*, que al ser cliqueados permitirán seleccionar la variante que ocupará dicho punto.

Por último, en lo que refiere al requerimiento **R3**, una vez que el usuario finalizó la adaptación del modelo, se le dará la posibilidad de decidir si desea subir al repositorio el archivo resultado de exportar el modelo final. Al igual que en el caso de importar el archivo, decidimos que el repositorio utilizado deberá ser *GitHub*.

La funcionalidad de publicar el modelo directamente desde nuestra herramienta, de la misma forma en que lo hace Eclipse EPF, optamos por no realizarla, ya que luego de un tiempo de investigación acotado no hallamos la forma de resolverlo.

En la siguiente sección detallaremos el diseño de la solución.

3.2.2.1. Diseño de la solución

Para el diseño de la solución elegimos una arquitectura de dos capas: presentación y lógica de negocio. La capa de presentación es la encargada de la interfaz de usuario, la responsable de la interacción entre éste y el sistema. Esta capa se comunica con la capa lógica, que recibe las peticiones del usuario y las procesa para luego retornarle una respuesta. La capa lógica, además, se comunica con el repositorio Github para obtener la información de los modelos requeridos por parte del usuario (en caso de haber seleccionado la opción de importar desde el repositorio) y luego, eventualmente, guardarlos en el repositorio nuevamente. Recordar que tanto la funcionalidad de

importar como la de exportar admiten que se realicen de forma local, en cuyo caso la comunicación con Github no se realiza.

El diagrama de componentes asociado se muestra en la Figura 3.6.

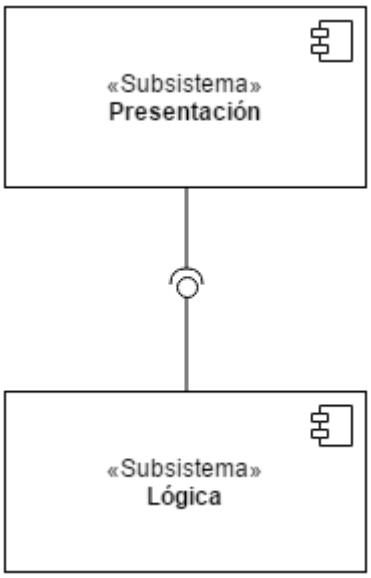


Figura 3.6. Diagrama de componentes

El subsistema “Presentación” (Figura 3.7) se compone principalmente de los siguientes componentes:

- ImportarModeloBean: Implementa todo lo referente a la interfaz de usuario relacionado con importar el proceso base desde el repositorio o directorio local.
- AdaptarModeloBean: Implementa todo lo referente a la interfaz de usuario que permite configurar una variante para el proceso base.
- ExportarModeloBean: Implementa todo lo referente a exportar la variante configurada.
- VistaBean: Mantiene información que es compartida entre los tres componentes antes mencionados.

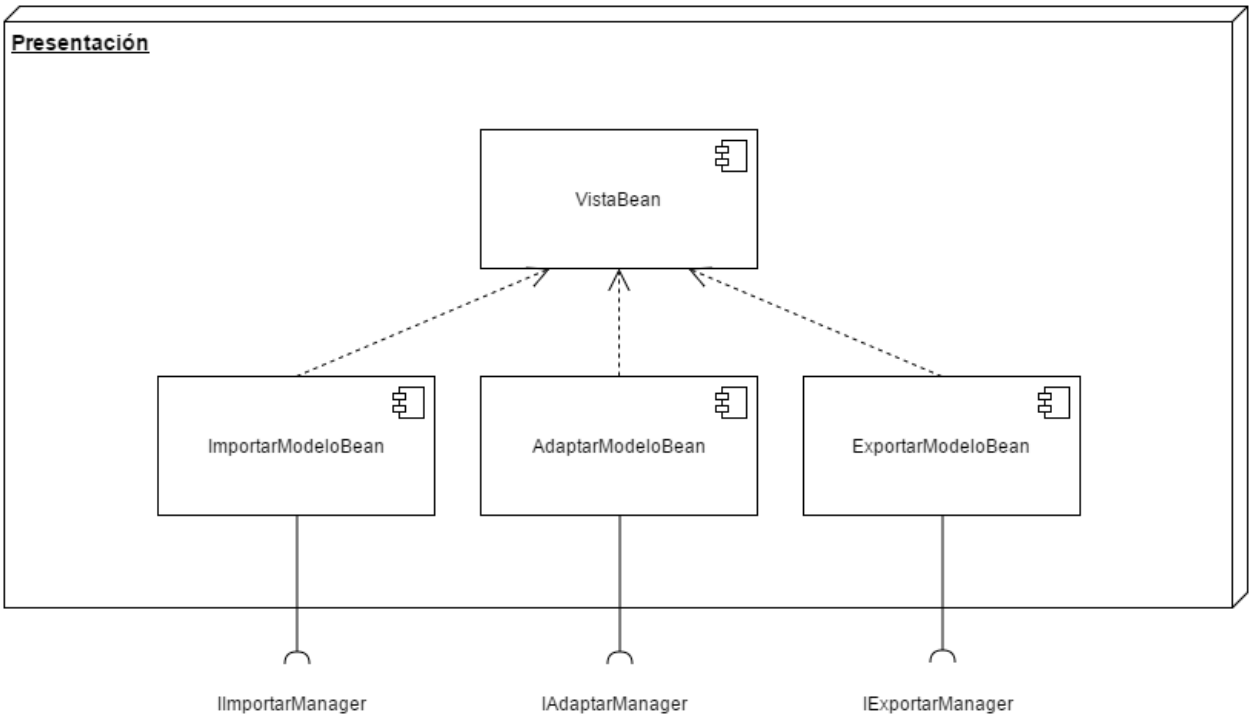


Figura 3.7. Subsistema Presentación

El subsistema “Lógica” (Figura 3.8), por otro lado, está compuesto principalmente de los componentes:

- ImportarManager: Implementa la lógica requerida para la descarga de los archivos correspondientes al proceso base (modelo a adaptar).
- XMIParser: Implementa la lógica necesaria para parsear los archivos descargados.
- AdaptarManager: Implementa la lógica necesaria para configurar una variante para el proceso base en cuestión.
- ExportarManager: Implementa la lógica necesaria para exportar la variante configurada.
- GitControl: Implementa la lógica necesaria para exportar archivos a GitHub.

IImportarManager, *IAdaptarManager* e *IExportarManager* son las interfaces responsables de la comunicación entre las capas Lógica y Presentación.

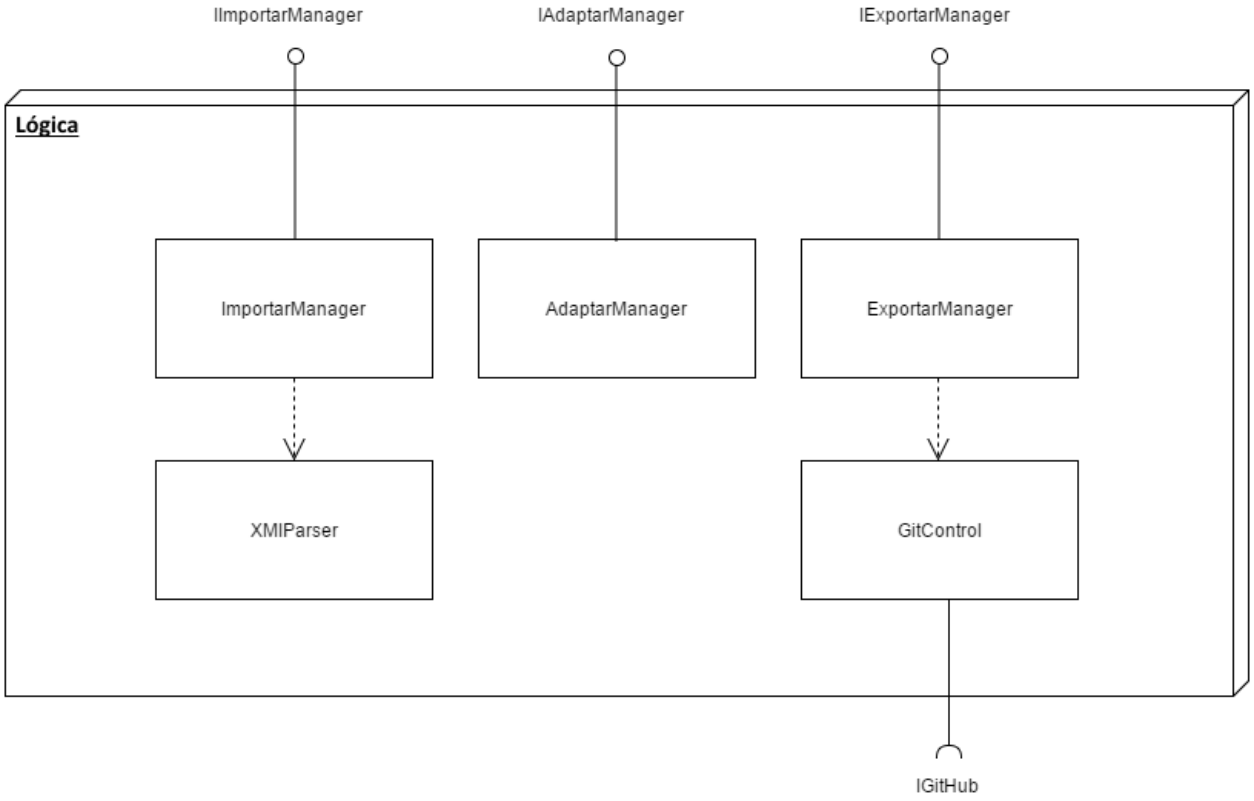


Figura 3.8. Subsistema Lógica

Dado que la solución no supone un sistema de gran porte y no se visualiza que vaya a crecer en un futuro, podemos pensar entonces en una *arquitectura de dos capas y un nivel*, lo que implica que ambas capas lógicas estarán ubicadas en un mismo servidor.

Figura 3.9. Arquitectura de la solución

En la Figura 3.10 se muestra el diagrama de despliegue.

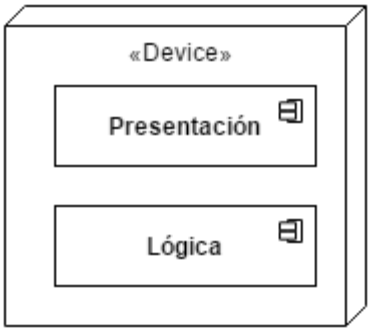


Figura 3.10. Diagrama de despliegue

3.2.3. Implementación

3.2.3.1. Tecnologías utilizadas

A continuación detallamos las principales tecnologías utilizadas para el desarrollo del proyecto. Las mismas fueron seleccionadas teniendo en cuenta las tendencias y versiones estables más actuales.

Github
Repositorio que utiliza el sistema de control de versiones y que decidimos utilizar tanto para el versionado del código fuente, como para alojar los modelos utilizados por la aplicación.

Google Drive
Centro de alojamiento de archivos en la nube, que permite compartir documentos y trabajar de forma simultánea sobre ellos, utilizado en este proyecto para gestionar toda la parte de documentación.

Java 8
Lenguaje de programación utilizado para el desarrollo del prototipo.

Red Hat JBoss Developer Studio 8.1.0GA
IDE (Entorno de Desarrollo Integrado) basado en Eclipse, utilizado para el desarrollo de la aplicación.

Maven
Herramienta de código abierto para la gestión y construcción de proyectos Java, que define, entre otras cosas, un sistema de gestión de dependencias.

Primefaces 5.2
Librería de componentes de código abierto para JSF 2.0 que brinda un amplio conjunto de componentes de interfaz de usuario.

JGit
Librería de código abierto que implementa el sistema de control de versiones Git en Java, utilizada en este proyecto para realizar la exportación al repositorio.

3.2.3.2. Detalles de implementación y verificación

Una vez definidos los requerimientos de la aplicación, comenzamos realizando algunos prototipos, tanto para la interfaz de usuario como para descartar posibles riesgos de implementación, principalmente en lo que refería a la utilización de *Github*.

Investigamos la posibilidad de incluir la funcionalidad de publicar el modelo adaptado desde la herramienta desarrollada sin tener que pasar por la herramienta vEPF, pero no hallamos la forma de reutilizar el plugin del Eclipse. Luego de esto, nos concentramos en ver si era factible actualizar la versión de EPF utilizada en la herramienta vEPF para poder trabajar con una tecnología más actual, pero después de un tiempo de investigación concluimos que no era posible realizarlo dentro de un tiempo razonable para el alcance del proyecto, ya que esta actualización implicaba una gran cantidad de cambios.

Luego de realizados estos prototipos, comenzamos sí a desarrollar las funcionalidades requeridas, de forma incremental. Empezamos con modelos sencillos, únicamente con

variabilidad en los elementos más pequeños y en el primer nivel, como ser las tareas y actividades, agregando luego los elementos restantes. En un principio solo permitimos importar un único archivo *xmi* (correspondiente a un *delivery process* o *capability pattern*), y una vez avanzados en el desarrollo, modificamos la funcionalidad de forma tal de que el archivo a importar sea el que contiene la información principal del modelo. Esto implica que al importar este archivo la aplicación busca todas las referencias a otros archivos y los descarga, teniendo así los archivos correspondientes al *delivery process*, los *capability patterns* asociados, y toda otra información que sea necesaria para cargar el modelo.

En paralelo con el desarrollo, fuimos realizando algunas pruebas funcionales, sencillas, de forma de verificar que los elementos nuevos que agregábamos al modelo tuvieran el funcionamiento esperado y no interfirieran con lo que ya teníamos implementado. Estas pruebas consistieron en construir con vEPF modelos que tuvieran variabilidad en los elementos que íbamos considerando en el desarrollo, y verificando que al configurar y exportar el modelo, todo resultara según lo esperado.

Al concluir con el desarrollo “más grande”, en una primera etapa, diseñamos algunos casos de prueba que utilizaban modelos más completos que los que veníamos usando hasta el momento. Esto nos permitió depurar un poco más las funcionalidades de importar y exportar, principalmente, agregando al modelo final más detalle respecto a los elementos que incluía.

Finalmente, diseñamos el caso de estudio que se describe en el capítulo siguiente, a fin de ser utilizado como principal caso de prueba, dado que contiene variabilidad en diferentes niveles y elementos.

Configuración

Todos los parámetros configurables de la aplicación se encuentran en la clase *configs.properties*, mostrada en la Figura 3.11.

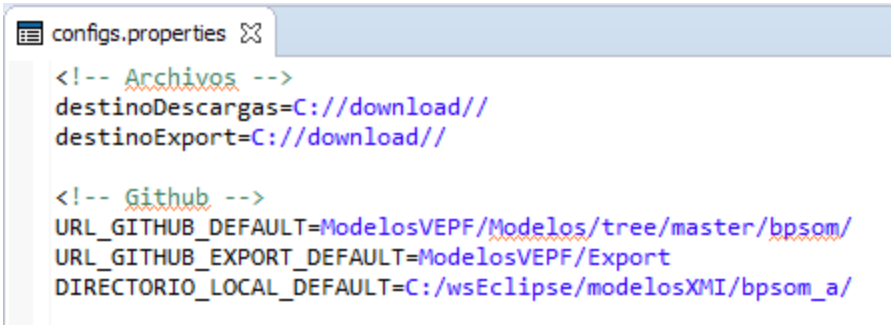


Figura 3.11. Archivo de configuración

Donde:

- *destinoDescargas* es el directorio local en el cual se efectuarán las descargas de los archivos correspondientes al modelo importado desde el repositorio.
- *destinoExport* es el directorio local en el cual se guardarán los archivos generados como resultado de adaptar el modelo.
- *URL_GITHUB_DEFAULT* es la URL del modelo que aparecerá por defecto para importar desde el repositorio.
- *URL_GITHUB_EXPORT_DEFAULT* es la URL del repositorio que aparecerá por defecto cuando se seleccione la opción de exportar al repositorio.
- *DIRECTORIO_LOCAL_DEFAULT* es la ruta del modelo que aparecerá por defecto para importar desde el directorio local.

Tener estos parámetros definidos en un archivo *.properties* permite principalmente, que puedan ser modificados las veces que sea necesario sin tener que recompliar el código cada vez, además de la facilidad que implica el tenerlos todos en un mismo archivo.

Capítulo 4

Caso de Estudio

El caso de estudio utilizado para probar el prototipo desarrollado, fue realizado en base a un proceso de desarrollo de software brindado por los tutores del proyecto, denominado BPSOM (*Business Process Service Oriented Methodology*) [24], el cual está enfocado en el desarrollo orientado a servicios desde procesos de negocio.

El caso de estudio proporcionado se muestra en la Figura 4.1, mediante un modelo representado en SPEM 2.0.

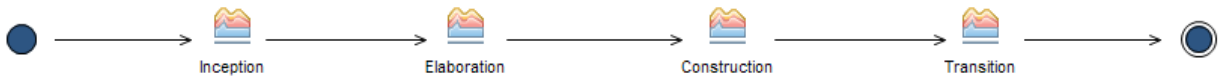


Figura 4.1. Proceso BPSOM

Como se puede observar, el proceso cuenta con cuatro fases: *Inception*, *Elaboration*, *Construction* y *Transition*. La fase *Inception* (en la Figura 4.2) tiene como objetivo el modelado del negocio, y se compone de las tareas: evaluar la organización de destino (*Asses the Target Organization*), identificar los procesos de negocio (*Identify Business Processes*), realizar el relevamiento de los requisitos (*Requirements Capture*) e identificar y categorizar los servicios (*Identify and Categorize Services*).

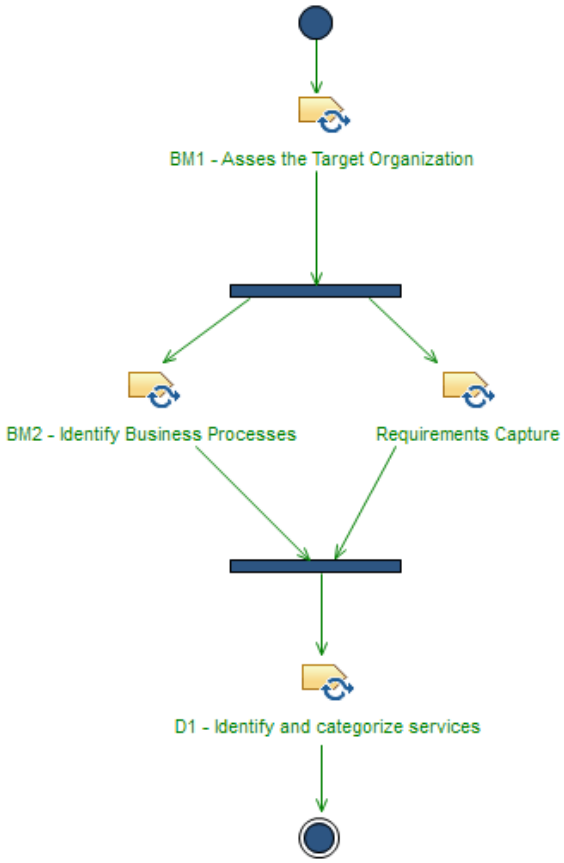


Figura 4.2. Tareas que componen la fase *Inception*

La fase *Elaboration* (en la Figura 4.3) se centra en el diseño del sistema orientado a servicios, mediante la identificación y especificación de los servicios necesarios (a través de las tareas *Identify and categorize services* y *Specify services*). Por otra parte, también permite seguir refinando los modelos o añadir nueva información o nuevos modelos, junto con la especificación de requerimientos para el sistema (mediante las actividades *Identify Business Processes* o *Requirements Capture*, por ejemplo).

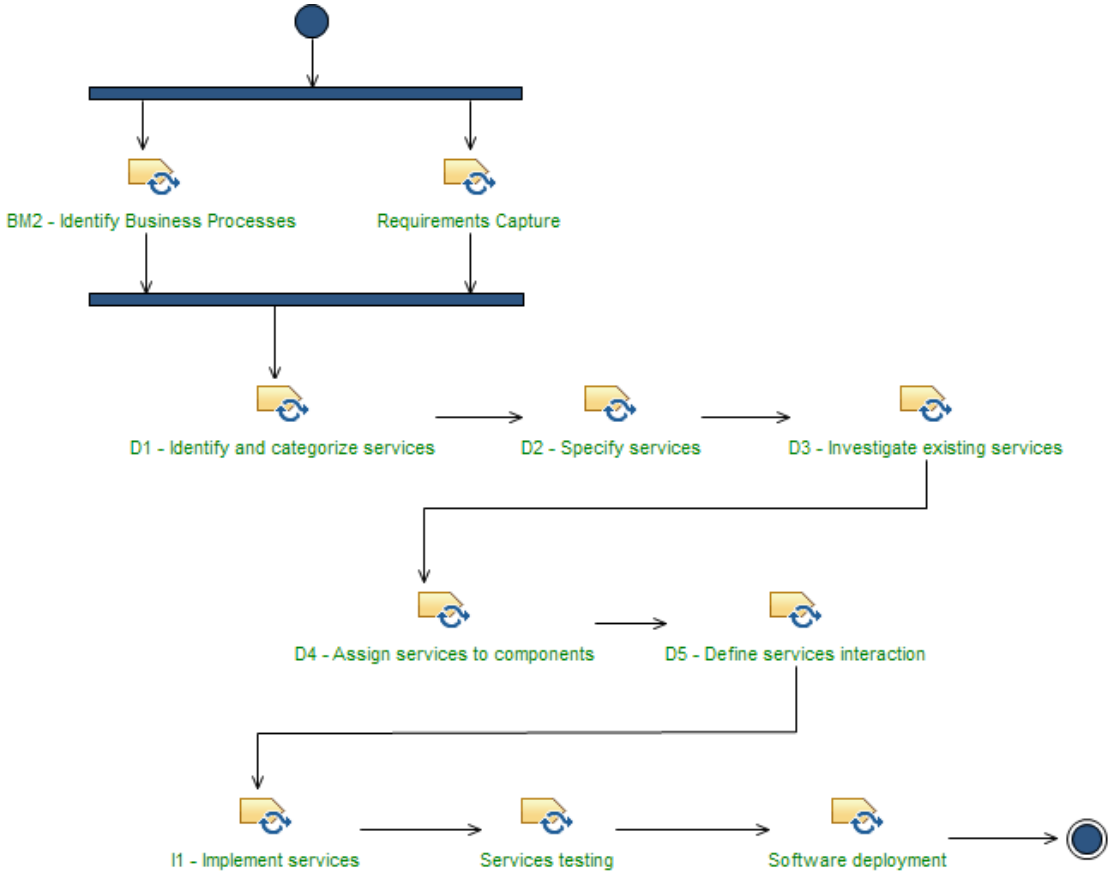


Figura 4.3. Tareas que componen la fase *Elaboration*

La fase *Construction* (en la Figura 4.4) tiene como finalidad el desarrollo de los servicios definidos para soportar los procesos modelados, aunque algunos servicios continúan siendo especificados en esta fase también. Las tareas que componen esta fase son: especificar servicios (*Specify services*), asignar servicios a los componentes (*Assign services to components*), definir la interacción entre servicios (*Define services interaction*), implementar los servicios (*Implement services*) y hacer el *deploy* del software (*Software deployment*).

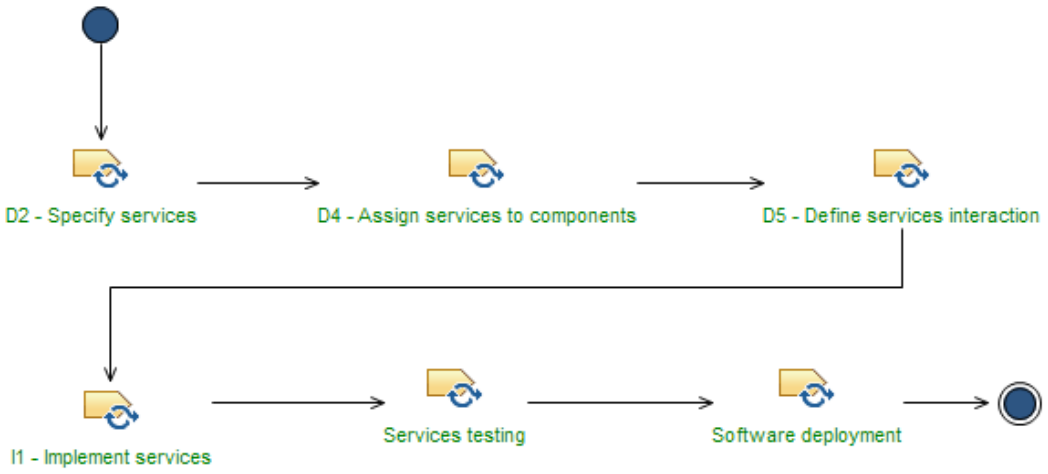


Figura 4.4. Tareas que componen la fase *Construction*

Por último, la fase *Transition* (en la Figura 4.5) pretende hacer el deploy del sistema orientado a servicios desarrollado en la infraestructura elegida, donde los servicios implementados se invocan desde los procesos ejecutados en un motor de procesos.

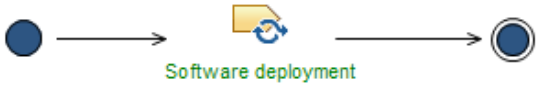


Figura 4.5. Tareas que componen la fase *Transition*

Sobre el modelo anterior agregamos algunos puntos de variación, de manera de tener variabilidad en todos los tipos de elementos.

En la Figura 4.6 podemos ver que se agrega una fase de nombre *VPPhase*, de tipo punto variación, la cual tiene como predecesor la fase *Construction* y como sucesor la fase *Transition*. Dicha fase tiene dos variantes de tipo fase, denominadas *VarPhase1* y *VarPhase2*.

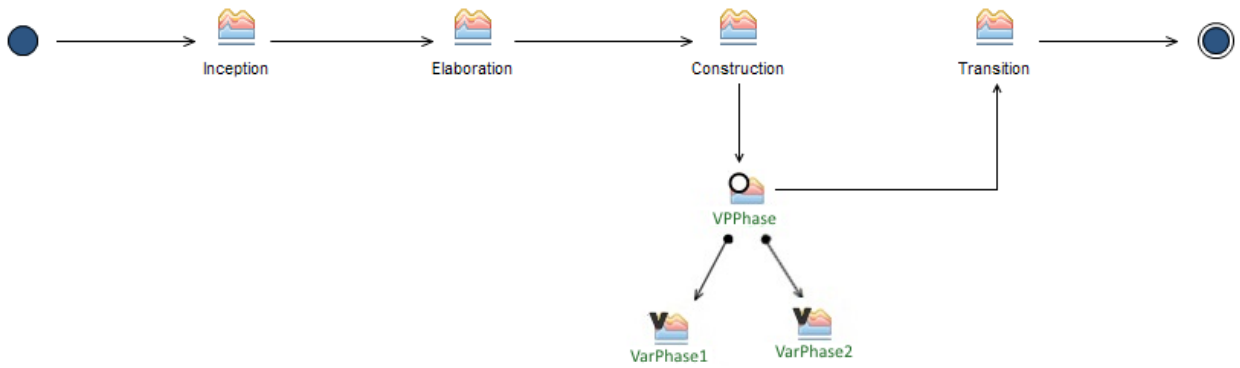


Figura 4.6. Proceso BPSOM - Agrega variabilidad en una fase

En la Figura 4.7 se muestra cómo continúa el modelo luego de haber elegido la variante en el punto anterior:

- Si se selecciona la variante *VarPhase1*, se realiza una iteración denominada *IterVarPhase1* y luego, un punto de variación de tipo actividad denominado *VPActivity*. Dicho punto de variación contiene dos variantes de tipo actividad, *VarActivity1* y *VarActivity2*; como restricción se agrega que al menos una de ellas debe ser seleccionada. Si se selecciona *VarActivity1,* se realiza la tarea *TaskVarActivity1*, y en caso contrario se realiza la tarea *TaskVarActivity2*.
- En caso de haber seleccionado la variante *VarPhase2,* se realiza una iteración de tipo punto de variación denominada *VPIter*, la cual consta de dos variantes, *VarIter1* y *VarIter2*. Si se selecciona *VarIter1* se realiza la tarea *TaskVarIter1*, y en caso contrario, se realiza la tarea *TaskVarIter2*.

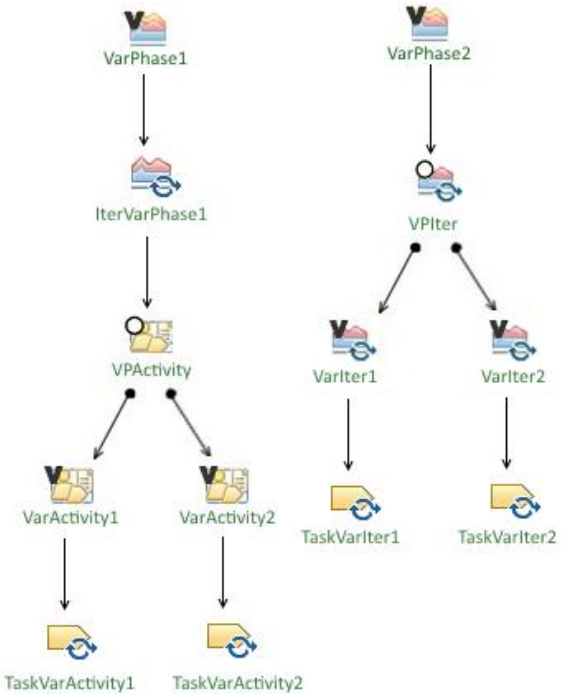


Figura 4.7. Elementos que componen las fases VarPhase1 y VarPhase2 - Agregan variabilidad en una actividad y una iteración respectivamente

En la Figura 4.8 se detallan las tareas realizadas en la fase *Elaboration*. En dicha fase se agrega una tarea de tipo punto de variación denominada *VPTask*. Este punto de variación consta de dos variantes, denominadas *VarTask1* y *VarTask2*.

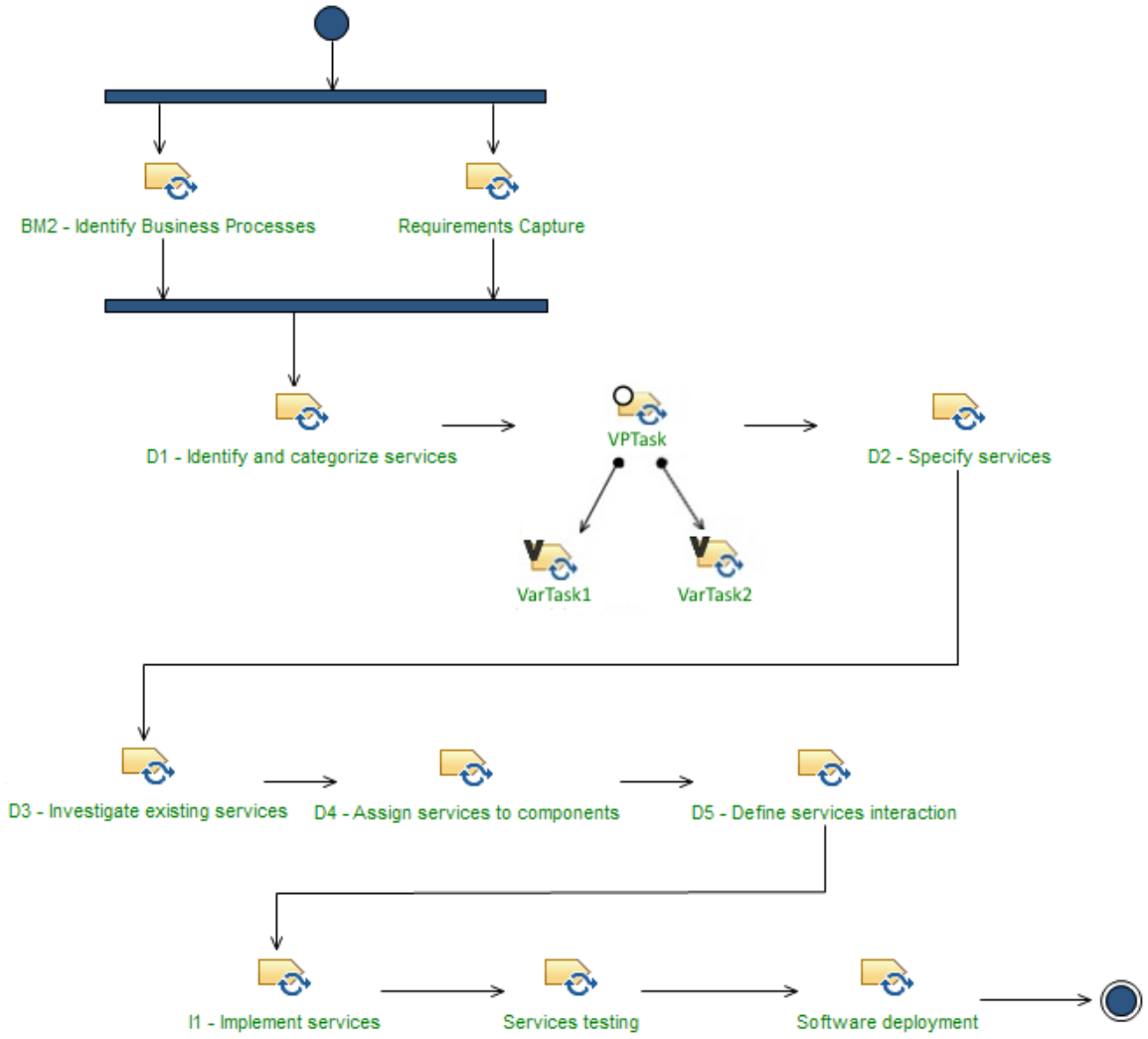


Figura 4.8. Tareas que componen la fase *Elaboration* - Agrega variabilidad en una tarea

Una vez definido el modelo con variabilidad, lo modelamos en vEPF de la siguiente manera:

- Definimos el proceso BPSOM de la Figura 4.6 como un *Delivery Process* bajo la sección *Process Lines*, ya que tiene variabilidad.
- Las tareas que componen las fases de *Inception*, *Construction* y *Transition* las definimos como *Capability Patterns* bajo la sección *Processes*, debido a que no presentan variabilidad.
- En cuanto a la variabilidad definida en el *Delivery Process*, debimos modelarla teniendo en cuenta las limitaciones de la herramienta en lo que respecta al

modelado, lo que implicó principalmente que no pudimos definir variabilidad en los niveles inferiores. Esto significa que no fue posible definir con vEPF puntos de variación como VPActivity o VPIter (Figura 4.7), por ejemplo, pero sí uno como VPPhase (Figura 4.6). La forma de modelar esto fue modificando los archivos *.xmi* generados al exportar el modelo.

- El *Capability Pattern Elaboration* debimos definirlo bajo la sección *Process Lines*, ya que presenta variabilidad en una de sus tareas. Esto también implicó que debiéramos modificar los *.xmi* para poder utilizarlo desde el *Delivery Process*, ya que vEPF tampoco permite utilizar *Capability Patterns* definidos bajo la sección *Process Lines*.

Luego, para exportar el modelo seguimos los siguientes pasos (Figura 4.9):

1. Vamos al menú 'File' y seleccionamos la opción 'Export'.
2. En el cuadro que se despliega a continuación seleccionamos la opción 'File System' y presionamos 'Next'.
3. En el siguiente cuadro marcamos la librería que deseamos exportar, el destino del export y presionamos 'Finish'.

Una vez realizadas todas las modificaciones sobre el modelo exportado, subimos la carpeta generada al repositorio deseado (en nuestro caso *https://github.com/ModelosVEPF/Modelos*).

A continuación, importamos el modelo en nuestra herramienta, ingresando la dirección del repositorio y seleccionando el archivo que contiene la información principal del modelo, tal como se muestra en las figuras 4.10 y 4.11. (Análogamente se puede hacer desde el directorio local).

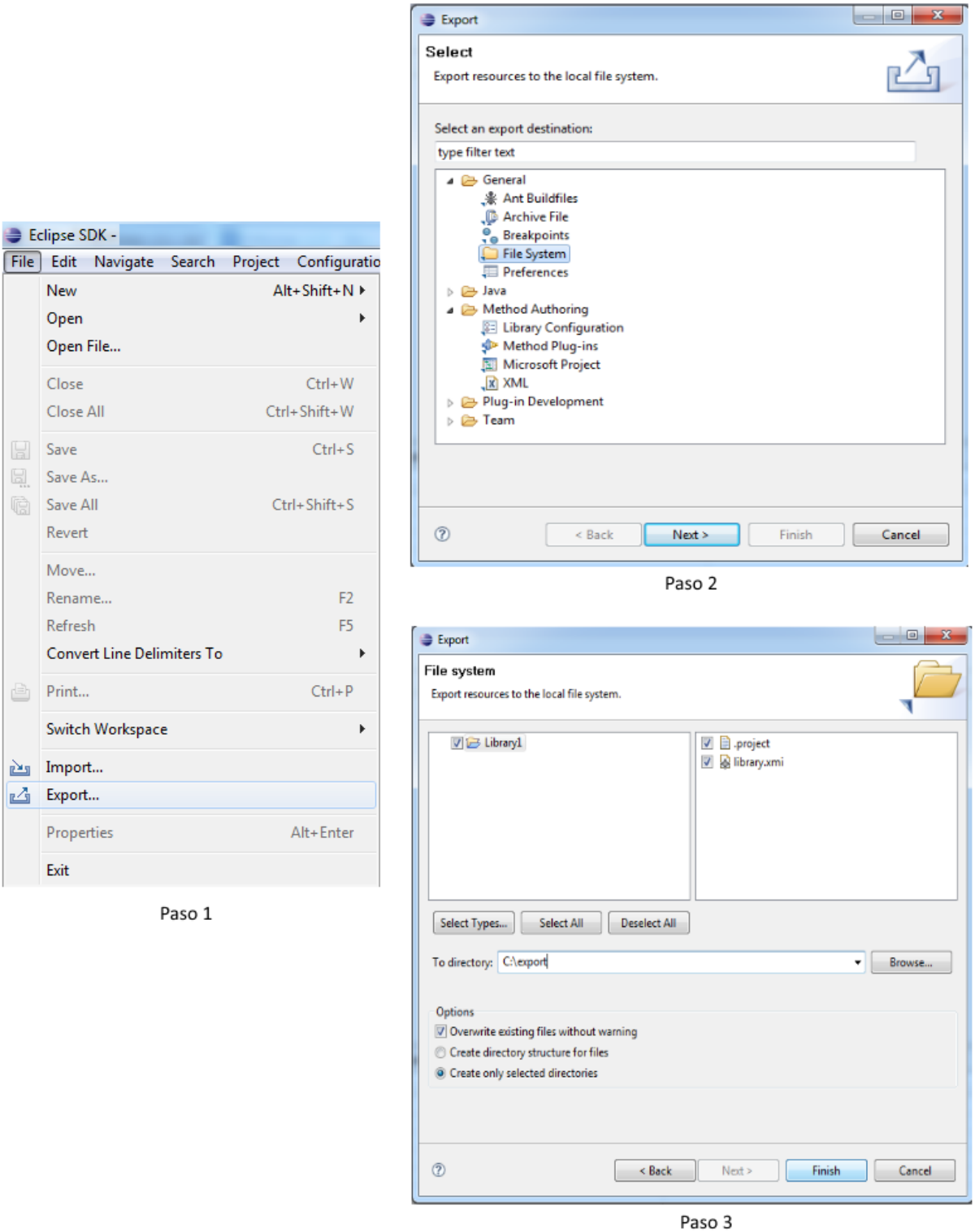


Figura 4.9. Exportar modelo

Figura 4.10. Ingresar URL repositorio

Figura 4.11. Seleccionar archivo a importar

Una vez finalizada la descarga del archivo, presionamos ‘Siguiente’ y se nos mostrará el modelo a adaptar, como se muestra en la Figura 4.12.

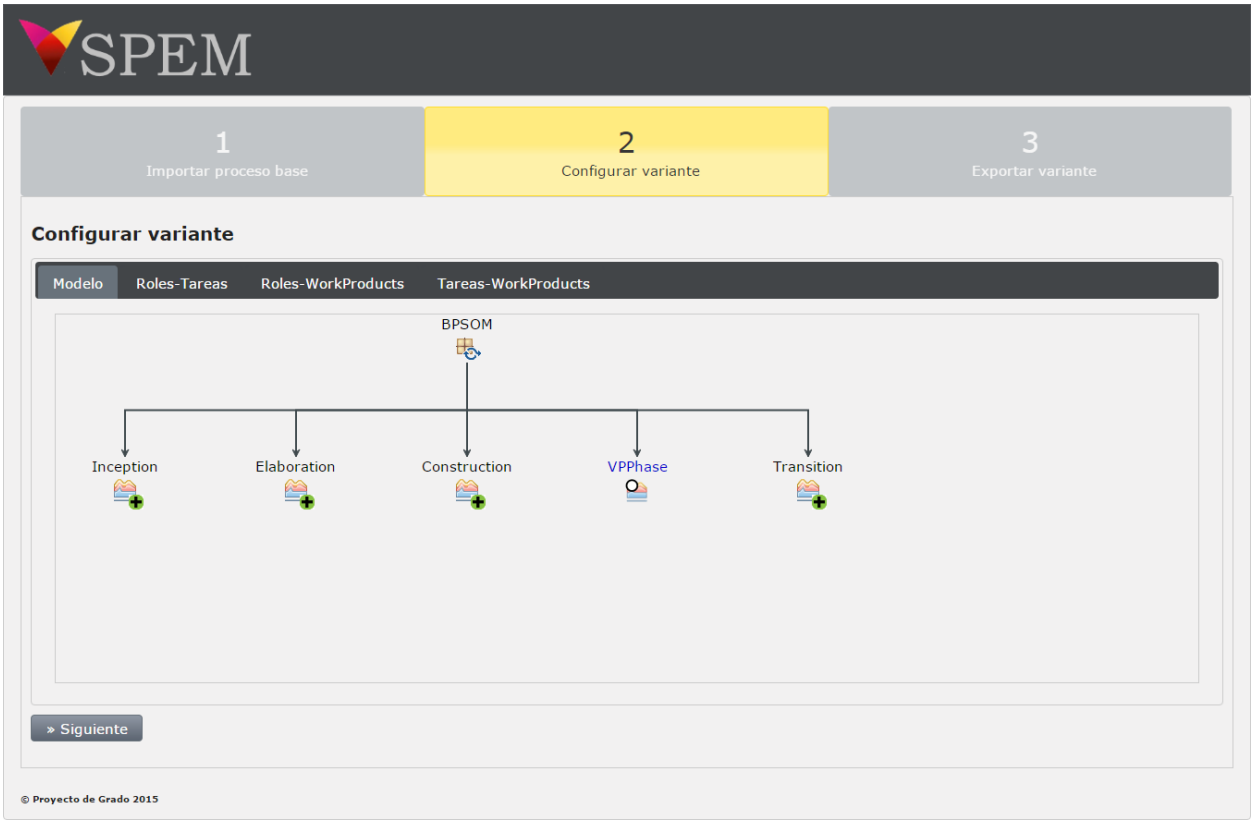


Figura 4.12. Vista del modelo

En dicha figura se puede observar que:

- Los íconos que se muestran con símbolos de más (+) pueden ser cliqueados para expandir (mostrar) los hijos del elemento en cuestión. Análogamente, los íconos que tengan un símbolo de menos (-) podrán ser cliqueados para ocultar sus hijos.
- Los íconos cuyo nombre tienen aspecto de link corresponden a puntos de variación, y pueden ser cliqueados para seleccionar la variante que ocupará el lugar de dicho punto (figuras 4.13 y 4.14).

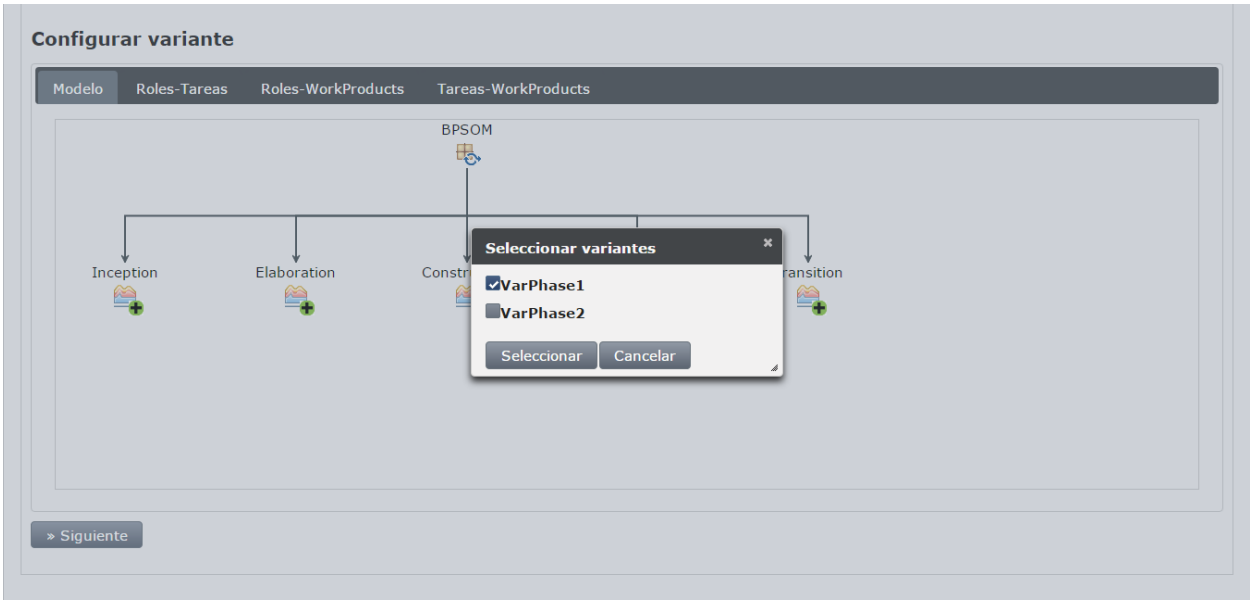


Figura 4.13. Cuadro de selección de variantes

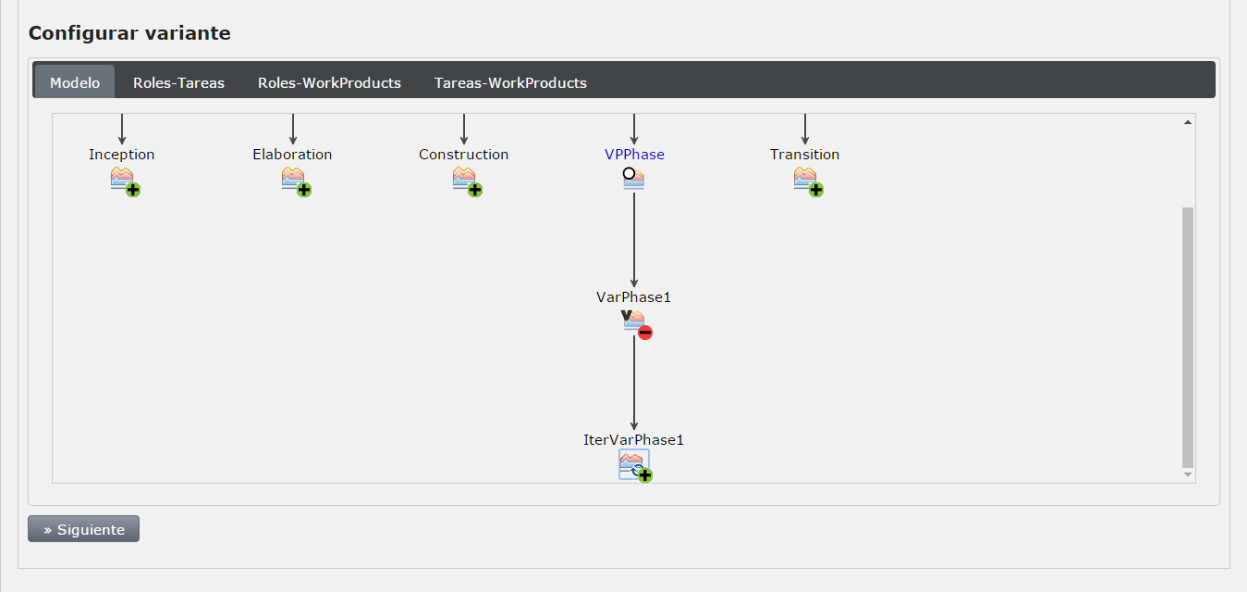


Figura 4.14. Variante seleccionada para el punto de variación

En la Figura 4.14 hemos seleccionado la variante *VarPhase1*, de tipo fase, para el punto de variación *VPPhase*. Dicha fase está compuesta por una iteración

IterVarPhase1 que, a su vez, tiene un punto de variación de actividad que posee la restricción de que no debe quedar vacío, lo que nos obliga a seleccionar una de las variantes posibles. Como muestra la Figura 4.15, hemos seleccionado la variante VarActivity2, que está formada por la tarea *TaskVar2* (no mostrada en la figura).

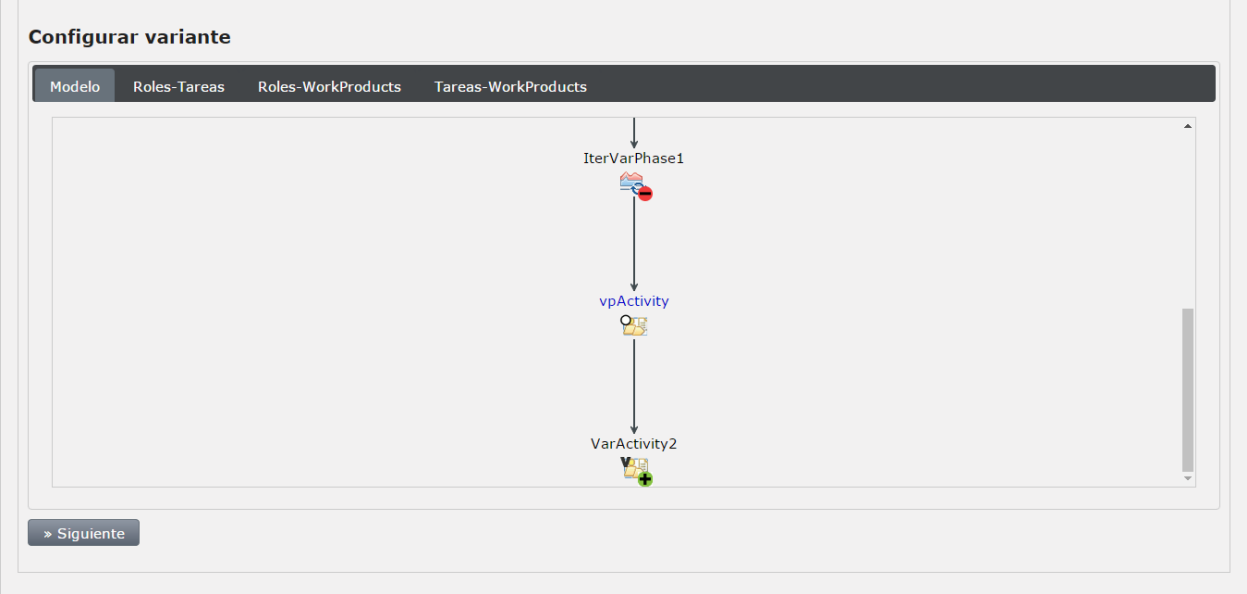


Figura 4.15. Variante seleccionada para *vpActivity*

Esta variante nos obliga ahora a seleccionar el rol que realizará la actividad TaskVar2, para lo que seleccionamos la variante de rol *VarRole1,* como se puede ver en la Figura 4.16.

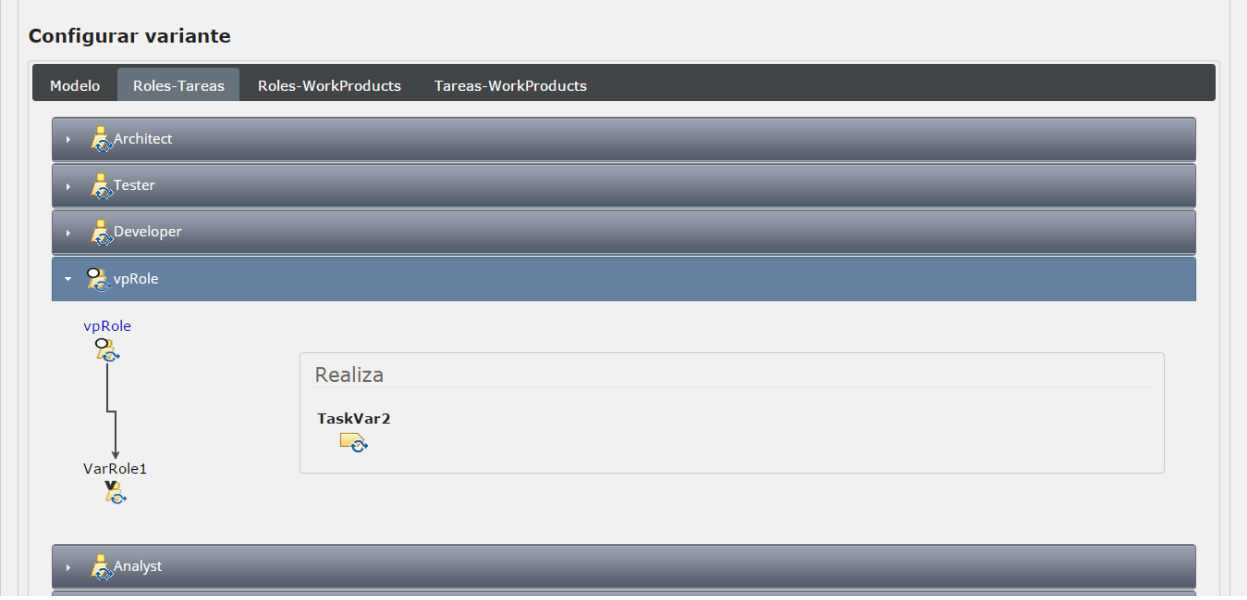


Figura 4.16. Variante seleccionada para *vpRole*

Ahora sí, una vez finalizado el proceso de adaptación del modelo, presionamos 'Siguiente' y vamos al último paso: exportar el modelo. Allí seleccionamos si deseamos exportar al repositorio o no, y presionamos 'Finalizar' (Figuras 4.17 y 4.18). En caso de seleccionar la opción de exportar al repositorio, se nos pedirá ingresar un comentario (con el cual se hará el push en el repositorio github), así como el usuario y contraseña del repositorio.

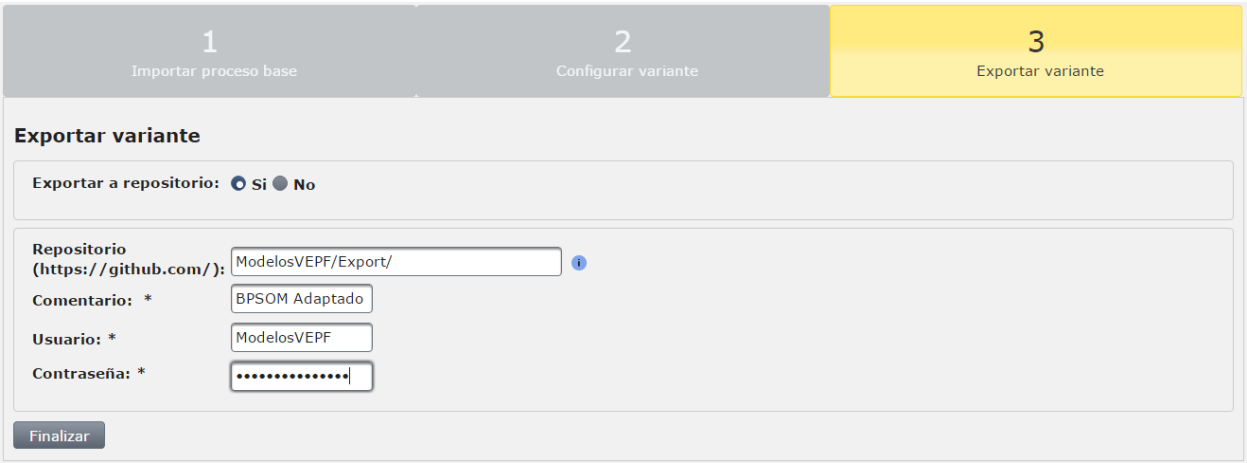


Figura 4.17. Exportar modelo a repositorio

Figura 4.18. Fin exportar

Una vez exportado el modelo, quedará en el directorio local configurado para esto un archivo *.xml* conteniendo toda la información del modelo adaptado, así como una carpeta con el nombre del proceso conteniendo imágenes y otros archivos incluídos en el *xml*. Esto mismo es subido al repositorio, en caso de haberse elegido esta opción.

Ahora, podemos abrir vEPF o bien, EPF Composer, e importar el archivo xml generado por la herramienta. Para ello, primero creamos una nueva librería seleccionando *File*, *New* y *Method Library,* tal como se muestra en la Figura 4.19.

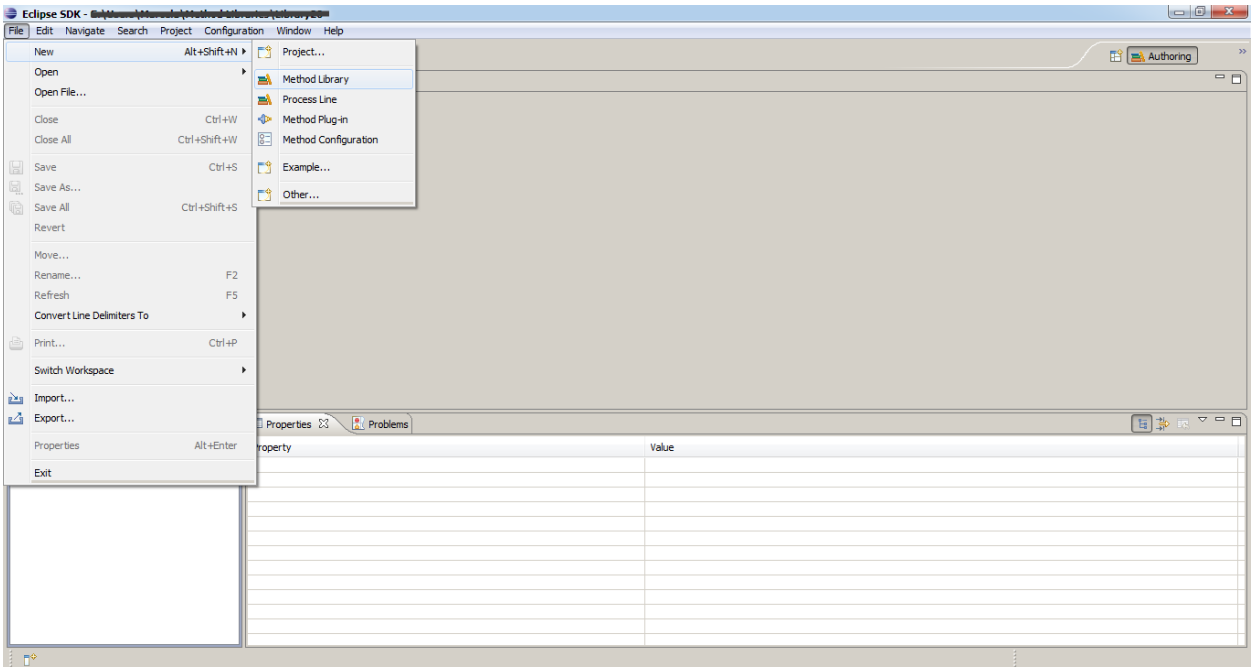
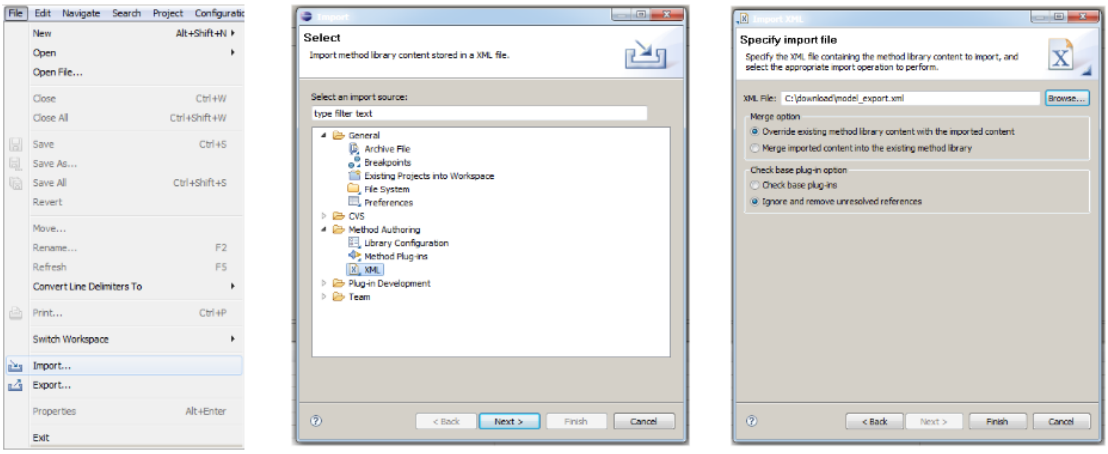


Figura 4.19. Crear nueva librería

A continuación, importamos el archivo xml yendo al menú *File*, *Import* y luego seleccionando la opción XML. En la ventana que aparece a continuación, seleccionamos el directorio en el que se encuentra el archivo a importar y presionamos *Finish* (Figura 4.20).



Paso 1 Paso 2 Paso 3

Figura 4.20. Importar XML

Una vez hecho esto tenemos cargado el modelo adaptado y podemos publicarlo, yendo al menú *Configuration* y luego seleccionando la opción *Publish*, como muestra la Figura 4.21.

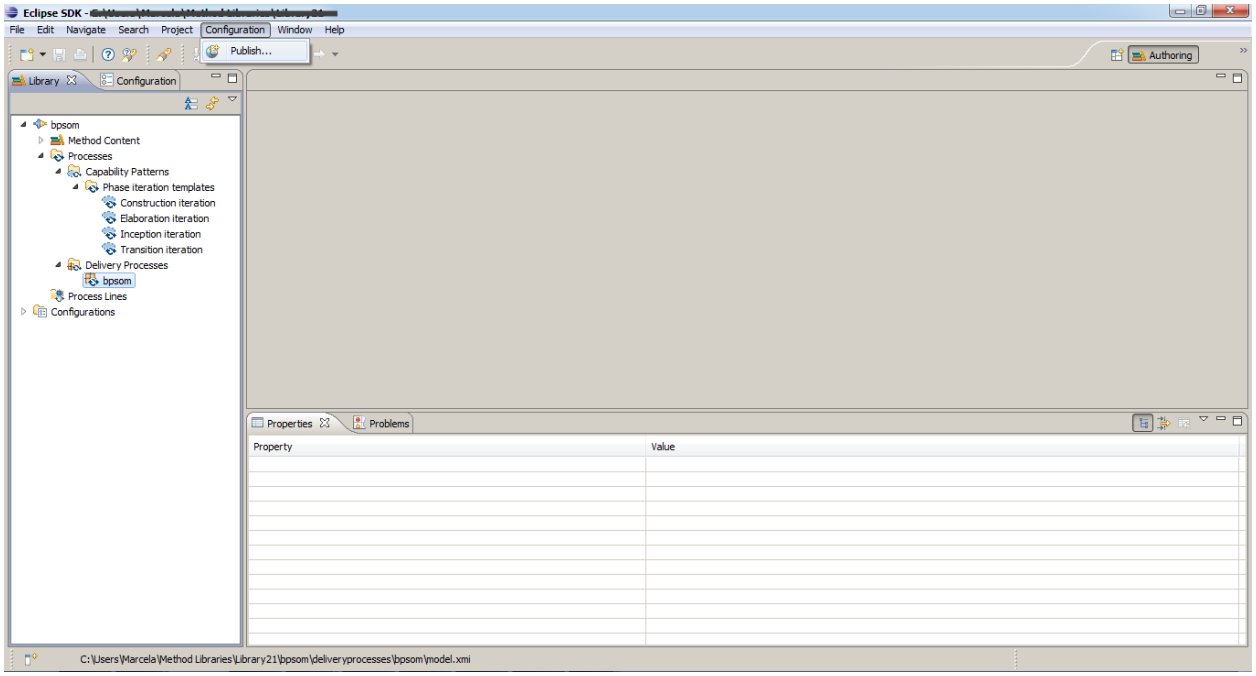


Figura 4.21. Publicar modelo

El resultado de publicar el modelo es el que se muestra en la Figura 4.22. En ella podemos ver que, en la sección *Workflow* aparece la fase *VarPhase1* seleccionada para ocupar el lugar del punto de variación *VPPhase*. Además, podemos ver que en la parte izquierda de la imagen se muestra la actividad *VarActivity2* seleccionada para el punto de variación *VPActivity*.

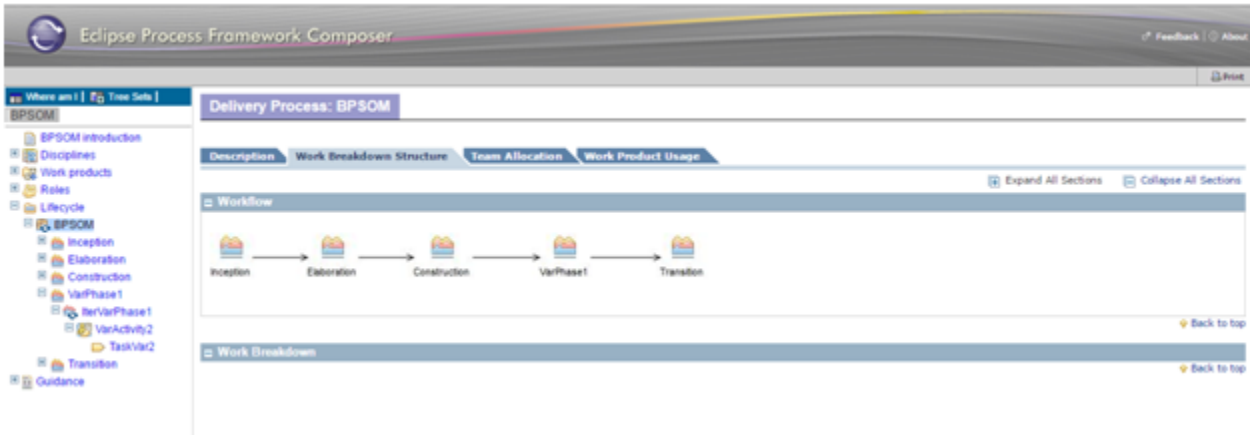


Figura 4.22. Modelo publicado

Capítulo 5

Conclusiones y trabajo a futuro

El objetivo principal de este proyecto fue la investigación de los distintos enfoques existentes para el modelado de la variabilidad en procesos de software. En este contexto se realizó un relevamiento exhaustivo de los mismos, centrándonos principalmente en aquellos que utilizan el lenguaje SPEM 2.0 [OE1]. Para cada uno de ellos se estudiaron las características del enfoque y se analizaron las ventajas y desventajas, así como el soporte tecnológico existente, brindando en cada caso un ejemplo de aplicación.

Dentro de los enfoques estudiados, encontramos que, en cuanto a los elementos sobre los que es posible aplicar la variabilidad no existe una diferencia sustancial entre los distintos enfoques. Por otro lado, si realizamos la comparación respecto a si permiten o no agregar nuevos elementos al proceso base (variabilidad por extensión) encontramos que el único que lo permite es BPMNt. Además no todos cuentan con una herramienta de soporte para el modelado de procesos; los únicos enfoques que tienen actualmente una herramienta de soporte, además de SPEM 2.0, son Modelo de características, vSPEM (aunque no es una herramienta estable), BPMNt, CASPER y CVL. Cabe destacar, además, que tanto CASPER como CVL permiten la derivación automática del modelo adaptado. Por último, comparando qué tan comprensible resulta la metodología, se destacan OVM, SmartySPEM y vSPEM.

En una etapa posterior se decidió, en conjunto con los tutores, y debido a que era el enfoque que cumplía con la mayor parte de las características, que utilizaríamos vSPEM [OE2] y que tomaríamos como base de nuestra implementación los modelos generados a partir de vEPF, herramienta desarrollada para modelar variabilidad con este enfoque. El resultado obtenido fue un prototipo de una aplicación que permite la adaptación de modelos de procesos de software con variabilidad mediante una interfaz web [OE3]. Uno de los aspectos principales a destacar de la aplicación es la facilidad y

rapidez con la que puede obtenerse un nuevo modelo a partir del modelo de proceso general. Otro punto interesante es que al permitir acceder a modelos que están disponibles en diferentes repositorios, hace posible la idea de un grupo de colaboradores que pone a disposición sus modelos de proceso para que otros puedan tomarlos y adaptarlos a sus necesidades.

Finalmente, modificamos y adaptamos un caso de estudio brindado por los tutores, con el fin de mostrar el uso de la herramienta desarrollada [OE4]. Para ello aplicamos variabilidad sobre todos los tipos de elementos considerados por vSPEM, logrando modelar la variabilidad en todos los casos. Dado que el caso de estudio está basado en un modelo de proceso de software real y posee un tamaño relativamente grande, podemos afirmar que el enfoque es aplicable a modelos reales. Sin embargo, una deficiencia encontrada es que no es posible realizar la derivación automática del proceso adaptado. Esta sería una mejora a realizar a fin de perfeccionar el enfoque. Una manera de hacerlo sería utilizando reglas de adaptación de manera similar a como lo hace CASPER, dada la facilidad con la que se plantea en el enfoque.

En cuanto a los conocimientos adquiridos durante la realización del proyecto, podemos mencionar que obtuvimos conocimientos sobre lo que son los procesos de negocio, la variabilidad de procesos y las diferentes formas de tratarla y modelarla. Además aprendimos sobre las herramientas EPF Composer y vEPF. Por otro lado, nos quedó conocimiento también sobre cómo manejar toda la información investigada y cómo realizar la posterior documentación.

Podemos concluir entonces que logramos cumplir con el objetivo principal del proyecto, habiendo logrado los objetivos específicos planteados.

Propuestas y líneas futuras

Queda como trabajo a futuro investigar la posibilidad de aplicar el enfoque vSPEM en procesos de negocio, definiendo una relación entre los elementos de vSPEM y, por ejemplo, BPMN.

Otro aspecto que podría ser considerado a futuro sería incluir mecanismos que permitan definir el contexto en el que se aplicará el modelo de proceo, y a través del mismo poder derivar automáticamente el modelo adaptado.

En cuanto a la herramienta desarrollada, queda como trabajo a futuro el desarrollo de la funcionalidad de publicar el modelo adaptado sin tener que pasar por EPF. Este

punto fue estudiado pero luego de un tiempo de investigación acotado no hallamos la forma de resolverlo.

También quedaría como trabajo a futuro la posibilidad de utilizar otros repositorios, y no quedar restringidos simplemente a GitHub.

En cuanto a aspectos que podrían ser mejorados, uno sería el comportamiento de la aplicación al refrescar el modelo luego de una modificación sobre éste, así como la superposición de algunos elementos al expandir el modelo.

Por último mencionar que, al realizar el export al repositorio, se realiza una copia (*clone*) del mismo, en caso de no haberse hecho antes, para luego recién hacer el *commit* y *push* sobre el mismo. Creemos que sería interesante investigar la posibilidad de realizarlo de una manera más eficiente.

Referencias

[1] M. Weske: BPM: Concepts, Languages, Architectures, Springer, 2007.

[2] D. Cruz, M. Bastarrica, H. Duarte , De Procesos SPEM a Procesos BPMN. Un enfoque basado en MDE, Proceedings of the XVII Ibero-American Conference on Software Engineering (2014)

[3] OMG. SW & Systems Process Engineering Metamodel Specification (SPEM), v2.0, 2008.

[4] E. A. Oliveira, M. G. Pazin, I. M. S. Gimenes, U. Kulesza, F. A. Aleixo, SMartySPEM: A SPEM-Based Approach for Variability Management in Software Process Lines, 14th International Conference, PROFES 2013, Paphos, Cyprus, June 12-14, 2013, Lecture Notes in Computer Science Volume 7983, 2013, pp 169-183.

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

[6] Mikael Svahnberg, Jilles van Gurp, and Jan Bosch. A taxonomy of variability realization techniques. Softw., Pract. Exper., 35(8):705–754, 2005.

[7] M. La Rosa, W. van der Aalst, M. Dumas, F. Milani. Business process variability modeling : A survey. ACM Computing Surveys, 2013.

[8] G. Valenca, C. Alves, V. Alves, N. Niu. A systematic mapping study on business process variability. International Journal of Computer Science & Information Technology (IJCSIT) Vol 5, No 1, 2013

[9] Propuesta de proyecto de grado, grupo COAL, Modelado de la variabilidad en Procesos de Software, abril 2015.

[10] Hammer M, Champy J (1993) Reengineering the Corporation: A Manifesto for Business Revolution. Harper Business.

[11] Davenport TH (1992) Process Innovation – Reengineering Work through Information Technology. Havard Business School Press.

[12] Fuggetta, A. (2000). Software process: a roadmap. In A. Finkelstein (Ed.), The Future of Software Engineering (pp. 25-34). Limerick, Ireland: ACM.

[13] Portela C., Vasconcelos A., Silva A., Sinimbu A., Silva E., Ronny M., Lira W., Oliveira S.: A Comparative Analysis between BPMN and SPEM Modeling Standards in the Software Processes Context. Journal of Software Engineering and Applications, 5(5): 330–339, May 2012.

[14] T. Martínez-Ruiz, F. García, M. Piattini, Towards a SPEM v2.0 Extension to Define Process Lines Variability Mechanisms, Software Engineering Research, Management and Applications, Studies in Computational Intelligence Volume 150, 2008, pp 115-130.

[15] M.A Jabar, Zarei M.B., Sidi F, N.F.M Sani. A Review of Software Product Line Adoption. Journal of Theoretical and Applied Information Technology, Nov 2013

[16] Ruiz, F., & Verdugo, J. (2008). Guía de Uso de SPEM 2.0 con EPF Composer. Ciudad Real: Grupo Alarcos.

[17] Martínez-Ruiz, T., García, F., Piattini, M., Münch, J.: Modelling software process variability: an empirical study. IET Software 5(2), 172–187 (2011a).

[18] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-21, Carnegie Mellon University, Nov. 1990.

[19] K. Pohl, G. B¨ockle, and F. J. van der Linden. Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Nov. 2010.

[20] Alegría, J., Bastarrica, M.: Building software process lines with CASPER. In: International Conference on Software and System Process (ICSSP), pp. 170–179. IEEE, Zurich (2012).

[21] Alegría, J., Bastarrica, M., Quispe, A., Ochoa, S.: An MDE approach to software process tailoring. In: International Conference on Software and Systems Process, pp. 43–52. ACM, New York (2011).

[22] Hurtado, J., Bastarrica, M., Ochoa, S., Simmonds, J.: MDE software process lines in small companies. Journal of Systems and Software 86(5), 1153–1171 (2013).

[23] Rouillé, E., Combemale, B., Barais, O., Touzet, D., & Jézéquel, J. M. (2012, December). Leveraging CVL to manage variability in software process lines. In Software Engineering Conference (APSEC), 2012 19th Asia-Pacific (Vol. 1, pp. 148-157). IEEE.

[24] A. Delgado. Bpsom methodology. http://alarcos.esi.uclm.es/minerva/bpsom/published/, 2010-2011.

Anexo

Anexo 1: Manual de usuario

El presente manual de usuario tiene como finalidad dar a conocer de forma detallada y sencilla la estructura de la aplicación web desarrollada en el marco del Proyecto de Grado. La aplicación fue desarrollada para que el usuario pueda, de forma intuitiva y sin mayor capacitación, realizar la adaptación de modelos de proceso de acuerdo a sus necesidades.

Página Principal

Una vez que el usuario ingresa a la aplicación vemos la página principal (véase Figura 1 Manual).

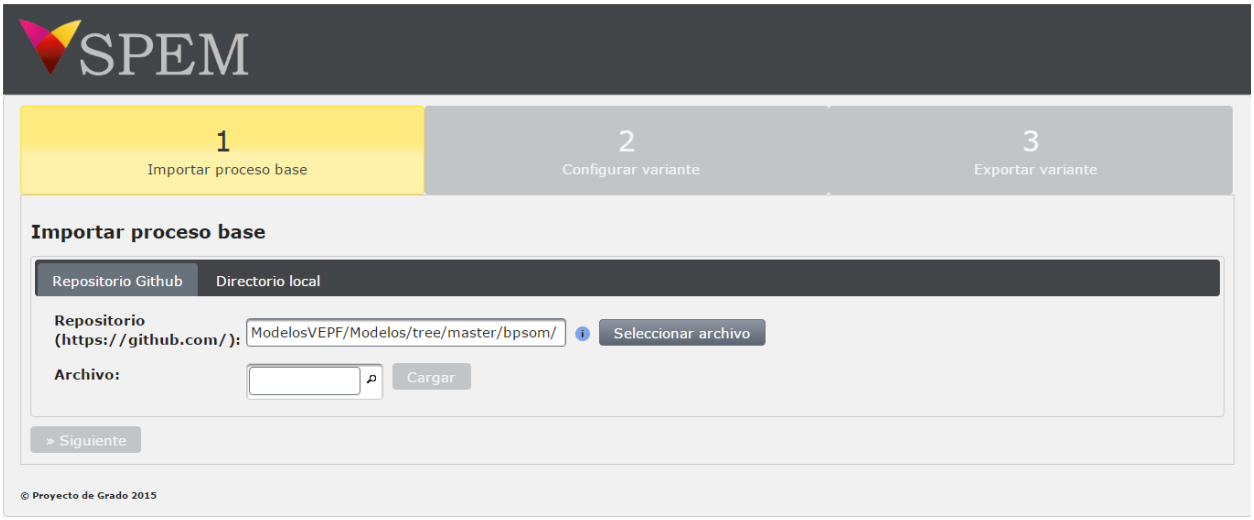


Figura 1 Manual. Página Principal

Dentro de dicha página el usuario sigue una serie de pasos que lo guían en el proceso de adaptación del modelo. Estos se listan a continuación:

Paso 1: Importar proceso base

El usuario puede importar el proceso base a configurar de dos formas posibles:

1) *Repositorio Github*: (véase Figura 2 Manual.) En el campo "Repositorio" colocar la url del repositorio github que contiene el modelo y luego elegir la opción "Seleccionar archivo".

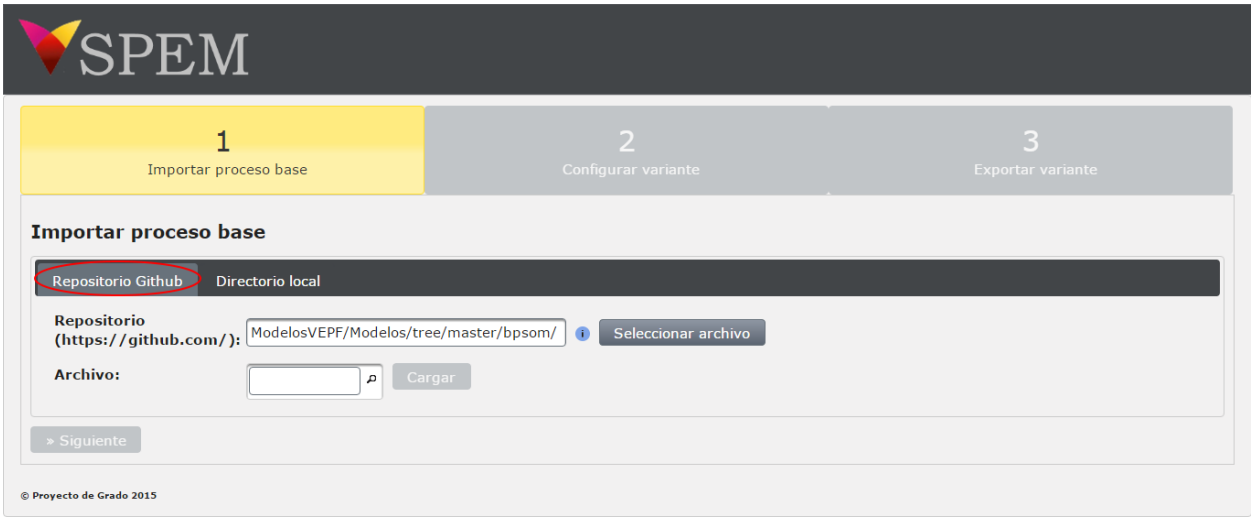


Figura 2 Manual. Importar desde repositorio

En el campo "Archivo" se listan los archivos .xmi contenidos en la url ingresada. Se debe seleccionar el archivo y elegir la opción "Cargar".

En dicho momento se indicará si el modelo pudo ser importado correctamente o no (véase Figura 3 Manual).

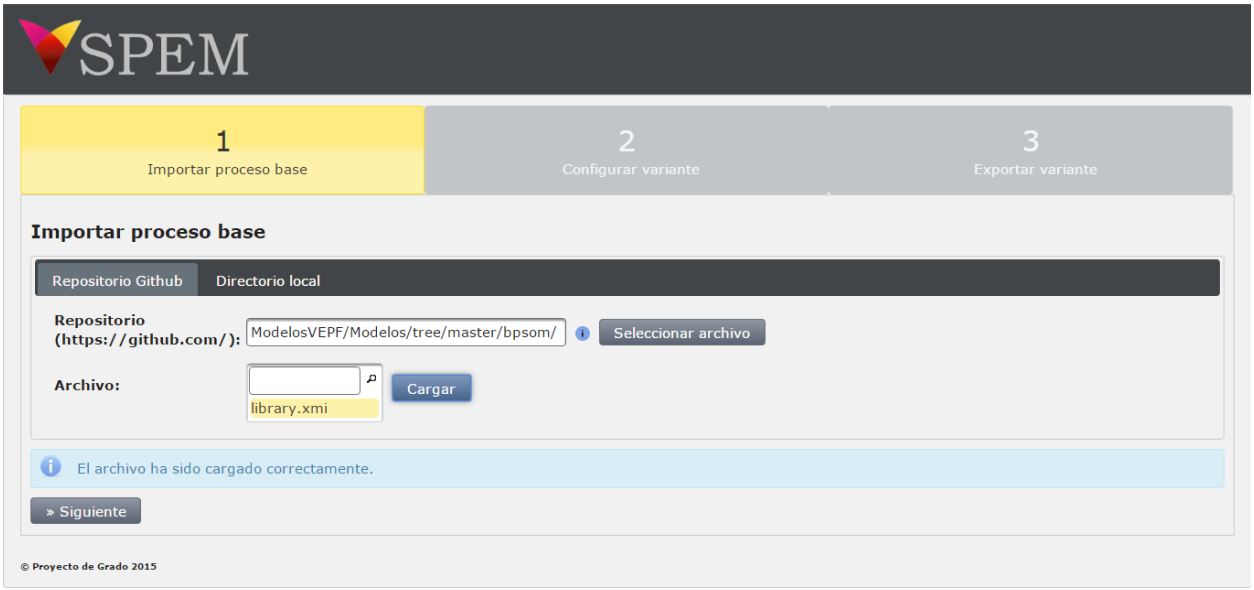


Figura 3 Manual. Finalizar importación

2) *Directorio local*: (véase Figura 4 Manual.) En el campo “Directorio local” ingresar la dirección física en la que se encuentra el modelo a importar y luego elegir la opción “Seleccionar archivo”.

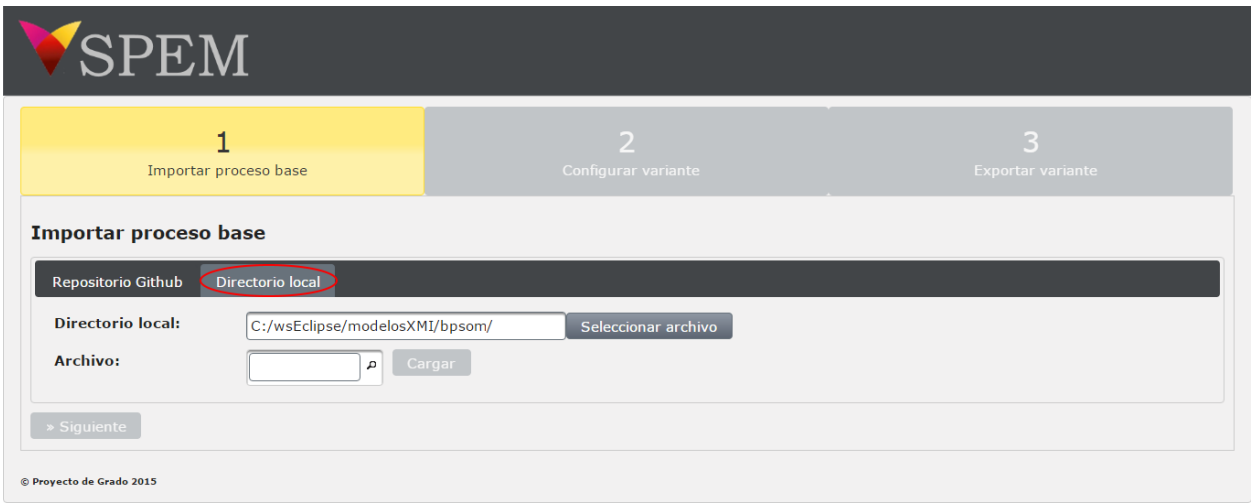


Figura 4 Manual. Importar desde directorio local

Luego repetir los pasos indicados en la opción 1 (Repositorio Github).

Para cualquiera de las dos opciones, una vez finalizada la importación del proceso base presionar “Siguiente”. En ese momento se pasará al paso de configuración de la variante.

Paso 2: Configurar variante

Dentro del paso de configuración de la variante existen cuatro pestañas: “Modelo”, “Roles-Tareas”, “Roles-WorkProducts” y “Tareas-WorkProducts”.

En la pestaña “Modelo” se mostrará una representación gráfica del modelo a adaptar (véase Figura 5 Manual.). Dicho modelo no incluye roles ni workProducts.

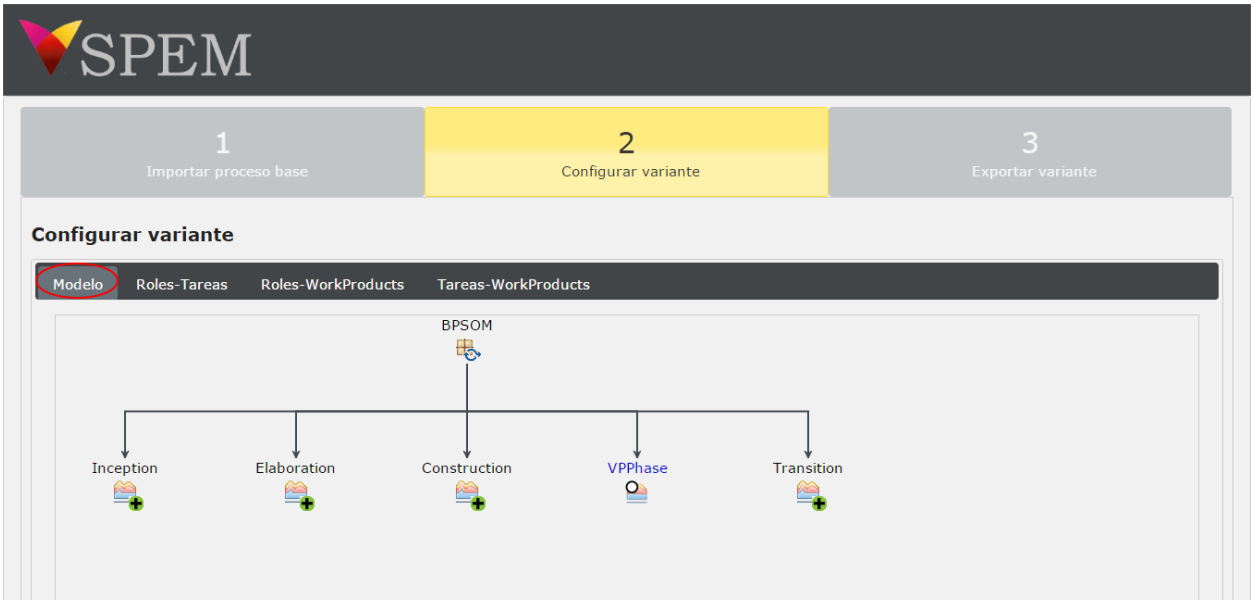


Figura 5 Manual. Representación gráfica del modelo

Mediante los íconos de “+” se puede ir expandiendo el modelo completo. Así mismo, mediante los íconos de “-” se pueden ir ocultando partes del mismo.

Además los elementos que corresponden a puntos de variación, al ser cliqueados, desplegarán las variantes correspondientes y se dará la opción de seleccionar las mismas (véase Figura 6 Manual.).

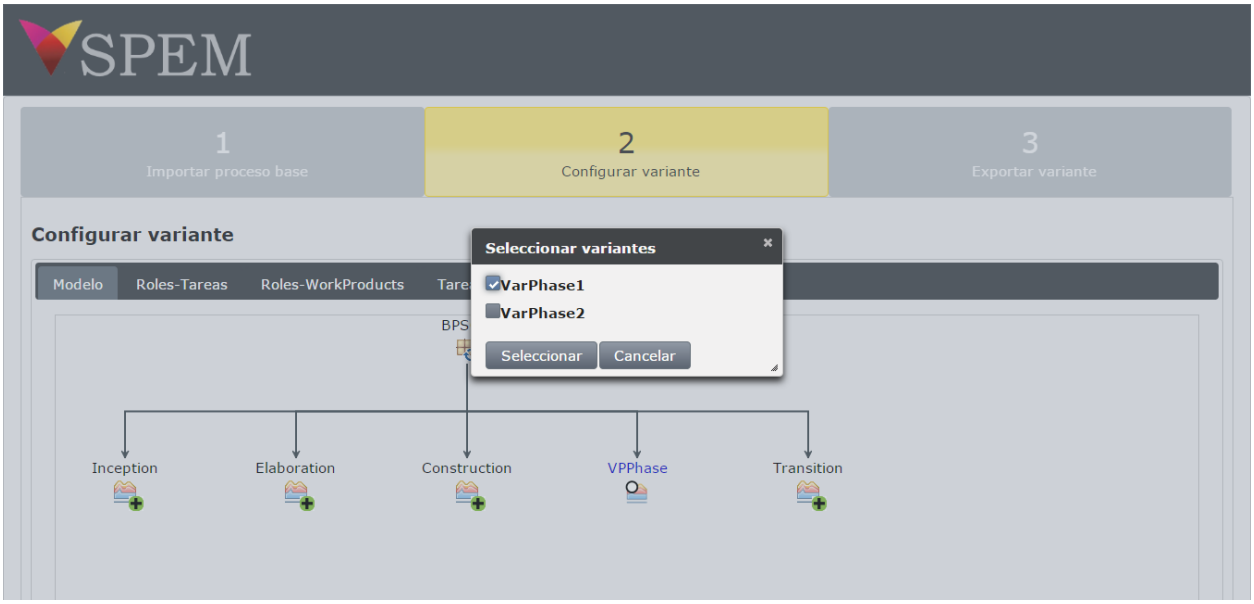


Figura 6 Manual. Selección de variantes

En las pestañas “Roles-Tareas”, “Roles-WorkProducts” y “Tareas-WorkProducts” se pueden ver las relaciones existentes en el modelo para dichos elementos (véase Figura 7 Manual., Figura 8 Manual., Figura 9 Manual.) y realizar la selección de variantes para roles y workProducts que se hayan definido como puntos de variación. Dicha selección se realiza de forma similar a la realizada en la pestaña “Modelo”.

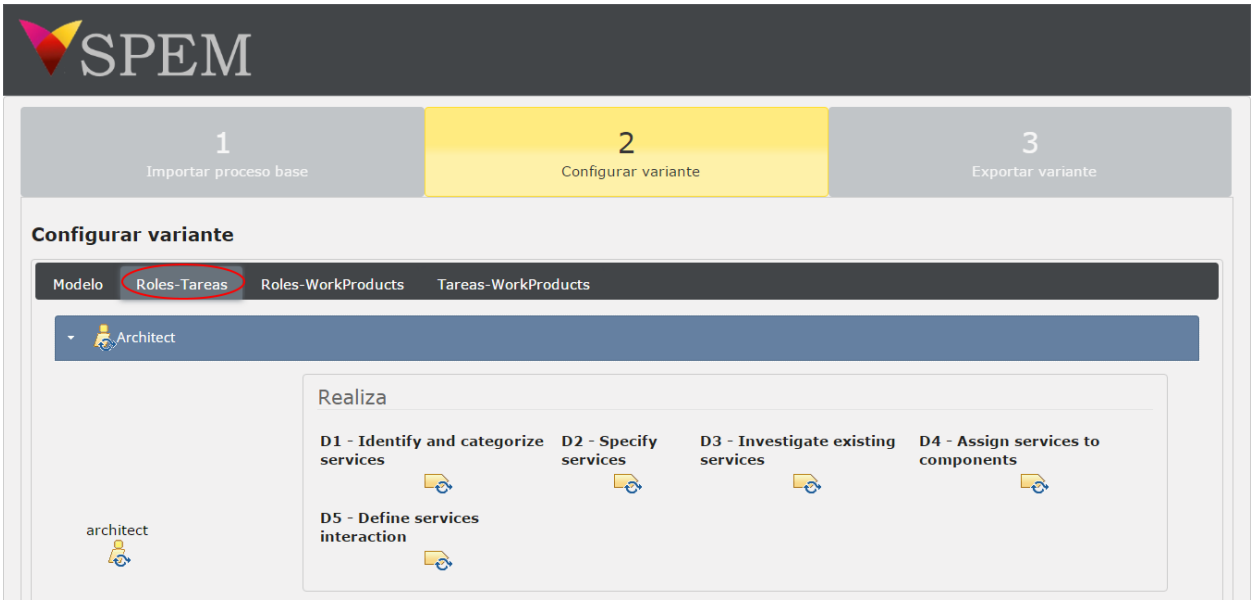


Figura 7 Manual. Modelo Roles-Tareas

Figura 8 Manual. Modelo Roles-WorkProducts

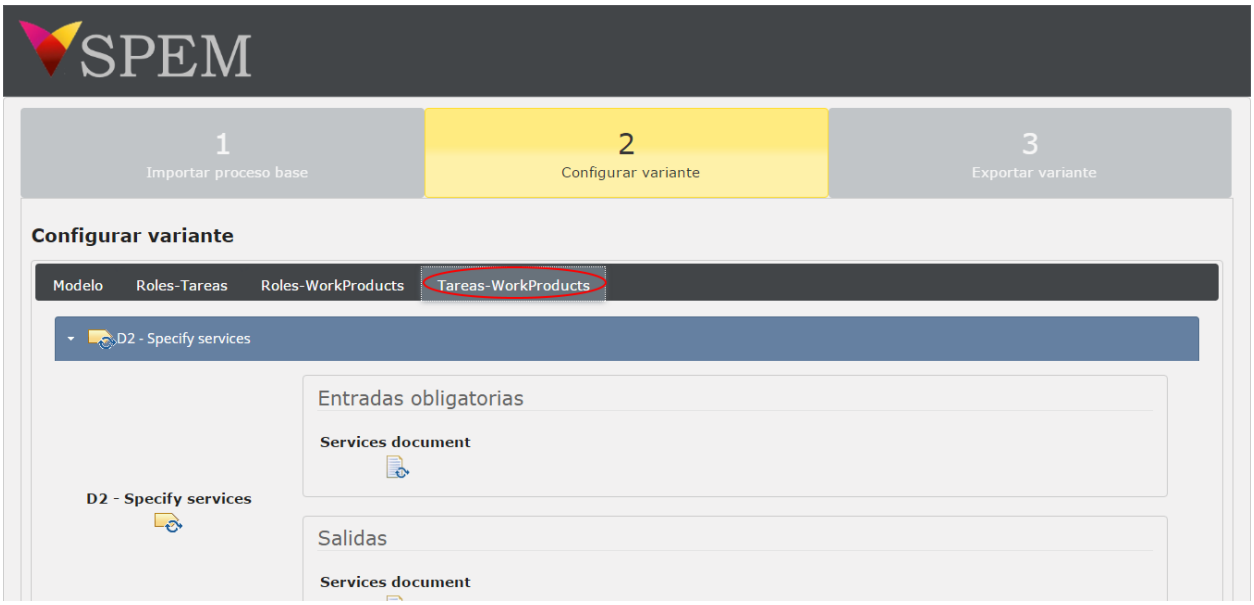


Figura 9 Manual. Modelo Tareas-WorkProducts

Una vez seleccionadas todas las opciones en los puntos de variación existentes en el modelo se debe elegir la opción “Siguiente” y llegar a la etapa de exportación.

Paso 3: Exportar variante

En el paso de exportación se presenta la opción de exportar la variante a un repositorio Github (véase Figura 10 Manual.).

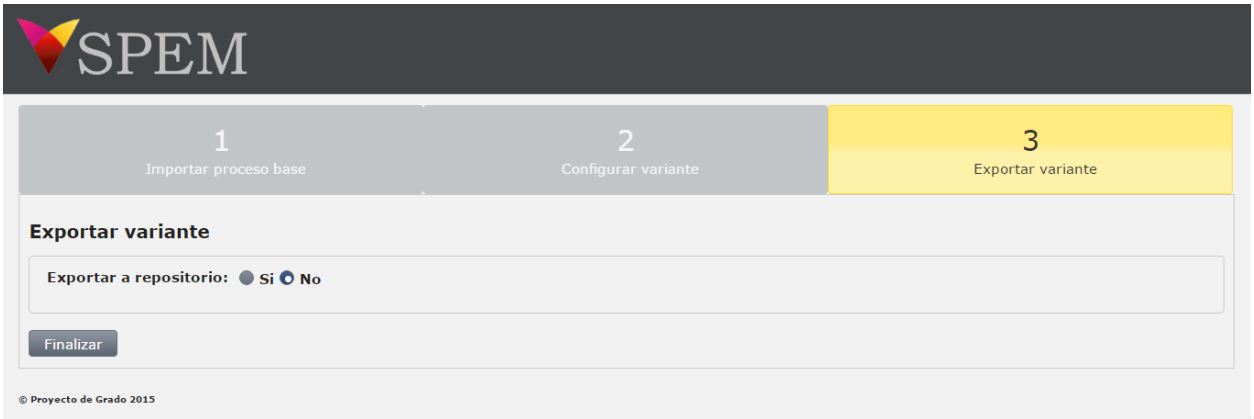


Figura 10 Manual. Opciones de exportación

Si se elige la opción "Exportar a repositorio" se deberán ingresar: la url del repositorio, un comentario para realizar el commit, así como usuario y contraseña correspondientes (Véase Figura 11 Manual.).

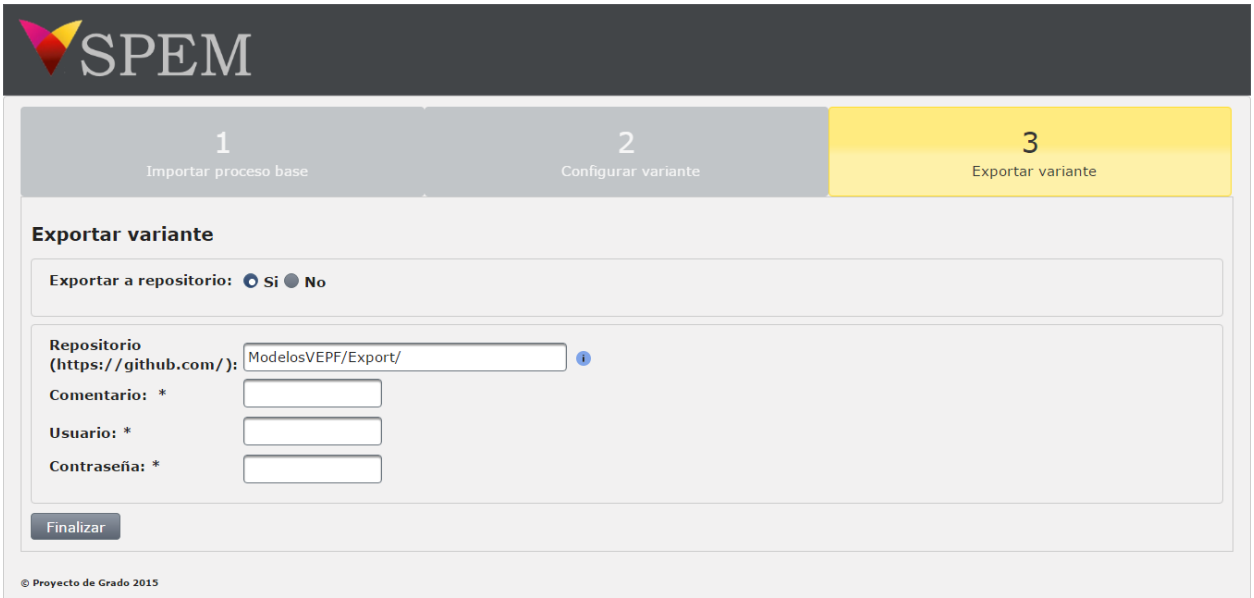


Figura 11 Manual. Ingreso de datos para exportar a repositorio

Para terminar el proceso se debe elegir “Finalizar”. En dicho momento quedará (en el directorio local y, eventualmente en el repositorio también, según la opción elegida) un archivo *xml* conteniendo toda la información de la variante configurada , así como una carpeta con el nombre del proceso conteniendo imágenes y otros archivos incluídos en el *xml*. (véase Figura 12 Manual.)

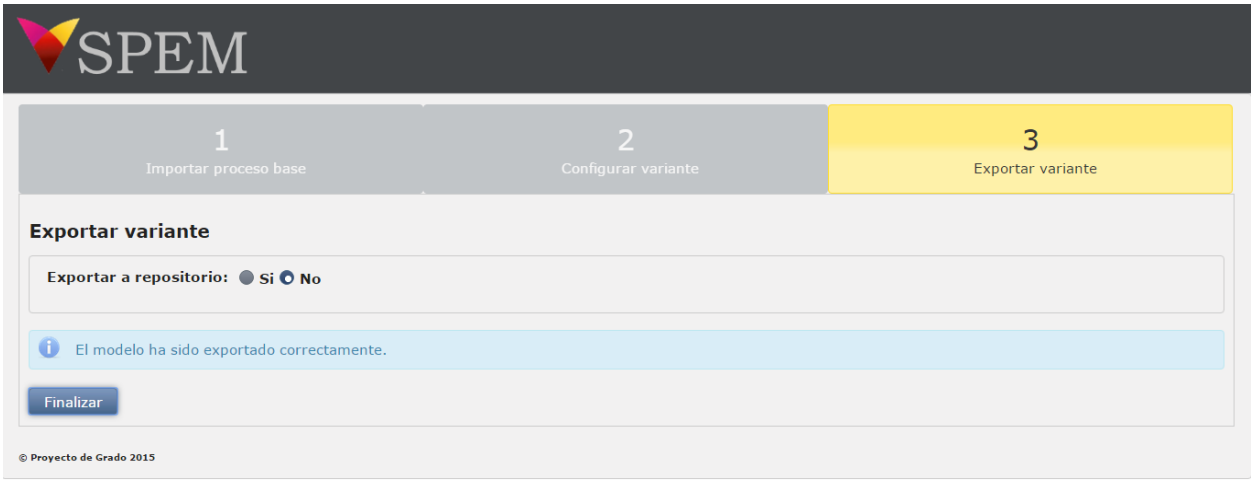


Figura 12 Manual. Finalización de la exportación

Anexo 2: Gestión del Proyecto

En el siguiente anexo se presenta la gestión del proyecto, se detalla la planificación, metodologías utilizadas y cronograma del mismo. Además se muestran las etapas que compusieron el proyecto junto a sus principales tareas.

A2.1. Comunicación

La comunicación para la realización del proyecto consistió básicamente en:

- Reuniones coordinadas con los tutores para información general, cambios de etapas, consultas, demostraciones, entre otras. Dichas reuniones se realizaron principalmente de forma mensual.
- Reuniones de trabajo del equipo de entre dos y tres veces semanales.

- Intercambio de mails con los tutores del proyecto para la coordinación de reuniones, consultas e intercambio de archivos.

Con respecto a las herramientas y tecnologías empleadas para la comunicación, las más utilizadas fueron:

- Documentación en *Google Drive*.
- Chat mediante *Gtalk*.
- Envío de emails.
- Mensaje de texto.
- Utilización de repositorios *Github*.

A2.2. Cronograma planificado

A continuación se detalla el cronograma planificado en base a la propuesta del proyecto:

- Estudio y análisis del estado del arte en procesos de negocio, modelado de la variabilidad de procesos de Software, en particular con lenguaje SPEM 2.0. (2 meses)
    - Producto: Informe del estado del arte para el modelado de la variabilidad en procesos de software.
- Propuesta de selección/adaptación/extensión de modelado de variabilidad en procesos de software. (1 mes)
    - Producto: Informe de propuesta de modelado de variabilidad en procesos de software.
- Implementación de aplicación de soporte a la propuesta. Definición de casos de uso, arquitectura, desarrollo y verificación. (3 meses)
    - Producto: Aplicación desarrollada y documentada.
- Caso de estudio de aplicación de la propuesta de modelado de variabilidad de procesos de software con la aplicación desarrollada. (1 mes)
    - Producto: Informe del caso de estudio realizado.
- Informe final del proyecto de grado organizado por capítulos.(1.5 meses)
    - Producto: Informe final.

Se establece como fecha inicial del proyecto la primera reunión con los tutores, 16 de marzo de 2015. Se calcula una cantidad de días de 260 para la realización del mismo.

A2.3. Cronograma real

A continuación se presenta el cronograma real del desarrollo del proyecto:

- Estudio y análisis del estado del arte en procesos de negocio, modelado de la variabilidad de procesos de software, en particular con lenguaje SPEM 2.0. (4 meses)
    - Producto: Informe del estado del arte para el modelado de la variabilidad en procesos de software.
- Propuesta de selección/adaptación/extensión de modelado de variabilidad en procesos de software. (1 mes)
    - Producto: Informe de propuesta de modelado de variabilidad en procesos de software.
- Implementación de aplicación de soporte a la propuesta. Definición de casos de uso, arquitectura, desarrollo y verificación. (8 meses)
    - Producto: Aplicación desarrollada y documentada.
- Caso de estudio de aplicación de la propuesta de modelado de variabilidad de procesos de software con la aplicación desarrollada. (3 semanas)
    - Producto: Informe del caso de estudio realizado.
- Informe final del proyecto de grado organizado por capítulos. (3 meses)
    - Producto: Informe final.

En la Figura 1 Gestión se interpretan a través de graficas de *Gantt* el cronograma planificado y el cronograma real.

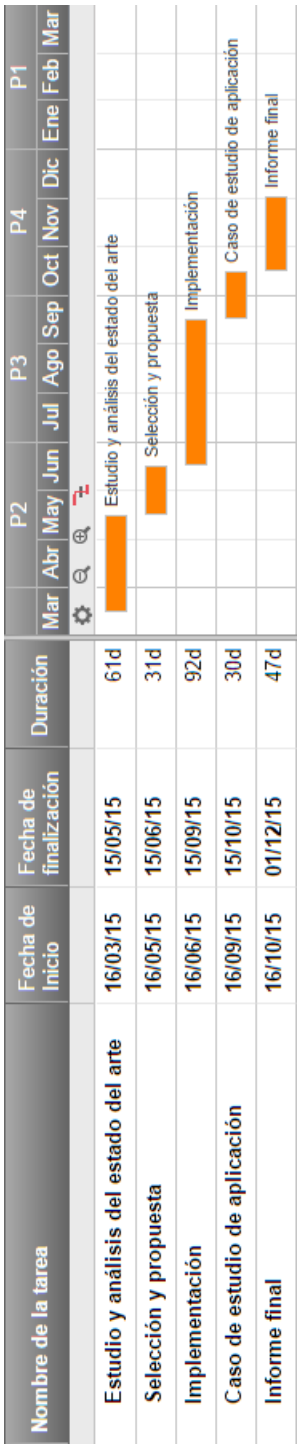


Figura 1 Gestión.

Gantt cronograma planificado **Gantt cronograma real**

A continuación se presenta lo realizado en cada etapa:

- **Estudio y análisis del estado del arte:** la misma se basó en la búsqueda, investigación y clasificación de material sobre SPEM 2.0 y los distintos enfoques existentes sobre la variabilidad en procesos de software. En una primera etapa se revisó la bibliografía recomendada por los tutores. En una segunda etapa se realizaron búsquedas en la herramienta *Timbó* para encontrar artículos científicos relacionados con dichos conceptos y otros que abordaran nuevos enfoques de variabilidad. Además se creó un caso de estudio que fue aplicado a todos los enfoques encontrados. Durante la investigación se fue registrando y documentando todo lo aprendido. Dicha etapa culminó con la entrega de la documentación del estado del arte.
- **Selección y propuesta:** se seleccionó de forma conjunta con los tutores del proyecto el enfoque vSPEM para realizar la implementación de la aplicación web. Se realizó una investigación del uso de la herramienta vEPF para crear modelos de entrada a la aplicación desarrollada.
- **Implementación:** en una primera instancia se realizaron un conjunto de prototipos para la interfaz web y se estudió el manejo de repositorios *Github* con Java, principalmente para descartar posibles complicaciones tecnológicas. Luego se realizó el desarrollo del resto de las funcionalidades explicadas en el Capítulo 3.
- **Caso de estudio de aplicación:** se tomó como base el modelo BPSOM creado en la herramienta EPF por la Dra. Ing. Andrea Delgado. Se realizó una réplica del mismo en la herramienta vEPF y se le agregó variabilidad de forma de cubrir todos los tipos considerados en VSPEM, tal como se explica en el Capítulo 4.
- **Informe final:** en esta etapa se recogió todo lo documentado durante el transcurso del proyecto y se elaboró el informe final. Durante el mismo se formalizó, corrigió y completó el informe, supervisado por los tutores.

Los principales acontecimientos que derivaron en el desfasaje del cronograma planificado y el real son los siguientes:

- La etapa de estudio del estado del arte se vio retrasada debido al desconocimiento por parte de los integrantes del equipo acerca de procesos de negocio y procesos de software. Se decidió, por recomendación de los tutores, concurrir a clases del taller de BPM dictado por ellos para adquirir los conceptos básicos del tema.
- Al comienzo de la etapa de implementación se realizó una investigación para ver la posibilidad de incluir la funcionalidad de publicar el modelo adaptado desde la

herramienta desarrollada, pero tras dos semanas no hallamos la forma de hacerlo. Luego de esto, nos concentramos en ver si era factible actualizar la versión de EPF utilizada en la herramienta vEPF, pero después de aproximadamente dos semanas más de investigación, concluimos que no era posible realizarlo dentro de un tiempo razonable para el alcance del proyecto ya que esta actualización implicaba una gran cantidad de cambios en funciones.

- Se realizó un parate de las actividades referidas al proyecto durante el período del 23/12/2015 al 15/02/2016 por vacaciones y preparación de exámenes.