caSPEM2.0: Extensión de SPEM para Adaptación de Procesos Software basado en el Contexto.

Monografía de Trabajo de Grado para Optar al Título de
Ingeniero de Sistemas

Alberto Ordóñez Idrobo
Jhonn Freddy Ordóñez Rodríguez

Director: PhD. Julio Ariel Hurtado

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Sistemas
Grupo IDIS – Investigación y Desarrollo en Ingeniería de Software
Línea de Investigación en Procesos Software
Popayán, Febrero de 2013

Agradecimientos Jhonn Freddy Ordóñez Rodríguez

*A El **Todopoderoso JEHOVÁ**... por tantas bondades y bendiciones recibidas por parte de él, entre ellas mi madre **Martha** y mi bisabuela **Ismenia**, quienes han sido el motor y eje principal de este logro, nunca se desesperaron e hicieron hasta lo imposible para que continuara con mis estudios, ustedes son el cimiento que me permitió mantenerme firme.*

A mi novia Sandra...quien me acompaño en este camino, siempre creyendo en mí, dándome su amor, apoyo, comprensión y paciencia.

Al Ingeniero Julio Ariel... quien fue nuestra guía en este proyecto, brindándonos su tiempo, experiencia y consejos, "Gracias por traspasar un poco de su luz hacia nosotros."

A los profesores del programa de Ingeniería de Sistemas de la Universidad del Cauca...su experiencia y manera de enseñar han sido determinantes en el momento de recorrer esta senda haciéndola muy interesante.

A mis familiares y amigos... quienes me apoyaron, me animaron e hicieron que este camino fuera más fácil de transitar.

A todos ellos GRACIAS TOTALES!!

Agradecimientos Alberto Ordóñez Idrobo

Como todo cristiano primero agradezco a Dios por darme la sabiduría y la fortaleza para culminar esta fase tan importante en mi vida y por colocar en mi camino a las personas indicadas en el momento indicado.

Agradezco a mis padres, Jesús y Nubia por todo el apoyo brindado durante mi proceso de formación en los momentos fáciles y difíciles. A mi padre Jesús que aunque no pudo ver este logro en vida, tengo Fe que desde el cielo lo está disfrutando tanto como yo. A mi madre Nubia, una mujer como pocas, que entrego su vida al hogar formando a mis hermanos y a mí con virtudes y valores. Gracias papá y mamá.

Indudablemente agradezco la dirección del PhD. Julio Ariel Hurtado, que con su entrega y profesionalismo, guio por el camino apropiado este trabajo y por medio de su motivación, combustible esencial, permitió su desarrollo con gran calidad.

Cada persona que pasa por nuestra vida, deja una imagen, agradezco a todas y cada una de ellas las cuales abrieron las puertas de su amistad, pues de cada una tomo lo mejor de sus virtudes, su buena influencia y la apropio para construir de mi una mejor persona.

Contenido

Tabla de Figuras

Lista de Tablas

1 Introducción

La fuerte competencia entre las empresas y el rápido cambio tecnológico ha llevado a las organizaciones de desarrollo software a la inversión en elementos que les permitan tomar una pequeña ventaja sobre su competencia. En la búsqueda de la calidad y productividad se han dado cuenta que uno de los factores importantes para el éxito es implementar procesos de desarrollo software que les permitan evaluar y mejorar la forma de hacer trabajo, debido a que existe una estrecha relación entre la calidad del proceso software y la calidad del producto obtenido a través de este proceso [1]. Los procesos de desarrollo software establecen qué actividades y en qué secuencia deben ser llevadas a cabo para crear un sistema software, indicando cuáles roles participan en dichas actividades para generar productos de trabajo específicos. Es de notar que no existe un proceso único que se pueda ajustar a todos los proyectos de desarrollo [2], debido a que cada proyecto varía según el dominio de aplicación, el equipo de desarrollo, los riesgos, la naturaleza del cliente, entre otros [3]. Sin embargo, la acción de especificar un proceso de desarrollo para cada proyecto resulta costosa desde la perspectiva del consumo de tiempo y recursos. Adicionalmente, al ser una labor tediosa, es deseable que el proceso de desarrollo definido pueda ser reutilizable y adaptable.

1.1 Motivación

En el contexto del mejoramiento de la calidad de sus procesos, las organizaciones buscan crear o replantear sus procesos de desarrollo software con la idea de que un buen proceso de desarrollo software genera un producto software de calidad, a bajo costo y con poco esfuerzo [4, 5]. Las organizaciones de software medianamente maduras y maduras describen sus procesos utilizando

especificaciones o lenguajes de modelado que representan de manera abstracta los aspectos del ciclo de vida del proceso de desarrollo software con el fin de lograr control, mantenimiento y mejora de su proceso. La comunidad de investigación centrada en procesos, ha respondido a esta necesidad con especificaciones estándar tales como UML [6], SPEM [7] y lenguajes de modelado de procesos como PROMENADE [8], UML4SPM [9] y enfoques como Di Nitto [10] y Chou [11]

A pesar de todos los estudios realizados y modelos propuestos, estos no abarcan todas las necesidades de una organización y mucho menos mecanismos para hacer calzar cada proyecto específico. Cada proyecto tiene particularidades que dependen de su contexto, haciéndolo único e irrepetible. Los lenguajes o enfoques no proveen mecanismos sistemáticos para su adaptación, de manera que las organizaciones aplican los modelos tal cual son estudiados y entendidos. Seguir un proceso general hace pesado el trabajo cuando el proyecto no requiere todos los elementos allí descritos, por lo que un seguimiento riguroso no es muy productivo, incluso si ha sido adecuado para otro proyecto no necesariamente lo va a hacer para el nuevo proyecto [5].

Si una organización desea adquirir ventajas sobre sus competidores podría llevarlo a cabo mediante evaluaciones de tipo CMMI o certificaciones ISO, pero lograrlas requiere que la organización defina bien sus procesos, lo cual consume tiempo y recursos [12], y el compromiso de seguirlos, documentarlos y mejorarlos. Sin embargo, implementaciones que cumplan con estos estándares y modelos no necesariamente cubren las necesidades de la organización y los proyectos [12], máxime cuando no proveen mecanismos de adaptación. También existe el caso de organizaciones que basándose sobre su propia experiencia definen un proceso desde cero por cada proyecto o creando un conjunto de plantillas de proceso por cada contexto potencial, lo cual puede generar más costos tanto en el consumo de recursos como en el mantenimiento de los procesos generados. Por lo anterior surge la necesidad de adaptar un proceso software considerando las características que el proyecto o la organización tengan, de una manera sistemática y con soporte tecnológico, de tal manera que sea posible disminuir el tiempo de adaptación del proceso teniendo en cuenta las necesidades y particularidades de cada proyecto u organización, dejando un registro de las decisiones tomadas en el proceso de adaptación.

1.2 Contexto del problema

Con la aparición de las nuevas tecnologías, el rápido cambio tecnológico y la fuerte competencia, en la industria del software se necesita definir procesos de desarrollo software, sin verlos necesariamente como un medio para obtener una certificación o una evaluación, sino como el primer paso para obtener el conocimiento de una manera explícita de cómo la organización trabaja o proyecta trabajar [13].

Las empresas de software utilizan los procesos definidos en la organización con el fin de sentar una base para medir y detectar fallos en la manera de desarrollar un producto software, además aprovechar la productividad de los desarrolladores generando productos de calidad. Sin embargo definir y especificar estos procesos teniendo en cuenta las particularidades de cada organización o proyecto implica tener personal capacitado con disposición de tiempo y con amplia participación de los procesos software propios de la organización. Además estos procesos deben ser adaptables a las necesidades específicas de cada proyecto, lo que implica un gasto de tiempo, recursos y por consiguiente gastos económicos [4, 14].

1.2.1 Problema

Definir un proceso estándar es una actividad esencial para que las organizaciones puedan alcanzar madurez, sin embargo el proceso más adecuado depende de las características particulares de cada proyecto, normalmente un ingeniero de procesos define un proceso específico para cada proyecto de manera informal o define una guía genérica de adaptación, de esto resulta que la adaptación sea costosa irrepetible y propensa al error. Varias propuestas tienen como reto derivar modelos de procesos específicos a partir de modelos de procesos generales, sin embargo en la mayoría de los casos no se tiene en cuenta información del contexto[1] de una manera formal. La propuesta de Hurtado et al. [12] propone un enfoque orientado al contexto, sistemático, repetible y no depende del ingeniero de procesos al momento de adaptar modelos de procesos a contextos particulares. Debido a que este enfoque es muy reciente e inmaduro, su adopción por parte de la industria resulta muy compleja dado que no se cuenta con un soporte lo suficientemente práctico. En particular, la complejidad se presenta debido a la

[1] Contexto: Características y particularidades que diferencian a cada organización o proyecto haciéndolo únicos.

presencia de dos modelos distintos, uno para definir procesos y otro para definir contextos, y por la dificultad de relacionar los dos modelos y para definir las reglas que encapsulan el conocimiento de adaptación.

-

1.2.2 Pregunta de Investigación

De esta manera se plantea la siguiente pregunta de investigación, "¿Cómo facilitar la aplicación práctica de la adaptación de modelos de proceso basada en el uso de información contextual sin agregar una complejidad significativa al modelo de proceso?".

1.3 Justificación

Cuando las organizaciones de desarrollo software no tienen procesos bien definidos, es necesario que para cada proyecto se defina un nuevo proceso ya sea siguiendo fielmente una metodología de desarrollo común o ajustándola a las características que posee el proyecto. Esto resulta en un gasto innecesario de tiempo y esfuerzo para la organización puesto que no se toma en cuenta su experiencia en proyectos anteriores. Además, la experiencia adquirida por el ingeniero que diseña el proceso no se convierte en un activo organizacional y esto lleva a que si en algún momento el ingeniero se desvincula de la organización, se lleve consigo todo el conocimiento del proceso y su adaptación.

Las herramientas disponibles para la definición de procesos, no proveen mecanismos sistematizados para adaptar los procesos usando información del contexto. Esto conlleva a que la adaptación de procesos se haga de forma manual y por lo tanto sea propensa al error. Además estas herramientas tampoco permiten la materialización del conocimiento adquirido por el ingeniero que se encarga de adaptar los procesos y si las hay son experimentales.

La presente propuesta ofrece soportar la adaptación de procesos de manera sistematizada pretendiendo disminuir el tiempo y esfuerzo invertidos al realizar la adaptación manual mediante extender un lenguaje de metamodelado de procesos ampliamente usado en la industria para definir modelos de proceso software adaptables al contexto.

1.4 Objetivos

1.4.1 Objetivo General

- Proponer una extensión a SPEM 2.0 para soportar la adaptación de modelos de procesos software basada en el contexto.

1.4.2 Objetivos Específicos

- Establecer los mecanismos y estrategias prácticos para la adaptación de procesos software en la industria teniendo en cuenta la información del contexto de aplicación.
- Definir e implementar una extensión al lenguaje estándar de procesos SPEM 2.0 (caSPEM2.0[2]) para incluir los mecanismos de adaptación basadas en el contexto.
- Evaluar caSPEM2.0 en un caso de estudio real en una empresa de software.

1.5 Solución Propuesta

Esta tesis propone extender la especificación de SPEM2.0 (Software & Systems Process Engineering Metamodel Specification) mediante constructos que le permitan al ingeniero de procesos definir procesos de desarrollo software con información contextual, además, que los procesos incluyan la lógica de adaptación del proceso, denominadas reglas de adaptación, para adaptar de forma sistemática los procesos software generales, teniendo en cuenta la información contextual del proyecto o la organización. En este sentido, la propuesta pretende darle importancia a la información contextual que rodea el proyecto u organización, incluyéndola junto a la definición del proceso y además que el conocimiento de la lógica de adaptación, traducido en reglas de adaptación, se incluya dentro de la definición del o los procesos software generales para realizar futuras adaptaciones.

De esta manera, las organizaciones tendrán dentro de sus activos de conocimiento, el esquema general del universo que rodea los proyectos software en forma de características contextuales y sus valores probables. También la

[2]caSPEM: Context-based Adaptation for SPEM 2.0

información materializada de las adaptaciones que realiza el ingeniero de procesos cuando adapta un proceso software general a las características propias del proyecto, que como consecuencia eliminará paulatinamente, a medida que las reglas de adaptación se hagan estables, la dependencia del ingeniero de procesos al momento de adaptar los procesos software. Esto no implica que el ingeniero de procesos no es requerido ya qué él definirá los procesos adaptables, siendo requerido únicamente cuando se re-diseñan los procesos, se requiera el diseño de nuevas reglas o cambiar las existentes en un marco de mejora del proceso. Para evaluar como la propuesta cumple los objetivos refiérase al anexo J (Cumplimiento de Objetivos).

1.6 Organización del Documento

Este documento se encuentra estructurado de la siguiente manera: el capítulo 2 incluye el estado del arte y los trabajos relacionados sobre la adaptación de procesos software y las extensiones de SPEM 2.0 para este propósito. El capítulo 3 presenta caSPSEM2.0 una extensión de SPEM 2.0 para la adaptación de procesos software teniendo en cuenta la información de contexto y la especificación de la lógica de adaptación. El capítulo 4 presenta el Prototipo caSPEMTool, validando la instrumentalización de la especificación de caSPEM2.0. El capítulo 5 se presenta el estudio de caso donde caSPEM2.0 y caSPEMTool son evaluados con grupo de ingenieros de proceso en el modelado de un proceso real de una empresa de software. Finalmente el capítulo 6 presenta las conclusiones, limitaciones y trabajo futuro planteado sobre esta tesis.

.

2 Marco Teórico y Estado del Arte

2.1 Marco Teórico

2.1.1 Procesos Software

Según Humphrey et al. [15] un proceso software es un marco de trabajo técnico y de gestión, establecido para aplicar herramientas, métodos y personas en la ejecución de una tarea de desarrollo de software. El proceso software es un conjunto de actividades realizadas para administrar, desarrollar y mantener sistemas software incluyendo las técnicas para ejecutar las tareas, los actores que realizan las actividades, los roles, sus restricciones y artefactos de uso [16]. Según O'Reagan et al. [17] un proceso de desarrollo software es el proceso utilizado por los Ingenieros de Software para diseñar y desarrollar Software, estos procesos reflejan de manera abstracta como es realizado el trabajo al interior de una organización o equipo, sincronizando la participación, seguimiento y uso de personas, procedimientos y herramientas respectivamente. El proceso software permite a la organización determinar: un plan de actividades, canales de comunicación necesarios para levantar los requerimientos e interactuar con todos los involucrados, con el objetivo de lograr un nivel de madurez que permita obtener como resultado mayor eficiencia y calidad, además de proveer mecanismos para el aprendizaje organizacional [15].

Pinto et al. [18] menciona la necesidad de definir el Proceso Software, como mecanismo para prever resultados, tendencias y características, así como coordinar el trabajo de las personas en proyectos comunes y entrenar el equipo de desarrollo en el aspecto de conocer cuando utilizar las tareas definidas y su ejecución exitosa. El estándar ISO/IEC 12207:2008 [19] se establecen un grupo de procesos para facilitar la comunicación y sincronización entre los involucrados del proyecto a la

hora de adquirir o suministrar productos software durante el ciclo de vida del producto, sin embargo no detalla el ciclo de vida del proceso en términos de métodos o procedimientos requeridos para satisfacer los requerimientos y salidas de un proceso, ni el modo en que son almacenados, dejando a los usuarios del estándar las decisiones sobre estos temas. En el estándar ISO/IEC 15288:2008 [20] se extiende el rango de acción del estándar ISO/IEC 12207:2008 a nivel de ciclo de vida de sistemas, pero no soluciona las limitaciones de su antecesor. El estándar ISO/IEC 15289:2011 [21] mejora los mecanismos en cuanto al contenido y gestión de productos o ítems de información durante el proceso. Los estándares antes mencionados fijan los conceptos para la definición del ciclo de vida de los procesos software y sistemas, poseen un anexo especial para adaptación de procesos, del cómo adaptar el estándar a circunstancias particulares del proyecto y establece un marco general para documentar las razones por las cuales se tomaron esas decisiones de adaptación sobre el proceso general que puede ser de gran utilidad para futuras mejoras de proceso, a diferencia de estos estándares nuestra propuesta ofrece unos constructos en un lenguaje de metamodelado de procesos formal para definir las decisiones de adaptación del proceso mediante relacionar la información del contexto con los elementos del proceso.

2.1.2 Modelos de Procesos Software

Los modelos de procesos software son una representación abstracta del proceso software, la cual provee una visión para el entendimiento y toma de decisiones [16]. Normalmente se usan metamodelos y lenguajes de modelado de procesos (PML) para su representación. El modelado de procesos software sirve como una línea base para la gestión de cambios del proceso permitiendo hacer análisis, facilita la comunicación entre todos los participantes del proyecto, es un soporte para la evaluación de productos, procesos y recursos dentro de la organización, facilita la administración y habilita la adaptación y la instanciación de los procesos software. Frecuentemente los modelos son representaciones análogas a las máquinas de estado donde sus nodos pueden representar actividades, las cuales tienen asociadas roles y productos de trabajo.

El modelado de procesos se puede manejar a diferentes niveles, dependiendo de la persona que se encuentra involucrada con él, es decir, los ejecutores de procesos se interesan en los modelos que representan el cómo, mientras que

administradores del proceso se interesan en el qué, mientras los ingenieros de procesos se interesan además en el porqué.

Dentro de los modelos de procesos de software se encuentran el modelo cascada (Waterfall) el cual es el modelo más clásico y uno de los más conocidos y usados en proyectos con alto control de calidad, éste se enfatiza en la planificación del proyecto en escenarios tempranos y una documentación rigurosa. En el modelo de Desarrollo Iterativo es donde el proyecto se divide en pequeñas partes, permite mostrar resultados más rápidos y obtener retroalimentación valiosa de los usuarios. Cada iteración dentro de este modelo se ejecuta como un mini-proceso cascada siendo más flexible que el anterior. El modelo Evolutivo al igual que el incremental desarrolla un producto en múltiples ciclos, pero a diferencia del incremental produce un modelo más refinado en cada iteración y no es necesario conocer completamente todos los requerimientos al inicio del proyecto. El modelo Espiral combina elementos de los modelos Cascada, Incremental y Evolutivo, siendo la administración de riesgos el elemento clave y en cada iteración identifica problemas con alto nivel de riesgo y desarrolla soluciones a ellos. El Desarrollo Iterativo e Incremental aprovecha las ventajas que brindan los modelos anteriormente mencionados pero se caracteriza por preponderar las personas, las comunicaciones, el trabajo software y la respuesta al cambio, haciendo el proceso de desarrollo más flexible. Es decir, el proceso no sea regulado, reglamentado y micro-gestionado, para que los desarrolladores ejecuten trabajos más efectivos y la documentación sea la estrictamente necesario para una comprensión rápida de los actuales y futuros involucrados del proyecto [18, 22].

2.1.3 El Metamodelo SPEM

El metamodelo SPEM 2.0 [7] es un estándar basado en MOF [23] (Meta-Object Facility) usado para definir procesos de desarrollo de sistemas y software junto con sus componentes. Tiene como objetivo definir un alto rango de métodos y procesos de desarrollo y no pretende ser un lenguaje de modelado de procesos genérico. Además proporciona estructuras de información adicionales que los ingenieros de proceso necesitan cuando modelan procesos con UML [24] o BPMN [25].

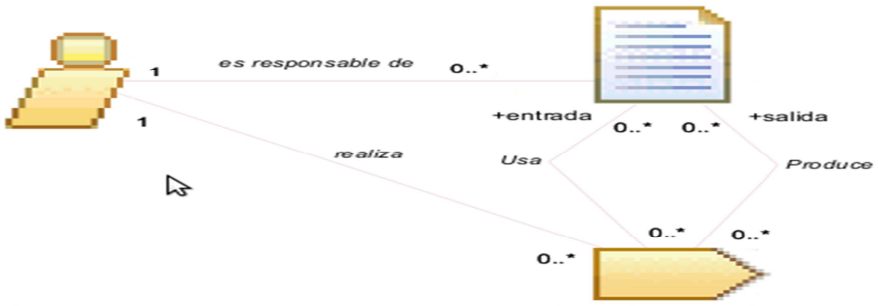

Figura 2.1. Patrón de Trabajo SPEM

El modelo conceptual de SPEM 2.0 está basado en el patrón de trabajo, como se muestra en la figura 2.1: un Rol (una persona o grupo de personas con ciertas habilidades y responsabilidades) lleva a cabo una o varias Tareas (una unidad atómica de trabajo) para obtener un Producto de Trabajo las cuales pueden ser un elemento de entrada, salida o entrada/salida para otras tareas.

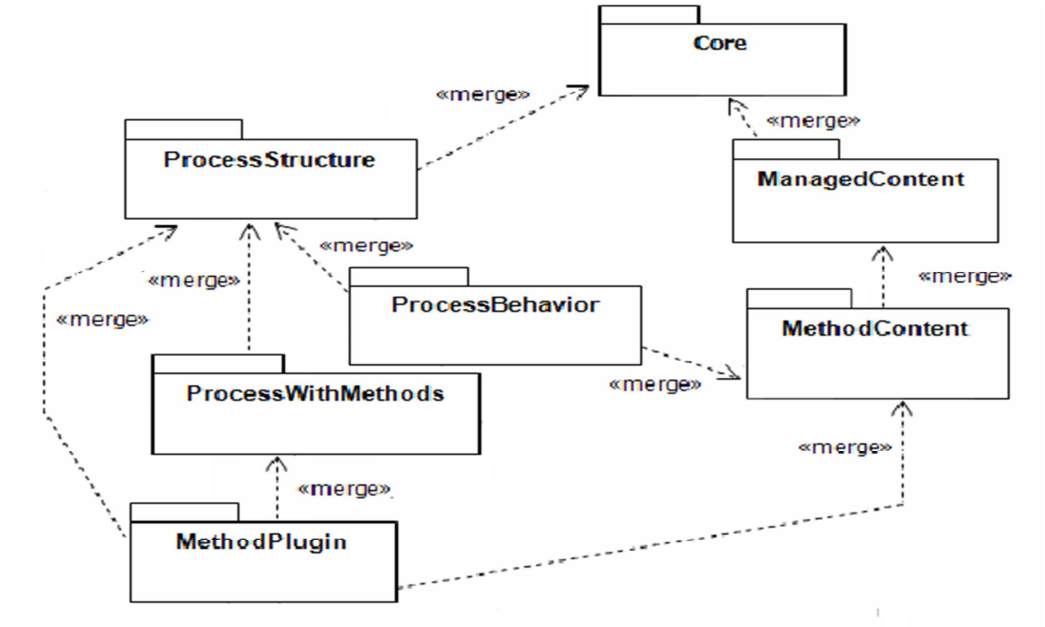

Figura 2.2. Estructura de Paquetes SPEM 2.0. Tomado de [7]

SPEM 2.0 como se muestra en la Figura 2.2, está estructurado en siete paquetes principales las cuales se describen brevemente:

Core: este paquete contiene clases comunes a los demás paquetes del modelo.

Process Structure: este paquete contiene los elementos básicos para definir procesos de desarrollo y como deben ser ejecutados.

Process Behavior: este paquete permite vincular modelos de procesos definidos por terceros (modelos externos, por ejemplo de diagramas de actividad UML) a los elementos del proceso definidos en el paquete "Process Structure".

Managed Content: este paquete contiene las clases encargadas de gestionar las descripciones textuales en lenguaje natural de los procesos y los elementos del contenido de método.

Method Content: este paquete provee los mecanismos básicos para construir la base del conocimiento de los procesos de desarrollo software a usuarios y organizaciones SPEM 2.0.

Process with Methods: este paquete define nuevas estructuras y redefine las existentes para facilitar la integración de los procesos definidos con los conceptos del paquete "Process Structure" a instancias de los conceptos del paquete "Method Content".

Method Plugin: define mecanismos para gestionar todas las librerías de "Method Content" y "Processes", es decir, permite su extensibilidad a través del uso del concepto de variabilidad de tal manera que se puedan crear diferentes variantes de "Method Content y Processes" dependiendo de las necesidades de los usuarios y/o el proyecto, además permite hacer mantenimiento a estos como unidades individuales.

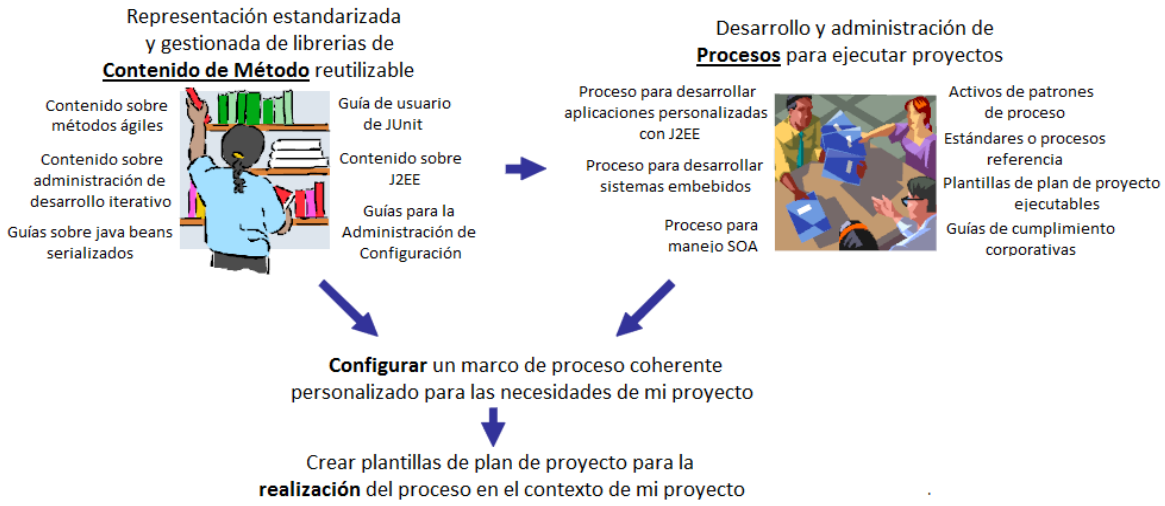

Figura 2.3. Marco de Trabajo Conceptual de SPEM 2.0. Adaptado de SPEM 2.0 [7]

SPEM 2.0 provee una representación estandarizada y una forma de administrar librerías de Contenido de Método reutilizables en donde se permite configurar la base de conocimiento para el desarrollo de software. Los participantes del proceso deben conocer los elementos involucrados para el desarrollo de software, en tareas tales como levantamiento de requerimientos, análisis y diseño, implementación y casos de prueba, administración del proyecto y manejo de cambios, y toda esta información puede ser configurada en un formato estandarizado dentro de la Librería de Método. Ejemplos de contenido de método son tareas, productos de trabajo, roles, documentos técnicos, directrices, fragmentos de método y procesos [7].

SPEM 2.0 da soporte al desarrollo sistemático, administración y crecimiento de procesos de desarrollo, mediante la combinación, reutilización y extensión de los elementos de método definidos previamente, de esta manera es posible instanciar un ciclo de vida de un proceso de desarrollo propio. Este proceso puede ser visto como un flujo de trabajo o una estructura de desglose. SPEM permite a los ingenieros de proceso y jefes de proyecto seleccionar, adaptar y ensamblar procesos para proyectos específicos e incluso utilizar elementos de contenido de método en formas distintas o en distintas fases de un mismo proyecto [7] (véase Figura 2.3).

Conceptualmente un *Method Plugin* representa una unidad de almacenamiento físico para la configuración, modularización, extensión, empaquetamiento y despliegue de contenidos de método y procesos.

La figura 2.4 muestra un ejemplo para un *Method Plugin* llamado "*Rational Unified Process Framework*", en esta se muestra los contenidos del *Plugin* y sus relaciones así como dos extensiones de éste. En éste *Plugin* se ha estructurado en dos paquetes principales separando los elementos de contenido de método "*Method Content*" de los elementos del proceso "*Process*". En el paquete de contenido de método se han alojado jerárquicamente un paquete de contenido de método llamado "*Requirements with Use Case*" el cual aloja elementos de contenido tales como tareas, roles, productos de trabajo y las relaciones entre estos. El paquete de procesos contiene diferentes clases de procesos como patrones de procesos, procesos entregables, fases, etc [7].

Adicionalmente en la figura 2.4 se muestra otros dos *Plugins* que han sido definidos como una extensión al *Plugin* "*RUP Framework"* llamados "*J2EE extensions for RUP*" y "*.NET Extensions for RUP*" los cuales podrían contener información adicional con pasos, tareas, productos de trabajo, directrices,

documentos técnicos, y demás elementos de contenido y proceso para llevar a cabo el proceso RUP teniendo en cuenta las diferentes tecnologías a utilizar [7].

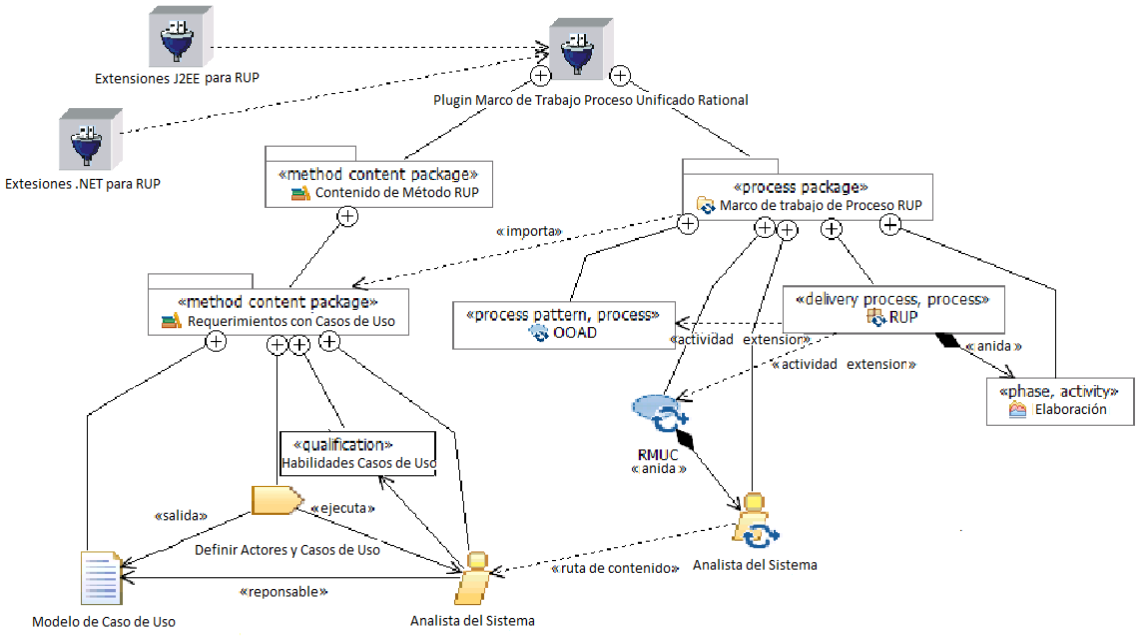

Figura 2.4. Ejemplo de un Method Plugin con sus Relaciones y Extensiones. Adaptado de SPEM 2.0 [7]

2.1.4 Ingeniería de Software Dirigida por Modelos

Según lo expresan en [26] el Desarrollo de Software Dirigido por Modelos es un enfoque donde los modelos abstractos de un sistema software son creados y sistemáticamente transformados a implementaciones concretas. Un metamodelo de software debe tener la suficiente información para construcción de modelos sujetos al dominio de aplicación. La ingeniería Dirigida por Modelos promueve el re-uso ya que en lugar de crear un sistema desde cero se puede especificar los modelos que se van a utilizar en el sistema y de esta manera realizar simulaciones y generar código haciendo sistemas de software completamente usables ayudando a reducir la complejidad y aumentando la eficiencia ya que su generación es más rápida.

El Concepto de Metamodelo

Un metamodelo es un modelo de un lenguaje de modelado que nos permiten representar abstracciones de un dominio en particular, dando significado a los modelos y restringiendo características irrelevantes [27]. Al describir un modelo por medio de metamodelos es posible eliminar la ambigüedad y falta de precisión,

además los metamodelos cuentan con información relevante que permite ser utilizada en el momento de realizar una transformación.

La ingeniería Dirigida por Modelos en la Ingeniería de Procesos

Es el conjunto de acciones para adecuar las definiciones generales y/o particularizar los términos de una descripción general del proceso para obtener un nuevo proceso de aplicación en un entorno alternativo [5], es decir, personalizar un proceso software dependiendo de las necesidades y características de una organización o proyecto. Un proceso bien definido es fundamental para lograr el éxito en la organización y los proyectos, sin embargo no existe un proceso único apropiado para todos los proyectos ni organizaciones.

El modelo CMMI en el área de procesos de gestión de procesos llamada "Definición de Procesos de la organización" propone establecer y mantener un conjunto usable de activos de proceso de la organización y de estándares de entornos de trabajo. Dichos activos de proceso deben ser adaptados para crear procesos adaptados a cada proyecto, incluso escoger el ciclo de vida correspondiente. Dado que un modelo de ciclo de vida que funcionó en un proyecto no necesariamente sea adecuado para otro, la organización debe considerar un conjunto de posibilidades, además de establecer y mantener criterios y guías de adaptación para el conjunto de procesos estándar [28].

-

2.2 Trabajos Relacionados

Las familias de procesos comparten un marco de proceso común, facilitando su reutilización, donde la variabilidad son las características y necesidades que diferencian un proyecto de otro. Dado que el presente proyecto se enfoca en la adaptación sistemática de procesos software y su soporte tecnológico en la industria, en este apartado se presentan las herramientas, enfoques y lenguajes más representativos referentes a la adaptación de procesos software que consideren como factor importante para la aplicación del proceso dentro del proyecto considerando las características del ambiente del proyecto. A continuación se presentan los trabajos más relacionados con nuestra investigación.

2.2.1 Little-JIL

Little-JIL [29] es un lenguaje de programación de procesos enfocado sobre la coordinación de procesos con una sintaxis formal y define rigurosamente semánticas operacionales. El concepto central en Little-Jil es el "paso" (step) el cual es el concepto base para la coordinación del proceso proporcionando un mecanismo de alcance para el control, datos y flujo de excepciones para el agente ejecutor (humano o software) y la asignación de recursos requeridos. Dado que es un lenguaje de alto nivel no está fuertemente ligado a cuestiones propias de algún lenguaje de programación, siendo muy entendible para los no programadores.

Maneja cuatro tipos de paso "non-leaf", los cuales son: "secuential", "parallel", "try", "choice" quienes controlan el flujo del proceso. "Requisites" son mecanismos que aseguran todas las condiciones necesarias para iniciar y terminar un paso de manera satisfactoria. "Exceptions" y "handlers" en Little-Jil son un mecanismo que se encargan de controlar situaciones excepcionales en tiempo de ejecución. "Parameters" es información de comunicación entre pasos. "Resource" es una representación de una entidad requerida para la ejecución de un paso.

Little-jil es un lenguaje de procesos de propósito general, en el que no se planifica explícitamente la variación, no es estándar, no es posible representar el contexto de manera explícita, y por tanto tampoco es considerado en la tarea de adaptación. Sin embargo, ha sido utilizado para representar familias [30, 31] y para el análisis de procesos [32].

2.2.2 Herramientas Composer

RMC[3] y EPFC[4] son soluciones desarrolladas bajo la especificación de metamodelos UMA[5], que define un metamodelo para estructurar el Contenido de Métodos y Procesos [33] usando una evolución de la especificación SPEM 1.1 de la OMG, la cual, gran mayoría de sus partes fue introducida a la especificación SPEM 2.0 [34]. Estas herramientas están dirigidas a ingenieros de procesos y líderes de proyectos, que tienen como objetivo crear, seleccionar, adaptar y ensamblar procesos según sea el proyecto de desarrollo específico; permite editar, configurar, ver y publicar procesos de desarrollo software, donde actúa como un sistema gestor

[3] Rational Method Composer
[4] Eclipse Process Framework Composer
[5] UMA: Unified Method Architecture

de contenido en el cual se almacena, mantiene y publica la base de conocimiento de procesos para los desarrolladores.

RMC y EPFC permiten crear un proceso desde cero o reutilizar material de uno o más procesos existentes; permite seleccionar una configuración o configurar un proceso existente ("tailoring") de acuerdo a las necesidades específicas, agregando, reemplazando o eliminando elementos del proceso; además permiten desarrollar "Method Content" en el cual se podrán definir sus propios roles, tipos de productos de trabajo, métodos de desarrollo y proveer directrices adicionales específicos de la organización.

EPFC es la versión libre de RMC pero no cuenta con todas sus características, se pueden definir activos de procesos mediante librerías principales, pero no se puede planificar la adaptación de los procesos, tampoco tiene en cuenta la definición del contexto y su uso en la adaptación, el reutilización de los activos del proceso es oportunista y la adaptación es reactiva de acuerdo a la clasificación definida por Ambrust et. al. [30].

2.2.3 Enfoques de Adaptación Manual de Procesos

Fontoura et al. [35] proponen un framework denominado PRiMA-F que integra las posibles actividades involucradas dentro de un proyecto software incluyendo acciones preventivas o correctivas para los riegos. Estas actividades son descritas detalladamente clasificadas a un nivel de patrón organizacional o patrón de proceso, que no solo sirve para ser un nodo dentro de la cadena del proceso, sino también, para contrarrestar los riesgos asociados a ellas dentro de un proyecto. Dentro de la organización estos patrones sirven para describir prácticas exitosas en gerencia de proyectos software, son usados para elaborar procesos software y describen acciones para prevenir riesgos. Esta propuesta realiza una instanciación del proceso software adaptado de acuerdo a los riesgos identificados del proyecto. El proceso adaptado se basa de la información descrita en los riesgos, los patrones y las asociaciones entre riesgos y patrones del proceso de desarrollo software general. Como podemos observar esta propuesta si tiene en cuenta el contexto del proyecto que asocian los riesgos, pero el proceso de adaptación del proceso software se realiza de forma manual informalmente mediante la selección de elementos de proceso o patrones de proceso. A diferencia de la propuesta de Fontoura, en nuestra propuesta la adaptación del proceso se realiza de una manera sistemática

mediante operaciones de edición y eliminación de los elementos del proceso que se han relacionado con el contexto del proyecto específico.

2.2.4 Adaptación usando Inteligencia Artificial

Park et al. [4] proponen una técnica para adaptar un proceso basado en el entrenamiento de una red neuronal. Esta propuesta configura un proceso genérico para adaptarlo a un entorno de proyecto dado, reutilizando el conocimiento adquirido por ingenieros de procesos desde su experiencia en la adaptación de procesos, mediante la utilización de un lenguaje formal para expresar reglas de adaptación. Estas reglas definen operaciones de tipo restrictivas y de substracción de elementos del proceso que deciden si un componente del proceso participa o no. A pesar de que esta técnica tiene en cuenta el contexto del proyecto no existe una herramienta basada en un estándar como SPEM 2.0 que la soporte. El beneficio se logra después de que varios procesos han sido adaptados y no considera a los procesos como una familia de procesos. Este enfoque es complementario al nuestro, puesto que las reglas de adaptación pueden ser extraídas de una red neuronal entrenada (si hubiere historial) o viceversa, esta técnica podría aportar un conjunto de reglas iniciales para que el entrenamiento de la red no empiece desde cero.

2.2.5 Adaptación Sistemática de Procesos

Yoon et al. [36] presentan un modelo de proceso usando módulos de proceso encapsulados (fragmentos de proceso) para definir el modelo, cada uno de estos módulos está compuesto de un conjunto finito de actividades y artefactos y las relaciones entre ellos. El proceso de adaptación está regido por cuatro operaciones básicas: adición, borrado, división y fusión las cuales son aplicadas a cada uno de los módulos de proceso mediante una herramienta llamada AAG. Además se verifica la correctitud y cumplimento de los módulos del proceso adaptados por el diseñador del proceso mediante la medición del número de dependencias entre los módulos del proceso, este trabajo ha sido considerado un buen trabajo de adaptación debido a que ha formalizado cuatro operaciones de adaptación sobre procesos software y ha resaltado la necesidad de un enfoque sistemático para la adaptación de procesos [37].

Kang et al. [14] presentan un método para obtener un proceso que se encuentra almacenado en un repositorio de casos, el cual tiene cierta similitud con la

información de un nuevo proyecto, a fin de que al momento de realizar la adaptación el proceso sufra el mínimo número de modificaciones. La acción de adaptación tiene tres tipos: adición, remover y reemplazar. Este método tiene en cuenta el contexto del proyecto, la definición del proceso utilizada en este método la basan en SPEM donde el paso (step) es la unidad de trabajo más pequeña, la cual sirve para calcular la similitud, sin embargo no existe un sistema de gestión de procesos que incluya el método, la proliferación de procesos por mantener es muy alta y la adaptación de procesos no es planificada.

Xu et al. [38] propone un repositorio para almacenar el conocimiento contextual sobre proyectos en el cual su proceso ha sido adaptado, desarrollando una herramienta prototipo que soporta el entendimiento, reutilización y mantenimiento del conocimiento del proceso. Para cada fragmento del proceso se tiene en cuenta información del contexto a través de los nodos contextuales, sin embargo no se basa en un lenguaje estándar debido a que fue pensado para la adaptación de procesos del framework RUP. La adaptación del proceso no es automática, ya que se toman decisiones locales en la adaptación general, por lo cual depende de la decisión que tome el ingeniero de procesos al momento de la adaptación.

2.2.6 FlexEPFC

Martinho et al. [39] proponen una herramienta FlexEPFC la cual es una extensión de Eclipse Process Framework Composer de IBM. Esta propuesta se basa en un enfoque de dos pasos para modelado de flexibilidad[6] controlada en los procesos software. Como primer paso, el ingeniero de procesos senior expresa cuáles, dónde y cómo pueden ser aplicados los cambios sobre los elementos del proceso, como segundo paso, los otros participantes de proceso pueden identificar fácilmente que cambios le son permitidos ejecutar, y actúan de acuerdo a ello. Esta herramienta extiende el UMA con elementos UML flexibility-modelling propuestos por los autores.

El FlexEPFC permite que los ingenieros de procesos puedan otorgar mayor flexibilidad a los procesos, sin embargo, esta propuesta depende del factor humano, es decir, los cambios que el proceso pueda tolerar tienen que ser previstos por un experto, por lo cual no se puede automatizar su derivación. Además aunque esta

[6] Flexibilidad: capacidad de cambiar el proceso por un participante externo, sin llegar a reemplazarlo completamente

propuesta tiene cierta similitud con nuestra propuesta, no tiene en cuenta el contexto del proyecto o la organización de una manera formal, de tal manera que las decisiones tomadas por el ingeniero no serán plasmadas explícitamente dentro del proceso. También al permitir la flexibilidad controlada del proceso, este será visto mas como una guía, dando la opción al participante del proceso de seguirla si lo desea, por lo tanto el proceso general no es adaptado.

2.2.7 Familias de Procesos

La familia de procesos software es un conjunto de procesos software cuyas similitudes representan un núcleo común definido como un proceso de referencia y donde las diferencias representan su variabilidad, satisfaciendo las necesidades especificas de una organización o proyecto en particular [30, 31]. Según Madhavji et al. [40] los beneficios de tener un proceso bien definido son: i) puede ser medido con respecto a otros procesos, ii) puede ayudar al entendimiento y adopción de estándares entre los desarrolladores, iii) puede ser reutilizado, iv) puede simplificar la gestión control y automatización de procesos, v) ayuda a identificar donde puede ser medido el proceso, vi) puede ser la base para mejorar el siguiente nivel de procesos, vii) permite la comunicación efectiva entre los procesos software y viii) soporta el trabajo cooperativo entre humanos. A través de la utilización de procesos bien definidos sería factible la generación de familias de procesos ya que los componentes del proceso bien definido pueden hacer parte del núcleo común de la familia y los componentes no comunes facilitarían la adaptación del proceso en algún contexto específico.

Ocampo et al. [41] proponen un mecanismo de adaptación manual de procesos mediante una técnica de análisis de similitudes entre elementos de dos o más procesos en el mismo dominio, en donde el ingeniero de procesos se encarga de establecer qué elementos de proceso son similares haciendo un análisis de las actividades, productos y roles dentro del proceso. Los autores proponen además una herramienta llamada SPEARSIM que se encarga de ayudar a los ingenieros de procesos en la tarea de adaptación para obtener elementos similares al hacer la comparación de dos procesos mediante reglas especificas, sin embargo en la adaptación de procesos no se tiene en cuenta de manera explícita el contexto en que se aplicará el proceso.

Cockburn et al. [42] proponen unas plantillas para adaptación de procesos, basándose en las metodologías ágiles, teniendo en cuenta la criticidad de cada

proyecto y el tamaño del equipo. Este enfoque está orientado a las personas y sus interacciones más que a los procesos y herramientas. Este enfoque no explicita un punto de partida para el diseño de la metodología que guía el proceso, esta situación está sujeta al resultado de las primeras interacciones entre las personas, que mediante el intercambio oral sincronizan sus acciones para alcanzar los objetivos del proyecto en desarrollo. En cuanto a la documentación se expresa lo escasamente necesario para que la siguiente fase o proyecto inicie, además dependiendo de la criticidad de cada proyecto y el tamaño del equipo es necesario mantener varios procesos que a la larga pueden resultar poco manejables. Lo autoadaptable de esta metodología radica en que entre cada incremento o en su transcurso, mediante talleres de crítica entre los involucrados, la metodología es refinada.

La ingeniería de método situacional (SME) se enfoca sobre el uso formal de componentes (fragmentos) de método, los cuales son bloques de construcción para metodologías de desarrollo de software específicos a cada situación, donde se define cómo diseñar, construir y adaptar métodos, técnicas y herramientas que son necesarias para su desarrollo [43]. Ralyté et al. [44] proponen un modelo de proceso genérico, el cual está soportado en tres técnicas de SME diferentes: (1) Ensamblar fragmentos de Método (2) Extender un método existente (3) Generar un método mediante la abstracción/instanciación de un modelo/metamodelo. Estos enfoques pueden ser aplicados de forma separada o combinados. Además definen un meta-proceso genérico llamado Map el cual es un grafo dirigido donde el nodo es denominado intensión (intention) el cual almacena la noción de la tarea que el ingeniero intenta ejecutar y el enlace entre nodos denominado estrategia (strategy) en el cual se aloja la manera como esa intensión es lograda. De esta manera nuevos fragmentos de método son construidos o adaptados dependiendo de la situación del proyecto que mejor se ajuste. Aharoni et al. [13] presentan un framework que utiliza el enfoque de meta-modelado ADOM, el cual define tres niveles de abstracción (aplicación dominion y lenguaje). La capa de dominio principalmente está basada sobre los cinco aspectos metodológicos de la especificación ISO/IEC 24744, cada entidad de los aspectos metodológicos posee atributos de información relevantes denominados “tailoring info”, los cuales son utilizados en la capa de aplicación para obtener los componentes de método apropiados utilizando diferentes algoritmos de similaridad a fin de crear métodos de desarrollo situacional mediante la combinación de fragmentos, donde se especifica la

situación o rango de situaciones exactos en el cual ese componente de método es adecuado.

Se han realizado algunas investigaciones en empresas reales, a fin de adaptar metodologías de desarrollo ya maduras a los procesos de desarrollo software dentro de la organización. Por ejemplo Bowers et al. [45] y Cao et al. [46] proponen una adaptación de la metodología XP para el desarrollo proyectos de software de gran escala y complejos, ya que esta metodología inicialmente se ha aplicado a proyectos de mediano tamaño. Bowers et al. [45] seleccionan los mejores principios que se aplican en la metodología XP tales como: Pequeñas Entregas, Integración Continua, Programación por Pares, Diseño Simple, Refactorización y Pruebas y las complementa con otras prácticas de metodologías fuertemente formalizadas como por ejemplo RUP. Por otro lado Cao et al. [46] reporta la adaptación de la metodología XP donde además se incorpora un diseño arquitectural "por adelantado" que es la diferencia notable entre las prácticas ágiles para proyectos de gran tamaño y los principios agiles. En el trabajo reportado por en Hanssen et al. [47] se adapta la metodología RUP basado en la experiencia de los participantes (stakeholders) y teniendo en cuenta ciertas características del proyecto en donde se reduce el tamaño de RUP al tipo especifico de proyecto.

Borges et al. [48] proponen, una configuración de RUP no a nivel global del proceso, sino focalizado hacia la obtención de un conjunto adecuado de roles, los cuales se adoptaran mejor en el contexto de equipos de Desarrollo Software pequeños o medianos, sin que se descuide ningún Rol crítico de RUP.

En Rombach et al. [49] basándose en los principios de la ingeniería de líneas de productos software, se propone un enfoque bottom-up aplicado a la gestión sistemática de las líneas (familias) de procesos software. Teniendo en cuenta las similitudes y variabilidades de un número de proyectos software, hace un análisis y modela los procesos en términos de sus semejanzas, permitiendo especificar sus variantes. Rombach propone una organización llamada Línea de Procesos y Productos Software – SPPL, donde los artefactos y procesos son capturados y organizados de acuerdo a discriminadores combinando requisitos de producto y proceso junto con las características del proyecto.

Schnieders et al. [50] soporta la investigación de ingeniería de familia de productos de software orientado al proceso, en otras palabras Ingeniería de Familia de Procesos, ya que el enfoque de Ingeniería de Familia de Productos no son adecuados para software orientado al proceso. Durante el diseño de la familia de

procesos una Arquitectura de Familia de Procesos es desarrollada teniendo en cuenta los requisitos de la familia de procesos, esta arquitectura actúa como referente de los miembros de la familia de procesos y describe la estructura básica para las aplicaciones de la familia de procesos. Aunque no es un enfoque para la adaptación de procesos, este permite administrar la familia de procesos software de una manera sistemática y soportada con arquitectura.

En Barreto et al. [51] se presenta un enfoque para la definición de procesos software basados en la reutilización mediante las líneas de procesos software. Este enfoque describe cuatro características principales tales como: i) Arquitectura de procesos software, ii) Línea de Procesos Software para representar similitudes y variabilidades, iii) La capacidad de permitir la derivación de procesos diferentes, iv) el uso de los componentes del proceso teniendo en cuenta las características de la organización o proyecto. Para soportar este enfoque se propone una herramienta de apoyo a la definición de procesos basado en la reutilización la cual ayuda a capturar las características del proceso y modelar la línea de proceso software. Este enfoque tiene en cuenta el contexto de la organización y/o proyecto al basarse en las necesidades que la línea de procesos debe de satisfacer, provee su propia arquitectura de definición de procesos por lo cual no está basado en un lenguaje estándar, es una herramienta que ayuda a la definición de procesos reutilizables pero su adaptación es manual.

En Hurtado et al. [12] se propone CASPER, un meta-proceso para definir modelos de procesos adaptables al contexto. Dentro de las bases en las que se fundamenta CASPER están las familias de proceso, define un proceso y/o elementos del proceso adecuados para ser aplicados a una situación específica (contexto). CASPER utiliza la técnica de desarrollo software de Ingeniería Dirigida por Modelos - MDE[7] para adaptar procesos organizacionales de una manera más rápida y con poco esfuerzo, basado en el contexto específico del proyecto a desarrollar. La transformación toma los requerimientos generales del proceso, las características del contexto y las reglas que dependen de los valores de las dimensiones del contexto y automáticamente produce un proceso adaptado al contexto. Para validar esta propuesta se desarrollo un prototipo experimental basado sobre SPEM 2.0, en la adaptación se tiene en cuenta el contexto del proyecto pero las reglas de la lógica de adaptación deben ser programadas en el lenguaje declarativo ATL, el cual es muy orientado a los programadores de software por lo cual el ingeniero de procesos

[7] MDE: Model-Driven Engineering por su acrónimo en inglés

debería preocuparse por aprender un lenguaje de programación. Así, la aplicación del enfoque, debido a la complejidad técnica que exhibe lo hace poco práctico para la industria. Esta tesis intenta fortalecer esta propuesta, ocultando cierta complejidad técnica al ingeniero de procesos.

2.2.8 Extensiones a SPEM

Dado que el propósito de este trabajo es una extensión, a continuación se presentará una revisión de extensiones propuestas en algunos trabajos, que pretenden ampliar la arquitectura de SPEM 2.0 de acuerdo a sus propósitos, además de alguna manera enriquecen la expresividad del metamodelo y se convierte en marcos generales a seguir para alcanzar nuestro objetivo.

Bendraou et al. [52] proponen una extensión de SPEM 2.0 denominada xSPEM (eXecutable SPEM) para modelar procesos ejecutables, a fin de tener en cuenta el soporte para su instanciación ("enactment") y se enfoca sobre la validación de procesos de desarrollo software mediante de redes de Petri sincronizadas, esta propuesta surge a raíz de la carencia de términos de ejecutabilidad en la propuesta final de SPEM 2.0, es decir, estructuras que brinden a los modelos de procesos la propiedad de ser ejecutables. La propuesta de Koudri et al. [53] de la misma manera que la de Bedraou propone una extensión a SPEM 2.0 que le permita construir y ejecutar procesos refinando las restricciones y definición de componentes del proceso, sin embargo ambas propuestas no tiene como propósito la adaptación de modelos de proceso.

Martínez-Ruiz et al. [54] proponen una extensión para SPEM 2.0 en el cual se sugiere nuevos mecanismos de variabilidad basados en el uso de puntos de variación y variantes representando la variabilidad necesaria en una familia de procesos. Dado que SPEM 2.0 no provee los mecanismos adecuados para modelar familias de procesos software, pues no permite definir el *"core"* (núcleo) de la familia de procesos, ni permite definir las variantes de esa familia de procesos. Los nuevos mecanismos que se definieron fueron los punto de variación y variantes, las relaciones y dependencias entre estos y las restricciones. Esta extensión está

orientada a definir los constructos específicos para definir las variabilidades de las familias de procesos, sin embargo no aborda la adaptación de procesos como tal.

Martinez-Ruiz et al. [55] proponen una extensión concreta a SPEM denominada vSPEM, en la cual enriquece las estructuras definidas por SPEM para manejar la variabilidad, esto con el objetivo de satisfacer los objetivos y características de la organización y los proyectos. vSPEM agrega un nuevo paquete denominado *"ProcessLineComponent"* en el cual define estructuras abstractas como "*VarPoint*" y *"Variant"* las cuales le dan a los elementos del proceso un comportamiento diferente al inicialmente definido por SPEM. Por ejemplo, si existe en el proceso un punto donde su comportamiento puede variar dependiendo de las características del proyecto, esté punto puede ser definido como Punto de Variación (*VarPoint*) y sus posibles variaciones que puede llegar a tomar se definen como Variante (V*ariant*), indicando en la definición del modelo del proceso los elementos del proceso "Variante" que dadas las características del proyecto remplazarían al "Punto de Variación". Para poder dar este comportamiento los autores ha seleccionado algunos de los elementos SPEM que pueden sufrir este comportamiento, entre ellos están *Activity*, *TaskUse*, *RoleUse*, *Tool* y *WorkProductUse*, y los han extendido en el paquete *ProcessLineComponent* como VP*Activity* y *VActivity*, *VPTask* y *VTask*, *VPRole* y *VRole*, *VPTool* y *VTool, y* VP*WorkP* y *VWorkP* respectivamente. De esta manera permite modelar puntos de variación y sus variantes a nivel de Actividad, Tarea, Role, Herramienta y Producto de Trabajo. Esta extensión mejora sobremanera la definición del modelo del proceso, teniendo en cuenta las características del proyecto *("Contexto"),* pero no menciona como el valor de esas características entrarían a modificar el proceso, lo que indica en conclusión que la adaptación del proceso software sigue siendo manual, solo mejora el hecho de que la variabilidad de un proceso software con vSPEM es modelada en forma más explícita que en SPEM 2.0.

En Martinho et al. [56] se presenta un framework para modelado y aprendizaje de flexibilidad en procesos software, que como contribución, en el metamodelo propone una extensión a SPEM 2.0, donde agrega un sub-metamodelo para la flexibilidad controlada. FlexSPMF provee al ingeniero de proceso el medio para

controlar la flexibilidad que los miembros del equipo pueden aprovechar cuando son guiados por el proceso software. El metamodelo es una extensión no intrusiva de SPEM 2.0 ya que no modifica su estructura original. Dado que su proyecto abarca a FlexEPFC comparte sus virtudes y carencias. Dado que el enfoque no hace uso de información contextualizada éste metamodelo tampoco incluye constructos al respecto, sin embargo es un metamodelo fuente muy útil para la definición del metamodelo que se propondrá en esta tesis.

Ellner et al. [57] propone una extensión a SPEM 2.0 para modelar el ciclo de vida y comportamiento de un proceso a un grano más fino denominada eSPEM, debido a que SPEM no provee un modelo propio para describir el comportamiento más detalladamente, dejando la integración de lenguajes de modelado de comportamiento a quienes utilicen SPEM y sugiere utilizar UML o BPMN para describirlo, porque los conceptos que brinda (conceptos especificados en el paquete "Process Behavior") solo son suficientes para descripciones generales del comportamiento. En cierto modo modelar el comportamiento nos permite ver como el proceso es afectado por el contexto de proyecto, pero la extensión no hace referencia alguna de éste, por tanto el contexto no es tenido en cuenta en su propósito. La extensión eSPEM es una extensión basada en MOF de SPEM y el metamodelo UML, permitiendo soporta instanciación (enactment) automatizada de procesos de desarrollo. Esta propuesta está enfocada a modelar el comportamiento de un proceso software en lugar de la adaptación de procesos.

En Hurtado et al. [12] se propone un metamodelo para la especificación de contextos denominado SPCM – *Software Process Context Metamodel*. Sin embargo este es independiente del lenguaje de procesos y no está implementado en alguna herramienta independiente. La idea detrás de esta tesis consiste en extender SPEM 2.0 con los constructos de SPCM, con el fin de enriquecer los modelos de proceso con información contextualizada que facilite la sistematización de la adaptación.

Kedji et al. [58] proponen una extensión para SPEM para soportar la descripción de la colaboración ad-hoc, es decir, darle a SPEM la posibilidad de manejar conceptos de trabajo colaborativo, y para que los profesionales tengan la habilidad de expresar mediante formalismos y herramientas adecuados la creciente

comprensión acerca de la colaboración, dado que SPEM carece de conceptos para describir tales situaciones y por la poca atención que se le presta al trabajo colaborativo. Debido a que dentro un proyecto, el trabajo colaborativo se presenta momentáneamente en aquellas actividades donde hay involucrados más de un rol y los cuales deben sincronizar su trabajo para cumplir el objetivo común. SPEM puede representar estas situaciones, el problema radica en que la estructura "RolUse" , "WorkProductUse" y "TaskUse" no especifican exactamente quién, qué o cuál, participante, artefacto y tarea real respectivamente, actúa en el proyecto, para ello lo hacen agregando estructuras como "Actor", "ActorSpecificArtifact" y "ActorSpecificTask" , además de otras estructuras para definir su comportamiento. Esta propuesta está enfocada a representar la modelos de procesos colaborativos, no toca aspectos de adaptación de procesos y fundamentalmente extiende los conceptos claves que SPEM ("Rol", "WorkProduct" y "Task") aplicados a procesos colaborativos.

Koudri-Champeau et al. [53] proponen una extensión a SPEM denominada MODAL para soportar la definición y construcción de Procesos Basado en Modelos que nace del enfoque de Ingeniería Basada en Modelos. Esta extensión refuerza la semántica de los modelos de proceso, donde separa las intensiones del proceso de su comprensión, alinea la definición de productos de trabajo al modelo, formaliza las restricciones del negocio o reglas estándar y amplia la definición de componentes de proceso. Los Modelos de Proceso MODAL resaltan la fuerte correlación entre procesos, productos de trabajo, lenguajes y herramientas, reforzando la semántica de los procesos software, abriendo el camino hacia procesos software ejecutables dado que especifica restricciones que pueden ser tomadas como punto de control y verificación revisando estas automáticamente. Esta extensión no aborda la adaptación de procesos, su pretensión es la de enriquecer la semántica de la definición del proceso software, para facilitar su comprensión y contribuye a darle comportamiento ejecutable a las acciones y permite que expresiones de restricción ("*constraints*") puedan ser automáticamente verificadas, dado que en SPEM las acciones y restricciones son informales.

Diaw et al. [59] Proponen una extensión a SPEM 2.0 y al UML 2.2 [60] mediante la agregación de conceptos y semánticas relacionados al modelado e instanciación de procesos software mediante Ingeniería Dirigida por Modelos (MDE).

El objetivo principal de esta propuesta es la utilización de modelos a diferentes niveles de abstracción y las transformaciones entre modelos para construir de una manera semi-automatizada sistemas complejos. Aunque esta propuesta trata sobre la transformación de modelos en ninguna parte se aborda el contexto como un elemento clave para la transformación en los elementos del proceso.

2.3 Mecanismos y Estrategias Prácticas para la Adaptación de Procesos Software

La adaptación como problema fundamental de esta tesis ha sido estudiada por varios investigadores como lo reporta la literatura. En general ha habido un estudio extendido asociado a flexibilizar el proceso de software, sin embargo muy pocos se han dedicado a la formalización del contexto y mucho menos a la reutilización de la lógica de adaptación. Las metodologías Crystal definen un mecanismo basado en la clasificación de proyectos por colores de acuerdo a una caracterización y de acuerdo a ésta se selecciona usar uno de ellos. Enfoques modernos como Incremental Commitment Model definen también mecanismos de selección. Los enfoques de frameworks brindan una variabilidad gigantesca, pero sin un análisis de la misma y sus consecuencias para los proyectos es muy compleja para construir y aún más para reutilizar la lógica de adaptación.

La adaptación basada en casos y en redes neuronales ofrecen mecanismos de aprendizaje en la selección del proceso más cercano al contexto, algunas guías de adaptación y su almacenamiento para la reutilización de procesos y en el mejor de los casos reglas de adaptación. Sin embargo las reglas de adaptación empiezan a ser útiles cuando se han adaptado un gran número de veces, y cuando seguramente el proceso de desarrollo puede haber cambiado significativamente. Los mecanismos más precisos para la construcción de procesos adaptables lo da CASPER, en el sentido que el contexto es modelado formalmente y considerado como una entrada a la lógica de adaptación. La lógica de adaptación reutiliza decisiones de adaptación basada en contextos y su incidencia en la selección de ciertas características en el proceso de desarrollo. Así que esta sección se ha concentrado en definir los dos mecanismos más relevantes: (1) Mecanismos para modelar variabilidad en modelos de proceso SPEM y (2) Lenguajes y métodos para modelar contextos y lógica de adaptación que resuelve la variabilidad de los procesos. Teniendo en cuenta el

estado del arte recopilado en este capítulo se identificaron una serie de estrategias y mecanismos utilizados en el proceso de adaptación de procesos software, algunos de estos mecanismos y estrategias han sido revisados desde la literatura y clasificados como manuales o automatizados (o semi-automatizados).

En SPEM 2.0 [7], al modelar el proceso Software, los elementos que componen una estructura de desglose de trabajo, tienen un atributo denominado "isOptional" el cual puede ser usado como indicación de que el elemento del proceso puede llegar a ser parte o no de la secuencia del proceso. Por medio de este atributo, podemos darle flexibilidad a nuestro proceso, convirtiendo las actividades, tareas o roles en elementos que al momento de la instanciación no son indispensables en el proceso o subproceso. Otro mecanismo que propone SPEM para modelar procesos software adaptables, es mediante el manejo de Variabilidad, en el cual le da a los elementos del tipo MethodContend, Section y Activity, la posibilidad de describir en elementos separados la diferentes adiciones, omisiones y cambios relativos al original. Este mecanismo le permite al ingeniero de procesos agregar o cambiar elementos según las reglas que gobiernan el uso de la variabilidad en el atributo "variabilityType" de los elementos variantes. Cuando se instancie el proceso, el proceso resultante no tendrá variabilidades, dado que este debe ser un proceso transparente (proceso que no refleje ninguna variante, que en un momento dado, implique la toma de decisiones) para que sea ejecutado. La desventaja de este mecanismo es que cada vez que se instancie el proceso ejecutará las mismas adiciones, omisiones y cambios que se modelan en el proceso, no ofrece otro mecanismo para controlar que adiciones, omisiones y cambios se desea ejecutar. Por lo tanto para hacer un uso apropiado de este mecanismo el ingeniero de procesos debe modificar manualmente las variantes satisfaciendo las necesidades del proyecto para el cual adapta el proceso. Otro mecanismos que propone SPEM es materializando las operaciones de adaptación mediante el uso de relaciones entre dos elementos del tipo Activity, mediante los atributos "usedActivity", "useKind" (toma valores posibles de la enumeración "ActivityUseKind") y "nestedBreakdownElement".

En Pillat et al. [61], se puede adaptar el proceso mediante aplicar reutilización de actividad enlazando la actividad reusada en *"usedActivity"* y el tipo de reutilización *"useKind"* con valor *"extend"*. Una vez se configuren estas características la estructura de la actividad que realiza el re-uso cambiará por lo general en el atributo *"nestedBreakdownElement",* las estructuras definidas en el atributo *"nestedBreakdownElement"* deben enlazar a la actividad homónima en la actividad

reusada por medio de el atributo *"usedActivity"* y configurar el atributo *"useKind"* con valores como "*localContribution*" o "*localReplacement*" que hacen adiciones y cambios respectivamente. Para realizar omisiones se hace uso del atributo *"suppressedBreakdownElement"* en la actividad que realiza el reutilización enlazando los elementos que desea omitir de la actividad reusada. Este mecanismo permite la representación e interpretación de elementos de proceso a través de relaciones que pueden ser especificadas cuando se modela el proceso software. El ingeniero puede realizar adaptaciones utilizando este mecanismo modificando manualmente los atributos *"usedActivity"* y *"useKind"*, también debe configurar actividades homónimas en el atributo "*nestedBreakdownElement*" y en estas actividades configurar también los atributos *"usedActivity"* y *"useKind"* indicando qué y tipo de re-uso de estas sub-actividades. La desventaja de este mecanismo es lo complejo que resulta soportarlo, si se realiza una adaptación mediante este mecanismo se debe indicar también en las sub-actividades las sub-actividades homónimas de la actividad reusada y su tipo de re-uso, además si hay omisiones hay que indicarlas también (en *"suppressedBreakdownElement"*), por lo que una regla de adaptación resulta compleja de manejar.

Park et al. [4] adaptan un proceso basado en el entrenamiento de una red neuronal. Esta propuesta configura un proceso genérico para adaptarlo a un entorno de proyecto dado, reutilizando el conocimiento adquirido por ingenieros de procesos desde su experiencia en la adaptación de procesos, mediante la utilización de un lenguaje formal para expresar reglas de adaptación. Yoon et al. [36] presentan un modelo de proceso usando módulos de proceso encapsulados (fragmentos de proceso) para definir el modelo. El proceso de adaptación está regido por cuatro operaciones básicas: adición, borrado, división y fusión las cuales son aplicadas a cada uno de los módulos de proceso.

Kang et al. [14] presentan un método para obtener un proceso que se encuentra almacenado en un repositorio de casos, con similitud con la información de un nuevo proyecto, con el objetivo de obtener un proceso que sufra el mínimo número de modificaciones. La acción de adaptación tiene tres tipos: adición, remover y reemplazar. En esta propuesta se representa el conocimiento, como una regla de adaptación, además al igual que [36], define operaciones (acciones) de adaptación. Xu et al. [38] propone un repositorio para almacenar el conocimiento contextual sobre proyectos en el cual su proceso ha sido adaptado, cuyo prototipo que soporta el entendimiento, reutilización y mantenimiento del conocimiento del proceso. De

CASPER [12] se toma el enfoque de transformación de modelos, con técnicas MDE, y el modelo de contexto de los proyectos.

La tabla 1 presenta una clasificación de los trabajos relacionados basados en la adaptación de procesos software y la información de contexto.

Propuesta	**Modelado SPEM 2.0**	**Modelado Contextual**	**Mecanismos de Adaptación**
Eclipse Fundation et al. [33]	Si	No – La definición del contexto no existe	Adaptación manual mediante mecanismos de variabilidad.
Fontoura et al [35]	No	No – Toma en cuenta la identificación y priorización de los riesgos del proyecto de desarrollo.	Adaptación manual, seleccionando elementos o patrones de proceso directamente asociados a los riegos del proyecto
Park et al.[4]	No	No – utiliza información contextual a modo de parámetros de adaptación	Adaptación semi-automatizada, utilizando reglas de adaptación y redes neuronales.
Kang et al. [14]	Si	No – el contexto es tomado como un factor	Adaptación automatizada, utilizando razonamiento basado en casos, no es planeada, el lenguaje de especificación de reglas es formal.
Xu et al. [38]	No	Si – mediante fragmentos contextuales.	Adaptación manual, herramienta de apoyo a toma de decisiones de adaptación del proceso.
Marthino et al.[56]	Si	No – La definición de contexto del proceso no existe.	No considera la Adaptación, en cambio permite expresar flexibilidad sobre elementos del proceso.
Rombach et al.[49]	No	No – la definición del contexto del proceso es informal.	Enfoque de familia de procesos, mecanismo de reutilización aprovechando similitudes y variabilidades.
Schnieders et al. [50]	No	No – la definición del contexto del proceso no existe	Enfoque de familia de procesos, mecanismo de reutilización aprovechando similitudes y variabilidades.
Barreto et al.[51]	No	No, toma en cuenta las características de la organización objetivo informalmente.	Enfoque de familia de procesos, mecanismo de reutilización aprovechando similitudes y variabilidades.
Hurtado et al. [12]	Si	Si, metamodelo de contexto SPCM.	Adaptación sistemática. Enfoque MDE utilizando

			reglas de transformación de modelos.
Martinez-Ruiz et al. [54]	Si	No – la definición del contexto del proceso no existe.	Enfoque de familia de procesos, mecanismo de reutilización aprovechando similitudes y variabilidades.
Kedji et al. [58]	Si	No	Propone hacer una adaptación manual, su propuesta está enfocada en un meta-proceso para modelar y ejecutar un proceso
Diaw et al. [59]	Si	No	Adaptación sistematizada. Utilizando el enfoque MDE.
Pillat et al [61]	Si	No	Adaptación manual, teniendo en cuenta los mecanismos de variabilidad de elementos del proceso.
caSPEM2.0	Si	Si – Modelado formal	Adaptacion de procesos semi-sistematizada mediante el enfoque MDE. Lenguaje de reglas formal

Tabla 1. Clasificación de trabajos relacionados con caSPEM 2.0

2.4 Síntesis y Discusión

En este capítulo hemos presentado un estado del arte sobre el dominio de la ingeniera de procesos, donde se ha puesto de relieve la necesidad de crear y mantener procesos de software adaptables al contexto, de una manera sistemática y soportada sobre lenguajes de metamodelado ampliamente usados en la industria del software. Este trabajo propone e implementa una extensión a la especificación de SPEM 2.0 mediante la cual se pueda diseñar procesos de desarrollo software incluyendo información de contexto la cual puede ser tenida en cuenta para la adaptación del mismo. La validación de la extensión incluye un prototipo que soporta el modelado de procesos software gráficamente, basado en la especificación del metamodelo SPEM 2.0, para adaptar procesos de una manera sistemática.

3 El Metamodelo caSPEM2.0

3.1 Introducción

A pesar de que SPEM es un lenguaje de meta-modelado para describir procesos, no cuenta con la capacidad de definir y configurar elementos relacionados con las características del contexto de cada proyecto o grupo de proyectos, lo cual hace que el proceso definido no cuente con información valiosa para la adaptación de procesos.

El metamodelo caSPEM2.0 es el resultado de un proceso de abstracción, clasificación y generalización de conceptos que permiten relacionar características del contexto de un proyecto con el proceso, a través de reglas el conocimiento esencial para adaptar procesos definidos por SPEM 2.0. En esta tesis se ha propuesto el metamodelo caSPEM2.0 como una extensión a SPEM 2.0 para ampliar su flexibilidad y de esta manera enriquecer la expresividad del metamodelo, permitiéndole especificar el contexto del proyecto unido a la definición del proceso y así darle la característica de "*adaptable*" desde su diseño en el contexto de la construcción de una familia de procesos.

3.2 Contexto de Uso del Metamodelo caSPEM2.0

El modelado de procesos bien definidos hace factible la generación de familias de procesos, dado que los elementos del proceso común hacen parte del núcleo de la familia y los elementos no comunes reflejarían las variabilidades de miembros de

la familia en algún contexto específico facilitando de esta manera la adaptación del proceso.

CASPER es un meta-proceso para definir modelos de procesos software adaptables al contexto, en donde se utiliza el enfoque de Ingeniería Dirigida por Modelos (MDE), en el cual propone el proceso de adaptación como una transformación de modelos, como se ve en la figura 3.1, en donde las entradas son un modelo de procesos organizacional (*Source Model 1*) general y un modelo de contexto (*Source Model 2*), y la salida es un modelo de proceso software adaptado (*Target Model*), el cual es resultado de aplicar unas reglas de transformación de modelos usando ATL (*Atlas Transformation Language*) sobre el proceso organizacional general.

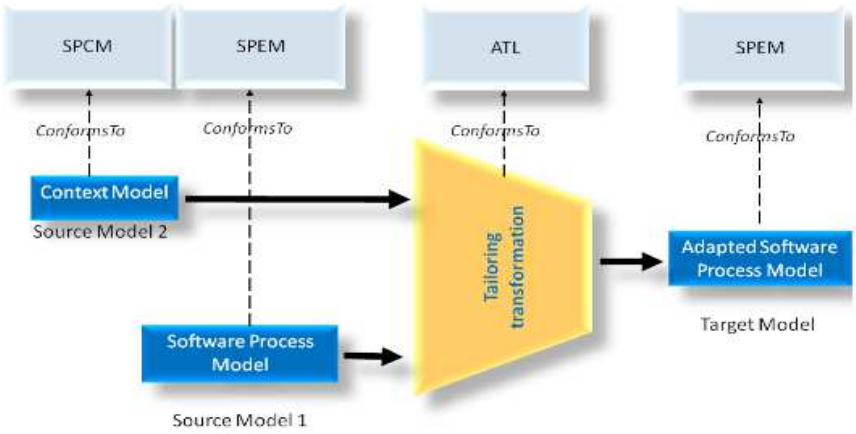

Figura 3.1. Estrategia de Adaptación de Procesos Utilizando el Enfoque CASPER. Tomado de [12]

caSPEM2.0 se basa en el enfoque CASPER y extiende la estrategia de transformación de modelos para hacer la adaptación de un modelo de proceso software previamente definido. caSPEM2.0 define como entrada un único modelo de procesos software enriquecido con la información del universo de contextos asociados a las características generales de la organización y proyectos, la configuración específica única del proyecto para el cual el proceso se adapta y la reglas que dirigen la adaptación del proceso. Esto permite que las reglas sean especificadas de manera gráfica, tomando la metáfora de decisión de los flujos de tareas, donde los nodos de decisión comparan el contexto esperado con la configuración de contexto específica del proyecto, durante la ejecución de la adaptación de los elementos del proceso. De esta manera se logra disminuir la

complejidad de adaptar un proceso software a un proyecto específico, primero, evitando la segregación de modelos que se encuentra en CASPER. Esto se logra debido a que caSPEM2.0 integra dentro en un único modelo, el proceso y la información de contexto; segundo, permite expresar la lógica de adaptación utilizada para trasformar un proceso, y también la integra en el modelo único, por medio de la metáfora de decisión de flujos evitando la necesidad de tener conocimientos de lenguajes de transformación de modelos (véase Figura 3.2).

Con el enfoque CASPER y el metamodelo caSPEM2.0, permite que la presencia del ingeniero de proceso, solo sea necesaria cuando se exprese el proceso y su lógica de adaptación, puesto que esta lógica la construirá el ingeniero de procesos partiendo de su experiencia y quedará especificada dentro del modelo de procesos general, dándole la posibilidad de que cualquier participante del proyecto que configure el contexto específico del proyecto, pueda adaptar de manera rápida el proceso al proyecto. Al tener especificada la lógica de adaptación del proceso dentro del modelo de procesos en general, se evita que se pierda el conocimiento utilizado por el ingeniero de procesos, en cada proyecto al momento de generar el proceso específico, haciendo este proceso sistemático y repetible.

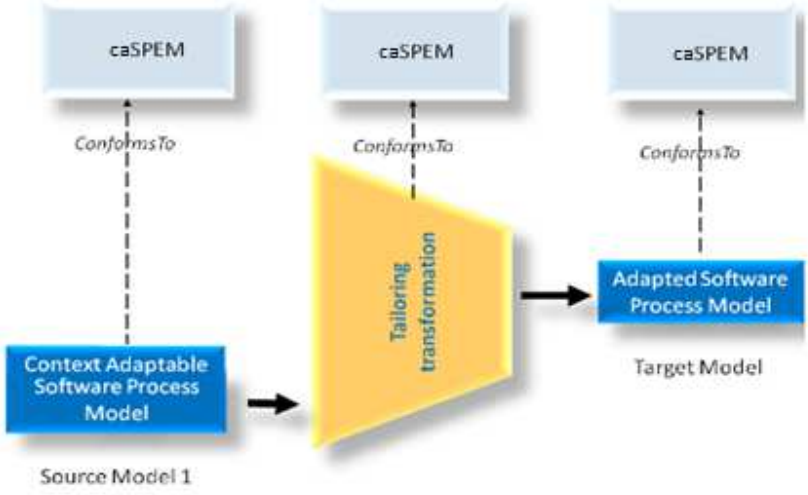

Figura 3.2. Estrategia de Adaptación de Procesos Utilizando el metamodelo caSPEM2.0

3.3 Arquitectura del Metamodelo caSPEM2.0

El metamodelo caSPEM2.0 al ser extensión de SPEM 2.0 extiende su arquitectura agregando dos paquetes, el paquete *Method Context* y el paquete *Rule Knowledge* y extendiendo en paquete Method Plugin, como se muestra en la Figura 3.3. El paquete del metamodelo *Method Context* contiene elementos estructurales para definir todas las circunstancias posibles del proyecto. Este modelo ha sido adaptado de la propuesta de Hurtado et al. [12]. El paquete Rule Knowledge define estructuras para especificar el conocimiento de la adaptación del proceso, relacionando elementos del proceso con elementos de contexto mediante reglas de condición simple.

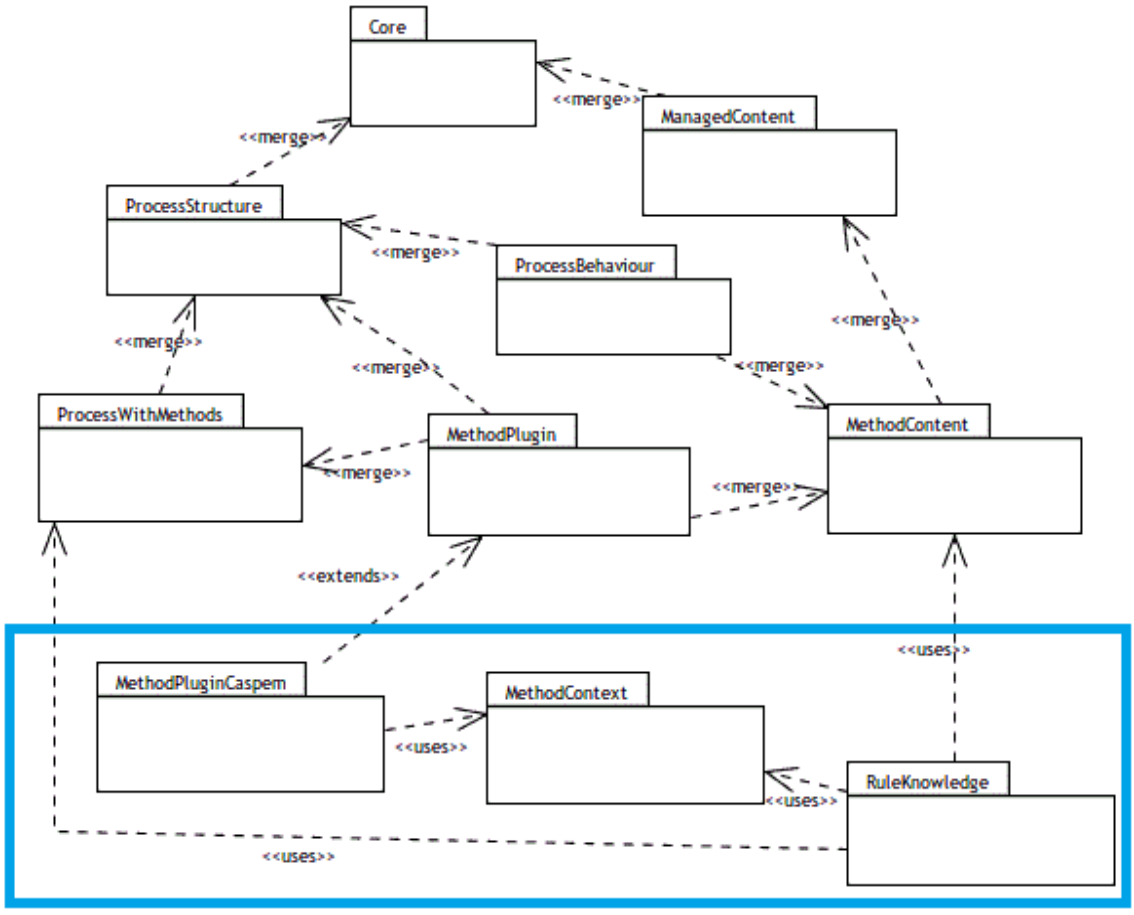

Figura 3.3. Paquetes del Metamodelo caSPEM2.0

3.4 Descripción del Metamodelo caSPEM2.0

3.4.1 Method Plugin caSPEM Package

El paquete Method Plugin de SPEM 2.0 cuenta con los constructos que permite diseñar, mantener y reutilizar las librerías o repositorios de Contenido de Método y Procesos de gran escala, permitiendo que los autores del proceso puedan seleccionar las capacidades del proceso en las que ellos estén interesados por medio de las **configuraciones de método**. Dentro de este paquete se ha definido una estructura llamada "*Method Plugin*" que representa un contenedor físico para *Method Content* y *Process*. En esta tesis *Method Plugin* del paquete Method Plugin de SPEM 2.0 fue extendida mediante el paquete Method Plugin Caspem y la clase Method Plugin Caspem agregando dos propiedades de asociación llamadas *ownedMethodContext* y *knowledgeRule*, de tipo *Method Context* y *Knowledge Package* respectivamente (véase Figura 3.4). La composición "*ownedMethodContext*" es una asociación de composición que relaciona cero o muchos *Method Context* a una librería de método enlazando la definición y configuración del contexto del proyecto con el proceso, y de esta forma poder representar de manera explícita "qué" situaciones específicas definen los proyectos. "*knowledgeRule*" es una asociación de composición que relaciona cero o muchos *Knowledge Package* a una librería de método enlazando la definición del conocimiento, representado en reglas de condición, para la adaptación del proceso teniendo en cuenta la configuración del contexto del proyecto. La multiplicidad de estas relaciones, en ambos casos, se especificó de cero a muchos para evitar incompatibilidades con procesos ya especificados en SPEM 2.0 y facilitar la integración con nuestra extensión. Visto desde la perspectiva del proceso de adaptación, semánticamente "*knowledgeRule*" representa el "como" de la adaptación sistemática del proceso dado que gráficamente se ve como un grafo dirigido, que puede ser recorrido por una máquina de estados y por tanto ejecutado automáticamente.

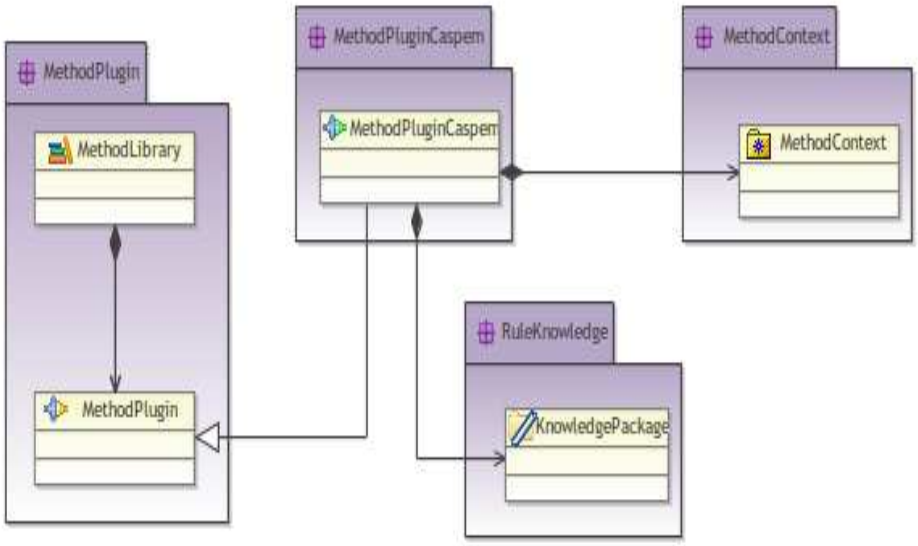

Figura 3.4. Extensión a SPEM 2.0

3.4.2 Method Context Package

El paquete *Method Context* contiene los constructos básicos para modelar contextos de adaptación. Un contexto de adaptación corresponde a un conjunto de características (*Context Attribute*) y sus posibles valores (*Context Attribute Value*). Estas características son organizadas por categorías de variables de contexto (*Dimensions*) y son usadas para definir situaciones específicas a través de una configuración de cada uno de los atributos (*Context Configuration*).

En concreto el paquete Method Context define las clases Context Element, Method Context, Context Package, Dimension, Context Attribute, Context Attribute Value, Configuration Context Element, Context Configuration, Context Attribute Configuration, y extiende la clase Method Library, las cuales han sido incorporados del metamodelo SPCM del enfoque CASPER [12] como se muestran en la figura 3.5 y que serán explicadas a continuación.

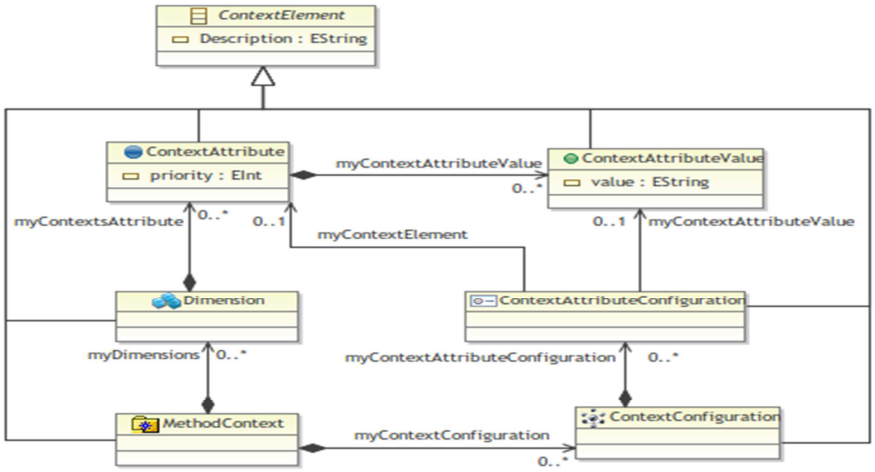

Figura 3.5. Metamodelo de Contexto caSPEM2.0

Context Element

Superclase: <<*metaclass*>> "*Element*"

Descripción: "*Context Element*" es una generalización abstracta que representa cualquier clase del paquete de Method Context.

Atributos:

"*Description*: *String*" característica que permite especificar la descripción del elemento del contexto.

Semántica: *Context Element* representa una estructura que provee a sus subclases, características, que permiten poseer campos para gestionar información importante para el usuario, en la identificación del elemento del contexto, es decir, atributos tales como nombre ("*name*"), heredado desde "*Element*" y descripción ("*Description*").

Notación Gráfica: Ninguna.

Method Context

Superclase: "Context Element*"*

Descripción: "*Method Context"* es un *Context element* concreto que representa un paquete que permite almacenar una colección de elementos del tipo *"Dimension"* y *Context Configuration.*

Propiedades Asociadas:

"myDimension: *Dimension"*. *Method Context* almacena un conjunto de *Dimensions*, las cuales son utilizadas para agrupar un conjunto de características de proyectos

semánticamente relacionadas. Esta asociación de composición representa que cada *Dimension* está contenida en un *Method Context*.

"myContextConfiguration: *Context Configuration". Method Context* almacena un conjunto de elementos del tipo *Context Configuration* que generalmente son utilizados para instanciar una configuración de contexto basada sobre las características de cada proyecto. Esta asociación de composición representa que cada *Context Configuration* está asociado a un *Method Context.*

Semántica: *Method Context* representa una estructura de almacenamiento que permite organizar y agrupar una colección de atributos de contexto contenidos en dimensiones, basándose en el nivel de detalle del diseñador del proceso, respecto a las particularidades de un grupo de proyectos y permite configurar los contextos específicos del proyecto.

Notación Gráfica: 

Dimension

Superclase: "*Context Element"*

Descripción: "*Dimension" es un contenedor para alojar elementos del tipo "Context Attribute".*

Propiedades Asociadas:

"*myContextAttribute: Context Attribute"*. La clase del tipo *Dimension* almacena un conjunto de atributos de contexto. Está asociación de composición representa que cada *Context Attribute* está asociado a una *Dimension*.

Semántica: *Dimension* representa una estructura de almacenamiento que permite organizar y agrupar una colección de atributos de contexto asociados conceptualmente.

Notación Gráfica: 

Context Configuration

Superclase: "*Context Element"*

Descripción: "*Context Configuration" es un contenedor para alojar elementos del tipo "Context Attribute Configuration".*

Propiedades Asociadas:

*"*myContextAttributeConfiguration: Context Attribute Configuration*"*. Está asociación de composición representa que cada Context Attribute Configuration hace parte de una Context Configuration.

Semántica: *Context Configuration* representa una estructura de almacenamiento que permite configurar una colección de atributos de contexto definidos en cada *Dimension* asociados a un valor único que hace parte del atributo de contexto y configurado como el contexto asociado al proyecto.

Notación Gráfica: 

Context Attribute Configuration

Superclase: "*Context Element"*

Descripción: "*Context Attribute Configuration" es una estructura que permite definir un elemento del Context Element con su respectivo valor asociado*

Propiedades Asociadas:

"*myContextAttributeValue*: *Context Attribute Value"*. Está asociación de composición representa que cada Context Attribute Value está asociado a un único Context Atribute Configuration.

"*myContextElement*: *Context Element"*. Esta Asociación de composición representa que cada *Context Element* está asociado a un único *Context Attribute Value*.

Semántica: *Context Attribute Configuration* permite configurar una pareja atributo de contexto, definido en cada *Dimension*, asociado a su respectivo valor de atributo de contexto.

Notación Gráfica: 

Context Attribute

Superclase: "*Context Element"*

Descripción: "*Context Attribute" es un contenedor para alojar elementos del tipo "Context Attribute Value".*

Propiedades Asociadas:

"myContextAttributeValue: Context Attribute Value*"*. Está asociación de composición representa que cada Context Attribute Value está asociado a una Context Attribute.

Semántica: Context Attribute representa una característica relevante del contexto del proceso que se encuentra alojada en un contenedor del tipo *Dimension*.

Notación Gráfica:

Context Attribute Value

Superclase: "*Context Element*"

Descripción: "*Context Attribute Value*" representa un valor específico para calificar o cualificar un atributo de contexto.

Atributos:

"*value*: *String*". Este atributo permite darle un valor específico a un atributo de contexto.

Notación Gráfica:

3.4.3 Rule Knowledge Package

El paquete *Rule Knowledge* permite definir y encapsular el conocimiento de adaptación del ingeniero de procesos. Este conocimiento es especificado a partir de reglas de adaptación (*Rule Knowledge*) basadas en condicionales lógicos con suficiente expresividad para expresarla.

En concreto el paquete *Rule Knowledge* define las clases *Knowledge Package*, *Abstract Condition*, *Rule*, *Action*, *Condition*, *Composite Condition*, *Simple Condition*, *And Condition*, *Or Condition*, *Rule Context* como se muestra en la figura 3.6. Estos constructos son explicados a continuación.

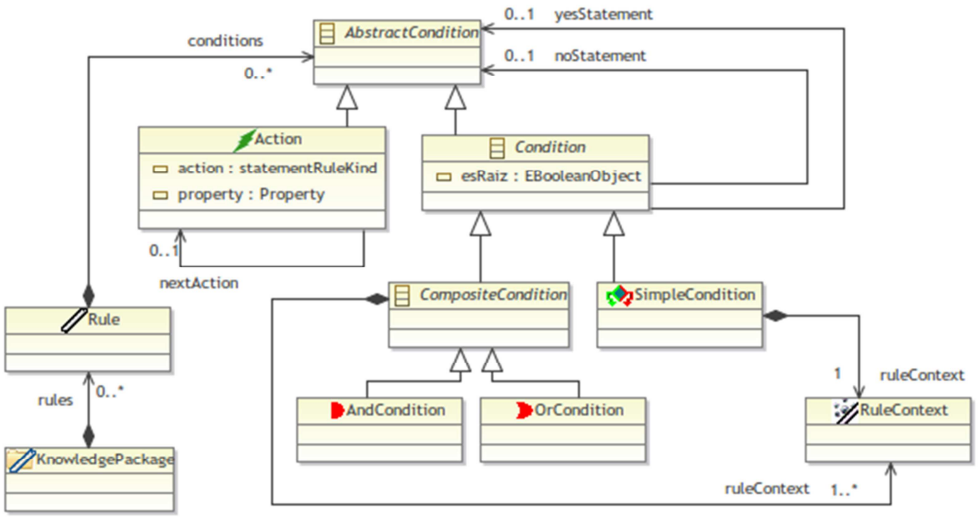

Figura 3.6. Metamodelo de Reglas de Conocimiento caSPEM2.0

Knowledge Package

Superclase: <<*metaclass*>> "*Package*"

Descripción: "*Knowledge Package*" es un contenedor que permite almacenar una colección de reglas denominadas *Rules*.

Propiedades Asociadas:

"*Rules*: *Rule*. *Knowledge Package"* almacena un conjunto de reglas que contienen la información asociada a la adaptación del proceso y el contexto del proyecto. Esta asociación de composición representa que cada Rule está incluida en un *Knowledge Package*.

Semántica: *Knowledge Package* representa una estructura de almacenamiento de conocimiento mediante un conjunto de reglas formada por elementos de decisión necesarios para la adaptación.

Notación Gráfica: 

Rule

Superclase: "<<*metaclass*>> *Element"*

Descripción: "*Rule" es un contenedor para alojar elementos del tipo "Condition"* y "*Action*".

Propiedades Asociadas:

"*conditions*: *Abstract Condition"*. Esta asociación de composición representa que cada *Abstract Condition* está asociada a una *Rule*.

Semántica: Rule representa una estructura de almacenamiento que permite organizar y agrupar una colección de estructuras condicionales simples que tienen encuentra elementos de contexto y elementos de proceso.

Notación Gráfica: 

Abstract Condition

Superclase: "<<*metaclass*>> *Element"*

Descripción: "*Abstract Condition"* es una generalización abstracta que representa cualquier tipo de condicional simple, al igual que las acciones que se pueden realizar sobre elementos del proceso.

Semántica: Abstract Condition representa un elemento que puede ser empaquetado dentro una regla.

Notación Gráfica: Ninguna.

Action

Superclase: "*Abstract Condition"*

Descripción: "*Action"* es una clase que relaciona los elementos que se encuentran dentro del proceso que se desea adaptar y la acción de adaptación que se desea hacer sobre los elementos relacionados (véase la Figura 3.7).

Atributos:

"*Action*: *statement Rule Kind"*. Este atributo representa la acción (eliminar o actualizar) que se desea aplicar sobre el elemento de proceso seleccionado en el campo *tailor element*.

"*Property*: *property"*. Este atributo establece el campo del elemento del proceso que se desea adaptar (actualizar o variar).

Propiedades Asociadas:

"*LinkedElement*: *Describable Element"*. Esta relación de composición relaciona un elemento o grupo de elementos del proceso que contribuirán en la adaptación del elemento seleccionado en el campo *tailor element*.

"*tailorElement*: *Breakdown Element"*. Está asociación de composición relaciona un elemento del proceso al que se le desea hacer la adaptación.

"*Next Action*: Action*"*. Esta relación enlaza a 0 o 1 acción con el propósito de extender el rango de acciones sobre elementos del proceso dentro de una regla de adaptación.

Semántica: Action es una clase que permite seleccionar el elemento dentro del proceso al que se desea hacer la adaptación dentro de la propiedad (*tailor Element*) y el (los) elemento(s) que influyen en la adaptación del elemento escogido, en la propiedad (*linked Element*).

Notación Gráfica: 

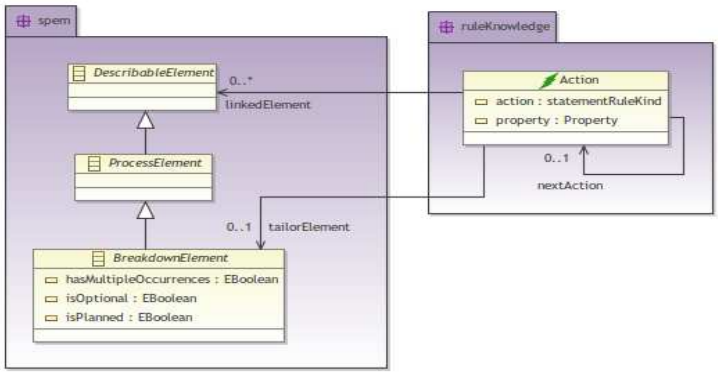

Figura 3.7. Relación entre elementos de SPEM con el paquete Rule Knowledge

Statement Rule Kind

Superclase: "*Enumeration"*

Descripción: Esta enumeración define el tipo de acciones que se van a realizar sobre el (los) elemento(s) del proceso al que se le desea hacer la adaptación (*tailor Element*) dentro de la clase *Action*, si se va a eliminar, actualizar o a adaptar teniendo en cuenta el tipo de variabilidad especificada.

Literales de Enumeración:

"Update": Enlaza la definición de un elemento de contenido (*Linked Element*) con la especificación de dicho elemento en el proceso (*Tailor Element*).

"Delete": Elimina el elemento de proceso seleccionado en el atributo *tailor Element.*

"*Update_Variability"*: Realiza la adaptación del elemento seleccionado en el atributo *tailor Element* basado sobre los mecanismos definidos por el tipo de variabilidad del elemento escogido, dentro del atributo *property* de la clase action bajo la propiedad *Variabilty Based On Element*.

Notación Gráfica: Ninguna.

Property

Superclase: *Enumeration*

Descripción: *Property* es una enumeración utilizada para definir la relación entre el elemento a adaptar (*Tailor Element*) y el elemento enlazado por el atributo *Linked Element*.

Literales de Enumeración:

"*Used Activity*": Permite enlazar una actividad previamente especificada por el atributo *Linked Element* a otra actividad especificada en el atributo *Tailor Element* en su propiedad *Used Activity*.

"*Variability Based On Element*": Permite enlazar una actividad previamente especificada por el atributo *Linked Element* a otra actividad especificada en el atributo *Tailor Element* en su propiedad *Variability Based On Element.*.

"*Target Work Product Use*": Permite enlazar uno u muchos producto de trabajo en uso como objetivos previamente especificada por el atributo *Linked Element* a una relación de producto de trabajo en uso especificada en el atributo *Tailor Element* en su propiedad *Target Work Product Use.*

"*Source Work Product Use*": Permite enlazar un producto de trabajo en uso como fuente previamente especificada por el atributo *Linked Element* a una relación de producto de trabajo en uso especificada en el atributo *Tailor Element* en su propiedad *Source Work Product Use.*

"*Parameter Type*": Permite enlazar uno u muchos definiciones de producto de trabajo previamente especificada por el atributo *Linked Element* a una definición de parámetro de tarea por defecto especificada en el atributo *Tailor Element* en su propiedad *Parameter Type.*

"*Linked Activity*": Permite enlazar una actividad previamente especificada por el atributo *Linked Element* a un ejecutor de proceso especificada en el atributo *Tailor Element* en su propiedad *Linked Activity.*

"*Linked Work Definition*": Permite enlazar una definition de trabajo previamente especificada por el atributo *Linked Element* a un ejecutor de definición de trabajo especificada en el atributo *Tailor Element* en su propiedad *Linked Work Definiton.*

Notación Gráfica: Ninguna.

Condition

Superclase: "*Abstract Condition*"

Descripción: "*Condition*" es una representación abstracta que permite definir elementos del tipo *Composite Condition* y *Simple Condition*

Atributos:

"*esRaiz*: Boolean*"*. Este campo permite establecer la condición de partida para evaluar la regla.

Semántica: Condition permite mapear una condición de tipo lógico como componente principal de una regla.
Notación Gráfica: Ninguna.

Composite Condition
Superclase: "*Condition"*
Descripción: "*Composite Condition"* es una clase abstracta contenedora que permite almacenar muchos elementos del tipo *Rule Context*.
Propiedades Asociadas:
"*ruleContext: Rule Context"*. Está asociación de composición representa que cada Rule Context está asociado a una Composite Condition.

Semántica: Composite Condition es una representación abstracta definida con el fin de poder mapear condiciones complejas, es decir, más de una condición lógica.
Notación Gráfica: Ninguna.

Simple Condition
Superclase: "*Condition"*
Descripción: "Simple *Condition"* permite almacenar un único elemento del tipo *Rule Context*.
Propiedades Asociadas:
"*ruleContext*: *Rule Context"*. Está asociación de composición representa que una instancia Simple Condición debe tener asociada un elemento del tipo Rule Context.

Semántica: Es una clase definida con el fin de mapear condiciones simples.
Notación Gráfica: Ninguna.

And Condition
Superclase: "*Composite Condition"*
Descripción: "*Composite Condition"* es una clase contenedora que permite alojar un grupo de elementos tipo *"Rule Context"* y que serán evaluados con el condicional lógico AND.
Semántica: And Condition es la representación de una conjunción lógica dentro de una regla.

Notación Gráfica: 

Or Condition

Superclase: "*Composite Condition"*

Descripción: "*Or Condition"* es una clase contenedora que permite alojar un grupo de elementos tipo *"Rule Context"* y que serán evaluados con el condicional lógico OR.

Semántica: Or Condition es la representación de una disyunción lógica dentro de una regla.

Notación Gráfica: 

Rule Context

Superclase: "<<metaclass>> *Element"*

Descripción: "*Rule Context"* es una clase asociación para relacionar elementos del tipo "*Context Attribute*" con elementos del tipo *"Context Attribute Value". (vease la Figura 3.8)*

Propiedades Asociadas:

"*contextAttributeElement: Context Attribute"*. Esta asociación de enlace representa que cada *Context Attribute* está asociado a una *Rule Context*.

"*contextAttributeValue*: *Context Attribute Value"*. Esta asociación de enlace representa que uno o muchos *Context Attribute Value* están asociados a una *Rule Context*.

Semántica: *Rule Context* es una clase que representa un atributo de contexto relacionado con su valor para permitir mapear elementos de una condición como por ejemplo "Tamaño del Proyecto: (*Context Attribute*) = Mediano: (*Context Attribute Value*)".

Notación Gráfica: 

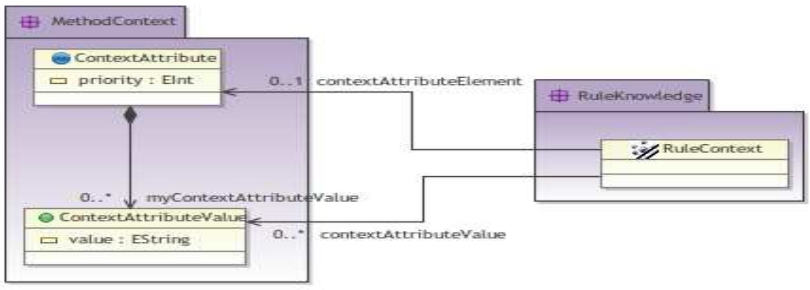

Figura 3.8. Relación entre los paquetes Method Context y Rule Knowledge

3.5 Especificación Lógica de Reglas de Adaptación de caSPEM2.0

La Especificación Lógica de Reglas tiene como objetivo proporcionar al usuario una semántica básica que estructura reglas para realizar adaptaciones de procesos software. Las reglas que se pueden definir con este lenguaje están orientadas hacia la modificación y actualización de los modelos, dado que en un bajo nivel, la plataforma realiza una copia completa de modelo del proceso y sobre esta copia trabaja la ejecución del proceso de adaptación. Las estructuras que se utilizan para realizar el lenguaje son las descritas en la sección 3.4.3, donde se definen reglas empaquetadas en objetos *KnowledgeContext*, cada una de las reglas son representadas por un objeto *Rule*, el cual compone condiciones (*AndCondition*, *OrCondition* y *SimpleCondition*) y acciones (*Action*).

Sintaxis Abstracta

La sintaxis abstracta del lenguaje de reglas se presenta a continuación:

Rule := Rule | Abstract Condition & Rule
Abstract Condition := Condition | Action
Condition := Simple Condition |Composite Condition
Simple Condition := Rule Context
Composite Condition := And Condition | Or Condition
And Condition := And Condition & Rule Context | Rule Context
Or Condition := Or Condition & Rule Context | Rule Context
Action := Action

Una Regla es un contenedor de reglas o de una condición abstracta asociada a otra regla, una condición abstracta puede ser una condición o una acción. Una condición puede ser una condición simple o una condición compuesta, la condición simple contiene una regla de contexto. La condición compuesta puede ser una condición de conjunción o una condición de disyunción, las condicionales de conjunción y disyunción contienen al menos dos reglas de contexto.

La regla es un bloque contenedor de condiciones y acciones conectadas por enlaces denominados *yesStatement* y *noStatement*, estos enlaces indican los dos casos posibles resultantes de evaluar una condición *(yesStatement* si se cumple la condición, *noStatement* si no se cumple*),* las condiciones representan si el atributo

de contexto *(ContextAttributeElement)* y el valor o valores *(ContextAttributeValue)* seleccionados en un *RuleContext*, coinciden dentro de una configuración de contexto especifica *(ContextAttributeConfiguration).* Las acciones (*Action*) representan las estrategias posibles (*update, delete* y *update_variability*) que se pueden realizar sobre un elemento del proceso (*TailorElement*), cada acción dependiendo de cómo se haya configurado tiene su propio comportamiento y puede enlazar a otra acción para adaptar otro elemento del proceso.

De manera general, la Especificación de la Lógica de Reglas representa condicionales del tipo "si <<*condición*>> entonces", las cuales evalúan dentro de su condición, si una regla de contexto está presente en la configuración de contexto especifico para un proyecto, un contexto que el usuario propone que debe satisfacer para realizar o no acciones de adaptación. Por ejemplo, si se tiene en un proceso general una tarea denominada "Diseño de la Arquitectura" encargada de producir un "modelo o representación técnica del software que se va a desarrollar" en un proyecto, pero el Dominio Técnico del Equipo de Desarrollo sobre el producto software es Alto, es decir el equipo de desarrollo ya ha desarrollado y entregado un producto software con características similares en muchas ocasiones, en este caso esta actividad no debe ser considerada." O en el caso donde el tipo de Proyecto a desarrollar sea un Mantenimiento entonces no es necesario tener en cuenta esta actividad. Así, la regla de adaptación que el ingeniero de proceso propone para realizar adaptaciones debe tener un esquema como el siguiente:

Si el Conocimiento Técnico es Alto entonces No realizar la Tarea en Uso Diseño de la Arquitectura, en caso contrario evaluar si el Tipo de Proyecto es Mantimiento, en cuyo caso Tampoco* debe ser tenido en cuenta *la Tarea en Uso, en los demás casos si debe ser realizada.

Otra manera de ver la misma regla puede ser la siguiente:

La Tarea en Uso *Diseño de la Arquitectura* solo debe ser tenida en cuenta Cuando el Conocimiento Técnico No sea Alto o el Tipo de Proyecto No sea un Mantenimiento.

Como se puede observar en la figura 3.9, en la regla de adaptación se evalúa el contexto *Conocimiento Técnico* y pregunta si éste es *Alto*. Si el Conocimiento Técnico para un proyecto específico cumple esta condición, entonces el proceso adaptado no debe tener dentro de su secuencia de actividades la Tarea en Uso *Diseño de la Arquitectura.* En caso de que esta regla no se cumpla se debe evaluar si el Tipo de Proyecto es Mantenimiento en cuyo caso la Tarea en Uso tampoco debe

ser tenida en cuenta. Este esquema de la lógica es la que pretende representar la Especificación de la Lógica de Reglas, donde las acciones de adaptación modifican o borran elementos de la secuencia del proceso.

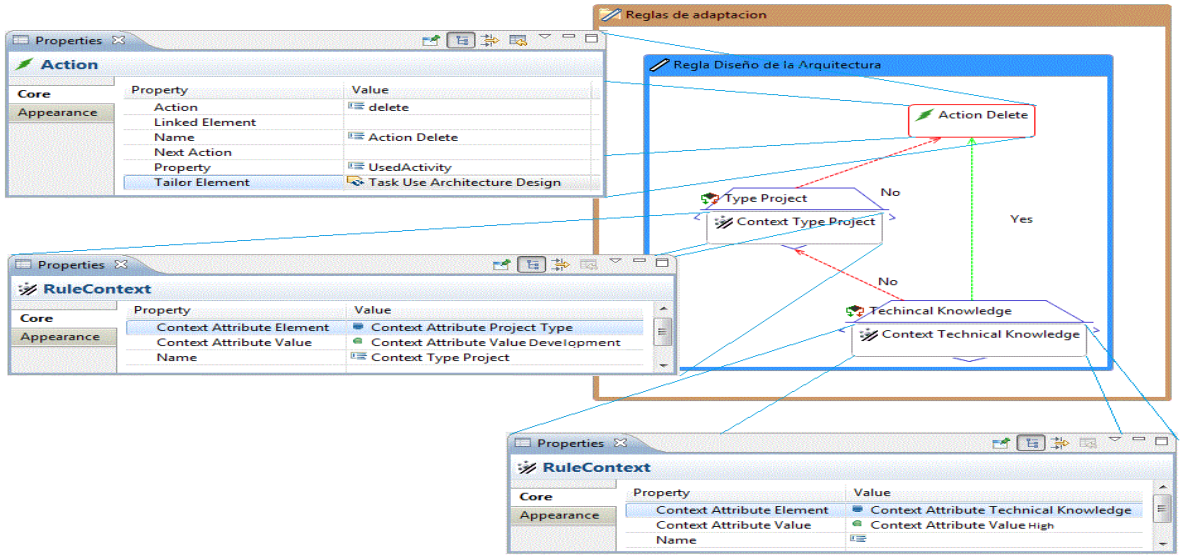

Figura 3.9. Diseño de una Regla de Adaptación.

Sintaxis Concreta

Para ser factible una regla dentro de la Especificación de la Lógica de Reglas, es indispensable que el usuario defina un modelo de contexto, en el cual se estructuran todas las características contextuales que rodean comúnmente a los proyectos software, y que afectan el comportamiento del proceso de acuerdo a estas características, como la mostrada en la Figura 3.10. Por tanto, es indispensable que el usuario defina una configuración de contexto especifica, la cual representa el contexto real que rodea el proyecto software para que la reglas de adaptación consulten en esta configuración los atributos de contexto y sus posibles valores. En la figura 3.10 (Configuración de Contexto Especifico) se muestra un ejemplo de una configuración, que representa un contexto específico de proyecto, con estos dos requisitos se tiene los parámetros indispensables para que una regla de la Especificación de la Lógica de Reglas sea diseñada.

Toda regla dentro de la Especificación de la Lógica de Reglas tiene que contener como mínimo una condición, de no tenerla la regla estaría mal formada y aunque en el proceso de adaptación no causaría problema, desde el punto de vista de la sintaxis y semántica que pretende representar la Especificación de la Lógica de

Reglas, no tendría significado. Como utilizaremos las reglas de la forma "*Si Conocimiento Técnico es Alto entonces no realizar la Tarea Diseño de la Arquitectura, de lo contrario evaluar Si el Tipo de Proyecto es Mantenimiento en este caso tampoco se debe realizarla en caso contrario realizarla* ", donde encontramos la condición que evalúa el contexto *"Conocimiento Técnico con valor Alto",* en este caso como sólo se evalúa una condición de contexto que sólo involucra un atributo de contexto, se utiliza el condicional simple denominado *"SimpleCondition"*. En otros casos cuando se evalúan más de un atributo de contexto dentro de una regla de adaptación se hace uso de los condicionales *"AndCondition"* y "*OrCondition*" que representan los operadores de conjunción y disyunción respectivamente. Estos condicionales exigen por lo mínimo se evalúen dos atributos de contexto distintos, aunque admite evaluar un atributo de contexto con valores diferentes, la regla no tendría un uso adecuado debido a que se perdería la capacidad de reutilización, puesto que caSPEMTool permite evaluar un mismo atributo de contexto con distintos valores mediante la lógica disyuntiva, observe la figura 3.11 (b).

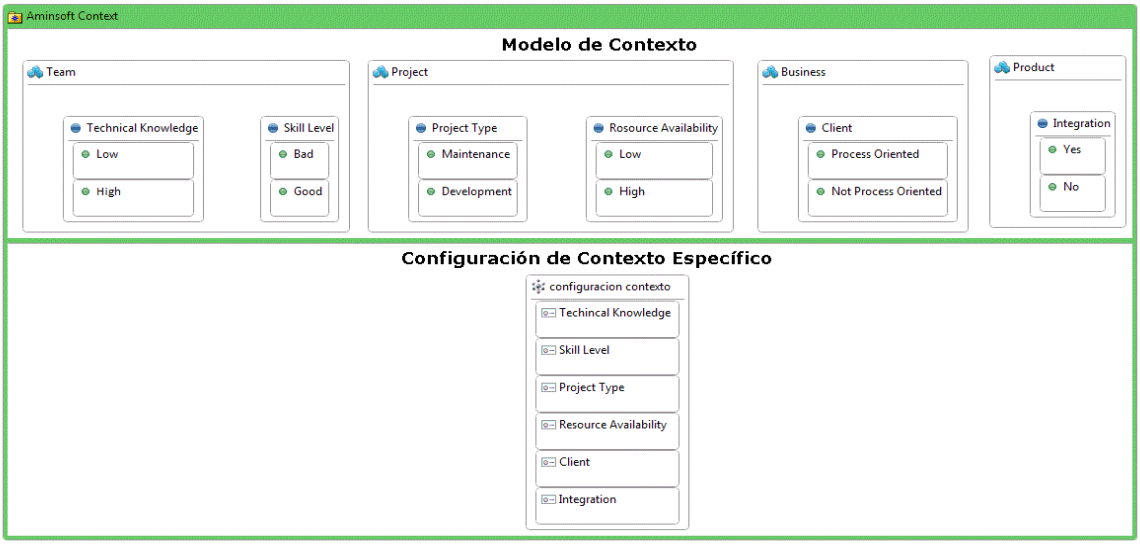

Figura 3.10. Contexto Modelado y Configuración Específica de un Proyecto

La manera como la Especificación de la Lógica de Reglas representa el valor del atributo de contexto que evalúa la verdad o falsedad de la condición, es mediante la estructura *"RuleContext".* Véase un ejemplo en la figura 3.11, en el cual las propiedades del objeto configuran un atributo de contexto y el valor del atributo de contexto (es uno o varios valores definidos en el modelo de contexto). Si el atributo de contexto no se configura correctamente con sus valores correspondientes, la regla no estará bien formada y por lo tanto no tendrá incidencia sobre el proceso adaptado,

salvo que la lógica del usuario este mal planteada. Hasta aquí, con los elementos ya mencionados las condiciones y los atributos de contexto junto con sus valores, se tiene la o las condiciones de verdad o falsedad de una regla formulada en la Especificación Lógica de Reglas.

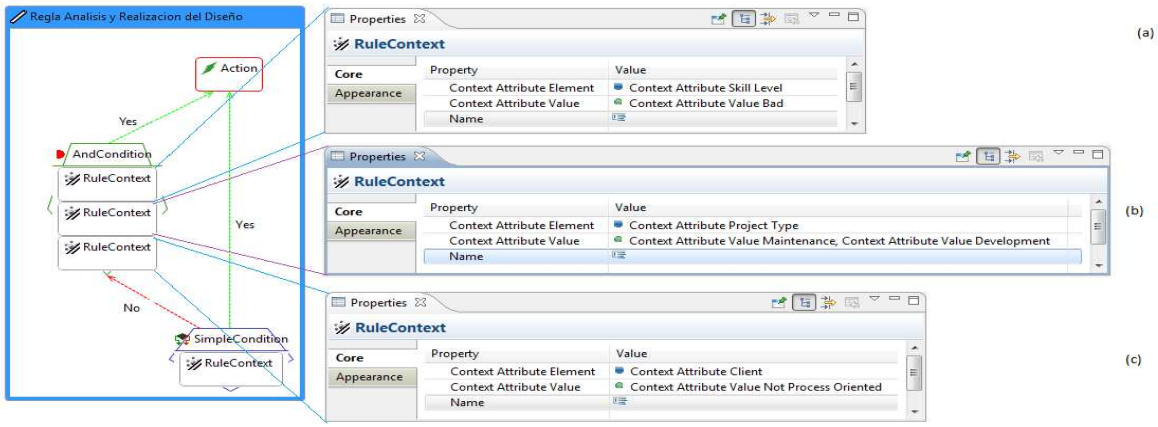

Figura 3.11. Ejemplo de una Regla con Condicional Compuesto.

La Especificación Lógica de Reglas define acciones de adaptación mediante el constructo *"Action"* en la cual el usuario configura la operación de adaptación más adecuada para su regla, en éste constructo se configura en la propiedad "*Action*" que indica a la regla la estrategia de adaptación que debe adoptar el proceso de adaptación de procesos software, esta propiedad puede tomar tres distintos valores "*update*", *"delete" y* "update_variability", cuyo comportamiento esta descrito en la seccion 3.4.3. La propiedad *"LinkedElement"* es una propiedad que se configura cuando se selecciona en la propiedad *"Action"* el valor *"update"* o *"update_variability",* indicando a la acción qué valor se enlaza en el elemento del proceso sobre el cual se hace la acción de adaptación *(TailorElement)*. En la Figura 3.12 se puede observar que se ha configurado en el nodo *Action,* la propiedad *linkedElement* (recuadro naranja) una *Task Definition Test Driven Components and Programs Development* y en el *TailorlElement* (recuadro rojo) se ha configurado una *Task Use Software Components Development* y el valor de la propiedad *Action* (recuadro verde) como "*update*", lo que indica que si la condición es verdadera entonces a la *Task Use* se le asignará la *Task Definition Test Driven Component and Program Development* en la propiedad *Task* (recuadro naranja).

Como dentro de la lógica cabe que una regla afecte a más de un elemento de proceso, la Especificación de la Lógica de Reglas en la estructura "*Action*" puede ser configurada para que de paso a más acciones, esto se logra mediante la propiedad

"NextAction", que indica en una regla su alcance en términos de elementos de proceso afectados. Por último, y ya para finalizar las condiciones se enlazan a otras condiciones o acciones mediante los enlaces *Yes* y *No,* indicando en la regla que si la condición de la regla es verdadera el proceso de adaptación de procesos software evaluará la condición o ejecutará la acción de adaptación que el enlace "*Yes"* indique, de la misma manera, si la condición es falsa el proceso de adaptación de procesos software evaluará la condición o ejecutará la acción de adaptación que el enlace "*No"* indique. De esta manera, con los elementos ya mencionados se tiene una regla de adaptación especificada en la Especificación Lógica de Reglas y que son el recurso primario para la adaptación de procesos software con este enfoque.

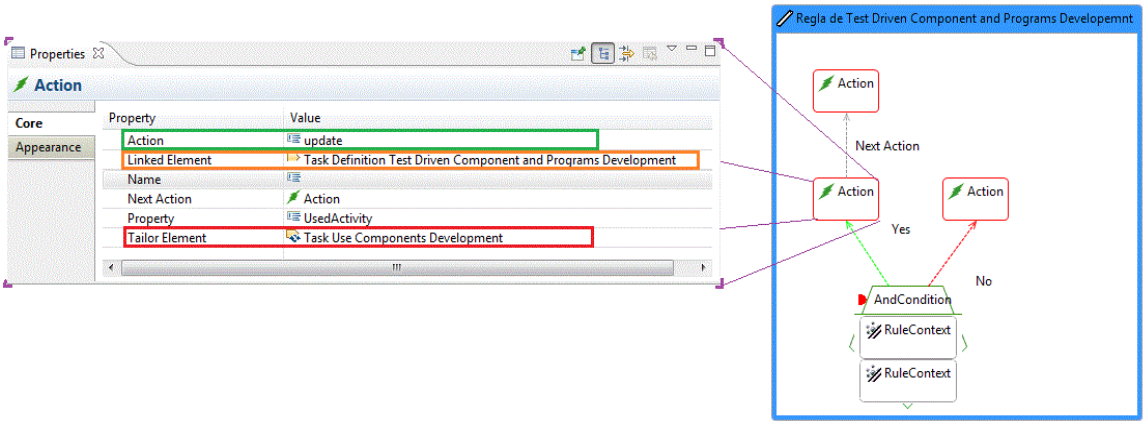

Figura 3.12. *Action* Configurada para Actualizar un Elemento del Proceso

3.6 Síntesis y Discusión

En este capítulo hemos presentado el metamodelo caSPEM2.0 como una extensión de SPEM 2.0, el cual permite el diseño de procesos software incluyendo la información de las características relevantes del entorno al que pertenece un proyecto o un grupo de proyectos, a fin de ser tenidas en cuenta al momento de adaptarlo. caSPEM2.0 permite instanciar un paquete de conocimiento mediante el agrupamiento de un conjunto de reglas que almacenan condicionales simples, a modo de un grafo dirigido, con el fin de que el ingeniero de procesos diseñe reglas de adaptación teniendo en cuenta las características de contexto basándose en el análisis sobre la forma en que las características del contexto impactan la aplicación de ciertos elementos del proceso.

4 Prototipo caSPEMTool: Validando la Especificación de caSPEM2.0

4.1 Introducción

Durante la exploración de las herramientas que modelen procesos software, útiles para realizar el prototipo, encontramos EPF Composer, dado que está hecho sobre la plataforma eclipse de Java y de código abierto, es viable realizar extensiones que permitan agregar nuevos elementos a los modelos de proceso. Como primer inconveniente, se presentaron problemas técnico (incompatibilidad de las librerías, dependencias con plug-ins eclipse), cuya solución tomo mucho tiempo en resolver, una vez se resolvieron, se procedió a realizar plug-ins de prueba para agregar funcionalidad a la aplicación, esto fue relativamente fácil, dado que existen documentación de buena calidad que guían este proceso. Al momento de querer incorporan nuestro enfoque en la herramienta EPF Composer, el metamodelo que respalda los modelos de proceso de la herramienta (UMA), no respalda completamente la especificación SPEM 2.0 lo cual no hace posible proponer una extensión, además el código de la aplicación es muy complejo y tomaría mucho tiempo entenderlo, y como último paso agregar los elementos nuevos, dado nuestra propuesta agrega nuevos paquetes en un nivel muy complicado (a nivel de "Method Content" y "Process") que hace muy difícil la construcción de un plug-in para esta herramienta.

El metamodelo caSPEM2.0 presentado en el capitulo anterior, proporciona los conceptos necesarios para el diseño y adaptación de un proceso software teniendo en cuenta la estructura del proceso, las características del contexto para representar contextos específicos a proyectos y el conocimiento de adaptación. Para validar la especificación del metamodelo se ha desarrollado un prototipo llamado caSPEMTool,

usando un enfoque IDM/MDE (Ingeniería Dirigida por Modelos / Model-Driven Engineering), en el marco del entorno de desarrollo de EuGENia, sobre Eclipse Modeling Framework. Desarrollar editores gráficos con esta tecnología permite generar prototipos altamente funcionales, de manera rápida, a partir de modelos especificados en un metamodelo Ecore con anotaciones y mediante transformaciones genera modelos alternos, que más tarde son la fuente para generar el código Java del editor.

4.2 Modelo de Requisitos caSPEMTool

caSPEMTool es un editor de procesos que permite modelar gráficamente procesos de sistemas y software bajo el estándar SPEM2.0, permitiendo la creación, modificación y eliminación de elementos de contenido de método, elementos de proceso, elementos de contexto, reglas de adaptación y las relaciones entre estos, en un solo entorno de trabajo, simulando las entidades a modelar en forma de nodos y sus relaciones en forma de enlaces (véase Figura 4.1).

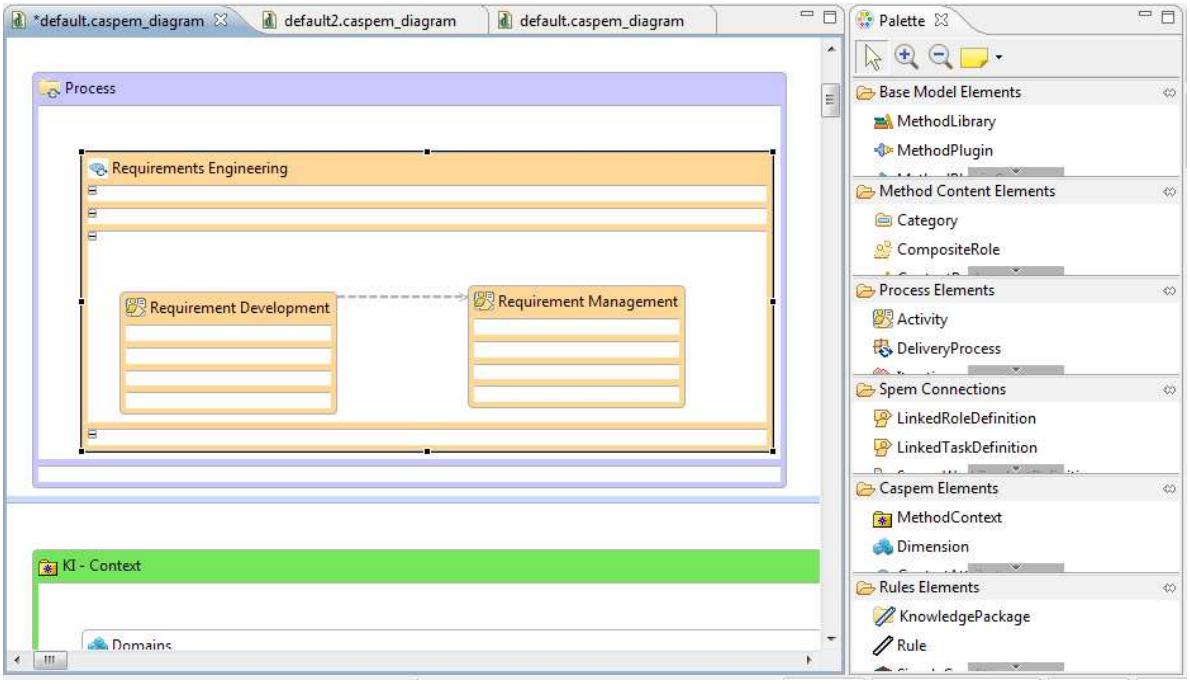

Figura 4.1. Espacio de trabajo caSPEMTool

En esta sección se explica la funcionalidad del editor gráfico caSPEMTool, descrito a través del diagrama de casos de uso de la figura 4.2, que explica la funcionalidad del editor.

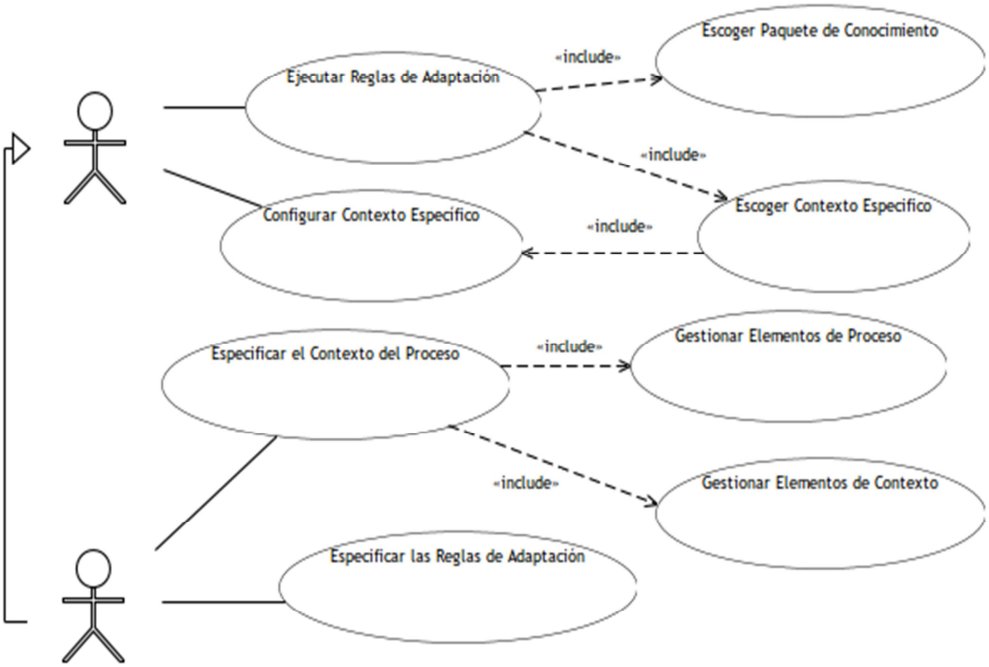

Figura 4.2. Diagrama de Casos de Uso del Editor Gráfico caSPEMTool

caSPEMTool es una herramienta dirigida a ingenieros de proceso (diseñadores del proceso) y a usuarios que participan activamente de la planeación y ejecución de un proyecto de desarrollo software. El ingeniero de procesos tiene la función de modelar el proceso general de la organización y establecer guías de adaptación del proceso a proyectos específicos. El usuario es un actor que tiene la necesidad de obtener un proceso adaptado a sus necesidades de proyecto para ejecutarlo.

El diseñador de procesos puede crear, modificar y eliminar elementos de un modelo de proceso software basándose en el lenguaje de metamodelado caSPEM2.0 que incluye información de las características que definen un proyecto o grupo de proyectos a través del caso de uso Gestionar Procesos. El ingeniero de procesos teniendo en cuenta las características definidas en el paquete de contexto puede definir la lógica de adaptación del proceso mediante reglas de condicional simples agrupadas dentro del paquete de conocimiento a través del caso de uso Especificar Reglas de Adaptación. El usuario del proceso puede configurar un contexto específico a un proyecto basado en las características definidas

anteriormente y ejecutar las reglas de adaptación a través del caso de uso Ejecutar Reglas de Adaptación; aunque definir las reglas de adaptación es una tarea sencilla que puede ser realizada por el desarrollador dentro del editor gráfico, se considera que esta tarea solo debe ser hecha por diseñadores de proceso debido al bagaje que poseen, puesto que se debe hacer un análisis de cómo se relacionan los atributos del contexto modelado con los elementos del modelo del proceso a variar.

Para facilitar al diseñador de proceso la construcción de la reglas se recomienda que las construya de tal manera que evalúen primero el contexto que debe rodear al proyecto y así encadenar las acciones de adaptación sobre los elementos del proceso que son afectados por el contexto. De esta manera se logran reglas mejor planteadas, permitiendo su comprensión y implementación rápida y son un mecanismo práctico para almacenar el conocimiento tácito del diseñador de procesos.

El caso de uso Ejecutar las Reglas de Adaptación implica que el usuario deba seleccionar qué paquetes de conocimiento (paquete de reglas) va a ejecutarse durante la adaptación del proceso y el contexto específico[8] del proyecto mediante los casos de uso Escoger el paquete de Conocimiento y Escoger el contexto especifico respectivamente, que permiten parametrizar la entrada con el propósito de ejecutar la adaptación, esta funcionalidad finalmente entrega un proceso software adaptado.

El ingeniero de software a través del caso de uso Configurar un contexto específico puede atribuir a un conjunto de atributos contextuales, valores específicos del proyecto de aplicación, en otras palabras, este es el medio por el cual se identifica de manera más exacta las condiciones del ambiente en el cual el proyecto se ejecutará. El ingeniero de procesos inicia el caso de uso Especificar un modelo de proceso para modelar los contenidos de método (Tareas, Artefactos, Productos de Trabajo, Guidance) y el proceso, donde se gestionan los elementos[9] que intervienen en el proceso de desarrollo Software y los elementos de contexto (dimensiones y atributos de contexto), mediante los casos de uso Gestionar elementos de proceso y Gestionar Elementos de Contexto respectivamente, en este punto el proceso estará enriquecido con la definición de atributos de contexto, los cuales se tomarán en cuenta para la adaptación sistemática de procesos software. El Ingeniero de procesos a través del caso de uso Especificar las Reglas de Adaptación mapea las reglas que guiarán la adaptación del proceso software, estás

[8] conjunto configurado de características del entorno que son específicas al proyecto.
[9] las actividades, tareas, roles y productos de trabajo.

siguen una metáfora gráfica similares a los condicionales de los diagramas de flujo como se observa en la figura 4.3 (cuadro resaltado en azul), donde los pentágonos evalúan las condiciones de contexto (RuleContext), las líneas roja y verde indican lo que pasaría si cumple o no la condición (verde cumple, roja no cumple), estas líneas pueden enlazar otras condiciones o una acción de adaptación (Action), en el caso de ser una acción de adaptación, este elemento es quien finalmente afecta los elementos del proceso durante la adaptación. En esta parte, el sistema provee un mecanismo mediante el cual si un condición de contexto afecta a más de un elemento de proceso, estas acciones pueden ir encadenadas mediante el enlace denominado "NextAction", y es el punto de control que tiene nuestro Script EOL para detener la ejecución de una regla, es decir, cuando no encuentre mas acciones de adaptación en la regla.

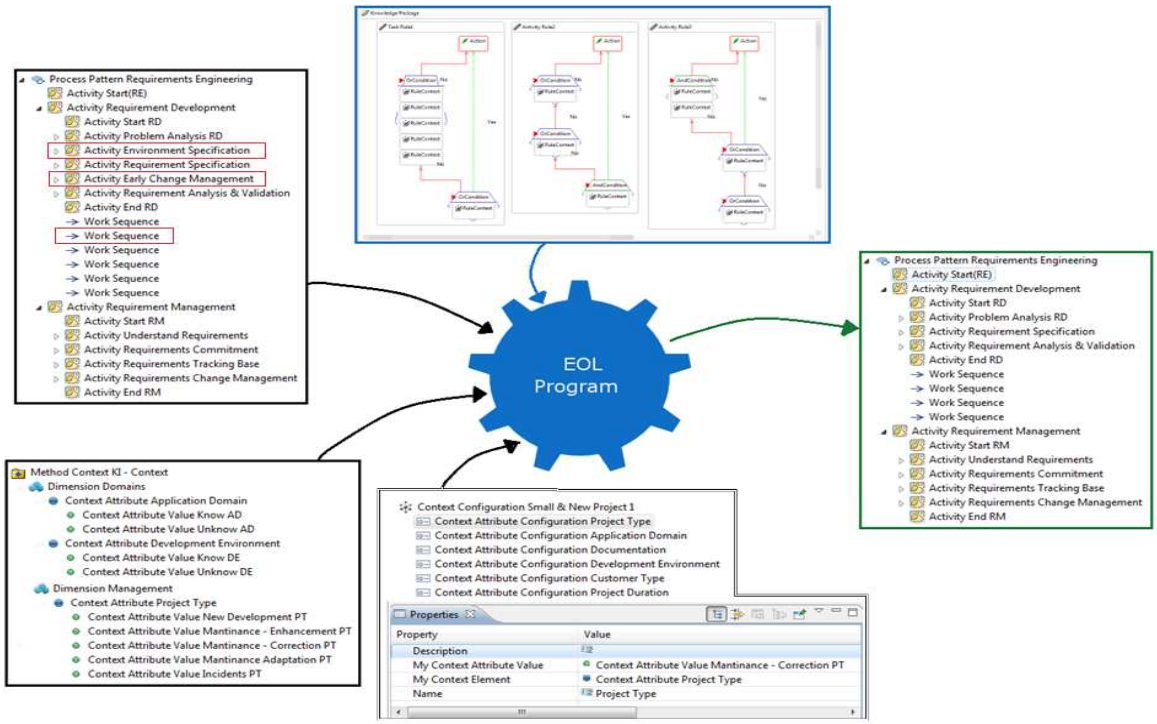

Figura 4.3. Modelo de Contexto caSPEMTool.

Como se observa en el metamodelo de contexto en la figura 4.3 una vez el desarrollador configure el contexto especifico del proyecto, para ejecutar las reglas de adaptación se provee de un script EOL, que toma el modelo del proceso, más específicamente el paquete de reglas, y las recorre como un grafo dirigido, cuyos nodos dirigen la adaptación del proceso software, y a través de una configuración

EOL Program de la Plataforma Eclipse, encargada de ejecutar el script sobre el modelo, genera un proceso Software adaptado con la definición del proceso y el contexto incluyendo la configuración de contexto en un nuevo archivo. El proceso software adaptado resultante carecerá de las reglas de adaptación, por la siguiente razón, el proceso resultante es la aplicación de las reglas de adaptación sobre el modelo y no tendría sentido mantenerlas, además, si hay una regla de adaptación que elimina elementos del proceso, en el proceso adaptado tal elemento no existirá, y por lo tanto la regla quedará inconsistente.

4.3 Tecnología de Desarrollo MDD

El Desarrollo del prototipo siguió una estrategia MDE donde los modelos abstractos de un sistema software son creados y transformados sistemáticamente en implementaciones concretas. La base del enfoque MDE es la implementación del modelo de dominio conceptual el cual se hay definido a un nivel de abstracción muy alto, para la generación de este metamodelo existen varios lenguajes de modelado entre los que se encuentran KM3 [62] MOF [23] y Ecore [63]. Este metamodelo es procesado por distintas herramientas para la generación semi-automatizada de modelos que ayudarán en la generación de editores gráficos. Por esta razón, caSPEMTool será construido mediante Eclipse Modeling Framework y Graphical Modeling Framework, dado que la naturaleza de nuestra propuesta se fundamente en la idea de un metamodelo y Ecore satisface esta pretensión.

4.3.1 Eclipse Modeling Framework

EMF[10] es un framework de modelado y generador de código que permite la construcción de aplicaciones Java partiendo de definiciones de modelo sobre la plataforma Eclipse. Las definiciones de modelo pueden ser pueden ser provistas desde herramientas de modelado UML, esquemas XML o incluso Java, en el caso de Java los modeladores deben proveer un conjunto de interfaces Java abstractas y EMF se encarga de generar el código restante. El metamodelo usado para representar modelos dentro de EMF es denominado Ecore, que a su vez es un

[10] EMF: Eclipse Modeling Framework. Más información puede encontrarse en: http://www.eclipse.org/modeling/emf/

modelo EMF en sí mismo, Ecore es un pequeño y simplificado UML, dado que el soporte completo a UML es mucho más ambicioso que el soportado por EMF [63].

Para representar los modelos, Ecore utiliza cuatro clases denominadas *Eclass*, *EAttribute*, *EReference* y *EDataType*, las cuales son utilizadas para representar clases, atributos, relaciones y datos primitivos respectivamente, su relación semántica dentro de Ecore puede verse en la Figura 4.4. Eclipse provee un editor simple basado en árbol por manipular modelos Ecore, es muy intuitivo dado que su notación es basada en UML. Para la construcción de la herramienta se utilizó el proyecto Eclipse Modeling Framework Technology (EMFT), que hace parte de las tecnologías que incuba Eclipse para extender EMF, el cual le da soporte tecnológico a un lenguaje denominado EMFatic diseñado para representar modelos Ecore de manera textual, esta opción resulto más factible dado que editar modelos complejos con el editor de árbol se hace una tarea tediosa, y las anotaciones que permiten hacer alusión a su representación gráfica, es más viable hacerlas de manera textual que haciendo uso del editor propio para modelos Ecore llamado Graphical Ecore Editor [63].

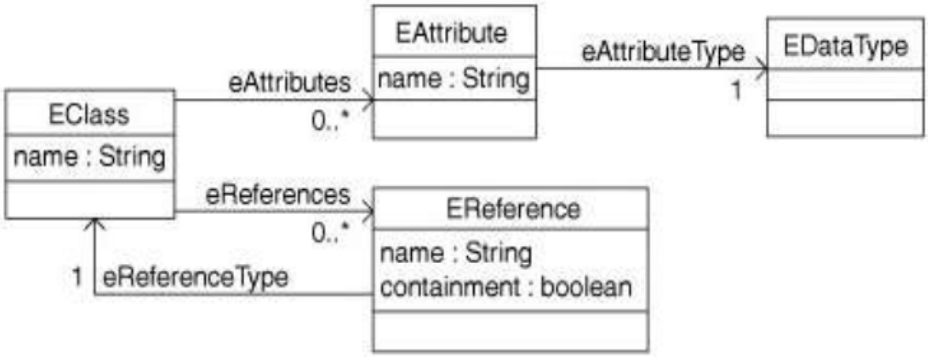

Figura 4.4. Subconjunto Simplificado del Metamodelo ECORE. Tomado de [63]

4.3.2 Graphical Modeling Framework

Graphical Modeling Framework (GMF) es un marco de trabajo de Eclipse que permite crear un editor gráfico, sobre un metamodelo especifico, siguiendo patrón arquitectónico Modelo-Vista-Controlador (MVC). GMF proporciona componentes e infraestructura para el desarrollo de editores gráficos basado en EMF (Eclipse Modeling Framework) y GEF (Graphical Editing Framework). GMF utiliza seis archivos específicos para la generación del editor gráfico, el modelo del dominio

(Domain Model) el cual es el metamodelo principal que contiene toda la información de las clases y objetos para la creación del editor gráfico, la generación de este metamodelo es posible mediante varios tipos de metamodelos diferentes: Código Java con anotaciones, Modelo Ecore, Modelos UML, Esquemas XML. El Modelo de Dominio Generado (Domain Gen Model - .genmodel) es el metamodelo usado para la generación del código del modelo de dominio. El Modelo de Definición Gráfica (Graphical Def Model - .gmfgraph) es el metamodelo usado para definir los elementos gráficos para el modelo de dominio, en este archivo es posible definir que clases del metamodelo será usadas como nodos y enlaces dentro del editor gráfico. El Modelo de Definición de Herramientas (Tooling Def Model - .gmftool) es usado para definir la paleta de herramientas que será usado en el editor gráfico. El Modelo de Mapeo (Mapping Model - .gmfmap) es usado para relacionar el modelo de Dominio, el Modelo de Definición Gráfica y el Modelo de Herramientas. Por último el Modelo de Generación de Diagrama del Editor es usado para generar el editor gráfico GMF adicional al código EMF generado por el archivo que contiene el Modelo de Dominio Generado (.genmodel).

La gráfica 4.5 muestra el proceso de generación de un editor grafico GMF mediante los modelos previamente explicados.

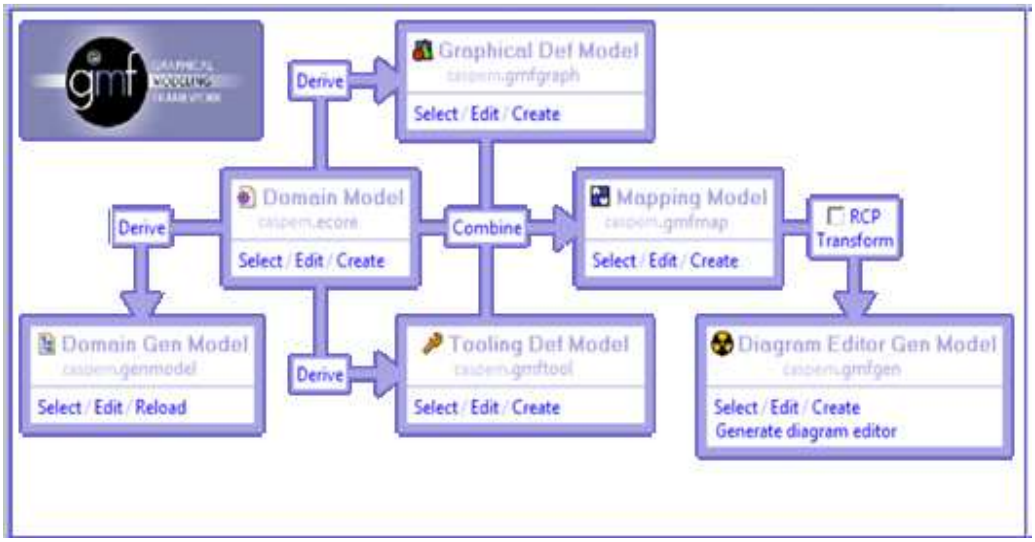

Figura 4.5. Proceso de Generación de un Editor Gráfico con el Dash Board de Eclipse

4.3.3 Epsilon y EuGENia

EuGENia es una herramienta que sigue los principios de la ingeniera dirigida por modelos, soportada sobre lenguajes de la plataforma Epsilon de Eclipse, la cual

adopta un enfoque de *fuente única*, basado sobre anotaciones[11] en el metamodelo y utiliza técnicas de transformación de modelos para producir y mantener todos los modelos EMF y generadores de código GMF de una manera semi-automatizada.

Lenguajes Epsilon

Epsilon es una plataforma para construir lenguajes de tarea específica consistentes e interoperables para tareas de gestión de modelos como por ejemplo transformación, comparación, fusión ("merging"), refactorización ("refactoring"), validación y generación de código de los modelos. Epsilon provee un conjunto de lenguajes los cuales se encargan de parte de la gestión de modelos, entre ellos Epsilon Object Language (EOL), Epsilon Validation Language (EVL), Epsilon Transformation Language (ETL), Epsilon Comparison Language (ECL), Epsilon Merging Language (EML), Epsilon Wizard Language (EWL) y Epsilon Generation Language (EGL).

El lenguaje EOL (Epsilon Object Language), es un lenguaje básico de restricción de objetos equivalente a OCL[12] en UML de la OMG, posee un conjunto limitado de sentencias comunes que sirven para implementar otros lenguajes, característica que permite usarlo como lenguaje de propósito general bajo Epsilon. El lenguaje EVL (Epsilon Validation Language) construido sobre EOL, es usado para especificar y validar las restricciones de los modelos del metamodelo, las restricciones que se especifican tienen una estructura parecida a las restricciones de OCL, permitiendo personalizar los mensajes de error y alerta, evaluar las restricciones propias del metamodelo tales como la no nulidad u obligatoriedad de definir un relación entre los elementos del modelo, además, hace posible que los errores encontrados sean reparados de manera automática cuando se ejecuta un script EOL sobre el modelo.

El lenguaje ETL (Epsilon Transformation Language) permite a Epsilon la posibilidad de hacer transformaciones de modelo de la forma modelo-a-modelo (M2M) de forma similar a ATL (Atlas Transformation Language), en los que se toma un modelo de entrada y lo transforma en un conjunto de modelos de salida. Se debe especificar previamente los metamodelos para los modelos de entrada y salida, su estructura se basa en reglas declarativas en donde su sintaxis toma el elemento del modelo de origen y lo convierte en un elemento del modelo destino. El lenguaje ECL (Epsilon Comparision Language) permite especificar algoritmos de comparación a

[11] Las anotaciones son un conjunto de estereotipos que brindan información adicional al metamodelo Ecore para enriquecer de características gráficas los elementos modelados útiles para el editor gráfico.
[12] Object Constraint Language. Más información puede encontrarse en: http://www.omg.org/spec/OCL/2.2/

manera de reglas entre dos modelos, las cuales identifican el par de elementos a ser comparados (uno de cada uno de los modelos), y el grado de similaridad, útil cuando se desea fusionar dos modelos, durante el proceso de unificación. El lenguaje EML (Epsilon Merging Language) permite fusionar dos modelos, especificando la fusión a maneras de reglas, donde toma los elementos a ser fusionados (uno de cada uno de los modelos) y mediante sentencias EOL modifica el modelo objetivo para realizar las correspondientes modificaciones, tomando en cuenta la comparaciones hechas mediante un script ECL.

El lenguaje EWL (Epsilon Wizard Language) permite realizar transformaciones de modelo directamente en el modelo fuente sobre los elementos que el usuario haya definido, esta herramienta es útil a la hora de automatizar ciertas tareas (adicionar mutadores, accesores, constructores, etc) durante la gestión de modelos, los scripts EWL se ejecutan configurando la plataforma Eclipse y son visibles en un menú contextual que aparece cuando seleccionamos (con Click derecho) el elemento que cumple con la restricción ("guard"). El lenguaje EGL (Epsilon Generation Language) permite ser usado para realizar transformaciones del tipo modelo-a-texto (M2T), esto es útil a la hora de generar código, dado que el script EGL se toma como una plantilla para la generación el código correspondiente para gestionar los modelos del metamodelo.

Enfoque EuGENia

EuGENia surge como solución al problema de la complejidad requerida para la implementación de editores gráficos mediante el framework GMF, dado que en primer lugar el desarrollador necesita especificar los elementos gráficos (figuras, conexiones, etiquetas, etc.) involucrados en el editor mediante la configuración del modelo GMF*Graph*; especificar los elementos que estarán disponibles en la paleta del editor mediante el modelo GMFT*ooling* y mapear los elementos gráficos del modelo GMFGraph y los elementos del modelo GMFTooling con la sintaxis abstracta del metamodelo Ecore mediante el modelo GMFM*ap,* mediante editores simples provistos por GMF. Éstos a su vez son la fuente para un modelo final (*GMFGen*) el cual es tomado por el generador de código para producir el código Java requerido para el editor gráfico, este proceso es largo, tedioso y se necesita de gran experticia por parte del desarrollador para mantenerlo; por otro lado el framework GMF no permite una actualización automática de los modelos previamente generados dado que si se personalizan los modelos o se realizan pequeños cambios, se debe replicar el cambio en los demás. Esto hace que el proceso sea semi-automatizado.

El propósito de EuGENia es disminuir la complejidad entre los frameworks EMF y GMF para la implementación de editores gráficos mediante transformaciones de modelos, dado que logra reducir el proceso de implementar el editor gráfico automatizando la generación los modelos GMF*Graph*, GMFT*ooling*, GMFM*ap* y *GMFGen*. Las anotaciones que incorpora EuGENia le permite a los desarrolladores incluir en el metamodelo Ecore la información necesaria a un alto nivel para implementar un editor gráfico que incluye metáforas gráficas como "contenedores", "nodos", "enlaces" y asociación de iconos [64], muy útiles a la hora de personalizar el editor y que son conceptualmente conocidos por la comunidad de desarrollo software en general.

4.4 Construcción del Prototipo caSPEMTool con EuGENia Siguiendo el Enfoque MDE

4.4.1 Especificación del Metamodelo

Con el fin de construir el prototipo *caSPEMTool* con *EuGENia* se generó un archivo *Emfatic*[13] del metamodelo caSPEM2.0 con las anotaciones[14] respectivas, en la tabla 2 se detallan las anotaciones utilizadas para el metamodelo. *EMF* permite generar el metamodelo *Ecore* desde el archivo *Emfatic*, donde se incluyen las anotaciones necesarias para la adecuada generación de código. Mediante las anotaciones que EuGENia soporta, toda clase definida en el metamodelo *Ecore* es provista de propiedades de representación gráfica, es decir que los objetos que son de la clase definida tendrán el comportamiento que la anotación indica, estas anotaciones son tenidas en cuenta al momento de generar los modelos de soporte gráfico *.gmfgraph*, *.gmftooling*, *.gmfmap.*

Por ejemplo en la figura 4.6 se muestra un fragmento de un archivo *emfatic* con anotaciones, indicando que los objetos del tipo *Action* aparecerán dentro del gráfico como nodos (*gmf.node*) con el borde de color rojo y será etiquetado por defecto con su nombre, la clase *Action* ha definido cinco atributos, uno de ellos llamado *nextAction* tiene una anotación asociada que indica que esta propiedad será representada como un enlace a elementos del mismo tipo de la clase, el enlace aparecerá como una flecha (*arrow*) punteada (*dash*). Los objetos del tipo *Rule* serán

[13] Emfatic: Es un lenguaje diseñado para representar modelos Ecore EMF de manera textual.
[14] Las anotaciones soportadas por Eugenia pueden encontrarse en: http://www.eclipse.org/epsilon/doc/articles/eugenia-gmf-tutorial/

representados como un nodo contenedor (*@gmf.comparment*) que alojará muchos elementos del tipo AbstractCondition. Para más información sobre este metamodelo remítase al Anexo A

```
@gmf.node(border.color="255,0,0", label="name")
class Action extends AbstractCondition {
  ref BreakdownElement tailorElement;
  attr statementRuleKind action;
  ref DescribableElement[*] linkedElement;
  @gmf.link(tool.name="Next Action", label="Next Action",
                  target.decoration="arrow", style="dash")
  ref Action nextAction;
  attr Property property;
}

@gmf.node(label="name", color="50,150,255")
class Rule extends SpemElement{
  @gmf.compartment(foo="bar")
  val AbstractCondition[*] conditions;
}
```

Figura 4.6. Fragmento de un archivo emfatic con anotaciones EuGENia.

4.4.2 Generación de Modelos de Soporte

Mediante la transformación modelo a modelo (*Ecore2GenModel*) implementada en *EuGENia*, se consume el metamodelo caSPEM *Ecore* con anotaciones y este se encarga de Producir el modelo caSPEM *GMFGen*, este modelo contiene los parámetros para la generación del código de las interfaces Java y las clases de implementación, que permite la gestión de instancias del metamodelo mediante el editor de árbol provisto por la plataforma *EMF*. De forma paralela, *EuGEnia* puede generar los modelos caSPEM *GMFGraph*, caSPEM *GMFTool* y caSPEM *GMFMap* en un solo paso mediante transformaciones de modelo a modelo (*Ecore2GMF*), el modelo caSPEM *GMFGraph* define las figuras que más tarde los elementos del metamodelo tomaran en cuenta para ser representados de manera gráfica, el modelo caSPEM *GMFTool* define los elementos de la paleta del editor, cuyos elementos a través de eventos de selección y edición con el mouse manipulan el modelo y por último el modelo caSPEM *GMFMap* es finalmente quien mapea elementos de definición del diagrama y los asigna con su correspondiente elemento en la paleta, en nuestro caso personalizamos el editor mediante un *script EOL* llamado *Ecore2GMF* (véase el Anexo A), que es ejecutado inmediatamente después que los modelos caSPEM *GMFGraph*, caSPEM *GMFTool* y caSPEM *GMFMap* son generados por Eugenia, donde reorganizamos la paleta de herramientas del modelo

caSPEM *GMFTool* para separar los elementos de contenido y proceso *SPEM*, elementos de contexto y elementos de conocimiento de adaptación de procesos, esta personalización es tediosa si se hiciera de manera manual, además tiende al error si no se manipula bien los modelos, lo cual facilita el desarrollo del prototipo. *GMF* se encarga de generar el modelo caSPEM *GMFGen* consumiendo el modelo caSPEM *GMFMap* generado previamente por *EuGENia*, este modelo agrega información adicional para la generación de código como por ejemplo agregar *plugins* adicionales para manejo de iconos que se incluye en una de las anotaciones en el modelo *Ecore* como parte de la personalización del editor.

4.4.3 Generación de la Herramienta

El modelo caSPEM *GMFGen* a continuación permite generar todo el código del paquete llamado *diagram* en un proyecto *Java* (*plugin*) aparte, el cual posee las clases Java necesaria para la funcionalidad del editor y es quien finalmente lanza el editor gráfico sobre la Plataforma Eclipse. En la figura 4.7 se muestra las tareas para la construcción del prototipo *caSPEMTool* con la herramienta *EuGENia*.

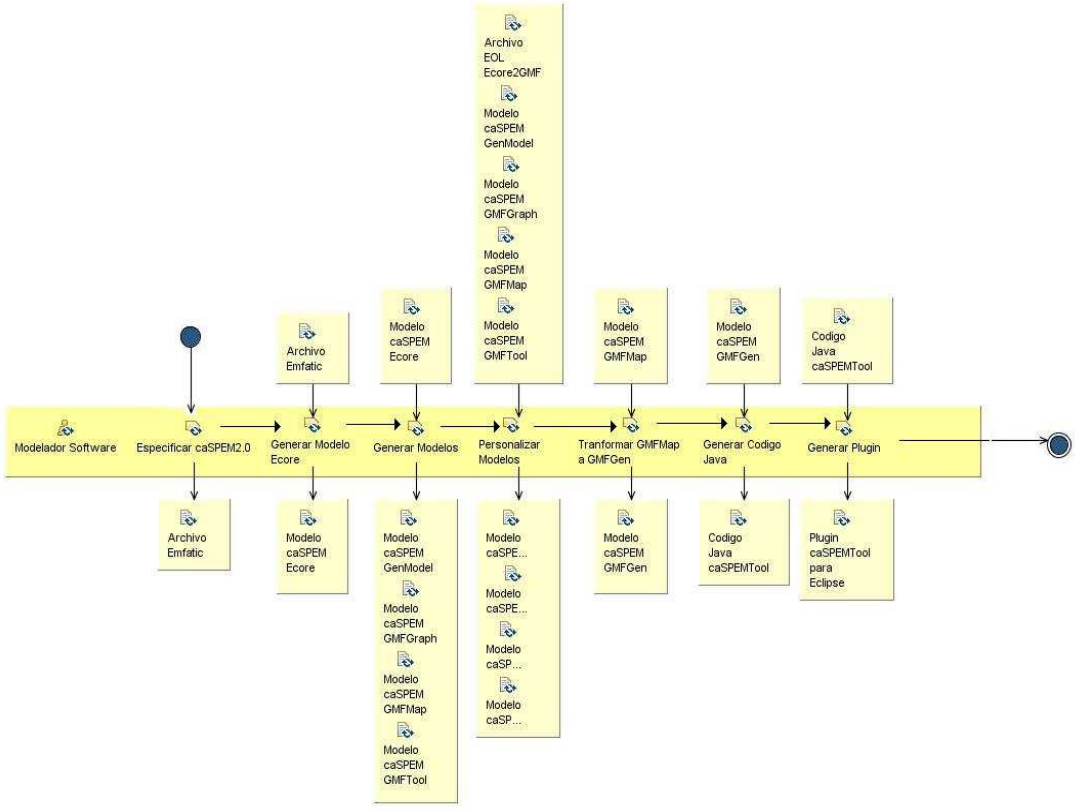

Figura 4.7. Tareas para la Construcción del Modelo GMFGen para el Editor Gráfico caSPEMTool

EClass	
Notación	**Descripción**
gmf.diagram	Es el objeto raíz del metamodelo. Solo se debe agregar a una sola clase y no debe ser abstracta.
gmf.node	Denota que el objeto debe aparecer como un nodo dentro del editor gráfico.
gmf.link	Denota que el objeto debe aparecer como un enlace dentro del editor gráfico
EReference	
gmf.link	Es una relación de asociación dentro de una clase que aparece como un enlace dentro del diagrama
gmf.compartment	Crea un compartimento donde irán los elementos del mismo tipo contenidos por la clase nodo.
EAttribute	
gmf.label	Define etiquetas adicionales para la EClass o EReference que contiene el atributo

Tabla 2. Anotaciones *EuGENia* utilizadas para *caSPEMTool*

Las anotaciones facilitan sobre manera la definición de elementos raíz, nodo, enlace y contenedor, que de otra manera los definiría utilizando el *DashBoard* de *GMF*, haciendo las configuraciones correspondientes en los *Wizards* y que tendría como consecuencia la de invertir tiempo en esta clase de tareas que no aportan a nuestro propósito, sabiendo aún más que el prototipo no sale inmediatamente en la primera iteración, este es el resultado de haber realizado varias veces las tareas descritas en la Figura 4.10.

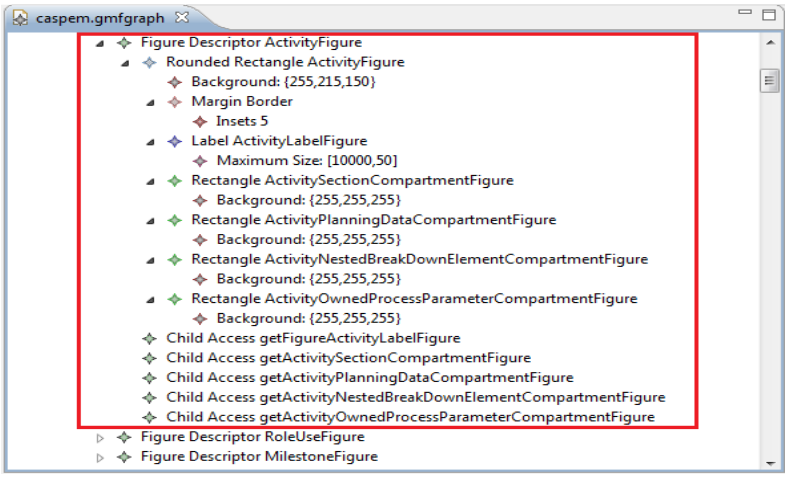

Figura 4.8. Modelo caSPEM2.0 GMFGraph generado por EuGENia

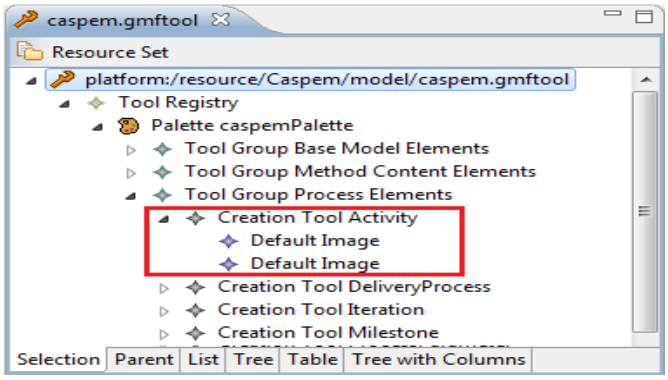

Figura 4.9. Modelo caSPEM2.0 GMFTool generado por EuGENia

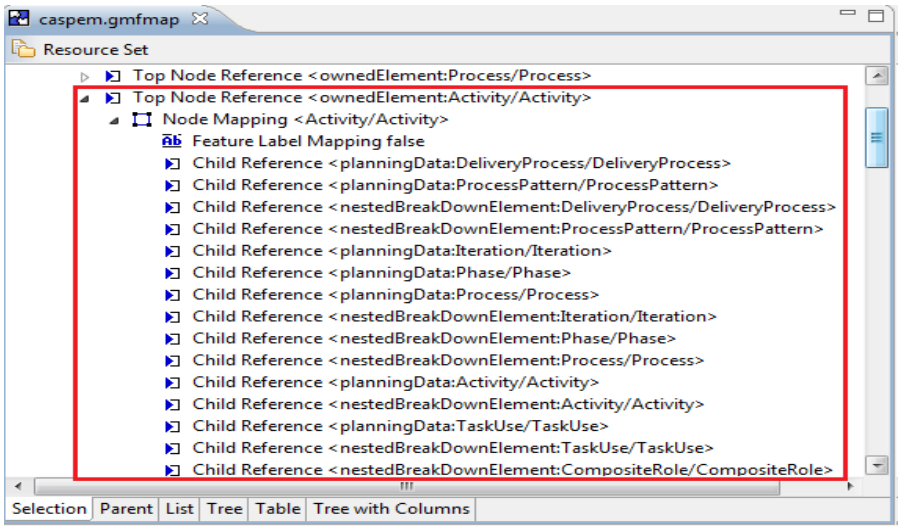

Figura 4.10. Modelo caSPEM2.0 GMFMap generado por EuGENia

Las Figuras 4.8, 4.9 y 4.10 muestran los modelos caSPEM2.0 *GMFGraph*, caSPEM2.0 *GMFTool*, y caSPEM2.0 *GMFMap* respectivamente, en forma de árbol, generados por *EuGENia*, se ha encerrado en cada uno de los modelos un elemento (en este caso *Activity*), a modo de ejemplo para ver la relación del nodo en cada uno de ellos. Como se puede observar en el modelo *GMFGraph* (ubicado a la izquierda de la gráfica) se define el nodo *Activity* como un figura rectangular de esquinas redondeadas y dividido por unas secciones (*Child Access*…) contenedoras de otros elementos gráficos. En el caso del modelo *GMFTool* (ubicado al centro de la gráfica) se define el icono de creación de la paleta para el elemento *Activity* dentro de un grupo llamado *Process Element* el cual es el resultado de una personalización hecha por parte del archivo *Ecore2GMF.EOL* adjuntado a los modelos, dado que por defecto genera una distribución distinta separando los enlaces de los nodos. En el modelo *GMFMap* se observa la definición de qué elementos pueden alojar las

secciones, previamente definidas por el modelo *GMFGraph*, como por ejemplo en la sección *ActivitySectionCompartment*, pueden ser alojados los elementos *Section* y *Step*.

Una vez generado el modelo *GMFGen*, se procede a generar el código Java que da soporte al Editor Gráfico caSPEMTool. Cómo primer paso se abre el modelo *GenModel* con su editor correspondiente, el cual nos permite generar el código java correspondiente, mediante el proceso descrito en [55], el cual nos genera dos proyectos java tipo plugin, que darán soporte básico para gestionar los modelos basados en el metamodelo caSPEM2.0, los tres plugins son básicos para poder ejecutar un editor propio para el metamodelo, este editor tiene la característica de ser muy parecido o es un editor Reflectivo, característico en el ambiente eclipse cuando se modelan y prueban los metamodelos. Como siguiente paso, sobre el modelo GMFGen se abre un menú contextual, que tiene la opción de generar el código correspondiente al Editor Gráfico caSPEMTool, el cual da soporte gráfico de los modelos especificados en el metamodelo caSPEM2.0, el editor simula una pizarra de trabajo y en uno de sus lados se encuentra una paleta con los objetos que pueden modelarse, en este editor es más factible diseñar un regla de adaptación que con el editor reflectivo, dado que la regla formulada puede apreciarse mejor, mientras que para entender una regla con el editor reflectivo, se necesita ser usuario avanzado del metamodelo caSPEM2.0. Para más información sobre la instalación y mantenimiento del prototipo véase el Anexo C

4.4.4 Restricciones de EuGENia para Implementar el Metamodelo

Inicialmente consideramos metamodelar toda la especificación de SPEM, incluyendo la arquitectura de paquetes, pero al final cuando generamos el código, las clases e interfaces java dependientes, no se importaban adecuadamente, para poder corregir este problema y continuar el desarrollo, tomamos cada clase código Java donde aparecían errores de compilación, en cuanto a la líneas de código "import", y cambiamos el código fuente generado por el código adecuado para la corrección. Esta técnica inicialmente ayudo muy bien en las primeras iteraciones, pero a medida que el metamodelo crecía, en cuanto a más paquetes, clases y relaciones, la tarea consumía más tiempo y era tediosa de realizar; por esta razón, para desarrollar un prototipo se tomó la decisión de no incluir la estructura de los paquetes en el metamodelo por ser una restricción de EuGENia, lo cual no implica que no cumpla con la especificación del metamodelo SPEM 2.0, solo es para facilitar su

construcción rápida e incremental, comprobando que Eugenia para meta-modelos de alta complejidad (incluyendo paquetes), no es una opción viable. Dentro del código generado, se crearon clases que no tenemos en cuenta en ningún momento, exploramos la opción de eliminarlas, pero esta acción generó problemas en otras partes del código de la aplicación, que se resuelven manualmente, y a medida resolvíamos unos aparecían nuevos problemas en clases java que si se tienen en cuenta. Esta técnica la llevamos hasta el final, cuando eliminamos una clase Java, pero por el trabajo que cuesta resolver todos y cada una de los errores que se presentan y los nuevos que se generan al eliminar una clase, considerando el tiempo de hacerlo por cada vez que la generación de la herramienta se haga (la herramienta ha sido generada incrementalmente unas 30 veces), se tomó la decisión de mantenerlas y comentar el código apropiado donde la clase no es útil a nuestro propósito.

4.5 Implementación de la funcionalidad de evaluación de los modelos de Proceso en caSPEM2.0

Una vez hecho los pasos comentados anteriormente, creamos un plugin adicional utilizando Epsilon EVL, que permite hacer verificaciones de consistencia de los modelos creados por medio del editor, este se encarga de revisar si los elementos del modelo se encuentran bien conectados, en las parte de las reglas (y más importante) verifica que los valores configurados en el contexto para el proceso, estén acorde a la definición del contexto. Inicialmente en nuestras primeras iteraciones en la construcción del editor, el prototipo resultante lanzaba muchas notificaciones acerca de la inconsistencia de los modelos, esto era porque el meta-modelo presenta multiplicidades no nulas, por ejemplo *0..1*, *1..0*, *1..1*, *1..**, en las relaciones y esta clase de restricciones son verificadas también por defecto por el plugin, por esta razón y debido a que no está dentro de nuestro propósito utilizar todas la restricciones del meta-modelo SPEM tomamos la decisión de modificar las relaciones de multiplicidad no nulas a nulas, por ejemplo cambiar de *1..0* a *0..0*, y así evitar todo un conjunto de notificaciones que en algunos casos no se necesitan validar, dado que algunas relaciones en el meta-modelo se utilizan bajo demanda y no necesariamente son obligatorias para el ingeniero de procesos.

4.6 Algoritmo Traductor de la Especificación Lógica de la Reglas

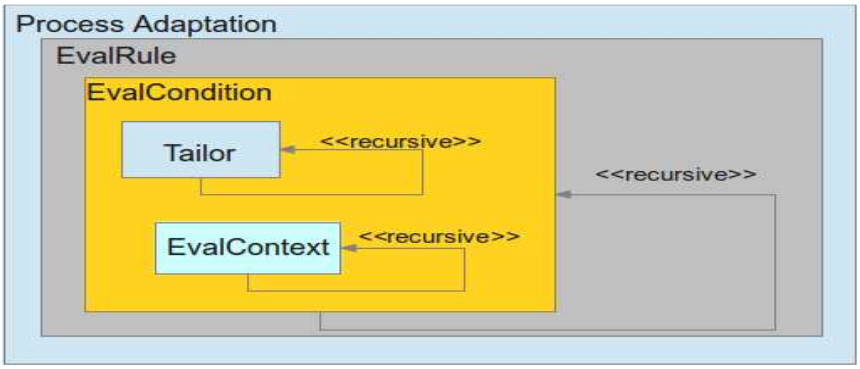

Figura 4.11. Esquema Jerárquico del Algoritmo

El proceso de ejecución gráfico de reglas puede ser ejecutado cuando se tiene configurado el contexto especifico y las reglas de adaptación han sido implementadas con la Especificación Lógica de las Reglas, las cuales se ejecutan sobre el modelo de proceso. Durante la ejecución, el código EOL toma el paquete de reglas seleccionado y empieza a evaluar cada una de las reglas definidas en el paquete de conocimiento, esto lo hace mediante la función *EvalRule*(), la cual tiene por parámetro el contexto específico seleccionado. Al evaluar las reglas de la Especificación Lógica de las Reglas sobre el modelo, el resultado es un modelo de proceso adaptado a las características de contexto específico que el usuario haya configurado previamente. Una vez se evalúan todas las reglas se procede a borrar el paquete de Conocimiento y se almacena el proceso adaptado en un nuevo archivo.

```
Alghorithm EvalRule
  input rule : Rule, context : ContextConfiguration
begin
  var conditions = { s / s e rule.conditions & (s is SimpleCondition | s is
  OrCondition | s is AndCondition) }
  for i = 0 to i < conditions.size() do
          if (isRoot (conditions[i] )) then  // isRoot -> condition without precedences.
                  EvalCondition(context, conditions[i]);
          end if;
          end for;
  end;
```

Figura 4.12.Algoritmo de la función que Evalúa la Regla de Adaptación

En la Figura 4.12 se muestra la función *EvalRule*, la cual establece desde qué condicional ejecutará la regla, es decir, busca el elemento raíz de la regla. Tal

operación fue posible especificar e implementar debido a qué en el script EOL, se ha implementado un buscador de esta clase de nodos, la características de estos nodos es que son del tipo *SimpleCondition*, *AndCondition* u *OrCondition* y dan inicio a la lógica de la regla, dado que estos nodos no son precedidos por ningún otro. Por lo general se pretende modelar reglas que empiecen a ser evaluadas desde una condición inicial (condición que no tiene condiciones precedentes), pero la implementación del prototipo permite la ejecución de reglas con múltiples condicionales iniciales. Una vez obtenidos todos los nodos raíz de la regla, por cada uno, se invoca una función recursiva denominada *EvalCondition*, la cual recibe como parámetro de entrada el contexto del proyecto, y es quien finalmente evalúa los condicionales de la regla.

```
Alghorithm EvalCondition
  input context : ContextConfiguration, self : AbstractCondition
begin
  var eval;
  if ( self  is  Action) then  return Tailor(self);
  else
        if(self is SimpleCondition)  then  eval <-
self.ruleContext.EvalContext(context);
        else
              if(self is And Condition) then
                    eval <- true;
                    for i = 0 to i < self.ruleContext.size() do
                          eval <- (eval and EvalContext(context,
                                self.ruleContext[i]));
                    end for;
              else
                    eval <- false;
                    for i = 0 to i < self.ruleContext.size() do
                          eval <- (eval or EvalContext(context,
                                self.ruleContext[i]));
                    end for;
              end if;
        end if;
  end if;
  if (eval = true ) then return YesStatement.EvalCondition(context, self);
  else return NoStatement.EvalCondition(context, self.);
  end if;
end;
```

Figura 4.13. Algoritmo de la Función *EvalCondition*

En la Figura 4.13 se muestra la función llamada *EvalCondition()*, el cual evalúa la lógica modelada en la regla, como primer paso se evalúa si el nodo es del tipo

Action, dado que éste es el encargado de representar las acciones de adaptación del proceso software, por lo tanto inmediatamente el algoritmo reconoce este tipo de nodo, invoca a una función recursiva denominada *Tailor()* (véase la figura 4.15), la cual toma los elementos modelados en el nodo *Action* (*TailorElement*, *LinkedElement*, etc.) y ejecuta la acción de adaptación. Una vez termine la acción consulta si en el atributo *NextAction* del nodo *Action* se ha definido otra acción, de ser afirmativa para el elemento NextAction se invoca de nuevo la función *Tailor()*, de ser negativa finaliza el algoritmo. Si el elemento no es del tipo *Action*, se consulta si es del tipo *SimpleCondition*, este paso se realiza debido a que una *SimpleCondition* difiere de una *AndCondition* o una *OrCondition*, porque un elemento de tipo *SimpleCondition* solo contiene un objeto *RuleContext* (operando), en el caso de ser *AndCondition* u *OrCondition* van a tener como mínimo dos *RuleContext*, es por ello que el comportamiento del algoritmo cambia dependiendo del tipo de condicional.

```
Alghorithm EvalContext
  input context : ContextConfiguration, self : RuleContext
begin
  var eval = false;
  for i=0 to i<context.myContextAtributeConfiguration.size() do
        if(context.myContextAtributeConfiguration[i].myContextElement =
                                          self.ContextAtributeElement) then
              for j=0 to j<self.ContextAtributeValue.size() do
                    if (context.myContextAtributeConfiguration[i].
                          myContextAtributeValue =
                                      self.ContextAtributeValue[i]) then
                          eval = true;
                          j = self.ContextAtributeValue.size();
                    end if;
              end for;
        end if;
        if (eval = true) then i = context.myContextAtributeConfiguration.size();
        end if;
  end for;
  return eval;
end;
```

Figura 4.14. Función *EvalContext*.

A continuación, por ambos lados del condicional, se invoca a la función *EvalContext()* (véase figura 4.14), qué es la encargada de verificar la coincidencia de la regla modelada con el contexto configurado y devuelve verdadero o falso si coinciden, como se puede observar en la figura 4.13 si es un condicional de tipo *SimpleCondition* la función se invocará una vez, en otro caso (*AndCondition* u

OrCondition) la función se invocará por cada *RuleContext* definido en la regla. Independiente de cuál de los casos sea, este resultado es guardado en una variable booleana, quien define el nuevo rumbo del flujo, de ser verdadera ("true") dirigirá el flujo del algoritmo hacia la condicional o la acción que referencia el atributo *YesStatement*, ó de ser falsa lo dirigirá hacia la condicional que referencia el atributo *NoStatement*, cabe aclarar que *YesStatement* y *NoStatement* pueden referenciar a elementos del tipo *SimpleCondition*, *AndCondition, OrCondition* o *Action*, cualquiera sea el camino que la regla haya demarcado, este invocará recursivamente a *EvalCondition()* sobre la condicional referenciada, una vez sean evaluadas las condicionales de la regla el flujo termina.

```
Alghorithm Tailor
        input self : AbstractCondition
begin
        if ( self.Action is "Update") then UpdateTailor(self);
        else
                if (self.Action is "Delete") then
                        ...
                        // remove self.TailorElement from Process
                        ...
                else
                        if (self.Action is "Update_Variability") then
                                VariabilitySPEM(self);
                end if;
        end if;

        return Tailor(self.NextAction);
end;
```

Figura 4.15. Función *Tailor*

Como característica general del algoritmo observamos que la función *EvalCondition()* dejará de llamarse así misma cuando esta sea invocada sobre un elemento del tipo *Action*, es aquí donde se le da el paso otra función recursiva, mencionada anteriormente "*Tailor*", que dependiendo del valor que se ha modelado en el atributo *Action* (*update*, *delete*, *update_variability*) del elemento *Action*(nodo), invocará la función *UpdateTailor* en el caso de definir el valor como "*update*" y la función *VariabilitySPEM* en el caso de definir el valor como "*update_variability*", si se define el valor como "*delete*" la función *Tailor()* hará las acciones respectivas para realizar la adaptación.

Para estos tres casos dentro del Script EOL, se ha programado la algoritmia suficiente para llevar a cabo el comportamiento de las acciones de adaptación, para

el caso de ser "*update*" realizará la asignación del elemento escogido en *LinkedElement* en el atributo que corresponda con el elemento escogido en *TailorElement*, por otro lado en el caso de ser "*delete*" el algoritmo borrará el elemento escogido en *TailorElement* del modelo de proceso. Por último de ser "*update_variability*" aplicará la variabilidad que la especificación SPEM 2.0 define, de no definirse una regla que adapte un proceso de acuerdo a la variabilidad definida en el modelo de proceso, el algoritmo borrará automáticamente los elementos del proceso en donde se ha modelado puntos de variación y que no han sido tomados en cuenta en el momento de la adaptación.

Una vez se ejecuten todas estas adaptaciones definidas en las reglas, el algoritmo entregará un proceso software adaptado al contexto especifico, el cual el ingeniero de software ha sido definido en la configuración de contexto, el proceso software se almacena en un archivo aparte con el sufijo "*Tailor*", a partir de este archivo el proceso adaptado se podrá ver en una vista de árbol con el editor reflexivo que provee la plataforma Eclipse o si lo prefiere el prototipo permite generar un diagrama a partir de él, que brinda un vista gráfica del proceso visualizable por caSPEMTool. Para ver la implementación completa del algoritmo véase el Anexo B.

4.7 Proceso de Ejecución de las Reglas de Transformación

Para ejecutar la Especificación Lógica de las Reglas, que el ingeniero de procesos ha diseñado, el prototipo hace uso de un script EOL con las funciones explicadas en la sección 4.7, además de las expresiones en lenguaje EOL que solicitan al usuario el paquete de conocimiento y la configuración de contexto específica que rodea un proyecto. Una vez estos datos de entrada son provistos dará inicio a la adaptación de procesos software, convirtiendo toda la lógica diseñada por el ingeniero de procesos en acciones de transformación sobre el proceso general dando como resultado un proceso adaptado. En la figura 4.16 se presenta el pseudocódigo del algoritmo encargado de la adaptación del proceso, que como datos de entrada son requeridos el conjunto de reglas que el diseñador de procesos desea ejecutar y la configuración del contexto del proyecto (representa el contexto del proyecto). Las reglas son evaluadas una por una, y al final el algoritmo borra los paquetes de conocimiento, elementos con variabilidad y almacena el proceso

entregable adaptado a las características del proyecto, pasos que en el algoritmo se representan por las tres últimas funciones, que solo indican los pasos que se deben seguir una vez las reglas de adaptación son ejecutadas y el proceso adaptado quede consistente.

```
Alghorithm ProcessAdaptation
  input knowledgePackage : KnowledgePackage, contex:
                                ContextConfiguration
begin
  for i=0 to i<knowledgePackage.Rules.size() do
    EvalRule(knowledgePackage.Rules[i], context);
  end for;
  removeAllKnowledgePackage();
  removeVariabilityElements();
  saveAdaptedProcess();
end;
```

Figura 4.16. Algoritmo de Adaptación de Procesos Software.

4.8 Instanciación del Metamodelo en el Caso CC51A-RE. Caso de Prueba

El caso CC51A-RE es un diseño de un proceso de ingeniería de requerimientos de un curso avanzado de estudiantes de Ciencias de la Computación de la Universidad de Chile, el cual ha sido previamente implementado con eSPEM, SPCM 1.0 y ATL, como caso de prueba del metamodelo y su especificación por medio de caSPEMTool. Un proceso de Ingeniería de Requerimientos incluye tres componentes principales: la Exploración, la especificación y validación de Requerimientos de Usuario, y la Especificación y Validación de Requerimientos Software.

La Actividad de *Exploración* esté encargada de identificar el problema del negocio a resolver y el contexto del proyecto. La actividad *Especificación y Validación de Requerimientos de Usuario* es la encargada de establecer los objetivos del proyecto y alcance del mismo en términos de requerimientos de usuario. Ésta actividad puede utilizar prototipos simples con el fin de ayudar a clientes y miembros del equipo de desarrollo a aceptar los requerimientos de usuario y concebir una solución abstracta. Por último, la actividad *Especificación y Validación de*

Requerimientos Software es la encargada de convertir los requerimientos de usuario en uno o más requerimientos software. Esta actividad produce una lista detallada que representa los requerimientos software para el proyecto. El resultado de esta actividad debe ser complementado con una validación de parte de los usuarios y clientes, para comprobar que la elicitación hecha por los miembros del equipo de desarrollo satisfaga las necesidades de los usuarios.

Algunos de los componentes de proceso pueden variar de acuerdo a las características del contexto del proyecto, por ejemplo, la actividad *Exploración* es *Opcional* porque solo se realiza si se trata de un nuevo proyecto. Por otro lado, la actividad *Prototipado Operacional*, (sub-actividad de *Especificación y Validación de Requerimientos Software*) es *Opcional* ya que debe ser realizada solamente para dominios de aplicación *No Conocidos*. Además, la tarea *validación de requerimientos software* puede tener comportamientos diferentes dependiendo de la familiaridad que tengan los miembros del equipo con el dominio de aplicación del proyecto, en este caso las alternativas son: una *Validación Interna* cuando el dominio de aplicación es conocido o una *Validación Basada en Prototipo* en caso contrario.

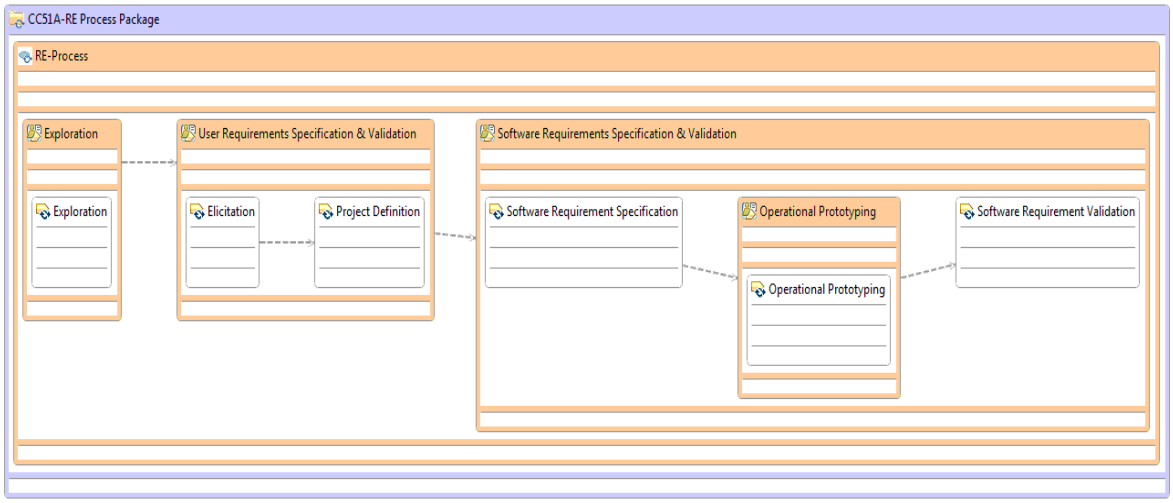

Figura 4.17. Proceso de Ingeniería de Requerimientos General

En los párrafos anteriores de esta sección, describen de manera implícita como la información de contexto afecta la realización o no de ciertas actividades del proceso, con esta información se puede modelar el contexto dentro del cual el proceso de ingeniería de requerimientos puede variar, luego la información de contexto del proyecto se traduce en valores contextuales, cuyo conjunto de valores

se denomina configuración de contexto (Context Configuration según la Figura 4.18), y son parámetros para tener en cuenta al momento de adaptar el proceso.

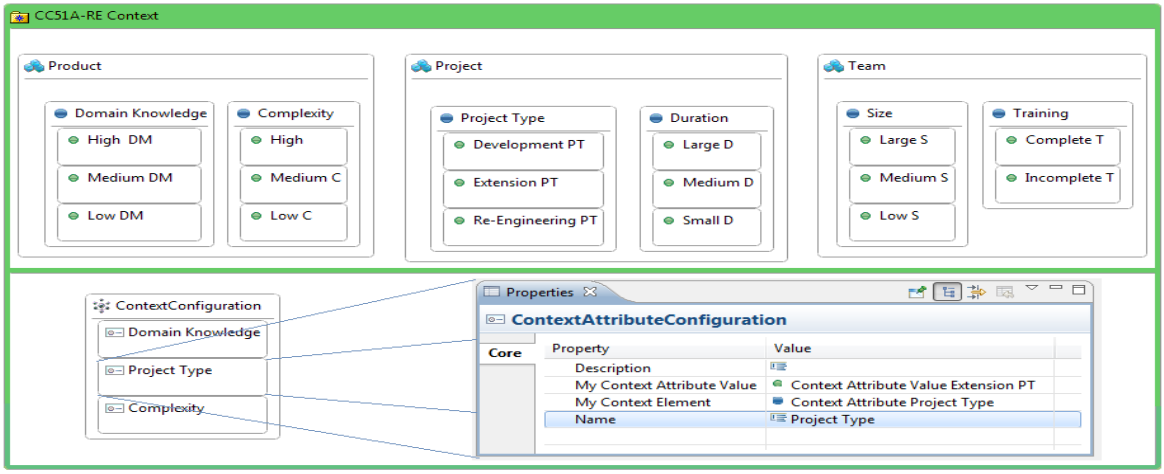

Figura 4.18. Estructura del Contexto Modelado y su Configuración

Una regla de adaptación tiene como propósito transformar el proceso de ingeniería de requerimientos general en un proceso específico teniendo en cuenta la configuración única del contexto del proyecto. Estas reglas están compuestas de operadores condicionales de disyunción y conjunción que comparan el contexto deseado con el contexto específico a cada proyecto y que desencadenan acciones de adaptación sobre elementos del proceso. Como ejemplo, la actividad *Exploración* será eliminada cuando el *Tipo de Proyecto* sea una *Extensión*. Similarmente, la actividad *Prototipado Operacional* será eliminada cuando el Dominio de *Conocimiento* sea distinto de *Bajo,* ya que los miembros del equipo tienen familiaridad con el dominio de la aplicación. En la Figura 4.19 se muestran los elementos del paquete de reglas modelados en el prototipo caSPEMTool.

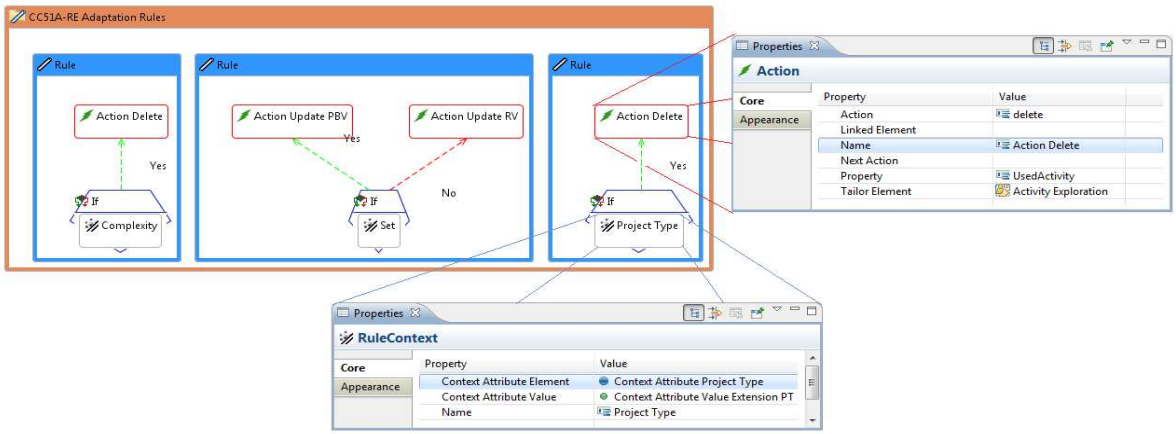

Figura 4.19. Reglas de Adaptación

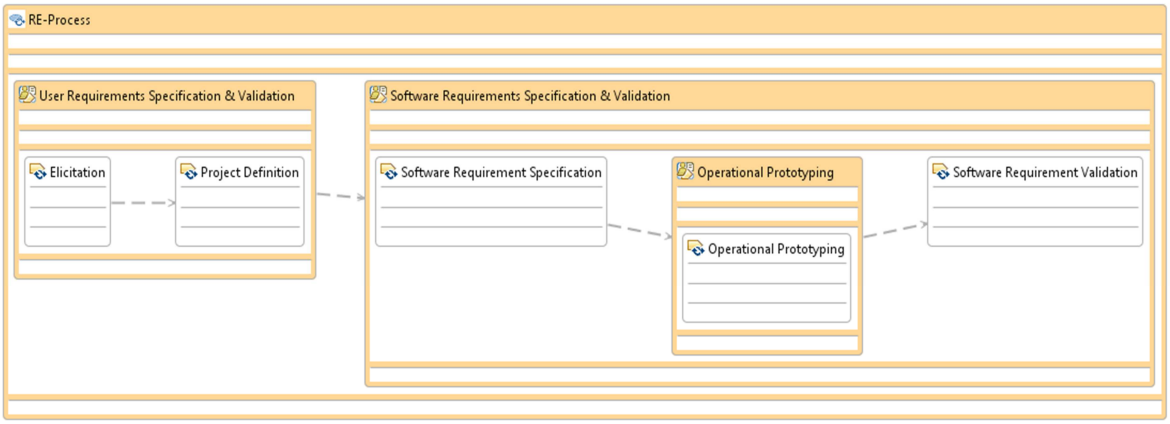

Figura 4.20. Proceso Adaptado

La Figura 4.20 muestra el proceso adaptado teniendo en cuentas las reglas de adaptación expresadas el paquete de conocimiento. Debido a que el tipo de proyecto es una extensión la actividad de exploración ha sido eliminada del proceso. El proceso de software previamente modelado puede ser visto en el Anexo D.

4.9 Síntesis y Discusión

En este capítulo se ha presentado el prototipo caSPEMTool el cual constituye una implementación del metamodelo caSPEM2.0. Se ha utilizado la herramienta EuGENia para el desarrollo del prototipo. La utilización de EuGENia permite generar automáticamente el editor gráfico y personalizarlo mediante agregar scripts, que en tiempo de ejecución y automáticamente permiten realizar los cambios deseados.

5 Estudio de Caso: Modelado del Proceso APF – Solución Técnica

En este capítulo se reporta una aplicación empírica para evaluar el diseño de un proceso real de software mediante caSPEMTool por un grupo selecto de ingenieros de procesos. El proceso de desarrollo pertenece a una pequeña empresa de software chilena, llamada Amisoft. Con el fin de comprobar que el metamodelo caSPEM2.0 y su herramienta de soporte disminuyen la complejidad técnica de definir la lógica de adaptación y permite hacer un mejor análisis de la lógica de adaptación al incluir información de contexto referente a la organización o a un proyecto en específico.

5.1 Metodología

Según lo descrito por Runeson et al. [65], el estudio de caso es una metodología de investigación adecuada para la ingeniería del software debido a que estudia un fenómeno contemporáneo en su contexto real, buscando mantener la integridad y las características significativas de los eventos, y es ejecutado cuando el investigador tiene poco control sobre los eventos y cuando los sujeto de estudio son más fáciles de observar en grupo que de manera aislada.

Para la conducción de este caso de estudio ha sido necesario seguir los siguientes pasos:

- **Diseño**: el estudio de caso es establecido para comprobar si el metamodelo caSPEM2.0 y su herramienta de soporte decrementan la complejidad técnica para modelar procesos adaptables. Alrededor de éste objetivo se determinó el tipo de caso de estudio, su unidad de análisis, se seleccionó el caso, se diseñaron indicadores, métricas e

instrumentos. Así mismo se determinaron los aspectos de recolección, análisis y reporte del caso de estudio.

- **Preparación**: Se han diseñado los instrumentos para desarrollar el caso: capacitación, documentación del proceso, protocolo de observación y los instrumentos de recolección (Encuesta, Planilla de Tiempos, Planilla de Observaciones, preguntas abiertas).
- **Ejecución y Recolección de información**: el caso fue desarrollado en una sesión de cinco horas con tres participantes. En una hora se explicaron los conceptos y los elementos extendidos al metamodelo, de igual manera se realizó una pequeña demostración del prototipo, después se procedió a darles una guía de desarrollo, la cual incluía el proceso de desarrollo, y la información informal sobre su adaptación. De forma independiente cada ingeniero de procesos usando el prototipo diseñó el proceso junto con la lógica de adaptación, mientras el protocolo de observación fue seguido por los miembros del equipo.
- **Análisis**: Se realizó un análisis de datos de tipo cuantitativo y cualitativo teniendo en cuenta la información obtenida de la observación de los sujetos investigados. Además, se obtuvo datos de la encuesta y las recomendaciones hechas por los sujetos investigados.
- **Reporte:** los resultados del estudio de caso fueron reportados y hacen parte de éste documento.

5.2 Pregunta de Investigación

caSPEMTool es un prototipo que se ha implementado teniendo en cuenta la especificación caSPEM2.0, con el fin de definir y adaptar procesos software basados en el contexto, buscando resolver el problema de la adaptación manual, la cual consume recursos de tiempo, humanos y por consiguiente económicos, siendo además propensa al error [4, 5, 13, 14]. Esto condujo a formular la siguiente pregunta: ¿El metamodelo caSPEM2.0, a través de su instrumentación en caSPEMTool, reduce la complejidad técnica y aumenta la eficiencia en el proceso de adaptación de procesos software? La comparación viene restringida a SPEM 2.0, por ser el lenguaje estándar de referencia y la propuesta de adaptación sistemática actual propuestas por Hurtado et al.[12, 68] y Ruiz et al [69], debido a que son los

trabajos relacionados más cercanos y completos reportados por la literatura. Respondiendo a esta pregunta se pretende aportar a responder la pregunta de investigación de este proyecto de tesis, esto es "¿Cómo facilitar la aplicación práctica de la adaptación de modelos de proceso basada en el uso de información contextual sin agregar una complejidad significativa al modelo de proceso?

5.3 Objetivo del Estudio de Caso

El objetivo de este caso de estudio es validar el metamodelo y su instrumentalización a través del diseño práctico en un proceso software de una organización, analizando los resultados en la definición del proceso y de su lógica de adaptación con tres ingenieros de sistemas con experiencia en la formalización de procesos, y comparándolos con los resultados presentados por trabajos previos presentados por Hurtado et al. [12, 68] y Ruiz et al.[69].

5.4 Selección del Estudio de Caso

Debido a que la práctica de la definición de procesos adaptables es desarrollada por un ingeniero de procesos, son ellos los indicados para evaluar la complejidad y practicidad de la propuesta. Así, la unidad de análisis es el proyecto de modelado utilizando el metamodelo caSPEM2.0 y las fuentes de información primarias son los ingenieros de procesos responsables del modelado, su selección respondió a criterios de disponibilidad [70] de ingenieros con conocimientos en SPEM, debido a que la propuesta aborda el tema de extensión de este metamodelo. De acuerdo a Benbasat et al. [70], el caso de estudio es de tipo ***Holístico*** considerando una unidad de análisis con tres sujetos de investigación, el modelo de proceso seleccionado fue debido a su ***tipo***[15] (caso real en la industria) y por ser ***revelatorio***[16] (es un caso lo suficientemente completo para evaluar la aplicabilidad de la propuesta).

[15] Un caso de estudio típico tiene como objetivo capturar las circunstancias y condiciones de situaciones comunes.
[16] Un caso de estudio revelatorio existe cuando un investigador tiene la oportunidad de observar y analizar un fenómeno previamente inaccesible a la investigación científica.

5.5 Contexto del Estudio de Caso

5.5.1 La organización

Amisoft es una compañía de desarrollo de software que se enfoca sobre el desarrollo de sistemas de información jurídica y está considerada como una PyME [71], ya que tiene alrededor de 50 empleados. Esta compañía está especializada en proveer servicios de tecnología de la información y acompañamiento en los procesos críticos de negocios propios y de sus clientes, con un equipo de profesionales con sólida formación académica y avanzado perfil técnico. Desde el 2009, esta compañía tiene su propio proceso definido, para desarrollar software a medida, denominado Amisoft Process Framework (AFP), el cual implementa el estándar ISO9001:2008, con las mejores prácticas de UP (Unified Process) y están actualmente en el proceso de una evaluación CMMI de nivel 2.

5.5.2 El caso y sus sujetos de investigación

El estudio de caso es de tipo holístico [66] considerando una unidad de análisis - el modelado del proceso, y tres fuentes primarias de información - los ingenieros de sistemas con alguna experiencia en formalización de procesos con SPEM 2.0, modelando en forma paralela y simultánea el mismo proceso software industrial. El estudio de caso, de acuerdo a su propósito es positivista [67], debido a que busca probar una mayor practicidad de un nuevo mecanismo en caSPEM2.0, respecto a lo reportado por el estado del arte.

A fin de aplicar la definición de procesos, incluyendo la lógica de adaptación del proceso Amisoft, se organizó el caso como una sesión de trabajo con tres ingenieros de sistemas con experiencia en definición de procesos software y conocimientos básicos sobre la especificación de SPEM 2.0, con fines de hacer una evaluación empírica del metamodelo caSPEM2.0 a través de su prototipo caSPEMTool.

De los tres sujetos investigados se debe resaltar que dos de ellos tienen experiencia en la especificación de procesos y un conocimiento adecuado del metamodelo SPEM 2.0, ambos poseen experiencia en la adaptación de proceso, el otro sujeto de investigación tiene un conocimiento básico tanto en la especificación de procesos, su adaptación y SPEM 2.0

5.5.3 El proceso de desarrollo del caso

5.5.3.1 Modelo del Proceso Organizacional

El proceso organizacional de Amisoft tiene en cuenta, en principio dos tipos de proyectos: mantenimiento y desarrollo. En este caso se aborda solamente la especificación del área de proceso *Solución Técnica* de la compañía, la cual consiste de tres actividades secuenciales *Diseño de la Arquitectura, Análisis y Diseño* e *Implementación*.

La actividad *Diseño de la Arquitectura* es una actividad opcional, que no es requerida para proyectos de mantenimiento. La actividad *Análisis y Diseño* involucra cinco tareas: *Realización del Análisis*, *Análisis de Re-Uso*, *Diseño de Componentes*, *Diseño de Interfaces y Comunicación*, *Diseño de la Base de Datos.* Dentro de la tarea *Realización del Análisis* los ingenieros determinan qué componentes deben hacer parte de la solución; durante la tarea *Análisis de Re-Uso* ellos identifican cuáles componentes pueden reutilizar de proyectos previos, estas dos actividades son opcionales cuando el proyecto es mantenimiento. La tarea *Diseño de Componentes* trata de modelar los componentes que aún no han sido diseñados; la tarea *Diseño de Interfaces y Comunicación* trata con aspectos relacionados a la usabilidad y diseño del producto software y la comunicación con otros sistemas, es una tarea opcional cuando los proyectos son de mantenimiento. Puesto que la compañía trabaja con sistemas de información la tarea *Diseño de la Base de Datos* es obligatoria y se enfoca sobre el modelo de la base de datos.

Finalmente la Actividad de *Implementación* consiste de tres Actividades secuenciales: *Desarrollo de los componentes*, *Desarrollo de la Base de Datos* y *Desarrollo de las Interfaces y Comunicación*. Cada una de estas actividades tiene asociadas unas tareas; para la actividad *Desarrollo de los Componentes* deben realizarse dos tareas: *Desarrollo de Componentes* y *Pruebas Unitarias de los Componentes,* para la actividad *Desarrollo de la Base de Datos* deben realizarse dos tareas: *Desarrollo de la Base de Datos* y *Pruebas Unitarias a la Base de Datos*; para la actividad *Desarrollo de Interfaces y Comunicación* deben ejecutarse dos tareas: *Desarrollo de Interfaces y Comunicación* y *Pruebas de Unidad de las Interfaces y Comunicación.* Durante estas tareas los roles encargados deben implementar los diseños producidos en la actividad *Análisis y Diseño.*

5.5.3.2 Modelo del Contexto

Cuatro *Dimensiones* distintas fueron identificadas por la organización para agrupar las características que afectan los proyectos de desarrollo que ellos manejan: Equipo, Proyecto, Negocio y Producto. La *dimensión Equipo* tiene dos atributos asociados: *Conocimiento Técnico* el cual puede tomar tres valores (*Bajo*, *Medio* o *Alto*) y *Habilidades de Desarrollo* el cual puede tomar dos valores (*Malo* o *Bueno*). La *dimensión Proyecto* tiene asociada dos atributos *Tipo de Proyecto* el cual puede tomar dos posibles valores (*Mantenimiento* o *Desarrollo*) y *Disponibilidad de Recursos* el cual puede tomar dos valores (*Alta* o *Baja*). Dentro de la *dimensión Negocio* tiene asociada un atributo *Cliente* el cual puede tomar dos posibles valores (*Proceso orientado* o *Proceso no Orientado*). Finalmente la *dimensión Producto* considera solamente un atributo: *Integración* indicando si el producto debe ser integrado a una solución existente o no.

5.5.3.3 Especificación de la Adaptación

A continuación se especifican las adaptaciones expresadas en lenguaje natural sobre el proceso software al a ser adaptado.

***Adaptación 1*:**

Si el Conocimiento Técnico no es bajo entonces debe **Considerarse** la Actividad *Diseño de la Arquitectura*, en caso contrario, se debe evaluar el *Tipo de Proyecto*, si este es un proyecto de Desarrollo, entonces también debe ser considerada esta Tarea.

***Adaptación 2*:**

La tarea *Diseño de Interfaces y Comunicación* debe **considerarse** si dentro del proyecto existe Integración. La Actividad *Desarrollo de Interface y Comunicación* es seleccionada siempre y cuando la tarea *Diseño de Interface y Comunicación* es seleccionada también.

***Adaptación 3*:**

La tarea *Realización de Análisis* debe ser **considerada** si la Disponibilidad de Recursos no es Baja o en el caso contrario, donde el *Nivel de Habilidades* de desarrollo del equipo es malo y el *Tipo de Proyecto* es mantenimiento o desarrollo y el *Cliente* no es orientado a procesos

***Adaptación 4*:**

Si el *Tipo de Proyecto* es un Desarrollo entonces debe **considerarse** usar la tarea *Análisis de Re-uso* en caso contrario no.

***Adaptación 5*:**

Si el *Nivel de Habilidades* de desarrollo del equipo es Bueno y el *Tipo de Proyecto* es de Desarrollo entonces **asociar** una definición de tarea Desarrollo de Componentes y Programas Dirigido por *Pruebas* a la tarea *Unidad de Pruebas de Componente y Programas* de lo contrario no.

5.6 Indicadores y Métricas

Para evaluar de manera objetiva éste caso, particularmente, para dar respuesta a la pregunta de investigación fue necesario definir un conjunto de métricas e indicadores. La tabla 3 muestra un resume de los indicadores y métricas identificados.

Pregunta de investigación	Indicadores	Métricas	Instrumentos
¿El metamodelo caSPEM2.0, a través de su instrumentación en caSPEMTool, reduce la complejidad técnica y aumenta la eficiencia en el proceso de adaptación de procesos software?	Complejidad	Complejidad Percibida por los ingenieros del proceso asociada a la extensión de SPEM 2.0. Complejidad percibida por los ingenieros de proceso al especificar la lógica de adaptación.	Encuesta. Protocolo de Observación
	Efectividad	Número de Elementos de proceso, contexto y reglas modelados. Número de elementos de proceso, contexto y reglas modelados correctamente.	Modelos de Proceso caSPEM2.0 de Amisoft. Protocolo de Observación
	Eficiencia	Número de Elementos del proceso, contexto y reglas modelados correctamente. Esfuerzo requerido al modelar.	Modelos de Proceso caSPEM2.0 de Amisoft. Protocolo de Observación
	Usabilidad	Facilidad de uso de los elementos del metamodelo caSPEM	Encuesta. Protocolo de Observación

		percibida por los ingenieros de proceso. Facilidad de aprendizaje de los nuevos elementos de caSPEM percibida por los ingenieros de proceso.	

Tabla 3. Indicadores y Métricas

A continuación se describen en detalle los indicadores y la forma en qué estos son calculados a través de las métricas identificadas:

Complejidad: la complejidad se define como la diversidad de elementos que componen una situación, los cuales se encuentran entrelazados y/o interconectados que contiene información adicional y oculta al observador. La fórmula que hemos definido para hacer el cálculo de la complejidad percibida está dada por:

$$c = 1 - \frac{\sum_{i=1}^{n} \frac{RE_i}{RD_i}}{n}$$

Donde *c* es la complejidad, 1 es el valor más alto que puede tomar la complejidad, RE_i es el resultado evaluado en la encuesta para la pregunta *i*, el cual puede tomar valores de uno a cinco, RD_i es el resultado deseado en la encuesta a la pregunta *i*, cuyo resultado esperado es 5, *n* es el número de preguntas evaluadas en la encuesta por los ingenieros.

Efectividad: La efectividad se define como el grado en que se producen los resultados esperados. La relación entre los resultados previstos y no previstos y los objetivos. La fórmula que hemos definido para hacer el cálculo de la efectividad es:

$$ef_{i\,j} = \frac{C_{i\,j}}{T_{i\,j}}$$

Donde *ef* es la *Efectividad, C* es la cantidad de elementos diseñados correctamente*, T* es el *total de elementos de la guía esperados a ser modelados, i* ∈ {los elementos del proceso, elementos del contexto, lógica de adaptación}, *j* ∈ {primera unidad, segunda unidad, tercera unidad}.

Eficiencia: La eficiencia se define como el grado en que se cumplen los objetivos al menor costo posible. Siendo en este caso de estudio el menor costo posible de tiempo, dado que el recurso es constante. La fórmula que hemos definido para hacer el cálculo de la eficiencia es:

$$efc_{ij} = \frac{c_{i\,j}}{t_{i\,j}}$$

Donde *efc* indica la *eficiencia, c* es la cantidad de elementos diseñados correctamente (*correctitud*), *t* es el tiempo en horas utilizado para su diseño, *i* ∈ {los elementos del proceso, elementos del contexto, lógica de adaptación}, *j* ∈ {primera unidad, segunda unidad, tercera unidad}.

Usabilidad: la usabilidad se ha definido como el grado en que un producto puede ser usado por determinados usuarios para lograr sus propósitos con eficacia, eficiencia y satisfacción en un contexto de uso específico, siendo en este caso la extensión del metamodelo SPEM 2.0 para adaptar procesos de desarrollo software basados en el contexto de un proyecto u organización específicos. La fórmula que hemos definido para hacer el cálculo de la usabilidad percibida es:

$$us = \frac{\sum_{i=1}^{n} \frac{RE_i}{RD_i}}{n}$$

Donde *us* es la *usabilidad*, RE_i es el resultado evaluado para la pregunta *i,* el cual puede tomar valores de uno a cinco, RD_i es el resultado deseado en la encuesta a la pregunta *i*, cuyo resultado esperado es 5, *n* es el número de ítems evaluados en la encuesta por los ingenieros.

5.7 Ejecución del Estudio de Caso

En una sesión de una hora contando con los tres ingenieros de procesos, se hizo una introducción de los elementos del metamodelo que harían parte caSPEM2.0, ya que los ingenieros de procesos poseían un conocimiento básico sobre la especificación de SPEM 2.0; además, se dio paso a explicar cómo estaban distribuidos los elementos dentro del prototipo y cómo era su funcionamiento. Al iniciar la segunda hora se les entregó el proceso en forma textual sin formalizar y se procedió a la observación del diseño del proceso por parte de los ingenieros dentro de la herramienta caSPEMTool, de igual forma se registraron los tiempos en cada etapa del proceso de acuerdo al protocolo establecido. Como en este caso se hizo una capacitación muy corta, en todo el proceso en donde los ingenieros diseñaban el proceso de software se hizo un acompañamiento y observación de las reacciones ante el prototipo durante la aplicación de la guía del proceso. Se midieron los tiempos y se anotaron las observaciones de acuerdo a los protocolos y las plantillas establecidos. Durante toda la sesión se hizo acompañamiento y se resolvieron dudas que surgieron al momento de aplicar la guía con la información del proceso a diseñar en el prototipo caSPEMTool. Al final de la sesión se entregó una encuesta a los sujetos investigados. Al finalizar, a cada unidad de análisis se le entregó una

encuesta con el fin de obtener información cuantitativa, cualitativa y observaciones del prototipo de acuerdo al diseño del caso.

5.8 Resultados

5.8.1 Resultados Cuantitativos

Mediciones directas:

A continuación se muestra el registro de los datos tomados en el caso donde los sujetos investigados diseñaron el proceso de software y la lógica de adaptación basándose en la guía provista, la medida de tiempo ha sido tomada en horas. La tabla 3 muestra la información del tiempo que cada unidad utilizó para diseñar los elementos del proceso.

ítems Modelados	Unidad de Análisis (Tiempo en horas)			
	Sujeto 1	Sujeto 2	Sujeto 3	Promedio
Proceso	1.92	2.2	2.75	2.29
Contexto	0.25	0.17	0,17	0.19
Lógica de Adaptación	1.42	1.33	1.5	1.42
Totales	3.59	3.7	4.42	3.9

Tabla 4. Registro del tiempo empleado por los ingenieros de procesos para diseñar la guía.

La tabla 4 muestra la cantidad de elementos del proceso y de la especificación de la lógica de reglas totales para ser diseñados, teniendo en cuenta la guía entregada a los ingenieros de procesos. En el anexo I puede observar los modelos de cada uno de los sujetos investigados.

ítems Totales a Modelar	
Elementos de Proceso	147
Elementos de Contexto	41
Reglas de Adaptación	5

Tabla 5. Número total de elementos dados en la guía para ser diseñados en caSPEMTool.

La tabla 5 muestra la cantidad de los elementos del proceso y de la especificación lógica de las reglas modelados de manera correcta por los ingenieros de proceso en caSPEMTool.

Ítems Modelados Correctamente	Unidad de Análisis			
	Sujeto 1	Sujeto 2	Sujeto 3	Promedio
Elementos de	137	145	138	140

Proceso				
Elementos de Contexto	41	41	41	41
Reglas de Adaptación	5	3	3	3.67
Totales	183	198	182	184.67

Tabla 6. Número de elementos correctamente modelados por los ingenieros de procesos en caSPEMTool.

Complejidad:

Teniendo en cuenta la encuesta entregada al finalizar la sesión de diseño del proceso por parte de los ingenieros de procesos, se analizó los datos recogidos con el fin de obtener un valor cuantitativo de la complejidad al diseñar un proceso y la lógica de reglas necesaria para obtener un proceso adaptado. Las preguntas analizadas en la encuesta respecto a la complejidad fueron:

Pregunta de Evaluación	**Descripción**
P2 – P3	Comprensión de los elementos para modelar una regla
P11	Sencillez de la metáfora de las reglas.
P12	Comprensión de los nodos condicionales.
P13	Comprensión de los nodos de elementos de proceso
P14	Operaciones de Adaptación suficientes
P15	Sencillez de los constructos de contexto
P16	Sencillez de los constructos de reglas de adaptación

	Complejidad de la Extensión a SPEM 2.0			
	Sujeto 1	Sujeto 2	Sujeto 3	Promedio
Nivel de Complejidad	0.22	0.22	0.16	0.20

Tabla 7. Indicadores de la Complejidad de Extender SPEM 2.0

Como se puede observar en la tabla 7 y cotejando la fórmula que se ha definido para evaluar la complejidad, en la cual entre más cercano a 0 sea el valor, el grado de complejidad es menor y podemos ver que la complejidad para los sujetos investigados resulta muy baja. Esto comprueba que la complejidad de construcción del modelo de proceso enriquecido con atributos de contexto y reglas de adaptación no afecta sobremanera la complejidad percibida por los ingenieros de proceso. Aunque la complejidad del metamodelo aumenta un poco, ésta no es significativamente percibida por los ingenieros de procesos y su beneficio resulta visible en comparación con la adaptación un proceso software con la especificación SPEM 2.0 sin estos constructos.

Efectividad:

En este apartado se muestra la información de los datos obtenidos en función de los objetivos propuestos con respecto al diseño de los elementos del modelo y la especificación lógica de las reglas de adaptación.

	Unidad de Análisis (%)			
	Sujeto 1	Sujeto 2	Sujeto 3	Promedio
Elementos del proceso	0.93	0.99	0.94	0.95
Elementos del Contexto	1	1	1	1
Lógica de Adaptación	1	0.6	0.6	0.73

Tabla 8. Indicador de Efectividad por cada Unidad de Análisis

De acuerdo a los resultados encontrados en la tabla anterior se puede observar que modelar los elementos del contexto es la tarea en donde los sujetos investigados fueron más eficientes obteniendo un 100% de efectividad, ya que no requiere mayor rigor. Un efecto parecido sucedió en el momento de modelar los elementos del proceso donde el porcentaje fue superior al 90% en segundo caso. Sin embargo la tarea donde más diferencia se ve, es el momento de especificar las reglas de adaptación, donde existe mayor porcentaje de error cometido por los sujetos investigados, esto debido a que la definición de las reglas es la parte más compleja del enfoque.

Eficiencia:

En este apartado se muestra la información de los datos de tiempo recogidos en el caso del diseño del proceso y la especificación de la lógica de reglas de adaptación en donde cada elemento de la guía, excluyendo los elementos mal diseñados, fue tenido en cuenta para hacer el análisis estadístico.

Ítems Modelados Correctamente	Unidad de Análisis (Elementos Modelados por Hora)			
	Sujeto 1	Sujeto 2	Sujeto 3	Promedio
Elementos de proceso	71.48	65.91	50.18	62.52
Elementos de Contexto	164	233.95	221.95	206.63
Lógica de Adaptación	3.53	2.25	2	2.59

Tabla 9. Indicador de Eficiencia por cada Unidad de Análisis

De acuerdo a los resultados obtenidos en la tabla anterior se puede observar cuantos elementos de proceso, elementos de contexto y reglas de adaptación son modelados correctamente en una hora por cada unidad de análisis. El ingeniero de procesos que corresponde a la primera unidad el más eficiente en el momento de modelar los elementos del proceso y especificar las reglas de adaptación. Sin embargo los ingenieros que corresponden a la segunda y tercera unidad de análisis son más eficientes en el momento de modelar los elementos del proceso. En promedio los ingenieros de proceso modelan 2.59 reglas por hora.

Usabilidad:

Teniendo en cuenta la encuesta entregada al finalizar la sesión de diseño del proceso por parte de los ingenieros de procesos, se analizaron los datos recogidos con el fin de obtener un valor cuantitativo de la usabilidad de tener integrado el proceso con información contextual y la lógica de reglas en un único modelo para obtener un proceso adaptado de una manera más efectiva. Las preguntas analizadas en la encuesta respecto a la usabilidad de un metamodelo integrando información de contexto y reglas de adaptación fueron:

Pregunta de Evaluación	Descripción
P1	Suficiencia en los constructos para modelar el contexto.
P4	Representación de las reglas de manera apropiada.
P5	Simplificación en la generación de las reglas
P6 - P7	El proceso resultante es el esperado
P8 - P9	Sencillez en la manera de expresar el contexto
P10	La información de contexto permite hacer un mejor análisis
P17 - P18	Retroalimentación de la lógica expresada
P19 - P20	Usabilidad del prototipo

	Unidad de Análisis			
	Sujeto 1	Sujeto 2	Sujeto 3	Promedio
Nivel de Usabilidad	0.94	0.88	0.88	0.90

Tabla 10. Indicador de la Usabilidad del Metamodelo para Adaptar Procesos Software

Teniendo en cuenta los resultados obtenidos por los ingenieros de procesos en el momento de modelar los elementos del proceso, los elementos de contexto y las reglas de adaptación en el prototipo implementado se puede observar que el nivel de satisfacción reflejado por ellos al momento de responder las preguntas es alto,

teniendo en cuenta que fueron hechas sobre la base de cómo un metamodelo basado en SPEM con constructos necesarios para el modelado de contexto y reglas de adaptación permite hacer una adaptación de un proceso software de una manera más amigable al usuario.

Resultados de la Encuesta a Nivel General

De acuerdo a los resultados encontrados en la tabla 11 y basándose en las métricas previamente agregadas, en el indicador de complejidad se puede observar que el 100% de los encuestados está completamente de acuerdo en que el metamodelo permite expresar la lógica de adaptación de una manera adecuada. Además el 100% de los sujetos investigados están de acuerdo en que los constructos agregados al metamodelo SPEM 2.0 sobrecargan poco el metamodelo original, reafirmando el resultado que la extensión aumenta la complejidad de una manera mínimamente significativa. Por otro lado teniendo en cuenta el indicador de usabilidad se puede observar que el 100% de los encuestados está completamente de acuerdo en que definir el contexto junto al proceso permite hacer un mejor análisis de los elementos a modelar y que las reglas de adaptación permiten obtener una retroalimentación rápida de la lógica de adaptación expresada en el prototipo en el momento de ejecutar la adaptación. El 67% de los encuestados está completamente de acuerdo que el proceso adaptado cumple con sus expectativas mientras el porcentaje restante está de acuerdo solamente. Por otro lado el 67% de los encuestados están de acuerdo en que la manera de expresar las reglas de adaptación es apropiada y que el prototipo es usable.

Ítem	Descripción	Suj.1	Suj.2	Suj.3
1	¿Las reglas descritas por el metamodelo caSPEM2.0 le permite expresar la lógica de adaptación que usted desea?	CA	CA	CA
2	¿El proceso adaptado por el prototipo cumple con sus expectativas?	CA	A	CA
3	¿La definición de contexto junto con el proceso resulta de mayor utilidad para hacer la adaptación del proceso que por separado?	CA	CA	CA
4	¿La manera de expresar las reglas de adaptación es adecuada (respecto a la complejidad) para un ingeniero de procesos?	A	A	CA
5	¿Considera que los mecanismos de adaptación de caSPEM2.0 No sobrecargan el metamodelo SPEM original?	A	A	A

6	¿El lenguaje de reglas facilita definir en forma rápida con alternativas antes de capturar las reglas de adaptación definitivas?	CA	CA	CA
7	¿Considera que el prototipo es usable?	A	CA	A

Tabla 11. Resultados Generales de la Encuesta hecha a los sujetos investigados

Donde el nivel de Aceptación de los ingenieros de proceso está dado por: CA = Completamente de Acuerdo, A= de Acuerdo, I = Indiferente, D = en Desacuerdo, CD = Completamente en Desacuerdo.

5.8.2 Resultados cualitativos

Apreciaciones de los ingenieros de procesos

Durante la experiencia, uno de los ingenieros de procesos manifestó, que el tener todo los elementos necesarios para modelar un proceso software en el mismo espacio de trabajo es más cómodo, y permite una edición más rápida del proceso software. Sin embargo, otro ingeniero de procesos mencionó que si se modelara un proceso más complejo, es decir, con mayor número de actividades, descripciones, productos de trabajo, mayor números de roles e interacciones, la vista del espacio de trabajo sería más pesada, si el proceso es mucho más grande. El archivo que almacena el modelo se volvería muy grande, y como consecuencia, la gestión y adaptación del proceso estará limitada a los recursos hardware donde caSPEMTool sea utilizado. Aunque los sujetos investigados tienen un conocimiento heterogéneo , los tres coincidieron en que expresar la lógica de adaptación es más fácil debido a que sólo es necesario tener un conocimiento básico sobre lógica matemática. Uno de los sujetos investigados que tiene una amplia experiencia con la adaptación de procesos basándose en un enfoque multi-modelo y utilizando el lenguaje ATL como motor para hacer la adaptación, expreso qué este enfoque disminuye de manera significativa la complejidad del proceso de adaptación y qué la retroalimentación de la lógica de adaptación plasmada dentro del prototipo de modelado en caSPEM2.0 es muy rápida. Para ver los resultados de la encuesta a los ingenieros de proceso diríjase al Anexo E.

Apreciaciones de los Investigadores durante el desarrollo del caso

Durante la evaluación del caSPEMTool, observamos que los ingenieros de procesos encontraron en la interface gráfica, algunas confusiones con el diseño de los elementos del proceso dentro del prototipo ya que este está basado completamente bajo la especificación de SPEM 2.0. Por ejemplo, cuando buscaban anidar elementos, como es el caso de las *tareas* anidadas dentro de las *actividades*, *patrones de proceso* con *actividades* anidadas, etc., o en el caso de los *productos de*

trabajo como parámetros de entrada y salida en la *tarea*. Aunque el prototipo indicaba el lugar correcto donde el elemento escogido debería ir (indicaba gráficamente donde debería ir), desde un punto de vista lógico no lo localizaban bien dentro del modelo, ya que los contenedores de los objetos no permiten una visualización general y el texto de ayuda es muy orientado a las asociaciones de los elementos SPEM 2.0. Esta confusión se puede justificar dado que muchas de las propiedades que presentaba algunos elementos del modelo, sobre todo elementos SPEM, pueden contener elementos del mismo tipo. La herramienta trabaja muy directamente con los detalles del modelo SPEM 2.0 y por tanto expone una complejidad innecesaria al modelador. Aún así, teniendo en cuenta la corta capacitación sobre los constructos nuevos agregados a la especificación de SPEM 2.0 y el funcionamiento del prototipo, la percepción de los ingenieros de proceso en el momento de modelar la guía fue buena ya se pudo apreciar que no hubo frustración al utilizarlo en el modelado, por lo cual concluimos que el prototipo es lo suficientemente usable, sin embargo como lo expresó uno de los ingenieros, es posible que para modelos más grandes el tema de la navegabilidad dentro del espacio de trabajo pueda complicarse.

caSPEMTool, en el momento de modelar el proceso general, lo hace de manera organizada como lo propone las especificación de SPEM 2.0, donde los elementos de Contenido y Proceso están debidamente separados y relacionados, al mirar el comportamiento de los sujetos investigados, el configurar un elemento del modelo, utilizando la sección Propiedades de la herramienta, fue una tarea un poco confusa al principio, dado que en algunos, el objeto a modelar tienen muchos atributos, esta fue la razón que generó que algunos objetos no se configuraran adecuadamente en su totalidad.

5.9 Análisis de Resultados

5.9.1 Análisis de Resultados Cuantitativos

El implementar el diseño del proceso en promedio tiene un esfuerzo de 2.29 personas-hora, en promedio la implementación de los elementos del contexto tiene un esfuerzo de 0.25 horas-persona, la implementación de las reglas de adaptación en promedio tiene un esfuerzo de 1.42 horas-persona.

En cuanto a la lógica de adaptación (las reglas), la guía proveía enunciados a partir de los cuales se debían inferir reglas para la lógica de adaptación, y es aquí nuestro principal punto de observación, dado que en trabajos previos como [69], es en esta fase la que más esfuerzo demanda y por tanto nuestro principal aporte, que como resultado, se requiere de un 35.81% del esfuerzo total al implementar las reglas de adaptación de la guía dada.

Los resultados obtenidos de la métrica para calcular la complejidad muestra que el metamodelo propuesto no la incrementa sustancialmente, los constructos utilizados para esta propuesta son muy cercanos al entorno de la ingeniería y la matemática, se tuvo en cuenta los trabajos previos, sobre todo en el diseño de los mecanismos para contexto, y el comportamiento de la lógica matemática para darle forma a las reglas. Como resultado el nivel de complejidad es cercano a cero, alejado del límite máximo (lo que corresponde a una evaluación de complejidad cercana al nivel mínimo, siendo le resultado percibido por los sujetos de investigación "*Completamente de Acuerdo*", la diferencia de resultados de nuestra fórmula, en donde mas "*completamente de acuerdo*" estén los sujetos de investigación en las repuestas mas se decrementa el nivel de complejidad percibida por ellos).

La efectividad es una medida la cual nos permite observar, el grado de correctitud con que los sujetos investigados realizaron el proyecto de modelado, nos permite observar el grado en que se cumple el objetivo del modelado correcto del proceso y medir si el modelo con sus nuevos constructos afectan la efectividad de esta tarea. Si observamos la efectividad de los elementos de proceso y contexto, tuvieron un promedio mayor al 90%, lo cual nos indica que los nuevos elementos del metamodelo no impactaron su efectividad. Sin embargo los elementos de regla sugiere otro análisis, el cual consiste, una misma regla puede ser modelada en multitud de maneras, para evaluar su efectividad se toman el numero de reglas correctas, esta medida depende no de los elementos correctamente modelados, si no de la lógica correctamente modelada, aun así, las unidades que menor efectividad tuvieron (60%) es muy bueno considerando que mapear la regla con estos elementos es una técnica nueva dentro de la adaptación de procesos software.

La eficiencia es una medida que nos permite observar la velocidad con que un elemento correcto es modelado, los sujetos investigados tuvieron en esta fase un buen comportamiento, a nivel de proceso, en promedio 62.5 elementos de proceso bien modelados por hora, que representan alrededor del 50% de modelado total del proceso, es un buen indicador que nos permite decir que modelar el proceso es

bastante rápido y si es un proceso más grande en número de elementos, la tarea disminuiría su duración con respecto al tiempo que tardaría normalmente. Con respecto al contexto, los sujetos investigados no tuvieron mayor problema, dado que en promedio 206.7 elementos correctamente modelados por hora, hace prever que modelar el contexto para enriquecer el modelo de proceso no afecta en nada el modelado eficiente del proceso. En cuanto a las reglas, las unidades tuvieron un buen comportamiento, en promedio se modelaron 2.6 reglas correctamente modeladas por hora, que nos indica que un ingeniero de proceso puede probar cada hora el resultado de al menos dos reglas y su efecto en el proceso, este resultado trae beneficios a la hora de comprobar incrementalmente si la regla satisface las pretensiones del diseñador, pero puede mejorar a medida que se gana experiencia en el uso de esta propuesta, dado que la lógica con que se mapea un regla es sencilla.

La usabilidad es una medida que indica el grado de utilidad y facilidad de aprendizaje de la propuesta, los resultados de los sujetos investigados demuestran que en promedio 0.90 (lo que corresponde a una evaluación de usabilidad cercano al nivel máximo "*Completamente de Acuerdo*" percibida por los sujetos de investigación), es un indicador sobresaliente, dado que para todos los sujetos investigados era su primera experiencia mapeando el contexto y reglas de adaptación de un proyecto en el mismo modelo de proceso. Esta usabilidad no incluye todos los conceptos propios que promulga los estudios de usabilidad, se ve sólo desde la utilidad de tener todos los elementos necesarios para modelar el proceso, el contexto que afecta el proceso y la lógica de adaptación en un único modelo, además de poder probar rápidamente las reglas que adaptaran futuros procesos software y la facilidad de aprendizaje percibida por los sujetos de investigación utilizados en el estudio de caso.

5.10 Amenazas de Validez

Debido a la pequeña disponibilidad de los sujetos investigados es muy difícil generalizar sobre los datos cuantitativos debido a que la experiencia partió desde el modelado del proceso, mas no desde el levantamiento del proceso, es posible que los tiempos en un caso con el ciclo completo se vean incrementados considerablemente, sin embargo aunque el tiempo de modelado fue más corto,

proporcional a este, las reglas de adaptación fueron implementadas en un tiempo notoriamente más corto que los reportes anteriores en forma más eficiente, lo que confirma las ventajas de caSPEM2.0 respecto a una solución multi-modelo.

5.11 Síntesis y Discusión

Teniendo en cuenta la propuesta CASPER hecha por Hurtado et al. [12] consideramos que la segregación de los modelos para el diseño de un proceso software incluyendo la información de contexto puede aumentar la complejidad técnica debido a que en este caso es necesario manejar dos metamodelos diferentes (el metamodelo del proceso y el metamodelo de contexto) manualmente, ya que es necesario obtener los elementos tanto del proceso como del contexto, que son necesarios para hacer la adaptación de tal manera que el proceso de adaptación en este caso puede ser propenso a error debido a que no existe una retroalimentación apropiada sobre los elementos mal enlazados y por tal motivo el proceso de adaptación no será llevado a cabo. Además para el proceso de adaptación es necesario que el ingeniero de procesos tenga un conocimiento previo sobre algún lenguaje de transformación de modelos, como es el caso de ATL agregándole complejidad al proceso de especificación de reglas.

El porcentaje resultante (35.81%) del esfuerzo total de llevar a cabo la adaptación de procesos software y comparándolo con el porcentaje resultante de Ruiz et al. [69], el cual es de un 56.35%, la reducción del esfuerzo en esta fase es muy significativa, dado que en el trabajo previo la mayor cantidad de esfuerzo utilizada, fue al implementar las reglas, ya que Ruiz reporta 18 horas para implementar tres reglas en un lenguaje de transformación. La mayor cantidad de esfuerzo fue en ésta implementación, lo que significa que al menos 9 horas fueron invertidas en la implementación dando como resultado una eficiencia de 0.33 reglas por hora, mientras que con esta propuesta que no involucra aprender un lenguaje de transformación para realizar adaptaciones, solo se necesita manejar lógica matemática básica para construir las reglas de adaptación, que luego son interpretadas para realizar las adaptaciones, proceso que para el usuario es transparente, y como resultado la eficiencia reportada con esta propuesta es de 2.6 reglas por hora. Adicionalmente el porcentaje de esfuerzo reportado por Ruiz es relativo a un esfuerzo mucho mayor en el modelado del proceso, debido a que en su

caso requirió de mayor trabajo en el levantamiento de la información del proceso y de sus contextos, lo cual fue reducido en este caso, de tal forma que el 35.81% en nuestro caso es un porcentaje significativamente menor que un 35.81% en el caso de Ruiz y por tanto mucho más significativamente menor que el 56.35% por él reportado.

Por otro lado es necesario resaltar que el prototipo implementado ha sido desarrollado sobre un gran porcentaje de la especificación de SPEM 2.0 a diferencia de la propuesta hecha en CASPER en Hurtado et al. [12] agregando mayor esfuerzo en el momento de modelar el proceso ya que es necesario tener un conocimiento un poco más detallado acerca del comportamiento de los elementos de la especificación SPEM 2.0.

Basados en los resultados de este caso de estudio se puede concluir que existe un nivel de complejidad relativamente bajo, y que la eficiencia se puede considerar alta de acuerdo a las comparaciones hechas sobre los trabajos de Hurtado et a. [12, 68] y Ruiz et al. [69] donde se ha reportado mucho más esfuerzo en la especificación de las reglas que en el diseño del proceso. Adicionalmente la información cualitativa confirma que el mantener un metamodelo único ayuda a analizar los elementos de proceso que deben ser adaptados, además permite edición y retroalimentación rápida de lógica de adaptación utilizada para genera un proceso de software bajo características de contexto especificas. Por otro lado el hecho de que uno de los sujetos investigados haya trabajado directamente con el enfoque CASPER sin el soporte de caSPEM2.0, su testimonio ha sido de gran utilidad para cotejar si los resultados empíricos obtenidos en este estudio de caso concuerdan con sus experiencias.

6 Conclusiones, Limitaciones y Trabajo Futuro

La adaptación de procesos software es una tarea necesaria para tener proyectos productivos sin poner en riesgo la calidad del producto, además de establecer un marco de trabajo para agilizar y organizar el trabajo de los involucrados (principalmente los que construyen el software) del proyecto, fomentando la optimización del proceso. Sin embargo esta tarea ha recibido una atención informal, ya que no se han provisto los medios necesarios para formalizar la adaptación, ni bases de conocimiento obtenidas de la experiencia del ingeniero de procesos en el momento de especificar su lógica y por tanto resulta impracticable, por lo que muchas veces se omite o se realiza de forma descuidada.

Favorecer esta tarea de adaptación, haciéndola práctica es un interés relevante para la ingeniería de software porque apunta a resolver aspectos determinantes para tener una industria de software competitiva, los procesos adaptados pueden ser fuente hacia un marco de mejora de procesos, cada proyecto que desarrollen las organización no arrancaran desde cero a la hora de organizar el trabajo, dado que el proceso general enmarca una estructura básica de actividades para la producción y\o mantenimiento de productos software, sirve de guía e instrumento de aprendizaje para miembros antiguos y sobre todo nuevos dentro de los proyectos para que se acoplen rápidamente a la forma de trabajo en la organización.

El prototipo caSPEMTool permite a los diseñadores de procesos especificar un modelo de proceso e incluir información de contexto por medio de un editor gráfico (más amigable, adicional al editor por defecto EMF), de tal manera que el ingeniero de procesos puede hacer un análisis de la información contextual modelada y

relacionarla con los elementos del proceso previamente especificados, que tendrán modificaciones en procesos adaptados futuros.

6.1 Conclusiones

- SPEM 2.0 es un metamodelo adecuado para la instanciación de modelos de proceso e ingeniería sin embargo carece de mecanismos para expresar los procesos basándose en las características de un proyecto o de la organización donde estos serán aplicados, por ello se ha buscado con éste proyecto enriquecer el metamodelo proveyéndolo de constructos que permitan expresar el contexto en particular. A través del estudio de caso de éste proyecto se ha mostrado una reducción de la complejidad del modelado y cómo esto ha simplificado la práctica al modelar el proceso, su contexto y las reglas de adaptación dentro de un mismo modelo de proceso de software.
- caSPEM2.0 permite instanciar un paquete de conocimiento que agrupa un conjunto de reglas, las cuales se modelan como condicionales simples a modo de un grafo dirigido y cuyas condicionales evalúan el contexto del proyecto, con el fin de que el ingeniero de procesos diseñe reglas teniendo en cuenta las características de contexto y de esta manera instrumentalizar la adaptación procesos software.
- Para la evaluación del metamodelo, en ésta tesis se ha desarrollado una herramienta prototipo la cual ha seguido un desarrollo basado en la ingeniería dirigida por modelos (MDE), el cual es un enfoque que ha resultado de gran valor tanto para el desarrollo del producto, así como para soportar la filosofía de la herramienta, la cual busca definir y adaptar procesos de desarrollo software de una manera sistemática.
- El enfoque MDE permite construir rápidamente un producto software a partir de la especificación de metamodelos y por medio de generadores de código, generar el código que soporta la gestión de instancias de modelo. Es importante tener en cuenta que en este enfoque si el metamodelo no es bien diseñado, acorde con los requisitos que demanda un producto, el resultado no será correcto y por supuesto, el tiempo invertido y sus demás recursos demandados habrían sido

utilizados en vano. Pero esta es a su vez una ventaja, dado que el factor de intervención humano durante la generación de código es bajo, puesto que si hay un error en la especificación del metamodelo que no fue advertido, se puede corregir y volver a generar, quitando cierto grado de dependencia del programador, como no sucede en otro tipo de aplicaciones, en las cuales si los modelos no fueron diseñados apropiadamente, el impacto sobre el producto final sería catastrófico, y el grado de dependencia del programador aumentaría para solucionar tales situaciones.

- El prototipo caSPEMTool permite a los diseñadores de procesos especificar un modelo de proceso e incluir información de contexto por medio de un editor gráfico (más amigable, adicional al editor por defecto EMF).
- El prototipo caSPEMTool permite al ingeniero de procesos hacer un análisis de la información contextual modelada y relacionarla con los elementos del proceso previamente especificados, que tendrán modificaciones en procesos adaptados futuros.
- La principal ventaja que tiene caSPEM2.0 es que se ha permitido enriquecerlo con la capacidad de expresar la lógica de adaptación utilizando las características de un proyecto u organización, utilizando lógica matemática, disminuyendo substancialmente el tiempo de adaptación. Esto debido a que se oculta la complejidad de un lenguaje de transformación de modelos y se evita que el ingeniero de procesos cometa errores en el momento de enlazar los elementos del contexto con los elementos de proceso a ser adaptados.

6.2 Limitaciones

En el caso de estudio y discusiones se han evidenciado algunas limitaciones sobre la tesis propuesta:

- La herramienta prototipo caSPEMTool está basado completamente sobre la especificación de SPEM 2.0 de tal manera que al ingeniero de proceso se le sobrecarga el esfuerzo en el momento de modelar un proceso. Herramientas de uso industrial como EPF y RPC (rational)

ocultan gran cantidad de mecanismos de SPEM a fin de facilitar su uso. La construcción del prototipo fue limitada por la capacidad dirigida por modelos de las herramientas participantes de la generación de editores gráficos dado que las metáforas gráficas provistas por ésta para representar objetos no son las suficientes.

- caSPEM2.0 es un prototipo diseñado para probar las ventajas de no tener modelos segregados en el proceso de adaptación de Procesos Software, resaltando sus ventajas. Sin embargo, el prototipo no ha sido aplicado en el modelado de procesos más grandes en cuanto a cantidad de elementos modelados, lo cual podría afectar los resultados aquí reportados. Esto fue suficiente para el contexto de evaluación del metamodelo, pero para llevarlo a la práctica, se requiere resolver asuntos de usabilidad debido a que caSPEM2.0 refleja todos los constructos de la especificación SPEM 2.0, generando confusión a los ingenieros de proceso en la tarea de modelado de proceso y asuntos de complejidad al momento de soportar procesos más grandes, debido a que se hace necesario especificar de una manera más concreta los mecanismos de variabilidad soportados por SPEM 2.0.
- No todas las posibles adaptaciones sobre los modelos de proceso que un diseñador de procesos pueda tener en mente han sido soportadas por el metamodelo, debido a que él objetivo de esta tesis es mostrar la capacidad del metamodelo y para ello sólo fue necesario implementar el subconjunto más usado de casos de adaptación. Las adaptaciones que pueden realizarse solo aplican para elementos del proceso, en el caso de realizar la adaptación de los elementos de contenido el diseñador no tendría los constructos en el lenguaje de las reglas necesarios para tal tarea, aunque están disponibles en el metamodelo para cualquier adaptación manual. Sin embargo, la herramienta no imposibilita extensiones en las que esto pueda ser agregado, con poca complejidad técnica, debido que es necesario incluir nuevos constructos y/o características a los constructos del metamodelo ya implementado, generar la herramienta nuevamente e incluir la programación suficiente dentro del script EOL para que esos nuevos constructos sean tenidos en cuenta en la adaptación de procesos, ya que es necesario hacer un análisis de los nuevos constructos que deben ser agregados, las

relaciones entre ellos y cómo las operación soportarían las modificaciones sobre estos nuevos constructos y de esta manera incluir esa información en el metamodelo y mediante MDD hacer iteraciones incrementales agregando poco a poco los nuevos elementos.

6.3 Lecciones Aprendidas y Problemas Enfrentados

- La gestión del riesgo técnico es un factor fundamental en el momento de realizar un trabajo de grado, en nuestro caso el investigar y trabajar sobre las tecnologías y herramientas de las que podíamos soportarnos en paralelo con la formulación del anteproyecto nos permitió sentar unas bases sólidas para analizar el alance del proyecto y adelantar trabajo para terminarlo dentro de los plazos establecidos.
- El desarrollo de aplicaciones bajo el enfoque MDE facilita la producción rápida de prototipos funcionales debido a que el esfuerzo es trasladado desde la producción de código al diseño de modelos. Sin embargo, adaptarse a esta nueva metodología requiere un diseño más detallado de los modelos para producir el comportamiento deseado dentro de una solución informática, lo cual implica un nivel de abstracción alto.
- En la práctica, un caso de estudio requiere un diseño muy detallado y un análisis de los posibles inconvenientes que pueden surgir en el momento de realizarlo, de tal manera que si llegan a pasar se los pueda enfrentar de una manera proactiva.
- Graphical Modeling Framework permite la representación de modelos muy básica, por tal motivo no es posible construir una herramienta rica en interfaces gráficas apoyado únicamente de esta tecnología.
- No se han encontrado estudios que permitan hacer una evaluación más concreta del impacto que esta extensión ha tenido al facilitar un lenguaje de metamodelado para especificar la lógica de adaptación de una manera más abstracta.

6.4 Trabajo Futuro

En esta tesis se han analizado algunos puntos que pueden ser tenidos en cuenta para trabajo futuro a corto y mediano plazo, entre ellos se incluye:

- Mejorar la herramienta caSPEMTool de tal forma que se le permita al ingeniero abastecer el modelado de un proceso de desarrollo software, sin que tenga que conocer y aplicar todas las relaciones y asociaciones en detalle de SPEM.
- Se espera hacer un experimento controlado sobre la manera de especificar la lógica de adaptación en caSPEM2.0 para evidenciar qué constructos son necesarios agregar al metamodelo, con el fin de definir reglas más completas semántica y sintácticamente, así como determinar qué elementos SPEM 2.0 ocultar.
- Incorporar la estrategia aplicada por caSPEM2.0 en herramientas de uso en la industria como Rational Composer y Eclipse Process Framework. Por ejemplo producir una versión caUMA y basada en ésta extender la funcionalidad de EPF para soportar la propuesta presentada en esta tesis. Sin embargo, la complejidad de realizar tal estrategia es alta, requerirá tiempo, dado que en primer lugar si se escoge Eclipse Process Framework se deberá entender la arquitectura y el código que soporta tal herramienta, claro esta no en su totalidad, pero basado en nuestra experiencia, es una tarea que puede llevar mucho tiempo, puesto que las estructuras que son deseables extender (siendo especifico “MethodPlugin”) poseen un conjunto grande de dependencias y además su mecanismos de almacenamiento de tal estructura es complejo, hecho por el cual esta propuesta no siguió ese camino.
- caSPEM2.0 definió los constructos necesarios para adaptar elementos de proceso dentro de SPEM 2.0 dejando de lado los elementos de contenido de método, por lo cual sería deseable estudiar la posibilidad de ampliar los mecanismos de adaptación hacia estos elementos de la especificación.

7 Bibliografía

1. Conradi, H. and A. Fuggetta, *Improving software process improvement.* Software, IEEE, 2002. **19**(4): p. 92-99.
2. Firesmith, D., *Creating a Project-Specific Requirements Engineering Process.* Journal of Object Technology, 2004. **3**(5): p. 31-44.
3. Xu, P., *Knowledge Support in Software Process Tailoring*, in *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 3 - Volume 03*. 2005, IEEE Computer Society. p. 87.3.
4. Park, S., et al., *A semi-automated filtering technique for software process tailoring using neural network.* Expert Systems with Applications, 2006. **30**(2): p. 179-189.
5. Pedreira, O., et al., *A systematic review of software process tailoring.* SIGSOFT Softw. Eng. Notes, 2007. **32**(3): p. 1-6.
6. OMG, O.M.G.I., ***Unified Modeling Language - UML***. 2011.
7. OMG, O.M.G.I., *Software & System Process Engineering Metamodel Specification (SPEM)*. 2008.
8. Franch, X. and J.M. Ribo. *A structured approach to software process modelling*. in *Euromicro Conference, 1998. Proceedings. 24th*. 1998.
9. Bendraou, R., M.-P. Gervais, and X. Blanc, *UML4SPM: A UML2.0-Based Metamodel for Software Process Modelling Model Driven Engineering Languages and Systems*, L. Briand and C. Williams, Editors. 2005, Springer Berlin / Heidelberg. p. 17-38.
10. Nitto, E.D., et al., *Deriving executable process descriptions from UML*, in *Proceedings of the 24th International Conference on Software Engineering*. 2002, ACM: Orlando, Florida. p. 155-165.
11. Chou, S.-C., *A Process Modeling Language Consisting of High Level UML-Based Diagrams and Low Level Process Language.* Journal of Object Technology, 2002. **1**: p. 137-163.
12. Hurtado, J.A., et al., *An MDE approach to software process tailoring*, in *Proceedings of the 2011 International Conference on on Software and Systems Process*. 2011, ACM: Waikiki, Honolulu, HI, USA. p. 43-52.
13. Aharoni, A. and I. Reinhartz-Berger, *A Domain Engineering Approach for Situational Method Engineering*, in *Proceedings of the 27th International*

Conference on Conceptual Modeling. 2008, Springer-Verlag: Barcelona, Spain. p. 455-468.
14. Kang, D., et al., *A Case Retrieval Method for Knowledge-Based Software Process Tailoring Using Structural Similarity*, in *Proceedings of the 2008 15th Asia-Pacific Software Engineering Conference*. 2008, IEEE Computer Society. p. 51-58.
15. Humphrey, W.S. and M.I. Kellner, *Software process modeling: principles of entity process models*, in *Proceedings of the 11th international conference on Software engineering*. 1989, ACM: Pittsburgh, Pennsylvania, United States. p. 331-342.
16. Acuña, S.T. and X. Ferré, *Software Process Modelling*. Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics: Information Systems Development-Volume I - Volume I. 2001: IIIS. 237-242.
17. O'Regan, G., *Motivation for Software Process Improvement*, in *Introduction to Software Process Improvement*. 2011, Springer London. p. 1-12.
18. Pinto, P., *Software Improvement Process at CERN CO-AP*, in *Facultade de Engenharia*. 2008, Universidade Do Porto.
19. *ISO/IEC/IEEE Standard for Systems and Software Engineering - Software Life Cycle Processes.* IEEE STD 12207-2008, 2008: p. c1-138.
20. *ISO/IEC/IEEE Systems and Software Engineering - System Life Cycle Processes.* IEEE STD 15288-2008, 2008: p. c1-84.
21. *Systems and software engineering -- Content of life-cycle information products (documentation).* ISO/IEC/IEEE 15289:2011(E), 2011: p. 1-94.
22. Munassar, N. and A. Govardhan, *A Comparison Between Five Models Of Software Engineering.* International Journal of Computer Science Issues, 2010. **7**(5): p. 94-101.
23. OMG, O.M.G.I., *MOF 2 XMI Mapping, Version 2.4*. 2010. p. 124.
24. OMG, O.M.G.I., *OMG Unified Modeling LanguageTM (OMG UML), Infrastructure*. 2009. p. 226.
25. OMG, O.M.G.I., *Business Process Model and Notation*. 2011. p. 538.
26. France, R. and B. Rumpe, *Model-driven Development of Complex Software: A Research Roadmap*, in *2007 Future of Software Engineering*. 2007, IEEE Computer Society. p. 37-54.
27. Mellor, S.J., et al., eds. *MDA Distilled: Principles of Model-Driven Architecture*. 2004, Addison-Wesley Professional. 176.
28. CMMI, T.P., *CMMI for Development, Version 1.3*. 2010, Software Engineering Institute - Carnegie Mellon University. p. 483.
29. Cass, A.G., et al., *Little-JIL/Juliette: a process definition language and interpreter*, in *Proceedings of the 22nd international conference on Software engineering*. 2000, ACM: Limerick, Ireland. p. 754-757.
30. Armbrust, O., et al., *Scoping software process lines.* Softw. Process, 2009. **14**(3): p. 181-197.
31. Simidchieva, B., L. Clarke, and L. Osterweil, *Representing Process Variation with a Process Family*, in *Software Process Dynamics and Agility*, Q. Wang, D. Pfahl, and D. Raffo, Editors. 2007, Springer Berlin / Heidelberg. p. 109-120.

32. Osterweil, L. and A. Wise, *Using Process Definitions to Support Reasoning about Satisfaction of Process Requirements*, in *New Modeling Concepts for Today's Software Processes*, J. Münch, Y. Yang, and W. Schäfer, Editors. 2010, Springer Berlin Heidelberg. p. 2-13.
33. Foundation, T.E. *Eclipse process framework (epf) composer 1.0 architecture overview*. 2012; Available from: http://eclipse.org/epf/composer_architecture/.
34. Garcia, F., A. Vizcaino, and C. Ebert, *Process Management Tools.* IEEE Softw., 2011. **28**(2): p. 15-18.
35. Fontoura, L.M. and R.T. Price, *A Framework for Tailoring Software Process*, in *SEKE*. 2007, Knowledge Systems Institute Graduate School. p. 63-66.
36. Yoon, I.-C., S.-Y. Min, and D.-H. Bae, *Tailoring and Verifying Software Process*, in *Proceedings of the Eighth Asia-Pacific on Software Engineering Conference*. 2001, IEEE Computer Society. p. 202.
37. Akbar, R., M. Hassan, and A. Abdullah, *A Review of Prominent Work on Agile Processes Software Process Improvement and Process Tailoring Practices*, in *Software Engineering and Computer Systems*, J. Zain, W. Wan Mohd, and E. El-Qawasmeh, Editors. 2011, Springer Berlin Heidelberg. p. 571-585.
38. Xu, P. and B. Ramesh, *A Tool for the Capture and Use of Process Knowledge in Process Tailoring*, in *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 3 - Volume 3*. 2003, IEEE Computer Society. p. 96.3.
39. Martinho, R., J. Varajao, and D. Domingos, *A Two-Step Approach for Modelling Flexibility in Software Processes*, in *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. 2008, IEEE Computer Society. p. 427-430.
40. Madhavji, N.H., *The process cycle.* Softw. Eng. J., 1991. **6**(5): p. 234-242.
41. Ocampo, A., J. Münch, and F. Bella, *Software Process Commonality Analysis.* Software Process: Improvement and Practice, 2005.
42. Cockburn, A., *Agile software development*. 2002: Addison-Wesley Longman Publishing Co., Inc. 278.
43. Ralyté, J. and B. Henderson-Sellers, *Situational Method Engineering: State-of-the-Art Review.* Journal of Universal Computer Science, 2010. **16**: p. 424-478.
44. Ralyté, J., R. Deneckère, and C. Rolland, *Towards a Generic Model for Situational Method Engineering Advanced Information Systems Engineering*, J. Eder and M. Missikoff, Editors. 2003, Springer Berlin / Heidelberg. p. 1029-1029.
45. Bowers, J., et al., *Tailoring XP for Large System Mission Critical Software Development*, in *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe 2002*. 2002, Springer-Verlag. p. 100-111.
46. Cao, L., et al., *How Extreme Does Extreme Programming Have to Be? Adapting XP Practices to Large-Scale Projects*, in *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 3 - Volume 3*. 2004, IEEE Computer Society. p. 30083.3.

47. Hanssen, G., H. Westerheim, and F. Bjørnson, *Tailoring RUP to a Defined Project Type: A Case Study*, in *Product Focused Software Process Improvement*, F. Bomarius and S. Komi-Sirviö, Editors. 2005, Springer Berlin / Heidelberg. p. 209-228.
48. Borges, P., P. Monteiro, and R.J. Machado, *Tailoring RUP to Small Software Development Teams*, in *Proceedings of the 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*. 2011, IEEE Computer Society. p. 306-309.
49. Rombach, D., *Integrated Software Process and Product Lines*, in *Unifying the Software Process Spectrum*, M. Li, B. Boehm, and L. Osterweil, Editors. 2006, Springer Berlin / Heidelberg. p. 83-90.
50. Schnieders, A. and M. Weske, *Activity Diagram Based Process Family Architectures for Enterprise Application Families*, in *Enterprise Interoperability*, G. Doumeingts, et al., Editors. 2007, Springer London. p. 67-76.
51. Barreto, A., et al., *Supporting the Definition of Software Processes at Consulting Organizations via Software Process Lines*, in *Proceedings of the 2010 Seventh International Conference on the Quality of Information and Communications Technology*. 2010, IEEE Computer Society. p. 15-24.
52. Bendraou, R., et al., *Definition of an Executable SPEM 2.0*, in *Proceedings of the 14th Asia-Pacific Software Engineering Conference*. 2007, IEEE Computer Society. p. 390-397.
53. Koudri, A. and J. Champeau, *MODAL: A SPEM Extension to Improve Co-design Process Models*, in *New Modeling Concepts for Today's Software Processes*, J. Münch, Y. Yang, and W. Schäfer, Editors. 2010, Springer Berlin / Heidelberg. p. 248-259.
54. Martínez-Ruiz, T., F. García, and M. Piattini, *Towards a SPEM v2.0 Extension to Define Process Lines Variability Mechanisms*, in *Software Engineering Research, Management and Applications*, R. Lee, Editor. 2008, Springer Berlin / Heidelberg. p. 115-130.
55. Martínez-Ruiz, T., et al., *Modelling software process variability: an empirical study.* Software, IET, 2011. **5**(2): p. 172-187.
56. Martinho, R., et al., *FlexSPMF: A Framework for Modelling and Learning Flexibility in Software Processes*, in *Proceedings of the 2nd World Summit on the Knowledge Society: Visioning and Engineering the Knowledge Society. A Web Science Perspective*. 2009, Springer-Verlag: Chania, Crete, Greece. p. 78-87.
57. Ellner, R., et al., *eSPEM – A SPEM Extension for Enactable Behavior Modeling*, in *Modelling Foundations and Applications*, T. Kühne, et al., Editors. 2010, Springer Berlin / Heidelberg. p. 116-131.
58. Kedji, K.A., et al. *Towards a Tool-Supported Approach for Collaborative Process Modeling and Enactment*. in *Software Engineering Conference (APSEC), 2011 18th Asia Pacific*. 2011.
59. Diaw, S., et al., *Specification and Implementation of Spem4mde, A Metamodel for Mde Software Processes*, in *SEKE 2011 - Proceedings of the 23rd International Conference on Software Engineering and Knowledge*. 2011. p. 646-653.

60. OMG, O.M.G.I., *OMG Unified Modeling LanguageTM (OMG UML), Infrastructure Version 2.2*. 2009.
61. Pillat, R.M., T.C. Oliveira, and F.L. Fonseca. *Introducing Software Process Tailoring to BPMN: BPMNt*. in *Software and System Process (ICSSP), 2012 International Conference on*. 2012.
62. Jouault, F. and J. Bézivin, *KM3: a DSL for metamodel specification*, in *Proceedings of the 8th IFIP WG 6.1 international conference on Formal Methods for Open Object-Based Distributed Systems*. 2006, Springer-Verlag: Bologna, Italy. p. 171-185.
63. Steinberg, D., et al., eds. *EMF: Eclipse Modeling Framework (2nd Edition)* 2nd Edition ed., ed. E. Gamma, L. Nackman, and J. Wiegand. 2008.
64. Kolovos, D., et al., *Taming EMF and GMF Using Model Transformation Model Driven Engineering Languages and Systems*, D. Petriu, N. Rouquette, and Ø. Haugen, Editors. 2010, Springer Berlin / Heidelberg. p. 211-225.
65. Runeson, P. and M. Höst, *Guidelines for conducting and reporting case study research in software engineering.* Empirical Softw. Engg., 2009. **14**(2): p. 131-164.
66. Yin, R., *Case study research: Design and methods*. 3dr Edition ed. Vol. 5. 2003: Sage Publications.
67. Klein, H.K. and M.D. Myers, *A set of principles for conducting and evaluating interpretive field studies in information systems.* MIS Q., 1999. **23**(1): p. 67-93.
68. Hurtado, J.A., et al., *MDE Software Process Lines in Small Companies.* 2012.
69. Ruiz, P. and J.A. Hurtado, *Una Línea de Procesos de Software basada en el Proceso Unificado.* 2012.
70. Benbasat, I., D.K. Goldstein, and M. Mead, *The case research strategy in studies of information systems.* MIS Q., 1987. **11**(3): p. 369-386.
71. Cardozo, E., Y. Velasquez de Naime, and M.C. Rodríguez, *La definición de PYME en América: Una revisión del estado del arte*, in *6th International Conference on Industrial Engineering and Industrial Management*. 2012: Vigo.