UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
ESCUELA DE POSTGRADO Y EDUCACIÓN CONTINUA

ANÁLISIS DE PROBLEMAS DE ESPECIFICACIÓN DE PROCESOS DE DESARROLLO DE SOFTWARE

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN TECNOLOGÍAS DE LA INFORMACIÓN

CHRISTIAN ANDRÉS AVENDAÑO JELDRES

PROFESORES GUÍA:
DANIEL PEROVICH GEROSA
JOCELYN SIMMONDS WAGEMANN

MIEMBROS DE LA COMISIÓN:
MARÍA CECILIA BASTARRICA PIÑEYRO
FRANCISCO GUTIERREZ FIGUEROA
LUIS SILVESTRE QUIROGA

Este trabajo ha sido financiado por CONICYT-PFCHA/Magíster Nacional/2014-22140053

SANTIAGO DE CHILE
2024

Resumen

El modelado de procesos de software por lo general, son especificaciones sofisticadas las cuales demandan un enorme esfuerzo pero una vez especificados existen pocos enfoques y aún menos herramientas que ayuden al ingeniero de procesos a analizar la calidad del proceso.[11] El proceso, al tener errores, impacta al equipo de desarrollo del software extendiendo el tiempo en generar la solución final hacia el cliente y en algunos casos sobredimensionamiento de los mismos.

Para asistir al ingeniero de procesos en el análisis de problemas y de especificación, en esta tesis propone la herramienta SPS. En la construcción de SPS se realizó una especificación de los patrones de error implementados en AVISPA, para lo cual se utiliza OCL como lenguaje de alto nivel. SPS es una herramienta implementada como plugin de EPFC [6] que presenta diferentes patrones de error de un modelo de proceso de software resaltando los errores potenciales a través de indicadores. El enfoque de la herramienta ha sido validado por un panel de expertos aplicándolo en el análisis de algunos modelos de procesos.

El resultado del trabajo es una herramienta que asiste al ingeniero de procesos al advertir sobre posibles errores, teniendo en cuenta la posibilidad de utilizar umbrales que se ajusten a la realidad de cada empresa. Esto significa que la herramienta proporciona alertas o indicadores cuando se detectan situaciones que podrían llevar a errores en los procesos, y también permite configurar umbrales específicos que reflejen las necesidades y características particulares de cada empresa. De esta manera, la herramienta puede adaptarse mejor a las circunstancias y facilitar la toma de decisiones para mejorar los procesos. Sin embargo resulta importante considerar el costo de mantenibilidad de la herramienta ya que lamentablemente el software EPFC no ha tenido actualizaciones lo cual dificulta la posibilidad de generar nuevas versiones.

Agradecimientos

Quiero expresar mi más profundo agradecimiento a mis profesores guía, Jocelyn Simmonds y Daniel Perovich. Su paciencia, orientación y apoyo constante han sido fundamentales para el desarrollo y culminación de este proyecto de tesis.

De manera especial, quiero agradecer al profesor Sergio Ochoa, cuya inestimable ayuda, conocimientos y motivación han sido pilares en mi proceso de aprendizaje y realización de esta investigación. Sus consejos y su disposición para resolver dudas en cualquier momento fueron esenciales para superar los retos encontrados en el camino.

Asimismo, deseo expresar mi sincero agradecimiento al profesor Julio Hurtado, cuya ayuda fue clave en la etapa de validación de este proyecto. Su experiencia y apoyo fueron determinantes para el éxito de esta fase crucial.

También quiero agradecer a mi familia, cuyo apoyo incondicional y constante ánimo me han permitido seguir adelante en momentos difíciles. Su amor y comprensión han sido una fuente invaluable de fortaleza y motivación.

Gracias a todos por su dedicación y compromiso. Este logro no habría sido posible sin ustedes.

Tabla de Contenido

Índice de Tablas

Índice de Figuras

Capítulo 1

Introducción

Sommerville define modelo de proceso de software como "Una representación simplificada de un proceso de software, representada desde una perspectiva específica. Por su naturaleza los modelos son simplificados, por lo tanto un modelo de procesos del software es una abstracción de un proceso real."[18] Los modelos genéricos no son descripciones definitivas de procesos de software; sin embargo, son abstracciones útiles que pueden ser utilizadas para explicar diferentes enfoques del desarrollo de software. Es por esto que, contar con una buena definición de un proceso de software es determinante para lograr calidad de software y productividad los proyectos. Muchas compañías han llevado a cabo la especificación de procesos de software y mejora como proyecto prioritario.

Sin embargo, conceptualizar el modelo de proceso de software demanda un enorme esfuerzo para hacer explícitas prácticas comunes y definir prácticas que no existen aún dentro de la compañía. Estándares como ISO/IEC15504 [5] y modelos de madurez como CMMI [9] son generalmente usados como guías para definir estos procesos. Pero todavía no hay un estándar generalizado para determinar la calidad del proceso especificado, y por lo tanto el retorno de la inversión de la definición del proceso de software no es siempre clara.

Durante los últimos años se trabajó en ayudar a las PyMEs de software en Chile para definir sus procesos de desarrollo en un esfuerzo de mejorar el estándar nacional de la industria. Como parte de esta experiencia práctica, se ha encontrado que hay algunos errores típicos recurrentes en las especificaciones de los procesos de software. Algunos de ellos debido a errores conceptuales en el diseño del proceso y otros errores introducidos durante el proceso de especificación. Pero ninguno de ellos es fácil de identificar, debido a la cantidad de elementos de proceso involucrados en la especificación de un proceso, las múltiples vistas que implican y las notaciones informales que algunas veces pueden introducir ambigüedad.

Esto motivó la creación de la herramienta AVISPA(Analysis and Visualization for Software Process Assesment)[10], una herramienta gráfica que analiza la calidad de un modelo de procesos de software, identificando una serie de patrones de error resaltándolos en diferentes blueprints. Estos blueprints son representaciones gráficas destinadas para ayudar a los diseñadores de procesos de software a evaluar la calidad de un modelo de proceso de software utilizando una fórmula de cálculo en cada caso, identificando las anomalías en un modelos.

Contando con esta herramienta, un ingeniero de procesos el cual es el encargado de definir, mantener y optimizar el proceso de desarrollo de software, sólo necesita analizar los elementos resaltados, demandando menos experiencia, menos conocimiento previo para un análisis de modelos de procesos y añadiendo usabilidad. Un ejemplo de blueprint que considera AVISPA: si un artefacto, no es requerido por ninguna tarea y tampoco es un entregable final, estamos en presencia de desperdicio ya que estamos solicitando un entregable innecesariamente. Otro ejemplo sería un rol que tiene demasiadas tareas asignadas con respecto al resto de tareas asignadas a los otros roles. Esto es considerado como una sobrecarga del rol. AVISPA en ambos casos indicará gráficamente qué blueprint está sustentando la alerta, pero en ningún caso indica las causas y algún tipo de acción ante tal escenario, siempre será requerido el conocimiento experto de un ingeniero de procesos.

1.1. Problema abordado

Actualmente, existe un desafío en la identificación y solución de problemas en la especificación y declaración de procesos de desarrollo de software.

A pesar de contar con herramientas que pueden detectar problemas potenciales, como AVISPA , el proceso de detección de los potenciales problemas está codificado en la propia herramienta y no hay una especificación de alto nivel de cómo AVISPA toma esas decisiones, por lo cual, es muy difícil explicar al ingeniero de procesos, qué significan las distintas alertas que entrega este programa en términos de un modelo conceptual de proceso de software, sin embargo, el blueprint como tal resulta insuficiente para la mejora de procesos y es necesario el conocimiento experto del ingeniero de procesos en caso de tener que corregir la especificación del proceso.

La falta de una solución integral y extensible en un solo entorno dificulta la eficacia en la gestión de estos problemas. Por lo tanto, se requiere una solución que permita abordar estos desafíos al integrar la identificación y solución de problemas en un entorno único, además de ofrecer la capacidad de realizar extensiones para adaptarse a diferentes contextos y necesidades.

1.2. Solución al Problema

La solución al problema es especificar los distintos patrones de errores que provee AVISPA utilizando un lenguaje de alto nivel como OCL, el cual nos permita validar la especificación con AVISPA. Una vez realizada la especificación, se desarrolló una herramienta que asista al ingeniero de procesos con una mejor visualización e integrada al framework de procesos de desarrollo en nuestro caso un plugin para EPFC donde se desenvuelve habitualmente. La razón de utilizar EPFC es porque se han desarrollado herramientas que ayudan en otras aristas del proceso de desarrollo de software[3]. Esta herramienta incluirá los blueprints contenidos en AVISPA brindando la posibilidad de utilizar umbrales de tolerancia, los cuales permitirán al ingeniero de procesos, una mayor flexibilidad a la hora de evaluar la calidad

del proceso, además de mejorar la experiencia al usuario en a la visualización de los resultados e información adicional que AVISPA no provee. Para poder especificar los blueprints, se ejecutó la herramienta AVISPA con procesos ya especificados en 4 empresas, los cuales fueron formalizados utilizando OCL. En base a esto, la solución se compone de los siguientes entregables:

- Una especificación utilizando OCL para cada blueprint implementado en AVISPA.
- Una implementación de un plugin para EPFC(Eclipse Framework Composer) que incorpora los blueprints implementados en AVISPA utilizando umbrales.

Al contar con estos entregables, obtendremos un mecanismo para ayudar al ingeniero de procesos en la validación y mejora de la especificación de su proceso de desarrollo de software.

1.3. Objetivos de la Tesis

El objetivo general es proveer a los ingenieros de procesos de una herramienta que los asista en la especificación de sus procesos de software, detectando potenciales errores en etapas iniciales del desarrollo del proceso de una forma ágil, que interactúa con el entorno de desarrollo donde diseña el proceso, utilizando EPFC. Para lograr este objetivo, se definen los siguientes objetivos específicos:

1. Especificar los errores indicados por los blueprints que muestra AVISPA en lenguaje de alto nivel.
2. Proveer de un entorno para realizar la definición y revisión de procesos de desarrollo de software.
3. Validar la implementación de los blueprints en la nueva herramienta basándonos con los resultados que despliega AVISPA gráficamente.

1.4. Metodología

A continuación se detallan las actividades realizadas.

1. Definir el modelo conceptual de un proceso de software. En esta actividad se definirá la estructura de un modelo de proceso de software, para simplificar la interacción con el ingeniero de procesos.
2. Revisar blueprints de AVISPA. En esta actividad se estudiarán los diferentes blueprints implementados por AVISPA, y su fórmula de cálculo en función del modelo conceptual.
3. Implementar una herramienta dentro del framework EPF que contenga la especificación de los blueprints implementados en AVISPA.
4. Desarrollar la estructura y funcionalidades de la herramienta, basándose en los blueprints de AVISPA.
5. Integrar umbrales de tolerancia en la herramienta para mejorar la detección y advertencia de posibles errores en los procesos de software.

6. Revisar procesos de desarrollo. En esta actividad se recopilarán los procesos de desarrollo que ya fueron especificados por los ingenieros de procesos y se estudiarán que elementos contienen.
7. Ejecutar AVISPA sobre los procesos recopilados. En esta actividad se ejecutará AVISPA sobre todos los procesos recopilados.
8. Validar la nueva herramienta ejecutando los procesos especificados con los resultados obtenidos usando AVISPA.

1.5. Estructura del Documento

En el capítulo I, incluye una breve introducción al problema, los objetivos del trabajo de tesis, la propuesta de solución y la metodología seguir. En el capítulo II, detalla el marco teórico en el cual se introducen los conceptos generales como la definición de un proceso de software hasta conceptos específicos como especificación de procesos de desarrollo de software, sus estándares y las herramientas que serán utilizadas. En el capítulo III, se formaliza utilizando OCL respecto a los patrones de errores de un proceso de desarrollo de software. En el capítulo IV muestra el detalle de la implementación del plugin SPS para EPFC, abarcando una breve historia del framework y las clases involucradas para su desarrollo. En el capítulo V muestra como se validó la herramienta con mediante juicio de expertos. En el capítulo VI las conclusiones y trabajos futuros de este proyecto de tesis. En el capítulo VII los anexos complementarios a este proyecto de tesis.

Capítulo 2

Marco Teórico

Este capítulo define los conceptos relevantes involucrados en esta tesis. Ellos abarcan desde conceptos básicos como lo que es un proceso de software hasta conceptos específicos como especificación de procesos de desarrollo de software, sus estándares y las herramientas que serán utilizadas.

2.1. Especificación de Procesos

El desarrollo de software es un proceso complejo consistente de muchas tareas interdependientes, incluyendo desarrollo técnico, gestión de proyectos, aseguramiento de la calidad y actividades de soporte al cliente. Han existido muchos intentos por capturar las abstracciones necesarias en orden de realizar el desarrollo de software sea una tarea sencilla. En la sección 2.1.1 definiremos los elementos que involucran el desarrollo de un proceso de desarrollo de software y en la sección 2.1.2 los estándares que se han utilizado para realizar dicha especificación.

2.1.1. Niveles de abstracción en los procesos de desarrollo de software

Proceso de Software

El proceso de software se refiere al conjunto de actividades, métodos, prácticas y transformaciones que se utilizan para desarrollar y mantener software y sus productos asociados. Encompassa todo, desde la concepción inicial hasta la entrega final y el mantenimiento.[17]

Especificación del Proceso de Software

La especificación del proceso de software se refiere a la formalización del proceso de software. Esto incluye definir el modelo del proceso, las actividades, las tareas, los roles y el orden en que se realizan. El nivel de formalidad puede variar desde pautas informales hasta estándares rigurosamente definidos.[18]

Modelado del Proceso de Software

El modelado del proceso de software implica representar el proceso de software en una forma visual o textual. Esto puede incluir el uso de diagramas, gráficos o descripciones textuales para describir el flujo de actividades, las dependencias entre tareas y los roles involucrados. Se pueden utilizar varios estándares y notaciones, como el Lenguaje Unificado de Modelado (UML) y la Notación de Modelado de Procesos de Negocio (BPMN), para el modelado del proceso de software.[16]

Estándares en el Modelado del Proceso de Software

Existen varios estándares para el modelado del proceso de software, que proporcionan un lenguaje común y un marco para describir y analizar los procesos de software. Algunos estándares ampliamente utilizados incluyen el Modelo de Madurez de la Capacidad Integrada (CMMI), ISO/IEC 12207, IEEE/EIA 12207 y el Metamodelo de Ingeniería de Procesos de Software (SPEM 2.0).[17][18][16]

2.1.2. Elementos de un proceso

En un proceso de desarrollo de software, los elementos incluyen la definición de roles, tareas y productos de trabajo. Los roles, como desarrollador, analista de requisitos, diseñador y tester, tienen responsabilidades específicas que se relacionan directamente con las tareas que realizan, como la especificación de requisitos, el diseño de software, la implementación y las pruebas. Estas tareas, a su vez, están diseñadas para generar productos de trabajo tangibles, como documentos de requisitos, diseños de arquitectura, código fuente y reportes de pruebas. La correcta definición de roles y tareas, así como la comprensión de cómo se relacionan con los productos de trabajo, es esencial para asegurar la colaboración efectiva entre los miembros del equipo y la entrega exitosa del producto final. [17]

El modelo conceptual de SPEM, representado en la Figura 2.1 describe un ejemplo del modelo conceptual fundamental usando la notación UML para una clase [12].Además permite identificar roles que representan un conjunto de habilidades relacionadas, competencias y responsabilidades en el equipo de desarrollo. Roles son responsables de productos de trabajo (workproducts) específicos. Para crear y modificar productos de trabajo roles son asignados como ejecutores(performers) en tareas(tasks) donde tiposde productos de trabajo específicos son consumidos (inputs) y producidos (outputs).

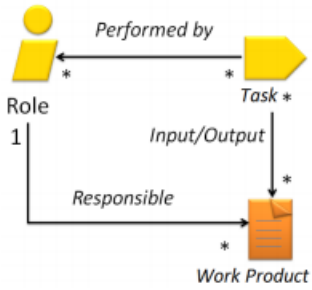

Figura 2.1: Modelo conceptual

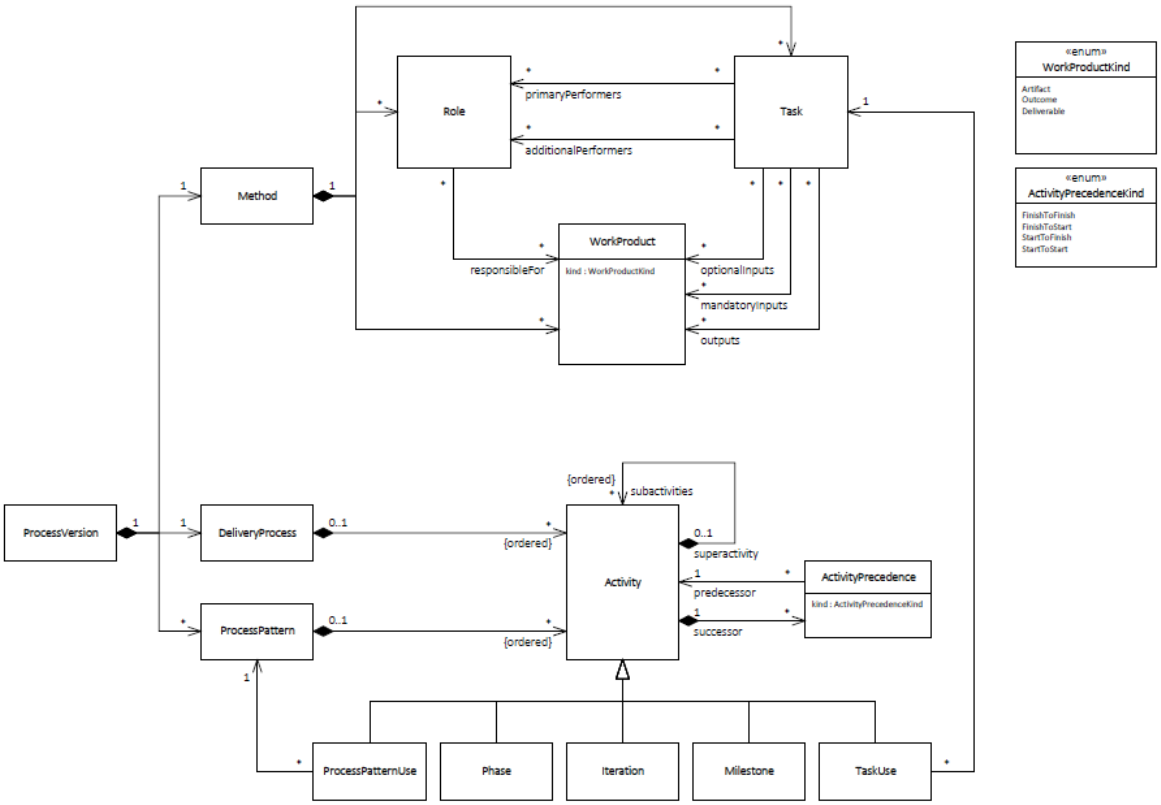

Figura 2.2: Metamodelo Process Versions

Uno de los aspectos es el hecho de la falta de consenso en los conceptos relacionados con los procesos de software. Diferentes propuestas se han desarrollado para recomendar los elementos requeridos en un proceso. Todas ellas coinciden en el principal, es decir, en el proceso de software, pero varían en formato, contenido y nivel de descripción. En la Figura 2.2 describe el metamodelo de esta propuesta realizado por Daniel Perovich [3].Este modelo ha sido utilizado ya que ha sido diseñado para otras áreas de investigación involucradas en procesos de software, en las cuales ya existen herramientas para asistir al ingeniero de proceso. Los elementos que incorpora el metamodelo son los básicos para la definición de los procesos de una organización de software. A continuación una breve descripción de las principales clases que intervienen en el metamodelo y sus respectivas asociaciones, las cuales

serán la base para la especificación de patrones que será discutido en capítulos posteriores.

La metaclase *method* es la clase principal del metamodelo y representa una instancia del proceso de desarrollo, la cual está compuesta de las metaclases *Role*, *Task* y *WorkProduct* las cuales son una especialización de *MethodElement*. A partir de esta metaclase genera las referencias *roles* para la metaclase *Role*, *tasks* para la metaclase *Task* y *workproducts* para la clase *WorkProduct*.

Role(Rol de Proceso): define responsabilidades sobre productos de trabajo específicos y define los roles que ejecutan y asisten en actividades específicas. Posee referencias tales como: *perfomedTasks* para las tareas de las cuales es responsable y *additionalPerformedTasks* para las tareas en las cuales asiste. Además posee la referencia *responsibleFor* la cual indica la referencia hacia *Workproduct*.

Task(Tarea) es la principal subclase de MethodElement. Esta describe una parte del trabajo desarrollado por un Role: las tareas, operaciones y acciones que son desempeñadas por un rol o las que el rol puede asistir. Para referirse al rol que la ejecuta existe una referencia llamada *primaryPerformers* y los roles que asisten a esta tarea está definido por *additionalPerformedTasks*. Las realaciones hacia un *WorkProduct* es *mandatoryInputs* en las cuales esta tarea no se puede realizar sin un producto de trabajo, *optionalInputs* para los cuales los productos de trabajo son opcionales. La referencia hacia la metaclase Workproduct es *outputs*.

WorkProduct(Producto de Trabajo): es cualquier elemento producido, consumido, o modificado por un proceso. Esto puede ser una pieza de información, un documento, un modelo, código fuente, etc. Un WorkProduct describe una clase de producto de trabajo producido en un proceso.

2.1.3. Estándares para formalizar procesos

La multitud de notaciones de modelado de procesos existentes se ha desarrollado debido a diferentes motivaciones y necesidades. Como las necesidades suelen diferir en gran medida para las diferentes partes interesadas, propósitos y contextos, no existe una mejor representación para los procesos, y por lo tanto las representaciones diferentes no pueden evaluarse desde un punto de vista general. Pero puede ser útil comparar diferentes conceptos para comprender los aspectos específicos que aborda una representación específica.

Esta sección presentará conceptos para caracterizar las notaciones de modelado de procesos y, además, definir los requisitos para las anotaciones de modelado de procesos desde diferentes perspectivas. Estos conceptos son útiles para comparar diferentes notaciones para la representación de procesos.

SPEM 1.0

En 1989 se forma el consorcio Object Management Group (OMG), dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos. Es una organización sin fines de lucro que promueve el uso de tecnología orientada a objetos mediante guías y especificaciones. La recomendación del OMG para la definición de procesos de software es el metamodelo SPEM (Software Process Engineering Meta-Model). SPEM ha sido diseñado para describir procesos y sus componentes, siguiendo un enfoque de modelado orientado a objetos con base en UML, es decir que extiende los mecanismos de UML de una forma estandarizada, con el propósito de modelar procesos de desarrollo de software [7].

En la base de SPEM está la idea de que un proceso de desarrollo de software es una colaboración entre las entidades activas abstractas llamadas roles del proceso que realizan operaciones llamadas actividades en entidades concretas, tangibles llamadas productos de trabajo. Múltiples roles interactúan o colaboran para intercambiar productos de trabajo y provocar la ejecución, o representación de ciertas actividades. El objetivo principal de un proceso es traer unos productos de trabajo a un estado bien definido. Para esto, un primer paso consistirá en reestructurar el rol, la actividad y el producto de trabajo. Las actividades utilizan productos de trabajo con el propósito de crear nuevos productos de trabajo. El trabajo es realizado usualmente por personas que desempeñan roles. Ellos realizan las actividades y son responsables de mantener directamente el conjunto de productos de trabajo.

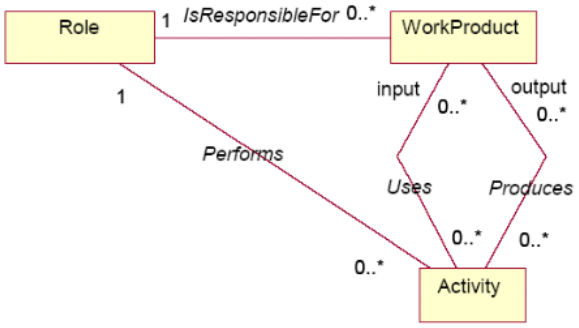

Figura 2.3: Relación de elementos fundamentales de SPEM

UMA

UMA es una evolución de estándar SPEM v1.1 [4] y ha sido desarrollado como unificación de diferentes métodos y lenguajes de ingeniería de proceso tales como [3]: la extensión SPEM para UML, los lenguajes usados para IBM Rational RUP v2003, el proceso unificado, y el método de Servicios Globales de IBM. En 2005, IBM y otros asociados de la OMG comenzaron a trabajar para convertir UMA, junto a algunas mejoras sugeridas por los asociados, en SPEM v2.0, la cual es liberada en Abril de 2008 [1]. Las principales características de UMA son las siguientes:

- Separación del Contenido de método y Proceso: UMA proporciona una clara separación

de las definiciones del contenido del método (Method Content) de su aplicación en los procesos. Esto se obtiene a través de la definición por separado de una

  - Base de conocimiento de desarrollo reutilizable (Method Content) , en forma de descripciones de contenido general como roles, tareas, productos de trabajo y guías.

  - Aplicación específica para cada proyecto donde se combinen y reutilicen los elementos del Method Content.

- Reutilización del contenido: UMA permite que cada proceso haga referencia a contenido de método en un repositorio común. A causa de estas referencias, los cambios en el contenido del método se reflejarán automáticamente en todos los procesos que lo utilicen. Sin embargo, la flexibilidad de UMA permite sobrescribir ciertos contenidos relacionados con el método en un proceso así como definir relaciones individuales específicas del proceso para cada elemento tales como añadir un producto de trabajo a una tarea, renombrar a un rol, o eliminar pasos de una tarea.

- Familias de Proceso: Además de soportar la representación de un proceso de desarrollo específico o el mantenimiento de varios procesos no relacionados, UMA proporciona un conjunto de conceptos para gestionar de forma coherente familias enteras de procesos relacionados. Para esto, en UMA se definen los conceptos de Capability Patterns y Delivery Processes, así como relaciones de reutilización específicas entre estos tipos de procesos. Además, permite obtener distintas variantes de procesos específicos construidos mediante la reutilización dinámica del mismo Method Content y los mismos Capability Patterns, pero aplicada con distintos niveles de detalle y de escala a través de algunos mecanismos de variabilidad.

- Múltiples ciclos de vida: Como metamodelo de ingeniería de procesos, UMA proporciona soporte a diferentes modelos de ciclo de vida de desarrollo e inclusive a combinaciones de estos. Procesos gestionados a partir de ciclos como cascada, espiral, desarrollo iterativo, incremental, evolutivo, entre otros, pueden ser representados como modelos que conforman con el metamodelo UMA.

- Extensibilidad y mecanismos de plug-in: UMA proporciona dos tipos de plug-ins que trabajan de modos distintos. Los Method Plug-ins proporcionan una forma de particularizar y adaptar el Method Content. Del mismo modo, proporciona los Process Plug-ins para obtener variaciones sobre Delivery Process sin modificar su contenido original.

- Múltiples vistas sobre procesos: UMA proporciona diferentes vistas para la definición de procesos. Estas vistas permiten a los ingenieros de procesos enfocar la definición del proceso basándose en sus preferencias personales. Un ingeniero de procesos puede optar por definir sus procesos centrándose en cualquiera de los aspectos siguientes:

  - Work Breakdown (desglose de trabajo): vista centrada en el trabajo, desglosa las tareas asociadas a una actividad de mayor nivel.

  - Work Product Usage (Uso del producto de trabajo): vista basada en resultados,

define el estado de entregables y artefactos en distintos puntos del proceso.

  - Team Allocation (Asignación de equipos): vista basada en responsabilidades, define los roles necesarios y los productos de trabajo asociados.

- Patrones de Procesos reutilizables: Los Capability Patterns son bloques de construcción reusables para crear nuevos procesos de desarrollo. Seleccionar y aplicar uno puede ser realizado mediante:

  - La acción de copiar y modificar, lo que permite al ingeniero de procesos personalizar el contenido acorde a sus necesidades de aplicación.

  - Vinculación dinámica, lo que permite aplicar muchas veces el mismo patrón en un proceso. Si el patrón se modifica, dicha modificación se verá reflejada en todas las aplicaciones del patrón en el proceso.

SPEM 2.0

Además de las ventajas aportadas por la inclusión de UMA a SPEM (detalladas en la sección anterior), se agregan otras características y mejoras a SPEM 2.0 con respecto a sus versiones previas, tales como [1]:

- Al basarse en UML 2.0 toma ventaja de su nueva funcionalidad en la mejora de las técnicas y capacidades del modelado de procesos.

- Define un nuevo esquema XML SPEM, basado en MOF 2.0. Los esquemas XML proporcionan una mayor riqueza y control que en SPEM 1.1.

- Proporciona una guía para la migración de modelos de procesos de SPEM 1.1 a SPEM 2.0.

- Considera la retroalimentación de las primeras implementaciones para identificar inconsistencias relacionadas con la funcionalidad y practicidad de SPEM 1.1.

- Define extensiones para que SPEM utilice herramientas de automatización de procesos.

- Establece una relación más estrecha con otros estándares adicionales a UML como Business Process Definition Meta-model y Business Process Runtime Interfaces.

- Define extensiones al metamodelo de procesos que pueden ser usados tanto en los procesos de desarrollo de software como en los procesos de ingeniería de sistemas.

SPEM 2.0 está estructurado en siete paquetes de metamodelos principales. La estructura divide el modelo en unidades lógicas. Cada unidad extiende la unidad de la que depende, proporcionando estructuras y capacidades adicionales a los elementos definidos:

1. Core: contiene las clases y abstracciones del metamodelo que constituyen la base para crear todas las clases de los demás paquetes.

2. Process Structure: Establece la base para todos los modelos de proceso y soporta la creación de estos en una forma fácil y flexible.

3. Process Behavior: Representa un proceso como una estructura estática, permitiendo anidar actividades y definir dependencias entre ellos.

4. Managed Content: Establece conceptos para la gestión de descripciones y contenido textual que apoyen a los conceptos de un proceso.

5. Method Content: Permite la construcción de una base de conocimientos del desarrollo definiendo conceptos del ciclo de vida y métodos, técnicas y las mejores prácticas.

6. Process with Methods: Define estructuras para integrar procesos definidos con Process Structure con conceptos de Method Content.

7. Method Plugin: Introduce conceptos para diseñar y gestionar repositorios de procesos reutilizables

En la Figura 2.4 se puede apreciar un ejemplo de diagrama construido utilizando el estereotipo UML de SPEM v2.0.

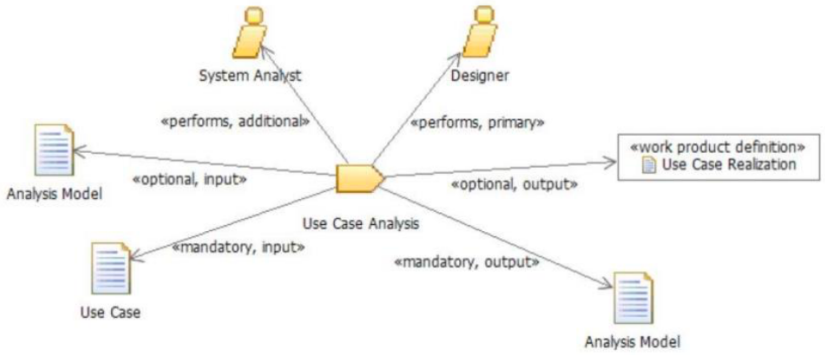

Figura 2.4: Ejemplo de uso SPEM 2.0

2.2. Herramientas de especifiación de procesos

En esta sección incorpora las herrramientas que serán utilizadas en este proyecto de tesis. Para comenzar EPFC como framework base para la implementación de la nueva herramienta, AVISPA software que permite representaciones graficas de potenciales problemas de especificaciones de procesos de desarrollo de software y OCL como lenguaje de especificación.

2.2.1. Eclipse Process Framework Composer EPFC

Eclipse Process Framework Composer (EPF Composer) es una herramienta para ingenieros de procesos, líderes y administradores de proyectos, quienes son responsables de mantener e implementar procesos para organizaciones dedicadas al desarrollo o para proyectos individuales. Se distribuye de forma gratuita bajo licencia GNU.[6] Eclipse EPF Composer tiene dos propósitos básicos:

- Proveer a los desarrolladores con una base de conocimiento de capital intelectual que les permita buscar, administrar y desplegar contenido. Esta base de conocimiento puede ser usada como referencia y material educativo, y forma la base para el proceso de desarrollo. EPF Composer está diseñado para ser un administrador de contenido que provee una estructura de administración y un aspecto comunes para todo el contenido, en vez de ser un sistema administrador de documentos en el cual se almacenan y se acceden documentos difíciles de mantener, que tienen cada uno su propia forma y formato.

- Proveer capacidades de ingeniería de procesos para la selección, adecuación y rápido ensamblado de procesos para proyectos de desarrollo concretos. EPF Composer tiene catálogos de procesos predefinidos para situaciones típicas de procesos que pueden ser adaptados a necesidades individuales.

Entre los principales objetivos del EPF Composer [6] están un conjunto de necesidades comunes de los equipos de desarrollo cuando se enfrentan a la asimilación de métodos y procesos entre las que se pueden citar:

- Acceso fácil y centralizado a la información.

- Formatos estándar que permitan una fácil integración.

- Base de conocimiento actualizada para que ellos mismos aprendan sobre métodos y mejores prácticas.

- Soporte para dimensionar correctamente sus procesos.

- Habilidad de estandarizar prácticas y procesos dentro de las organizaciones.

- Cerrar la brecha entre la ingeniería de procesos y el establecimiento de los mismos en las organizaciones por medio del uso de representaciones y terminologías similares.

EPF Composer se alinea con el metamodelo SPEM v2.0, en su versión obtenida desde UMA, por lo que describe un proceso esencialmente a través de la creación de un Method Content y los respectivos Process.

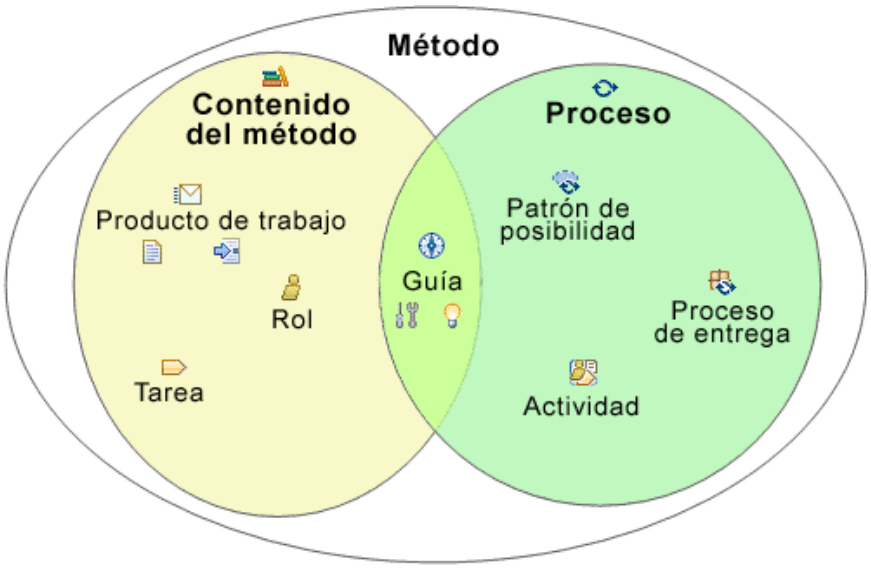

Figura 2.5: Ejemplo de uso SPEM 2.0

Dado lo anterior, para crear un modelo de desarrollo de software en EPF Composer, como muestra la Figura 2.6 se debe crear una Librería de Método, dentro de la cual se debe crear el o los plug-in de procesos correspondientes y la configuración. Dentro del plug-in de procesos, se debe crear el Method Content (con sus respectivos elementos) y el Process correspondiente que utilice los elementos definidos en el Method Content Figura 2.5.

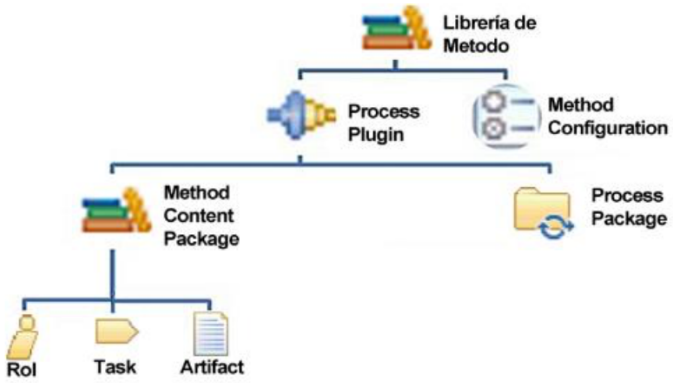

Figura 2.6: Esquema de creación de un modelo en EPFC

Arquitectura de EPFC

EPF Composer es una aplicación Java construida sobre una colección de componentes y estándares libres. La Figura 2.7 muestra la organización de los componentes claves en su arquitectura.

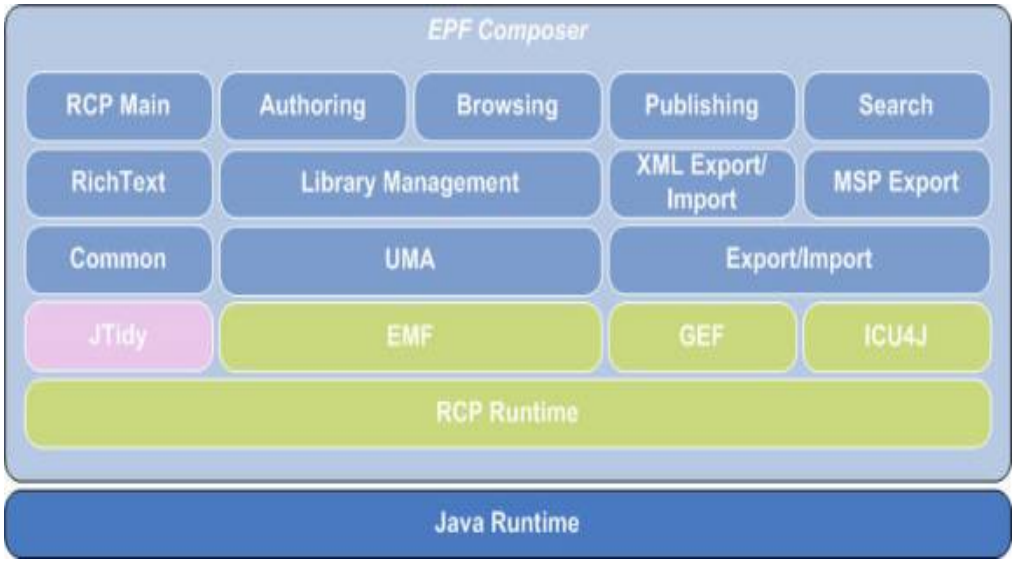

Figura 2.7: Arquitectura de complementos (plugins) de EPFC

Componentes de EPFC

A continuación, veremos una pequeña descripción de los componentes de EPFC

1. Common (Común)

   Este componente provee servicios de infraestructura común a todos los componentes de EPFC. Los servicios clave incluyen, logs, manejo de errores, manipulación de cadenas de caracteres, entrada y salida de archivos, parseo de XML, procesamiento XLST y formateo de HTML.

2. UMA Este componente provee acceso básico y soporte de edición a los métodos y elementos de proceso guardados en una libería de método (method library). El acrónimo UMA significa "Unified Method Architecture". Esto define el metamodelo para como el contenido de método de EPF y los procesos están estructurados. Las clases del modelo UMA pueden ser agrupadas en dos categorías y muchas sub categorías.

   - Method Content (Contenido de Método)

     El Method Content como se muestra en la Figura 2.8, describe roles, las tareas (tasks) que deben realizar, los productos de trabajo (work products) producidos por las tareas y guías de soporte (guidance). Los elementos del Method content son independientes del proceso de desarrollo. Ellos son reusados frecuentemente en múltiples ciclos de desarrollo.

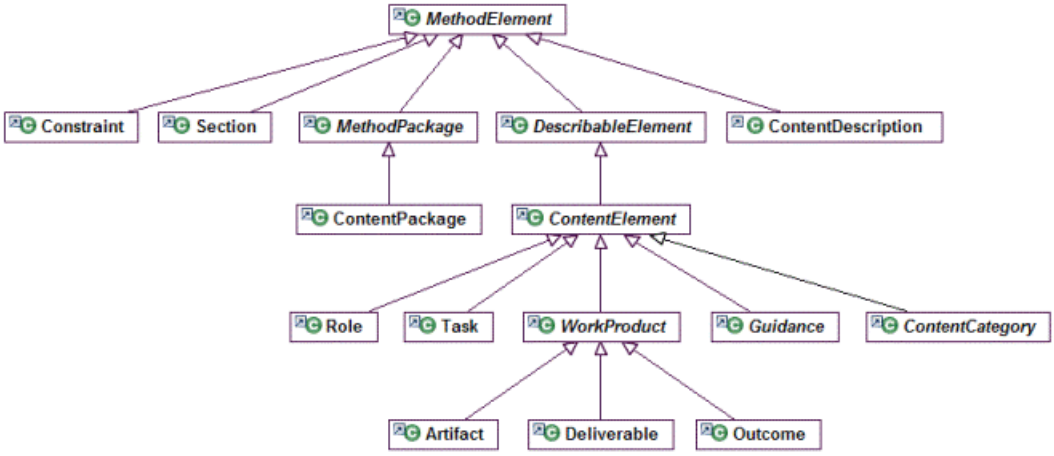

Figura 2.8: Diagrama UML para Method Content en EPFC

2.2.2. AVISPA

Avispa (Analysis and VIsualization for Software Process Assessment) ha sido utilizado como parte de esta tesis. Esta herramienta que construye los blueprints y resalta los patrones de error dado un modelo de proceso. Patrones de errores son identificados con elementos de proceso que son gráficamente "anormalmente diferentes" del resto de elementos[4]. Contando con esta herramienta, el ingeniero de procesos sólo necesita analizar los elementos resaltados, exigiendo menos experiencia y también menos conocimiento previo para un análisis efectivo del modelo de proceso de desarrollo, y añadiendo usabilidad también.

La visualización es un método ampliamente reconocido para detectar anomalías y errores que de otra manera podrían pasar desapercibidos, ya que permite a los ingenieros de software utilizar su capacidad para reconocer patrones cognitivos[15].Existe un extenso cuerpo de investigación que emplea patrones visuales para identificar aspectos positivos o negativos de los sistemas de software [14]. Sin embargo, hasta donde sabemos, ninguno de los trabajos relacionados ha profundizado en el uso de patrones visuales para detectar problemas en modelos de procesos de software.

Con estos patrones se analiza la información sobre la estimación del esfuerzo, y la longitud y secuencia de las actividades de resolución de problemas.

La Figura 2.9 muestra las tecnologías involucradas en la construcción de AVISPA. Se supone que existen modelos de procesos de software especificados en SPEM 2.0. En la parte superior de ellos se definen series de métricas de proceso de software; Estas métricas se utilizarán para identificar errores y oportunidades de mejora. Los blueprints de proceso de software se crean utilizando estas métricas. Los patrones de error se identifican como aquellos elementos o construcciones dentro de blueprints cuyos valores para ciertas métricas satisfacen algunas restricciones. Los elementos identificados se destacan visualmente con AVISPA.

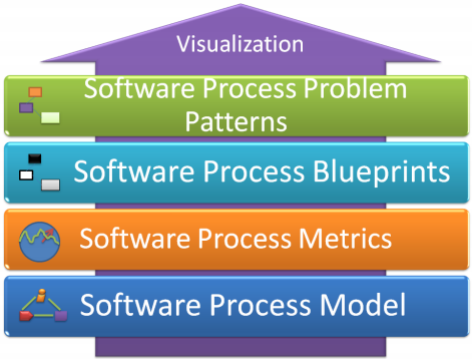

Figura 2.9: AVISPA: Localización de oportunidades de mejora en modelos de procesos de software.

Interfaz de Usuario de AVISPA

AVISPA se ha convertido en una herramienta útil para importar y visualizar modelos de procesos basados en SPEM 2.0. Está construido en la parte superior de Moose y el lenguaje de programación Pharo, y así se beneficia de un gran conjunto de herramientas para la navegación y visualización. La Figura 2.10 muestra la interfaz principal de usuario. El modelo de ejemplo ha sido cargado y listo para ser analizado. El panel de navegación muestra cuatro puntos de entradas para comenzar el análisis. *activities, artifacts, roles y tasks.* La navegación es realizada a través de la información disponible en el metamodelo. Aunque no se representan en la figura, las métricas y otra información específica (Por ejemplo, descripciones y anotaciones) también están disponibles en la pestaña de propiedades. Además permite importar modelos de procesos de software realizados en EPFC a través de archivos XML.

2.3. OCL

El lenguaje OCL (Object Constraint Language) aparece como un esfuerzo para superar las limitaciones de UML cuando se trata de especificar con precisión los aspectos detallados de un diseño de sistema. OCL fue desarrollado por primera vez en 1995 dentro de IBM como una evolución de un lenguaje expresivo dentro del método Syntropy. [13] El trabajo en OCL fue parte de una propuesta conjunta con ObjecTime Limited presentada como respuesta a la RFP por un estándar de análisis y diseño de lenguaje orientado a objetos emitido por el Object Management Group (OMG) [13]. Ese estándar llegó a ser lo que ahora conocemos como UML y OCL se integró en él en 1997. Inicialmente, OCL fue usado solamente como un lenguaje de restricciones para UML, pero rápidamente expandió su alcance y ahora OCL se ha convertido en un componente clave de cualquier técnica de ingeniería dirigida por modelos (MDE) como el idioma predeterminado para expresar todo tipo de requisitos de especificación, manipulación y consulta de modelos (meta). Entre muchas otras aplicaciones, OCL se usa con frecuencia para expresar transformaciones de modelos (como parte de los patrones de origen y destino de las reglas de transformación), reglas de buena formación (como parte de la definición de nuevos lenguajes específicos del dominio o plantillas de generación de códigos). (como forma de expresar los patrones y reglas de generación). Para adaptar el

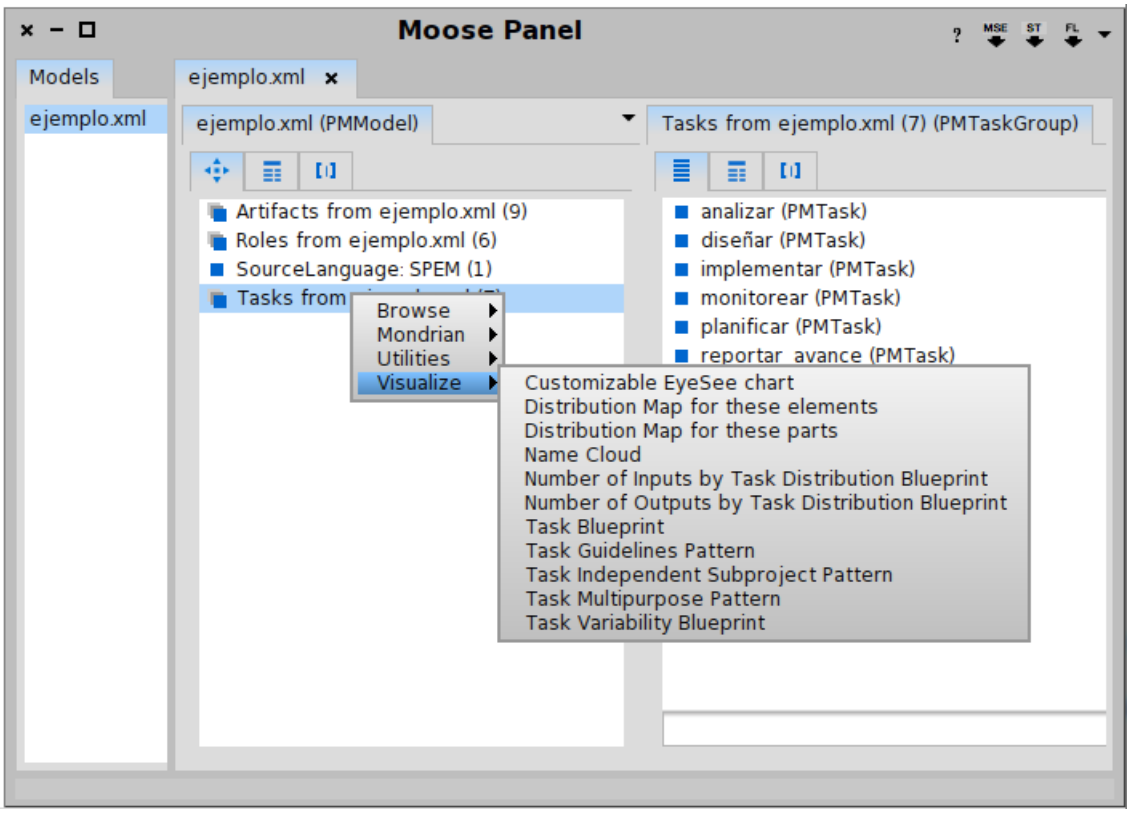

Figura 2.10: Interfaz de Usuario de AVISPA

lenguaje a estas nuevas aplicaciones, varias nuevas (sub) versiones han sido publicadas. Esta sección pretende ofrecer una visión completa de este lenguaje, sus múltiples aplicaciones y el soporte disponible de herramientas.

2.3.1. Lenguajes gráficos vs. textuales en el modelado de dominios

Los lenguajes de modelado gráfico son la opción preferida por muchos diseñadores cuando se trata de definir los aspectos estructurales de un dominio (es decir, sus conceptos principales, sus propiedades y las relaciones entre ellos). El ejemplo más típico de una notación gráfica es UML [8], especialmente su diagrama de clase que es, con mucho, el diagrama UML más utilizado [2]. Sin embargo, esta facilidad de uso tiene un precio. Para mantener el número de elementos notables manejables, los diseñadores de lenguaje deben limitar la expresividad del lenguaje. Esto significa que las notaciones gráficas solo pueden expresar un subconjunto limitado de toda la información relevante de un dominio. Aquí es donde OCL (y en general, cualquier otro lenguaje textual) entra en juego. Son un complemento necesario de la notación UML (u otros lenguajes gráficos) para poder especificar con precisión todos los aspectos detallados de un diseño de sistema. Como un ejemplo, observemos el diagrama de clases de la Figura 2.2 que será utilizado como ejemplo de ejecución.

Este diagrama es un extracto de la especificación de un modelo de proceso de desarrollo de software, el cual contiene al menos Roles(Role), Actividades(Activity), Tareas y productos

de trabajo (WorkProduct) los cuales son consumidos por una tarea en particular o bien son la salida de una tarea.

Esto puede parecer una definición bastante completa del problema, pero en realidad es solo la punta del iceberg. Muchos detalles importantes no se pueden definir simplemente usando la notación disponible para los diagramas de clase UML. Solo por mencionar algunos aspectos que el diagrama UML no responde:

¿Puede una tarea no generar un producto de trabajo? ¿cuántas tareas tiene un rol? ¿puede una tarea producir un producto de trabajo y este no ser utilizado en ninguna otra tarea?

En la siguiente sección mostraremos una breve introducción de OCL el cual será utilizado como lenguaje de especificación para modelos de proceso de desarrollo de software.

2.3.2. Introducción al lenguaje

El propósito de esta sección es brindar una breve descripción informal de OCL y mostrar su usabilidad ejemplificando como puede ser usado.

OCL es un lenguaje formal de propósito general adoptada como estándar por la OMG, usado para definir distintos tipos de expresiones que complementan la información de modelos UML. OCL es un lenguaje tipado, declarativo y libre de efectos colaterales. Tipado significa que cada expresión OCL evalúa a un tipo (ya sea uno de los tipos OCL predefinidos o un tipo en el modelo donde se usa la expresión OCL) y debe cumplir con las reglas y operaciones de ese tipo. Libre de efectos secundarios implica que las expresiones OCL pueden consultar o restringir el estado del sistema pero no modificarlo. Declarativo significa que OCL no incluye construcciones imperativas como asignaciones. Y, finalmente, la especificación se refiere al hecho de que la definición de lenguaje no incluye ningún detalle de implementación ni directrices de implementación. Entre las muchas aplicaciones de OCL, se puede usar para definir los siguientes tipos de expresiones:

Cada restricción es vinculada a un elemento del modelo. Este elemento es llamado el contexto de la restricción. Una palabra especial *self* se refiere al objeto que está en el contexto de la restricción. Si la restricción es un invariante, el contexto es una clase. Si la restricción es una pre o post condición el contexto es una operación de una clase. El elemento por el cual la restricción está siendo especificada es subrayado y la propiedad o atributo es especificado por dos puntos después del elemento. Las restricciones son especificadas en la siguiente línea.

En la Figura 2.11 podemos ver como un modelo de proceso instanciado podemos realizar consultas sobre él. A partir del Method m, consultamos respecto a las referencias de roles y las tareas (tasks) que posee el modelo y los resultados muestran en azul Role r1, Role r2 y para las tareas(Tasks) Task t1 y Task t2.

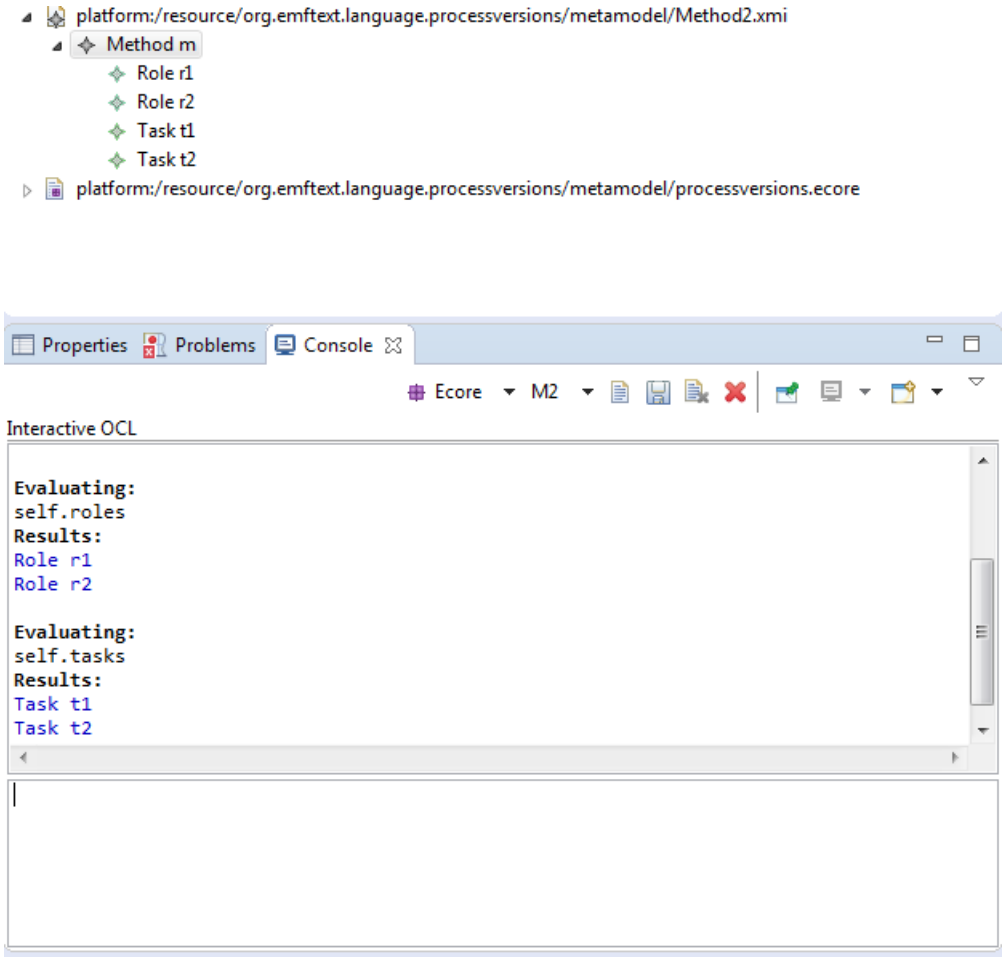

Figura 2.11: Interfaz de Usuario de OCL en Eclipse

2.3.3. Herramientas OCL

OCL no es un lenguaje de programación, por lo que no esperamos un código programable desde este lenguaje. En su lugar, se requieren herramientas para verificar la sintaxis del lenguaje para que sea claro y fácil en el modelado del sistema. Las siguientes son algunas de las herramientas enumeradas a continuación para verificar la sintaxis del idioma:

- En la Universidad de Dresden poseen un OCL parser, escrito en Java.[1]
- El OCL 1.4 un chequeador de sintaxis es una herramienta desde Klasse Objecten que solo realiza chequeos de sintaxis.
- OCL Classic SDK: Ecore/UML Parsers, Evaluator : Provee el evaluador de parseo y soporte de edición para OCL usando Ecore o UML APIs. Este SDK contiene: tiempos de ejecución, soporte de edición EMF, documentación y código fuente para OCL. Esta herramienta es un plugin que funciona en Eclipse y será la base elegida para realizar la nueva herramienta, dada la compatibilidad con Eclipse y la facilidad de uso.

En este capítulo sentamos las bases para poder comprender este proyecto de tesis. Los

elementos de un proceso, el modelo conceptual representado en la Figura 2.3. Además definimos el metamodelo de un proceso de desarrollo de software el cual constituye la base para la especificación de modelos de procesos. En particular hacemos un recorrido por la historia de los estándares para formalización de procesos SPEM 1.0, la transición por UMA creando la herramienta EPFC y posterior migración al estándar actual SPEM 2.0. Explicamos además el origen, los componentes de Eclipse Process Framework Composer que sera la herramienta para especificar procesos de desarrollo, Por otro lado introducimos la herramienta AVISPA que trabaja con los modelos especificados en EPFC la cual es una ayuda al ingeniero de procesos ya que resalta los potenciales problemas según patrones de errores particulares o blueprints. Por ultimo realizamos mención a las limitaciones de UML para especificar restricciones dando origen a OCL como lenguaje no tan solo para especificar restricciones en un modelo UML sino que además nos permite realizar validaciones a modelos de procesos. Además explicamos la forma en que se definen funciones en OCL y un breve ejemplo de como se utilizan las funciones más frecuentes de OCL en esta tesis. Por ultimo mencionamos brevemente las herramientas disponible para trabajar con OCL seleccionando el plugin de OCL para Eclipse.

En el siguiente capítulo, realizaremos la formalización de los patrones de errores utilizando el proceso de ejemplo de la Figura 2.2 mostrando una brevemente los resultados de la ejecución de AVISPA y la formalización utilizando OCL. Por último, para cada patrón de error mostraremos un breve análisis respecto a la evaluación de AVISPA contra OCL.

Capítulo 3

Patrones de Errores

En este capítulo realizaremos una formalización utilizando OCL de los patrones de errores que AVISPA posee. Esta formalización nace debido a que AVISPA es una herramienta muy difícil de mantener y evolucionar (está basado en Pharo y utiliza moose para su representación gráfica) lo cual hace necesario una especificación de los patrones de errores abstrayéndose del lenguaje. Además existen una serie de anomalías en las especificaciones de proceso de software que son recurrentes, los cuales son llamados patrones de error. Aquí se describen algunos de ellos junto con sus consecuencias y cómo se verían en el plano donde pueden encontrarse.

Se ha escogido OCL como lenguaje, ya que, no sólo permite especificar las restricciones a un modelo sino que también nos permite realizar consultas y validar la formalización. Si bien es cierto no es el único lenguaje que permite especificar restricciones a un modelo, fue el único posible de utilizar en la versión de eclipse compatible con EPFC.

3.1. Proceso de Ejemplo

Para realizar esta validación tomaremos como base un proceso de ejemplo cuyas relaciones están basadas en el metamodelo de un proceso de desarrollo de software de la Figura 2.2, el cual consta de:

Elemento	Cantidad
Roles	6
Tareas	7
Artefactos	9

Tabla 3.1: Elementos del proceso de ejemplo

Analizar es una tarea realizada principalmente por el rol de analista y apoyada por el rol de Arquitecto a partir del documento de visión. Esta tarea genera como WorkProduct los documentos de análisis y test_case, como se aprecia en Figura 3.1

Diseñar es una tarea realizada principalmente por el rol de diseñador y apoyada por el arquitecto. Consume el documento de análisis y genera como WorkProduct el documento de diseño.

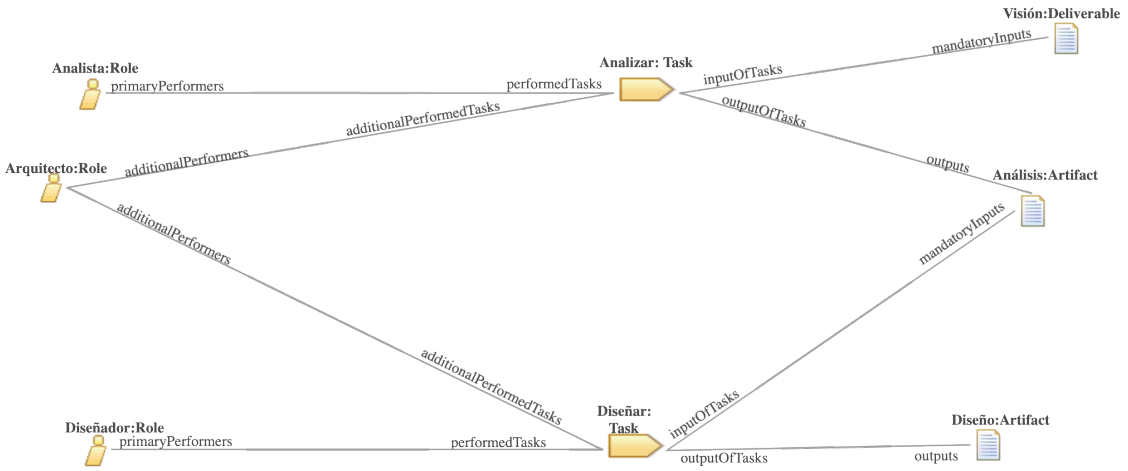

Figura 3.1: Proceso de ejemplo Parte I

Implementar es una tarea realizada principalmente por el Rol de implementador y apoyada por el Rol de Arquitecto. Consume los WorkProduct de documento de diseño, test_case, test_report y genera los WorkProduct de implementacion y test. Testear es una tarea realizada principalmente por el Rol de tester. Consume los WorkProduct implementacion y tests y genera el WorkProduct test_report, como se aprecia en Figura 3.2

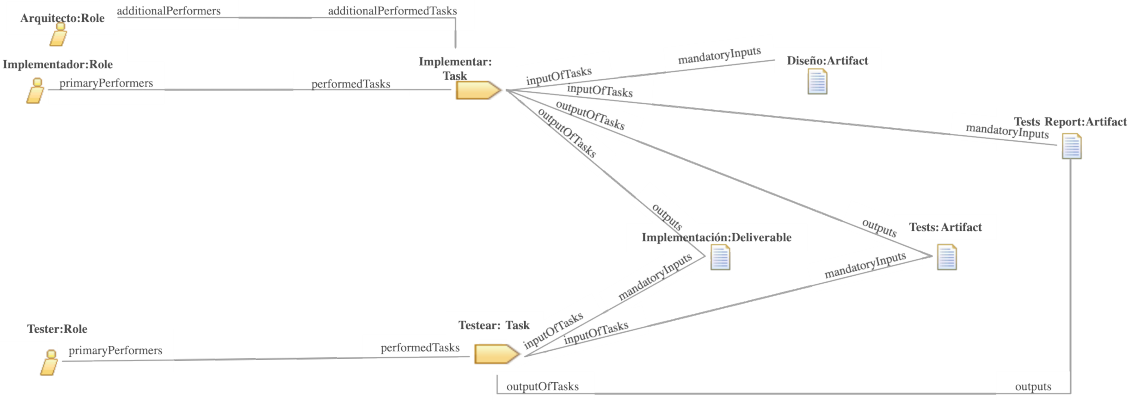

Figura 3.2: Proceso de ejemplo Parte II

Planificar es una tarea realizada principalemente por el rol de Jefe de Proyecto. Genera como WorkProduct el documento Gantt. Monitorear es una tarea realizada principalmente por el rol de Jefe de Proyecto. Consume el WorkProduct Gantt y genera el WorkProduct Gantt (nueva versión). Reportar Avance es una tarea realizada por el rol de Jefe de Pro-

yecto. Consume el WorkProduct Gantt y genera como salida WorkProduct el documento de reporte_de_avance, como se aprecia en Figura 3.3.

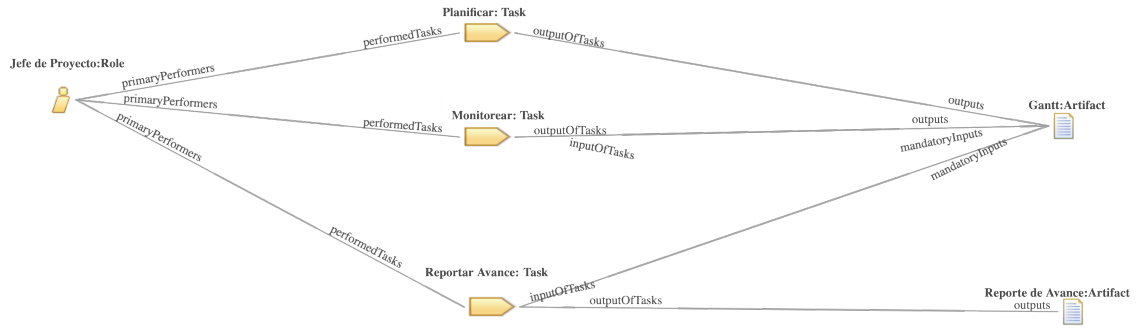

Figura 3.3: Proceso de ejemplo Parte III

3.2. Rol Aislado

Se produce cuando un rol colabora en al menos un cierto umbral de tareas. En general este tipo de patrón de error muestra una especificación errónea. Un rol debería haber sido asignado a tomar parte en una cierta tarea pero parece ser dejado aparte.

- Especificación reportada en AVISPA

  Un rol está aislado cuando no colabora en ninguna tarea.

- Formalización con OCL

```
context Task
def: allRoles: Set(Role)
=self.primaryPerformers->union(self.additionalPerformers)->asSet()

context Role
def: isolated(th: Integer): Boolean
=self.performedTasks->select(t | t.allRoles->size()=1)->size()>th

context Method
def: isolatedRoles(th: Integer): Set(Role)
=self.roles->select(r | r.isolated(th))->asSet()
```

- Evaluación con AVISPA
  En la Figura 3.4 podemos apreciar la salida de AVISPA al aplicar el patrón de error rol aislado. AVISPA dibuja los nodos que representan cada rol en el proceso cuyo tamaño será mayor dependiendo de la cantidad de tareas en las que están involucrados. Las uniones entre los nodos representan las tareas que colaboran entre sí. En este caso resalta con otro color los roles que detecta como aislados, en este caso el rol asociado

al jefe de proyecto y el rol asociado al tester.

Figura 3.4: Patrón de error rol aislado

- Evaluación con OCL

```
self.isolatedRoles(0) = Set{Jefe de Proyecto, Tester}
self.isolatedRoles(1) = Set{Jefe de Proyecto}
self.isolatedRoles(2) = Set{Jefe de Proyecto}
self.isolatedRoles(3) = Set{}
```

- Análisis
  Al comparar los resultados podemos darnos cuenta que comparten el resultado cuando el umbral mínimo de tareas es 0. Aquí estamos definiendo que entendemos por aislado, en este caso si la cantidad de tareas que colaboran es mayor a 0 se considera aislado. En los casos donde el umbral es 2 y 3 tareas respectivamente, solo considera al jefe de proyecto que está aislado y no colabora con ningún otro rol.

3.3. Rol Sobrecargado

Si un rol está involucrado en una gran cantidad de tareas, se convierte en un riesgo. Si falla todas las tareas asociadas también fallarán. Este es un posible error en la concepción del modelo. Una mejor opción podría ser especificar el rol dividiendo sus responsabilidades o reasignando algunas tareas a otros roles. Diremos que un rol está sobrecargado cuando la cantidad de tareas asignadas es mayor a un umbral definido por el usuario.

- Especificación reportada en AVISPA

  Un rol está sobrecargado si es más de una desviación estándar mayor que el tamaño promedio.

- Formalización con OCL

```
context Role
def:overloaded(max:Integer):Boolean
=self.performedTasks->size()>max

context Method
def:overloadedRoles(max:Integer):Set(Role)
=self.roles->select(r |r.overloaded(max))->asSet()
```

- Formalización con OCL del umbral utilizado en AVISPA

```
context Method
def:averageTasks:Real
=(self.roles.performedTasks->size()+
self.roles.additionalPerformedTasks->size())/self.roles->size()

context Role
def:numberOfTasks:Integer
=self.performedTasks->size()+self.additionalPerformedTasks->size()




context Method
def:sumStandardRoles:Real
=self.roles->iterate(e;acc:Real=0 | acc +
(self.averageTasks-e.numberOfTasks)*
(self.averageTasks-e.numberOfTasks))

context Role
def:overloaded(devstd:Real):Boolean
=self.performedTasks->size() > devstd->sqrt()

context Method
def:overloadedRolesAvispa:Set(Role)
```

- Evaluación con AVISPA
  En la Figura 3.5 se puede apreciar que AVISPA ha resaltado en rojo todos los roles, siendo los más grandes arquitecto e implementador.

- Evaluación con OCL

```
self.overloadedRoles(0)= Set{Jefe de Proyecto,Tester, Analista,
                             Implementador,Diseñador}
self.overloadedRoles(1) =Set{Jefe de Proyecto}
self.overloadedRoles(2)= Set{Jefe de Proyecto}
self.overloadedRoles(3)= Set{}
```

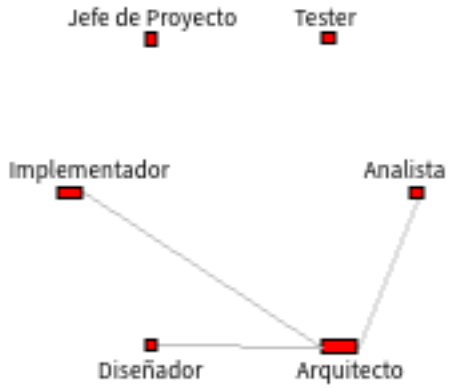

Figura 3.5: Patrón de error rol sobrecargado

- Análisis
  Para este patrón de error AVISPA considera que un rol está sobrecargado si es una desviación estándar mayor que el tamaño promedio, por ese motivo se ajustaría al primer caso de la formalización usando OCL ya que considera que un rol está sobrecargado si tiene más de 0 tareas que realiza. Sin embargo se podría considerar como una restricción muy severa el hecho que un rol realice solo una tarea por lo cual tendría más sentido en los otros casos con más de 1 y 2 tareas en los cuales el jefe de proyecto es el que realiza más tareas.

3.4. Productos de Trabajo Demandados

Productos de trabajo requeridos para un gran número de tareas pueden causar cuellos de botella graves cuando no están disponibles, y por lo tanto este patrón podría revelar un error. La cantidad de tareas es un parámetro definido por el usuario.

- Especificación reportada en AVISPA

  Productos de Trabajo requeridos por muchas tareas son aquellos con más de una desviación por encima de la media.

- Formalización con OCL

```
context WorkProduct
def:demanded(threshold:Integer):Boolean
=self.outputOfTasks->union(self.optionalInputOfTasks)->size()>threshold

context Method
def:demandedWP(threshold:Integer):Set(WorkProduct)
=self.workProducts->select(w | w.demanded(threshold))->asSet()
```

- Formalización con OCL del umbral utilizado en AVISPA

```
context Method
def:averageInputOfTasks:Real
=self.workProducts.inputOfTasks->size()/self.workProducts->size()

context Method
def:sumStandardWP:Real
=self.roles->iterate(e;acc:Real=0 | acc +
(self.averageInputOfTasks-e.numberOfTasks)*
(self.averageInputOfTasks-e.numberOfTasks))

context WorkProduct
def:DemandedWPAvispa(stddv:Real):Boolean
=self.inputOfTasks->size()>stddv->sqrt()

context Method
```

El umbral es calculado la desde una desviación estándar del número de tareas asignadas a los roles con respecto a la media.

- Evaluación con AVISPA
En la Figura 3.6 se puede apreciar que AVISPA ha resaltado en rojo todos los artefactos, siendo el más grande el artefacto carta Gantt, el cual es solicitado por la tarea monitorear y reportar avance. Los vínculos representan la relación de dependencia entre un producto de trabajo y otro.

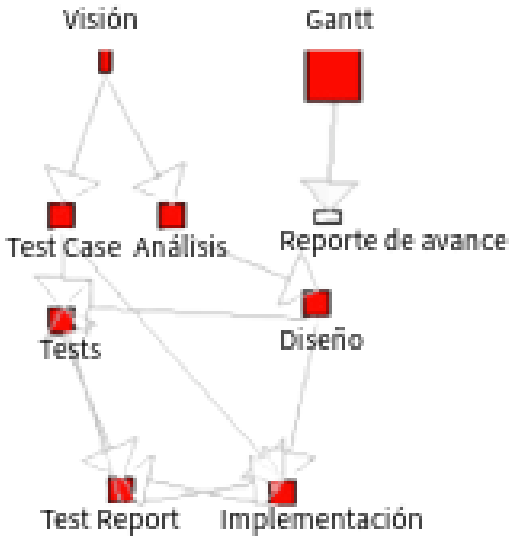

Figura 3.6: Patrón de error productos de trabajo demandados

- Evaluación con OCL

```
self.demandedWP(0)= Set{Test Report, Reporte de Avance, Gantt, Test,
                        Implementación, Analisis, Test Case, Diseño}
self.demandedWP(1)= Set{Gantt}
self.demandedWP(2)= Set{}
```

- Análisis
  Si comparamos los resultados, AVISPA indica que todos los productos de trabajo a excepción del reporte de avance están demandados, sin embargo en la formalización podemos darnos cuenta de dos matices, el primero si consideramos demandado aquel producto de trabajo que es requerido en al menos una tarea y el segundo que al considerar demandado aquel producto de trabajo que es requerido en al menos dos tareas podemos ver que el artefacto Gantt es el cuadrado más grande representado en AVISPA con lo cual cubrimos ambos puntos de vista.

3.5. Productos de Trabajo Basura

Se produce cuando existen productos de trabajo que no están marcados como entregables o entrada para alguna tarea dentro del proceso.

- Especificación reportada en AVISPA

  Son aquellos Productos de Trabajo no que no son requeridos por ninguna tarea y no se encuentran definidos como entregables al cliente.

- Formalización con OCL

```
context WorkProduct
def:waste:Boolean
=self.inputOfTasks->size()=0 and not self.oclIsTypeOf(Deliverable)

context Method
def:wasteWP:Set(WorkProduct)
=self.workProducts->select(w | w.waste)->asSet()
```

- Evaluación con AVISPA
  En la Figura 3.7 se puede apreciar que AVISPA ha resaltado en color azul todos los artefactos que son waste, en particular el artefacto reporte de avance, el cual no es entrada de ninguna tarea. Los vínculos representan la relación de dependencia entre un producto de trabajo y otro.

- Evaluación con OCL

```
self.wasteWP = Set{Reporte de Avance}
```

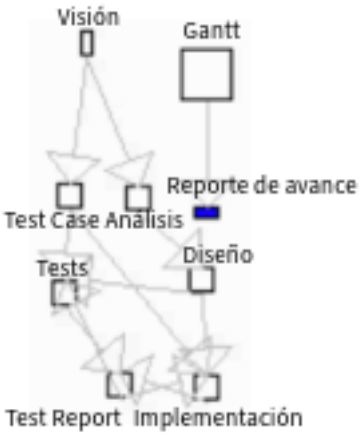

Figura 3.7: Patrón de error productos de trabajo basura

- Análisis
  En ambos casos se puede apreciar el mismo resultado ya que existe un único artefacto reporte de avance que no es entrada de ninguna tarea y tampoco es un entregable del proceso.

3.6. Tareas Multipropósito

Un proceso donde las tareas tienen demasiados productos de trabajo de salida pueden revelar que esas tareas no están especificadas con la granularidad apropiada. Una tarea con demasiados productos de trabajo de salida puede ser muy complejo ya que su meta no es única. Esto puede reflejar un concepto erróneo del proceso.

- Especificación reportada en AVISPA

  Tareas con demasiados productos de trabajo con más de una desviación mayor que la media.

- Formalización con OCL

```
context Task
def:multiPurpose(threshold:Integer):Boolean
=self.outputs->size()>threshold

context Method
def:multipurposeTasks(threshold:Integer):Set(Task)
=self.tasks->select(t | t.multiPurpose(threshold))->asSet()
```

- Formalización con OCL del umbral utilizado en AVISPA

El umbral para este caso es calculado entre la distancia desde el número de productos de trabajo de salida a la media es más grande que una desviación estándar.

```
context Method
def:meanOutWP:Real
=self.tasks.outputs->size()/self.tasks->size()

context Task
def:multiPurposeAvispa(mean:Real):Boolean
=(self.outputs->size() - mean) > ((self.outputs->size()- mean)*(self.outputs->size()- mean))

context Method
def:multipurposeTasksAvispa:Set(Task)
=self.tasks->select(t | t.multiPurposeAvispa(self.meanOutWP))
                    ->asSet()
```

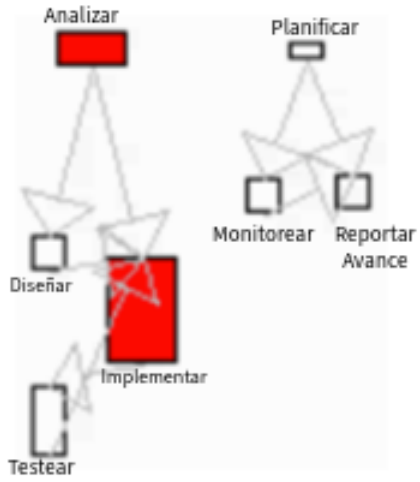

Figura 3.8: Patrón de error tareas multipropósito

- Evaluación con OCL

```
self.multipurposeTasks(0)= Set{Reportar Avance, testear, diseñar,
                             implementar, monitorear, analizar,
                             planificar}
self.multipurposeTasks(1)= Set{Implementar, analizar}
self.multipurposeTasks(2)= Set{}
```

- Análisis Utilizando ambas herramientas entregan los mismos resultados cuando el umbral es 1. Las tareas Implementar y analizar son destacadas en el caso de AVISPA en rojo como muestra la Figura 3.8, siendo la tarea de implementar la menos granulada ya que tiene muchos productos de trabajo de salida.

3.7. Tareas Desconectadas

Tener tareas desconectadas revela una falta de especificación en el modelo de proceso porque las tareas y productos de trabajo deben ser útiles para los objetivos del proyecto, y como tales deben estar conectados.

- Especificación reportada en AVISPA

  Teniendo en cuenta que el modelo de proceso especifica la forma de proceder cuando se trabaja en un proyecto único, es conceptualmente extraño tener subgrafos desconectados, en el blueprint de tareas. AVISPA indicará un grafo con más de un nodo de color indica que hay tareas desconectadas.

- Formalización con OCL

```
context Task
def:followingTasks:Set(Task)
=self.outputs->collect(wp | wp.inputOfTasks->
union(wp.optionalInputOfTasks))->asSet()

context Task
def:previousTasks:Set(Task)
=self.mandatoryInputs->union(self.optionalInputs)->
collect(wp|wp.outputOfTasks)->asSet()

context Task
def:connectedTasks:Set(Task)
=self.previousTasks->union(self.followingTasks)->including(self)->asSet()

context Task
def:allconnectedTasks:Set(Task)
=self.connectedTasks->closure(t|t.connectedTasks)->asSet()

context Method
def:subgraphsofTasks:Set(Set(Task))
=self.tasks->collectNested(t|t.allconnectedTasks)->asSet()
```

- Evaluación con AVISPA
  En la Figura 3.9 se puede apreciar que AVISPA ha agrupado en dos colores verde y amarillo todos los conjuntos de tareas que tienen una entrada en común. El conjunto en verde son las tareas de: analizar, diseñar, implementar, testear y el conjunto en amarillo corresponde a las tareas de planificar, monitorear y reportar avance.

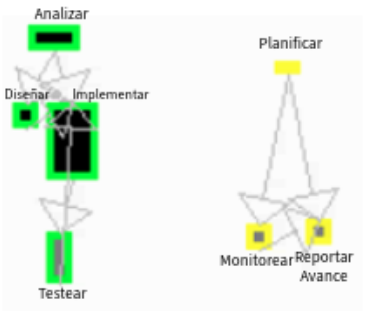

Figura 3.9: Patrón de error tareas desconectadas

- Evaluación con OCL

```
self.subgraphofTasks= Set{
                        Set{Planificar, Reportar Avance, Monitorear},
                        Set{Testear, Analizar, Diseñar, Implementar}
                        }
```

- Análisis Utilizando ambas herramientas entregan los mismos resultados. Las tareas de monitorear, planificar y reportar avance en forma disjunta de las tareas analizar, implementar, testear lo cual es un error ya que las tareas del jefe de proyecto deben estar ligadas entre sí indicando un posible error de especificación.

Capítulo 4

Diseño y Construcción

En este capítulo mostraremos la implementación del plugin SPS (Software Process Smells) para Eclipse Process Framework. En la primera sección brevemente realizaremos un recorrido acerca de los plugins en Eclipse, los componentes gráficos SWT, JFace y el entorno de desarrollo del plugin (PDE). En la siguiente sección detallaremos la implementación del Plugin SPS, las clases involucradas para entender su mecanismo de funcionamiento. Por último desarrollaremos la implementación de dos patrones de errores que conforman parte de nuestro plugin: Rol Sobrecargado y Tareas Desconectadas

4.1. Plugins en Eclipse

Eclipse está hecho de un núcleo pequeño, con muchos complementos superpuestos en la parte superior. Algunos complementos no son más que bibliotecas que otros complementos pueden usar. Estas son las muchas herramientas a su disposición para el trabajo. Las bibliotecas base utilizadas por todos los complementos son:

- **Standard Widget Toolkit (SWT)**. Componentes gráficos utilizados en todas partes en Eclipse: botones, imágenes, cursores, etiquetas, etc. Clases de administración de diseño. Esta biblioteca generalmente está destinada a ser utilizada en lugar de Swing.

- **JFace**.Clases para menús y barras de herramientas, diálogos, preferencias, fuentes, imágenes, archivos de texto y clases base para asistentes.

- **Plugin Developer Environment (PDE)**. Clases para ayudar con la manipulación de datos, extensiones, proceso de compilación y asistentes.

- **Java Developer Toolkit (JDT)**. Clases utilizadas para manipular programáticamente código Java.

Cada uno de estos tiene su especialidad, y algunos pueden usarse de forma independiente (aunque pueden depender internamente de otros). Por ejemplo, como se observa en la Figura

4.1, SWT no tiene que usarse solo para complementos, se puede usar para crear aplicaciones independientes que no sean Eclipse. Hay otras bibliotecas que no figuran aquí también.

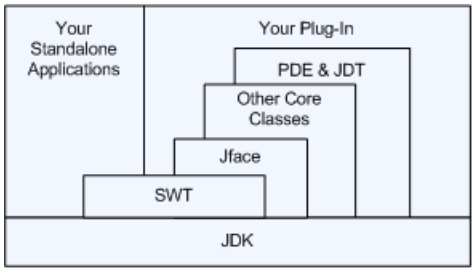

Figura 4.1: Capa de Librerías de EPFC

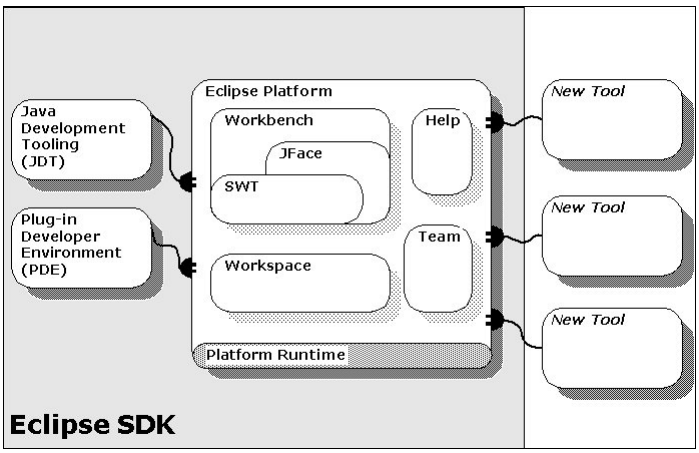

Figura 4.2: Arquitectura SDK

4.2. Diseño de la herramienta

En el capítulo anterior realizamos una formalización utilizando OCL a cada uno de los smells. Sería lógico utilizar OCL como parte del diseño de nuestra herramienta. Sin embargo, la versión de EPFC (Eclipse Galileo) es incompatible con la versión de OCL. Se intentaron probar varias librerías como Dresden-OCL[1] pero no hubo buenos resultados.

A pesar de esto, al contar con la especificación en OCL estamos en condiciones de poder implementar en cualquier lenguaje de programación en particular, Java. La elección del lenguaje es basada en la utilización que tiene EPFC como plataforma para desarrollar modelos de procesos. Existen contribuciones realizadas para esta plataforma como CASPLE[CASPLE] plugin dedicado a realizar tailoring de procesos de desarrollo de software.

Para esta solución se considera como base el entorno de EPFC, ya que el ingeniero de procesos define el proceso utilizando esta herramienta. La idea principal es tener una plataforma centralizada donde el ingeniero de procesos pueda ir ajustando inmediatamente el diseño del proceso de desarrollo.

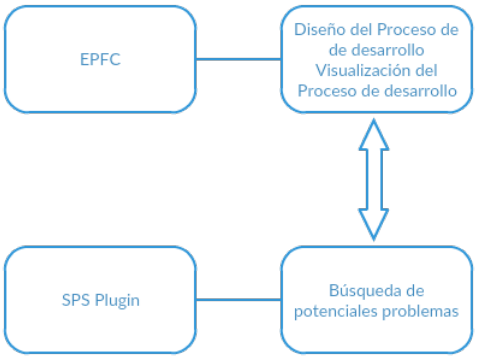

Figura 4.3: Diseño Conceptual

A nivel conceptual, el ingeniero de procesos estará utilizando solo un software, EPFC, al cual nos integraremos con la nueva herramienta. Esta herramienta leerá el modelo de proceso de desarrollo y buscará posibles problemas, proporcionando al ingeniero de procesos información sobre dónde debería realizar al menos una revisión.

A nivel de packages el nuevo plugin utiliza las siguientes dependencias dentro de org.eclipse.epf correspoden a :

- Uma. Incorpora las clases representativas de UMA para el recorrido dentro de un proceso
- Import. Permite leer a través de XML un modelo de proceso
- Authoring. Incorpora las clases para realizar la interfaz de usuario

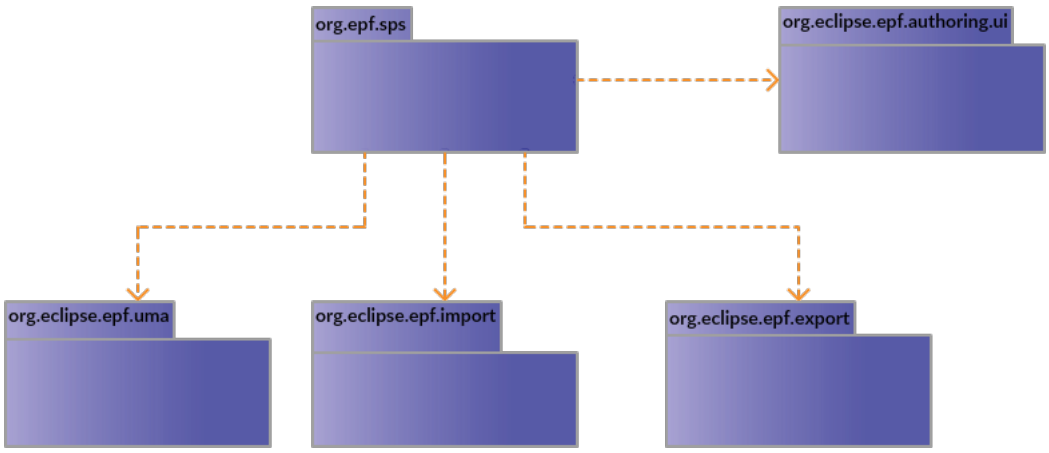

Figura 4.4: Dependencias del Plugin SPS

4.3. Plugin SPS

En esta sección se describe la implementación del plugin para eclipse SPS (Software Process Smells) el cual se integra sobre Eclipse Process Framework con el fin de centralizar todo en una sola herramienta que permita al ingeniero de procesos poder ir visualizando y controlando el desarrollo del proceso. Este plugin incluye los blueprints desarrollados en AVISPA además de la posibilidad de especificar umbrales los cuales en AVISPA son calculados matemáticamente utilizando herramientas de estadística descriptiva.

4.3.1. Vista

Una vista es frecuentemente usada para navegar una jerarquía de información, abrir un editor o desplegar información adicional para alguna cosa abierta en el editor activo. Modificaciones que son realizadas en una vista son guardadas inmediatamente. Su implementación por defecto es llamada ViewPart.

Las vistas pueden ser movidas a cualquier parte de la página y pueden ser minimizadas. Existe generalmente solo una instancia de una vista dada por cada página del workbench. Cada vista tiene su propia barra de herramientas y menú. Ellas también permiten a contribuir botones para la barra de herramientas y menú ítems para el menú principal.

En la Figura 4.5 se observa la herencia desde ViewPart.

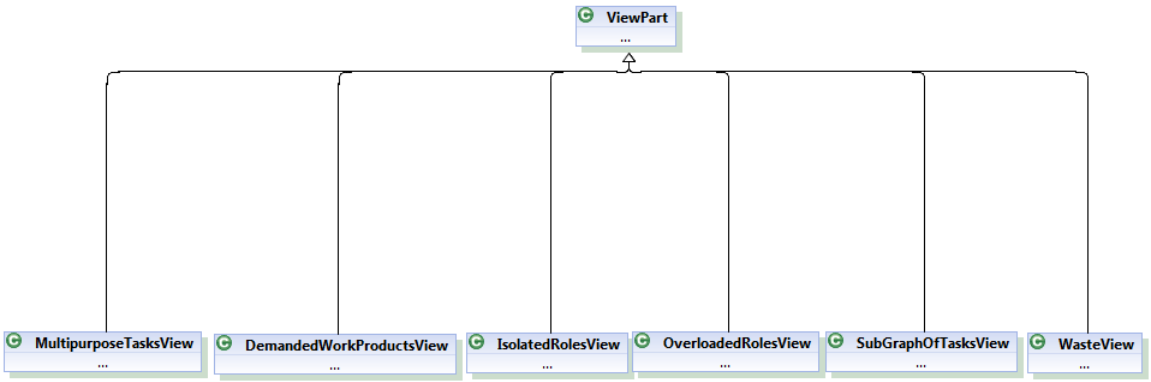

Figura 4.5: Vistas en SPS

Debemos ahora realizar el layout de nuestra aplicación, la cual se compone de: Un campo donde ingresamos el umbral con el cual queremos trabajar. Dos botones uno para calcular el software process smell en función del umbral y otro que calcula automáticamente utilizando el mismo umbral que utiliza AVISPA. EPFC utiliza SWT para la interfaz del usuario por lo que debemos utilizar GridLayout para poder dividir la ventana con los elementos anteriormente señalado. Esto se puede ver en el Listado 1.

```
parent.setLayout(new GridLayout(7,false));
l1 = new Label(parent,SWT.NONE);
l1.setLayoutData(new GridData(SWT.RIGHT,SWT.CENTER,false,false,1,1));
th = new Text(parent, SWT.BORDER);
th.setLayoutData(new GridData(SWT.FILL,SWT.CENTER,true,false,1,1));
b1 = new Button(parent,SWT.NONE);
b1.setLayoutData(new GridData(SWT.RIGHT,SWT.CENTER,false,false,1,1));
b2 = new Button(parent,SWT.NONE);
b2.setLayoutData(new GridData(SWT.RIGHT,SWT.CENTER,false,false,1,1));
b1.setText("Actualizar");
b2.setText("Calcular TH automático");
```

Listado 1: Código layout interfaz gráfica

4.3.2. Menú y Acciones

Eclipse PDT posee un submenú de extensiones, las cuales nos permiten poder crear menús y a estos asignarles distintas acciones, sin embargo, eclipse separa el menú de su acción.

Las acciones de menú en Eclipse se incorporan al menú principal mediante el uso del punto de extensión org.eclipse.ui.actionSets. Este punto de extensión permite contribuir acciones al menú principal y a otros menús en el entorno de desarrollo integrado (IDE) de Eclipse. Al definir las acciones en un conjunto de acciones y especificar su ubicación en la estructura del menú, es posible personalizar la interfaz de usuario de Eclipse para incluir acciones personalizadas.

En la Figura 4.6 se puede apreciar la definición del menú para el plugin SPS y sus diferentes acciones. En el Listado 2 se observa el codigo XML que implementa el actionSet para SPS junto con la acción para el patrón de error Rol Sobrecargado. El sistema permite colocar íconos y etiquetas personalizables ademas de referenciar a la clases que implementan cada acción.

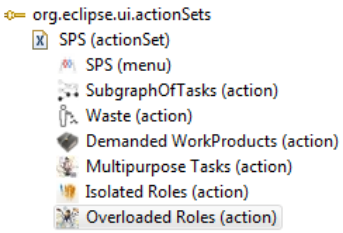

Figura 4.6: Menús y Acciones

```
<extension point="org.eclipse.ui.actionSets">
      <actionSet id="org.sps.epf.ActionSet"
                  label="%SpsActionSet_label"
                 visible="true">
         <menu icon="icons/spsMenuIcon.gif"
               id="org.sps.epf.ui.menu"
               label="%SpsMenu_label"
               path="additions">
         </menu>
         <action class="org.sps.epf.overloaded.actions.OverloadedAction"
               icon="icons/overloaded_role.gif"
               id="org.sps.epf.OverloadedRole"
               label="%Overloaded_label"
               menubarPath="org.sps.epf.ui.menu/additions"
               state="true"
               style="push">
         </action>
      </actionSet>
   </extension>
```

Listado 2: Action Sets en SPS

4.3.3. Implementación

En esta sección mostraremos en detalle la implementación de un software process smells: Rol Sobrecargado y Tareas Desconectadas

Rol Sobrecargado

Como mencionamos anteriormente los menús nos permiten ejecutar acciones y estás acciones permiten mostrar diferentes vistas. En particular se definen acciones para el punto de menú llamado Overloaded Roles (Rol Sobrecargado) que va a buscar en la clase que maneja las acciones OverloadedRolesAction.

```
public class OverloadedAction extends Action
                            implements IWorkbenchWindowActionDelegate{
  private ISelection selection;
  private IWorkbenchWindow window;

  public void dispose(){}
  public void init(IWorkbenchWindow window){this.window = window;}
  public void run(IAction action){
        IWorkbench workbench = null;
        if (this.window != null){workbench = this.window.getWorkbench();}
        if (workbench == null){workbench = PlatformUI.getWorkbench();}
        if (LibraryService.getInstance().getCurrentMethodLibrary() == null){
         MessageDialog.openError(window.getShell(),
         Resources.Overloaded_OverloadedAction_errorNoLibrary_title,
         Resources.Overloaded_OverloadedAction_errorNoLibrary_message);}
        else{
         try{
              window.getActivePage().showView("org.sps.epf.views.OverloadedRolesView");}
         catch (Exception e){e.printStackTrace();}}}
  public void selectionChanged(IAction action, ISelection selection)
    {this.selection = selection;}}
```

Listado 3: Código vista inicial Rol Sobrecargado

Lo más importante de esta clase son dos cosas: la primera es si no se encuentra alguna librería instanciada el programa enviará una alerta de error y terminará. Por otro lado, mostrará la Vista asociada, en este caso la vista que queremos mostrar en EPF es OverloadedRolesView.

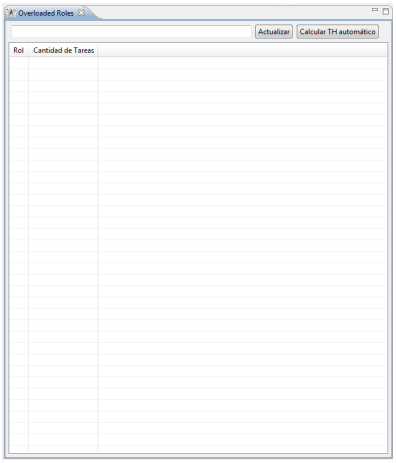

Figura 4.7: Interfaz de Usuario Overloaded Roles

Como vemos en la Figura 4.7 agregamos una grilla con el nombre del rol y la cantidad de tareas que tiene cada rol sobrecargado. Para poder aumentar la usabilidad se incorpora la posibilidad de hacer click en la grilla de resultados para poder rápidamente buscar en la librería el rol que deseamos analizar.

```
table.addSelectionListener(new SelectionListener(){
    public void widgetSelected(SelectionEvent e) {
        LibraryViewFindElementAction action = new LibraryViewFindElementAction("");
        action.selectionChanged(new StructuredSelection(e.item.getData()));
        action.run();}
    public void widgetDefaultSelected(SelectionEvent e) {this.widgetSelected(e);}});
```

Listado 4: Código selección de rol desde SPS en EPFC

En el Listado 4 se realiza la integración entre el nuevo plugin y EPFC utilizando la clase LibraryViewFindElementAction, podemos realizar la búsqueda de un objeto utilizando el método selectionChanged al cual le entregamos como parámetro el objeto que deseamos buscar, en nuestro caso, un Rol, el cual proviene de cada selección en la grilla.

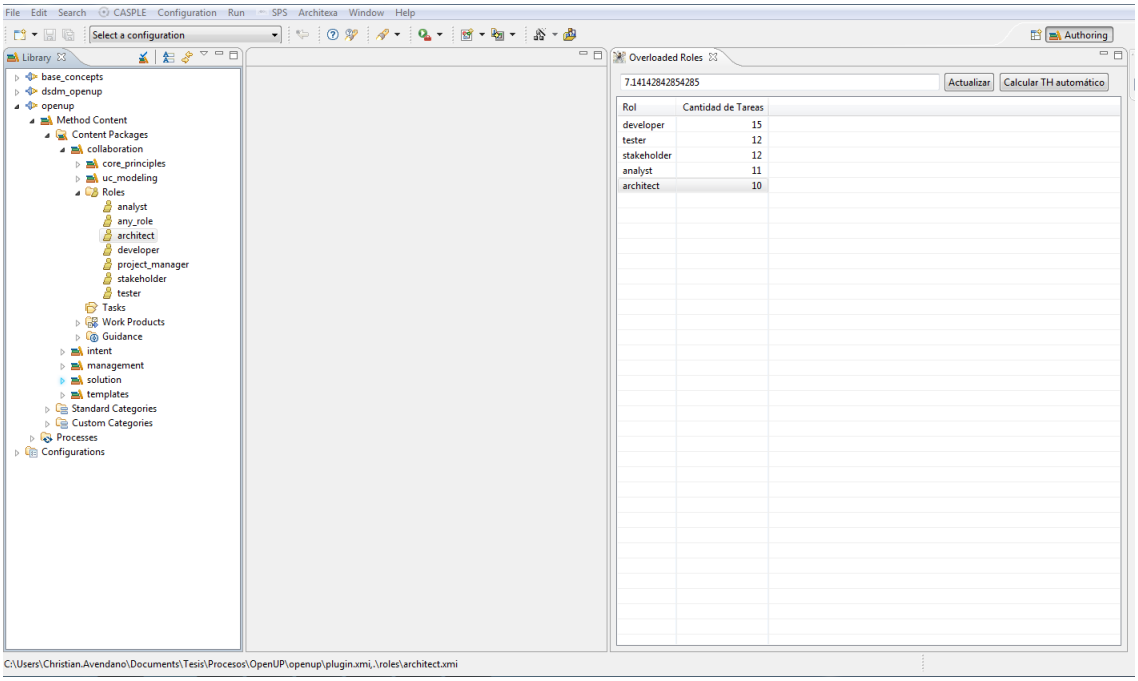

Figura 4.8: Vínculo de EPFC con SPS

En la Figura 4.8 podemos ver la selección sobre el rol architect del proceso OpenUP. En el library nos muestra inmediatamente donde se encuentra el rol para poder realizar modificaciones rápidamente. Ahora analizaremos la implementación del Smell en Java.

Lo primero es comprender como EPFC relaciona Roles, Tareas y Productos de trabajo. En este caso intervienen las siguientes clases:

- MethodPlugin
- MethodPackage
- ContentPackage
- ContentElement

- Task
- Role

```
public void add(MethodPlugin plugin) {
    for (MethodPackage mp : plugin.getMethodPackages()) {this.add(mp);}}
private void add(MethodPackage mp) {
    if (mp instanceof ContentPackage) {this.add((ContentPackage) mp);}}
private void add(ContentPackage cp) {
    for (MethodPackage child : cp.getChildPackages()) {this.add(child)}
    for (ContentElement ce : cp.getContentElements()) {this.add(ce);}}
private void add(ContentElement ce) {
    if(ce instanceof Task) {
        this.add((Task) ce);
        this.cantidadtareas++;}
    if(ce instanceof Role){
        this.addRole((Role)ce);
        this.cantidadroles++;}}
private void addRole(Role r) {
        if (this.counts.containsKey(r)) {
        this.counts.put(r, this.counts.get(r) + 1);}
    else {this.counts.put(r, 0);}}
private void add(Task t) {
        for (Role r : t.getAdditionallyPerformedBy()) {this.add(r);}
        for (Role r : t.getPerformedBy()) {this.add(r);}}
private void add(Role r) {
    if (this.counts.containsKey(r)) {
        this.counts.put(r, this.counts.get(r) + 1);}
    else{ this.counts.put(r, 1);}}
```

Listado 5: Código cálculo cantidad de tareas por rol

Una vez que se termina de recorrer la jerarquía necesitamos recorrer la clase Task ya que dentro de una tarea están las referencias a los roles y una tarea puede tener más de un rol.

La idea principal es guardar en un map la lista de roles con la cantidad de tareas que realiza cada rol. Luego comparamos la cantidad de tareas que fijamos como umbral y guardamos aquellos que excedan el umbral de tareas. Al finalizar, el sistema retorna un map con los roles que cumplen la condición como se muestra en el Listado 6.

```
public List<Role> getOverloadedRoles(double threshold){
    List<Role> roles = new ArrayList<Role>();
           for (Entry<Role, Integer> pair : this.counts.entrySet()){
            if (pair.getValue() > threshold){
                roles.add(pair.getKey());}}
        return roles;}
```

Listado 6: Código lista roles sobrecargados

```
private double calculateThreshold(){
    int sumstandard=0;
        double promedio= this.getCantidadTareas()/this.getCantidadRoles();
        for (Entry<Role, Integer> pair : this.counts.entrySet()){
        sumstandard += Math.pow(promedio - pair.getValue(),2);}
        return Math.sqrt(sumstandard/cantidadroles);}
```

Listado 7: Código lista roles sobrecargados

En el Listado 7 se muestra la implementación del umbral predeterminado que usa utiliza AVISPA para su cálculo interno. Además para facilitar la búsqueda de aquellos elementos que potencialmente podrían tener un problema, podemos ordenar la grilla en forma descendente la grilla final utilizando la clase Collections de Java lo cual se muestra en el Listado 8.

```
Comparator<Role> c = new Comparator<Role>(){
    public int compare(Role arg0, Role arg1){
        int count0 = counts.get(arg0);
        int count1 = counts.get(arg1);
        return count0 == count1 ? 0 : count0 < count1 ? 1 : -1;}};
Collections.sort(roles, c);
```

Listado 8: Código ordenar roles sobrecargados

Tareas Desconectadas

Para lograr implementar este software process smell, debemos considerar lo siguiente:

- Se deben encontrar las tareas que estén conectadas ya sea por entradas o salidas.
- Agrupar los subconjuntos que contienen tareas se encuentran relacionadas por transitividad
- El resultado es un map con un conjunto de elementos de tipo Task. cuyo primer elemento debería ser el proceso completo. Si existen más elementos, se deben considerar como desconectados del proceso y analizar caso a caso.

En el Listado 9, incluimos el pseudocódigo para realizar la búsqueda de las tareas desconectadas. En la Figura 4.9 ejecutamos el smell SubgraphOfTasks. En la primera posición encontramos todas las tareas que conforman la mayor cantidad de tareas del proyecto. Sin embargo existen 3 elementos los cuales debemos analizar por qué no están relacionados.

```
Map<Task,Set<Task>> map;
    for (t in alltasks){
    ts = {t} ∪ allConnected(t);
    map.put(t,ts);}
Set<Set<Task>> sgs;
while(map not empty){
    sg = map.first;
    map.remove(first);
    for(k,v) in map{
        if sg ∩ v ≠ ∅{
            sg.addAll(v);
            map.remove(k);}}
        sgs.add(sg);}
public Set<Task> allConnected(Task t){
    Set<Task> ts = new Set<Task>();
    allConnected(ts,t);
    return ts;}

public void allConnected(Set<Task> ts,Task t){
    ts.add(t);
    Set<Task> cs = connected(t);
    for(c in cs){
        if (c ∉ cs){
        ts.addAll(allConnected(ts,c));}}}
public Set<Task> connected(Task t){
    Set<Task> cs = new Set<Task>();
    Set<WorkProduct> wps = outputs(t);
    for(wp in wps){
    cs.addAll(inputOf(wp));}}
```

Listado 9: Algoritmo para encontrar tareas desconectadas

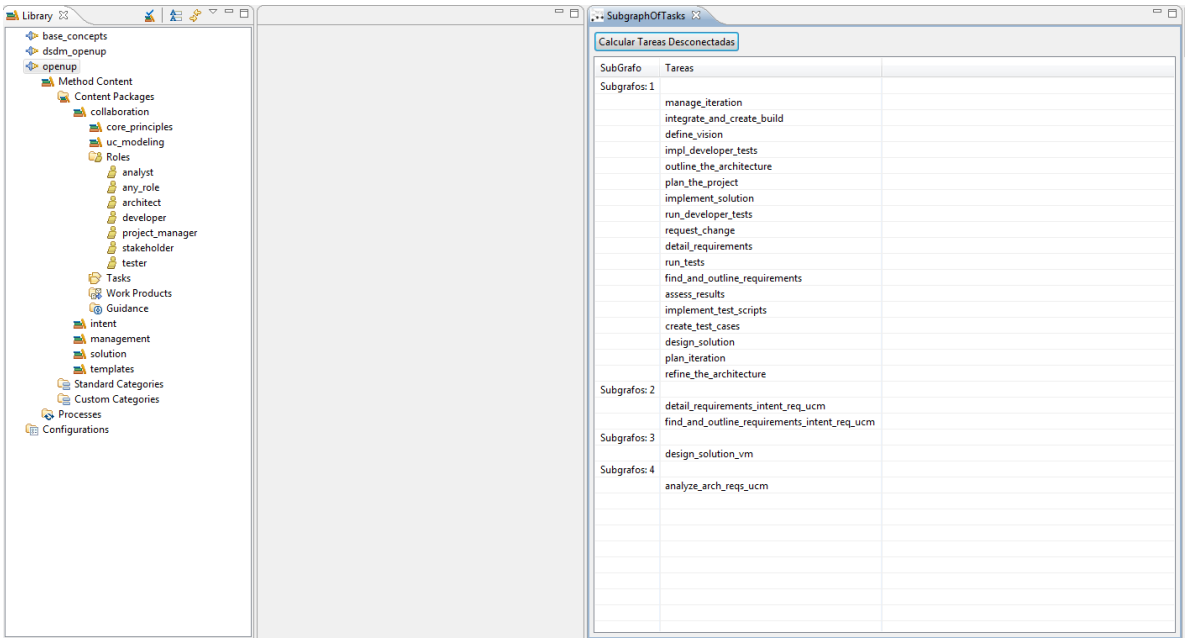

Figura 4.9: Tareas Desconectadas en OpenUP

Capítulo 5

Validación

En este capítulo presenta la validación de la herramienta descrita en el capítulo anterior, utilizando un panel de expertos. En la sección 5.1 se describe la implementación del ambiente de pruebas construido. En la sección 5.2 se reporta la evaluación de la herramienta realizada por un panel de 3 jueces con experiencia en el área.

5.1. Perfil de los Expertos

El panel de expertos fue conformado por profesionales de la ingeniería de software con énfasis en el desarrollo de procesos de desarrollo de software tanto a nivel académico como empresarial.

- Juez N°1. Doctor en Ciencias de la Computación Universidad de Chile, Profesor Titular en la universidad del Cauca Especialista en procesos para el desarrollo de Software
- Juez N°2. Doctor en Ciencia de la Electrónica, Universidad del Cauca, Investigador Universidad del Cauca con especialización en procesos de desarrollo de software
- Juez N°3. Ingeniero Civil en Informática y Telecomunicaciones, Ingeniero de procesos de desarrollo de software con más de 10 años de experiencia en empresas de desarrollo de Software

5.2. Ambiente de Pruebas

Con el fin de proveer un ambiente de pruebas, se utilizó una instancia virtual alojada en el cloud Amazon Web Services con Microsoft Windows Server, Eclipse Process Framework Composer junto con la nueva herramienta desplegada como plugin de EPFC SPS. De esta manera a través de una conexión vía escritorio remoto es posible la conexión de forma remota.

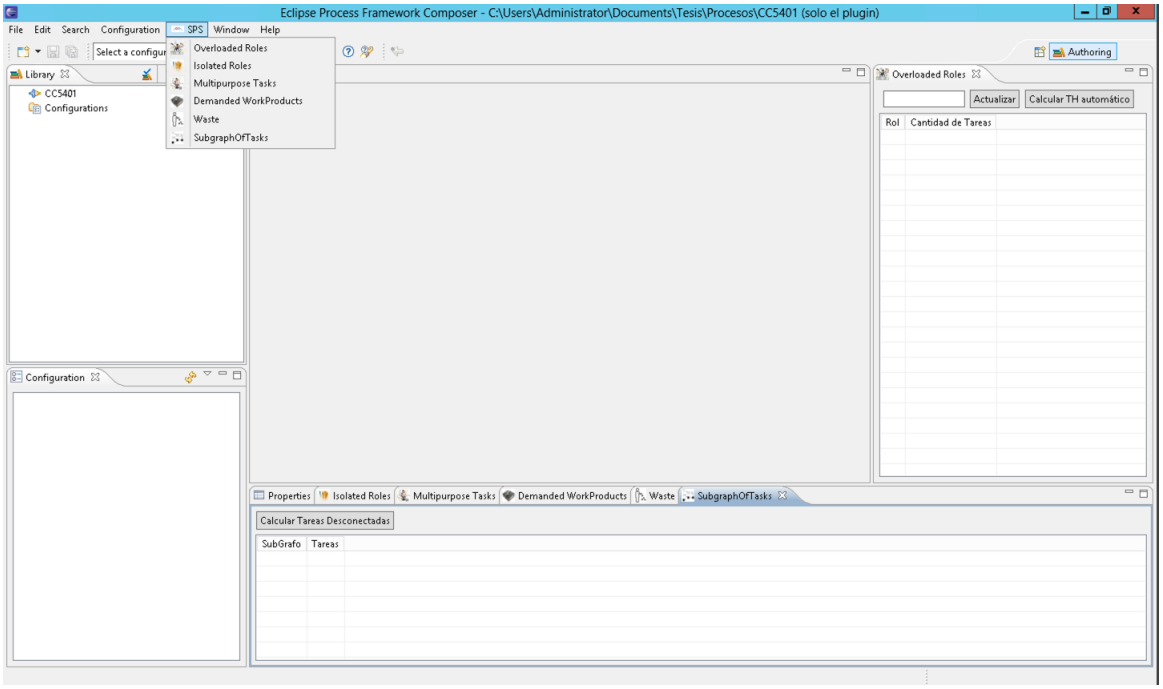

Figura 5.1: Menú Principal EPFC SPS

En la imagen anterior se aprecia el menú principal del nuevo plugin SPS como parte de EPFC. De esta forma el ingeniero de procesos tiene una herramienta para ir probando las distintas funcionalidades a medida que está creando su proceso.

5.2.1. Instrumento para Evaluación

Con el fin de tener un parámetro cualitativo respecto de ciertos aspectos de la nueva herramienta se diseñó una breve encuesta para ser llenada por los usuarios que probarán la herramienta

Instrumento para la evaluación de plugin EPFC Software Process Smells (SPS)

INSTRUCCIONES

En el computador que está utilizando encontrará instalado el *software* identificado EPFC (Eclipse Process Framework Composer). Inicie el *software* haciendo doble clic en el ícono respectivo.

I. Seguidamente, comience a utilizar el *plugin SPS (Software Process Smells)*. Siéntase libre de hacer cualquier pregunta con respecto a cómo utilizarlo. Basta con abrir un proceso de desarrollo de software en EPF y ya puede utilizarlo. Una vez que haya terminado, por favor, evalúe cada uno de los aspectos que se presentan a continuación marcando una X en la columna que mejor represente su apreciación acerca del aspecto a evaluar donde 1 es deficiente y 5 es excelente.

	Aspecto	1	2	3	4	5
1	Menú del plugin en pantalla de inicio					
2	Claridad de la información entregada por Smell					
3	Ubicación de los botones de acción					
4	Interacción entre el plugin y EPF					

II. En esta parte de la evaluación, agradeceríamos sus observaciones, opiniones y comentarios con respecto a cada uno de los aspectos que se enumeran a continuación.

- Diseño gráfico de la interfaz (distribución, cantidad y selección de objetos tales como botones, colores de fondo, texto, a lo largo de las diferentes pantallas del *software*).
- Idoneidad del *software* para la audiencia pretendida (Ingenieros de proceso de software con experiencia en software tales como EPF).

III. Finalmente, responda las siguientes preguntas.

¿Qué ventajas ve usted en el uso de este *software*?

¿Qué desventajas ve usted en el uso de este *software*?

¿Qué *software usaría AVISPA o SPS* ? ¿Por qué?

¿Qué sugerencias o recomendaciones haría para mejorar?

Figura 5.2: Instrumento para la Validación del Plugin SPS

5.3. Ejecución del experimiento

Para la ejecución del experimiento, se llevó a cabo una reunión por videoconferencia, en la cual se presentó el contexto del problema y se expuso la solución de AVISPA junto con el plugin SPS. Ambos están instalados en la máquina virtual que contiene el entorno de desarrollo de procesos EPFC.

Durante la validación de la solución propuesta, se instruyó a los sujetos de prueba que realizaran las siguientes acciones: cargar un proceso en la máquina virtual que contiene AVISPA y el plugin SPS, analizar el proceso de ejemplo definido en el Capítulo 3.1. Cabe señalar que los jueces también podían cargar sus propios procesos compatibles con EPFC. Este enfoque permitió evaluar la usabilidad de la solución en un entorno realista.

5.4. Resultados

De lo anterior se obtienen los siguientes resultados:

El 67 % opina que el Menú del Plugin en la pantalla de inicio es excelente mientras un 33 % que es bueno.

Con respecto a la claridad de la información entregada por el Smell un 67 % opina que es excelente y un 33 % opina que es regular.

Con respecto a la ubicación de los botones de acción todos los expertos concuerdan que es excelente.

Por último la interacción entre el plugin y EPF un 67 % considera excelente y un 33 % bueno.

Con respecto a los comentarios realizados por los expertos se rescata lo siguiente

- Perspectiva Industria
  - Es factible que un entorno empresarial lo utilice dado que es muy fácil su adopción
  - Se sugiere el desarrollo de un ambiente colaborativo para equipos de trabajo
- Perspectiva Investigación
  - Al usar SPS no requiere instalar otros productos de software para la evaluación, se puede ir evaluando a medida que uno va especificando el proceso
  - Destaca la interacción entre el plugin y EPFC, siendo sencillo el seguimiento del mismo.
  - Se sugiere para un futuro la incorporación de elementos gráficos que permitan

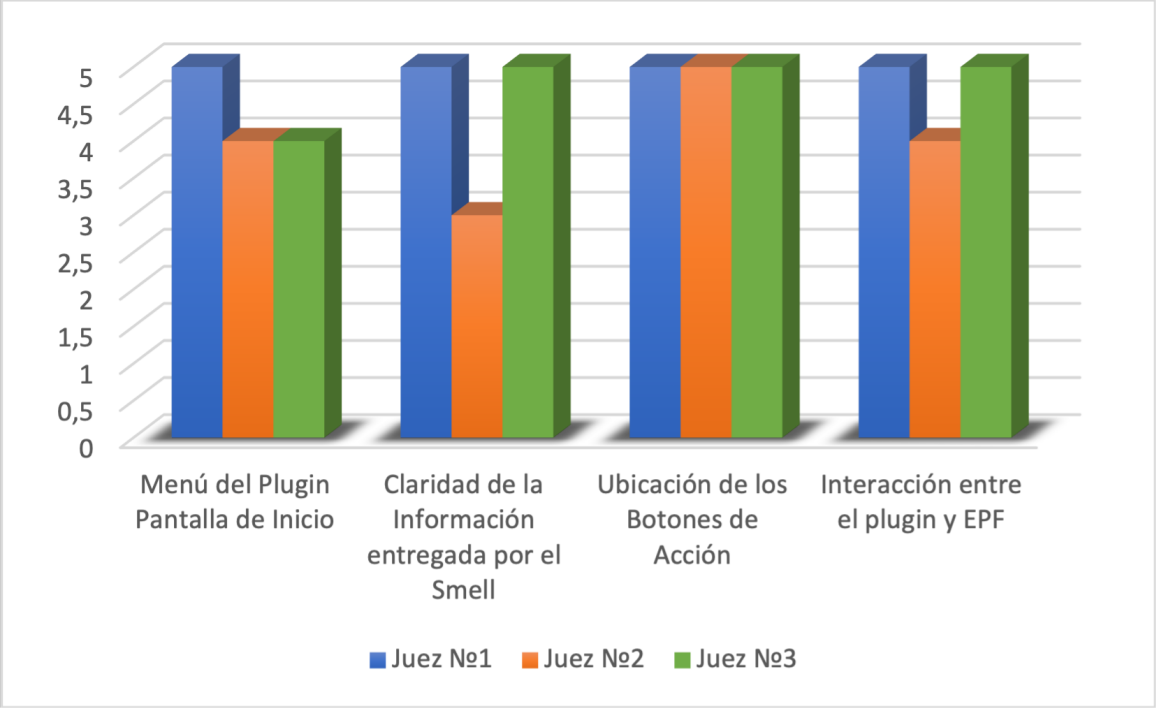

Figura 5.3: Resultados Juicio de Expertos

complementar la visualización de errores.

Como desventaja se encuentra que es un software de escritorio, lo cual dificulta el trabajo colaborativo.

Un elemento destacable por los expertos fue la interacción lograda entre EPFC y el plugin que permite hacer un seguimiento rápidamente al elemento independiente en qué parte del proceso se encuentre. Por otro lado se hace necesaria la implementación de un entorno más colaborativo el cual esta aplicación no cuenta.

Además, al estar EPFC en un lenguaje conocido se hace mucho más fácil su extensibilidad como herramienta a pesar de las librerías y versiones de entorno de desarrollo bastante antiguas.

Capítulo 6

Conclusiones y Trabajos Futuros

A lo largo de este trabajo se estudiaron herramientas para la formalización de especificación de modelos de procesos de software como OCL. Posterior a ello se especificaron en OCL los blueprints contenidos en la herramienta AVISPA, estudiando sus pro y contras lo cual generó la creación de una nueva herramienta, el plugin SPS. Creado sobre EPFC para dar una extensibilidad al framework de tal forma de centralizar las nuevas herramientas al ingeniero de proceso en sólo un entorno de trabajo, posibilitando la interacción directa en la edición de los procesos de desarrollo de software. Para probar la nueva herramienta se utilizaron procesos de desarrollo ya formalizados e incluso procesos de desarrollo como OpenUP y Scrum.

Además, cabe destacar que el plugin desarrollado puede utilizarse, directamente en la plataforma EPFC, logrando reducir errores y el esfuerzo que nace de la validación manual de un modelo de proceso de desarrollo.

Dentro de los mayores desafíos enfrentados durante el desarrollo de este trabajo estuvo la formalización de modelos de procesos y integración con herramientas existentes. Es decir, hubo un extenso estudio previo a la implementación de la solución. El desafío radicaba en que las herramientas no disponen de documentación detallada de ciertos aspectos esenciales para el desarrollo de la solución, sin embargo, fueron una fuente de aprendizaje en relación al día a día que enfrenta un ingeniero civil en informática donde la adaptación al cambio es vital.

Considerando todo lo anteriormente mencionado, se puede concluir que el objetivo general de esta memoria de tesis fue alcanzado. Se provee a los ingenieros de procesos de una herramienta que los asista en la especificación de sus procesos de software, detectando potenciales errores en etapas iniciales del desarrollo del proceso de una forma ágil, que interactúa con el entorno de desarrollo donde diseña el proceso.

En un futuro esto le permitirá a las PyMEs de desarrollo agilizar y obtener modelos de procesos de desarrollo más certeros detectando y corrigiendo inconsistencias de forma temprana.

Dentro de los trabajos futuros a considerar es el estudio en profundidad de los umbrales,

para realmente saber si debe ser un parámetro definido por el usuario o basta con fórmula matemática que AVISPA emplea en sus cálculos. Por otro lado se debe rediseñar el entorno que permita trabajo colaborativo, quizás EPFC se debe redefinir como una herramienta web y se actualice su framework a versiones actuales de Java, ya que existe el riesgo de incompatibilidad en el futuro.

Bibliografía

[1] Demuth B. and Wilke C. Model and Object Verification by Using Dresden OCL. Proceedings of of the Russian-German Workshop Innovation Information Technologies: theory and practice, 2009.

[2] Dobing B. and Parsons J. How uml is used. commun. *ACM*, 49:109–113, 2006.

[3] M. Bastarrica, D. Perovich, J. Marin, and L Rioseco. Process-based project management and spi. *ICSSP 2017 Proceedings of the 2017 International Conference on Software and System Process*, pages 124–133, 2017.

[4] Ducasse S. Demeyer S. and Nierstrasz O. *Object-Oriented Reengineering Patterns*. Square Bracket Associates, 2008.

[5] Int Organization for Standardization. Iso/ iec 15504: Information technology – software process assessment and improvement. technical report, 1998.

[6] Eclipse Fundation. Eclipse Process Framwework Composer. `http://www.eclipse.org/epf`. Accessed: 2018-07-04.

[7] Object Management Group. Object Management Group. 2008. Software & Systems Process Engineering Meta- Model Specification. `http://www.omg.org/spec/SPEM/2.0/`. Accessed: 2018-07-04.

[8] Object Management Group. *UML 2.4.1 Superstructure Specification*. Object Management Group, 2011.

[9] Software Engineering Institute. Sei. cmmi for development, version 1.2. technical report cmu/sei-2006-tr-008, 2006.

[10] Hurtado J. *A Meta-Process for Defining Adaptable Software Processes*. PhD thesis, Universidad de Chile, 2012.

[11] Hurtado J., Lagos A., Bergel A., and Bastarrica M. Software process model blueprints. *International Conference on Software Process*, 2010.

[12] Hurtado J. and Bastarrica C. Hacia una línea de procesos Ágiles agile spsl. Universidad del Cauca, 2005.

[13] Warmer J. and Kleppe A. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison Wesley, 2003.

[14] Ducasse S. Lanza M. Polymetric views-a lightweight visual approach to reverse engineering. *Transactions on Software Engineering*, 29:782–795, 2003.

[15] Fry C. Lieberman H. Zstep 95: A reversible, animated source code stepper. in software visualization — programming as a multimedia experience, 1998.

[16] S. Pfleeger and Atlee. J. *Software Engineering: Theory and Practice*. Pearson, 4th edition, 2009.

[17] R.S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 2014.

[18] Ian Sommerville. *Ingeniería de Software*. 10. Pearson Education, 7th edition, 2005.

Anexos

Anexo A

Implementación de Patrones

Productos Demandados

```
...
public class DemandedWorkProducts {
        private Map<WorkProduct,Integer> workproducts;
        private Map<Role,Integer> roles;
        private int cantidadroles;
        private int cantidadtareas;
        private int cantidadworkproducts;

        public DemandedWorkProducts() {
                this.workproducts = new HashMap<WorkProduct,Integer>();
                this.roles = new HashMap<Role,Integer>();
        }
        public int getCantidadRoles(){ return this.cantidadroles;}
        public int getCantidadTareas(){return this.cantidadtareas;}
        public int getCantidadWorkProducts(){return this.cantidadworkproducts;}

        public int getWorkProductCount(WorkProduct w){return this.workproducts.get(w);}
        public int getRoleCount(Role r){return this.roles.get(r);}
        public double getThreshold(){return this.calculateThreshold();}
        public Map<WorkProduct, Integer> getWorkProductsCounts(){return this.workproducts;}
        public Map<Role, Integer> getRoleCounts(){return this.roles;}
        public void add(MethodPlugin plugin) {
                for (MethodPackage mp : plugin.getMethodPackages()) {this.add(mp);}}
        private void add(MethodPackage mp) {if (mp instanceof ContentPackage) {
                this.add((ContentPackage) mp);}}
        private void add(ContentPackage cp) {
                for (MethodPackage child : cp.getChildPackages()) {this.add(child);}
                for (ContentElement ce : cp.getContentElements()) {this.add(ce);}
        }

        private void add(ContentElement ce) {
                if (ce instanceof Task) {this.add((Task) ce);this.cantidadtareas++;}
                if (ce instanceof WorkProduct){this.cantidadworkproducts++;}
```

```
                if (ce instanceof Role){this.cantidadroles++;}}
        private void add(Task t) {
                for (WorkProduct w : t.getMandatoryInput()) {this.add(w);}
                for (WorkProduct w: t.getOptionalInput()) {this.add(w);}
                for (Role r : t.getPerformedBy()){this.add(r);}}
        private void add(WorkProduct w){
                if (this.workproducts.containsKey(w)) {
                 this.workproducts.put(w, this.workproducts.get(w) + 1);}
        else {this.workproducts.put(w, 1);}}
        private void add(Role r){
                if (this.roles.containsKey(r)) {this.roles.put(r, this.roles.get(r) + 1);}
         else {this.roles.put(r, 1);}}
        private double calculateThreshold()
        {
         int sumstandard=0;
         double promedio= this.workproducts.size()/this.getCantidadWorkProducts();
                for (Entry<Role, Integer> pair : this.roles.entrySet()){
                 sumstandard += Math.pow(promedio - pair.getValue(),2);}
         return Math.sqrt(sumstandard/cantidadroles);
        }
        public List<WorkProduct> getDemandedWorkProduct(){
            return this.getDemandedWorkProducts(this.calculateThreshold());}
        public List<WorkProduct> getDemandedWorkProducts(double threshold){
                List<WorkProduct> wp = new ArrayList<WorkProduct>();
                for (Entry<WorkProduct, Integer> pair : this.workproducts.entrySet()){
                 if (pair.getValue() > threshold){wp.add(pair.getKey());}}
                Comparator<WorkProduct> c = new Comparator<WorkProduct>(){
                 public int compare(WorkProduct arg0, WorkProduct arg1){
                            int count0 = workproducts.get(arg0);
                            int count1 = workproducts.get(arg1);
                            return count0 == count1 ? 0 : count0 < count1 ? 1 : -1;}};
          Collections.sort(wp, c);
          return wp;
        }
}
```

Rol Aislado

```
...
public class IsolatedRoles {
    private Map<Role, Integer> counts;
    private Map<Task,Integer> tasks;
    private Map<Role, List<Task>> roletask;
        private int cantidadroles;
        private int cantidadtareas;
        private int contador=0;
        public IsolatedRoles() {
                this.counts = new HashMap<Role, Integer>();
                this.tasks = new HashMap<Task,Integer>();
                this.roletask = new HashMap<Role,List<Task>>();}
        public int getCount(Role r){return this.counts.get(r);}
        public Map<Role, Integer> getCounts() {return this.counts;}
        public Map<Role,List<Task>> getRoleTask(){return this.roletask;}
```

```
16          public List<Task> getTaskRole(Role r){return this.roletask.get(r);}
17          public void add(MethodPlugin plugin){
18              for (MethodPackage mp : plugin.getMethodPackages()) {this.add(mp);}}
19          private void add(MethodPackage mp) {
20                  if(mp instanceof ContentPackage) {this.add((ContentPackage) mp);}
21          private void add(ContentPackage cp) {
22              for(MethodPackage child : cp.getChildPackages()) {this.add(child);}
23                    for(ContentElement ce : cp.getContentElements()) {this.add(ce);}}
24          private void add(ContentElement ce) {
25                  if(ce instanceof Task) {this.add((Task) ce);this.cantidadtareas++;}}
26          private void addRoleTask(Role r, Task t){
27              if(this.roletask.containsKey(r)){
28                  List<Task> l2 = (List<Task>)this.roletask.get(r);
29                  l2.add(t);
30                  this.roletask.put(r,l2);}
31              else{
32                  List<Task> l1 = new ArrayList<Task>();
33                  l1.add(t);
34                  this.roletask.put(r,l1);}}
35          private void add(Task t) {
36                  List<Role> apby = t.getAdditionallyPerformedBy();
37                  List<Role> pby = t.getPerformedBy();
38                  if(pby.size() ==1 && apby.size() == 0){
39                      this.add(pby.get(0));
40                      this.addRoleTask(pby.get(0), t);}
41                        else if (pby.size()==0 && apby.size()==1){
42                      this.add(apby.get(0));
43                      this.addRoleTask(pby.get(0), t);}}
44          private void add(Role r){
45              if(this.counts.containsKey(r)) {this.counts.put(r, this.counts.get(r) + 1);}
46              else {this.counts.put(r, 1);}}
47          public List<Role> getIsolatedRoles(){
48                  List<Role> roles = new ArrayList<Role>();
49                  for (Entry<Role, Integer> pair : this.counts.entrySet()){roles.add(pair.getKey());}
50                  Comparator<Role> c = new Comparator<Role>(){
51                  public int compare(Role arg0, Role arg1){
52                              int count0 = counts.get(arg0);
53                              int count1 = counts.get(arg1);
54                  return count0 == count1 ? 0 : count0 < count1 ? 1 : -1;}};
55                  Collections.sort(roles, c);
56                  return roles;}
57          public List<Role> getIsolatedRoles(double threshold){
58                  List<Role> roles = new ArrayList<Role>();
59                  for (Entry<Role, Integer> pair : this.counts.entrySet()){
60                          if (pair.getValue() > threshold){roles.add(pair.getKey());}}
61                  Comparator<Role> c = new Comparator<Role>(){
62                          public int compare(Role arg0, Role arg1){
63                                  int count0 = counts.get(arg0);
64                                  int count1 = counts.get(arg1);
65                                  return count0 == count1 ? 0 : count0 < count1 ? 1 : -1;}};
66                  Collections.sort(roles, c);
67                  return roles;}}
```

Tareas Multipropósito

```
...
public class MultipurposeTasks {
        private Map<Role, Integer> counts;
        private Map<Task,Integer> tasks;
        private int cantidadroles;
        private int cantidadtareas;
        public MultipurposeTasks() {
                this.counts = new HashMap<Role, Integer>();
                this.tasks = new HashMap<Task,Integer>();}
        public int getCantidadRoles(){return this.cantidadroles;}
        public int getCantidadTareas(){return this.cantidadtareas;}
        public List<WorkProduct> getOutputTasks(){
        List<WorkProduct> wp = new ArrayList<WorkProduct>();
                for(Entry<Task, Integer> pair : this.tasks.entrySet()){
                  List<WorkProduct> aux = pair.getKey().getOutput();
                for(WorkProduct w:aux){wp.add(w);}}
            return wp;}
        public int getTaskSizeOutput(){
                List<WorkProduct> w1 = this.getOutputTasks();return w1.size();}
        public int getCount(Role r){return this.counts.get(r);}
        public int getTaskCount(Task t){return this.tasks.get(t);}
        public Map<Role, Integer> getCounts() {return this.counts;}
        public Map<Task, Integer> getTaskCounts(){return this.tasks;}
        public void add(MethodPlugin plugin) {
                for (MethodPackage mp : plugin.getMethodPackages()) {this.add(mp);}}
        private void add(MethodPackage mp) {
                if (mp instanceof ContentPackage) {this.add((ContentPackage) mp);}}
        private void add(ContentPackage cp) {
                for (MethodPackage child : cp.getChildPackages()) {this.add(child);}
                for (ContentElement ce : cp.getContentElements()) {this.add(ce);}}
        private void add(ContentElement ce) {
        if (ce instanceof Task) {this.add((Task) ce);this.cantidadtareas++;}}
        private void add(Task t) {this.tasks.put(t, t.getOutput().size());}
        public List<Task> getMultipurposeTasks(){
               List<Task> t = new ArrayList<Task>();
                double mean= this.getTaskSizeOutput()/this.getCantidadTareas();
                        for (Entry<Task, Integer> pair : this.tasks.entrySet()){
                        if(Math.abs(pair.getKey().getOutput().size()- mean) >
        Math.sqrt(Math.pow(pair.getKey().getOutput().size()-mean,2)/this.getCantidadTareas())){
                        t.add(pair.getKey());}}
                        Comparator<Task> c = new Comparator<Task>(){
                                public int compare(Task arg0, Task arg1){
                                    int count0 = tasks.get(arg0);
                                    int count1 = tasks.get(arg1);
                                    return count0 == count1 ? 0 : count0 < count1 ? 1 : -1;}};
           Collections.sort(t, c);
           return t;}
```

```
54          public List<Task> getMultipurposeTasks(double threshold){
55                  List<Task> t = new ArrayList<Task>();
56                  for (Entry<Task, Integer> pair : this.tasks.entrySet()){
57                          if (pair.getValue() > threshold){t.add(pair.getKey());}}
58                  Comparator<Task> c = new Comparator<Task>(){
59                    public int compare(Task arg0, Task arg1){
60                           int count0 = tasks.get(arg0);
61                           int count1 = tasks.get(arg1);
62                       return count0 == count1 ? 0 : count0 < count1 ? 1 : -1;}};
63                  Collections.sort(t, c);
64                  return t;}
```

Productos de Trabajo Basura

```
1   ...
2   public class Waste {
3       private Map<WorkProduct, Integer> workproducts;
4           private Map<Task,Integer> inputs;
5           private int cantidadroles;
6           private int cantidadtareas;
7           public Waste() {
8                   this.workproducts = new HashMap<WorkProduct, Integer>();
9                   this.inputs = new HashMap<Task,Integer>();}
10          public int getCantidadRoles(){return this.cantidadroles;}
11          public int getCantidadTareas(){return this.cantidadtareas;}
12          public Map<WorkProduct, Integer> getCounts(){return this.workproducts;}
13          public Map<Task, Integer> getTaskCounts(){return this.inputs;}
14          public List<WorkProduct> getWaste(){
15          List<WorkProduct> waste = new ArrayList<WorkProduct>();
16          Map<WorkProduct,Integer> inputs = this.getInputTasks();
17                  for (Entry<WorkProduct, Integer> pair : this.workproducts.entrySet(){
18          if(!inputs.containsKey(pair.getKey())&& !(pair.getKey() instanceof Deliverable)){
19                          waste.add(pair.getKey());}}
20                  Comparator<WorkProduct> c = new Comparator<WorkProduct>(){
21                    public int compare(WorkProduct arg0, WorkProduct arg1){
22                                  int count0 = workproducts.get(arg0);
23                                  int count1 = workproducts.get(arg1);
24                                  return count0 == count1 ? 0 : count0 < count1 ? 1 : -1;}};
25                  Collections.sort(waste, c);
26                  return waste;}
27          public void add(MethodPlugin plugin) {
28                  for (MethodPackage mp : plugin.getMethodPackages()){this.add(mp);}}
29          private void add(MethodPackage mp) {
30          if (mp instanceof ContentPackage) {this.add((ContentPackage) mp);}}
31          private void add(ContentPackage cp) {
32                  for (MethodPackage child : cp.getChildPackages()){this.add(child);}
33                  for (ContentElement ce : cp.getContentElements()){this.add(ce);}}
34          private void add(WorkProduct wp){
35                  if(this.workproducts.containsKey(wp)){ this.workproducts.put(wp,this.workproducts.get(wp)+1
36          else{this.workproducts.put(wp,1);}}
37          private void add(ContentElement ce){
38                  if(ce instanceof Task) {this.add((Task) ce);this.cantidadtareas++;}
39                  if (ce instanceof Role){this.cantidadroles++;}
```

```
40                 if (ce instanceof WorkProduct){this.add((WorkProduct)ce);}}
41         private void add(Task t) {
42                 if(this.inputs.containsKey(t)){this.inputs.put(t,this.inputs.get(t)+1);}
43                 else{this.inputs.put(t,1);}}
44         public Map<WorkProduct,Integer> getInputTasks(){
45             Map<WorkProduct,Integer> wp = new HashMap<WorkProduct,Integer>();
46                 for(Entry<Task, Integer> pair : this.inputs.entrySet()){
47                         List<WorkProduct> aux = pair.getKey().getMandatoryInput();
48                         for(WorkProduct w:aux){
49                         if(wp.containsKey(w)){wp.put(w,wp.get(w)+1);}
50                         else{wp.put(w,1);}}}
51                 return wp;}}
```

Anexo B

Enlace a la Herramienta

Para obtener Eclipse Process Framework Composer con el complemento SPS y el proceso de ejemplo, se puede acceder a través del siguiente enlace: https://www.tinyurl.com/epfc-sps