UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

INCORPORACIÓN DE UN PARSER XML - XMI PARA MODELAMIENTO DE PROCESOS COMPUTACIONALES

TESIS PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

ALDO SEBASTIÁN BERTERO GONZÁLEZ

PROFESOR GUÍA:
MARÍA CECILIA BASTARRICA PIÑEYRO

MIEMBROS DE LA COMISIÓN:
CLAUDIO GUTIÉRREZ GALLARDO
LUIS MATEU BRULÉ

SANTIAGO DE CHILE
AGOSTO 2012

1. Resumen

El Proyecto ADAPTE tiene como objetivo optimizar los procesos de negocio de las Pequeñas y Medianas Empresas. Para lograrlo es necesario conocer el proceso en sí y el contexto de cada empresa. Existe una herramienta de nombre Eclipse Process Framework Composer (EPF) que se usa para definir de manera estándar los procesos de desarrollo de software, que posee todas las funcionalidades necesarias para esto. Por otro lado, se utiliza Eclipse Modeling Tools (EMT) para modelar el contexto. Además en esta herramienta se ejecutan las trasformaciones necesarias para adaptar el proceso (genérico) a un contexto en particular.

El problema en lo descrito anteriormente está en que EPF entrega el proceso en un documento XML haciendo uso del metamodelo UMA, mientras que en EMT se reconoce como válido un archivo de extensión XMI, que usa además un metamodelo diferente denominado SPEM. En este documento se propone realizar una aplicación que sirva para convertir archivos XML con metamodelo UMA a XMI con metamodelo SPEM, y viceversa. Se proponen dos soluciones a la problemática, la primera de estas es hacer una aplicación Java que realice la conversión en ambos sentidos. En esta se utiliza programación orientada a objetos, junto con librerías para manejar documentos con estructura XML. La segunda solución es utilizar TCS para definir las sintaxis concretas de ambos metamodelos, y luego utilizar ATL para hacer la transformación entre dichos lenguajes.

Se realizaron tres experimentos por solución, con tres documentos que representan procesos definidos por empresas reales. Para la primera solución, se ingresa el documento XML y se ejecuta la aplicación para obtener el documento XMI, este se visualiza en EMT para comprobar su correctitud, y luego, se realiza la conversión para volver a XML, abriendo el documento con EPF y realizar nuevamente esta comprobación. Para la segunda solución se abre el documento con TCS y se espera que la aplicación reconozca las sintaxis válidas. Los resultados obtenidos presentan a la solución en Java como la más completa de las dos, debido a que se logró hacer la conversión XML a XMI, perdiendo una cantidad importante de información, posteriormente al realizar la conversión XMI a XML se recuperó esta información con el archivo XML original, identificando que estos no hayan sido incorporados originalmente al XMI, obteniendo un proceso que contiene información congruente con la original (no es necesariamente idéntico, porque el XMI es adaptado al contexto) que permite visualizar toda la información que fue ingresada originalmente por el usuario.

Se concluye que los objetivos propuestos originalmente se cumplen a cabalidad, si bien la solución en TCS no logró satisfacer una parte de ellos, si lo hizo la solución en Java, que logró resolver el problema de compatibilidad entre aplicaciones, y a su vez, perfecciona la solución entregada por ADAPTE al problema de optimización de procesos organizacionales.

Este trabajo está dedicado a mi abuelo Jaime.
Que en paz descanse.

Agradecimientos

Quisiera agradecer principalmente a la profesora Cecilia por sus consejos y enseñanzas durante este tiempo, y a Luis Silvestre quien me ayudó durante todo el trabajo. Además de mi familia y amigos por estar siempre conmigo, apoyándome, y por comprender el corto tiempo que tenía para ofrecerles durante estos meses que trabajé en la memoria.

Índice

Índice de figuras

2. Introducción

Para comenzar se dará una noción del proyecto ADAPTE y se expondrá la problemática a resolver dentro del proyecto. Luego se expondrán los objetivos que se propone alcanzar.

2.1. Motivación y Justificación

2.1.1. Contexto: Adaptable Domain and Process Technology Engineering

Adaptable Domain and Process Technology Engineering (ADAPTE) es un proyecto Fondef encargado de optimizar los procesos de desarrollo de software de las Pequeñas y Medianas Empresas (PyMEs) con tal de mejorar su funcionamiento y competitividad en el largo plazo. Para esto se enfoca en los procesos de desarrollo de software, los cuales permiten planificar la producción de un software de calidad, prediciéndolos en términos de tiempo o de recursos. Esto aumenta la productividad y desempeño de la empresa.

La mayor parte de las empresas de desarrollo de software en Chile son PyMEs. De ellas, muy pocas tienen un proceso de desarrollo organizacional definido, aunque muchas son conscientes de su relevancia. Estas empresas en general desarrollan proyectos cortos y de mediana complejidad, cuyo objetivo es esencialmente la construcción del software en sí, optimizando las actividades relativas a la gestión del proceso [3].

2.1.2. El problema que aborda ADAPTE

El problema al emplear un proceso organizacional es que éste no se aplica de la misma forma a cada proyecto, puesto que depende del tamaño del grupo de trabajo, sus habilidades, el conocimiento del dominio y el riesgo asociado, entre otros factores, por lo que debe adaptarse continuamente no solo a los cambios del personal, sino también a cada proyecto que se aborda (una persona puede poseer distinto grado de conocimiento para cada proyecto). Definir un proceso de desarrollo organizacional involucra un proyecto de mejora de procesos a nivel de organización.

Un proceso debe adaptarse al comenzar cada proyecto de desarrollo de software, por ser la base de la planificación. Esta adaptación debe ser rápida y eficiente.

En la práctica, las organizaciones generalmente enfrentan el problema de adaptación del proceso organizacional como un proceso específico del proyecto, siguiendo alguna de las siguientes estrategias:

1. Se aplica el proceso organizacional estrictamente como fue definido.

2. Se tiene un repositorio de método y un configurador de proceso.

3. Se tiene un conjunto de procesos de referencia para varios tipos de proyecto identificados en la organización, similar al conjunto de procesos plantilla sugeridos por Crystal Methodology [2].

4. Se aplica las dos estrategias anteriores en forma combinada.

Debido a la rigidez de estos procedimientos, resulta conveniente tener una nueva estrategia de adaptación que requiera menor esfuerzo humano, de tiempo y de costo.

En la actualidad, el proceso de adaptación e instanciación requiere de tres tipos de conocimiento: diseño de procesos, caracterización de proyectos, y caracterización del entorno de desarrollo, haciendo que la adaptación sea compleja y costosa, y que rara vez se realice en la práctica. Y por lo mismo las PyMEs que desarrollan proyectos de corta duración, realizan planificaciones muy ajustadas, no considerando tiempo para actividades como la adaptación del proceso a seguir y, en general, ven a la definición de un proceso organizacional principalmente como un medio para la certificación o evaluación. Un proceso definido en este contexto no genera beneficios tales como reducción de esfuerzo y costos que se podría esperar haciendo la adaptación [1].

2.1.3. Cómo ADAPTE aborda el problema

El proyecto aquí propuesto plantea una nueva estrategia de adaptación de procesos que implica trasladar parte del esfuerzo de la adaptación a la definición del proceso organizacional. Para ello, se propone aplicar un enfoque planificado de reuso de activos en la definición del proceso organizacional, de manera similar a como se planifica el reuso de software.

Existe un número finito pero potencialmente grande de procesos adaptados, por lo que se requiere de un mecanismo genérico bien definido con el fin de generar el proceso adecuado para cada proyecto. Para ello, la definición del proceso organizacional puede ser vista como una línea de producción de procesos de software. Esta estrategia logra resolver muchos de los problemas de los otros enfoques, como la proliferación de procesos adaptados (si se define un alcance), el enorme tiempo de adaptación (si hay un mecanismo de producción eficiente), y menos probabilidad de error (si hay un mecanismo de producción eficaz).

En ADAPTE se desarrolla una solución metodológica y tecnológica para producir familias de procesos de software y validarla en el contexto de al menos cinco PyMEs de software chilenas. Para ello, la solución involucra la definición de los siguientes aspectos:

i. Una forma para especificar la variabilidad en los procesos organizacionales.

ii. Una forma de caracterizar y representar el contexto de un proyecto.

iii. Una forma de catalogar las herramientas que forman parte de la plataforma tecnológica de la organización.

iv. Un método con soporte de herramientas que permita adaptar los procesos organizacionales a contextos específicos a un proyecto y que permita también definir la plataforma tecnológica requerida para su ejecución.

Esto le dará más valor a la definición del proceso organizacional, ya que tendrá incorporada la posibilidad de adaptación. Por tanto, en el contexto de estas organizaciones habrá una reducción en el esfuerzo y en el costo de los proyectos. Esto mejorará el camino hacia una industria de software nacional más competitiva [1].

2.1.4. El problema a resolver dentro de ADAPTE

La solución que presenta ADAPTE parte empleando una herramienta estándar para modelar los procesos de negocio esta se llama Eclipse Process Framework Composer (EPF Composer). Luego de compuesto el proceso de negocio, este se exporta en un documento XML (Extensible Markup Language) que utiliza el metamodelo UMA (Unified Method Architecture) el cual deberá ser transformado por un parser, para que sea compatible con el input del framework Eclipse Modeling Framework, donde se desarrolla el software encargado de realizar las transformaciones, en el lenguaje ATL, pues éste aplica todas las heurísticas que permiten automatizar el proceso de adaptación del proceso ante un contexto dado (que es el otro input del transformador). El input de este software es un archivo XMI (XML Metadata Interchange) con metamodelo SPEM (Software & System Process Engineering Metamodel). Para entender de mejor manera lo antes mencionado se puede observar la figura 1.

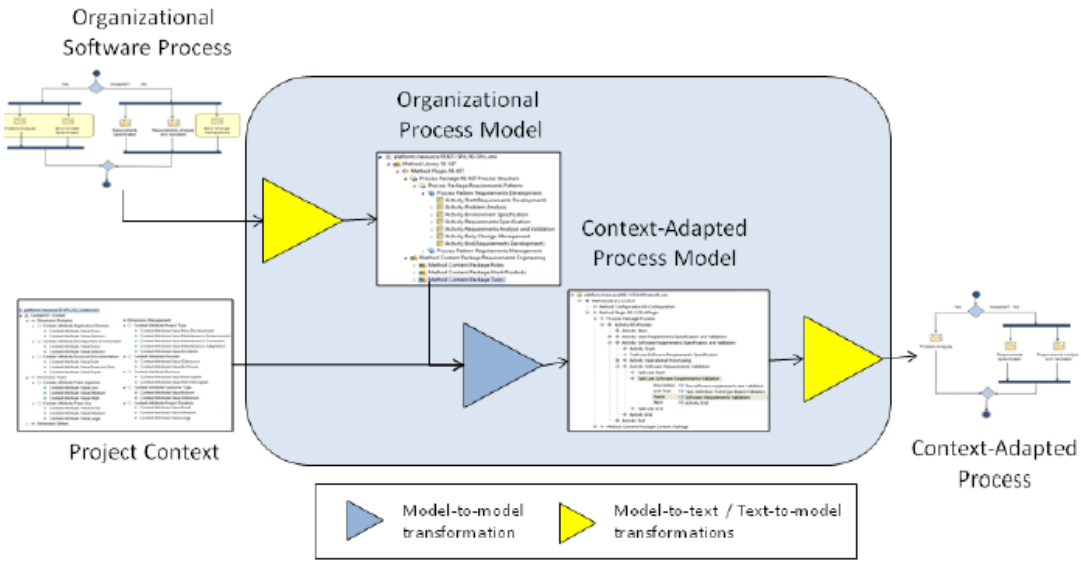


Figura 1: Esquema que muestra el proceso de adaptación que propone ADAPTE

Como al final del proceso se obtiene un archivo en formato XMI, el cual no es importable directamente a EPF Composer, se propone hacer otro parser que realice el camino inverso, de XMI con metamodelo SPEM a XML con metamodelo UMA de manera que el usuario de la aplicación no deje de usar EPF Composer para visualizar el proceso adaptado.
Por tanto lo que se requiere es:

1. Hacer un parser que, dado un documento XML instancia del metamodelo UMA, genere un documento XMI instancia del metamodelo SPEM (definido durante el proyecto). Es decir, realizar una conversión de instancias de metamodelos distintos.

2. Realizar el proceso de, una vez adaptado el proceso al contexto de proyecto particular en el documento XMI, volver al formato de documento XML importable por EPF Composer.

Por tanto es necesario realizar los dos parser (tanto el de conversión XML - XMI como el de XMI - XML) porque se manejará un documento XML con metamodelo UMA para realizar la visualización del proceso en EPF Composer, y a su vez es necesario tener un documento XMI con metamodelo SPEM para que sea manipulado por el software desarrollado en Eclipse Modeling Framework.

2.2. Objetivos

2.2.1. Objetivo Principal

El objetivo principal del trabajo es generar una aplicación que compatibilice dos de las herramientas principales del proyecto ADAPTE como lo son EPF Composer (que como se dijo, permite modelar procesos de negocio) y Eclipse Modeling Framework (donde se realiza la adaptación del proceso a un contexto dado) de manera tal que no haya impedimento en adaptar un proceso generado (por EPF Composer) en Eclipse Modeling Tools, o visualizar un proyecto ya adaptado (en Eclipse Modeling Tools) en EPF Composer.

2.2.2. Objetivo Secundario

Los objetivos secundarios son:

OBJ1. Construir un parser de XML a XMI.

OBJ2. Lograr que un proceso generado en EPF Composer sea parseado a XMI y abierto con Eclipse Modeling Framework.

OBJ3. Lograr que un proceso adaptado a un contexto con Eclipse Modeling Framework sea parseado a XML y visualizado en EPF Composer (que no haya problemas de compatibilidad con la aplicación original).

OBJ4. Comprender y manejar bien la aplicación EPF Composer que se considera un conocimiento valioso.

3. Conceptos Previos

Antes de describir la solución que se entrega para la problemática descrita, se investiga una serie de conceptos previos que entreguen el marco teórico necesario para construir una solución adecuada. Esta sección describe un resumen de los conceptos más importantes dentro de esta categoría. Cabe mencionar que en esta sección se describirán aquellos conceptos que son necesarios para comprender la problemática. Los conceptos que están involucrados al aprendizaje de la solución están al inicio de la sección correspondiente.

3.1. Eclipse Process Framework Composer

Eclipse Process Framework Composer (EPF Composer) es una plataforma asociada a los procesos de ingeniería, y destinada a los jefes de proyecto, quienes son los responsables de mantener e implementar procesos para el desarrollo de organizaciones o proyectos individuales. Para esto, los jefes de proyecto deben estar familiarizados con conceptos claves en el desarrollo de tareas, como la comprensión de requisitos, el análisis y diseño, implementación dado el diseño y generación de casos de prueba, cruzamiento de requisitos y módulos implementados, etc. Aunque muchas veces organizaciones pequeñas opten por tener equipos ágiles, es necesario, igualmente tener documentación para establecer prácticas en común. El punto anterior es importante, pero lo es más el hecho de saber aplicar dichas prácticas correctamente durante el ciclo de vida del proyecto.

EPF Composer por sobre todo, es una plataforma de aprendizaje que pretende, además de proveer las herramientas para generar un proceso de manera autónoma, presentar distintos métodos con tal de mejorar los procesos como templates, principios, buenas prácticas y material de entrenamiento. Dentro de estos se destaca el poseer una cantidad de métodos predefinidos que representan situaciones típicas de proyectos que pueden ser adaptados para un caso en particular [4].

3.2. Model Driven Engineering (MDE)

Es una metodología de desarrollo de software que se enfoca en crear y explotar modelos de dominio (esto es, representaciones abstractas del conocimiento y actividades que gobiernan una aplicación del dominio en particular) en vez de conceptos de cómputo (o de algoritmos). El principal objetivo de esto es maximizar la compatibilidad entre sistemas (con una problemática parecida), simplificando el proceso de diseño, promoviendo la comunicación entre individuos y equipos trabajando en el sistema [5]. Existen dos visiones sobre MDE: GPL y DSL.

3.2.1. General Purpose Languages (GPL)

Es un lenguaje de programación estándar universalmente conocido como C++, C# o Java que, como lo indica el nombre de la categoría, no está asociado a un dominio en particular. A partir

de este tipo de lenguajes se puede formar los DSL, o bien, resolver la problemática del modelo directamente.

3.2.2. Domain Specific Language (DSL)

Se usa un lenguaje pequeño pero bien enfocado en el modelo de cada sistema y coordina la relación entre los objetos del modelo.

3.3. Eclipse Modeling Framework

Eclipse Modeling Framework (EMF) es un framework de modelamiento que facilita la generación de código para la creación de herramientas y otras aplicaciones MDE. Provee herramientas y soporte en ejecución para producir un conjunto de clases Java para el modelo desde la especificación de un modelo descrita en XMI. Además cuenta con clases adapter que permiten la visión y edición basada en comandos del modelo y un editor para describir modelos y soporte en ejecución para los mismos. Posee un notificador de cambios, soporte a la persistencia con serialización XMI por defecto y una API muy completa para manipular objetos EMF de manera genérica [5].

Una de las principales funcionalidades que se puede hacer con este framework es la definición de un metamodelo el cual se formaliza en un archivo de extensión *.ecore*.

3.4. Modelos, Metamodelos y Metametamodelos

En el área de ingeniería de modelamiento, un **modelo** es considerado como una entidad de primera clase[1] y es definido de acuerdo a las reglas semánticas definidas en su **metamodelo**. A su vez, un metamodelo conforma con su **metametamodelo**. El metametamodelo generalmente se conforma a si mismo de acuerdo a su propia semántica. Metametamodelos existentes en la actualidad incluyen *MOF*, que ha sido definido por la OMG, y *Ecore* que fue introducido en el Eclipse Modeling Framework[17].

3.5. El metamodelo UMA

Unified Method Architecture (UMA) es un metamodelo que ha sido desarrollado como la unificación de diferentes lenguajes para métodos y procesos de ingeniería como la extensión SPEM de UML para ingenieros de procesos de software, los lenguajes usados para IBM Rational RUP v2003, Unified Prcess (UP), IBM Global Services Method, así como el IBM Rational Summit Ascendant. En este se proveen conceptos y capacidades de todos los modelos de origen unificándolos de manera consistente, pero aún así permitiendo usar cada uno de estos con sus características específicas. Esto habla de manera general de las capacidades de UMA [6].

[1]Entidad de Primera Clase: es una entidad construida en tiempo de ejecución, puede pasarse como parámetro, retornarse de una subrutina o ser asignada en una variable [16].

3.6. El metamodelo SPEM

SPEM 2.0 es usado para definir procesos de desarrollo de software y sus componentes. El alcance de SPEM está limitado (a propósito) a los mínimos elementos necesarios para definir un proceso de desarrollo de sistemas y de software, sin adherir funcionalidades especiales para un desarrollo particular de dominios o disciplinas (como por ejemplo la gestión de proyectos). El objetivo es acomodar un gran número de métodos y procesos de desarrollo de diferentes estilos, contextos culturales, niveles de formalismo, modelos de ciclo de vida y comunidades. Sin embargo, el foco principal de SPEM es el desarrollo de proyectos, no apunta a ser un lenguaje de modelamiento de procesos genéricos, y tampoco provee sus propios conceptos en el modelamiento de comportamiento. Más bien, SPEM permite que el implementador escoja el modelo de comportamiento genérico que considere que cubre mejor sus necesidades. También provee estructuras específicas para mejorar dicho modelo de comportamiento genérico como lo son las características para describir los procesos de desarrollo. En otras palabras, SPEM se enfoca en agregar las estructuras y la información adicional necesarias que se necesitan para modelar procesos con UML 2.0 Activities o para describir un proceso real de desarrollo con Business Process Model and Notation (BPMN)/Business Process Definition Metamodel (BPDM) [7]. Para el alcance de este trabajo, se utiliza una simplificación del modelo SPEM 2.0 que ha sido construido en el Departamento de Ciencias de la Computación de la Universidad de Chile.

4. Análisis de Requisitos

Como se mencionó en la introducción, se tiene un problema concreto, el cual es, realizar la conversión entre archivos con diferentes metamodelos, aunque con estructura similar (ambos poseen una estructura XML). Por tanto se procede a analizar los requisitos que se necesitará concretar previo al desarrollo e implementación de la solución.

Primero que todo y como es lógico, ambos archivos poseen un fin común, el cual es describir el proceso de desarrollo de empresas. Sin embargo lo hacen con fines distintos: el archivo salida de **EPF Composer** para representar gráficamente los diagramas necesarios y que el usuario de la aplicación pueda editar el proceso resultado, mientras que el archivo de entrada de **EMT** para optimizar el proceso descrito anteriormente dado un contexto. Por tanto, el contenido de ambos archivos debe ser semejante. Para comprobar esta semejanza es necesario identificar las entidades principales que se ven envueltas en este proceso de negocio:

1. **MethodElement**
   MethodElement es la generalización para todos los elementos contenidos. Define un conjunto común de atributos que debe contener todo elemento del modelo.

2. **MethodLibrary**

Elemento raíz de la aplicación. Es un contenedor físico para *MethodPlugins* y *MethodConfigurations*. Todos los *MethodElements* están contenidos en una librería.

3. **MethodPlugin**
   Es un *MethodElement* que representa un contenedor físico para *MethodPackages*, define un nivel de granularidad para la organización y modularización de procesos y contenidos. Puede extender a muchos *MethodPlugins* (o ser extendido por estos) o ser stand-alone. Conceptualmente representa la unidad para configuración, modularización, extensión, empaquetamiento y desarrollo de contenidos y procesos.

4. **MethodConfiguration**
   Es una colección de modelos y paquetes seleccionados.

5. **MethodPackage**
   Es una clase abstracta que empaqueta *MethodElements*. Cada elemento debe estar en al menos una de las especializaciones concretas de esta clase. Un *MethodPackage* define propiedades en común para todas sus especializaciones.

   a) **ContentPackage**
   Es un *MethodPackage* especial que contiene únicamente *ContentElements*.

   b) **ProcessPackage**
   Es un *MethodPackage* especial que contiene únicamente *ProcessElements*. Existe una clase que representa una especialización del *ProcessPackage*: el **ProcessComponent**, que presenta una o más interfaces con inputs y outputs específicos de la componente. Esto se traduce en que *ProcessComponent* sigue los principios de encapsulación.

6. **ContentElement**
   Es un elemento que representa una generalización de todos los elementos que pueden considerarse como contenido administrable. Las especializaciones de esta clase abstracta son: **Categories**, **Roles**, **Tasks** y **WorkProductDefinitions**.

7. **ProcessElement**
   Es un elemento que representa una generalización de todos los elementos definidos en el *ProcessPackage*. Un *ProcessElement* representa un elemento específico del proceso. La separación entre elementos de proceso y de contenido se hace para distinguir claramente entre elementos que son netamente contenido, y los que son procesos.

8. **WorkBreakdownElement**
   Un *WorkBreakdownElement* es una especialización de un *ProcessElement*, es una representación para cualquier tipo de *MethodElement* que forma parte de una estructura de desglose. Define un conjunto de propiedades para sus especializaciones. Sus especializaciones son: *Activity*, **TaskDescriptor**, **RoleDescriptor** y **WorkProductDescriptor**.

9. **Activity**

   Es una especialización de un *WorkBreakdownElement* que soporta el agrupamiento anidado y lógico de *WorkBreakdownElements* formando estructuras de desglose. Aunque *Activity* es una clase concreta, posee otras que heredan de ella como: **Process**, **Phase**, **Iteration**, **DeliveryProcess** y **CapabilityPattern**.

   a) **Phase**

   Es una *Activity* especial que agrega valores predefinidos para sus instancias para los atributos `prefix = 'Phase'` y `isRepeatable = 'False'` y representa un periodo significativo de un proyecto.

   b) **Iteration**

   Es una *Activity* especial que agrega valores predefinidos para sus instancias para los atributos `prefix = 'Iteration'` y `isRepeatable = 'True'` y representa una actividad que se realiza comúnmente en la ejecución de un proyecto.

   c) **Process**

   Es una clase abstracta que describe la estructura para un tipo particular de proyectos de desarrollo. Para ejecutar dichos proyectos, un proceso debe ser instanciado y adaptado para la situación específica.

   d) **DeliveryProcess**

   Describe por completo el ciclo de vida del proyecto y debe usarse de referencia para ejecutar proyectos de características similares.

   e) **CapabilityPattern**

   Es un proceso que describe un grupo reutilizable de elementos, que indican como se hace el trabajo en áreas de proceso común.

Estas entidades se encuentran presentes en ambos metamodelos UMA y SPEM, y representan la columna vertebral del problema en sí, pero existen muchos otros elementos que no se presentan en ambos metamodelos. El metamodelo UMA consta de aproximadamente de 123 clases, mientras que SPEM consta sólo de 31 clases, por lo tanto se puede decir que se está perdiendo información al realizar la conversión de un archivo UMA a SPEM. Si bien esto es indiscutible, no es tan grave como pueda pensarse en un principio. El **EPF Composer** posee muchas otras funcionalidades asociadas a un proceso de desarrollo, que es lo que nos compete de la aplicación, como por ejemplo, asociar informes de desempeño, adjuntar documentos en las distintas etapas, y hacer diagramas visualizables gráficamente, entre otras. Esta funcionalidad si bien es importante para el usuario de la aplicación, no se necesita para realizar la optimización del proceso asociado al contexto (objetivo de ADAPTE), por tanto, no se pierden funcionalidades estructurales, pero sí se pierde información de valor para el usuario (que visualizará después el proceso optimizado). Por tanto se considera aceptable realizar el proceso considerando solo lo estructuralmente necesario para la optimización con ADAPTE, y luego, observar si es posible incluir los elementos que en un principio se descartaron. Para esto

es fundamental observar experimentalmente si se permite observar la información en el framework correspondiente, debido a que, de no ser posible obtener una solución adecuada, será necesario recuperar la información restante.

4.1. La conversión propuesta

Se propone realizar dos soluciones para el problema de convertir un archivo de un formato a otro: la primera será construida en el lenguaje **Java** (ver la sección 5) debido a la gran cantidad de librerías que existen para manejar los nodos de documentos XML, y la segunda solución es construida en **TCS** (ver la sección 6) que es un lenguaje para definir sintaxis textuales concretas.

4.2. Experimentos a realizar

Para comprobar la correctitud de las soluciones se usará tres plugins que representan un proceso de desarrollo real. Los plugins corresponden a tres empresas que están asociadas al desarrollo de ADAPTE: el plugin **Rhiscom**, **DTS** e **Imagen**, los tres lógicamente fueron elaborados en **EPF Composer**. Se convertirá este archivo para ser leído con **EMT** y luego se procesará este archivo para convertirse nuevamente al formato de lectura de **EPF Composer**.

4.3. Correspondencia entre las clases

En esta sección se describirá el mapeo entre las clases de SPEM y UMA, pero para facilitar la notación solo se numerarán las clases que están involucradas en el mapeo.

La figura 2 contiene el detalle de la conversión entre clases de ambos metamodelos. Como comentario adicional se agrega que **ContentCategory** de SPEM (clase concreta) pasa a ser **CustomCategory** porque este representa la generalización de las especialidades de la clase **ContentCategory** de UMA que es abstracta. El otro punto importante está en **ContentPackage** y **ContentCategoryPackage** de UMA, que se distinguen en que el segundo posee categorías; el objeto **MethodContentPackage** posee categorías también, así que diferenciando en si tiene categorías o no, se convierte en uno u otro objeto.

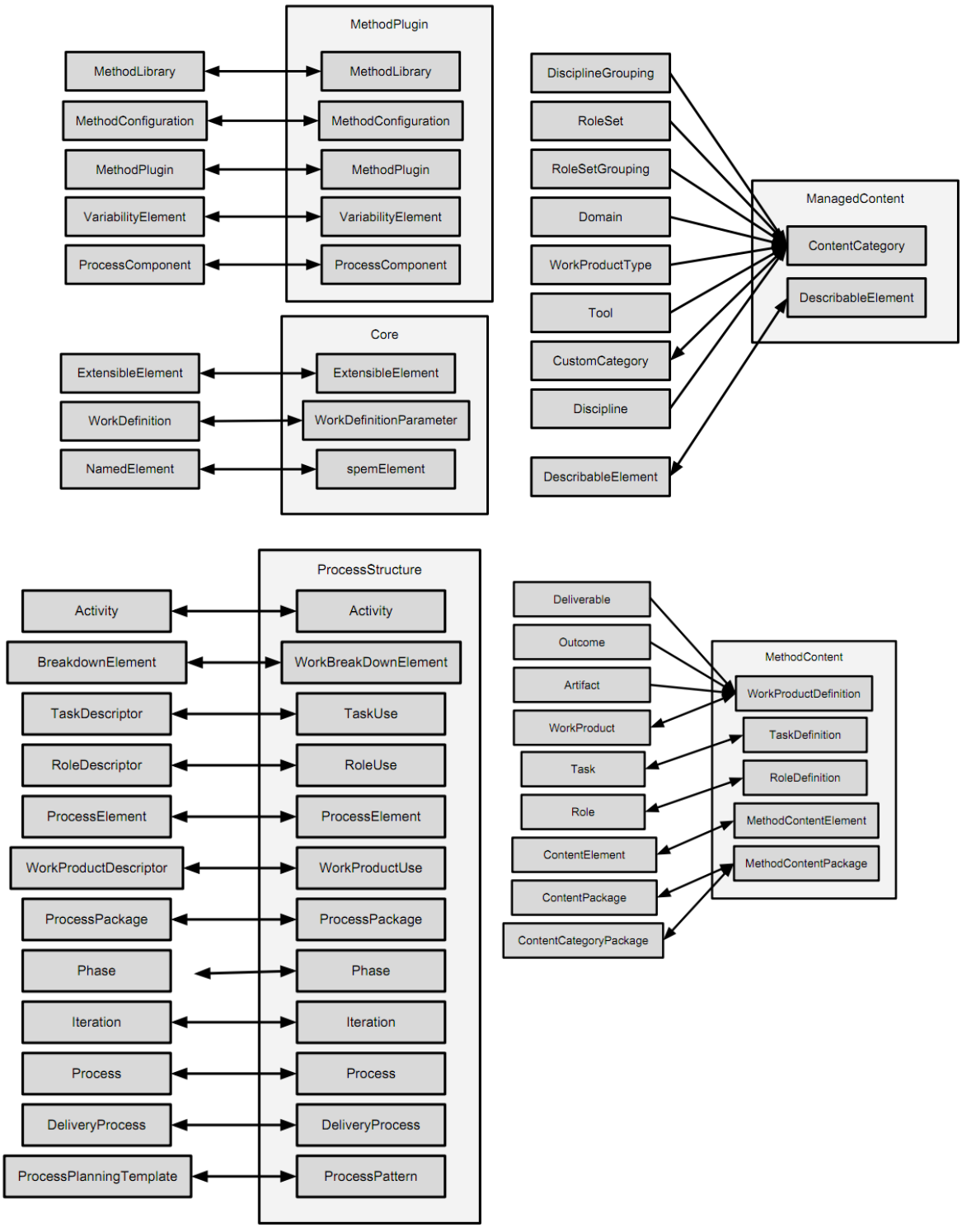


Figura 2: Mapeo de clases: UMA a la izquierda, SPEM a la derecha

5. Solución 1: Java

Debido a que ambos modelos están especificados detalladamente en los archivos *.ecore*, se realiza una primera solución a la problemática de relacionar las clases de un metamodelo con otro usando

la programación orientada a objetos de Java.

Antes de comenzar a describir la solución, existe una serie de conceptos relacionados que serán expuestos a continuación para comprender mejor la solución y la manera de proceder.

5.1. Conceptos Previos

5.1.1. Java

Java es un lenguaje de programación orientado a objetos basado en clases, concurrente y pensado para propósitos generales. Está diseñado para ser lo suficientemente simple para que muchos programadores puedan lograr fluidez en el lenguaje y está relacionado a `C` y `C++` en su sintaxis, pero está organizado de manera bastante diferente. Hay muchos aspectos de dichos lenguajes que se omiten y un par de ideas de otros lenguajes que se han incorporado. La intención es que sea un lenguaje de producción y no de investigación.

Algunas propiedades del lenguaje son que es fuertemente y estáticamente tipado, lo que permite distinguir entre errores en tiempo de compilación, y los que ocurren en ejecución. Es un lenguaje de alto nivel, incluye manejo automático del almacenamiento, usando un recolector de basura. No se incluye ningún constructor inseguro en el sentido de permitir accesos a arreglos sin chequeo de indexación.[8]

5.1.2. Object Oriented Programming

Se llama Orientación a Objetos a la metodología de programación basada en objetos, en lugar de funciones y procedimientos. Un objeto en esta metodología es una instancia de una clase y posee una estructura similar a otros objetos en la clase, pero se le asignan características individuales. La orientación a objetos permite a los desarrolladores mejorar el dominio de problemas, y así, resolver una mayor variedad de estos, dándole una nueva perspectiva a la computación [13].
La programación orientada a objetos provee soporte para los siguientes conceptos:

- Objectos y Clases.
- Herencia.
- Polimorfismo y enlazamiento dinámico.

5.2. Metodología

A continuación se enumerará la metodología asociada para lograr un parser de metamodelos en Java.

- Se transforma la clase del metamodelo en un objeto Java.
- Se agrega un método a cada objeto que permita ingresar información al objeto mediante el nodo XML (expresado en un objeto org.w3c.dom.Element) correspondiente a la clase.
- Se agrega un método a cada objeto que permita obtener un nodo XML (expresado en un objeto org.w3c.dom.Element) con los atributos y nodos hijos del objeto.

Realizando los pasos descritos anteriormente, tanto para el metamodelo UMA como para SPEM, permitirá reducir el problema de convertir un modelo a otro, a comunicarse entre los objetos Java correspondientes. Esto debido a que cada objeto Java tiene métodos necesarios para generarse a partir de un nodo del documento XML (o XMI) correspondiente, y a su vez entregar un documento XML (o XMI) a partir de la información que posee.

Un primer paso es hacer que el elemento UMA pueda tomar un documento XML (con metamodelo UMA), inicialice sus atributos y los complete con los valores correspondientes, y luego pueda formar de nuevo el mismo documento XML a partir de la información que se alojó. De esta manera se chequeará que funciona correctamente. El paso siguiente es hacer lo mismo con el elemento SPEM, para chequear que este también emite un documento correcto. Finalmente se crean las clases que permiten transformar de un metamodelo a otro, siguiendo la lógica del párrafo anterior, demostrando que se genera un documento válido para ambas transformaciones.

5.3. Elements y Documents

Durante el desarrollo de esta solución, hubo dos objetos principales que sirvieron para facilitar la comprensión y el manejo de los documentos XML (o XMI) a convertir. Estos serán descritos brevemente junto con las principales facultades que fueron empleadas a lo largo del desarrollo de la aplicación.

5.3.1. DocumentBuilder (javax.xml.parsers.DocumentBuilder)

Este elemento parsea (internamente) el archivo XML y lo transforma en un objeto `Document`. Para crear un objeto `DocumentBuilder`, es necesario tener un `DocumentBuilderFactory`, que crea el objeto con `DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder()`. Para crear un nuevo objeto `Document`, a partir de un documento (puede ser su ruta como `String`, o un objeto `java.io.File`), se usa `Document document = documentBuilder.parse(‘‘file.xml’’)`.

5.3.2. Document (org.w3c.dom.Document)

Esta clase representa el documento XML (o XMI) en sí. Se utiliza principalmente para obtener el objeto Element, mediante la llamada `Element element = document.getDocumentElement()`, que permite crear un `element` con el primer hijo del documento. También se necesita el documento

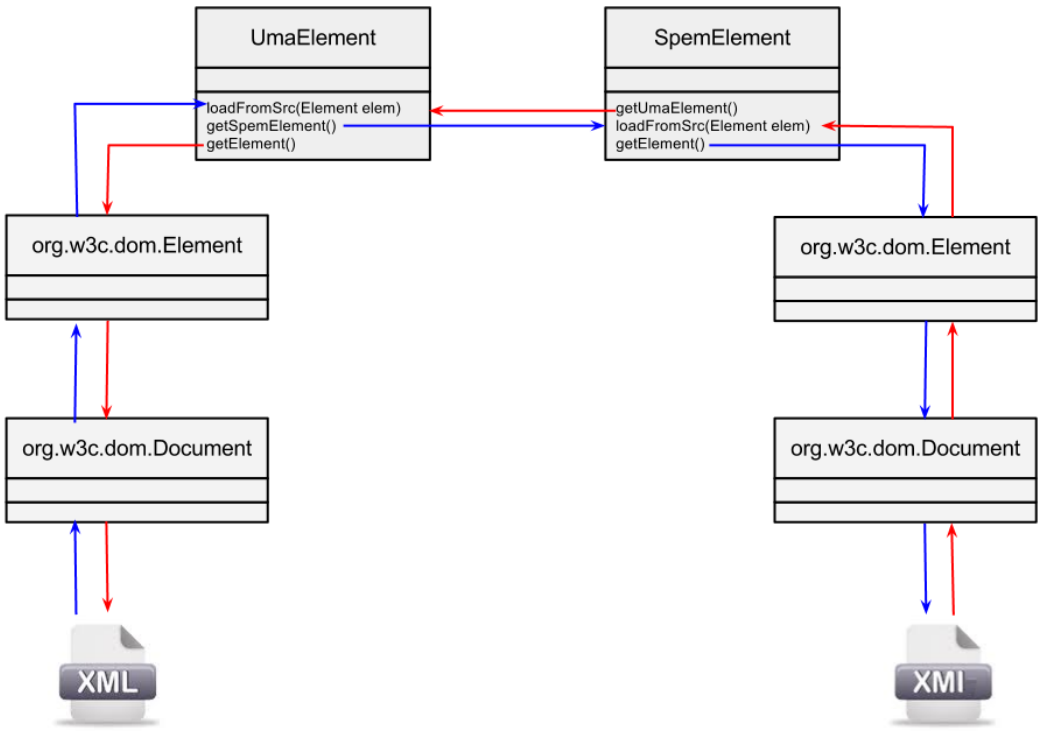


Figura 3: Descripción del diseño de la solución

para obtener un nuevo elemento, que no necesariamente será una nueva raíz del documento, ya que también puede ser un hijo de otro elemento. Para esto es necesario entregarle un nombre al nuevo elemento (nodo); esto se hace con `Element anElement = document.createElement(nodeName)`.

5.3.3. Element (org.w3c.dom.Element)

`Element` representa un nodo dentro del documento XML (o XMI). Extiende de la clase `Node` (`org.w3c.dom.Node`) y contiene un nombre (`getNodeName()`), un valor (`getNodeValue()`), y contenido (`getTextContent()`).

Además se puede acceder a los atributos del nodo en un objeto `NamedNodeMap` (`org.w3c.dom.NamedNodeMap`) mediante el método `NamedNodeMap attributes = element.getAttributes()`. El objeto `NodeList` (`org.w3c.dom.NodeList`) permite obtener todos los elementos hijos del nodo mediante `NodeList listOfChils = element.getChildNodes()`. También se puede obtener un atributo por su nombre con `String name = element.getAttribute(‘‘name’’)`, y todos los elementos hijos de la jerarquía por su tag name, almacenándolos en el ya mencionado objeto `NodeList`, con `NodeList tagelems = element.getElementsByTagName(‘‘TagName’’)`.

Finalmente se puede asignar un atributo mediante `element.setAttribute(‘‘name’’, name)`,

y agregar un nuevo elemento hijo a este nodo con `element.appendChild(childElement)`.

5.3.4. Transformer (javax.xml.transform.Transformer)

Esta clase sirve a la hora de generar el output, es decir, transformar nuevamente, el `Document` en un documento XML (o XMI). Para tener un `transformer` es necesario primero inicializar un `TransformerFactory` (`javax.xml.transform.TransformerFactory`). Es en este objeto donde se agregan atributos como el encoding y la indentación, y para la transformación final es necesario tener un `Source` (`javax.xml.transform.dom.Source`), el cual para inicializarse necesita un `Element`, `Source src = new DOMSource(element)`, y un `Result` (javax.xml.transform.Result), que para inicializarse necesita un `Writer` (`java.io.Writer`), `Result rst = new StreamResult(writer)`. Finalmente la transformación se realiza con `transformer.transform(src, rst)`.

5.4. Clases utilitarias

Durante esta sección se describirá brevemente las clases que sirvieron para facilitar el manejo de `Element`, `Document` y `String`, o para lograr una solución más completa.

5.4.1. ElementUtils

Esta clase permite manejar de mejor manera los problemas relacionados a un objeto `Element`, en especial para hacer más limpio el código de los objetos `UmaElement` y `SpemElement`.

```
public static void addSingleChild(Element element, Document document,
       String nodeName, String value)
```

En estos documentos en repetidas ocasiones aparece un nodo que solamente tiene contenido, como el siguiente: `<Element>Content</Element>`, en este método se agrega un hijo de este tipo al `Element element`, para esto, es necesario tener el `document` ya que se creará un nuevo `Element`, y tendrá el nombre `nodeName` y el contenido `value`.

```
public static void addNewChilds(Element element, Document document,
       String nodeName, List<String> values)
```

Método similar al descrito anteriormente pero que crea `N` valores, los que están almacenados en la lista `values` al `Element element`.

```
public static String loadFromSingleChild(Element element, String nodeName)
```

Este método obtiene el valor de un nodo que solamente tiene contenido (como el descrito en `addSingleChild`) que es hijo del `Element element` y tiene el nombre `nodeName` y lo retorna como `String`.

```
public static void loadFromChilds(Element element, String nodeName,
       List<String> listToLoad)
```

Una versión equivalente del anterior pero aplicable para cuando el `Element element` contiene muchos hijos con el nombre `nodeName`, estos valores son almacenados en la lista `listToLoad`.

5.4.2. ReferenceObject

`ReferenceObject` es un objeto fundamental para la aplicación, puesto que se lo diseñó para almacenar ambos objetos `Document document` necesarios para generar los objetos `UmaElement` y `SpemElement`. De esta manera, cuando se crea un nuevo elemento, se utiliza el documento correspondiente. También, a modo de debug este objeto guarda un `Map<String, UmaElement>` que guarda los objetos UMA que se van generando, y otro `Map<String, SpemElement>` que hace lo mismo con los objetos SPEM: la `key` de ambos map es el id del objeto. Esto último permite realizar dos operaciones que, a posteriori serán importantes en el desarrollo de la aplicación: hacer el chequeo de referencias (que un elemento, luego de ser convertido, no referencie elementos que debido a la conversión ya no existen en el documento) y la carga de elementos en UMA (cuando un elemento no fue convertido de UMA a SPEM se quiere recuperar al realizar la conversión de SPEM a UMA).

5.4.3. StringUtils

Los métodos empleados en esta sección se utilizan principalmente para chequear la existencia de referencias (que son cadenas de caracteres) en un objeto repositorio de referencias (`ReferenceObject`).

```
public static String checkUmaSingleReference(ReferenceObject reference,
       String str)
```

Esta función toma un identificador `str` y chequea si está en las referencias del objeto `reference`. Para esto, primero que todo obtiene el map de referencias de objetos UMA asociados a la conversión, y comprueba si este contiene un objeto con `key` igual al identificador; de ser así retorna el mismo identificador, de lo contrario retorna una cadena vacía.

```
public static String checkSpemSingleReference(ReferenceObject reference,
       String str)
```

Función equivalente a la anterior con el map de referencias de SPEM.

```
public static List<String> checkUmaReferences(ReferenceObject reference,
       List<String> lst)
```

Esta función itera sobre cada elemento identificador de la lista `lst` y para cada uno chequea si es que el map de referencias de objetos UMA contiene una `key` con el identificador. De ser así, agrega el `String` a una nueva lista, de lo contrario, continua con la iteración. En caso de no existir ninguna referencia se retorna la lista inicializada.

```
public static List<String> checkSpemReferences(ReferenceObject reference,
        List<String> lst)
```

Método equivalente al anterior con el map de referencias de SPEM.

5.4.4. IdGenerator

Se utiliza esta clase para generar un `String` aleatorio que sirva como id; en caso que un elemento no posea id (esto en la práctica no ocurre nunca), el constructor de `UmaElement` y `SpemElement` genera un id para el elemento que posteriormente puede ser modificado (con el setter).

5.5. El objeto UmaElement

El metamodelo UMA consta de aproximadamente 120 clases por lo que hablar de cada una de ellas sería tedioso. Sin embargo, todas estas comparten métodos en común y además están organizadas jerárquicamente, es decir, están organizadas en relaciones padre-hijo entre sí. Lo que diferencia una clase de otra son los atributos que contienen. Hay que agregar que, en una jerarquía de clases, la clase hija hereda los atributos y métodos de la clase padre y, en caso de usar un método de forma distinta, éste es sobrescrito con respecto al de la clase padre (en Java esto se hace con la anotación `Override`).

El primer ancestro de la jerarquía de clases (en el sentido que todos los demás elementos son descendientes de esta clase) descrita es la clase abstracta[2] `UmaElement` (el equivalente en SPEM es `SpemElement`), y contiene un atributo identificador del elemento (`id`), el nombre del nodo (`nodeName`) y el objeto `ReferenceObject reference` descrito en la sección 5.4.2 que permite obtener cualquier elemento a partir de su id y además guarda la referencia a los objetos `Document`.

Los dos primeros atributos que contiene la clase (`id` y `nodeName`) están ahí porque se espera que todo elemento definido dentro del documento contenga, al menos, la siguiente información:

```
<nodeName id=‘‘anID’’ ... />
```

En Java el objeto que contiene esta información se representa de esta forma:

```
public class ObjectX {
  private String id;
  private String nodeName;
  private ReferenceObject reference;
  ...
}
```

[2]Una clase abstracta no puede ser instanciada.

Como se dijo, los elementos descendientes de esta clase irán agregando progresivamente atributos, donde guardan la información relativa al proceso que se describe en EPF Composer. Es así que los nodos hijos deberían ser de la forma:

```
<nodeName id=‘‘anID’’ attr1=‘‘val1’’ attr2=‘‘val2’’ ... />
```

Los atributos de un nodo en Java se representan como atributos de tipo `String` en la clase.

```
public class ObjectX {
  ...
  private String attr1;
  private String attr2;
  ...
}
```

A su vez, un elemento puede contener otros elementos. En XML esta relación tiene la forma siguiente:

```
<nodeName id=‘‘anID’’ attr1=‘‘val1’’ attr2=‘‘val2’’ ... >
   <childNodeName id=‘‘otID’’ attr3=‘‘val3’’ ... />
 <anotherChildNodeName id=‘‘antID1’’ attr4=‘‘antVal1’’ ... />
 ...
 <anotherChildNodeName id=‘‘antIDN’’ attr4=‘‘antValN’’ ... />
</nodeName>
```

En este ejemplo el elemento contiene un nodo hijo con nombre “child” y `N` nodos hijos de nombre “anotherChild”. Como se espera, esta definición es recurrente, es decir, el hijo a la vez puede tener más hijos contenidos. La relación que en XML se expresa como un elemento contenido físicamente en otro, en Java se expresa como una relación de referencia de objetos, se vería de la manera siguiente:

```
public class ObjectX {
  ...
  private ObjectY child;
  private List<ObjectZ> anotherChild;
  ...
}
```

Otro tipo de relación que se maneja en UMA (en esto se distingue de SPEM) es cuando un elemento contiene sólo el id de referencia de muchos otros elementos dentro del documento. Esto en XML se observa de la manera siguiente:

```
<nodeName id=‘‘anID’’ attr1=‘‘val1’’ attr2=‘‘val2’’ ... >
   <aRef>referenceID</aRef>
 <anotherRef>anotherReferenceID1</anotherRef>
 ...
 <anotherRef>anotherReferenceIDN</anotherRef>
</nodeName>
```

En este caso contiene una referencia al “aRef” de id “referenceID” y N referencias a “anotherRef” con sus ids correspondientes. Esto en Java se maneja como contener un `String` o una lista de estos, y se observa a continuación:

```
public class ObjectX {
  ...
  private String aRefID;
  private List<String> anotherRefsID;
  ...
}
```

En el documento sólo ocurren estos tres casos, por lo que, ahora que se explicó cómo se maneja cada uno de ellos, se procede a explicar los métodos que contiene cada elemento de UMA (tanto `UmaElement` como sus descendientes). Dos de estos, `getElement()` y `setElement(result)` son sobrescritos cada vez que se incorpore información nueva al documento, es decir, cuando una clase tiene nuevos atributos que agregar. Los otros dos métodos simplemente llaman a los dos primeros y son definidos solamente en `UmaElement`.

`protected void setAttributes(Element result)`

Este método permite asignar los atributos que posee el objeto en el `Element` correspondiente. Es sobrescrito por todas las clases que posean nuevos atributos que agregar, siempre llamando al comienzo de este método a `super.setAttributes(result)`. Esto permite que los atributos heredados por la clase correspondiente sean asociados correctamente al `Element result`. Se define este método como `protected`[3] debido a que se espera que sea utilizado sólo por el método (descrito más adelante) `getElement()`, sin embargo, toda la lógica asociada se encuentra en este método.

Existen tres tipos de elementos que se le puede asociar a un `Element`, según los atributos que tenga la clase:

1. *Atributo*
   El atributo de un nodo XML es representado directamente por un atributo `String` de la clase; es así que si se tiene el atributo `String name`, se utiliza `result.setAttribute(‘‘name’’, name)`.

[3] `protected` indica que el método será visible únicamente por otros métodos en la clase y sus herederos

2. *Nodo hijo, con contenido*
   El siguiente elemento es un ejemplo de nodo hijo sin atributos pero con valores, siendo el contenido del nodo, un id de otro nodo del documento. En este caso se utiliza el método `ElementUtils.addNewChild` descrito anteriormente, en caso de tratarse de una lista de valores, o `ElementUtils.addSingleChild`, al tratarse de un solo valor.

3. *Nodo hijo*
   Al haber un nodo hijo (`child`) quiere decir que la clase misma tiene un atributo que es otro objeto del metamodelo o bien una lista de estos; en ambos casos se traspasa la responsabilidad de agregar el nuevo elemento al resultado con `result.appendChild(child.getElement())`.

`protected void getAttributes(Element src)`

Este método permite obtener los atributos desde un `Element src`. Al igual que el método descrito anteriormente, este método es sobrescrito por todas las clases que heredan del objeto y que posean algún atributo que no esté incorporado. Estos parten, al igual que con el método anterior, con una llamada a `super.getAttributes(src)`, y se define como `protected` al ser utilizado por el método `loadFromSrc(src)`, sin embargo, toda la lógica la realiza este método.

Se tiene de igual manera, tres posibles valores a obtener desde el `Element src`.

1. *Atributo*
   En este caso, el valor se almacenará en un atributo `String` de la clase, para esto sólo es necesario conocer el nombre del atributo en el elemento y obtenerlo con `src.getAttribute(‘‘name’’)`.

2. *Nodo hijo, con contenido*
   Se usa `ElementUtils.loadFromChilds` para cargar una lista de atributos y `ElementUtils.loadFromSingleChild` para cargar un solo atributo.

3. *Nodo hijo*
   En este caso se utiliza el método `ElementUtils.getChildsByTagName` para cargar una lista de un objeto del modelo, o `ElementUtils.getSingleChildByTagName` para cargar un elemento.

Además, sólo la clase principal obtiene el nombre del nodo, con `nodeName = src.getNodeName()`.

`public Element getElement()`

Este método necesita del elemento `document` para generar un nuevo elemento (`src`) con `Element src = document.createElement(nodeName)` luego, utiliza `setAttributes(result)` para cargarlo con los atributos de la clase, y sus hijos, y finalmente retorna el elemento.

`public void loadFromSrc(Element src)`

Este método simplemente llama a `getAttributes(src)`.

```
public SpemElement getSpemElement()
```

Este método permite obtener un `SpemElement`. La lógica involucrada depende mucho de la clase que se esté convirtiendo puesto que como se explica en las secciones anteriores, no es completa la correspondencia de clases entre los modelos, y si la hay, tampoco hay correspondencia entre los atributos de estas, así que no es directa la relación, pero lo que se hace en este método es crear el otro elemento, y asignarle los atributos (mediante los setters de la clase) que correspondan.

5.5.1. Clases Abstractas

Existen clases abstractas dentro del metamodelo UMA que a la vez son referenciadas como atributos dentro de otras clases, estas son las clases `BreakdownElement`, `ProcessElement` y `ContentElement`, todas estas poseen un atributo que permite distinguir a que clase hijo pertenecen, este es `‘‘xsi:type’’`, por tanto, estas clases poseen un método estático que permite obtener el hijo a partir del `xsiType`, para el `ContentElement`, el método es el siguiente:

```
public static UmaContentElement getSubtypeResult
           (ReferenceObject reference, Element element)
```

Además se tiene un método que permite saber si existe un hijo con el `xsiType` dado, `public static boolean containsSubclassType(String type)`, esto es útil porque estas clases tienen más de una generación en la jerarquía de clases, entonces `getSubtypeResult` puede preguntar si es que algún hijo contiene en su rama genealógica un heredero con el `xsiType` dado. Lo mismo cuando se hace la conversión a un `SpemElement`, en tal caso se tiene otro método estático que realiza la misma labor, pero entregando el `SpemElement` correspondiente al `xsiType`, para el `ContentElement` sería

```
public static SpemMethodContentElement getSpemElementFromAbstract
           (UmaContentElement element, String nodeName)
```

5.6. El objeto SpemElement

La lógica involucrada en el diseño de este elemento es equivalente a la usada con `UmaElement`. Con respecto a las clases abstractas, en SPEM se habla de `WorkBreakDownElement`, `ProcessElement` y `MethodContentElement` que, al igual que con las clases mencionadas en UMA, tienen un atributo `‘‘xsi:type’’` que permite diferenciar a que subclase pertenecen.

5.7. Conversión SPEM a UMA: Cargar datos de archivo original

Como se ha mencionado ya varias veces, el metamodelo SPEM contiene mucha menos información de la que contiene UMA, por lo que no se puede esperar que no se pierda información al pasar

a SPEM. Ahora bien, para realizar el proceso inverso de SPEM a UMA se hace necesario poder recuperar dicha información, debido a que primero que todo, se hace imprescindible para el proyecto mismo que el documento resultado de la adaptación, pueda visualizarse en el formato original, en la herramienta original.

Viendo el sistema desde un punto de vista lógico, se haría tedioso para un usuario final del producto el tener que redefinir toda aquella información (que es información relevante para el usuario final de la aplicación, y representa la mayoría de la información contenida en el proceso) que no es usada por la aplicación en ATL luego de ejecutada la adaptación.

Es por esto que se tomó como un punto sumamente importante el realizar la conversión inversa de SPEM a UMA recuperando toda información que no fue utilizada en el proceso. Para esto se tuvieron que agregar los siguientes métodos a la aplicación original:

```
public void loadFromOriginal(UmaElement element)
```

Se agregó a cada elemento en el modelo UMA que tuviese información no contenida en su semejante en SPEM. Permite obtener dicha información a partir del elemento original, el cual es buscado en el `ReferenceObject`. En el caso de contener nodos enteros que no son utilizados en SPEM, estos se pasan al elemento y luego se revisarán las referencias del mismo.

```
public void checkReferences(ReferenceObject reference)
```

Este método es el encargado de revisar las referencias del elemento. Se especificó en el diseño que al momento de crear un elemento del modelo (tanto en SPEM como en UMA) se necesita un `ReferenceObject` asociado al mismo, debido a que este objeto contiene un Map que referencia cada elemento con su id. En el paso anterior que carga datos del elemento original, muchas veces se accesará información que viene mal referenciada, apuntando al `ReferenceObject` anterior (el del `MethodLibrary` original) por lo que se ingresan estos elementos y se cambia la referencia al `ReferenceObject` actual. Este paso es importante porque el `ReferenceObject` guarda los `Document` que guardan la información del archivo en que se generará la conversión; de estar mal referenciado un objeto, la aplicación no podrá realizar la conversión.

```
public void references()
```

Método similar al `check` (definido en la sección 5.9.2) pero que en vez de chequear si los elementos pertenecen al map de referencias, ingresa el elemento a dicho map, en caso de no encontrarlo. Es decir, este método va recorriendo la jerarquía de clases cambiando las referencias (con el método anterior) con tal de que sea consistente.

5.8. Parser

Parser es la clase que realiza las conversiones de datos, está implementada en forma de Singleton de manera que solo puede haber una instancia. Utiliza todo lo descrito anteriormente para que la conversión sea sencilla. Los métodos utilizados son los siguientes:

```
public SpemMethodLibrary parseUmaToSpem(UmaMethodLibrary umaElement)
```

Este método retorna un `MethodLibrary` de SPEM a partir del mismo en UMA, recordar que el elemento principal de ambos metamodelos es el mencionado.

```
private UmaMethodLibrary simpleParseSpemToUma(SpemMethodLibrary spemElement)
```

Realiza la conversión inversa, se le agrega el "simple" debido a que no utiliza el archivo UMA original para recuperar información no contenida en SPEM.

```
public UmaMethodLibrary parseSpemToUma(SpemMethodLibrary spemElement,
       UmaMethodLibrary original)
```

Método idéntico al anterior tomando en cuenta el `MethodLibrary` original en UMA. Esto permite recuperar información que no soporta el metamodelo SPEM y que es importante para definir el proceso, pero no se puede optimizar.

Además de estos métodos existen otros que toman directamente los archivos de entrada y de salida correspondientes a cada `MethodLibrary`, lo que facilita la llamada desde la clase principal.

5.9. Consideraciones durante el desarrollo

Durante el desarrollo de la aplicación se encontró un par de incidencias que no se tomaron en cuenta en el diseño original y que no permitían obtener una solución correcta. En esta sección se enumerarán estas dificultades y se explicará cómo se resolvió.

5.9.1. Herencia múltiple

En el lenguaje Java no se permite que se extienda a más de una clase, sin embargo, en la especificación de ambos metamodelos, se observa como algo común la herencia múltiple, de hecho la clase `Activity` tanto en SPEM como en UMA hereda de al menos tres clases. Dentro de la solución que se desarrolla en Java se considera que, tanto para obtener los atributos de una clase desde un documento de entrada, como para agregárselos al atributo del documento de salida, se hace de manera que se llame primero el método `super` correspondiente.

En este caso, la herencia se usa solamente para heredar atributos de las clases padre, lo cual supone una facilidad para afrontar el problema. Luego, la manera en que se soluciona la problemática

de la herencia es copiando los atributos de la jerarquía (padres, abuelos, etc.) que no es cubierta por la clase de la cual se extiende. Para esto es necesario ser consistente en todo momento y así evitar errores en el resultado final.

5.9.2. Sobre las referencias al realizar conversión

Otra consideración importante luego de comenzados los tests de compatibilidad entre los archivos de salida (en ambos metamodelos) fue la de borrar las referencias de los documentos, en especial al realizar la conversión de UMA a SPEM que es donde se eliminan más nodos al documento. Para esto, es necesario que la conversión de un objeto a otro esté finalizada y luego, recorrer el objeto resultado para eliminar las referencias que no estén contenidas en el mismo. Para esto se utiliza la función `check()` que es implementada por los objetos base (`UmaElement` y `SpemElement`), esta clase utiliza los métodos descritos en la subsección 5.4.3.

5.10. Cómo usar el parser

El único requisito de sistema para ejecutar el parser es tener instalado Java en su versión JRE (Java Runtime Environment) o JDK (Java Development Kit), ambas en su versión 6 o posterior[9][10]. Una vez instalado, es necesario dirigirse al directorio donde se encuentra el archivo "memoria.jar" con una terminal, y ejecutar la sentencia siguiente:

```
java -jar memoria.jar <mode> <input-file> <output-file> <original-file>
```

Donde:

- **mode:** Es el modo en que se ejecutará el parser, hay dos opciones, "spem2uma" y "uma2spem" que realizan la conversión de SPEM a UMA y de UMA a SPEM, respectivamente.
- **input-file:** Es la dirección al archivo de entrada, que será parseado.
- **output-file:** Es la dirección donde se alojará (o sobrescribirá) el archivo de salida. Esta sentencia no crea un directorio en caso de no existir.
- **original-file:** Solo usado en el modo "spem2uma", se ingresa el archivo original para recuperar información que no fue considerada en la conversión de SPEM a UMA. En caso de no estar, se hace la conversión solo a partir de la información de SPEM.

6. Solución 2: TCS

La segunda solución desarrollada consiste en utilizar la herramienta **TCS** que existe dentro del proyecto **AMMA**, que es una variación del framework **EMF** (ver 3.3). Todas estas herramientas están inmersas dentro del área de modelamiento.

6.1. Conceptos Previos

6.1.1. Domain Specific Language

Un DSL es un lenguaje diseñado para resolver una cantidad limitada de problemas. En contraste con un GPL (Visto en 3.2.1) que está hecho para resolver tareas más genéricas. Un determinado DSL está en un dominio de interés bien definido y acotado, usualmente poseen una sintaxis concreta (aunque pueden tener una sintaxis abstracta), y tienen una semántica implícita o explícitamente definida.

En el contexto de un **MDE** (ver 3.2), un DSL es visto como un conjunto de modelos coordinados. A continuación se describirán qué modelos están incluidos en un DSL y su propósito:

- **Metamodelo Definición del Dominio**

  Como se dijo, los DSL están acotados a un dominio en particular, es por esto, que cada sentencia en un programa representa interacciones entre los elementos del dominio. Una entidad abstracta representa una conceptualización del dominio, con sus abstracciones y relaciones. Cuando esta entidad abstracta está representada explícitamente se transforma en un metamodelo para el modelo definido en el DSL, y se denomina **Domain Definition MetaModel (DDMM)**[15].

- **Sintaxis Concreta**

  Una sintaxis concreta puede ser definida por un modelo de transformación que mapea el **DDMM** en una metamodelo de despliegue de superficie. Este metamodelo por lo general está expresado en un archivo XML. Un DSL puede tener muchas sintaxis concretas diferentes[15].

- **Semánticas**

  La semántica puede definirse mediante un modelo de transformación que mapea el **DDMM** en otro DSL que posee por sí solo semánticas precisas de ejecución, e incluso en un GPL[15].

6.1.2. El Framework AMMA

En AMMA los DLSs están definidos como un conjunto de modelos, y provee una cantidad de DSLs que ayudan en la definición de más DSLs, estos conforman el core del framework. El core posee un lenguaje para describir metamodelos, llamado **KM3**, y un lenguaje para la transformación de modelos, llamado **ATL**. A estas funcionalidades se les agrega un lenguaje para la definición de sintaxis concretas de un DSL, **TCS** [15].

6.1.3. Kernel MetaMetaModel

Kernel MetaMetaModel (**KM3**) es un metametamodelo similar a un MetaObject-Facility[11] (**MOF**) mucho más simple. En palabras simples, en **KM3** se definen las clases que contendrá el

metamodelo en cuestión, con sus respectivas referencias y atributos (además de la multiplicidad de cada uno), todo esto en una sintaxis sencilla de entender [15]. Una vista simplificada de lo que contiene KM3 se puede observar en la figura 4.

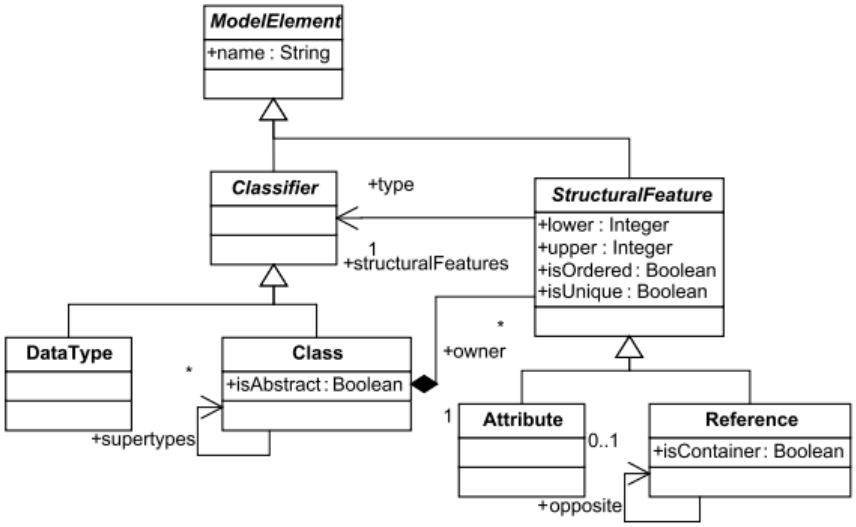


Figura 4: Diagrama de clases simplificado de KM3

6.1.4. Technical Spaces

Un Technical Space (**TS**) es un framework para la administración de modelos que contiene además herramientas para operar sobre los modelos definidos en el framework [14]. La intención principal al hablar de **TS** es tratar las tecnologías a un nivel más abstracto, esto permite que se hable de sus diferencias y similitudes, o su capacidad de integración.

6.1.5. Textual Concrete Syntax

TCS es un lenguaje que permite la especificación de proyectores (y su generación automática) entre el **TS** de la gramática y el **TS** del **MDE** para un cierto **DSL**. La realización del puente entre estos dos **TS** se realiza con un inyector y un extractor. El inyector toma un modelo de un **DSL**, expresado en su sintaxis textual concreta, y genera un modelo afín al metamodelo de dicho **DSL** en el **TS** del **MDE**.
Este proceso se inicia definiendo el metamodelo y la sintaxis textual concreta de un cierto **DSL**, a partir de los cuales se obtienen tres entidades: La gramática expresada en **ANTLR**, el inyector y extractor. La gramática es usada para generar el inyector, con el **ANTLR** parse generator (**ANTLR GEN**). Ver la figura 5.

El extractor trabaja sobre la representación interna de modelos expresados en el **DSL** y crea su representación textual, por lo que sería posible generar un extractor por cada lenguaje **DSL**,

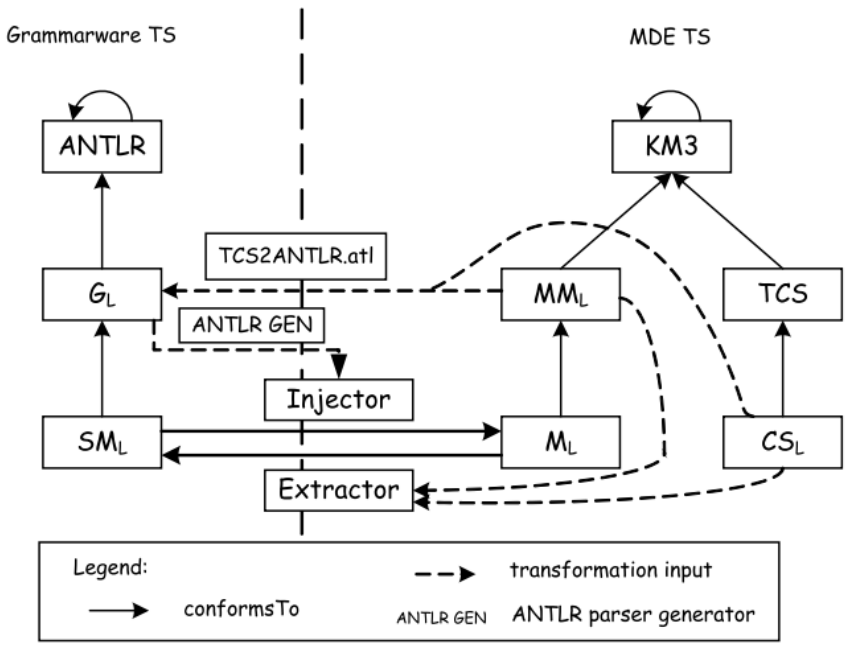


Figura 5: Vista del funcionamiento de TCS

sin embargo, lo que provee **TCS** es implementar sólo un extractor que pueda interpretar todos los lenguajes. De esta forma, el extractor toma la sintaxis concreta y el metamodelo del **DSL** y genera una representación textual del modelo.
Usar **TCS** es mucho más simple que desarrollar inyectores y extractores ad-hoc a cada lenguaje. Una sola especificación sirve para ambas direcciones. Sin embargo, todas estas facilidades se traducen en restricciones. Es por esto que al momento de escribir la sintaxis concreta, o se adapta a las posibilidades de TCS, o se simplifica el metamodelo. De hecho, una restricción importante impuesta por **TCS**, es que los metamodelos deben contener un elemento raíz, lo equivalente a un símbolo inicial en la gramática correspondiente [15].

6.1.6. Atlas Transformation Language

ATL es un lenguaje de transformación de modelos que permite especificar cómo uno (o más) modelos de destino, pueden ser producidos a partir de un conjunto de modelos de partida. Es decir, permite realizar transformaciones de modelos (Figura 6).

Formalmente, una transformación de modelo simple tiene que definir la manera de generar un modelo M_b, que se conforma a partir de su metamodelo MM_b, a partir de un modelo M_a, que a su vez se conforma de su metamodelo MM_a. La transformación de modelo en sí debe ser definida como un modelo (M_t) que a su vez se conforma a partir de su metamodelo (MM_t). A su vez, estos tres metamodelos se conforman a partir del mismo metametamodelo MMM[18].

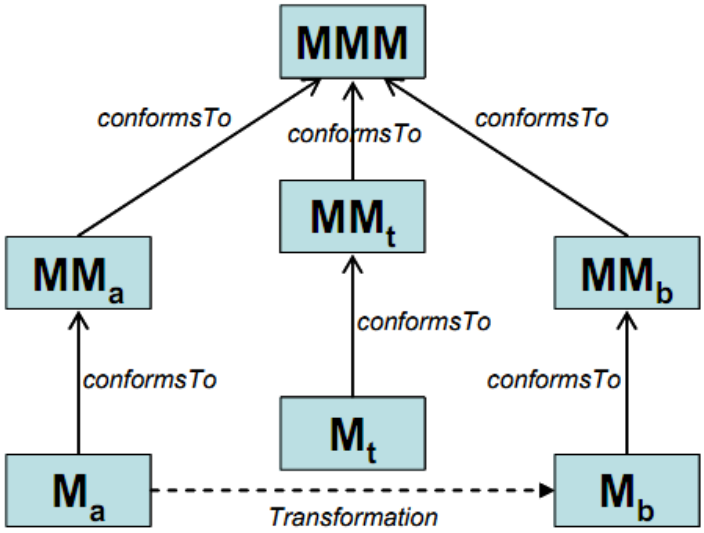


Figura 6: Transformaciones de modelo

6.2. Metodología

Esta sección explicará cómo se realizará la solución de la conversión de archivos. Los archivos necesarios para realizar la conversión son precisamente los archivos que describen la estructura de ambos DSL: *uma.ecore* y *spem.ecore*, ambos archivos fueron realizados por Luis Silvestre, estudiante de doctorado del Departamento de Computación.

- A partir de dichos archivos *.ecore* se puede obtener con el mismo **EMT** los archivos *.km3* (ver 6.1.3). Este paso se realiza para ambos metamodelos, obteniendo al final de esta etapa los archivos *uma.km3* y *spem.km3*.
- Se escribe la sintaxis concreta de ambos lenguajes en TCS, obteniendo los archivos *uma.tcs* y *spem.tcs*.
- Con lo anterior se generará automáticamente la gramática de cada DSL, eso lo realiza ANTLR, obteniendo los archivos *spem_ANTLR3.g* y *uma_ANTLR3.g*.
- Una vez descritas las sintaxis concretas de cada metamodelo se realizará la conversión entre metamodelos con los archivos *uma2spem.atl* y *spem2uma.atl*[4].

6.3. La definición del metamodelo en KM3

La definición de los metamodelos en KM3 se obtiene a partir de los archivos *.ecore* correspondientes a cada DSL.

[4]Esta parte no se realizó en este trabajo (Ver sección 9.1).

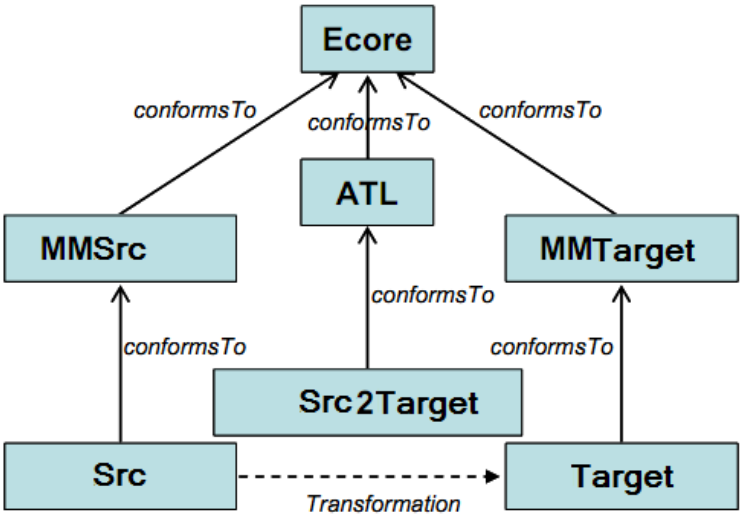


Figura 7: Overview de ATL

La estructura de un archivo Ecore es la de un documento XML que contiene en cada nodo una declaración de un clasificador del lenguaje. A continuación se muestra el nodo del documento correspondiente a la declaración del elemento "Activity" del lenguaje **UMA**.

```
<eClassifiers xsi:type="ecore:EClass" name="Activity"
eSuperTypes="//WorkBreakdownElement //FulfillableElement //VariabilityElement //WorkDefinition">
  <eStructuralFeatures xsi:type="ecore:EReference" name="breakdownElement" ordered="false"
  upperBound="-1" eType="//BreakdownElement" containment="true"/>
  <eStructuralFeatures xsi:type="ecore:EReference" name="roadmapID" ordered="false"
  upperBound="-1" eType="//RoadmapID" containment="true"/>
</eClassifiers>
```

Luego de realizada al conversión, este mismo elemento se observa de la manera siguiente

```
class Activity extends WorkBreakdownElement, FulfillableElement,
VariabilityElement, WorkDefinition {
    reference breakdownElement[*] container : BreakdownElement;
    reference roadmapID[*] container : RoadmapID;
}
```

Los documentos son equivalentes en contenido, pero expresados en un contexto distinto. No hay mucho más que decir respecto a esta parte, debido a que se hace de forma automatizada dentro de **AMMA**.

Para el elemento `Activity` en **SPEM** se tiene lo siguiente en el Ecore:

```
<eClassifiers xsi:type="ecore:EClass" name="Activity"
eSuperTypes="/0/ProcessStructure/WorkBreakDownElement /0/Core/WorkDefinition
/0/MethodPlugin/VariabilityElement">
  <eStructuralFeatures xsi:type="ecore:EReference" name="nestedElements"
  upperBound="-1" eType="/0/ProcessStructure/WorkBreakDownElement" containment="true"/>
  <eStructuralFeatures xsi:type="ecore:EReference" name="processPerformerID"
  ordered="false" upperBound="-1" eType="/0/ProcessPerformerID" containment="true"/>
  <eStructuralFeatures xsi:type="ecore:EReference" name="processParameterID"
  ordered="false" upperBound="-1" eType="/0/ProcessParameterID" containment="true"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="usedActivity"
  ordered="false" unique="false" eType="/1/String"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="useKind" ordered="false"
  unique="false" eType="/1/String"/>
</eClassifiers>
```

Mientras que el KM3 equivalente es:

```
class Activity extends WorkBreakDownElement, WorkDefinition, VariabilityElement {
    reference nestedElements[*] ordered container : WorkBreakDownElement;
    reference processPerformerID[*] container : ProcessPerformerID;
    reference processParameterID[*] container : ProcessParameterID;
    attribute usedActivity[0-1] : String;
    attribute useKind[0-1] : String;
}
```

6.4. La sintaxis concreta en TCS

Para esta sección se consideró como algo sumamente importante, el considerar lo siguiente: una clase del lenguaje puede ser utilizada dentro de un documento en distintos nodos, por así decirlo, de manera que el nombre del nodo sería un elemento más dentro del lenguaje, por ejemplo, las actividades (la clase `Activity` dentro de ambos metamodelos) son usadas en distintas partes de cada documento, en UMA puede ser:

```
<ProcessElement xsi:type=‘‘uma:Activity’’ ... >...</ProcessElement>
```

Sin embargo, también puede encontrarse como:

```
<BreakdownElement xsi:type=‘‘uma:Activity’’ ... >...</BreakdownElement>
```

Por lo que agregar el nombre del nodo al elemento es una opción que ayudaría mucho a solucionar este problema, sin embargo, dejar los nombres de los nodos explícitos ayuda a mantener la estructura del documento, debido a que, para el caso de la actividad, no es equivalente usar un *template*[5] con cualquier nombre de nodo. Es decir, estos nombres son keywords para la sintaxis. De hecho, usar una actividad en un nodo `BreakdownElement` dentro del contexto de `ProcessElement` es erróneo, por tanto se debe hacer un template distinto para cada nodo en que se puede contener una actividad.

[5]Se llama template a la definición de la sintaxis concreta para un elemento del metamodelo.

Esto se traduce en que se debe generar dos templates para la actividad que serán dos nodos que se pueden utilizar en distintos contextos dentro del documento. Como solo se puede tener un template por elemento en la definición de elementos del lenguaje (en el archivo km3), se genera otro elemento del lenguaje que será un hijo del original, es decir, se agrega un elemento `ActivityBE`.

```
class ActivityBE extends Activity {

}
```

Ahora se necesita que, en todas las partes donde antes se referenciaba únicamente a `Activity`, distinguir si se trata de `Activity` o `ActivityBE` según corresponda. Como los templates son demasiado grandes para incluirlos acá, se puede observarlos en el anexo A. Hay que agregar que tienen dicho tamaño porque contienen todos los atributos y referencias de sus ancestros.

Esta situación se repite en SPEM con las actividades, donde los nombres de los nodos pasan de ser `processElements` y `nestedElements`. Para los `processElements` se tiene:

```
<processElements xsi:type=‘‘processStructure:Activity’’ ... >...</processElements>
```

Mientras que para los `nestedElements`:

```
<nestedElements xsi:type=‘‘processStructure:Activity’’ ... >...</nestedElements>
```

6.5. Problemas durante el desarrollo

Respecto a la conversión realizada en UMA, se intentó realizar una conversión con el metamodelo ya definido, que en total sobrepasa las 50 definiciones. En este caso, TCS entrega un “out of memory error”, este error se intentó solucionar aumentando el tamaño del heap en Eclipse, como se hace cuando un proyecto Java se queda sin espacio. Esto se puede observar en la figura 8.

Este problema, junto con la posibilidad de perfeccionar la solución en Java (descrita en la sección 5) en el tiempo correspondiente al desarrollo de este trabajo, hicieron que la solución en TCS quede para el trabajo futuro, esto se describirá con más detalle en la sección 9.1. Además en la discusión (sección 8) se hablará más de las limitaciones de TCS para solucionar el problema original.

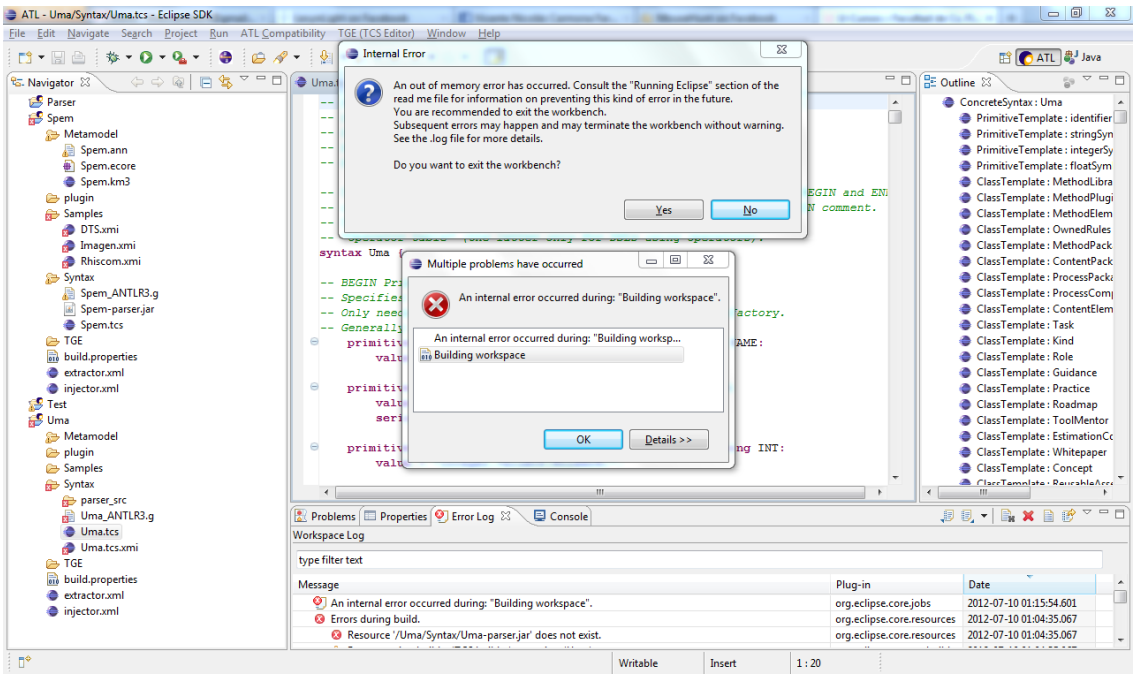


Figura 8: Problema con el tamaño de la memoria en TCS

7. Resultados

En esta sección se exponen los resultados obtenidos de la experimentación para ambas soluciones.

7.1. Experimentación en Java

Para comprobar la solución en Java se consideraron tres procesos de negocio reales de empresas establecidas: *Rhiscom*, *DTS* y *Imagen*. La manera de proceder es la siguiente:

1. Se toma el archivo correspondiente, que describe el proceso de negocio de la empresa, y se visualiza con EPF Composer.

2. Dicho archivo se convierte al archivo XMI usando el programa en Java.

3. Se toma el archivo XMI y se visualiza en Eclipse Modeling Tools.

4. Se reconvierte el archivo XMI a XML usando el programa en Java.

5. Se vuelve a visualizar el archivo correspondiente al proceso, en XML, con EPF Composer.

A continuación se mostrará lo obtenido con el seguimiento de estos pasos de manera comparativa, es decir, se mostrarán las visualizaciones del documento original en EPF Composer y la visualización del archivo convertido en EMT explicando la concordancia entre los elementos y a la vez exponiendo los elementos que no están considerados por SPEM y que no se visualizarán en la plataforma. De manera equivalente, para la segunda conversión, se comparará tanto la visualización original con la

final en EPF Composer, y la visualización en EMT con la resultante en EPF. Esto para cada uno de los tres procesos.

7.2. Conversión XML a XMI

Rhiscom

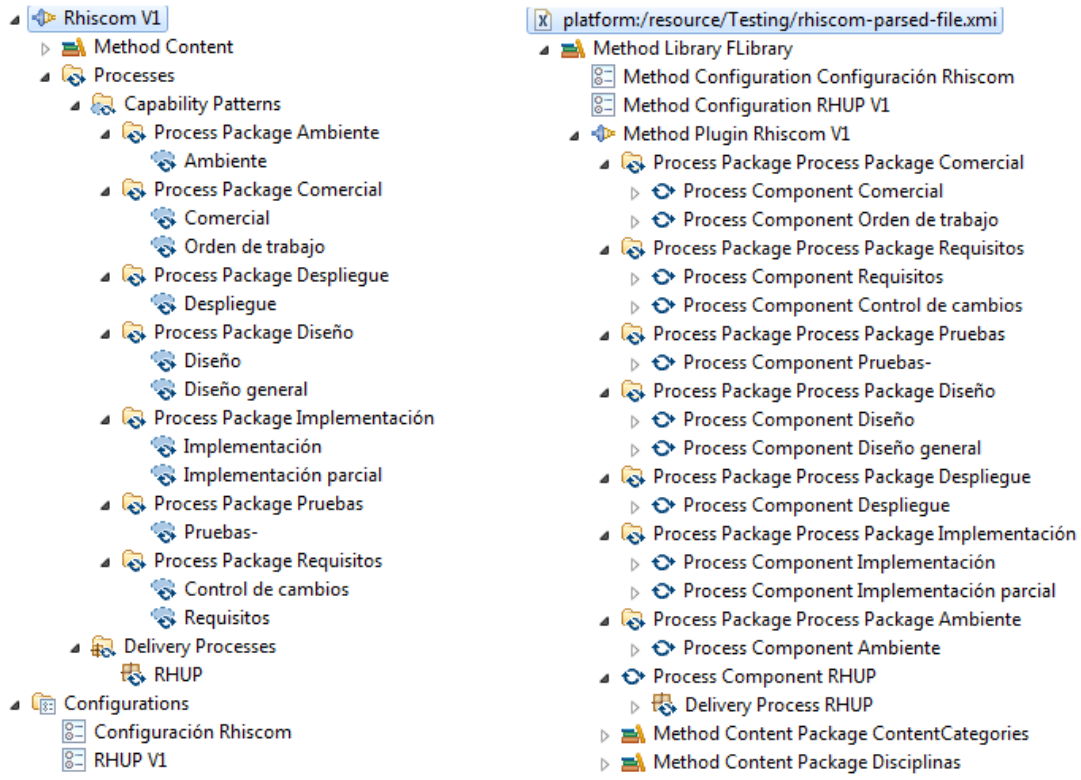


Figura 9: A la izquierda, la visualización en EPF, a la derecha, su equivalente en EMT para Rhiscom

La figura 9 muestra la correlación entre los paquetes de procesos *ProcessPackage*. El nivel jerárquico que aparece en la imagen no es necesariamente la que se refleja a nivel de código XML en los documentos, esto porque EPF Composer es una herramienta que está hecha especialmente para el metamodelo UMA, por tanto no es un despliegue tácito de los elementos, sino que hay toda una lógica involucrada detrás de los elementos que no necesariamente se refleja en el archivo de salida del proceso.

Lo anterior se nota especialmente en el hecho de que la aplicación (EPF Composer) ordena los procesos en un nivel superior, sin embargo en el documento XML están organizados de igual forma que en SPEM, contenidos en el *MethodPlugin*.

En EPF, se hace doble click sobre cualquier *CapabilityPattern* y se despliega una ventana con multiples pestañas con las referencias contenidas en éste, como se muestra figura 10 para “Ambi-

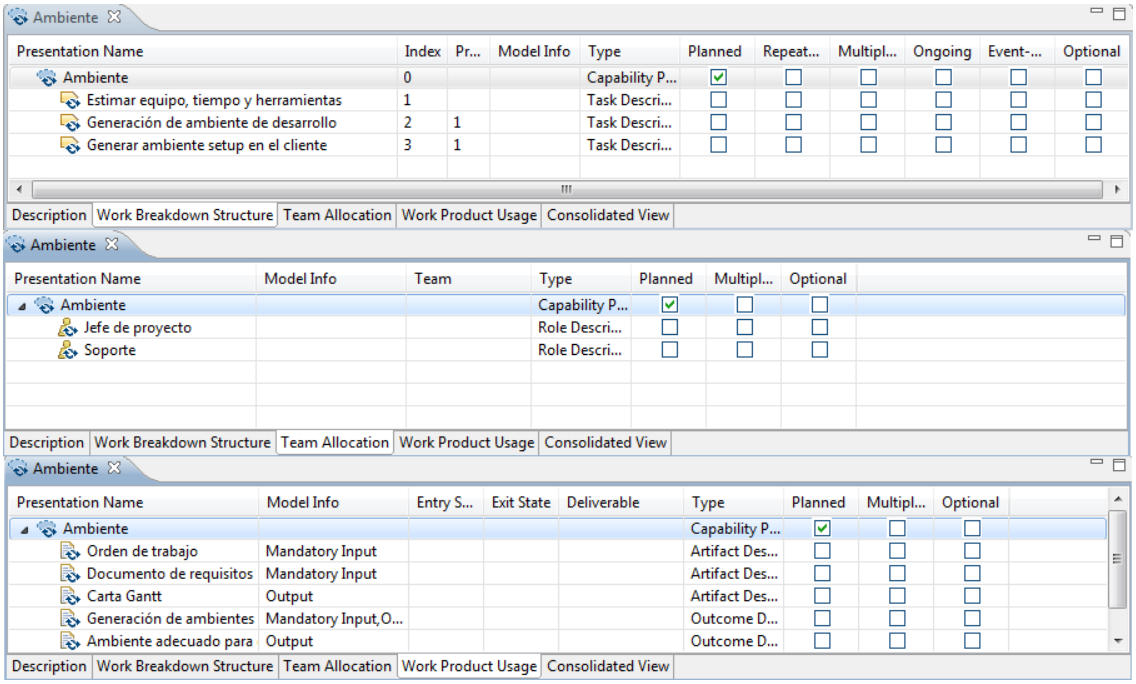


Figura 10: El CapabilityPattern Ambiente de Rhiscom en EPF Composer

ente".

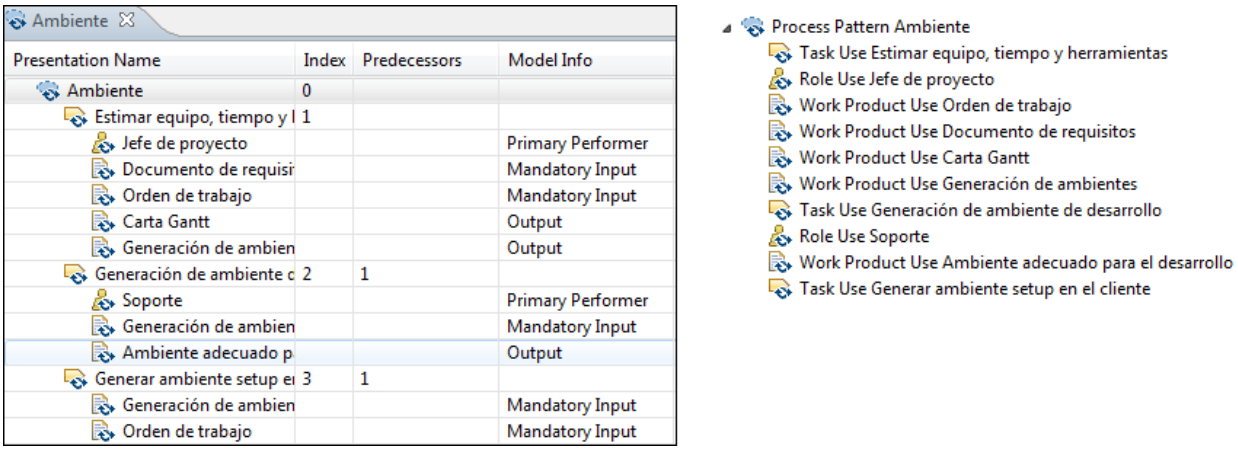


Figura 11: Ambiente de Rhiscom visualizado en EPF Composer y EMT

En EMT en cambio, las referencias contenidas en los objetos simplemente se muestran como nodos hijos dentro de la jerarquía, lo cual se asemeja a la última vista mostrada en EPF Composer, "Consolidated View", en esta vista los elementos se organizan de manera de distinguir los productos de negocio y roles asociados a cada tarea. Para "Ambiente" se puede observar esto en la figura 11.

Como se puede observar en la última figura, efectivamente se pierde información de orden jerárquico entre Tareas, Roles y Productos de Negocio al realizar la conversión. Esto se debe tomar

como una limitación del modelo, que no soporta, para una tarea *TaskUse* por ejemplo, guardar información de roles y documentos involucrados con la realización de la misma. Se menciona en este punto por ser algo que no se observó a priori como limitación entre los lenguajes.

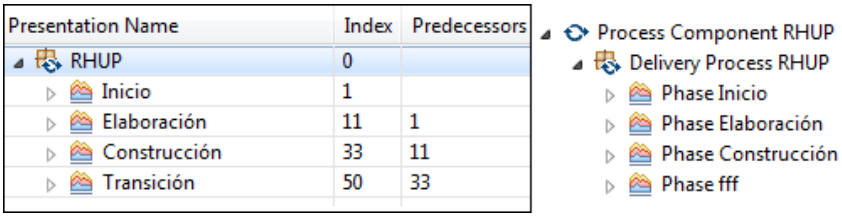


Figura 12: DeliveryProcess de Rhiscom en EPF Composer y EMT

Un *DeliveryProcess* (como se explica en la sección 4) agrupa de principio a fin el ciclo de vida de un proyecto. En la figura 12 se observa el *DeliveryProcess RHUP* que contiene cuatro fases (*Phase*), en la figura 13 se observa la *Phase Construcción*.

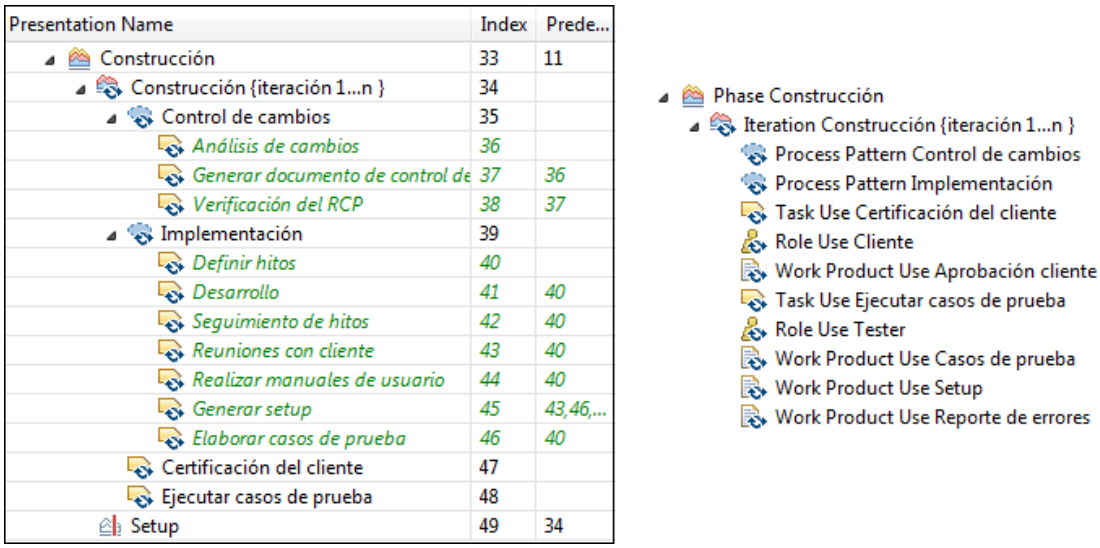


Figura 13: Descripción de la Phase Construcción de Rhiscom

La fase de construcción está compuesta únicamente de la *Iteration Construcción*. Al igual que en la figura 11, en EPF Composer se muestra únicamente las tareas involucradas en la iteración, debido a que EPF Composer organiza jerárquicamente los roles y productos de trabajo incluidos por tarea. Además, la iteración contiene dos *CapabilityPattern* (que en SPEM se llama *ProcessPattern*): "Control de cambios" e "Implementación". Estos elementos a nivel de código no están contenidos físicamente en la iteración, sino que están referenciados a otro elemento en el documento, es por esto que en la visualización en EMT no se observa el contenido de los *ProcessPattern*. Sin embargo, en EPF Composer, se traen los elementos para facilitar la visualización, debido a que, por definición, un *CapabilityPattern* contiene información genérica que es aplicable a muchos procesos semejantes.

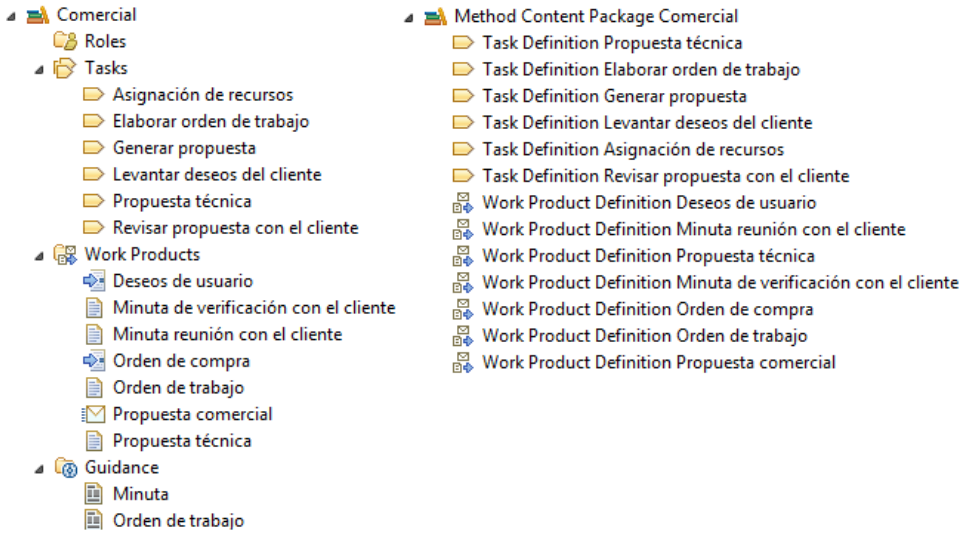


Figura 14: Comparación entre los Method Content de Rhiscom con ambos metamodelos

Finalmente se expone una comparación realizada entre los elementos del *MethodContent Comercial* que contiene las definiciones de todos los elementos usados por actividades, procesos, etc. Como se puede observar en la figura 14, en EPF Composer, se agrupan los elementos por tipo, es así como se tienen los roles y guías agrupados en carpetas, además de los documentos *Outcome* y productos de negocio *WorkProduct* en general se agrupan dentro de la carpeta WorkProduct (En UMA *Outcome* extiende de *WorkProduct*). En EMT solo se tiene la relación de contenedor-contenido, donde los elementos pueden organizarse de manera arbitraria solo respetando el nodo contenedor.

DTS

En la figura 15 se muestra los *CapabilityPattern* o *ProcessPattern* en SPEM

Nuevamente en la figura 16 se puede observar lo que sucede al hacer doble click (en EPF Composer) sobre el patrón de proceso “Acordar Enfoque Técnico”, y su visualización equivalente en EMT.

Un punto importante a considerar está en que, para la visualización de un *MethodContent* que no contiene relaciones a otros elementos, en EPF Composer de todas maneras se observan las distintas categorías que podría tomar un elemento en caso de estar relacionado, esto se observa en la figura 17.

Finalmente en el *ProcessPattern CicloVidaTransporte* se puede observar que en EMT solo se mantiene la referencia a las actividades que componen cada fase, pues estas ya fueron definidas anteriormente, de hecho, en la *Phase Inicio* se puede observar la actividad “Acordar Enfoque Técnico” que es la misma descrita en la figura 16. En EPF Composer en cambio, para facilitar la lectura del documento se traen los elementos de dicho patrón de proceso.

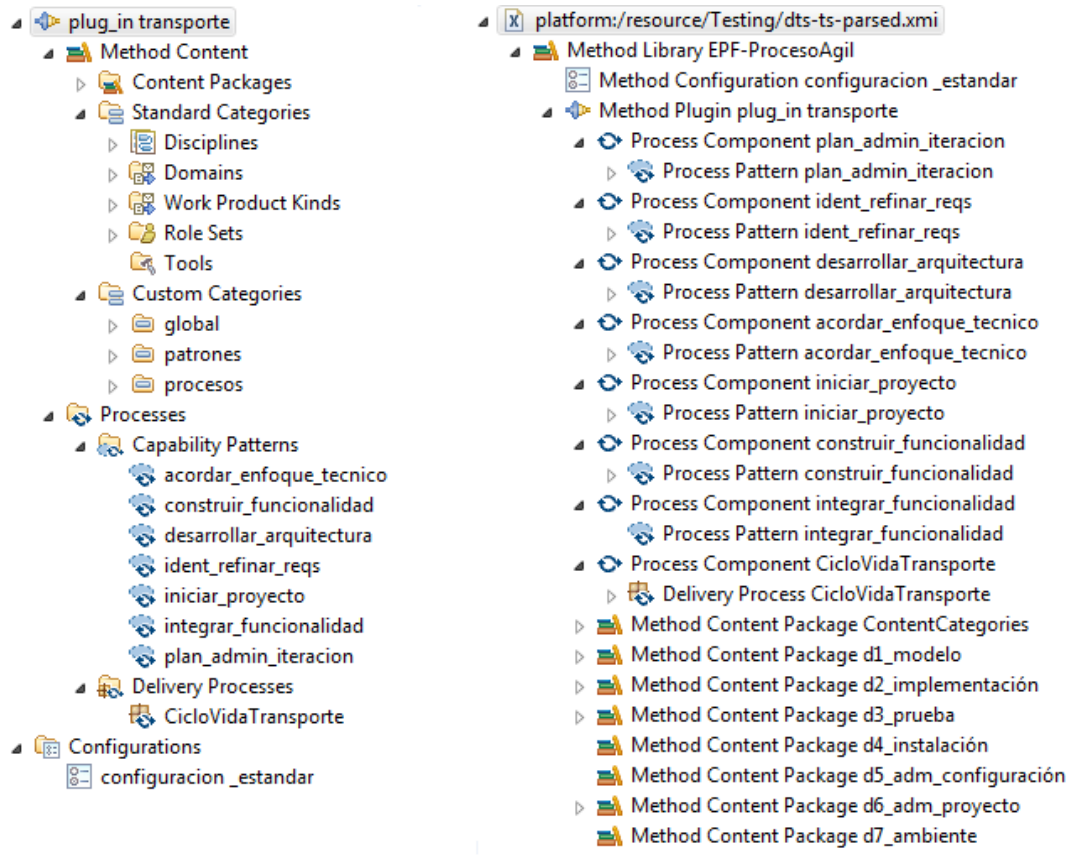


Figura 15: A la izquierda, la visualización en EPF, a la derecha, su equivalente en EMT para DTS

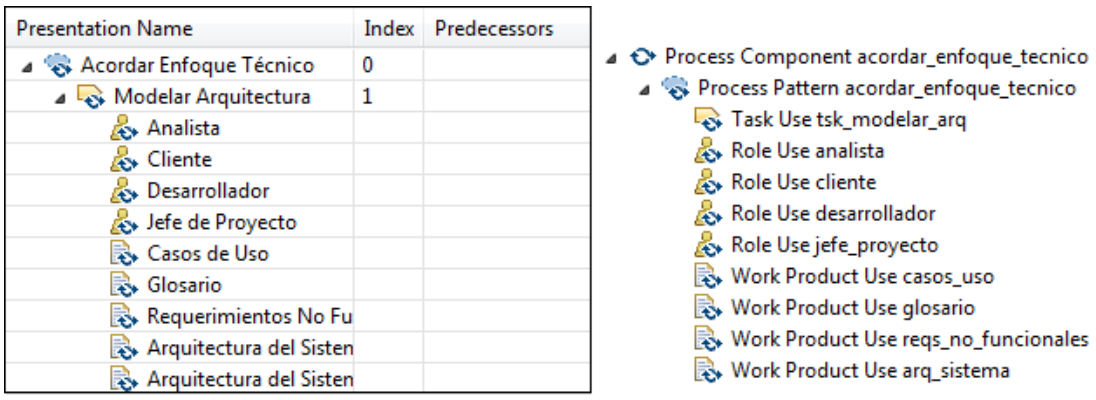


Figura 16: El patrón de proceso Acordar Enfoque Técnico de DTS

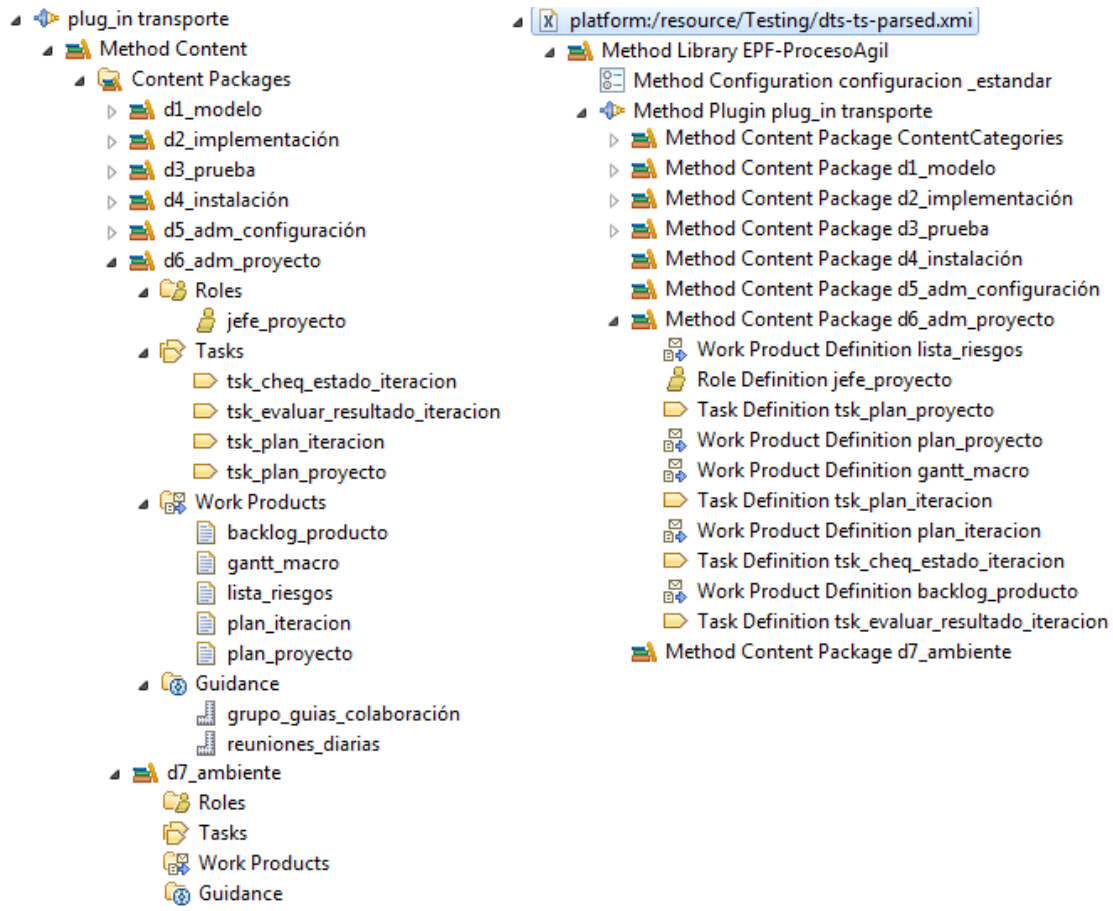


Figura 17: Los MethodContent de DTS

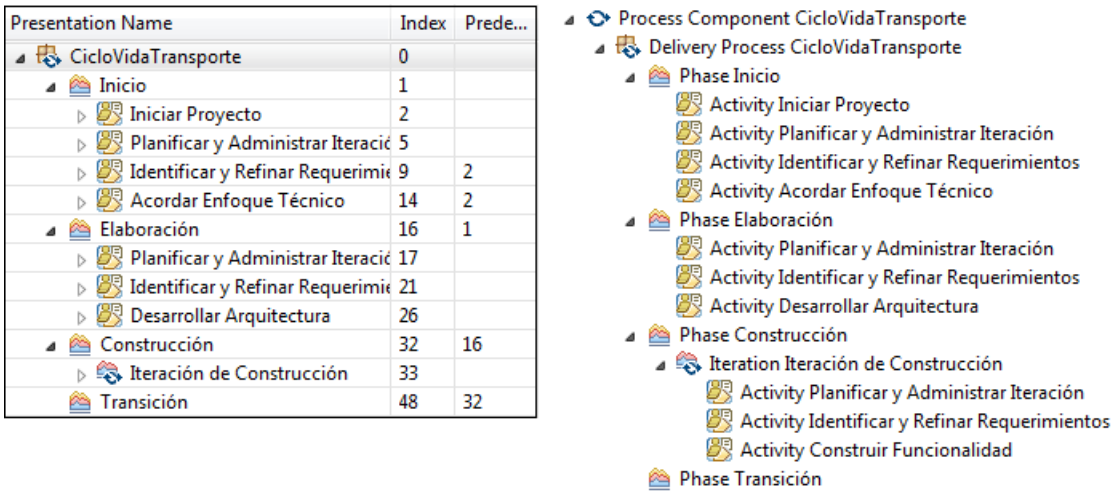


Figura 18: El ProcessPattern de DTS

Imagen

En UMA, `name` y `presentationName` representan el nombre interno con el que se conoce el elemento, y el nombre de despliegue, para el caso anterior se dio que eran iguales, pero esto no necesariamente será cierto. Lo que se observa en casi todas las figuras de esta sección es que para EMT no es una palabra clave "presentationName" por lo que usa el nombre por defecto del elemento para representarlo en pantalla. Esto se observa en la figura 19.

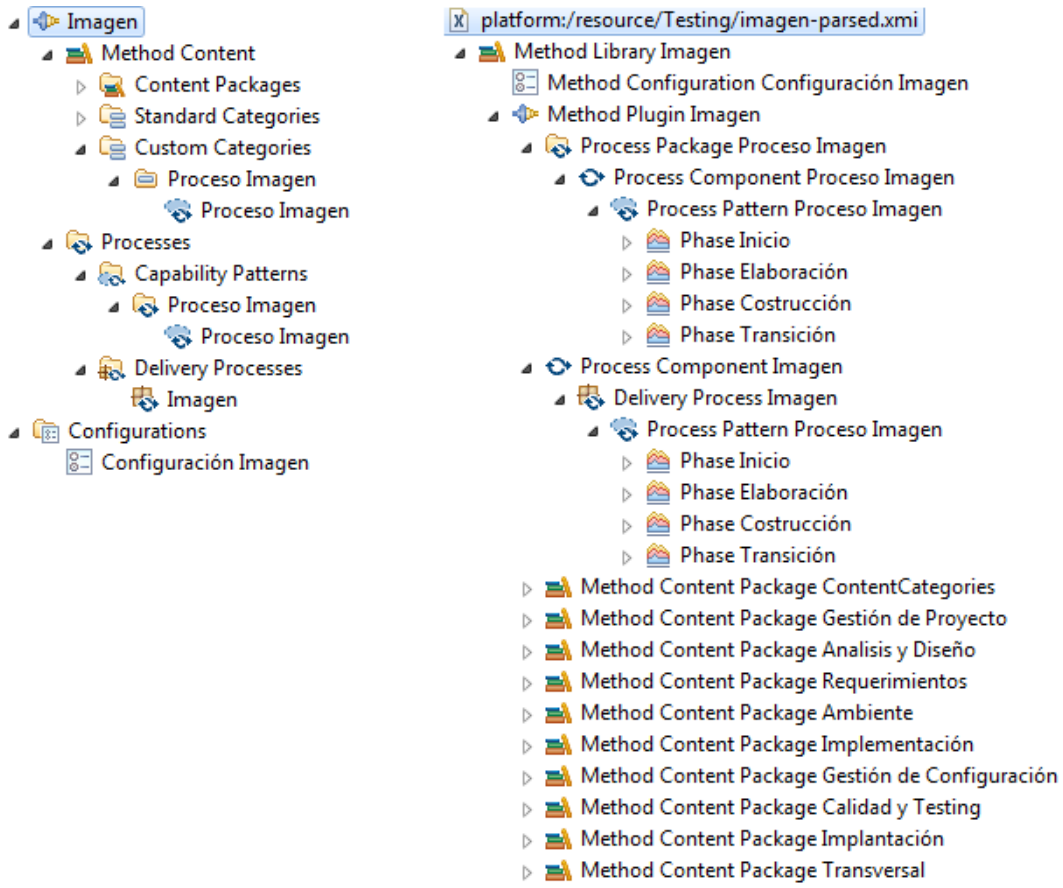


Figura 19: A la izquierda, la visualización en EPF, a la derecha, su equivalente en EMT para Imagen

En la figura 20 se observa que Imagen cuenta sólo con el *CapabilityPattern Proceso Imagen* en la sección de Capability Patterns. También está el *DeliveryProcess Imagen*, y, aunque no se alcanza a observar en la parte EPF de la figura, pero sí en EMT, este último referencia el proceso descrito en el primero.

La cantidad de información que cuenta este proceso es mucho mayor a la de los metamodelos anteriores. Por lo mismo, para la visualización de cualquier elemento del proceso en EMT (que no hace ordenes jerárquicos de información) se debe navegar una lista enorme de tareas, roles y productos de negocio. Un ejemplo de esto es la figura 21 donde se observa el contenido de la tarea

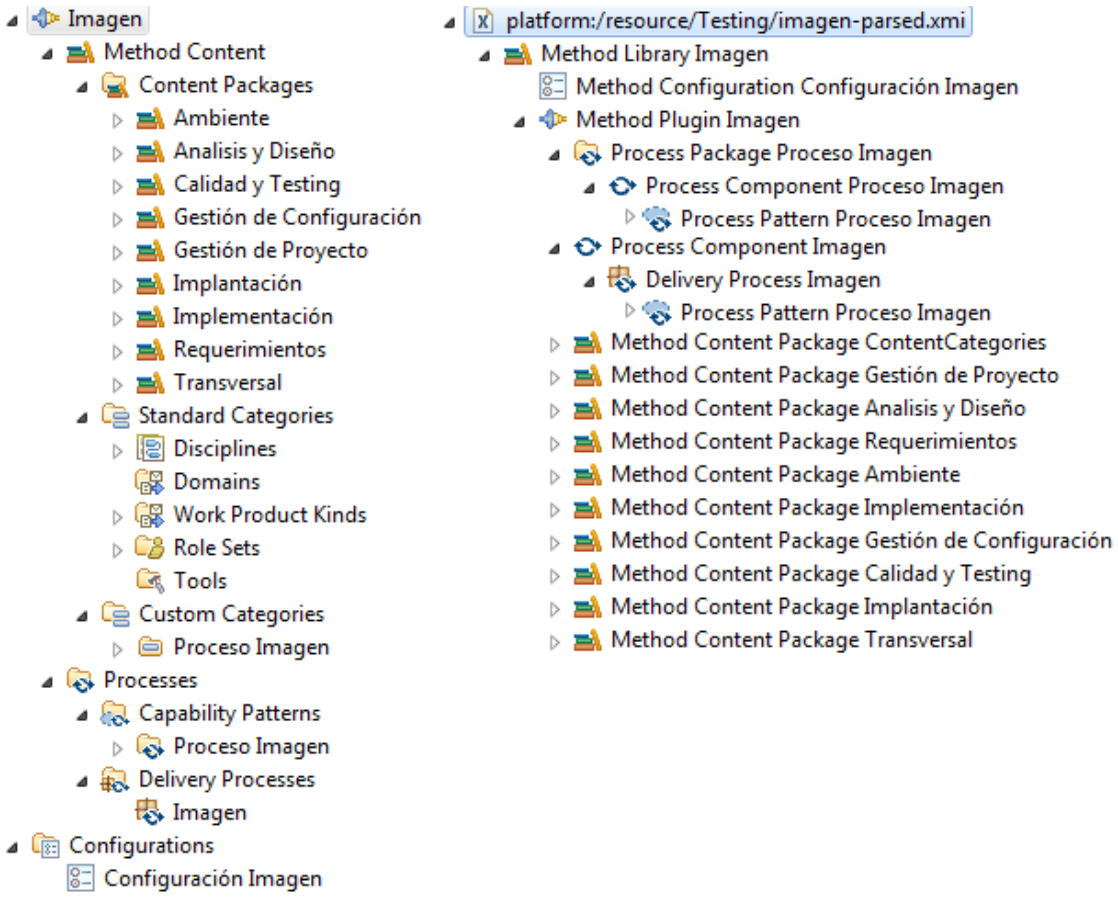
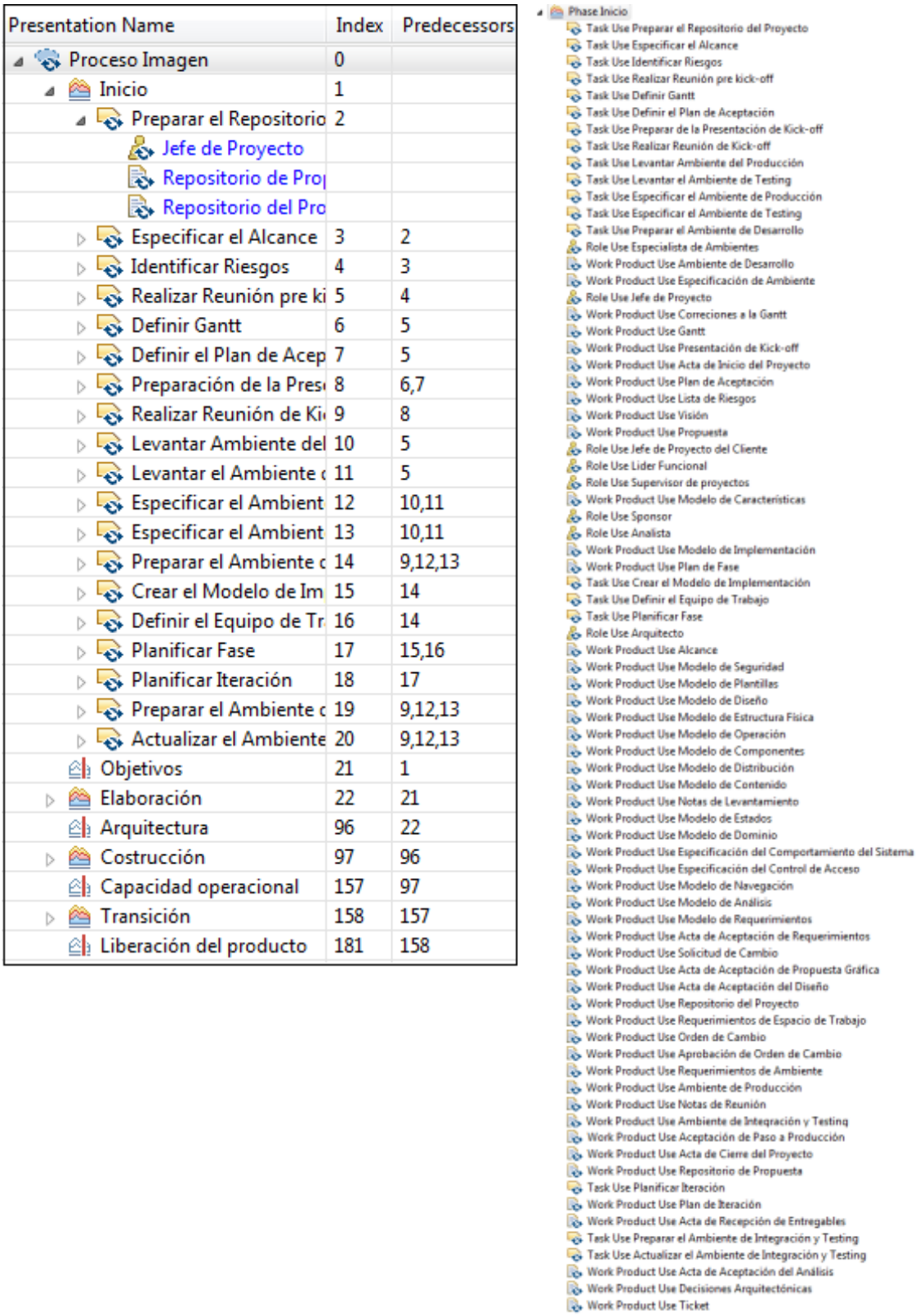


Figura 20: Procesos de Imagen

"Preparar el repositorio" que está contenida en la fase "Inicio".

La figura 22 muestra el *DeliveryProcess Imagen* que, como se mencionó anteriormente, utiliza el mismo *CapabilityPattern Proceso Imagen*, por lo que se observan las cuatro fases contenidas dentro del patrón. En esta figura se observa además que dos de las cuatro fases están compuestas de un conjunto de actividades ("Elaboración" y "Construcción"), dentro de estas se eligió la más compacta "Realizar Revisión Técnica" para observar sus atributos, nuevamente se observa que, mientras EPF agrupa los elementos por tarea, EMT los muestra a todos en un mismo nivel.



Presentation Name	Index	Predecessors
Proceso Imagen	0	
Inicio	1	
Preparar el Repositorio	2	
Jefe de Proyecto		
Repositorio de Proj		
Repositorio del Pro		
Especificar el Alcance	3	2
Identificar Riesgos	4	3
Realizar Reunión pre ki	5	4
Definir Gantt	6	5
Definir el Plan de Acep	7	5
Preparación de la Pres	8	6,7
Realizar Reunión de Ki	9	8
Levantar Ambiente del	10	5
Levantar el Ambiente d	11	5
Especificar el Ambient	12	10,11
Especificar el Ambient	13	10,11
Preparar el Ambiente d	14	9,12,13
Crear el Modelo de Im	15	14
Definir el Equipo de Tr	16	14
Planificar Fase	17	15,16
Planificar Iteración	18	17
Preparar el Ambiente d	19	9,12,13
Actualizar el Ambiente	20	9,12,13
Objetivos	21	1
Elaboración	22	21
Arquitectura	96	22
Costrucción	97	96
Capacidad operacional	157	97
Transición	158	157
Liberación del producto	181	158

Figura 21: El CapabilityPattern Proceso Imagen

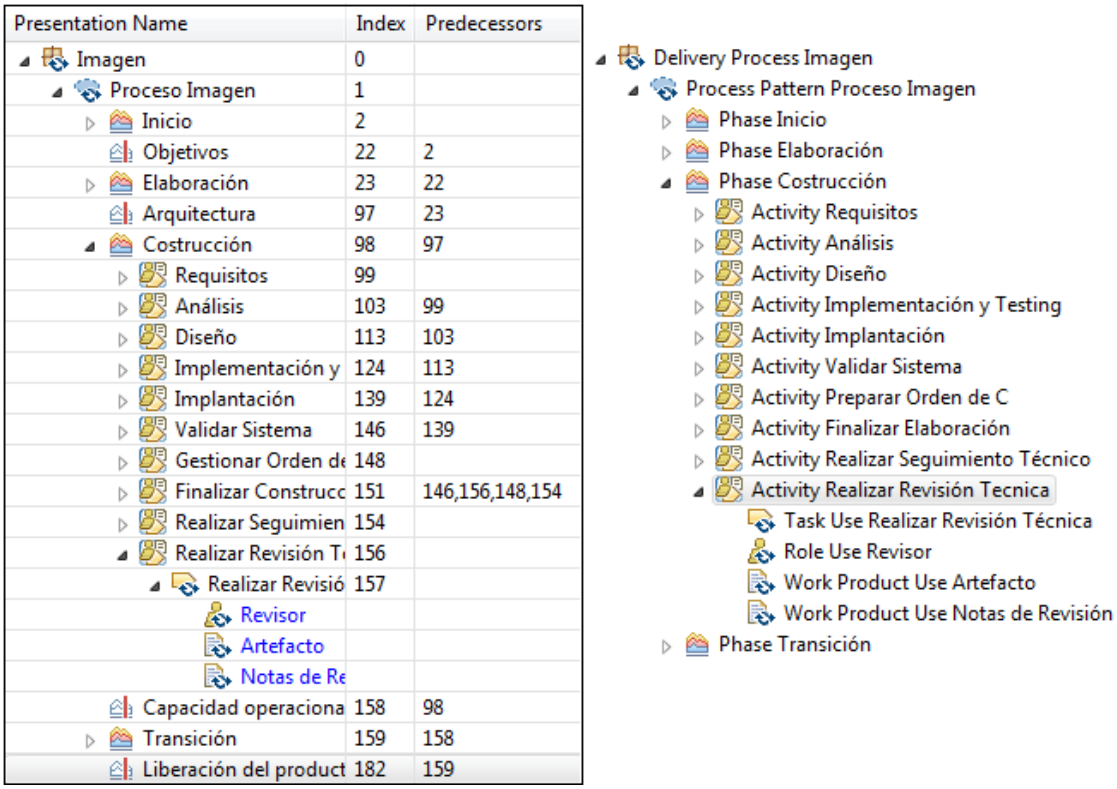


Figura 22: El DeliveryProcess de Imagen

7.3. Conversión XMI a XML

Esta sección se divide en dos experimentos principales que se realizaron para mostrar la diferencia de resultados, la primera de esta compara los archivos convertidos para mostrar la necesidad de utilizar el archivo original, mientras que la segunda muestra lo que sucede al eliminar elementos en XMI con respecto al archivo original. Debido a que los resultados fueron semejantes para los tres procesos (DTS, Rhiscom e Imagen), sólo se muestra los resultados para DTS.

Comparación entre conversión con archivo original y sin éste

La conversión de XMI a XML se realiza convirtiendo el archivo XMI a XML directamente. Si bien esta conversión se realiza de manera efectiva sin tener problemas de compatibilidad con la herramienta EPF Composer, sí se pierde mucha información, lo que hace que sea más que insuficiente la conversión. Luego se realiza la conversión empleando, para cada plugin, el documento original junto con el documento SPEM correspondiente a su conversión. Se espera observar qué elementos están presentes al realizar la conversión directa, y cuáles fueron agregados sólo al emplear el documento original.

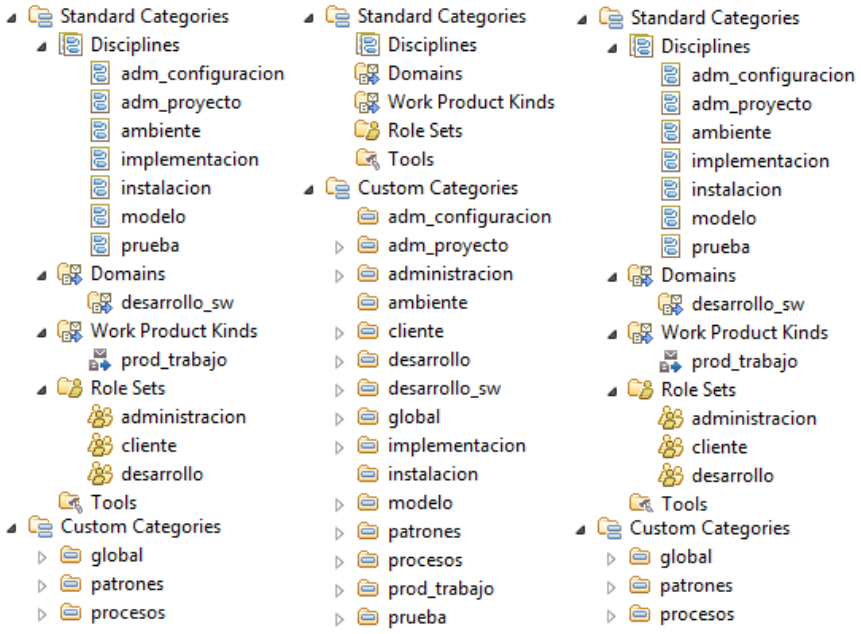


Figura 23: Categorías: Izquierda, Archivo Original; Centro, Conversión sin original y Derecha, Conversión con original

La figura 23 compara la conversión de categorías con y sin el archivo original; en el caso de la conversión con el original, este es idéntico al original, mientras que al no tener el archivo original, la conversión convierte todas las categorías existentes en `CustomCategory`, no permitiendo observar de qué tipo es cada elemento. Es así como el *Role cliente* pasa a ser *CustomCategory cliente* sin el

archivo original, siendo este un comportamiento no deseado. Un aspecto importante a destacar de esta figura es que, si bien se pierde el tipo, se mantiene la cantidad de elementos contenidos (en la conversión sin el documento original).

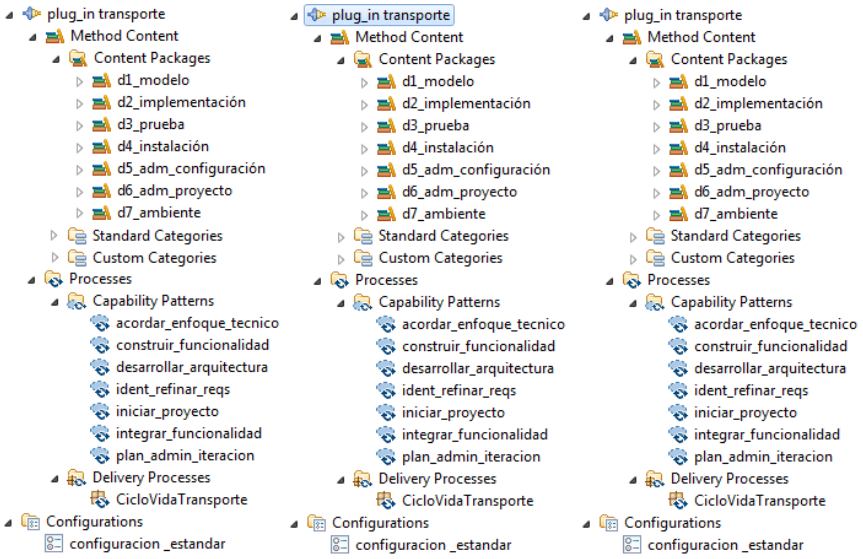


Figura 24: Procesos y Contenido: Izquierda, Archivo Original; Centro, Conversión sin original y Derecha, Conversión con original

Como se puede apreciar en la figura 24, no hay diferencia entre `MethodContent` y `Process` en las conversiones. En las dos figuras siguientes (Fig.25 y Fig.26) se observa que donde se pierde más información es en el `DeliveryProcess` y `CapabilityPattern` donde no se observa información alguna de los procesos y patrones al realizar la conversión sin el archivo original. Pero con este no se observa variación respecto al original.

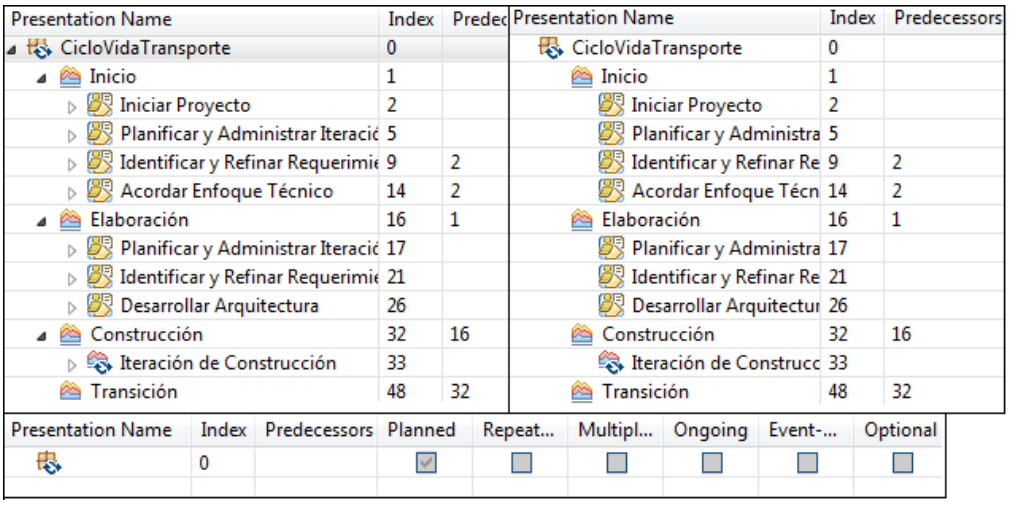


Figura 25: Delivery Process: Arriba Izquierda, Archivo Original; Derecha, Conversión con original y Abajo, Conversión sin original

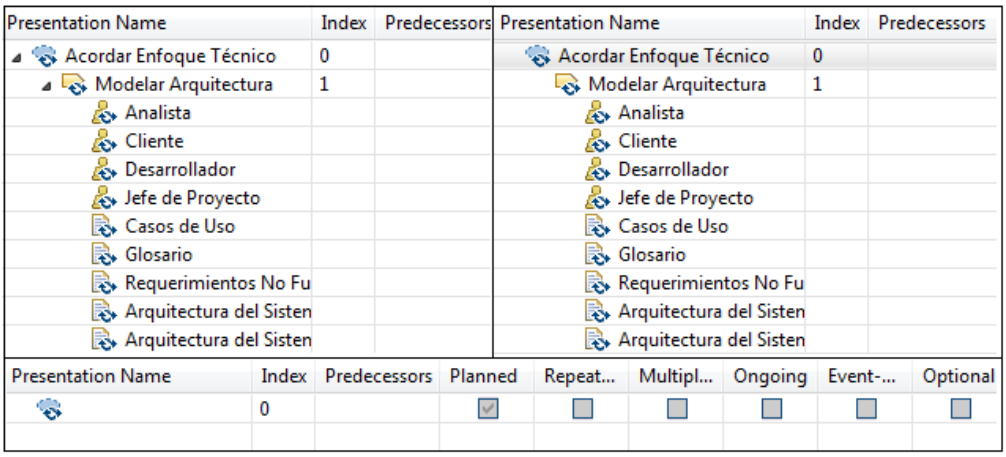


Figura 26: Capability Pattern: Arriba Izquierda, Archivo Original; Derecha, Conversión con original y Abajo, Conversión sin original

Eliminando elementos en el XMI antes de la conversión

Este tipo de pruebas consisten en eliminar elementos del proceso en EMT (equivale a eliminar nodos en el árbol de elementos) y luego realizar la conversión para ver que estos elementos no son sobrescritos con los del archivo original. Esto simula en parte lo que se realizará a futuro con la herramienta que realice la adaptación del proceso a un contexto determinado.

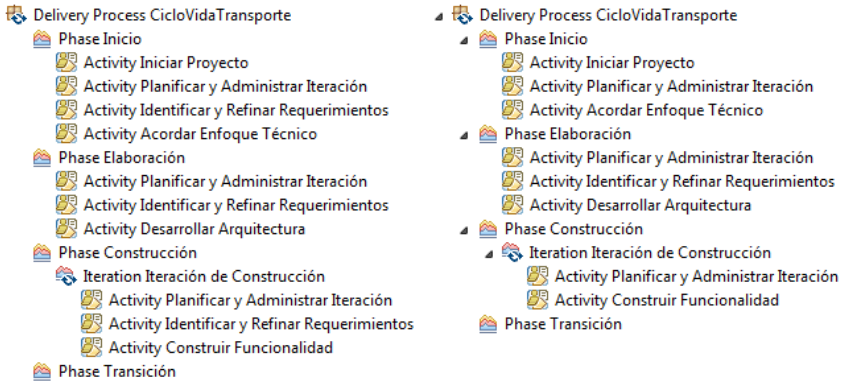


Figura 27: Eliminar actividades en SPEM, el antes y el después

Primera prueba

La primera prueba consiste en quitar actividades tanto de las fases como de la iteración, como se puede ver en la figura 27. Acá se ha eliminado "Identificar y Refinar Requerimientos" en la fase "Inicio" y en la iteración "Iteración de Construcción". Luego de realizado esto, se procede a utilizar el parser para obtener el archivo XML que, se supone, no debería contener ninguna de estas

actividades eliminadas anteriormente. Al abrir el proceso con EPF se obtiene el resultado de la figura 28.

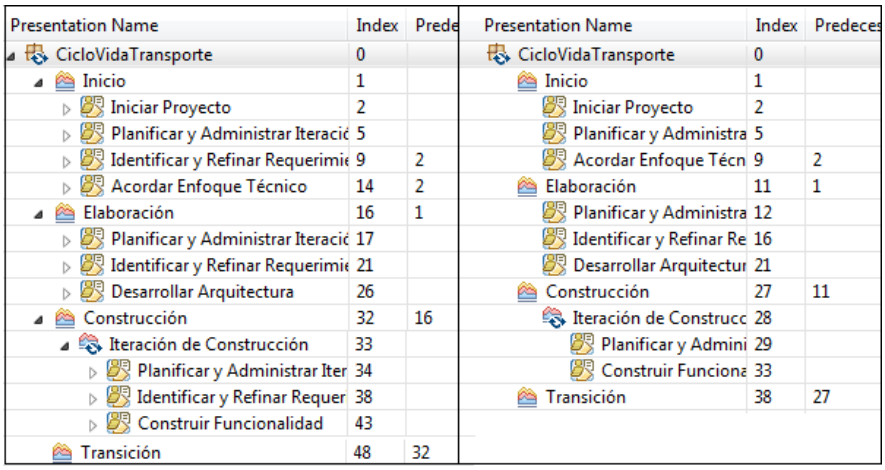


Figura 28: Visualización de las actividades eliminadas en EPF, el antes y el después

Segunda prueba

Luego de ver que esta prueba se supera correctamente se procede a eliminar más elementos, que eventualmente pueden ser eliminados en una adaptación. La figura 29 muestra el antes y el después de otra sección del proceso en EMT que fue recortada.

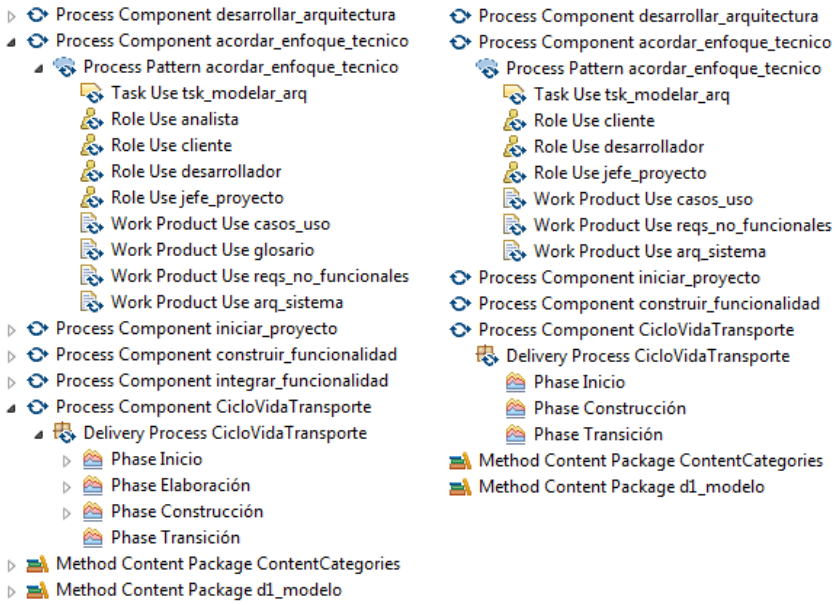


Figura 29: Se eliminan otros elementos

Los elementos eliminados son los siguientes: al patrón de proceso “acorar_enfoque_tecnico” se le elimina el rol “analista” y el producto de negocio “glosario”; se elimina la componente de proceso

"integrar_funcionalidad" y, finalmente, al proceso de entrega "CicloVida Transporte" se elimina la fase "Elaboración". En la figura 30 se observa el resultado de dicha edición. Se observa que no hay problemas en desplegar los elementos restantes.

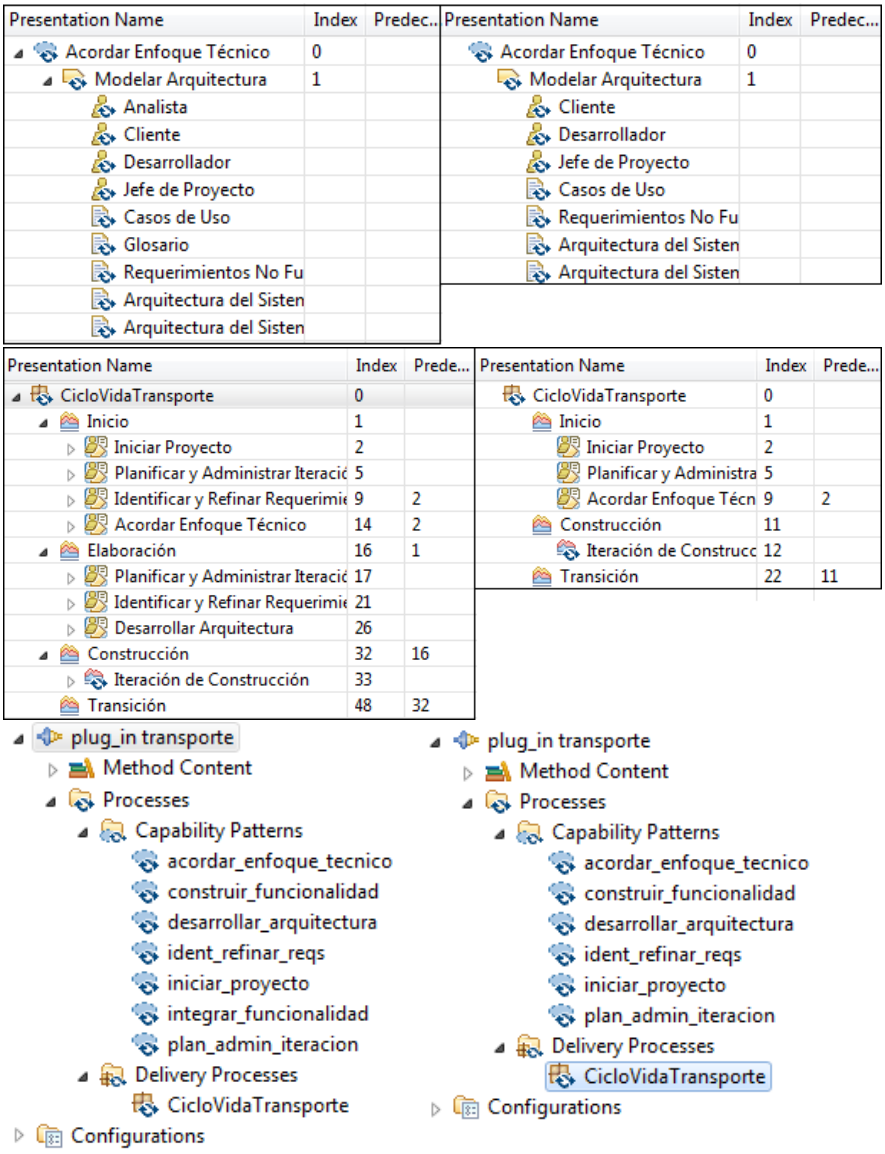


Figura 30: Visualización del resultado en EPF Composer

Finalmente se hace una comparación entre los gráficos obtenidos del *DeliveryProcess CicloVida Transporte* y la *Phase Inicio*, esto se puede observar en las figuras 31 y 32. Como se observa en ambas figuras, al realizar el parser sin eliminar ningún elemento, se obtiene el mismo diagrama que se tenía originalmente (comparar subfiguras **a** y **b**); al realizar el primer experimento, que eliminó, entre otras cosas, la actividad "Identificar y Refinar Requerimientos" se observa en la figura 32, subfigura **c**, que representa la *Phase Inicio* (fase que contenía el elemento eliminado), que se mantiene la referencia de los demás elementos (que no fueron alterados) dejando dos nodos conectados (de igual manera en que lo hacían inicialmente) y un nodo aislado. La otra figura no se

ve alterada con esta prueba, pero si lo hace con la segunda prueba, que elimina, entre otras cosas, el proceso “Elaboración”. En este caso se mantiene la relación entre los procesos “Construcción” → “Transición” y queda aislado el proceso “Inicio”.

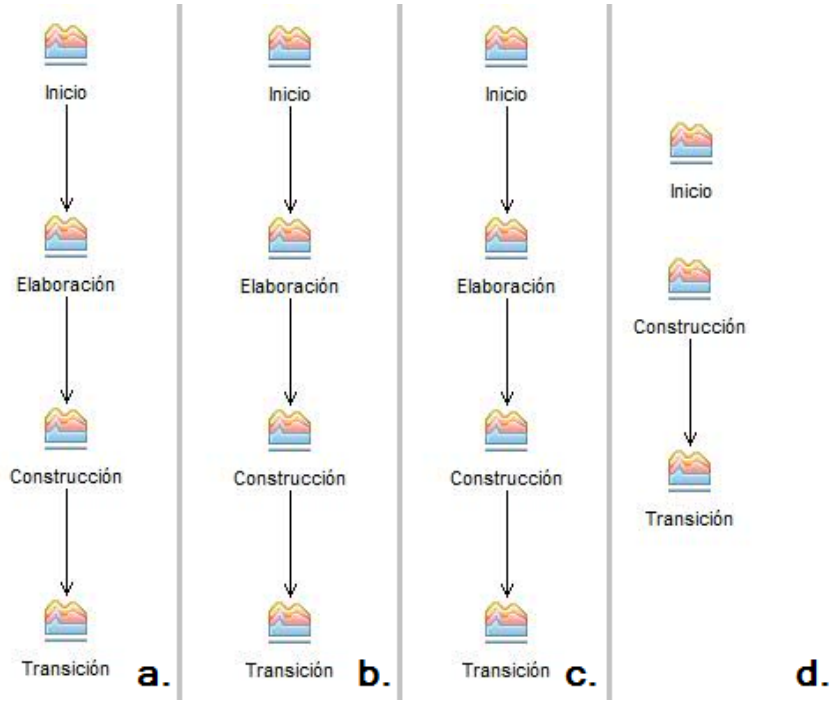


Figura 31: DeliveryProcess CicloVida Transporte: a. Original, b. Parseado, c. Primera prueba, d. Segunda prueba

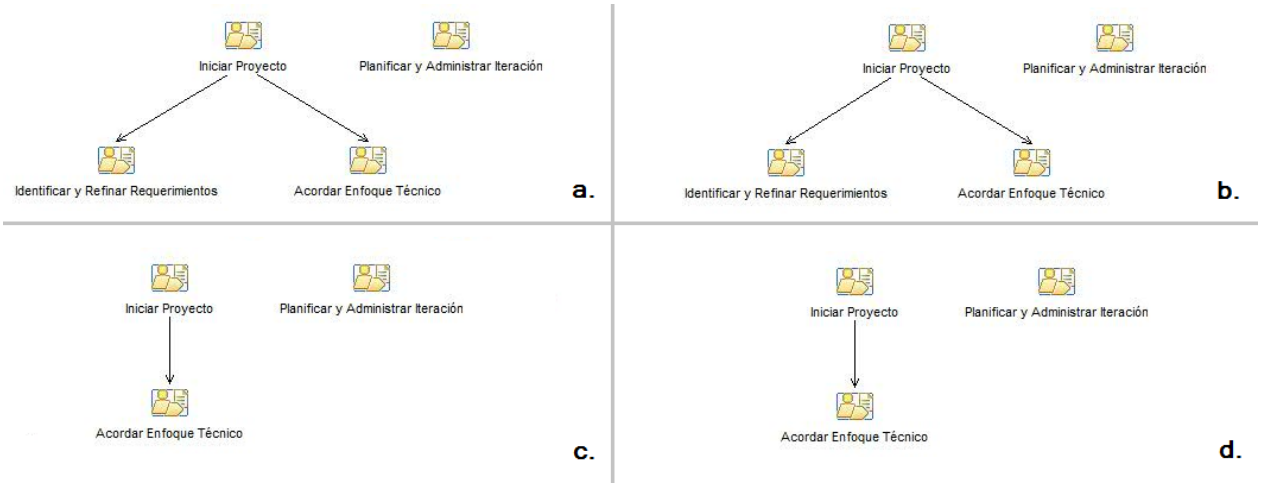


Figura 32: Phase Inicio: a. Original, b. Parseado, c. Primera prueba, d. Segunda prueba

7.4. Experimentación en TCS

La experimentación en TCS se llevó a cabo con los tres plugin usados en la experimentación de Java, lo que se realizó es simplemente abrir los archivos con TCS y observar si el editor reconoce los keywords, Strings y estructura en general. Los resultados de DTS están en la figura 33, los demás se pueden observar en el anexo C.

```
<?xml version="1.0" encoding="UTF-8"?>
<methodPlugin:MethodLibrary xmlns:methodPlugin="http://spem/1.0/MethodPlugin" xmlns:methodContent="http://spem/1.0/MethodCo
  <predefinedConfiguration defaultView="_HQFIcBfzEeCCUotUHcljeg" description="Configuración Estándar con visibilidad comple
  <ownedMethodPlugin description="plug_in transporte" name="plug_in transporte" xmi:id="_vNs9UAh3EeCMGbzZaZtzAg">
    <ownedProcessPackage description="" name="plan_admin_iteracion" xmi:id="_lJJRoAlHEeCu0q9REDKGUw" xsi:type="methodPlugin
      <processElements description="Iniciar la iteración y asignar los miembros del equipo a las tareas de desarrollo. Supe
        <nestedElements description="" linkTask="_aOcO4AlYEeCu0q9REDKGUw" name="tsk_plan_iteracion" xmi:id="_6OuHABdaEeCcpO
        <nestedElements description="" linkRole="_S37uoAlIEeCu0q9REDKGUw" name="jefe_proyecto" xmi:id="_6OuHARdaEeCcpO1e6Rj
        <nestedElements description="" linkRole="_pJzkYAh7EeCMGbzZaZtzAg" name="desarrollador" xmi:id="_6OuHAhdaEeCcpO1e6Rj
        <nestedElements description="" linkRole="_FbY1kAh6EeCMGbzZaZtzAg" name="analista" xmi:id="_6OuHAxdaEeCcpO1e6RjQHQ"
        <nestedElements description="" linkRole="_q1h70Ah6EeCMGbzZaZtzAg" name="cliente" xmi:id="_6OuHBBdaEeCcpO1e6RjQHQ" x
        <nestedElements description="" linkRole="_P7nrYAiAEeCMGbzZaZtzAg" name="jefe_testing" xmi:id="_6OuHBRdaEeCcpO1e6RjQ
        <nestedElements description="" linkRole="_MwWpQAiAEeCMGbzZaZtzAg" name="tester" xmi:id="_6OuHBhdaEeCcpO1e6RjQHQ" xs
        <nestedElements description="" linkRole="_nhvIcAh9EeCMGbzZaZtzAg" name="analista_qa" xmi:id="_6OuHBxdaEeCcpO1e6RjQH
        <nestedElements description="" linkTask="_ApHXkBdbEeCcpO1e6RjQHQ" name="tsk_cheq_estado_iteracion" xmi:id="_GCV3UBd
        <nestedElements description="" linkWorkProduct="_X0g1QBddEeCcpO1e6RjQHQ" name="backlog_producto" xmi:id="_GCV3URdhE
        <nestedElements description="" linkWorkProduct="_ylhvgAlNEeCu0q9REDKGUw" name="gantt_macro" xmi:id="_GCV3UhdhEeCcpO
        <nestedElements description="" linkWorkProduct="_4vhV8Ak0EeCu0q9REDKGUw" name="lista_riesgos" xmi:id="_GCV3UxdhEeCc
        <nestedElements description="" linkWorkProduct="_63VMoAlYEeCu0q9REDKGUw" name="plan_iteracion" xmi:id="_GCV3VBdhEeC
        <nestedElements description="" linkWorkProduct="_JimhgAlJEeCu0q9REDKGUw" name="plan_proyecto" xmi:id="_GCV3VRdhEeCc
        <nestedElements description="" linkTask="_NiAG0BdhEeCcpO1e6RjQHQ" name="tsk_evaluar_resultado_iteracion" xmi:id="_u
        <nestedElements description="" linkWorkProduct="_zdTp8Ah6EeCMGbzZaZtzAg" name="casos_uso" xmi:id="_uhuQMBdhEeCcpO1e
        <nestedElements description="" linkWorkProduct="_XHIvcAlHEeCu0q9REDKGUw" name="reqs_no_funcionales" xmi:id="_uhuQMR
      </processElements>
    </ownedProcessPackage>
    <ownedProcessPackage description="" name="ident_refinar_reqs" xmi:id="_VX4BcBdiEeCcpO1e6RjQHQ" xsi:type="methodPlugin:P
      <processElements description="" name="ident_refinar_reqs" useKind="na" variabilityType="na" xmi:id="_VX4BcRdiEeCcpO1e
        <nestedElements description="" linkTask="_UnB_oAlGEeCu0q9REDKGUw" name="tsk_detallar_cu" xmi:id="_TLLD4BdjEeCcpO1e6
        <nestedElements description="" linkTask="_P3xbMAlGEeCu0q9REDKGUw" name="tsk_detallar_reqs_no_funcionales" xmi:id="_
```

Figura 33: Vista de DTS en SPEM con TCS

Como se ve en las imágenes, se observa que los keywords son identificados adecuadamente. Además se identifican los Strings correctamente y la sintaxis está correctamente definida. El único problema en los tres plugins es el que se aprecia en la figura 34, aquí se puede apreciar que aparentemente el editor de textos no reconoce una cadena de caracteres que sí está presente.

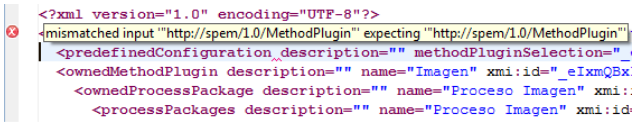


Figura 34: Problema al comprobar la sintaxis en TCS

8. Discusión

En los resultados se muestra claramente que la solución más completa es la que utilizó Java para su desarrollo. Si bien esto se debe en gran medida a la limitante de tiempo que se tiene para la realización de este trabajo, otro gran impedimento para realizar una mejor solución en TCS se debe a la poca documentación que existe al respecto. Salvo el documento citado en [15], no existe más documentación que describa los elementos del lenguaje y su sintaxis. Esto último representó una gran barrera a la hora de realizar una solución correcta. Los principales problemas que tiene esta solución son:

1. Se modificó el metametamodelo KM3 para ambos metamodelos al no poder referenciar los elementos por su ID. Es decir, el KM3 considera implícito en la definición la existencia de un elemento identificador, por lo que no es necesario definirlo explícitamente. Esto al parecer no se adoptó para la definición del KM3 para un lenguaje con sintaxis concreta, por lo que hubo que agregar, en ambos metamodelos un atributo id.

2. También se modificó el metametamodelo KM3 para ambos metamodelos para agregarle clases que son usadas en contextos distintos dentro del documento, es así que aparece la clase `ActivityBE` en UMA que es una extensión de `Activity`, lo mismo con `ActivityNE` en SPEM (ambos ejemplos se pueden ver en los anexos A y B) debido a que, como se dijo, ambos aparecen en nodos de distinto nombre, nombre que en TCS pasa a ser un keyword de la clase. No se observó una forma de agregar dos posibles valores a un keyword en TCS, por lo que hubo que crear dos clases que permitan usar un keyword distinto.

3. La última, y más invasiva modificación al metametamodelo fue la inclusión de elementos ID en UMA, esto porque dichos elementos tienen la forma `<N>id</N>`, lo que no se puede deducir directamente como una referencia (existe un método en el lenguaje que se llama `referTo` que permite referenciar a otro elemento por su IDENTIFIER (que en este caso sería el id)), este permite además agregar un delimeter, que en este caso sería de la forma `</N><N>` lo cual impediría que hubiesen saltos de línea entre cada elemento y dejarían siempre un nodo abierto.

Estos problemas sólo se percibieron a la hora de diseñar la solución en ATL, que al realizar las transformaciones, va chequeando los identificadores que se pasan a través de las clases de cada modelo. Además el conocimiento que hay que poseer sobre dicho lenguaje es demasiado profundo, siendo este el principal motivo por el que se decidió darle prioridad a la solución en Java por sobre

ésta.

Si bien el motivo anterior hizo tomar la decisión de seguir con Java, a posteriori se observó que para realizar la conversión de SPEM a UMA era necesario obtener la información que no es capturada por SPEM, debido a que sin ésta no se logra una visualización correcta del proceso en EPF Composer, tema no trivial a la hora de realizar transformaciones con ATL y que, si bien es posible hacer, aumenta en gran medida su dificultad. Una vez explicado por qué se decidió seguir con la solución en Java, se procede a realizar la discusión sobre los resultados obtenidos en Java.

Primero que todo, se procede a discutir los resultados obtenidos en la conversión UMA a SPEM, lo más importante a destacar está en la visualización de los objetos. Debido a que EPF Composer es una herramienta que funciona exclusivamente con el metamodelo UMA, un lenguaje acotado que define el problema que ahí se trata. Este posee, además de la visualización de elementos de la jerarquía, una serie de funcionalidades que no necesariamente respetan la jerarquía ahí expuesta. Uno de los ejemplos más notables, que incluso indujeron a error durante el desarrollo es el atributo *variabilityType*, el cual actúa en conjunto al *variabilityBasedOnElement* que referencia otro elemento en el proceso. La aplicación más vista con este par de atributos, es cuando el `variabilityType = ‘‘extends’’`; esto en EPF indica que el elemento extiende otro proceso, por tanto, trae todos los elementos que están en el proceso padre. En cambio en EMT, al ser una herramienta de modelamiento, observa los caracteres especiales de un documento a partir de su metametamodelo (archivo *.ecore*), por tanto los atributos los muestra en un listado de atributos, fue así como se pensó en algún momento que la conversión era incompleta.

Debido a que EMT solo muestra la relación de los nodos dentro de la jerarquía XML, la representación del archivo convertido no implicó mucha dificultad con respecto a la conversión inversa, se puede observar en los resultados que la conversión de elementos UMA a SPEM incluye todos los elementos que se usará para la transformación posterior. Además el “descubrimiento” de la utilidad en el uso de *variabilityType*/*variabilityBasedOnElement* ayudó a perfeccionar la transformación en sí, debido a que esta no consideró en una primera instancia este atributo como valioso para el problema.

La conversión realizada en SPEM a UMA se realizó, en una primera instancia, sin considerar el archivo original, de manera que sólo se obtenga la información de SPEM en UMA nuevamente. Si bien, se obtuvo un resultado no esperado, al menos lograba visualizar de manera adecuada los procesos y patrones. Sin embargo, al querer observar el contenido de cada uno de estos, no se observaban las tareas, roles ni productos de negocio asociados; siendo que en SPEM sí se obtenían estas referencias. Es por esto, que se consideró como de suma importancia la inclusión del archivo original. Realizar la conversión con el archivo original se planteó, a la hora de diseñar la solución, como un aspecto deseable de la aplicación al saber el tamaño de cada metamodelo (i.e. la cantidad

de información perdida en la conversión), sin embargo, no se valoró esta información al punto de verlo como una obligación para realizar la conversión inversa. Debido a que un proceso, patrón o actividad, sin las tareas ni el resto de la información que lo describe, no es aceptable como solución al problema que se intenta resolver. De hecho, visualizar solo patrones y procesos, no permite observar los cambios y optimizaciones realizadas por ADAPTE.

Una vez que se decidió tomar el archivo original para realizar la conversión, se dio tanto valor a los elementos que se traspasan a SPEM como a los que no, debido a que si un elemento o atributo no es cargado por la conversión original, se desea obtenerlos en su totalidad; por otro lado, si están presentes en SPEM, se deben leer y cargar recursivamente con la información que no es contenida. Los resultados obtenidos de este proceso fueron más que satisfactorios, los resultados reflejan primero que todo, el gran avance de esta conversión con respecto al intento inicial. Es así como ahora, además de observar adecuadamente los procesos y patrones, también se observa cada actividad que conforma un proceso, patrón, fase o iteración; sus tareas respectivas, y estas, a su vez, los roles y productos de negocio involucrados. Estos a su vez contenían las referencias adecuadas sobre predecesores y sucesores. Se observó además que en los contenidos, también hubo un cambio significativo, debido a que originalmente se tenían solo descripciones de tareas, productos de negocio y roles, ahora se cuenta con una biblioteca de contenido muy superior, y más cercana a la definida originalmente por el usuario en el proceso.

Problema aparte dentro de esta conversión fue el manejo de diagramas. En un inicio se planteó la necesidad de alterar los diagramas, pues se consideraba que EPF Composer, al exportar el proceso a XML, guardaba dentro de éste la información relativa a cada uno de los diagramas que aparecían definidos. De hecho, se leyó con suma importancia los elementos *DiagramElement*, *GraphElement*, entre otros, que, a priori realizaban la representación gráfica del proceso en los diagramas. Se creía además que, el proceso de integración del archivo original con el adaptado, debía considerar la problemática de mostrar solo los elementos que permanecían en el proceso (debido a que durante la transformación se pueden eliminar desde procesos enteros, hasta tareas o productos de negocio), por lo que se consideró un problema que, por su complejidad, no se trataría en este documento. Pero durante el desarrollo de esta solución se observó que, primero que todo, el archivo XML que exporta EPF Composer, no contiene en su estructura los elementos gráficos desplegados en diagramas, sin embargo, la exportación en una carpeta, si contiene documentos *diagram.xmi* ubicados en la jerarquía de clases que se representa como una jerarquía de directorios de un sistema de archivos (es así como el diagrama correspondiente al patrón *CapabilityPattern Inicio* se aloja en */capabilitypattern/inicio/diagram.xmi*). Además, estos archivos *diagram.xmi* contienen información en lenguaje UML[6]. Debido a que hacer un parser UML escapa completamente de los alcances del trabajo, se pensó no realizar este tema y dejarlo para un trabajo futuro, pero, sorprendentemente, al realizar

[6]Unified Modeling Language, para más información véase [12]

la conversión con el archivo original, e intentar observar el diagrama correspondiente a un proceso, patrón, iteración o fase, se generó automáticamente, al no encontrar el archivo, un diagrama con la información contenida en el objeto correspondiente. Por lo que con esto se obtiene una solución más que satisfactoria al respecto. Sin embargo, a esta información, que se considera la información del proceso en sí, se le puede agregar flechas o labels que si están descritas únicamente en el documento con estructura UML, pero aún así se considera una solución aceptable. De todas maneras se puede observar una mención a la necesidad de parseo del documento *diagram.xmi* en los trabajos futuros (véase la sección 9.1).

9. Conclusión

Inicialmente se plantearon dos posibles soluciones para resolver el problema de realizar la conversión entre distintos metamodelos. En las secciones anteriores se describió la metodología y el desarrollo de cada una de ellas.

La solución en Java propone utilizar toda la robustez de un lenguaje de propósito general, ampliamente utilizado, y con una gran cantidad de librerías a disposición para facilitar el desarrollo de aplicaciones. Lo cual permite obtener una solución bastante completa, que, según se puede observar en los resultados, permite realizar las conversiones propuestas para los tres casos de prueba con los que se cuenta en la actualidad. Además permite rescatar la metadata que no es soportada por SPEM, pero que sí es de utilidad para observar el proceso en el EPF Composer. Es por esto que, en líneas generales se satisface la problemática descrita dentro del proyecto ADAPTE (ver la sección 2.1.4) y que representa, a su vez, el objetivo principal del trabajo (ver sección 2.2.1).

La otra solución propuesta en TCS posee un poco más de complejidad al ser una tecnología en desarrollo, en la actualidad no cuenta con un gran soporte al desarrollador (en la actualidad no cuenta con una API establecida) y no es directa su integración con la otra herramienta del Eclipse Modeling Framework, ATL, la cual permite realizar la transformación entre modelos y metamodelos. Por tanto, si bien esta aplicación representó un desafío, no logró satisfacer las condiciones mínimas para representar una solución a la problemática planteada.

Respecto a los objetivos secundarios, planteados en la sección 2.2.2 se puede decir lo siguiente:

OBJ1. Construir un parser de XML a XMI.
Se logró construir un parser en Java que, dado cualquier archivo XML con metamodelo UMA, obtenga un archivo XMI con la estructura y clases del metamodelo SPEM.

OBJ2. Lograr que un proceso generado en EPF Composer sea parseado a XMI y abierto con Eclipse Modeling Framework.
Este proceso es exportado a XML y, con el uso del parser construido en Java, se transforma en un archivo XMI con estructura SPEM que es visto adecuadamente en el Eclipse Modeling Framework.

OBJ3. Lograr que un proceso adaptado a un contexto con Eclipse Modeling Framework sea parseado a XML y visualizado en EPF Composer (que no haya problemas de compatibilidad con la aplicación original)
Además de agregarle la funcionalidad de obtener un parser XML a XMI que transforme un elemento UMA a uno SPEM, se dotó a la aplicación con la funcionalidad necesaria para realizar el proceso inverso de XMI a XML. Al no obtener el resultado esperado con la conversión textual

del documento, se consideró como meta óptima el integrar a este archivo XML "reconvertido" la información que tuvo en un inicio y que no es considerada por el metamodelo SPEM. Es decir, se reconstruyó un archivo XML a partir de: el archivo XMI que es modificado por ADAPTE con tal de recortarlo y optimizarlo, adecuándose al contexto de la empresa; y el documento XML original que contiene, además de la información que luego fue editada, información que es competente solo para la aplicación EPF Composer.

OBJ4. Comprender y manejar bien la aplicación EPF Composer que se considera un conocimiento valioso.
Antes de comenzar con el desarrollo de la aplicación, se manejó de sobre medida cada elemento del metamodelo UMA, que corresponde a una funcionalidad distinta de la herramienta EPF Composer, además, a la hora de comprobar la correctitud del resultado para el parser en Java, se manejó la aplicación para visualizar cada elemento del archivo resultado y compararlo con el inicial.

Por tanto se podría concluir que, para la solución en Java, se cumplió con los objetivos planteados inicialmente. Por lo que el parser se puede integrar al proyecto ADAPTE, permitiendo manejar de manera simple la conversión de archivos entre aplicaciones y metamodelos.

9.1. Trabajo Futuro

En esta sección se distinguen los puntos que quedaron pendientes de la solución en TCS, de los puntos que se consideran mejoras al sistema actual en Java.

Parser en TCS

- Solucionar el problema en la definición en la sintaxis de UMA. A priori se sospechó que la solución para poder resolver el problema con el tamaño de la memoria era aumentando el tamaño del heap al momento de ejecutar Eclipse Modeling Tools, pero al parecer este problema está ligado a la configuración de TCS. De todas maneras esto no se pudo solucionar en este trabajo.

- Integrar ATL con las sintaxis concretas de SPEM y UMA. Si bien esto debiese ser directo, debido a que ambas aplicaciones están inmersas dentro del contexto del Framework AMMA, no se observa, al momento de definir un nuevo proyecto ATL, la opción de integración con sintaxis definidas en TCS.

- Hacer el ATL encargado de transformar un elemento del metamodelo UMA al metamodelo SPEM, para esto bastaría conformar el archivo *uma2spem.atl* que defina las reglas de conversión que a su vez muestran la relación entre los elementos de UMA y los de SPEM.

- Hacer el ATL encargado de transformar un elemento del metamodelo UMA al metamodelo SPEM, semejante al punto anterior, formando el archivo *spem2uma.atl* que realice la relación inversa entre elementos de SPEM y UMA.

- Realizar las pruebas de conversión en los plugins de DTS, Rhiscom e Imagen, tal como se hizo para Java.

- Hacer un ATL mucho más complejo que los anteriores, que se encargue de, tomar un elemento SPEM y otro UMA (el documento original) y retorne un documento UMA que contenga cierta información de SPEM y la información de UMA que no es modificada en SPEM, para esto se necesitaría un archivo *umaspem2uma.atl* que tome como base dos metamodelos y retorne una instancia del mismo metamodelo UMA.

Mejoras en Java

- Escribir la documentación detallada asociada a la solución, con la descripción arquitectónica y las decisiones más importantes.

- Realizar el parseo de SPEM a UMA tomando, no solo el archivo original como se hizo en este trabajo, sino que además, tomando la ruta a los diagramas que contienen metadata que no es exportada al XML por EPF Composer. Esta información que se considera como "maquillaje" para el mismo EPF Composer, no es más que diagramas UML que pueden contener, desde estados y arcos nuevos, hasta reorganización de subprocesos que si se contienen en el documento. Dichos diagramas se guardan en archivos de nombre *diagram.xmi*, y que están en la ruta respectiva al nombre del elemento, por ejemplo, el *CapabilityPattern Inicio* tendría su diagrama UML asociado en la ruta */capabilitypatterns/inicio/diagram.xmi*.

10. Referencias

[1] Proyecto ADAPTE
dirección URL: http://www.adapte.cl/

[2] A. Cockburn - Crystal Methodologies
dirección URL: http://alistair.cockburn.us/Crystal+methodologies

[3] GECHS - Quinto Diagnóstico Industria Nacional de Software y Servicios
dirección URL: http://www.gechs.cl/content/view/267573/6to-Diagnostico-de-la-Industria-de-Software-y-Servicios-de-Chile.html

[4] EPF Composer Overview, part 1
dirección URL: www.eclipse.org/epf/general/EPFComposerOverviewPart1.pdf

[5] Eclipse Modeling Framework, Home page
dirección URL: www.eclipse.org/modeling/emf/?project=emf

[6] Key Capabilities of the Unified Method Architecture (UMA), EPF Composer Home Page
dirección URL:
http://epf.eclipse.org/wikis/openupsp/base_concepts/guidances/concepts/introduction_to_uma,_94_eoO8LEdmKSqa_gSYthg.html

[7] Software & Systems Process Engineering Meta-Model Specification, Version 2.0 - Object Management Group official website.
dirección URL: http://www.omg.org/spec/SPEM/2.0/PDF

[8] J. Gosling, B. Joy, G. Steele, G. Bracha, A. Bucklay - The Java™ Language Specification, Java SE 7 Edition, Introduction.
dirección URL: http://docs.oracle.com/javase/specs/jls/se7/html/index.html

[9] Java Runtime Environment, descargar
dirección URL: http://www.java.com/es/download/

[10] Java, descargar desde la página de Oracle
dirección URL: http://www.oracle.com/technetwork/java/javase/downloads/index.html

[11] MetaObject Facility description - Object Management Group official website.
dirección URL http://www.omg.com/mof

[12] Unified Modeling Language - Object Management Group official website.
dirección URL http://www.uml.org/

[13] A. Pillay - Notes for the Computer Science Module: Object Oriented Programming COMP200, University of KwaZulu-Natal, Durban, February 2007.

[14] J.Bézivin, I.Kurtev - Model-based Technology Integration with the Technical Space Concept.

[15] F.Jouault, J.Bezivin, I. Kurtev - TCS: A DSL Specification of Textual Concrete Syntaxes in Model Engineering (Preliminar Versión) - ATLAS Team.

[16] M.Scott - Programming Language Pragmatics(2006). San Francisco, California - Morgan Kaufmann Publishers.

[17] LINA & INRIA - ATL Starter's Guide, version 0.1. December 2005 - ATLAS Team.

[18] LINA & INRIA - ATL User Manual, version 0.7. February 2006 - ATLAS Team.

A. Anexo: Templates de TCS para Actividades en UMA

```
template Activity :
"<ProcessElement"
"xsi:type" "=" "\"uma:Activity\"" "id" "=" id{as = stringSymbol}
"name" "=" name{as = stringSymbol} "guid" "=" guid{as = stringSymbol}
"presentationName" "=" presentationName{as = stringSymbol}
"briefDescription" "=" briefDescription{as = stringSymbol}
"suppressed" "=" suppressed{as = stringSymbol}
"orderingGuide" "=" orderingGuide{as = stringSymbol}
"prefix" "=" prefix{as = stringSymbol}
"isPlanned" "=" isPlanned{as = stringSymbol}
"hasMultipleOccurrences" "=" hasMultipleOccurrences{as = stringSymbol}
"isRepeatable" "=" isRepeatable{as = stringSymbol}
"isOngoing" "=" isOngoing{as = stringSymbol}
"isEventDriven" "=" isEventDriven{as = stringSymbol}
"isAbstract" "=" isAbstract{as = stringSymbol}
"shapeicon" "=" shapeicon{as = stringSymbol}
"nodeicon" "=" nodeicon{as = stringSymbol}">"
    presentation
    checklistID
    conceptID
    exampleID
    guidelineID
    reusableAssetID
    supportingMaterialID
    templateID
    reportID
    estimationConsiderationsID
    toolMentorID
    planningData
    fulfillID
    roadmapID
    precondition
    postcondition
    breakdownElement
    linkToPredecessorID
"</ProcessElement>";
```

```
template ActivityBE :
"<BreakdownElement"
"id" "=" id{as = stringSymbol} "xsi:type" "=" "\"uma:Activity\""
"name" "=" name{as = stringSymbol} "guid" "=" guid{as = stringSymbol}
"presentationName" "=" presentationName{as = stringSymbol}
"briefDescription" "=" briefDescription{as = stringSymbol}
"suppressed" "=" suppressed{as = stringSymbol}
"orderingGuide" "=" orderingGuide{as = stringSymbol}
"prefix" "=" prefix{as = stringSymbol}
"isPlanned" "=" isPlanned{as = stringSymbol}
"hasMultipleOccurrences" "=" hasMultipleOccurrences{as = stringSymbol}
"isRepeatable" "=" isRepeatable{as = stringSymbol}
"isOngoing" "=" isOngoing{as = stringSymbol}
"isEventDriven" "=" isEventDriven{as = stringSymbol}
"isAbstract" "=" isAbstract{as = stringSymbol}
"shapeicon" "=" shapeicon{as = stringSymbol}
"nodeicon" "=" nodeicon{as = stringSymbol}">"
    presentation
    checklistID
    conceptID
    exampleID
    guidelineID
    reusableAssetID
    supportingMaterialID
    templateID
    reportID
    estimationConsiderationsID
    toolMentorID
    planningData
    fulfillID
    roadmapID
    precondition
    postcondition
    breakdownElement
    linkToPredecessorID
"</BreakdownElement>";
```

B. Anexo: Templates de TCS para Actividades en SPEM

```
template Activity :
"<processElement"
"xsi:type" "=" "\"processStructure:Activity\""
"xmi:id" "=" id{as = stringSymbol}
"name" "=" name{as = stringSymbol}
"usedActivity" "=" usedActivity{as = stringSymbol}
"useKind" "=" useKind{as = stringSymbol}
"next" "=" next{as = stringSymbol}
"processPerformer" "=" processPerformerID{as = stringSymbol}
"processParameter" "=" processParameterID{as = stringSymbol}
">"
nestedElements
ownedParemeter
"</processElement>";


template ActivityNE :
"<nestedElements"
"xsi:type" "=" "\"processStructure:Activity\""
"xmi:id" "=" id{as = stringSymbol}
"name" "=" name{as = stringSymbol}
"usedActivity" "=" usedActivity{as = stringSymbol}
"useKind" "=" useKind{as = stringSymbol}
"next" "=" next{as = stringSymbol}
"processPerformer" "=" processPerformerID{as = stringSymbol}
"processParameter" "=" processParameterID{as = stringSymbol}
">"
nestedElements
ownedParemeter
"</nestedElements>";
```

C. Anexo: Vista de Plugins Imagen y Rhiscom en SPEM con TCS

C.1. Imagen

```
<?xml version="1.0" encoding="UTF-8"?>
<methodPlugin:MethodLibrary xmlns:methodPlugin="http://spem/1.0/MethodPlugin" xmlns:methodContent="http://spem/1.0
  <predefinedConfiguration description="" methodPluginSelection="_eIxmQBxkEeGnPf0pPmj_fg" name="Configuración Imag
  <ownedMethodPlugin description="" name="Imagen" xmi:id="_eIxmQBxkEeGnPf0pPmj_fg">
    <ownedProcessPackage description="" name="Proceso Imagen" xmi:id="_vYkKsBxyEeG4_tl6TkrxKQ" xsi:type="processSt
      <processPackages description="" name="Proceso Imagen" xmi:id="_z-U04BxyEeG4_tl6TkrxKQ" xsi:type="methodPlugi
        <processElements description="El Proceso Imagen, es el proceso de desarrollo de software que sigue la empr
          <nestedElements description="El principal objetivo de la fase de Inicio es conseguir un consenso entre t
            <nestedElements description="" linkTask="_lh_y0CW1EeG0lMeSYEuMEw" name="Preparar el Repositorio del Pr
            <nestedElements description="" linkTask="_BwGqEBxwEeG4_tl6TkrxKQ" name="Especificar el Alcance" xmi:id
            <nestedElements description="" linkTask="_eJEyIBxtEeG4_tl6TkrxKQ" name="Identificar Riesgos" xmi:id="_
            <nestedElements description="" linkTask="_pA2jEBxkEeGnPf0pPmj_fg" name="Realizar Reunión pre kick-off"
            <nestedElements description="" linkTask="_sQ4wcBxkEeGnPf0pPmj_fg" name="Definir Gantt" xmi:id="_NSmVgE
            <nestedElements description="" linkTask="__19uEBxuEeG4_tl6TkrxKQ" name="Definir el Plan de Aceptación"
            <nestedElements description="" linkTask="_FSXpMBxrEeG4_tl6TkrxKQ" name="Preparar de la Presentación de
            <nestedElements description="" linkTask="_JCM7gBxsEeG4_tl6TkrxKQ" name="Realizar Reunión de Kick-off"
            <nestedElements description="Esta actividad hay que hacerla en caso de clientes nuevos y de clientes c
            <nestedElements description="Esta actividad hay que hacerla en caso de clientes nuevos y de clientes c
            <nestedElements description="" linkTask="_ojTWgCZLEeGjkfKVfENWEw" name="Especificar el Ambiente de Prc
            <nestedElements description="" linkTask="_fsUuwCZLEeGjkfKVfENWEw" name="Especificar el Ambiente de Tes
            <nestedElements description="Crear la VM de desarrollo con la plataforma y las herramientas de desarro
            <nestedElements description="Es el responsable de planificar y proveer el ambiente e infraestructura c
            <nestedElements description="" linkWorkProduct="_8gCMsBxxEeG4_tl6TkrxKQ" name="Ambiente de Desarrollo"
            <nestedElements description="" linkWorkProduct="_L9bu0BxyEeG4_tl6TkrxKQ" name="Especificación de Ambie
            <nestedElements description="Es el responsable de planificar, gestionar y asignar recursos, determinar
            <nestedElements description="" linkWorkProduct="_19yTkBxkEeGnPf0pPmj_fg" name="Correciones a la Gantt"
            <nestedElements description="" linkWorkProduct="_vcSVcBxkEeGnPf0pPmj_fg" name="Gantt" xmi:id="_NSm8lBx
            <nestedElements description="" linkWorkProduct="_kZ5h4BxrEeG4_tl6TkrxKQ" name="Presentación de Kick-of
```

C.2. Rhiscom

```
<?xml version="1.0" encoding="UTF-8"?>
<methodPlugin:MethodLibrary xmlns:methodPlugin="http://spem/1.0/MethodPlugin" xmlns:methodContent="htt
  <predefinedConfiguration defaultView="_nW_ykK2FEeCUQ4-8J1_JDQ" description="" methodPluginSelection=
  <predefinedConfiguration description="" methodPluginSelection="_fHuIQKy6EeChn4F329Nsrw" name="Config
  <ownedMethodPlugin description="" name="Rhiscom V1" xmi:id="_fHuIQKy6EeChn4F329Nsrw">
    <ownedProcessPackage description="" name="Process Package Comercial" xmi:id="_i4dfoK1iEeCUQ4-8J1_J
      <processPackages description="" name="Comercial" xmi:id="_og48cKzNEeChn4F329Nsrw" xsi:type="meth
        <processElements description="" name="Comercial" useKind="na" variabilityType="na" xmi:id="_og
          <nestedElements description="" linkTask="_4ueJ4KzDEeChn4F329Nsrw" name="Analisis de factibil
          <nestedElements description="" linkRole="_MH5nQKzFEeChn4F329Nsrw" name="Gerente de desarroll
          <nestedElements description="" linkRole="_GkfCAKzFEeChn4F329Nsrw" name="Gerente de servicios
          <nestedElements description="" linkRole="_Uu0v4Ky8EeChn4F329Nsrw" name="Vendedor" xmi:id="_M
          <nestedElements description="" linkWorkProduct="_Wg2NgKzEEeChn4F329Nsrw" name="Minuta reunió
          <nestedElements description="" linkWorkProduct="_mUFw4KzREeChn4F329Nsrw" name="Minuta de fac
          <nestedElements description="" linkTask="_kzYhMKzTEeChn4F329Nsrw" name="Asignación de recurs
          <nestedElements description="" linkTask="_NWTBIKzSEeChn4F329Nsrw" name="Elaborar orden de tr
          <nestedElements description="" linkTask="_lh6fUKzMEeChn4F329Nsrw" name="Generar propuesta" x
          <nestedElements description="" linkTask="_jw95YKy8EeChn4F329Nsrw" name="Levantar deseos del
          <nestedElements description="" linkTask="_5l0yMKzPEeChn4F329Nsrw" name="Revisar propuesta co
          <nestedElements description="" linkRole="__KnG4KzDEeChn4F329Nsrw" name="Cliente" xmi:id="_N3
          <nestedElements description="" linkWorkProduct="_RHXpkKzTEeChn4F329Nsrw" name="Orden de trab
          <nestedElements description="" linkWorkProduct="_6h_m4KzMEeChn4F329Nsrw" name="Propuesta com
          <nestedElements description="" linkWorkProduct="_0Q_7MKzSEeChn4F329Nsrw" name="Orden de comp
          <nestedElements description="" linkWorkProduct="_UCOrcKzDEeChn4F329Nsrw" name="Deseos de usu
          <nestedElements description="" linkWorkProduct="_7VdR4KzREeChn4F329Nsrw" name="Minuta de ver
        </processElements>
      </processPackages>
      <processPackages description="" name="Orden de trabajo" xmi:id="_o1_OAK1iEeCUQ4-8J1_JDQ" xsi:typ
        <processElements description="" name="Orden de trabajo" useKind="na" variabilityType="na" xmi:
          <nestedElements description="" linkTask="_NWTBIKzSEeChn4F329Nsrw" name="Elaborar orden de tr
```