MECANISMO GENÉRICO PARA LA DEFINICIÓN DE TRANSFORMACIONES DE MODELOS EN EL MARCO DE UN ENFOQUE MULTIVISTAS PARA LA GENERACIÓN DE APLICACIONES CON BASE EN LA INGENIERÍA DIRIGIDA POR MODELOS

JOSÉ JULIÁN MONTOYA CORONADO
julian.montoya@gmail.com

UNIVERSIDAD DE MEDELLÍN
FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERÍA DE SOFTWARE
MEDELLÍN
2015

MECANISMO GENÉRICO PARA LA DEFINICIÓN DE TRANSFORMACIONES DE MODELOS EN EL MARCO DE UN ENFOQUE MULTIVISTAS PARA LA GENERACIÓN DE APLICACIONES CON BASE EN LA INGENIERÍA DIRIGIDA POR MODELOS

JOSÉ JULIÁN MONTOYA CORONADO
julian.montoya@gmail.com

Trabajo de grado para optar al título de Magister en Ingeniería de Software

Asesor temático: Jesús Andrés Hincapié – Magíster en Ingeniería Informática

Asesor metodológico: Bell Manrique Losada – Doctor en Ingeniería Informática

UNIVERSIDAD DE MEDELLÍN
FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERÍA DE SOFTWARE
MEDELLÍN
2015

CONTENIDO

LISTA DE ILUSTRACIONES

GLOSARIO

Modelo: es una descripción de un sistema (o parte de él) escrita en un lenguaje bien definido, que está normalmente presentado como una combinación entre dibujos y texto [1].

Metamodelo: lenguaje para especificar modelos [2].

UML (*Unified Modeling Languaje*): lenguaje de modelado unificado, utilizado para visualizar, especificar y documentar todas las fase del desarrollo de software [3].

MOF (*Meta-Object Facility*): es un meta-metamodelo creado por la OMG, el cual permite crear metamodelos [2].

EMF (*Eclipse Modeling Framework*): es un framework para el modelamiento y generación de código basado en un modelo de datos estructurado [4].

GMF (*Grafical Modeling Framework*): es un framework para la realización de editores gráficos basados en EMF [5].

Ecore: es un metamodelo para la descripción de modelos EMF que es capaz de dar soporte en tiempo de ejecución [6].

M2M (*Model to model*): transformación de modelo a modelo, es un aspecto del desarrollo de software dirigido por modelos para convertir un modelo origen a un modelo destino de acuerdo a un proceso o reglas de transformación [7].

M2T (*Model to Text*): tipo de transformación de un modelo a texto, generalmente el texto corresponde a código fuente con el propósito de generar una aplicación [8].

ITIL (*Information Technology Infrastructure Library*): conjunto de conceptos y buenas prácticas para optimizar los servicios tecnológicos [9].

MDSD (*Model Driven Software Development*): estilo de desarrollo de software que se enfoca en el refinamiento de modelos de un dominio en particular utilizado para producir software [10].

BPMN (*Business Process Management Notation*): notación estandar para modelar procesos de negocio [11].

MDE (Model Driven Engineering): disciplina de la ingeniería de software que se basa en modelos como entidades principales y tiene como objetivo desarrollar, mantener y evolucionar el software mediante la transformación de modelos [12].

XMI (*XML Metadata Interchange*): es el estándar para el transporte de modelos entre herramientas de diagramación UML [13].

XML (Extensible Markup Language): lenguaje de marcado ampliable, utilizado para representar información textual estructurada [14].

RESUMEN

La ingeniera de software se enfoca en el desarrollo de aplicaciones desde diferentes puntos de vista usando diversos enfoques, uno de ellos es el Desarrollo de Software Dirigido por Modelos (MDSD, por sus siglas en inglés); al desarrollar soluciones bajo esta propuesta se han visualizado grandes ventajas como velocidad, bajos costos [15] y calidad en los desarrollos [16], sin embargo también algunas desventajas como la dificultad de intervenir las transformaciones [17][18][19], falta expresividad en los modelos y la generación hacia múltiples plataformas [20]; este último debido a que no es posible delimitar con claridad las características de la plataforma destino al especificar los modelos y las transformaciones que constituyen el proceso de desarrollo.

Durante el progreso del presente trabajo se trata de mitigar las tres dificultades antes mencionadas por medio de la construcción de un Lenguaje de Dominio Específico [21] (DSL, por sus siglas en inglés) con toda la información funcional de la aplicación, usando diagramas de paquetes y de clases en UML [3] y diagramas de procesos de negocio en BPMN [22]. Este trabajo hace parte de la macro propuesta Metáfora donde se desarrolló un *plugin* de Eclipse que está basado en el *framework* de modelado de eclipse (EMF, por sus siglas en inglés) [5][10][4]. El *plugin* tiene las funciones de asistente guiando al usuario a través del proceso iterativo de transformaciones hasta llegar al código fuente [23].

El software que fue desarrollado para que el proceso de generación se pueda parametrizar de acuerdo a los modelos y transformaciones realizadas por el analista de desarrollo con ayuda del analista de negocio. Se tiene la total libertad para configurar las secuencias de transformación y aplicarlas en un orden determinado a un conjunto de modelos específicos con el fin de generar parte de una aplicación.

INTRODUCCIÓN

En la actualidad las empresas que desarrollan software necesitan ofrecer mejores productos y servicios que sus competidores, exige un alto grado de eficiencia y rapidez para construir las aplicaciones, este factor impacta de manera positiva las compañías donde los sistemas son base fundamental para apoyar la operación de manera transversal.

Uno de los enfoques orientado a mejorar el proceso de desarrollo es la Ingeniería Dirigida por Modelos o MDE [24] (*Model Driven Engineering*), disciplina que tiene como objetivo generar modelos como punto de partida de las soluciones y como base fundamental para la generación del software por medio de procesos automáticos utilizando transformaciones sistemáticas de los modelos hasta llegar al código fuente [12].

De acuerdo la investigación realizada en [25] se evidencian tres dificultades a la hora de utilizar la Ingeniería Dirigida por Modelos en el desarrollo de soluciones de software que da origen al macro proyecto Metáfora. Los inconvenientes principales son (i) falta de expresividad de los modelos, (ii) la generación hacia múltiples plataformas y (iii) dificultad a intervenir las transformaciones de modelos durante el proceso de desarrollo [8][26].

Para resolver los problemas antes mencionados se incluye en el proceso de desarrollo los modelos de comportamiento (Diagrama BPMN [27]), de funcionalidad y agrupación (Diagrama de Paquetes de Dominio [3]), de estructura (Diagrama de Clases) y de plataforma (DLS [24] de plataforma). De esta manera se están abarcando diversos aspectos funcionales y no funcionales que quedaran plasmados en un modelo de aplicación y uno de plataforma final necesarios para generar el código, estos modelos finales tendrán toda la información necesaria para derivar la solución de software en cualquier plataforma específica [28].

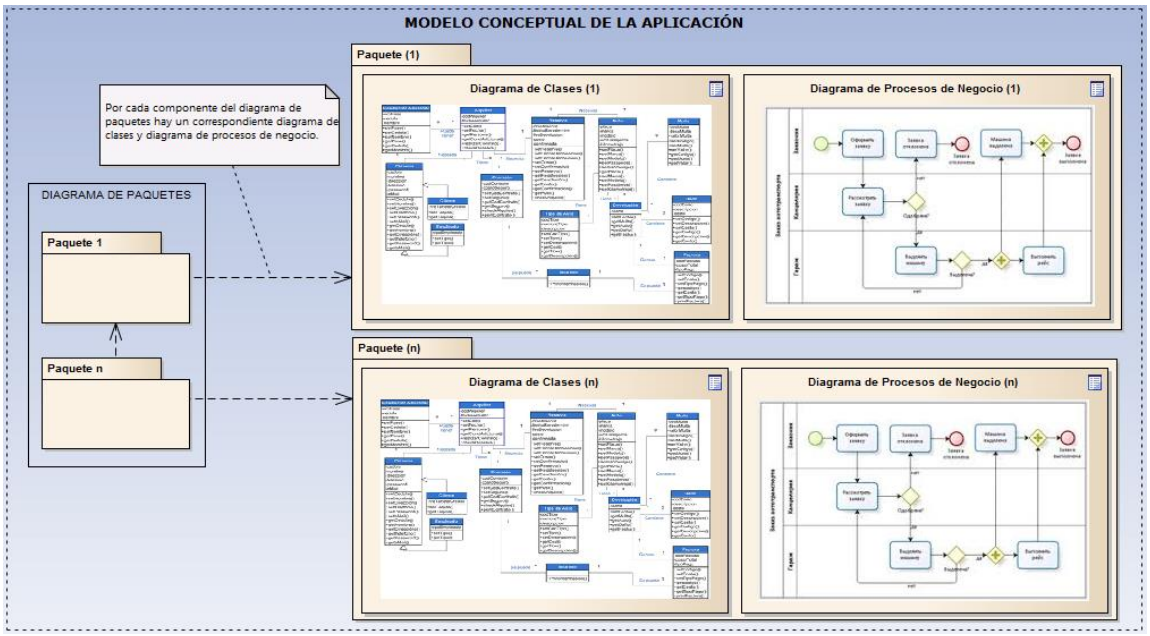


Ilustración 1: Modelo conceptual de la aplicación.

Para probar la aplicabilidad del proceso descrito en el presente trabajo, se seleccionó el caso de un sistema de información de gestión de incidentes basado en ITIL [29][9]. Dicho caso fue utilizado en el proyecto Metáfora [8], en el cual se plantea el proceso de desarrollo anteriormente descrito, en donde inicialmente se debe realizar el modelado del diagrama de paquetes de dominio, cada paquete de dominio tiene sus correspondientes diagramas de clases y procesos de negocio como se aprecia en la ilustración 1. Estos diagramas se mezclan por medio de transformaciones de modelo a modelo para dar origen al modelo conceptual de la aplicación.

Para generar el código de la aplicación, se apoya en el modelo de la plataforma, dicho modelo presenta los elementos propios del despliegue y en general los aspectos no funcionales del sistema [30]. Se debe aclarar que la creación del modelo de plataforma hace parte del proyecto macro Metáfora y se presentó como insumo inicial para el presente trabajo.

Teniendo como insumos el modelo conceptual y el modelo de plataforma se procede a realizar las transformaciones necesarias de modelo a texto dando como resultado todos los artefactos necesarios del sistema de información [12] como se indica en la ilustración 2.

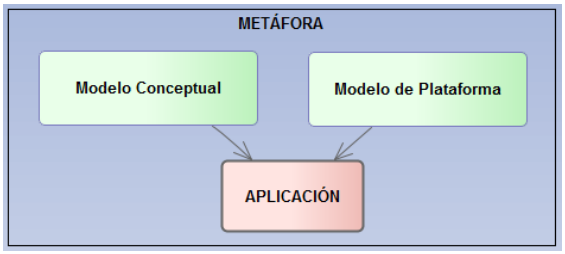


Ilustración 2: Generación de la aplicación.

Se debe proponer un esquema genérico que apoye el proceso de parametrización de las transformaciones permitiendo realizarlas de acuerdo a la configuración deseada, en donde el diseñador de la solución tenga la flexibilidad de adicionar y modificar los modelos y sus respectivas transformaciones.

Este documento contiene los siguientes capítulos donde se identifican los pasos realizados para cumplir con el trabajo propuesto. En el capítulo 1 se describe el marco del problema, los objetivos planteados y la estrategia que se va a utilizar para cumplir con dichos objetivos; en el capítulo 2 se documentan los trabajos relacionados y sus características; en el capítulo 3 se describe de manera detallada el caso de estudio utilizado para realizar el proceso de generación; el capítulo 4 tiene la descripción de los diagramas a utilizar y la forma es que dichos diagrama conforman el proceso de generación; en el capítulo 5 se describen los componentes que aporta toda la semántica necesaria para generar el sistema de información; En el capítulo 6 ya se realiza un paso a paso donde se describe la forma y orden de las actividades para generar el código fuente del sistema diseñado; En el capítulo 7 se realiza una explicación del enfoque genérico de la herramienta desarrollada para dar solución a los problemas planteados en el capítulo 1; en el capítulo 8 se describen las conclusiones encontradas y posibles mejoras a desarrollar dentro del prototipo generado.

PREGUNTA DE INVESTIGACIÓN

¿Se puede mejorar el proceso de transformación de los modelos que conforman la vista conceptual y la vista de escenarios mediante mecanismos genéricos implementados en una herramienta de software?

Frente a la pregunta de investigación, el desarrollo del caso de estudio identificó que efectivamente el uso de una herramienta que permita ordenar y automatizar los procesos de transformación mejora el proceso de desarrollo y permite una evolución más clara de los productos generados. El detalle de las características relevantes se menciona en el apartado de las conclusiones.

HIPÓTESIS

Si las empresas que desarrollan software utilizan un mecanismo genérico para la definición de transformaciones para la vista conceptual y de escenarios apoyados en una herramienta de software, el proceso de generación de modelos será más flexible y ágil, adicionando en los modelos generados información más detallada y útil, la cual es la base para las posteriores etapas del desarrollo de software.

La herramienta se desarrolló pensando en darle al usuario la posibilidad de modificar los modelos y transformaciones necesarias. Durante el proceso de desarrollo de una aplicación, el usuario tiene la opción de crear su propia secuencia de generación involucrando los modelos que se consideren convenientes de acuerdo a las características de la solución de software. El detalle se explica claramente en el apartado análisis del carácter genérico de las transformaciones y en las conclusiones.

METODOLOGIA

Para lograr el desarrollo de un mecanismo genérico se establecieron los siguientes objetivos que sirven de guía, el propósito de dichos objetivos es abarcar los aspectos relevantes de las transformaciones y cumplir con el presente trabajo.

- Realizar una revisión del estado del arte de la transformación de modelos en la ingeniería dirigida por modelos.
- Caracterizar las transformaciones de modelo a modelo en un enfoque arquitectónico multivistas en el marco de la Ingeniería Dirigida por Modelos.
- Proponer un esquema genérico de parametrización para las transformaciones de modelo a modelo.
- Construir un prototipo que implemente el mecanismo de transformación e integrarlo con un entorno de generación de aplicaciones.
- Validar el mecanismo aplicándolo a un caso de estudio específico.

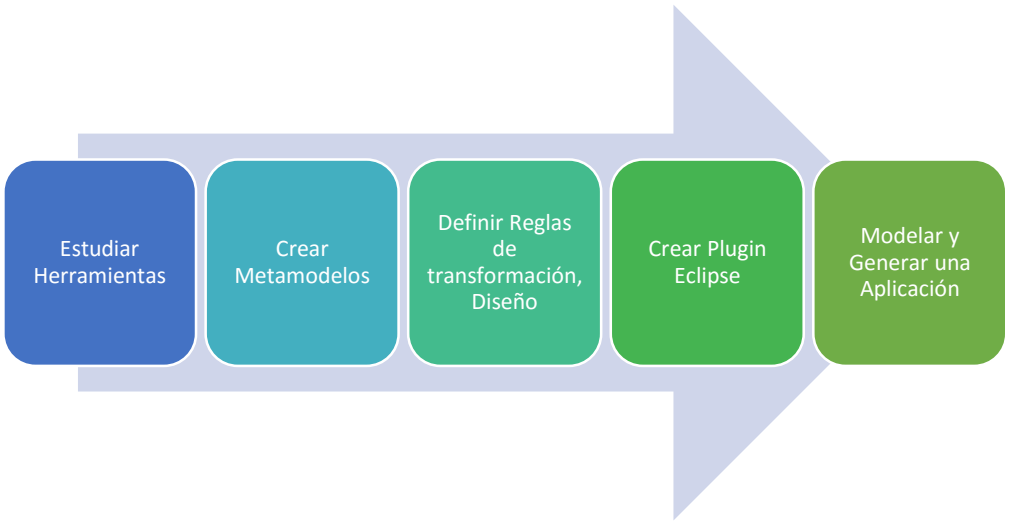


Ilustración 3: Pasos para el desarrollo del trabajo.

Para el alcance del trabajo se realizarán las siguientes actividades desde el punto de vista técnico con respecto a los objetivos planteados, como se muestra en la ilustración 3.

Objetivo 1. Realizar una revisión del estado del arte de la transformación de modelos en la ingeniería dirigida por modelos.

- Estudiar herramientas para realizar el asistente de generación de aplicaciones.

Objetivo 2. Caracterizar las transformaciones de modelo a modelo en un enfoque arquitectónico multivistas en el marco de la Ingeniería Dirigida por Modelos.

- Crear los metamodelos y perfiles [31] necesarios que apoyen el proceso de generación de la aplicación.
- Definir reglas [32] de transformación de modelo a modelo para trasladar toda la semántica entregada por los modelos de paquetes, clases y BPMN al modelo de la aplicación creado anteriormente.
- Definir las reglas de transformación de modelo a texto, donde se debe identificar que partes del modelo serán mapeadas a la plataforma especifica seleccionada.

Objetivo 3. Proponer un esquema genérico de parametrización para las transformaciones de modelo a modelo.

- Definir lo forma en la cual el prototipo ayudará a generar las aplicaciones por medio de las transformaciones desde el punto de vista genérico.

Objetivo 4. Construir un prototipo que implemente el mecanismo de transformación e integrarlo con un entorno de generación de aplicaciones.

- Crear un asistente que ayude a eliminar las tareas repetitivas y disminuya la complejidad posible durante todo el proceso de transformación.

Objetivo 5. Validar el mecanismo aplicándolo a un caso de estudio específico.

- Realizar el modelado de una aplicación y generar una aplicación totalmente funcional en una plataforma de desarrollo particular para probar el funcionamiento del trabajo realizado.

Se debe aclarar que la generación de código se apoyó en un modelo de plataforma generado desde el proyecto macro Metáfora [33], en el que se trabajó en el frente de plataforma sobre los aspectos no funcionales de la aplicación.

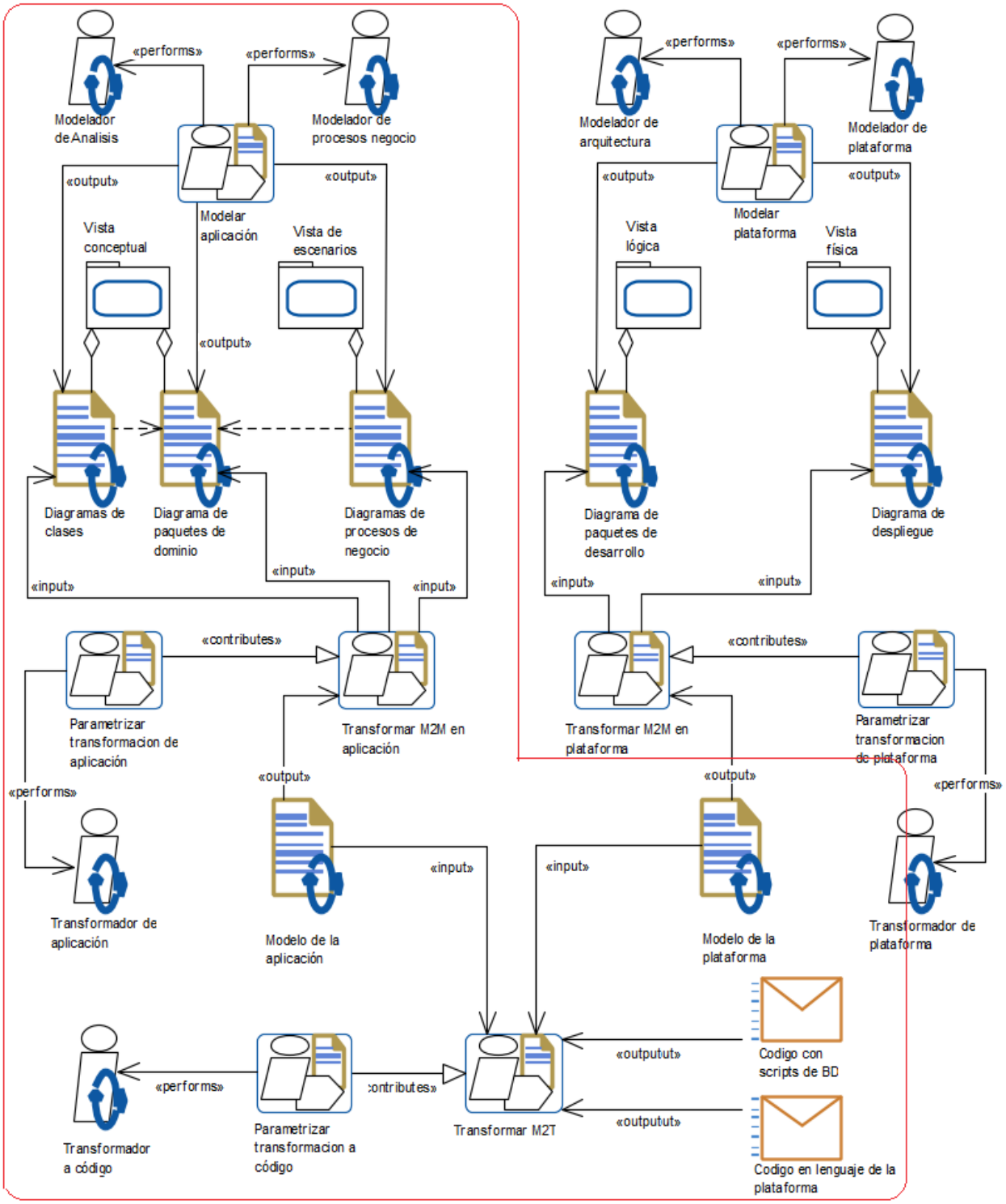


Ilustración 4: Esquema general de proyecto Metáfora.

En la ilustración 4 se puede observar las actividades que están enmarcadas dentro del proyecto macro Metáfora, la línea roja delimita las actividades que se deben realizar en este trabajo.

Inicialmente se realizan los diagramas de paquetes de dominio, diagrama de clases y diagrama de procesos de negocio. Con la información plasmada en los modelos anteriormente mencionados se realiza un proceso de validación y transformación de modelo a modelo para generar la vista conceptual de la aplicación que contiene toda la semántica proveniente de los tres tipos de diagramas.

Como un proceso en paralelo dentro del macro proyecto Metáfora se realizó la generación del metamodelo de plataforma. Para poder realizar el ciclo completo de concepción de una aplicación se generó una instancia del modelo de plataforma con toda la información no funcional necesaria para generar una aplicación.

Como proceso final con el propósito de generar una aplicación se realizó la validación y mezcla del modelo conceptual y de plataforma por medio de una transformación de modelo a texto. Se debe tener en cuenta que el usuario tiene la posibilidad de realizar modificaciones en cualquiera de los modelos que intervienen en el proceso iterativo con el fin de controlar los tipos y características de los artefactos que se generan.

1. ENFOQUE MULTIVISTAS DE METÁFORA Y MODELADO DE LA APLICACIÓN

Desde el proyecto Metáfora se establece el lineamiento de trabajar bajo el enfoque de MDSD [34] y con un enfoque multivistas. Con esta estrategia se tiene un panorama general donde se puede mitigar la falta de expresividad de los modelos debido a que cada modelo con su correspondiente semántica aporta en diferentes aspectos funcionales y no funcionales.

La estrategia inicial del proyecto para generar una aplicación utilizando MDSD es seleccionar las vistas necesarias con sus modelos correspondientes para tratar los tres problemas antes mencionados.

Se utilizan DSL para poder expresar todos los aspectos con el nivel de detalle necesario estableciendo los componentes requeridos en cada nuevo componente del Lenguaje de Dominio Específico (DSL, por sus siglas en inglés) que hace parte de la generación.

Con esta estrategia se está atacando la falta de expresividad [31] en los modelos, adicionalmente se está apoyando sobre la creación de perfiles [18] para caracterizar aún más los componentes en los modelos UML que apoyan la solución. El tener modelos o DSL’s nos ayuda a caracterizar la aplicación y mantenerla independiente de la plataforma de desarrollo [35].

Se realizó un *plugin* sobre la plataforma de desarrollo de Eclipse con el propósito de guiar el proceso de trasformación y generación de aplicativos [36][5][37]. Facilita la intervención de los modelos por medio de asistentes e interfaces de usuario amigables durante la generación de sistemática de sistemas de información.

En el proyecto Metáfora, la estrategia del desarrollo es utilizar varias vistas para representar un sistema de información. Para el alcance de este trabajo se van a utilizar las siguientes vistas para llegar al modelo de la aplicación.

- ✓ La vista conceptual, de la cual hacen parte los diagramas de clases y el diagrama de paquetes de dominio en UML.
- ✓ La vista de escenario, está representada por el diagrama de procesos de negocio en BPMN.

Como se puede ver en la ilustración 4 la sección que no está seleccionada es el modelo de la plataforma que corresponde a aspectos no funcionales y se limitará únicamente a utilizar dicho modelo para poder generar la aplicación final, pero la explicación de su generación hace parte de otro trabajo en el proyecto Metáfora [8].

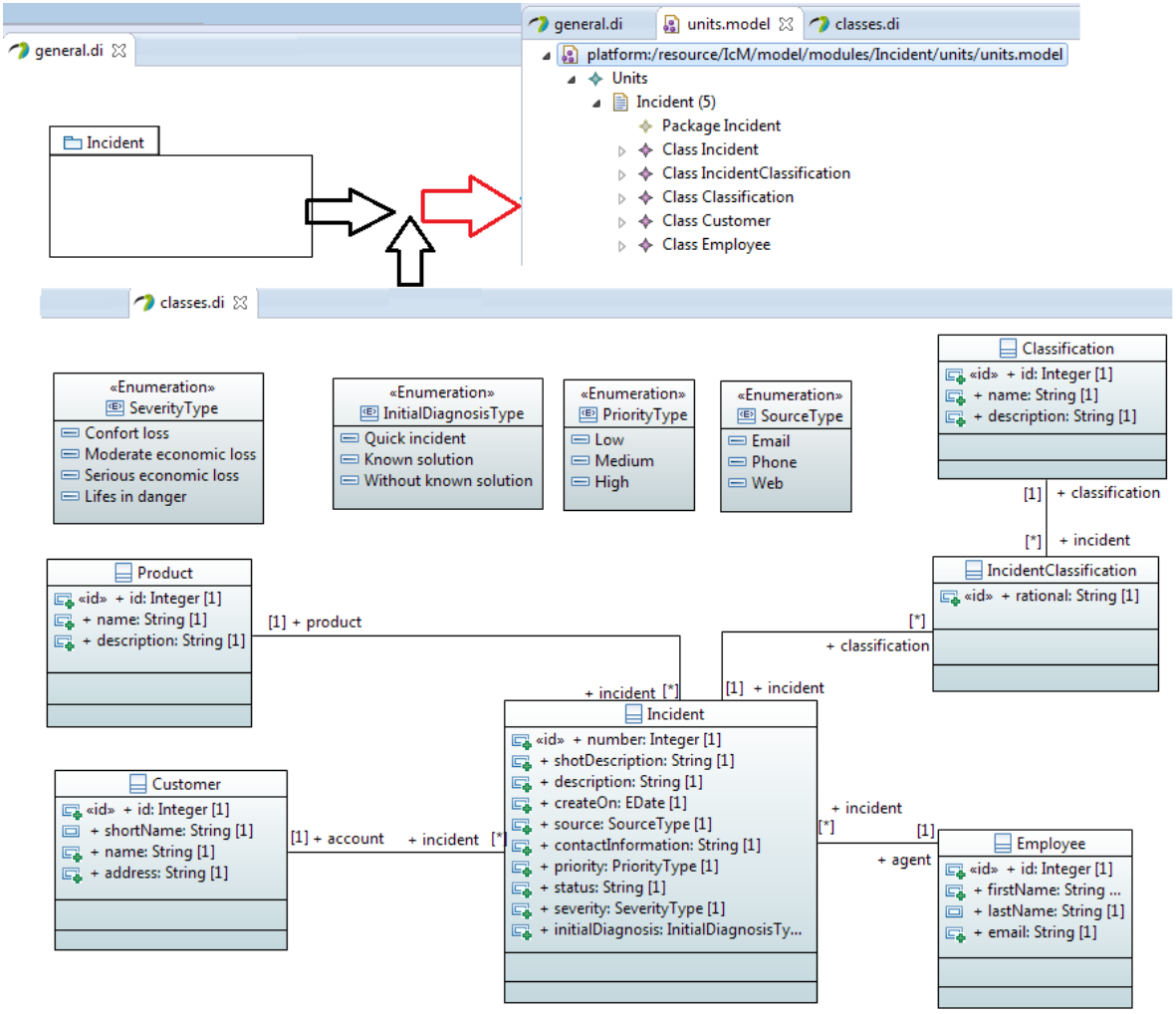


Ilustración 5: Generación modelo Units.

El proceso de transformación se divide en dos grandes bloques:

1.1. Generar Modelo de Aplicación

El proceso para generar el modelo conceptual o modelo de aplicación denominado *Web Application Metamodelo (WAMM)*, donde estará toda la semántica funcional del sistema se creará por medio de los siguientes pasos:

1. Generar el modelo de paquetes de dominio general de toda la aplicación.
2. Por cada paquete de dominio creado se debe generar el modelo de clases UML con todas las entidades necesarias y el diagrama de procesos de negocio correspondiente a la influencia funcional de dicho paquete.
3. Una vez se tengan creados todos los modelos se procede unir cada paquete de dominio con el correspondiente diagrama de clases dando lugar a un nuevo modelo que se definió con el nombre de Units, tal como se muestra en la ilustración 5. Esta mezcla se realiza de manera automática por el asistente de Eclipse bajo la opción “Generar Modelo de Aplicación” del menú contextual.
4. Se debe aclarar que se realiza un proceso de simplificado de cada diagrama BPMN a un modelo que se definió como BPMN_SIMPLE porque la herramienta ModeLink de Epsilon [38] en sus versiones Kepler y Luna que se encarga de enlazar modelos no soporta modelos BPMN2. Esta simplificación de BPMN la realiza de manera automática por el asistente de eclipse que fue desarrollado. Esta simplificación se puede observar en la ilustración 6.

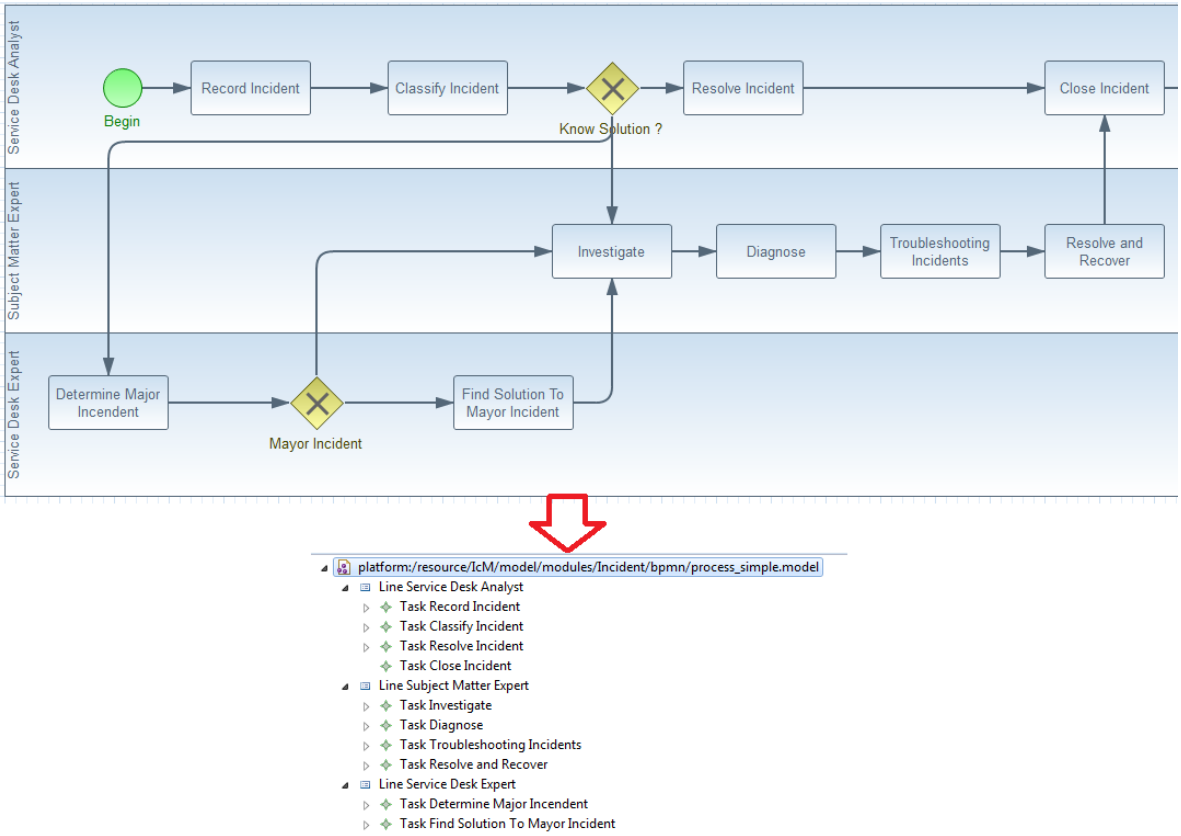


Ilustración 6: Simplificación del modelo BPMN.

5. Para poder unir el diagrama Units creado en el paso tres con el respectivo diagrama de procesos de acuerdo creado en el paso 4 se crea un modelo de enlace (weaving) [34] definido como UnitsBpmn.

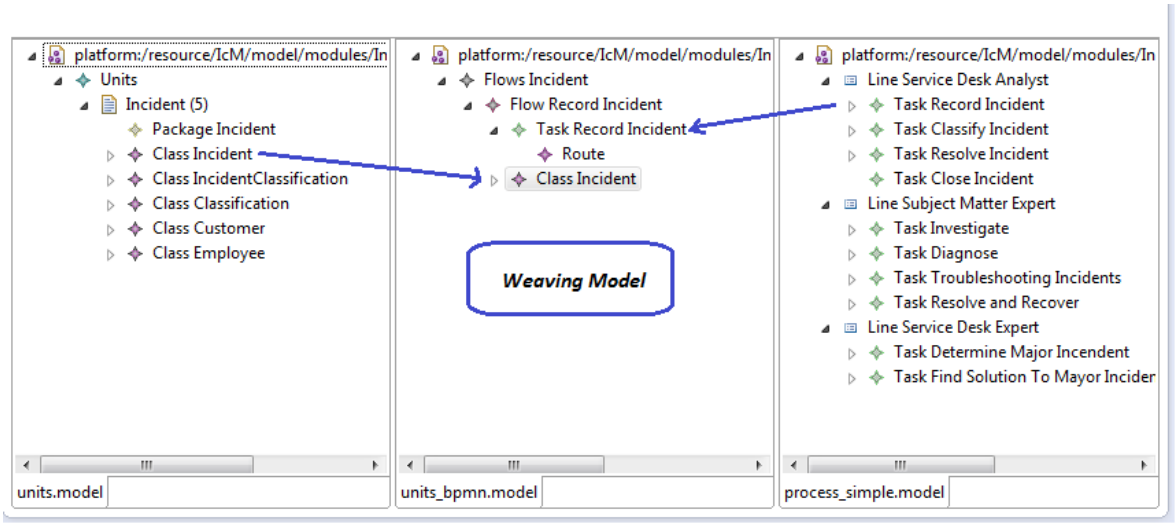


Ilustración 7: Creación del modelo UnitsBpmn.

6. Con el modelo de BPMN_SIMPLE y el modelo Units generado el usuario debe proceder a realizar el enlace entre modelos. Se debe adicionar toda la información necesaria para identificar cada uno de los flujos sus entidades correspondientes como se visualiza en la ilustración 7.

En este punto ya se tiene toda la información necesaria en todos los modelos de clases, procesos de negocio y paquete de dominio para generar finalmente el modelo de aplicación como se muestra en la ilustración 8.

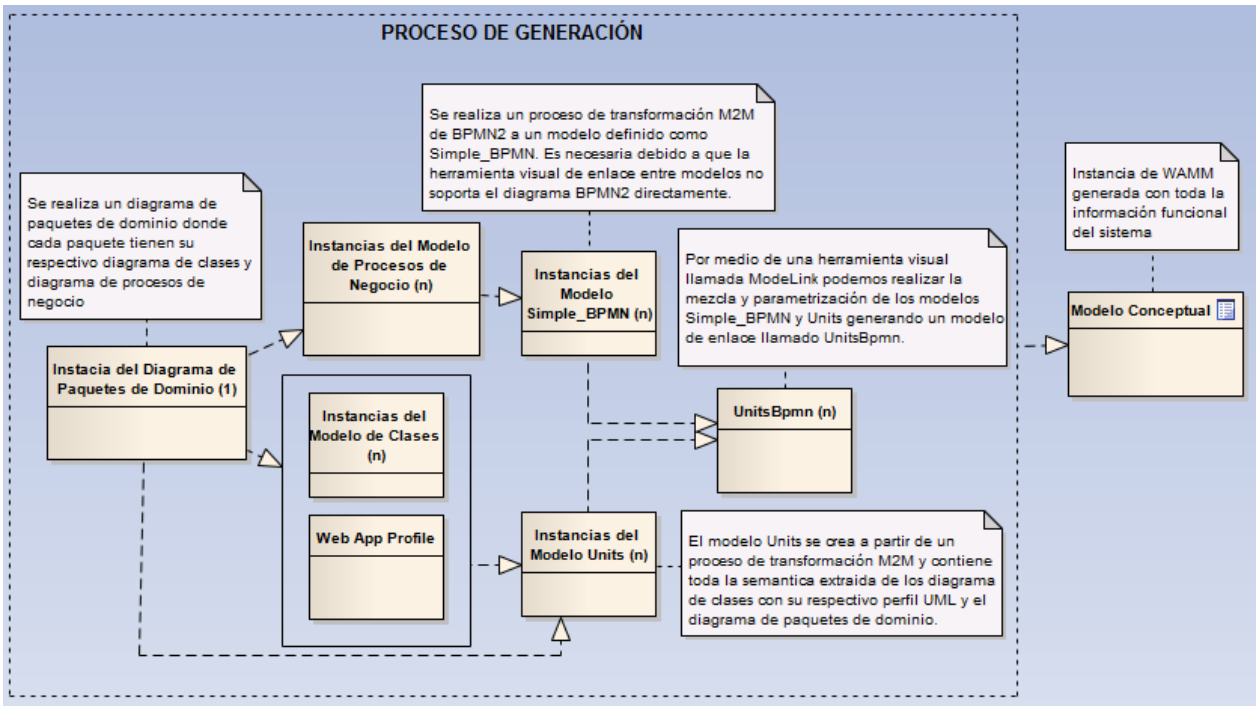


Ilustración 8: Proceso para obtener el modelo de aplicación.

1.2. Generar Modelo de Plataforma

Para poder llegar al código fuente de la aplicación se debe realizar las siguientes actividades debido a su complejidad en cuanto a cantidad y número de modelos:

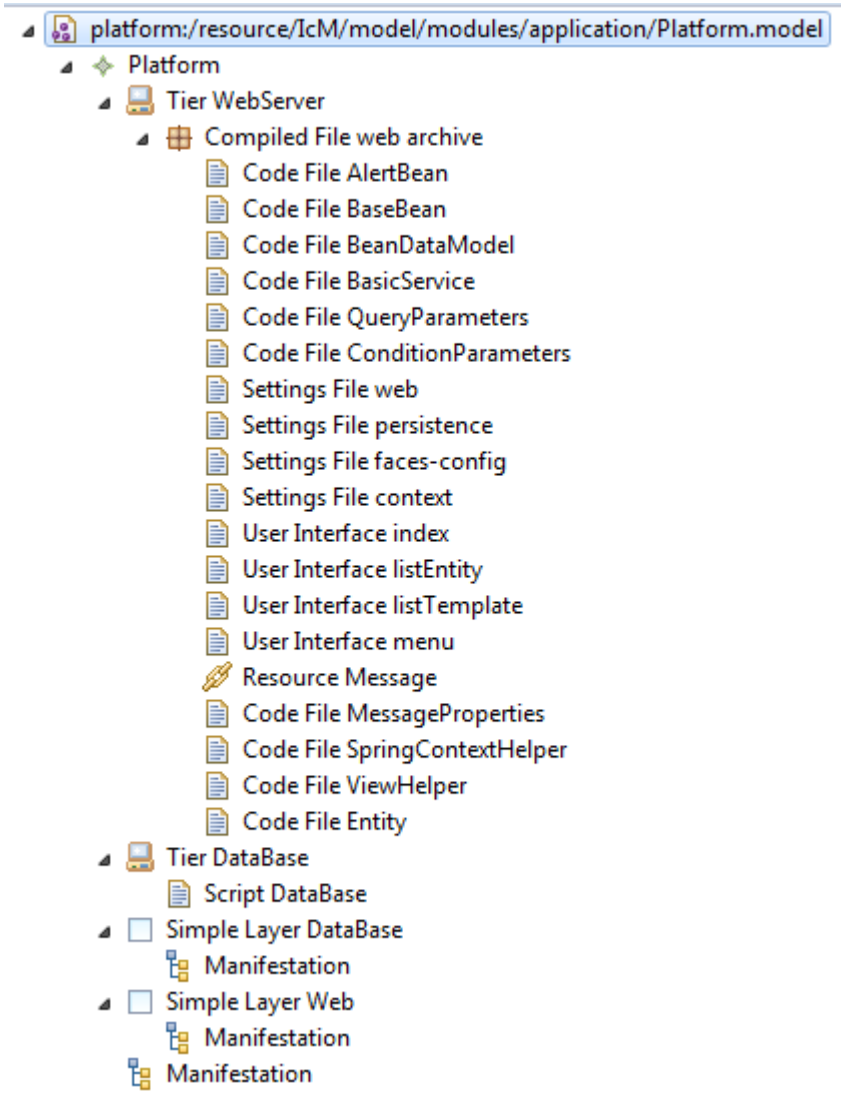


Ilustración 9: Modelo de plataforma base.

- Se debe crear un modelo de plataforma, el asistente o plugin de eclipse genera una plantilla base para aplicaciones web de la plataforma Java JEE como se muestra la ilustración 9.

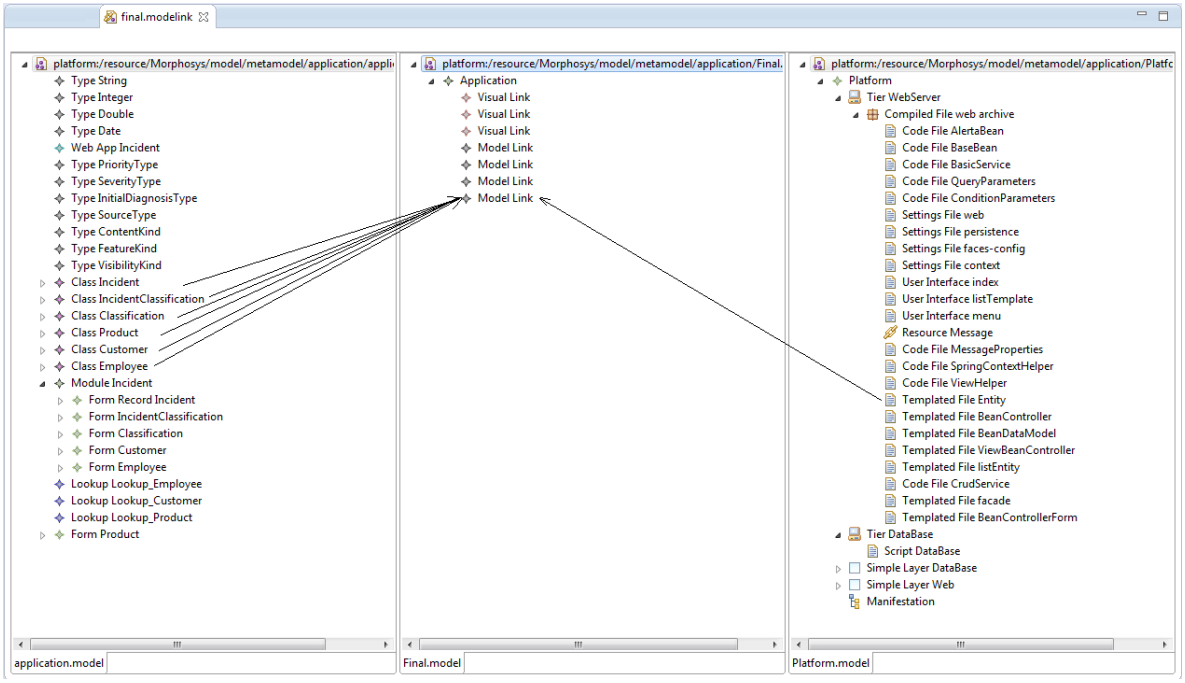


Ilustración 10: Enlace final de aplicación.

- Para el enlace final entre el modelo de la plataforma y el modelo de aplicación como se observa en la ilustración 10 se debe realizar un modelo weaving que representa la unión final entre los aspectos funcionales y los no funcionales como se muestra en la ilustración 11. El usuario debe relacionar las entidades que corresponden a la plantilla dentro de los componentes que dicta el diagrama de plataforma.

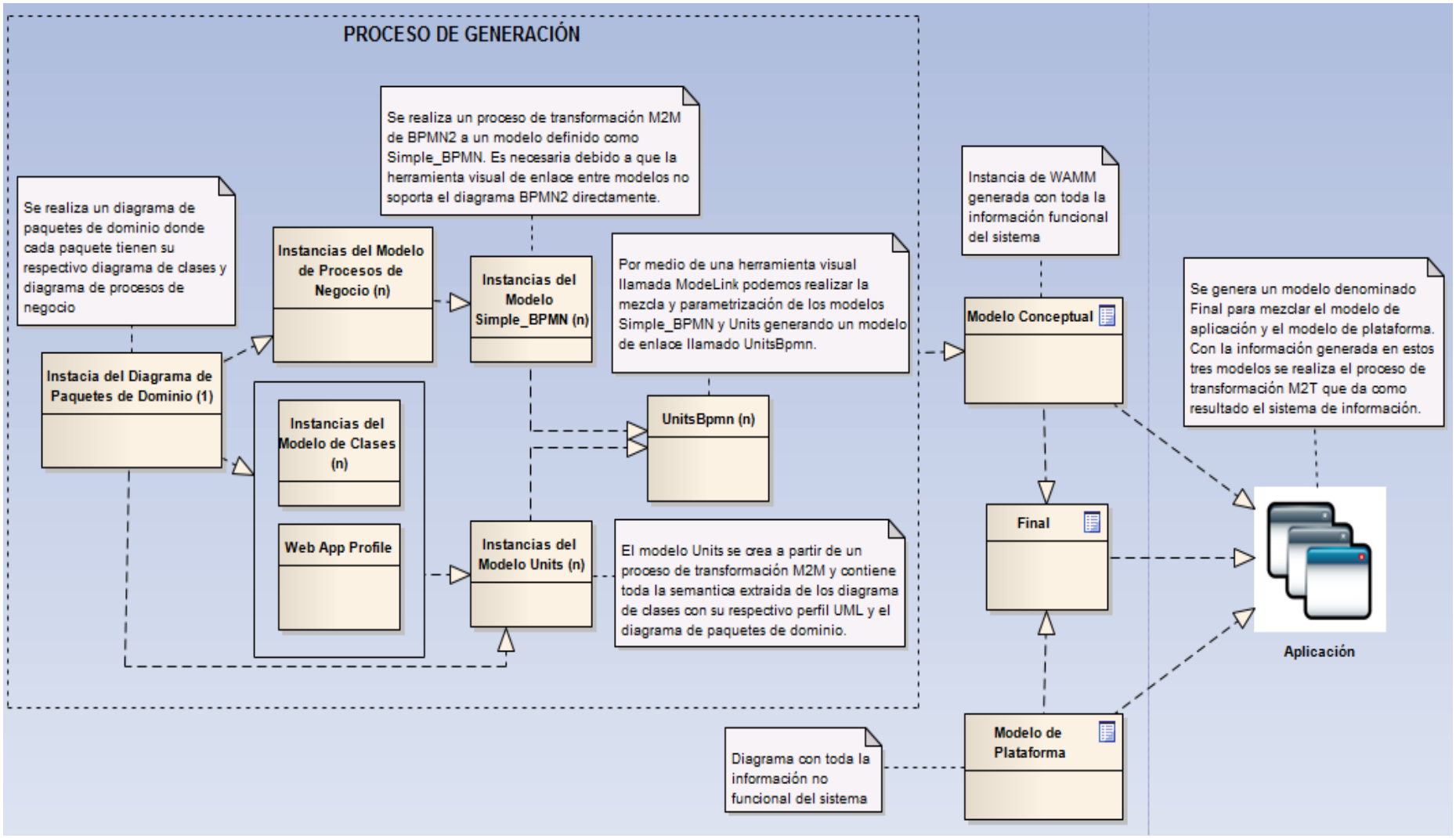


Ilustración 11: Proceso para obtener la aplicación.

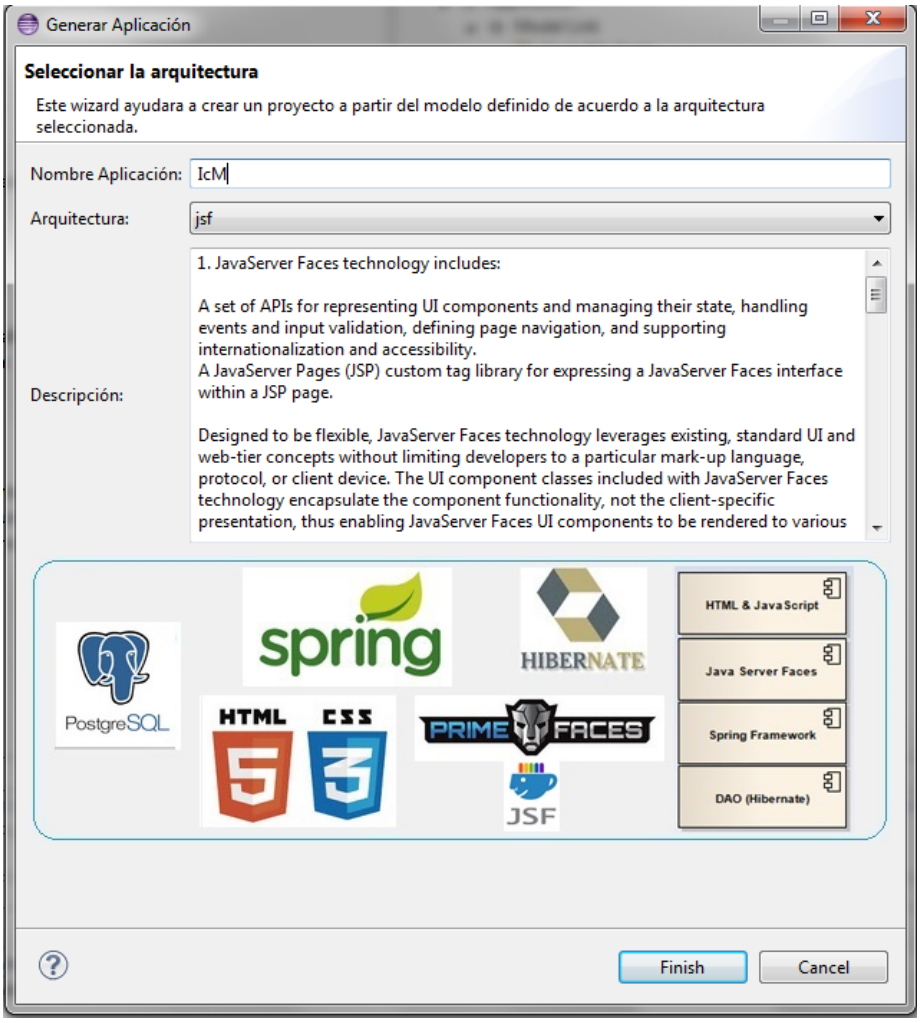


Ilustración 12: Selección de arquitectura de referencia.

- Finalmente se va a generar la aplicación a partir de las arquitecturas de referencia que se deben crear en el lenguaje de plantillas de tipo Epsilon EGL [39]. Por el momento se creó una arquitectura de referencia para Java, que consiste en un conjunto de plantillas donde están plasmados algunos componentes que debe tener una aplicación web JEE. Al seleccionarla se muestran las características y los *framework's* que están inmersas en dichas plantillas como se muestra en la ilustración 12.

La arquitectura de referencia que se va a utilizar es una arquitectura JEE, con tres capas Modelo, Vista y Controlador apoya sobre las tecnologías:

- Java Server Faces, Primefaces.
- Spring Framework.
- Java Persistence Api, Hibernate.
- BD Postgresql.

2. TRABAJOS RELACIONADOS

Algunas herramientas ya utilizan lenguajes de dominio específicos y reglas de transformación como MetAGeM (*Meta-Tool for the Automatic Generation of Model Transformations*), las cuales son la base de este trabajo. Con esta herramienta se pueden realizar transformaciones consecutivas y poder llegar a código fuente [40]; *aún* hay algunas limitantes debido a que *todavía* se necesita modificar el código generado, lo cual implica perdida de trazabilidad al modelo y conlleva problemas en las gestión de cambios de los aplicativos.

Adicionalmente, en el manejo de las transformaciones se deben hacer ajustes manuales para llegar al código generado. Sin embargo, es un punto de partida importante porque demuestra que si se puede mejorar el manejo de la complejidad de las transformaciones y da una guía de practica de usabilidad si se quiere realizar una herramienta similar. Otra herramienta en el mercado es UsiComp (Extensible Model-Driven Composer) [41], maneja el aspecto multiplaforma de una manera fácil donde por medio de un editor gráfico podría configurar y generar parte de su aplicativo, está muy enfocado en presentar soluciones con respecto a los requerimientos de interfaz gráfica. Es importante aclarar que hay poca documentación de esta herramienta.

Spring Roo [42], es una herramienta se utiliza para la generación de aplicaciones. Solo genera aplicaciones para la plataforma Java, adicionalmente esta herramienta solo genera aplicaciones para dos *frameworks* de presentación GWT y Spring MVC. También se debe aclarar que no hay un control total para decidir las características de los artefactos que se están generando. Otra herramienta que llevan mucho tiempo en el mercado es Webratio, esta *software* tiene costo y solo se pueden generar varios tipos particulares de aplicaciones de acuerdo a la versión utilizada; adicionalmente no tiene la posibilidad de modificar las transformaciones.

Con el enfoque del macro proyecto *Metáfora***,** donde se están utilizando unas vistas específicas de arquitectura dentro del Desarrollo de Software Dirigido por Modelos, no se encontraron estudios con estas características. Con respecto a trabajo que trate la intervención en las transformaciones, se han encontrado algunos focalizados en mejorar el proceso de transformación como se menciona en el punto 2.1.2.1.

2.1. Optimización en las Transformaciones

Para aportar flexibilidad, modificabilidad y personalización se están utilizando perfiles de transformación [17], donde se establecen varias posibilidades de distribución de componentes, de una manera muy acertada trabajaron la parametría de características de los componentes por medio de *Transformation Template* [26] donde se maneja el tema de prioridades para los componentes de la interfaz gráfica. También se tuvieron en cuenta indicadores (nivel de importancia y costo de desarrollo) aportando directamente contra los requisitos no funcionales del sistema. En el trabajo se identifican estrategias similares que se pueden usar para mejorar la intervención y el manejo de las transformaciones, como lo son las plantillas de transformación facilitando la personalización como también cediendo el control parcial de la transformación al usuario. También, evidencian los problemas actuales en el marco de MDSD con la gestión de transformaciones, como la experticia que se debe tener al manejar reglas de transformación y la inflexibilidad de las reglas, para adicionar información al momento de las transformaciones.

En el trabajo "*UsiComp : an Extensible Model-Driven Composer Ga*" trata de aportar en la semántica al realizar las transformaciones [43], donde se apoya en las combinaciones de colores para adicionar más información a los componentes del modelo, y así mejorar la calidad de las transformaciones debido a la información detallada generada en los elementos y sus relaciones.

En los trabajos anteriores se pueden evidenciar avances mejorando el proceso de transformación, donde se trata de apoyar más en configuraciones externas con base de plantillas, pero todavía se dificulta el nivel de estandarización porque los trabajos se realizaron sobre algunas herramientas comerciales, donde la *metadata* generada no es un lenguaje de socialización estándar. También, se evidencia la versatilidad que podría llegar a tener un esquema paramétrico durante la transformación, donde se pueden empezar a contemplar variables de requisitos no funcionales, este tipo de variables (Nivel de Importancia) serían muy útiles para el trabajo que debe realizar, debido a que permite guiar al usuario cuando se requieran tomar decisiones de diseño en el momento de ejecutar las transformaciones.

2.2. Orientado a configuraciones

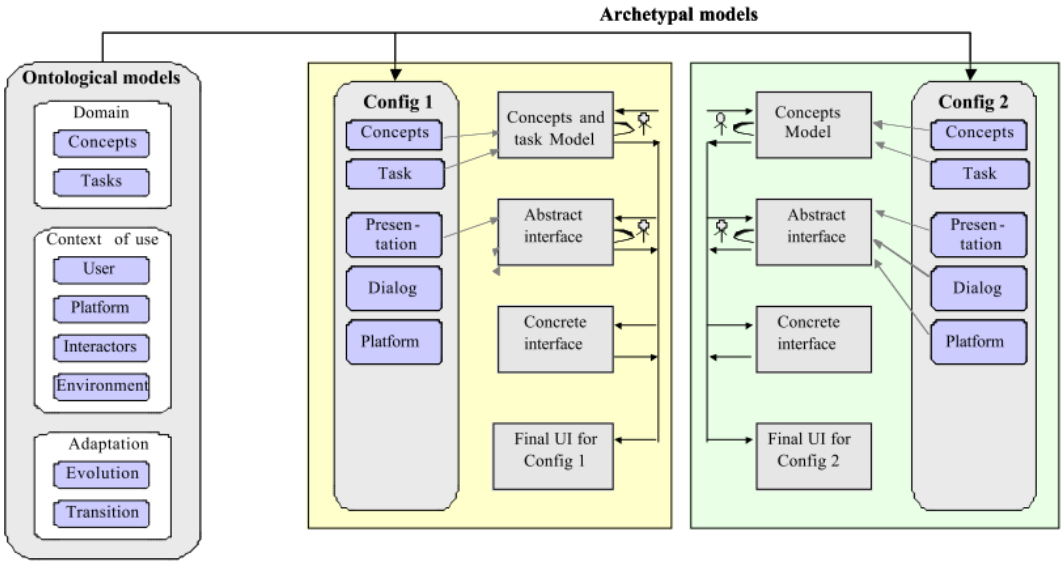


Ilustración 13: Marco de referencia Unifying [9].

Debido a los avances tecnológicos en el área de los dispositivos móviles (teléfonos celulares, tabletas, entre otros), surgen nuevas exigencias para los

sistemas de información entre ellos el rendimiento, usabilidad, seguridad, entre otros. Adicional a esto, se debe diseñar sistemas de información pensando en la compatibilidad con la gran variedad de plataformas vigentes en el mercado. El *Unifying Reference Framework* se presenta como un intento de caracterizar los modelos, métodos y procesos envueltos en el desarrollo de interfaces gráficas para diferentes tipos de dispositivos. Este framework trata de separar el contexto y la información de dominio de la representación gráfica particular, estableciendo configuraciones específicas para cada tipo de dispositivo [44], como se puede ver en la Ilustración 13.

El algunas ocasiones se mapea directamente el modelo de dominio con la interfaz gráfica, donde se utiliza UsiXML y la notación Concur Task Trees (CTT) para la representación de las tareas a ejecutar y poderlas enlazar con los componentes visuales que disparan dichas tareas [45]. La metodología mencionada es utilizada por el aplicativo IdealXml que tiene varias salidas preestablecidas (XHTML, Swing), a veces sugiere que se realicen cambios directamente sobre la interfaz generada, lo cual no es recomendable debido a que se pierde trazabilidad en el desarrollo y se comienza a generar sobrecarga en el mantenimiento del aplicativo. También, se debe tener en cuenta que se intenta llegar al código fuente casi de una manera directa desde el modelo, de esta manera hay muy pocas posibilidades de poder modularizar las transformaciones agregando complejidad en el desarrollo de las reglas.

2.3. Otros Modelos de Desarrollo

Para manejar el problema de desarrollo que conlleva contemplar múltiples plataformas de destinos, se están utilizando dos estrategias adicionales a MDE [46]:

- Compilación cruzada: esta estrategia busca tener un lenguaje común para su posterior traducción al lenguaje nativo deseado de acuerdo a la plataforma, el mapeo a lenguajes específicos es el mayor reto de esta estrategia, es mantener las reglas de mapeo teniendo en cuenta versiones, plataformas y diferentes contextos de los lenguajes destino. Algunas de las herramientas utilizadas son MoSync1, Corona2, Neomades3 y XMLVM.

- *Interpreters*: se utiliza una máquina virtual o también ésta enmarcada la utilización de lenguajes que pueden ser interpretados por el navegador web. Básicamente hay un motor que ejecuta las sentencias dictadas por el código intermedio y/o código fuente. Algunas de las herramientas que usan esta filosofía son: *J2ME Polish, Bedrock, AlcheMo*, en el caso de intérpretes web.

Estas soluciones están evolucionando rápidamente y están captando la atención de la comunidad en el desarrollo de software, algunos de los más reconocidos son: *PhoneGap, QuickConnectFamily* y *Rhomobile*. Este tipo de soluciones oculta muchos de los procesos utilizados en las estrategias de transformación y no usan estándares interoperables con otras herramientas, dificultando el espectro funcional de los desarrollos. Otro aspecto particular es que no se puede tener la versatilidad de escoger la plataforma destino, estas herramientas tienen una gama de posibilidades fija de plataformas en las que pueden generar el código.

Algunas de las herramientas antes mencionadas pueden llegar a ser muy costosas. También, se debe tener en cuenta que la libertad sobre el código generado está limitado, a lo que pueda ofrecer el entorno o *wizard* del aplicativo que genera. A diferencia de MDE, donde se tiene la libertad total de modificar las plantillas de acuerdo a los requisitos funcionales, no funcionales, mejores prácticas de desarrollo, patrones, entre otros aspectos.

3. ESTUDIO DE CASO Y TECNOLOGÍAS

La prueba para validar este proyecto está basada en la creación de una aplicación de gestión de incidentes heredada desde el proyecto macro Metáfora. El cual menciona lo siguiente:

“Para posibilitar la ilustración de las diversas fases de un proceso y las disciplinas de un método, es imperativo tener un estudio de caso que permita ejemplificar las vistas y tipo de modelos a utilizar. Es importante que dicho estudio de caso, permita trabajar las diferentes dimensiones de un sistema de información, de tal forma que se pueda pasar de una vista en la que se consiga implementar una aplicación, hasta múltiples vistas que permitan ver los diferentes frentes de la aplicación construida.

Considerando lo anterior, se tomará como referente de trabajo ITIL (Information Technology Infrastructure Library), marco para la gestión de servicios de tecnologías de la información, el cual propone los tópicos para consolidar el modelo de "ciclo de vida del servicio", separando y ampliando algunos subprocesos hasta convertirlos en procesos especializados. ITIL en su versión 3 consta de los siguientes 5 libros basados en el ciclo de vida del servicio: 1 -Estrategia del servicio, 2-Diseño del servicio, 3-Transición del servicio, 4-Operación del servicio y 5-Mejora continua del servicio; esto lo convierte en un referente con un campo de acción muy amplio. Por la naturaleza de este trabajo, el estudio de caso se basará en el libro 4-Operación del servicio, específicamente en el proceso de proceso de gestión de incidentes, frecuentemente llamado *IcM* por la sigla en inglés de Incident Management.

El estudio de caso se centrará entonces en el desarrollo de un sistema de información para manejar el ciclo de vida del proceso de gestión de incidentes, en este sentido se debe tener el proceso que describe ITIL alrededor de IcM. A continuación se hace una breve compilación de lo que propone el libro 4 en este frente,

y se adapta al área de servicios de soporte y mantenimiento de sistemas de información:

- El usuario final detecta un incidente en uno de los sistemas de información que son responsabilidad de la compañía.
- Cuando el analista de la mesa de ayuda recibe el reporte, realiza el registro del incidente, luego lo clasifica y determina si se conoce una solución al posible problema.
- Si existe una solución conocida, procede a resolver y cerrar el incidente.
- Si no existe una solución conocida y la prioridad no es alta, se escala el
- Incidente al experto de la mesa de ayuda, quien investiga, diagnostica y reparar el incidente, luego se resuelve y recupera el sistema, y finalmente se envía al analista de la mesa de ayuda para que cierre el incidente.
- Si no existe una solución conocida y la prioridad es alta, el incidente se
- envía al experto en la materia en cuestión, quien determina si efectivamente se trata de un incidente mayor.
- Si no se trata de un incidente mayor, se remite el caso al experto de la mesa de ayuda, para que este realice el proceso desde la investigación
- hasta el envío para el cierre.
- Si por el contrario si se trata de un incidente mayor, el experto en la materia en cuestión procede a encontrar una solución, antes de remitir el caso al experto de la mesa de ayuda, para que este realice el proceso desde la investigación hasta el envío para el cierre.” [47]

Como lineamiento desde el proyecto macro Metáfora se debe utilizar la familia de lenguajes Epsilon accesibles desde un plugin de Eclipse que es compatible con los modelos y metamodelos generados desde el framework de modelo de Eclipse. Este plugin aporta las siguientes características:

- ✓ Familia de Lenguajes para la generación de código, transformación de modelo a modelo, validación [32], comparación, migración y refactorización de modelos.

- ✓ Facilidad en la creación de metamodelos.
- ✓ Facilidad de interacción con los metamodelos existentes sobre el metamodelo de UML y BPMN para realizar lectura y escritura de dichos modelos.
- ✓ Proporcional la funcionalidad necesario para crear e intervenir modelos dentro del proceso de generación.

Para el modelado de paquetes se utilizó un plugin de Eclipse llamado Papyrus donde se realizaron los modelos de paquetes de dominio y diagramas de clases. Debido a que estos modelos son basados en EMF fue natural el trabajo de transformación que se realizó desde el plugin Epsilon.

El modelo de procesos de negocio se realizó bajo el plugin más conocido de BPMN para eclipse, Eclipse BPMN2.

Toda la configuración se realizó sobre la plataforma de Eclipse, la versión puntual sobre la que se trabajó fue Kepler debido a la inestabilidad de los plugins para otras versiones.

4. VISTA DE LA APLICACIÓN

Para poder llevar a cabo la generación de la aplicación con la metodología MDSD y bajo la estrategia antes definida en el macro del proyecto Metáfora se debe realizar un análisis de los componentes que conforman dicha vista para poder implementar modelos que tenga la semántica suficientes y sirvan de manera efectiva en el proceso de generación de la aplicación. La vista de aplicación se compone de dos vistas:

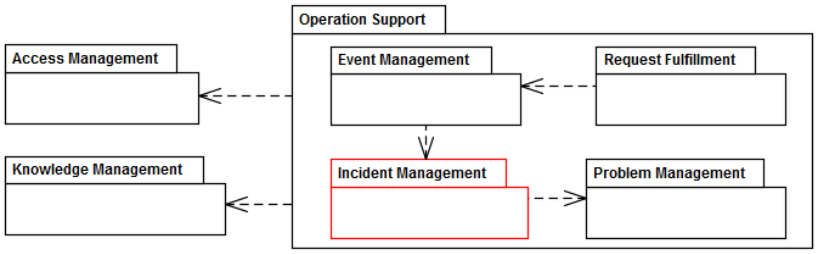


Ilustración 14: Diagrama de paquetes de dominio IcM.

- **Vista conceptual**: esta vista está compuesta a su vez por dos modelos:
    - Diagrama de Clases.
    - Diagrama de Paquetes de Dominio como se puede visualizar en la ilustración 14.

- **Vista de escenarios**: esta vista está compuesta por:
    - diagrama de procesos de negocio.

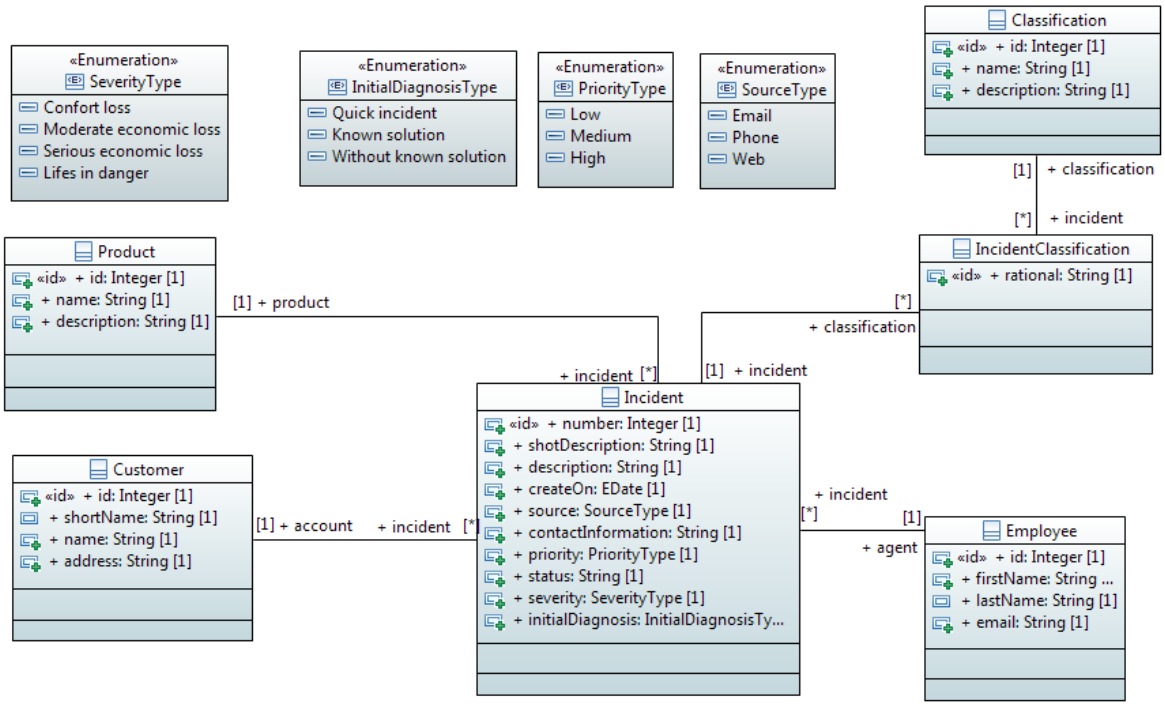


Ilustración 15: Diagrama de clases del aplicativo de gestión de incidentes.

4.1 Vista Conceptual

Durante el proceso de desarrollo generalmente esta vista la crea el analista de negocio junto con el analista de desarrollo.

4.1.1. Diagrama de Paquetes

El diagrama de paquetes es un diagrama que determina cuales son los subsistemas de una aplicación. Deben representar diferentes módulos funcionales del sistema que se está diseñando. En algunos casos se considera la entrada más importante en la fase de diseño.

4.1.1.1. Consideraciones del Diagrama de Paquetes

El diagrama de paquetes para efectos de este trabajo soportará todos los componentes UML 2.0, pero para las transformaciones en Metáfora no toda la información contenida se utilizará para el proceso de transformación iterativa. Actualmente se usa como agrupador funcional para las entidades relacionadas en los diagramas de clases y procesos de negocio.

4.1.2. Diagrama de Clases

El diagrama de clases como se muestra en la ilustración 15 se utiliza para representar los conceptos, sus atributos y relaciones relevantes en el dominio del ambiente de la solución en la cual se está trabajando para realizar el diseño [3].

4.1.2.1. Consideraciones del Diagrama de Clases

El diagrama de clases para efectos de este trabajo soportará todos los componentes UML 2.0, pero para las transformaciones en Metáfora no toda la información contenida se utilizará para el proceso iterativo de transformación entre modelos, los componentes utilizados del diagrama de clases son las clases, los atributos, los tipos de los atributos, las asociaciones. Se debe tener en cuenta también que para ayudar en la expresividad de este modelo se creó un perfil [31] con ciertas anotaciones que ayudan a identificar características de las entidades y sus relaciones.

4.2. Vista de Escenarios

Durante el proceso de desarrollo generalmente esta vista la crea el analista de negocio. Este diagrama se relaciona con el diagrama de clases por medio de un modelo adicional, pero es totalmente independiente.

4.2.1. Diagrama de Procesos de Negocio

Este diagrama determina el comportamiento del proceso [7] en el ambiente del problema o dominio en el que se está diseñando la solución como se puede ver en la ilustración 16. Este diagrama generalmente es el puente entre usuarios no técnicos y técnicos, ayuda a establecer un punto general de entendimiento del sistema para todos los involucrados desde usuarios finales hasta desarrolladores del producto.

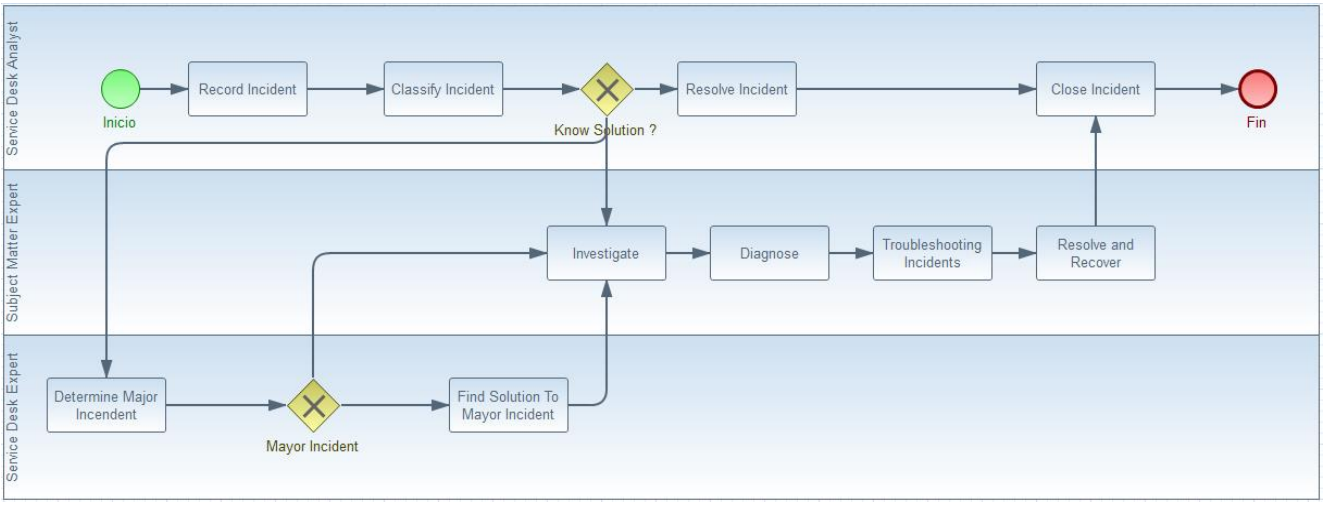


Ilustración 16: Diagrama de procesos de negocio del aplicativo de gestión de incidentes.

4.2.1.1. Consideraciones del Diagrama de Procesos de Negocio

El diagrama de procesos para efectos de este trabajo soportará todos los componentes BPMN 2, pero para las transformaciones en Metáfora no toda la información contenida se utilizará para el proceso de generación de modelos iterativa, se utilizan los componentes de inicio, fin, lanes, tareas, compuertas OR y compuertas AND. Este modelo determina flujos de información entre módulos, identifica condiciones sistémicas y ayuda a identificar aspectos de seguridad debido a la naturaleza de las actividades según su ubicación en los carriles (Lane).

5. CONSTRUCCIÓN DEL DSL

La construcción del DSL tiene varios frentes de trabajo para poder dotar al modelo de aplicación de toda la información necesaria y tener una generación de la aplicación lo más completa posible. Todos los metamodelos generados para este trabajo estarán realizados en ECORE pieza fundamental del *framework* de modelado de eclipse (EMF).

5.1. Modelo de Clases

Para enriquecer el modelo de clases con información adicional se le adicionó un perfil y este hace parte del proyecto Metáfora el cual se denomina *Web App Profile*, dicho perfil se puede apreciar en la ilustración 17.

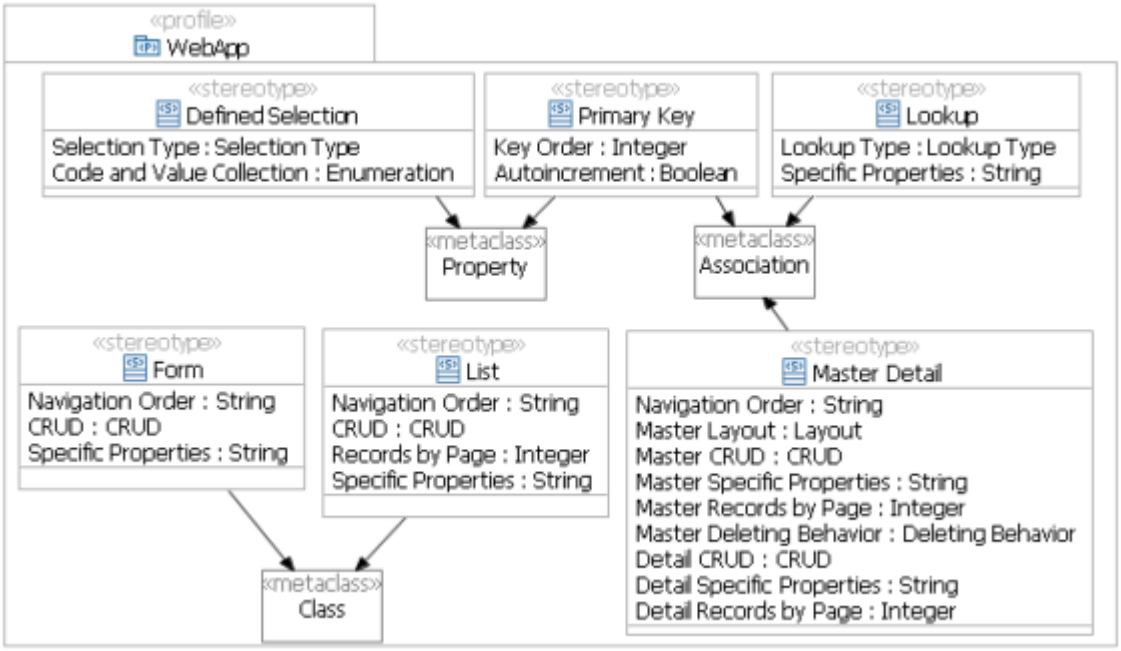


Ilustración 17: Perfil *Web App Profile* [48].

5.1.1. Semántica del Modelo de Clases

Como se puede observar en la imagen anterior el perfil está orientado a clases, propiedades y asociaciones dentro del modelo de clases como se muestra en la ilustración 18. Las siguientes reglas se establecen para el perfil de acuerdo a la generación de Modelo a Modelo (M2M):

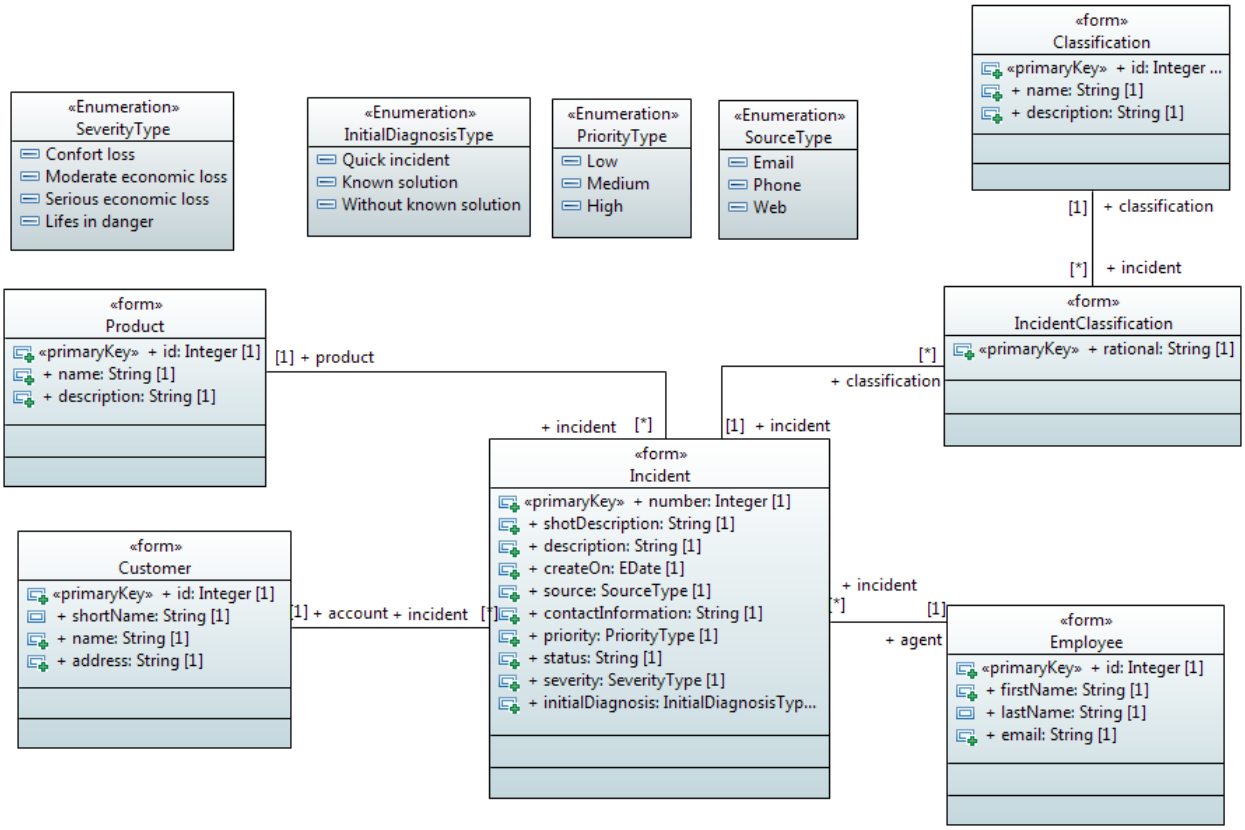


Ilustración 18: Instancia del modelo de clases con el perfil.

- Cada propiedad que está marcada con el estereotipo <*Primary Key*> genera un objeto de llave primaria en el modelo de aplicación.
- Los atributos que estén marcados con el estereotipo <*Defined Selection*> tiene que tener definida la propiedad <*Selection Type*> del estereotipo.
- En el diagrama de clases cada entidad que está marcada con el estereotipo <*Form*> corresponde a un objeto de tipo *Form* y su correspondiente objeto de tipo *BaseClass* en el modelo de aplicación.

- En el diagrama de clases cada entidad que está marcada con el estereotipo <*List*> corresponde a un objeto de tipo *List* y su correspondiente objeto de tipo BaseClass en el modelo de aplicación.
- En el diagrama de clases cada asociación que está marcada con el estereotipo <*Master Detail*> corresponde a un objeto de tipo *Master Detail* en el modelo de aplicación. Este objeto debe tener la referencia a los dos objetos de tipo AppBlock.
- En el diagrama de clases cada atributo del modelo de clases corresponde a objeto de tipo Field y su correspondiente objeto de tipo Property. Adicional los tipos de atributos se mapean directamente sin ningún tipo de transformación.
- La Enumeraciones en el diagrama de clases aparecen como tipos y sus correspondientes valores en el modelo de aplicación.

5.1.2. Validaciones del Modelo de Clases

Durante el proceso de transformación siempre se realiza un proceso de validación para garantizar que el modelo destino se genere de manera consistente. Las siguientes validaciones se realizan al modelo de clases.

- ✓ Todas las clases deben tener un nombre.
- ✓ Todas las clases deben tener atributos.
- ✓ Todas las clases deben tener un atributo con el estereotipo <id>.
- ✓ Todos los atributos deben tener un nombre y un tipo.

Estas validaciones se realizan si han sido configuradas respectivamente en la secuencia de configuración.

5.2. Metamodelo Units

Durante el proceso de transformación hasta llegar a el modelo de aplicación se debe especificar la asociación entre paquete de dominio y clases como se muestra en la ilustración 20. Esta fusión se debe realizar mediante un metamodelo Units como se muestra en la ilustración 19, que relacione componentes entre estos dos modelos.

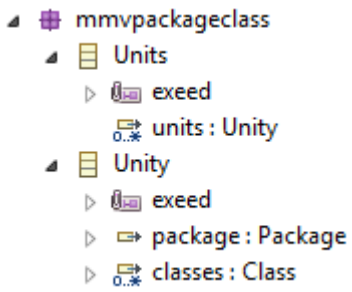


Ilustración 19: Metamodelo Units

Como se puede ver en la ilustración 20 se crean entidad Units que lo componen un paquete y clases.

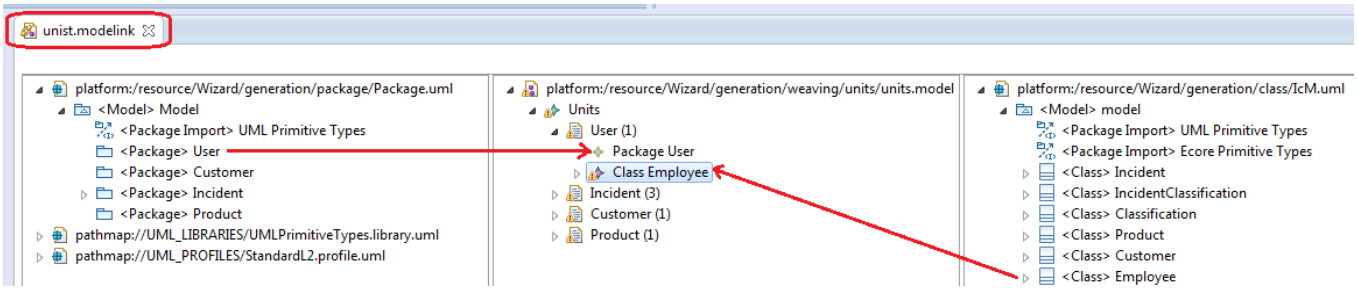
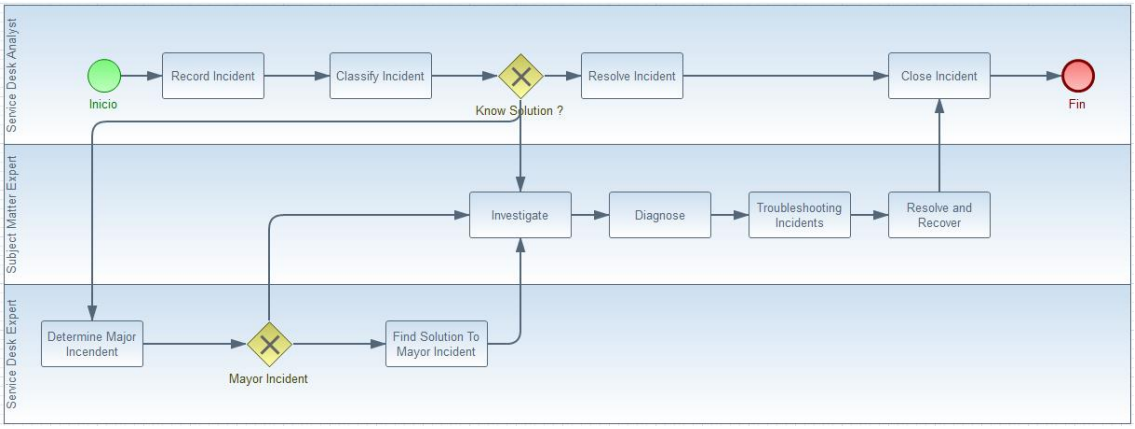


Ilustración 20: Combinación de modelos.

5.2.1. Semántica de Modelo Units

Como se puede observar en la imagen anterior el modelo de Units está orientado a unir clases y paquetes para identificarse como unidades. Las siguientes reglas se establecen para el modelo:

- Cada Unit será mapeado al modelo de aplicación como un módulo de sistema en el modelo de aplicación.
- Las clases que estén contenidas en un Units harán parte del módulo correspondiente en el modelo de aplicación. Si hay dos clases con el mismo nombre en los modelos de clases de diferentes Units, el sistema asumirá que es la misma entidad.



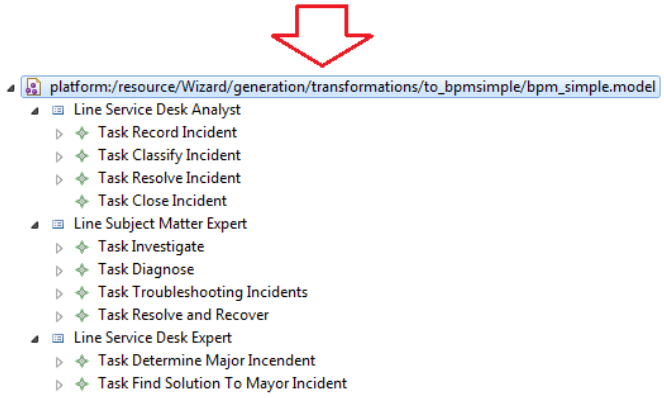


Ilustración 21: Simplificación Modelo de BPMN.

5.2.2. Validaciones del Modelo Units

Las siguientes son las validaciones realizadas al modelo **Units**.

- Una unidad solo puede contener un paquete.
- Un paquete puede contener de una a muchas clases.

Por el momento el prototipo no valida si hay dos clases con el mismo nombre dentro del mismo diagrama.

5.3. Metamodelo Simple_BPMN

El diagrama de procesos de negocio se realizó sobre el plugin de bpmn2, el cual no es soportado por la herramienta ModeLink de Epsilon utilizada para relacionar o enlazar modelos, es por esto que se creó un metamodelo más simple como se muestra en la ilustración 22, y realizar el paso de los componentes que tienen valor semántico para las transformaciones de BPMN2 a una instancia del modelo de que se definió como **Simple_BPMN,** este proceso de transformación se puede observar en la ilustración 21.

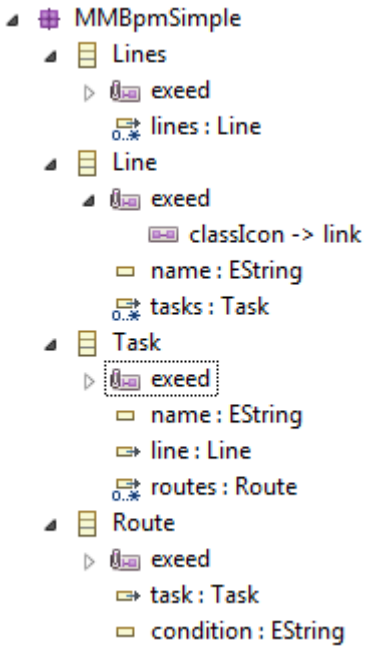


Ilustración 22: Metamodelo Simple BPMN

5.3.1. Semántica de Simple_BPMN

Este modelo aporta de la siguiente manera hacia la transformación M2M que da como resultado el modelo de aplicación:

- Cada Carril indica un rol dentro del metamodelo aplicación.
- Cada Task indica una interfaz gráfica dentro del modelo de aplicación.
- Cada Compuerta Exclusiva da a cada interfaz gráfica las condiciones para poder navegar hacia otra interfaz.
- Cada Compuerta Inclusiva da a cada interfaz gráfica las rutas donde podrá navegar hacia otra interfaz.

5.3.2. Validaciones del Simple_Bpmn

Las siguientes son las validaciones realizadas al modelo **Simple_Bpmn**.

- Todas las tareas deben tener un nombre.
- Todas las tareas deben estar dentro de un Carril.
- Todas las tareas deben estar conectas a otros componentes.
- El proceso debe tener un evento de inicio y uno o más eventos de fin.

5.4. Metamodelo UnitsBpmn

Para realizar la integración entre modelos **Units** (Clases vs Paquetes de Dominio) y **Simple_Bpmn** se debe realizar un metamodelo el cual fue definido como UnitsBpmn como se puede visualizar en la ilustración 23. Con el enlace de estos modelos ya se tiene la información suficiente para generar el modelo de aplicación porque se tienen todos los elementos necesarios tanto de la vista conceptual como de la vista de escenarios como se puede ver en la ilustración 24.

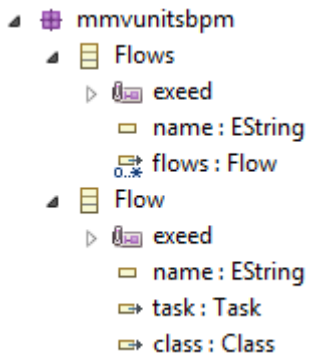


Ilustración 23: Metamodelo UnitsBpmn

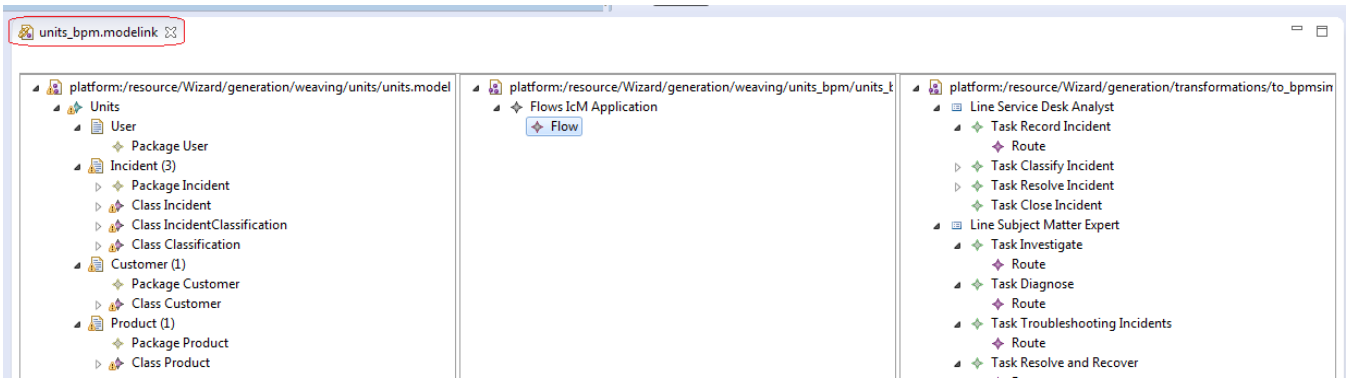


Ilustración 24: Modelo Weaving UnitsBpmn.

5.4.1. Semántica de UnitsBpmn

Este modelo aporta de la siguiente manera hacia la transformación M2M que da como resultado el modelo de aplicación:

- Cada Flow (Unidad de Flujo) debe tener un nombre.
- Cada Task indica una interfaz gráfica dentro del modelo de aplicación.
- Cada Compuerta Exclusiva da a cada interfaz gráfica las condiciones para poder navegar hacia otra interfaz.
- Cada Compuerta Inclusiva da a cada interfaz gráfica las rutas donde podrá navegar hacia siguiente interfaz.

5.4.2. Validaciones del UnitsBpmn

Las siguientes son las validaciones realizadas al modelo **UnitsBpmn**.

- Debe existir al menos un Flow.
- Cada Flow debe tener un nombre.
- Cada Flow debe tener una tarea y una clase asociada.

5.5. Metamodelo de Aplicación

Para obtener toda la información de la vista de aplicación se debe tener un metamodelo que contenga toda la información relevante de los siguientes modelos.

- Modelo de clases
- Modelo de paquetes de dominio
- Modelo de Units
- Modelo de Simple_Bpmn
- Modelo de UnitsBpmn

5.5.1. Semántica del Modelo Aplicación

Este modelo tiene todos los componentes necesarios junto con el modelo de plataforma para generar una aplicación. El modelo de aplicación como se puede observar en la ilustración 20 tiene los siguientes componentes:

- WebApp: componente general para identificar la aplicación.
- Class: para identificar las entidades representativas del contexto de la solución.
- Type: componente para identificar los tipos de atributos.
- Property: atributo identificable de un objeto de tipo Class.
- ForeingKey: identifica la relación entre objetos de tipo Class.
- PrimaryKey: determina la identificación de forma única de objeto de tipo Class.
- Field: corresponde a un campo dentro de una interfaz gráfica que tiene relación con un objeto de Property.
- AppBlock: corresponde a una interfaz gráfica, tiene diferentes especializaciones.
- Form: corresponde a una interfaz gráfica de tipo Formulario.
- List: corresponde a una interfaz gráfica de tipo Lista.
- Modules: sirve para realizar la agrupación de bloques de elementos (AppBlock).
- LookupTypes: tipos de relación entre interfaces graficas de acuerdo a la relación entre clases.
- SeleccionType: tipo de objeto que representa como se debe visualizar un atributo dentro de una interfaz gráfica.
- Master Detail: tipo de objeto que representa una asociación entre objetos de tipo AppBlock.

5.5.2. Validaciones del Modelo de Aplicación

Las siguientes son las validaciones realizadas al **Modelo de Aplicación**, ilustración 21.

- Debe existir un objeto de tipo WebApp.
- El objeto de tipo WebApp debe tener un nombre.
- El objeto de tipo WebApp debe tener un prefijo.
- Debe existir por lo menos un objeto de tipo Module.
- Debe existir por lo menos un objeto de tipo AppBlock.
- Debe existir por lo menos un atributo Field dentro de cada objeto de tipo AppBlock.
- Debe existir por lo menos un atributo PrimaryKey relacionado con un objeto de tipo AppBlock.
- Debe existir por lo menos un objeto de tipo Class.
- Debe existir por lo menos un atributo Property dentro de cada objeto de tipo Class.

Estas son las reglas básicas que se realizaron para el prototipo, pueden ser más extensas de acuerdo al nivel de madurez de la herramienta, por el momento el prototipo soporta estas validaciones.

Como se puede observar en la ilustración 25 el modelo de aplicación tiene los componentes necesarios para capturar toda la semántica proveniente de los diagramas de paquetes, diagramas de clases y diagramas de procesos de negocio.

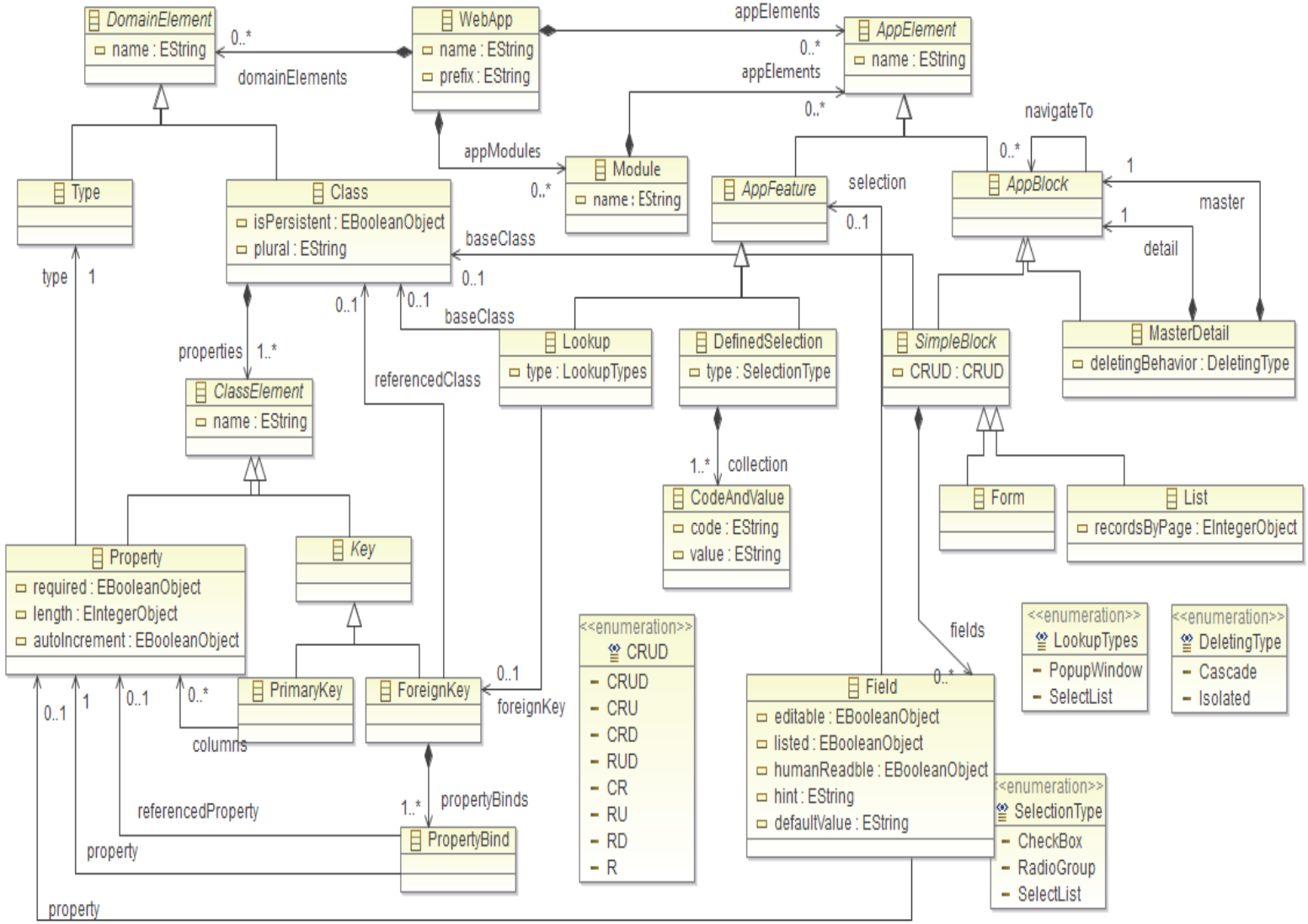


Ilustración 25: Metamodelo de Aplicación (WAMM).

6. GENERACION DE LA APLICACIÓN

Para el punto donde se realiza la generación de la aplicación ya se tienen como insumos dos modelos para llegar a la transformación de modelo a texto como se puede observar en la ilustración 26. Hay un tercer actor el cual es un modelo de enlace para establecer relaciones de aspectos funcionales que entrega el modelo de aplicación y aspectos no funcionales que entrega el modelo de la plataforma.

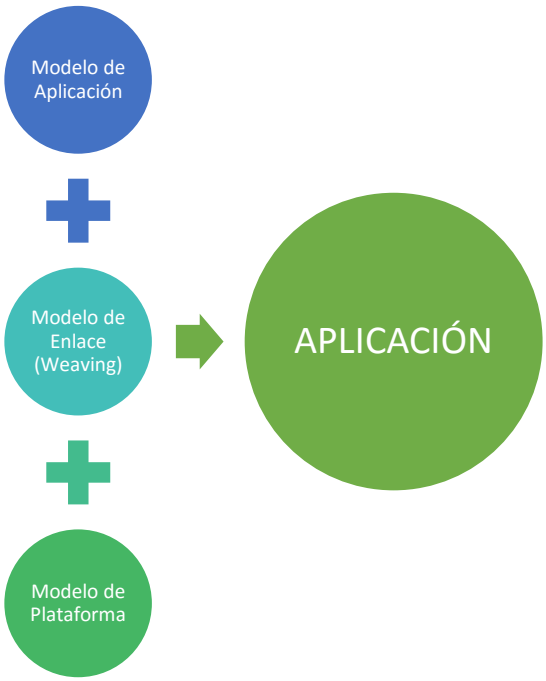


Ilustración 26: Combinación para la generación de la aplicación.

Con el propósito de mejorar el proceso de generación se realizó un plugin llamado ***Mophosys*** sobre la plataforma Eclipse para guiar al usuario durante todo el desarrollo. Este asistente trata de generar todos los artefactos necesarios y realizar todas las actividades hasta el punto donde se requiera necesariamente la intervención del usuario.

6.1. PROCESO DE GENERACIÓN

La generación de una aplicación tiene varias etapas de acuerdo al método usado en el proyecto Metáfora, dichas etapas se plasmaron dentro de un plugin de Eclipse. Los pasos para realizar una aplicación son los siguientes.

Como requisito inicial para poder realizar el desarrollo en Eclipse Kepler, se deben instalar los siguientes plugins:

1. Papyrus: plugin para la creación de modelos UML.
2. BPMN2: plugin para realizar los modelos de procesos de negocio.
3. Epsilon: plugin necesario para poder realizar la trasformaciones de modelo a modelo, modelo a texto. Además da la funcionalidad de poder intervenir las transformaciones por medio de los modelos de enlace (Weaving).

Se procederá a realizar el proceso de generación con el caso de estudio de Gestión de Incidentes (IcM).

6.1.1. CREACIÓN DEL PROYECTO

La fase inicial se debe crear un proyecto de tipo ***Mophosys*** como se puede ver en la ilustración 27 y la ilustración 28***.*** Este proyecto genera todos los artefactos necesarios para iniciar el proceso de diseño.

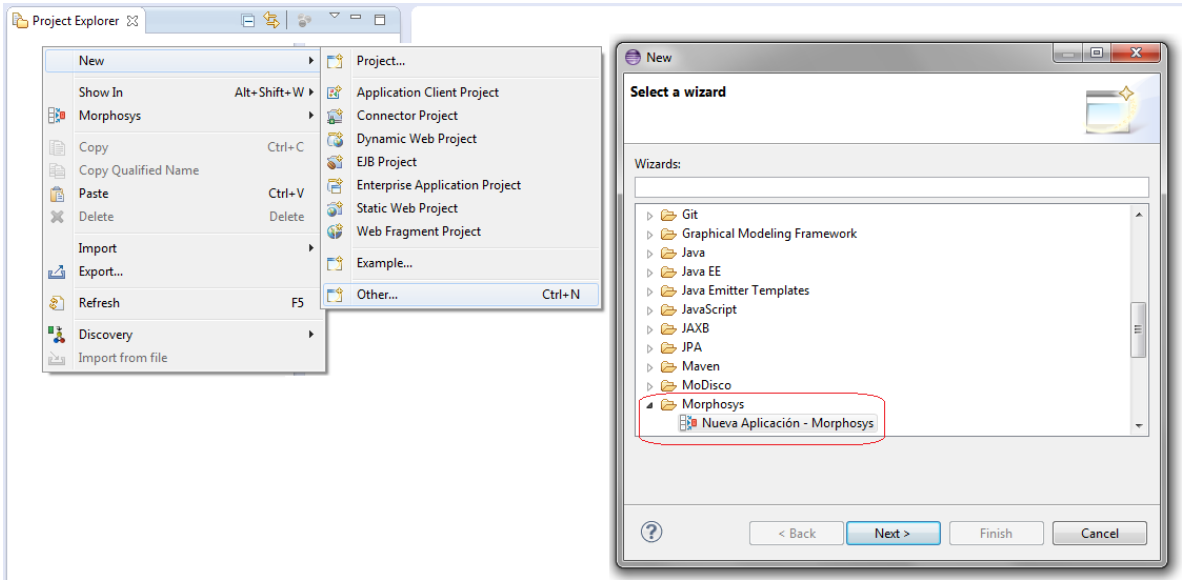


Ilustración 27: Creación de un proyecto de tipo Morphosys.

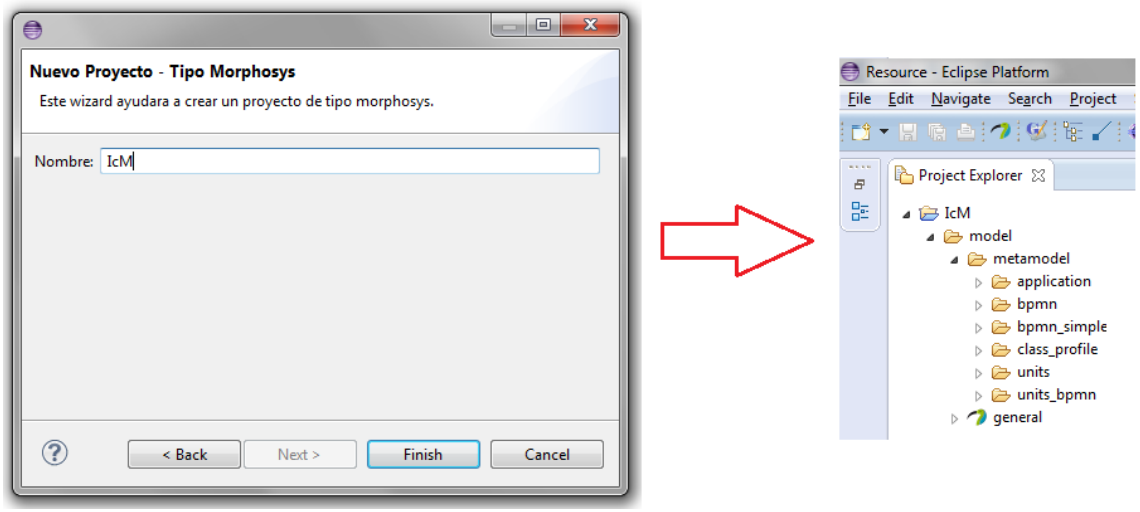


Ilustración 28: Componentes básicos de un proyecto de tipo Morphosys.

6.1.2. CREAR ESTRUCTURA DE DOMINIO

El punto de partida de diseño es crear todos los paquetes de dominio que se consideren necesarios para realizar el desarrollo de la solución por medio de plugin como se muestra en la ilustración 29, luego de crear los paquetes se selecciona el

proyecto Morphosys y el plugin generará la cantidad de carpetas de acuerdo a la cantidad de paquetes modelados en el diagrama de paquetes de dominio. Cada paquete de dominio permitirá realizar un diagrama de clases y un diagrama de BPMN esto permite realizar el diseño de la aplicación más modular y mejor estructurada.

Para el caso de estudio genera un paquete:

- Incident

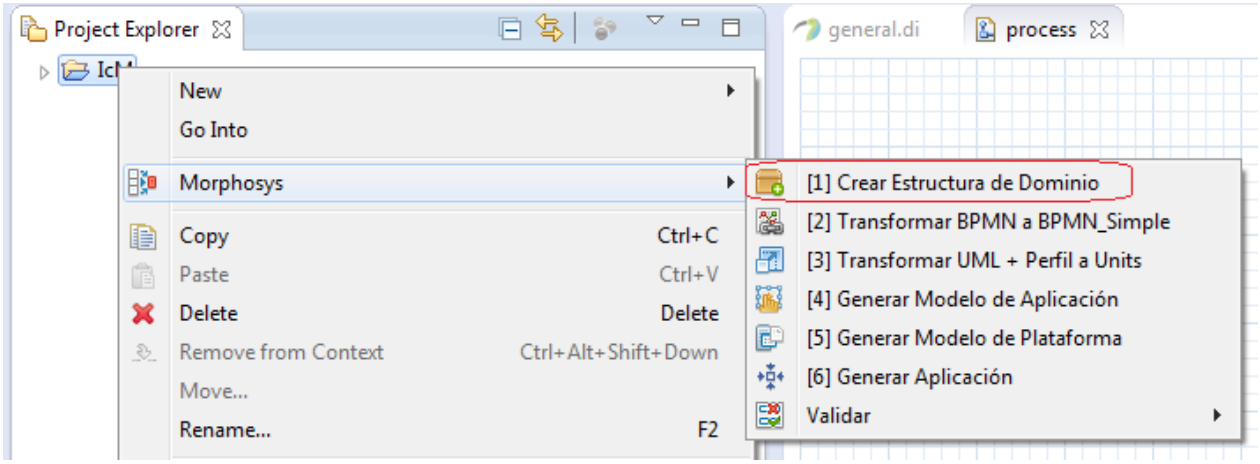


Ilustración 29: Opción en Morphosys para crear estructura de dominio.

En cada carpeta que representa un paquete de dominio se encuentran dos tipos de modelos:

- ✓ Modelo de Clases: el usuario debe realizar el modelamiento de todas las entidades de dicho paquete de dominio.
- ✓ Diagrama de procesos de negocio: el usuario se debe modelar el comportamiento de las entidades antes modeladas para el paquete.

6.1.3. GENERAR MODELO UNITSBPMN

La acción de generar el modelo UnitsBpmn se debe realizar para poder asociar elementos del modelo de **Units** (Clases y Paquetes) contra el modelo **Simple_Bpmn** (Elementos BPMN2). Se deben realizar las opciones 2 y 3 desde el menú contextual del proyecto tipo Morphosys como se observa en la ilustración 30.

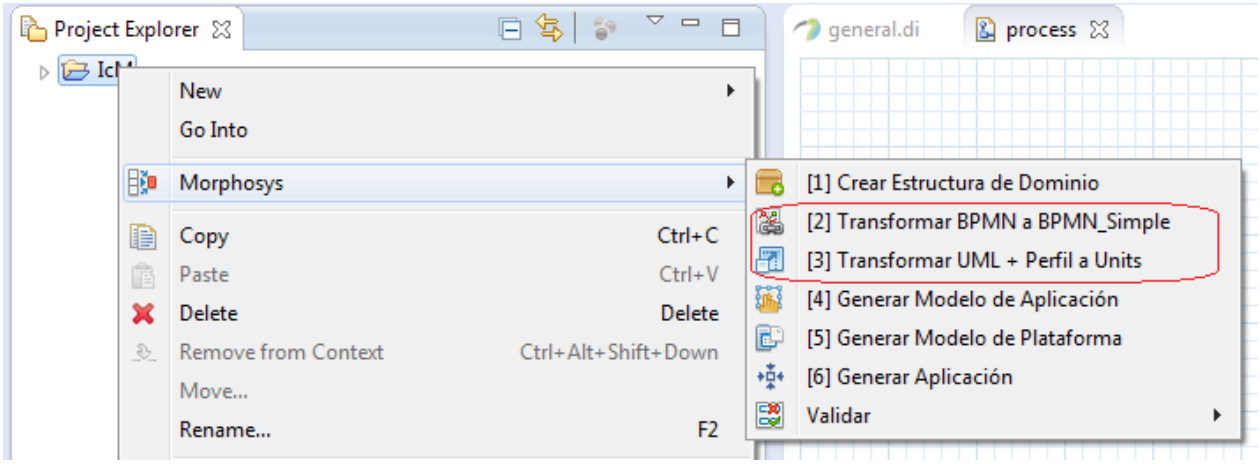


Ilustración 30: Opciones en Morphosys para crear modelos UnitsBpmn.

En estas acciones en el plugin generan modelos de enlace como se puede visualizar en la ilustración 31. El usuario debe realizar las asociaciones respectivas por medio de la herramienta del Plugin de Epsilon ModeLink donde podrá asociar elementos del modelo units que tiene la semántica correspondiente del diagrama de paquetes de dominio y el diagrama de clases con el modelo bpmn_simple que tiene la información de comportamiento correspondiente al diagrama de procesos de negocio, la relación entre los modelos se puede observar en la ilustración 32.

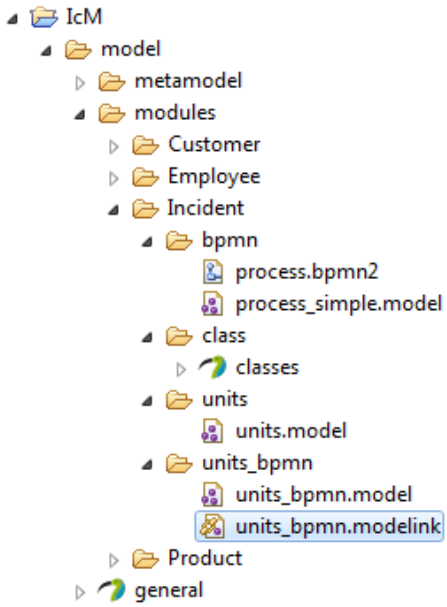


Ilustración 31: Componentes de Weaving UnitsBpmn.

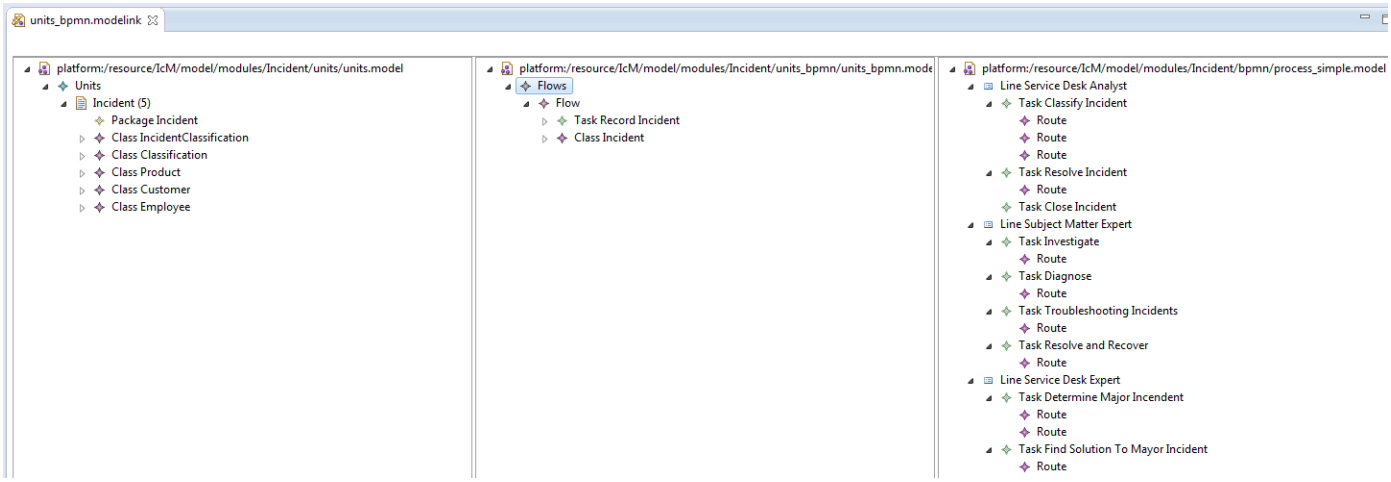


Ilustración 32: Visualizando combinación de modelos Units vs Simple_BPMN.

6.1.4. GENERAR MODELO DE APLICACIÓN

En este paso ya se tiene toda la información del modelo que se requiere generar desde el punto de vista funcional.

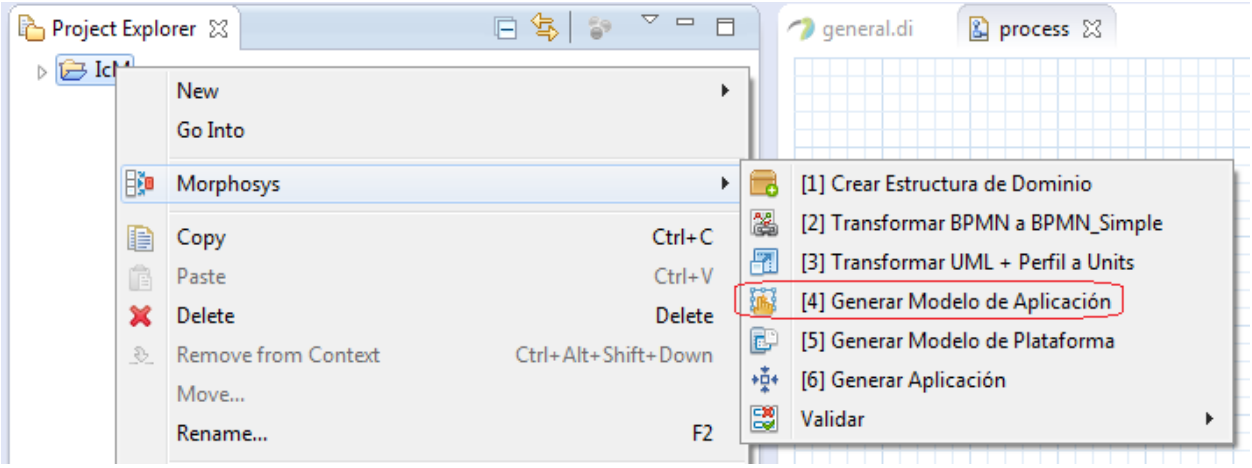


Ilustración 33: Opción en Morphosys para crear el modelo de aplicación.

Cuando se ejecuta esta acción desde el plugin como se visualiza en la ilustración 33 se genera el modelo de aplicación con toda la información que se creó en cada módulo de dominio, para el caso de estudio se consolidará lo siguiente:

1. Modelos de clases de los módulos:

    - Incident

2. Modelos de procesos de negocio de los módulos:

    - Incident

Como se puede observar en la ilustración 34 se realizó un trasformación de modelos de módulos a modelo de aplicación.

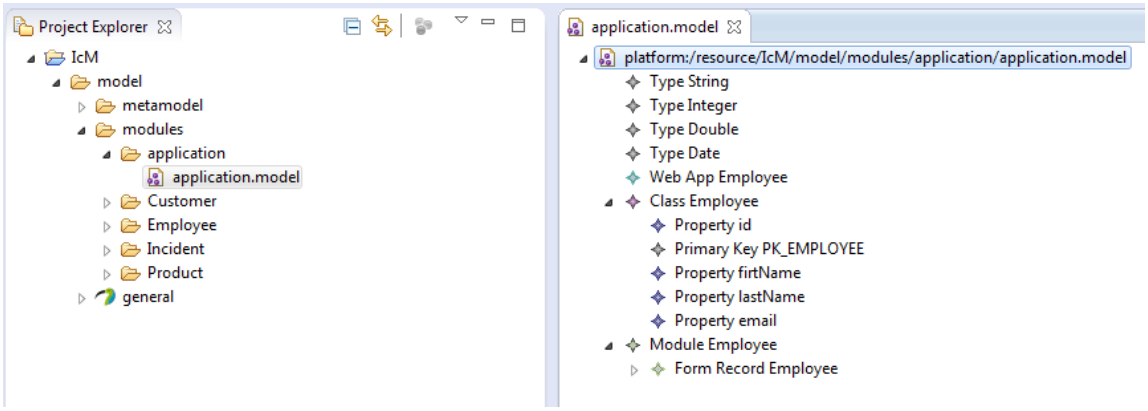


Ilustración 34: Modelo de Aplicación Generado.

6.1.5. GENERAR MODELO DE PLATAFORMA

En esta acción el componente genera el modelo de plataforma donde el usuario debe registrar toda la información no funcional del sistema, esta acción se realiza por medio del menú contextual del plugin de eclipse como se observa en la ilustración 35. Este modelo se necesita para llevar a cabo la generación del código y se dio como insumo inicial de este trabajo de grado y hace parte del proyecto macro Metáfora.

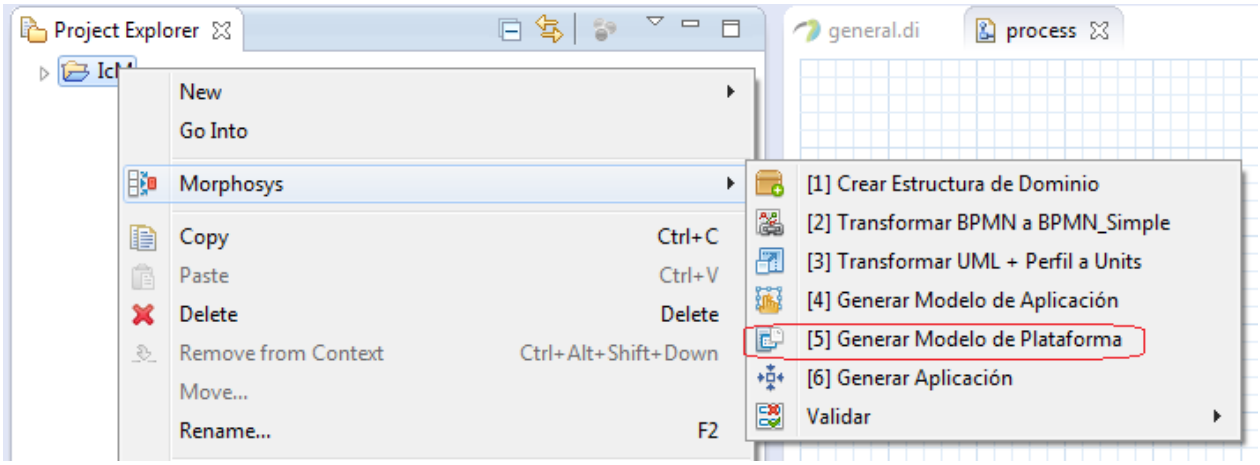


Ilustración 35: Opción en Morphosys para crear modelo de plataforma.

Al realizar esta acción el plugin genera un modelo del proyecto llamado *platform.model* el cual representa la información no funcional del sistema como se puede ver en la ilustración 36. Adicional crea un modelo de enlace entre el modelo de aplicación y el modelo de plataforma.

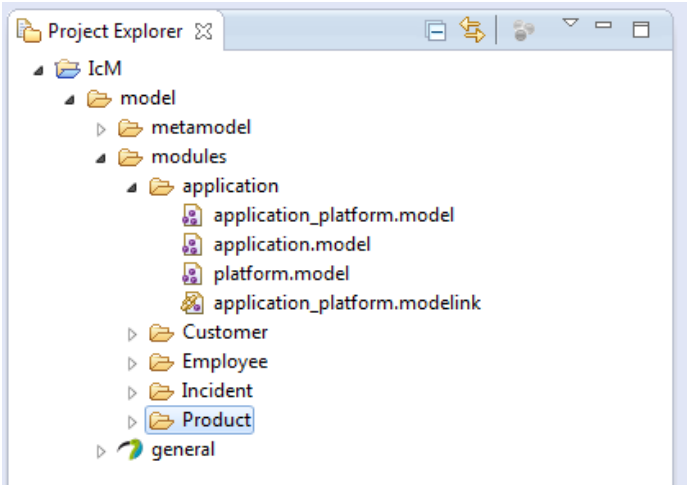


Ilustración 36: Resultado en Morphosys para crear modelo de plataforma.

6.1.6. GENERAR APLICACIÓN

Como pasó final cuando ya se tiene los modelos generados y enlazados donde se suministró toda la información relevante desde las diferentes vistas de arquitectura por parte del usuario se procede a generar la aplicación por medio de una transformación de modelo a texto, como se muestra en la ilustración 37. Este aspecto de poder mezclar el código realizado por los desarrolladores y el código generado por el wizard se cubre utilizando regiones protegidas donde se pueden realizar cambios sobre el modelo y regenerar nuevamente la aplicación sin afectar el código ya desarrollado por los analistas.

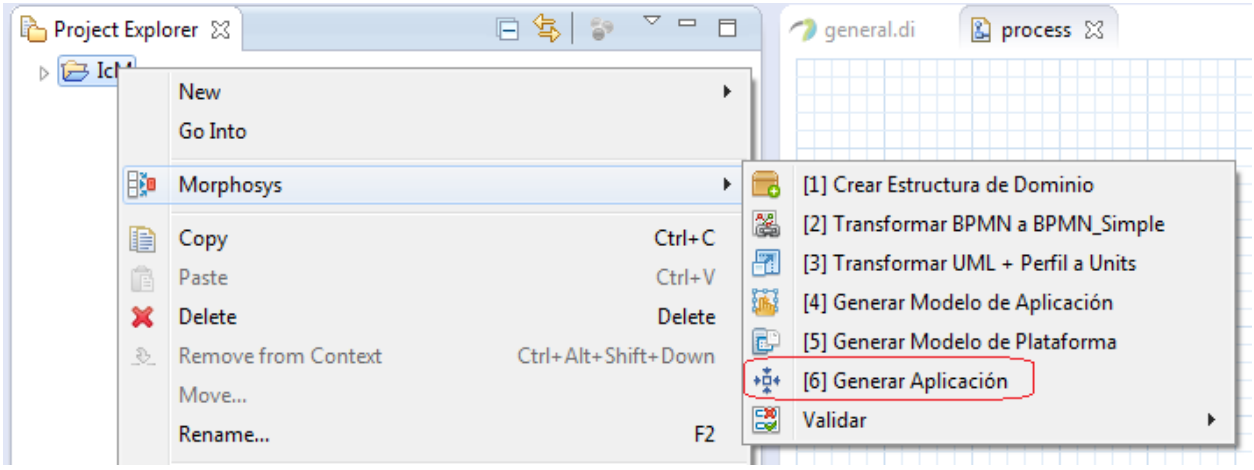


Ilustración 37: Opción en Morphosys para generar la aplicación.

Para validar el proceso de generación se escogió una arquitectura de referencia sobre una plataforma tecnológica como se visualiza en la ilustración 38.

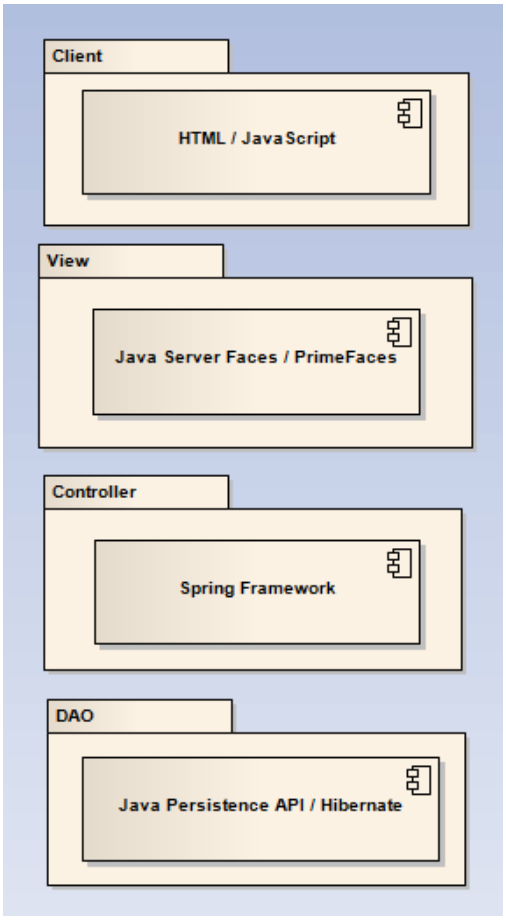


Ilustración 38: Arquitectura de referencia para el caso de estudio.

- ✓ **Plataforma Tecnológica:** se escogió JAVA debido a la robustez, documentación, popularidad y uso en nuestro entorno nacional.

- ✓ **Arquitectura:** se creó una arquitectura utilizando algunos patrones de JEE.

    - Vista Controlador (MVC, por sus siglas en ingles). Utilizado para separar responsabilidades entre capas de los aplicativos.
    - Singleton, utilizado para tener una sola instancia de un componente en una aplicación por motivos de control y manejo de recursos.
    - Data Access Object, sirve para abstraer el acceso a los datos dentro de las capa de servicios de una aplicación.
    - Object Relation Mapping, realiza la correcta interpretación entre el modelo objetual de una aplicación y una fuente de datos relacional.

6.1.6.1. SEMANTICA DE GENERACIÓN

Con los modelos de aplicación y plataforma completos y consistentes se determinan las siguientes reglas para la generación de la aplicación para crear las plantillas de generación en el lenguaje de Epsilon EGL:

- Las clases del modelo de aplicación se utilizan para generar:

    - ✓ Página HTML para gestión de la entidad.
    - ✓ Página HTML Tipo Lista para gestión de la entidad desde otras pantallas.
    - ✓ Clase de entidad para el transporte de información a través de las capas.
    - ✓ Clase de Control de acuerdo al patrón MVC para la clase de entidad.
    - ✓ Clase de Apoyo para el manejo de paginación a base de datos de la entidad.

- ✓ Creación de la sentencia de creación de la tabla para base de datos.

- Las clases (*Master Detail*) del modelo de aplicación se utiliza para generar:

  - ✓ Una página maestro detalle con respecto a las entidades relacionadas que son de tipo *List* o *Form*.

- Los atributos (Property) y los objetos de las clases del modelo de aplicación se utiliza para:

  - ✓ Crear los campos en las páginas HTML y se realiza generan las validaciones respectivas de acuerdo al tipo de dato del atributo.
  - ✓ Creación de la columna de la tabla para el script de base de datos.

- Los módulos del modelo de aplicación se utilizan para:

  - ✓ Generar las clases en la capa de servicio.
  - ✓ Generar el menú principal de la aplicación.

- Las clases ForeingKey del modelo de aplicación se utiliza para:

  - ✓ Generar las llaves foráneas en el script de base de datos.
  - ✓ Generar ventanas emergentes o listas de selección en las páginas HTML.

- Las clases PrimaryKey del modelo de aplicación se utiliza para:

  - ✓ Generar las llaves primarias en el script de base de datos.

6.1.6.2. ARTEFACTOS DE APLICACIÓN

Al realizar el proceso de generación de modelo a texto (M2T) se crea una aplicación en Eclipse de naturaleza web con la estructura de acuerdo al modelo de plataforma como se observa en la ilustración 39 y 40.

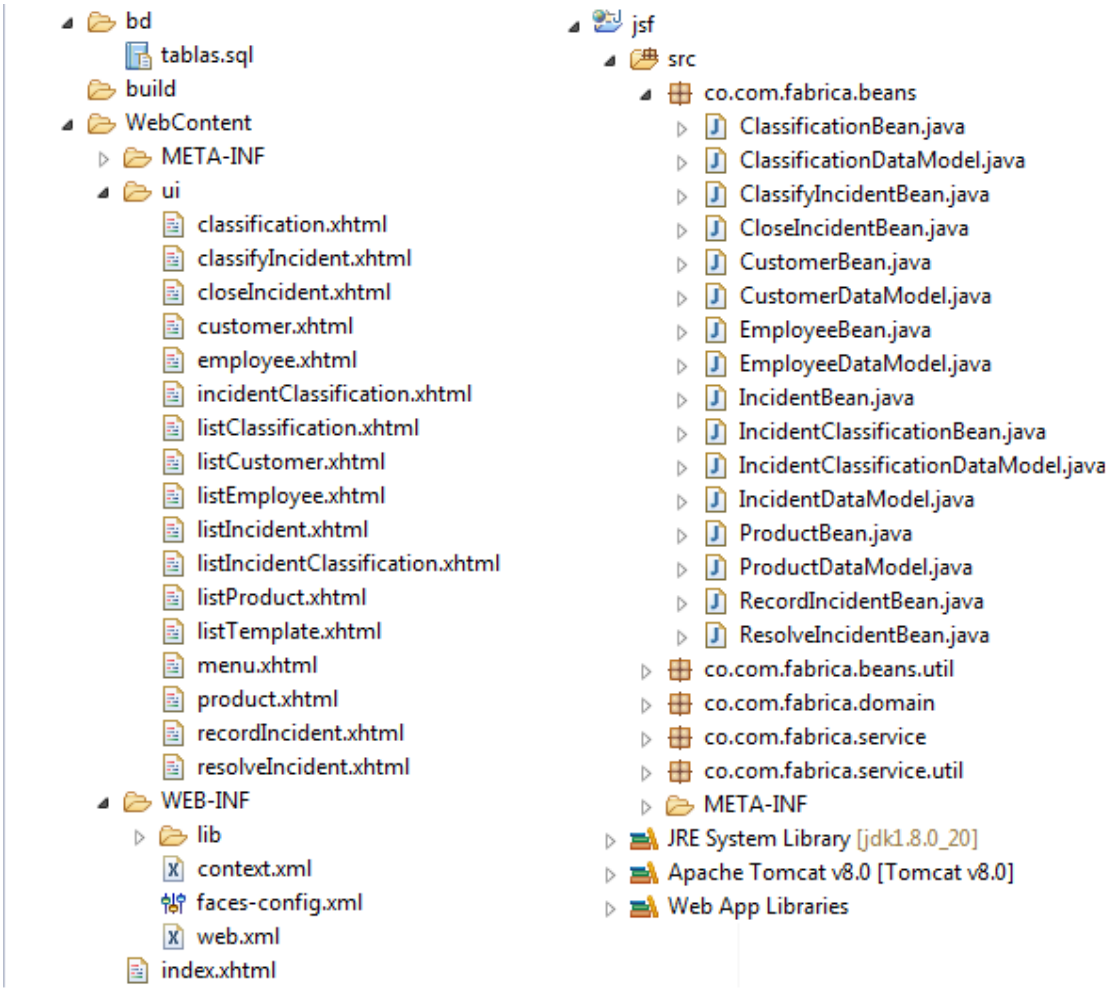


Ilustración 39: Artefactos generados de acuerdo a la arquitectura definida.

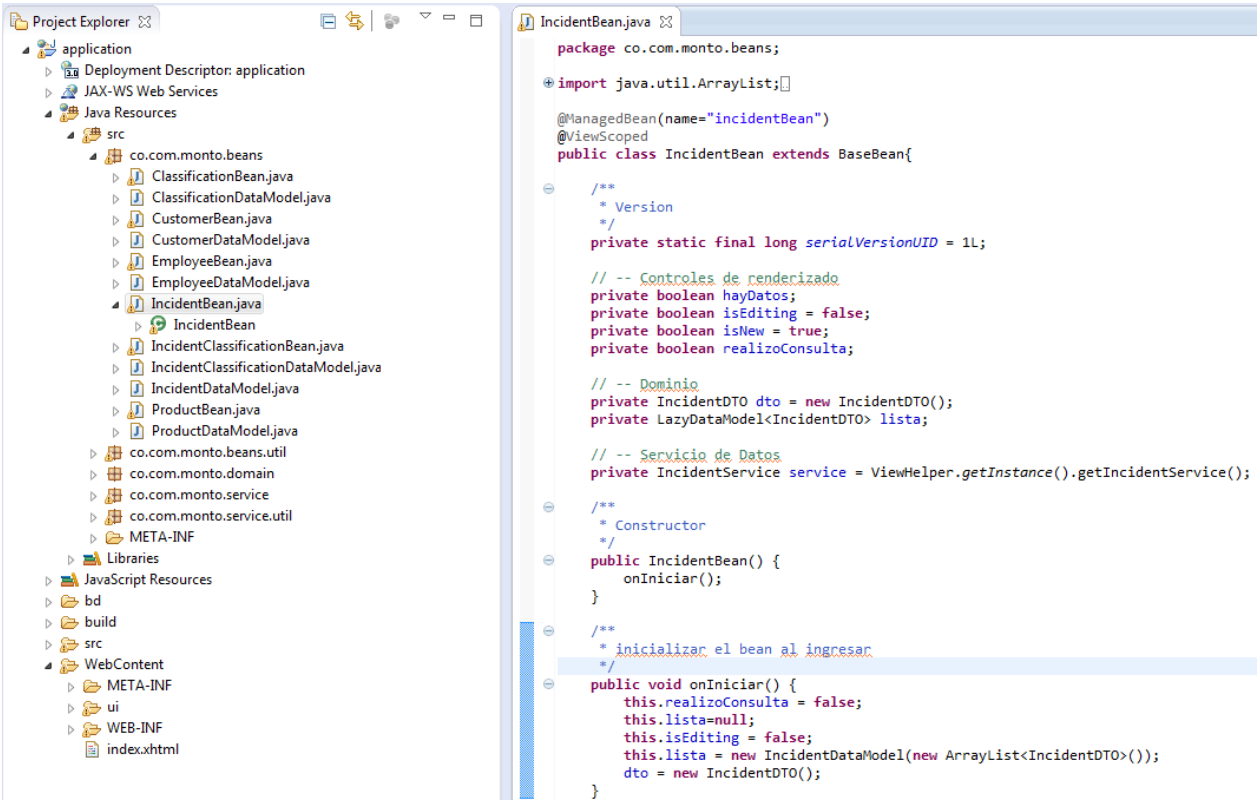


Ilustración 40: Código Fuente Generado

Otra de las oportunidades que ofrece la generación, es que también se pueden generar otros tipos de artefactos como el script de base de datos. Para la arquitectura se seleccionó una base de datos PosgreSql 9.4.

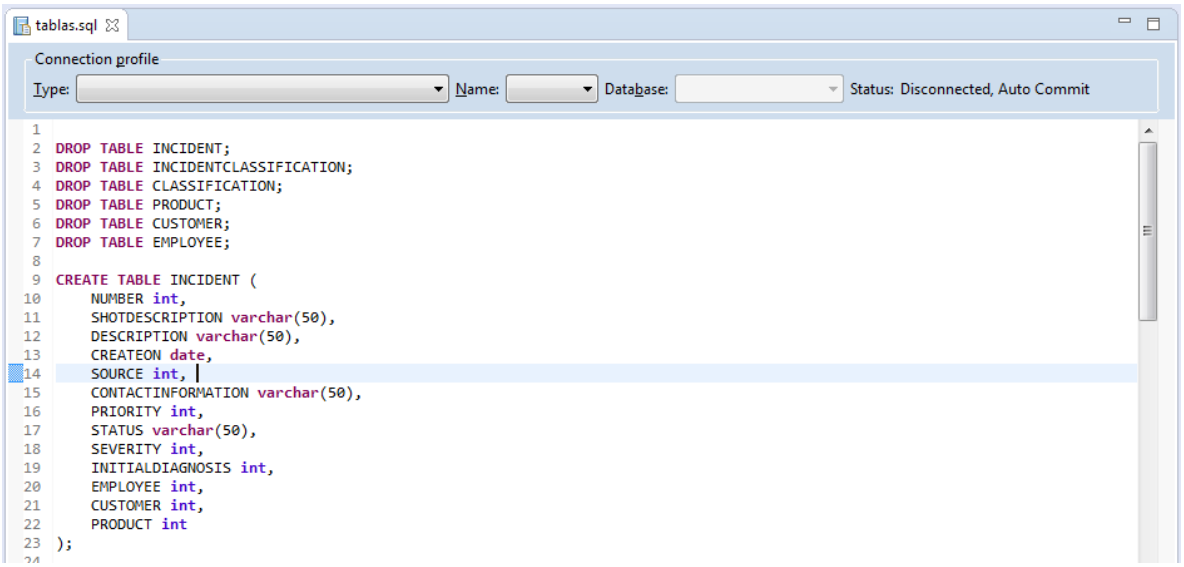


Ilustración 41: Script generado de base de datos.

Al realizar la ejecución del Script en base de datos que se muestra en la ilustración 41 y realizar el despliegue del aplicativo en un servidor de aplicación Tomcat 8.0. Se puede observar el correcto funcionamiento de la aplicación que se observa en la ilustración 42 y la ilustración 43.

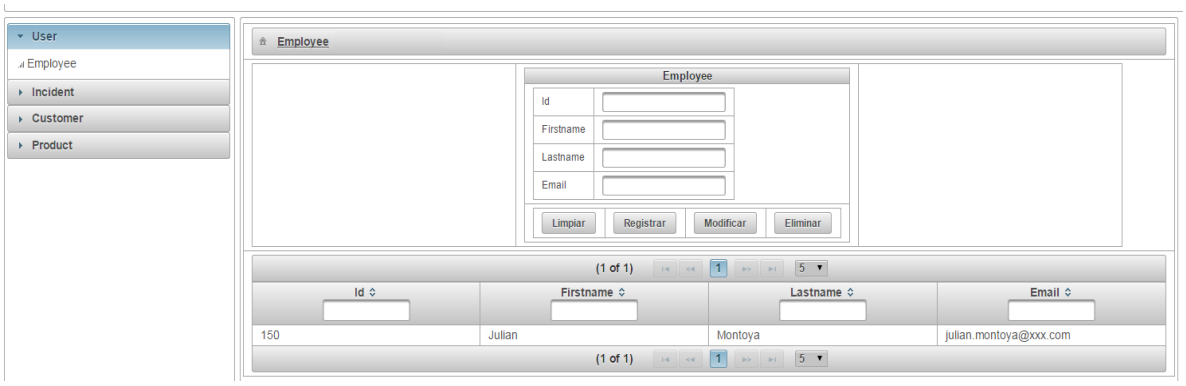


Ilustración 42: Pantalla administrativa de la aplicación generada.

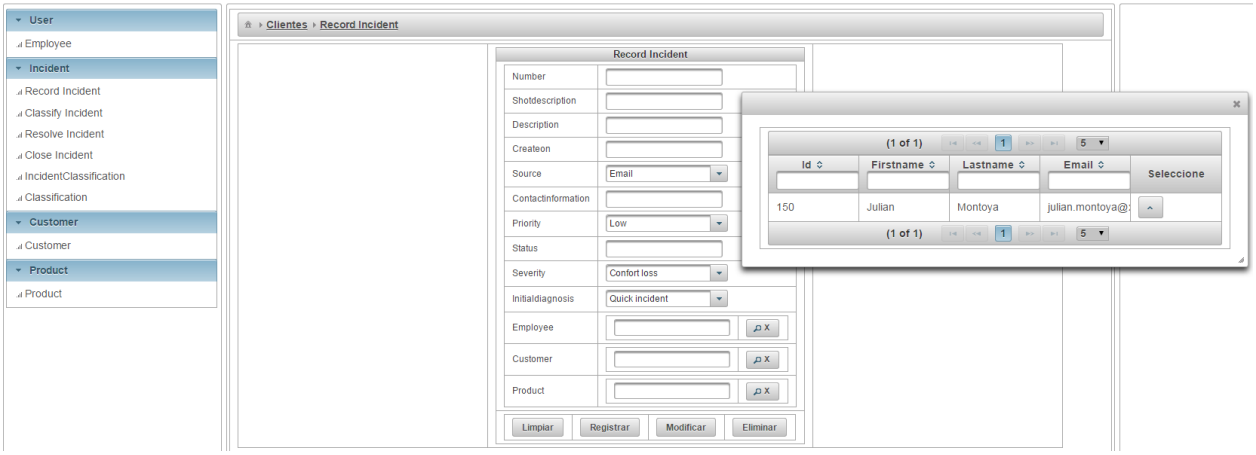


Ilustración 43: Pantalla de gestión de incidentes de la aplicación generada.

7. ANÁLISIS DEL CARÁCTER GENÉRICO DE LAS TRANSFORMACIONES

Para darle al mecanismo de transformación el carácter de genérico, la aplicabilidad de la herramienta tendría que ir más allá del caso de estudio. Se debe evidenciar la posibilidad de generar sistemas de información utilizando diferentes modelos, secuencias de generación y arquitecturas de referencia. Para esto inicialmente se debe configurar la ruta del repositorio como se visualiza en la ilustración 44.

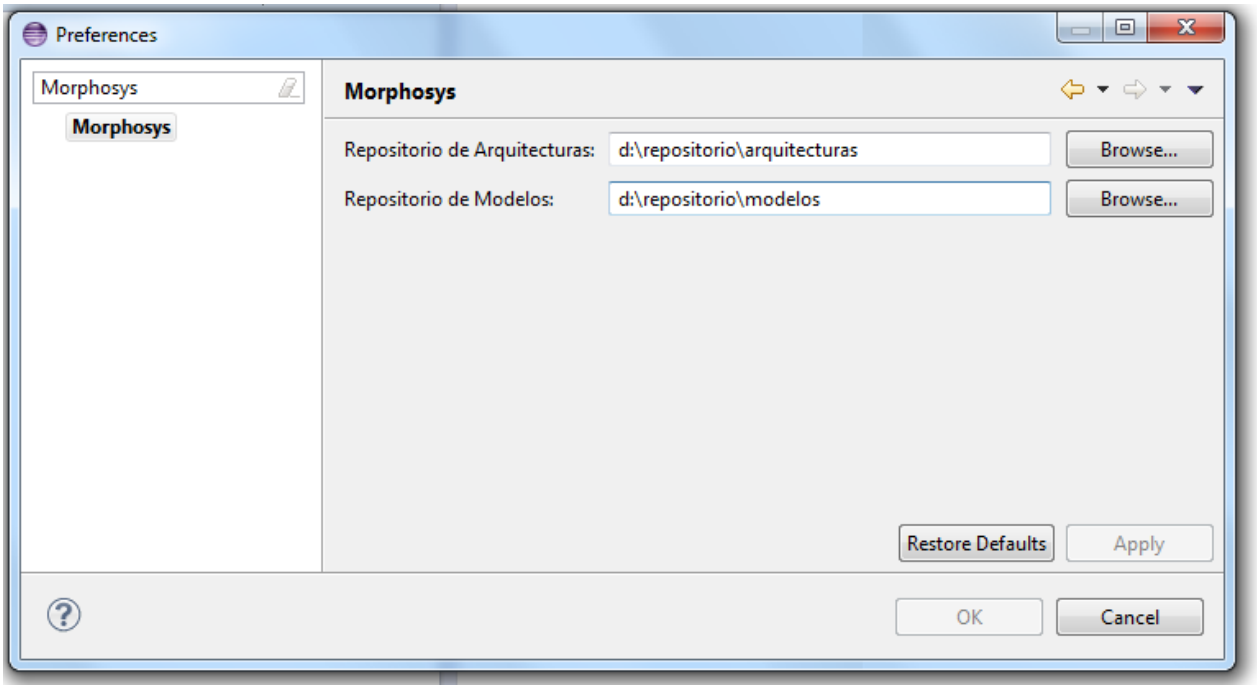


Ilustración 44: Configuración del Repositorio.

Para tener una visión general de cómo opera el plugin y como da la posibilidad al usuario de modificar la forma de generación de aplicaciones se van a explicar las funcionalidades de una manera detallada. Para realizar la generación de una aplicación se deben realizar los siguientes pasos:

- Cargar los archivos de tipo Epsilon, como son archivos de validación y archivos de transformación. Estos archivos se usan como insumos para crear los pasos o secuencias de transformación de los modelos. Estos archivos se crean en Eclipse con la ayuda del plugin de Epsilon. La carga de estos archivos se debe realizar como se puede ver en la ilustración 45 y 46.

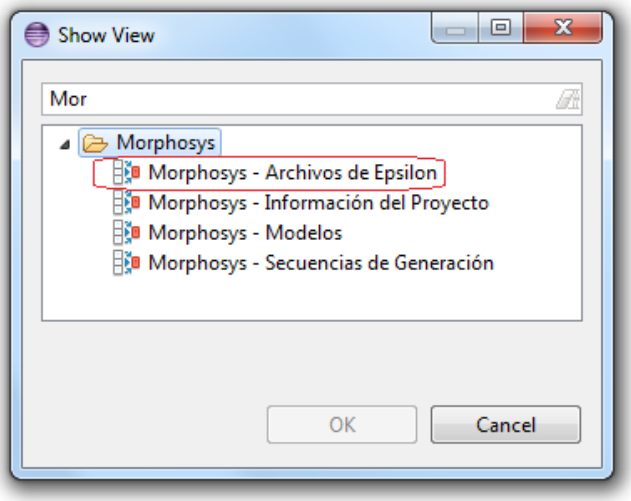


Ilustración 45: Búsqueda de la vista, Archivos Epsilon.

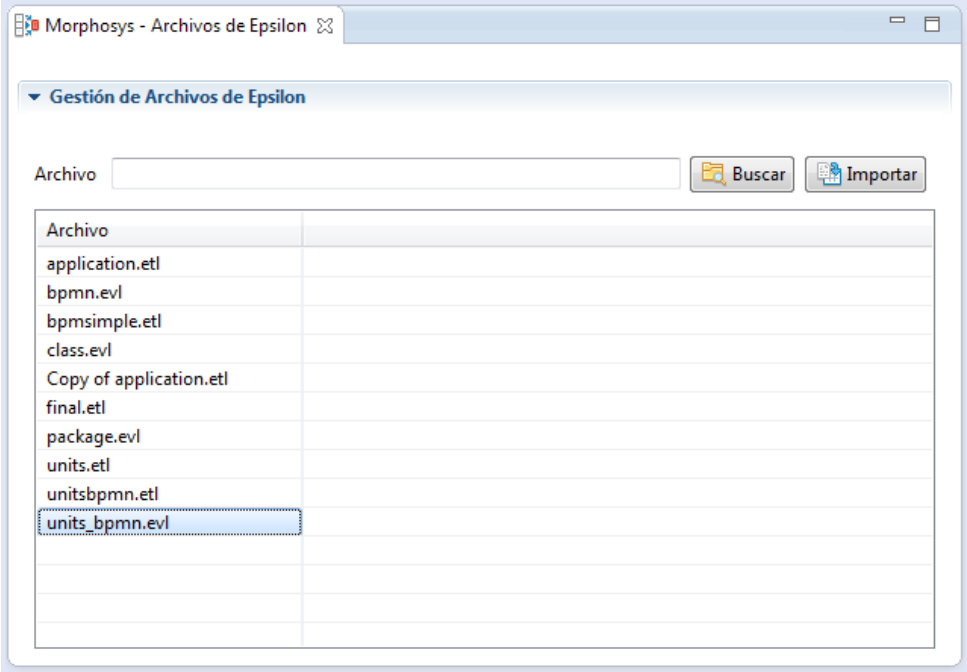


Ilustración 46: Vista de Gestión de Archivos de Epsilon.

- Cargar los metamodelos, para poder realizar secuencias de generación se debe registrar los tipos a utilizar como los modelos de clases e importar modelos propios como Units. La carga de los metamodelos y las instancias se realiza por medio de la vista de modelos como se muestra en la ilustración 47 y 48.

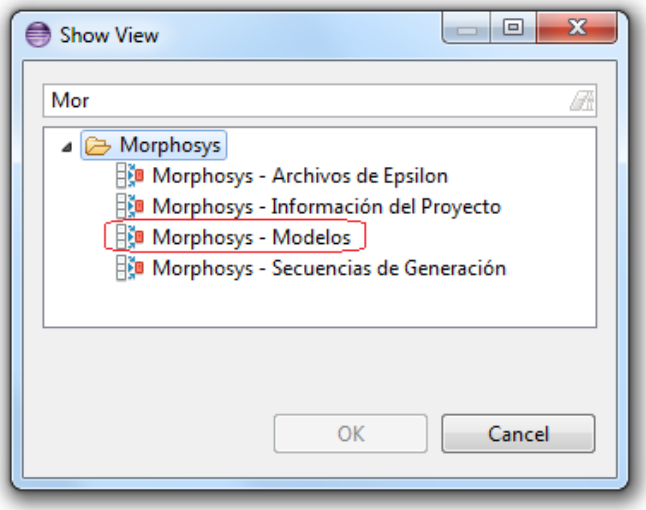


Ilustración 47: Búsqueda de la vista, Modelos de Morphosys.

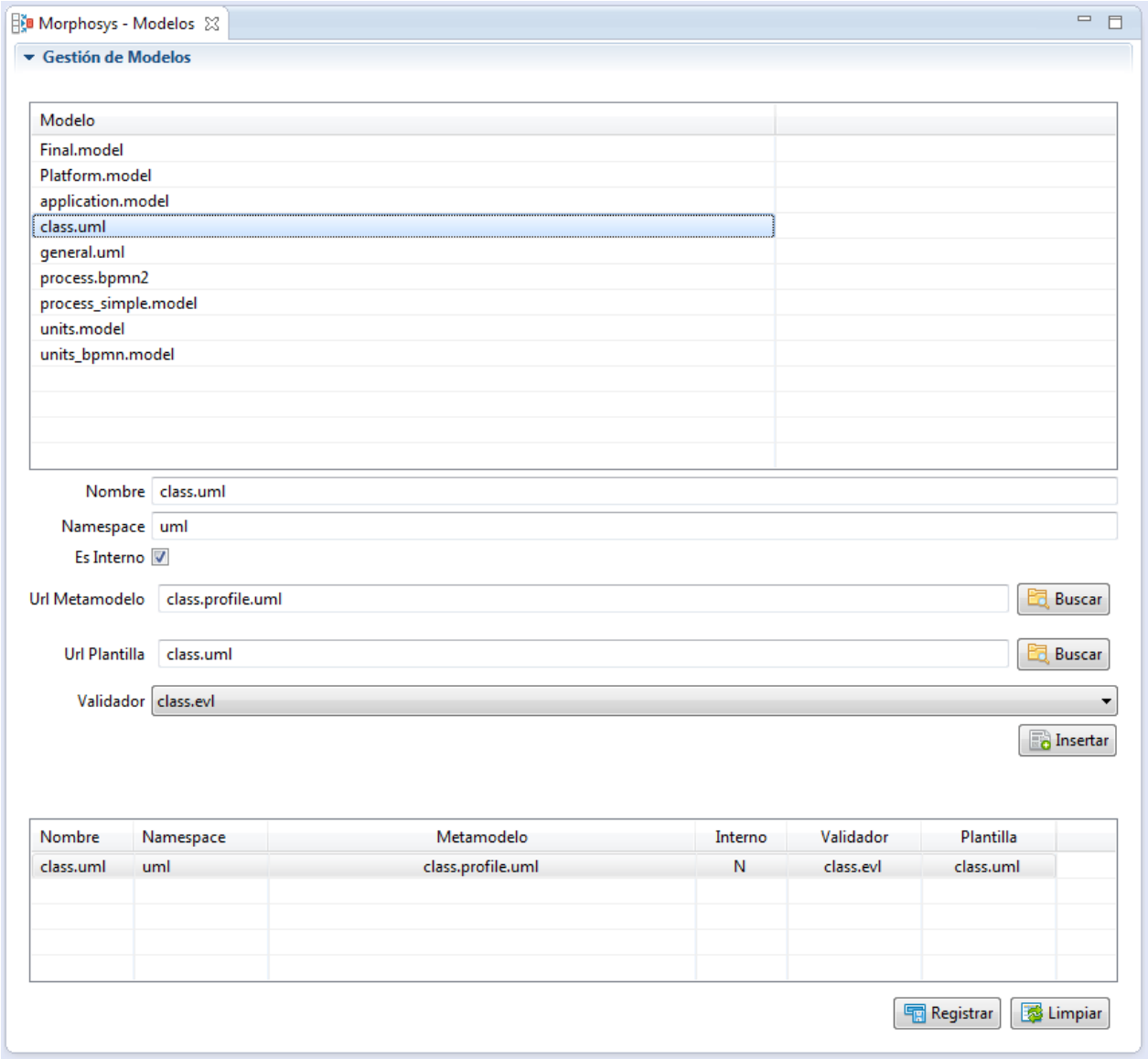


Ilustración 48: Vista de Gestión de Modelos de Epsilon.

Se debe aclarar que se tiene la posibilidad de cargar una instancia vacía del metamodelo a cargar para guiar al usuario y facilitar el modelado de la aplicación, a este modelo lo llamamos plantilla.

- Crear las secuencias de generación, con dichas series se garantiza la posibilidad de guiar al usuario con respecto a que modelos se utilizaran durante el desarrollo, adicionalmente se indicará que validaciones y transformaciones se realizan contra los modelos. La configuración de las secuencias se debe realizar como se puede ver en la ilustración 49 y 50.

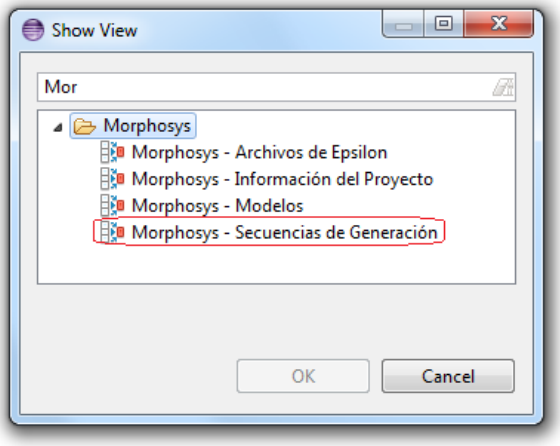


Ilustración 49: Búsqueda de la vista, Secuencias de Generación de Epsilon

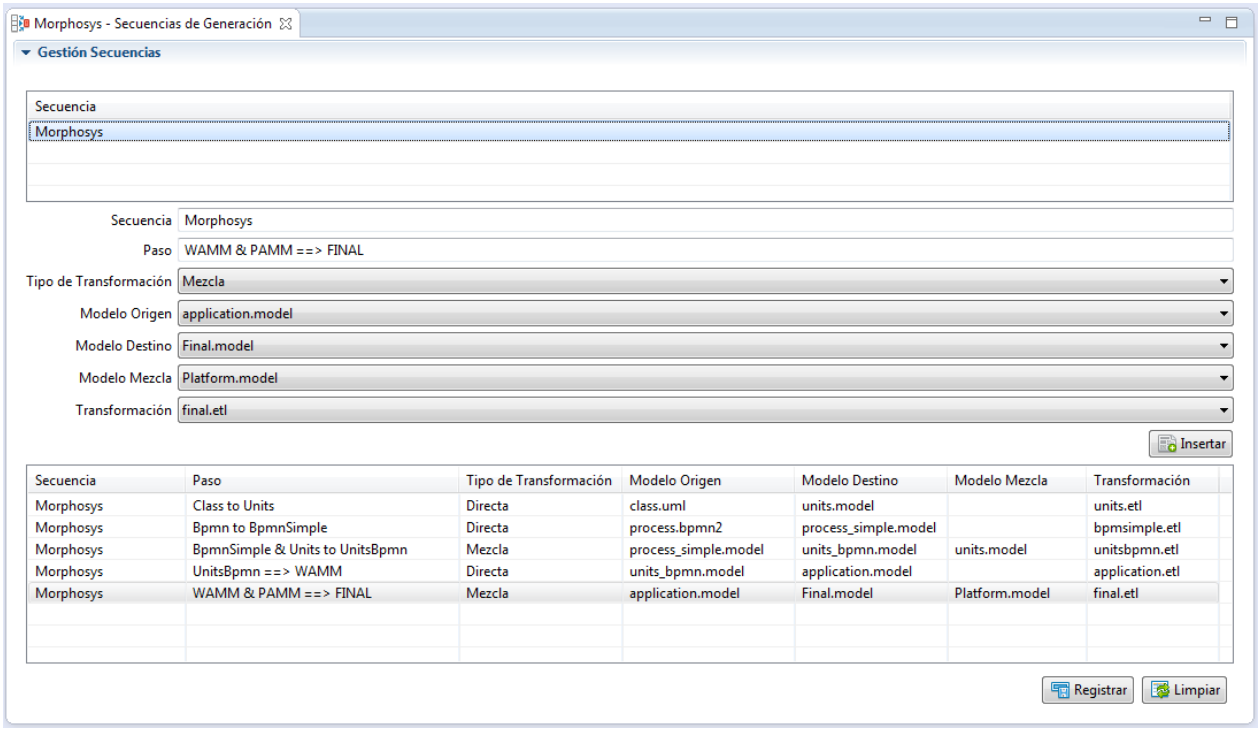


Ilustración 50: Gestión de Secuencias de Generación.

- Luego de tener una secuencia de generación definida. Se procede a generar una aplicación de tipo Morphosys donde se selecciona la secuencia antes configurada como se puede visualizar en la ilustración 51.

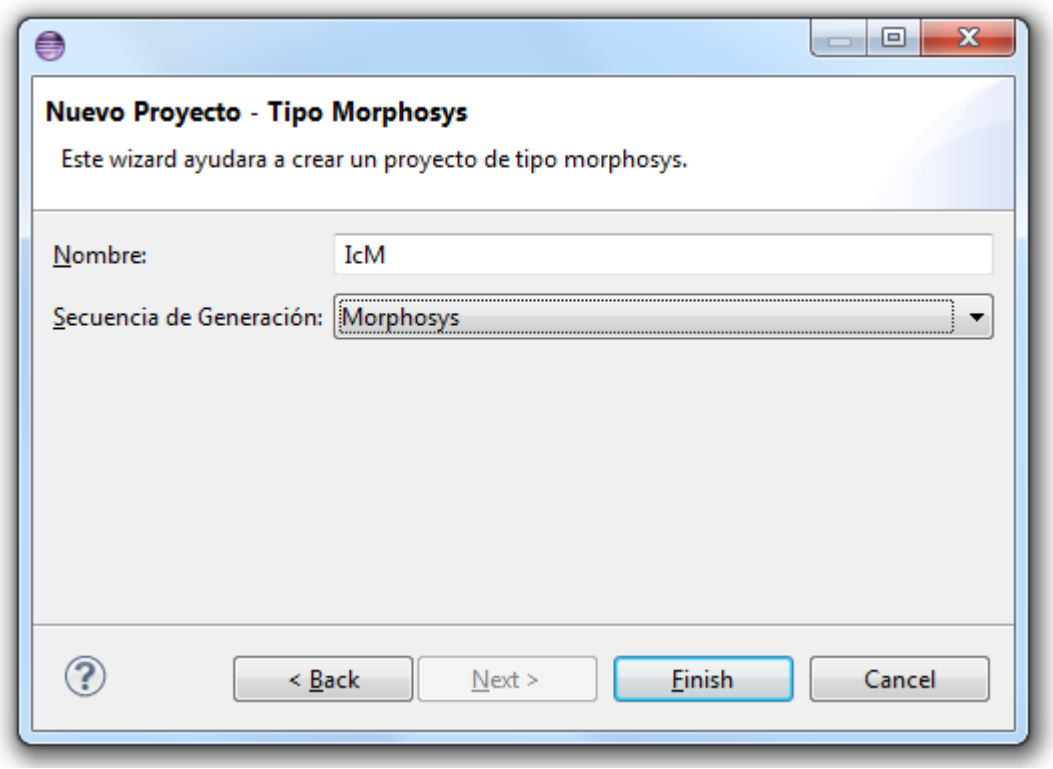


Ilustración 51: Creación de Aplicaciones de Modelado de Tipo Morphosys.

Al crear la aplicación ya se cuenta con el apoyo necesario de parte del plugin para validar y generar los modelos de acuerdo a la secuencia seleccionada como se puede visualizar en la ilustración 52.

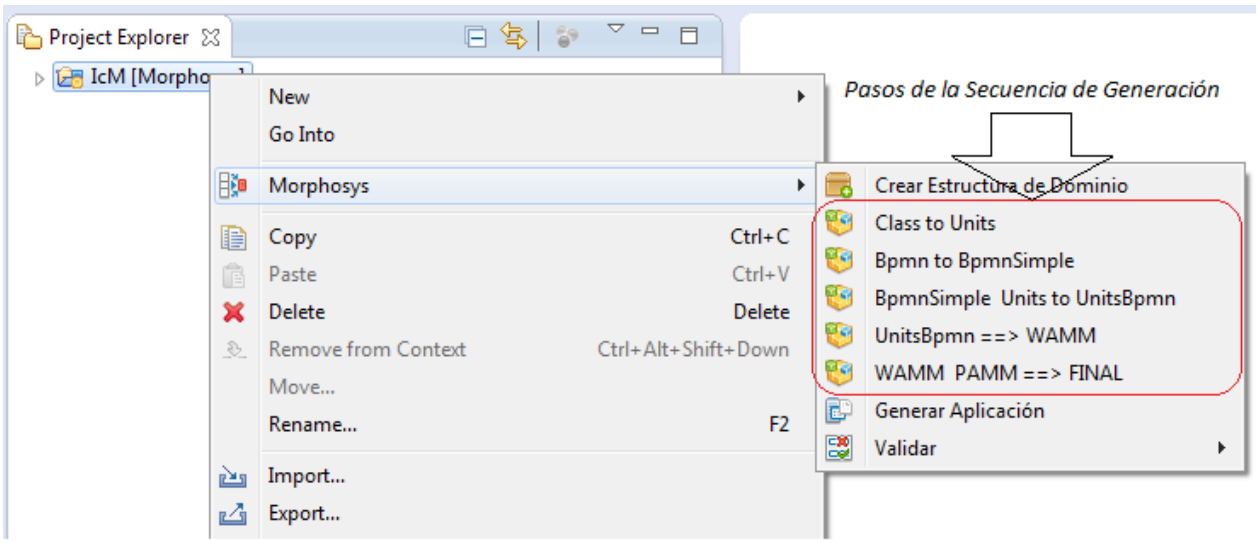


Ilustración 52: Visualización de Pasos de la Secuencia de Generación.

Un punto importante que está garantizando el aspecto genérico del desarrollo es la posibilidad de crear diferentes secuencias de transformación. Para efectos de este trabajo hay dos tipos de transformaciones:

- Transformación Frontal: este tipo de transformación se usa para convertir de manera automática por medio del plugin un modelo origen a un modelo destino sin la intervención del usuario.

  Para realizar un tipo de transformación frontal se deben tener en cuenta los siguientes pasos:

  1. Registrar el modelo Origen en el plugin.
  2. Registrar el modelo Destino en el plugin.
  3. Realizar la trasformación del modelo origen al modelo destino por medio del lenguaje ETL.
  4. Registrar el archivo con la transformación realizada en el punto 3 en el plugin.
  5. Como último paso se debe parametrizar el paso en una secuencia de generación. Donde se debe indicar el modelo origen, archivo de transformación y modelo destino.

- Transformación de Tipo Mezcla: es este tipo de transformaciones, el plugin crea los modelos a partir de plantillas, pero es el usuario final es quien realiza la mezcla de componentes en los modelos.

  1. Registrar el modelo Uno en el plugin.
  2. Registrar el modelo Dos en el plugin.
  3. Registrar el modelo Weaving que mezcla los modelos Uno y Dos.
  4. Realizar la trasformación del modelo origen al modelo destino por medio del lenguaje ETL.
  5. Registrar el archivo con la transformación realizada en el punto 4 en el plugin.
  6. Como último paso se debe parametrizar el paso en una secuencia de generación. Donde se debe indicar el modelo uno, modelo dos, modelo de weaving o de mezcla, archivo de transformación y modelo destino.

El plugin da la posibilidad al usuario de configurar las secuencias de generación para su posterior uso. Al momento de la creación de un proyecto de tipo Morphosys, se debe seleccionar en el asistente el tipo de secuencia a utilizar. Para el caso de estudio IcM tenemos la siguiente secuencia de transformación configurada.

Pasos:

1. Transformar una instancia del modelo BPMN2 a una instancia del modelo Simple_BPMN (Frontal).
2. Transformar una instancia del modelo de paquetes y una instancia del modelo de clases en una instancia del modelo Units (Frontal).
3. Transformar una instancia del modelo Units y Simple_BPMN en una instancia del modelo UnitsBpmn. (Mezcla).
4. Transformar las instancias de los modelos Units, Simple_BPMN y UnitsBpmn en una instancia de modelo de aplicación.
5. Transformar las instancias de los modelos de Aplicación y de Plataforma en una instancia del modelo Final (Mezcla).
6. Transformar las instancias de los modelos de Aplicación, de Plataforma y Final en todos los artefactos necesarios que hacen parte del sistema de información.

Adicionalmente a la posibilidad de escoger la secuencia de transformación, se agrega otro componente que aporta en el aspecto genérico, es la capacidad de agregar nuevos modelos al abanico de posibilidades ya existentes que actualmente ofrece Morphosys como se visualiza en la ilustración 46, los modelos iniciales son:

- Modelo de Clases.
- Modelo simplificado de BPMN.
- Modelo Units.
- Modelo UnitsBPMN.
- Modelo conceptual.
- Modelo de Plataforma.
- Modelo Final

Epsilon es una familia de lenguajes y herramientas utilizada para realizar validaciones, transformaciones, comparaciones de modelos y generación de código. El prototipo Morphosys realiza los enlaces de manera automática de los modelos de acuerdo a la secuencia de transformación seleccionada. Para realizar los enlaces Morphosys se apoya en la herramienta de Epsilon llamada *ModeLink,* esta herramienta permite visualizar y mezclar componentes de dos o tres modelos en la misma pantalla.

Con la posibilidad que se le ofrece al usuario para modificar las transformaciones de tipo Mezcla como se muestra en la ilustración 28, se identifica claramente que se puede controlar directamente el proceso de transformación aportando más flexibilidad al proceso iterativo de generación.

Finalmente lo que respecta a las arquitecturas de referencia son plantillas y archivos de Epsilon de tipo EGL utilizado para generar una aplicación como paso final. Cada carpeta en el repositorio de arquitecturas tiene un archivo llamado application.egl como se ve en la ilustración 53, dicho archivo es el punto de partida que invoca el plugin para generar la solución de software.

Ilustración 53: Estructura de la Arquitectura de Referencia.

Dentro del proyecto metáfora [8] se hace referencia a un entorno de desarrollo genérico, donde este permita reutilizar los modelos. Donde sirva como referencia para generar otro tipo de aplicaciones diferentes a la web. Estas características se cumplen en el desarrollo y poseen el componente genérico debido a la gran variedad de configuraciones que se pueden usar para cualquier tipo de aplicación basándose en múltiples modelos. Se puede decir que es un mecanismo genérico para definir transformaciones porque tengo la posibilidad de configurar una secuencia de generación o transformación de modelos y determinar que modelos y transformaciones hacen parte del desarrollo del aplicativo en particular; teniendo en cuenta que también se puede reutilizar cualquier arquitectura de referencia que corresponda al modelo final de generación de un proyecto de tipo Morphosys.

8. CONCLUSIONES Y TRABAJO FUTURO

El desarrollo iterativo a través de modelos hasta llegar al código fuente evidencia las siguientes ventajas como metodología de desarrollo [47]:

El trabajo de construir un DSL mejora la calidad de los modelos y motiva a los analistas a su uso debido a la facilidad descriptiva del sistema que se está modelando [24][40]. Adicionalmente con el uso perfiles se potencia la expresividad dando más contenido semántico a los modelos y demostrar que es un apoyo indispensable a la hora de realizar las trasformación de manera efectiva [23][13].

Los modelos en la aplicación toman un rol principal debido a que nunca se va a des actualizar porque el desarrollo iniciara siempre desde la documentación facilitando temas como el entendimiento y mantenimiento de las aplicaciones desde el punto de vista del analista funcional y analista desarrollador[23][15] también se debe resaltar la posible reutilización de la información contenida en ellos debido a que los modelos como el de aplicación tienen toda la información funcional de la aplicación y podría ser utilizado para crear el mismo aplicativo sobre diferentes plataformas de desarrollo [44].

Cuando se realizan las plantillas para generar una arquitectura en particular, se visualiza la velocidad de desarrollo debido a que sin importar cuán grande es el modelo, el aplicativo se genera con la misma calidad y facilidad plasmada en los archivos de generación EGL. Como la generación completa cubre un alto porcentaje desde los modelos, es necesario realizar algo de desarrollo para obtener el 100% de la funcionalidad.

Se debe resaltar que Épsilon como plataforma para la trasformación brindo todas las posibilidades necesarias para realizar validación, trasformación, mezclado de modelos y generación de modelos a texto. Adicionalmente está dotado de vistas particulares donde se pueden realizar la mezcla de modelos que fue indispensable para realizar este trabajo.

Una de las ventajas en el desarrollo dirigido por modelos es la posibilidad de realizar plantillas con las mejores prácticas de desarrollo [49], con esto se está garantizando calidad y todo el gobierno posible desde el punto de vista arquitectónico debido a que se cubren todos los aspectos funcionales y no funcionales de la aplicación [8][25]. Para el caso de estudio se abre la posibilidad de crear nuevas plantillas de generación de modelo a texto para otras tecnologías como PHP y .NET donde se pueden facilitar su proceso debido a que ya se tiene un camino recorrido con el trabajo realizado en Java.

Con la creación del plugin se dio una mejor forma de realizar el proceso de desarrollo guiando al usuario debido a que el aplicativo generaba y enlaza los artefactos si cargar al usuario de trabajo repetitivo y expuesto a errores. Adicionalmente se realizaron todas las validaciones necesarias para que el usuario no tuviera que entender directamente el lenguaje de metamodelos de los productos utilizados como Papyrus, Epsilon y BPMN2 entregando mensajes entendibles al usuario.

Para *trabajos futuros* se tienen las siguientes sugerencias para seguir mejorando el desarrollo de aplicaciones bajo el enfoque de Metáfora:

Dentro del proceso de desarrollo de Morphosys tener la posibilidad de enlazar elementos de diferentes paquetes de dominio para que se observará la relación completa de las entidades.

Al generar las aplicaciones poder versionarlas directamente en sistemas de control de versiones como SVN y GIT. También es de vital importancia que se pueda integrar con herramientas de integración continua como *Jenkins* y pulir aún más el ciclo de desarrollo de las aplicaciones mejorando la velocidad en las etapas de validación y pruebas.

Para las nuevas arquitecturas de referencia creadas sería muy interesante realizar generación de reglas de negocio sobre algún sistema de administración (BRMS, por

sus siglas en inglés) de igual manera se podría realizar la generación de la capa de persistencia para aplicación que usen bases de datos NoSql.

Crear un manual de usuario tanto para los usuarios encargados para generar las arquitecturas como para aquellos encargados de modelar la aplicación.

Estas aclaraciones se realizan porque no están dentro del alcance del presente trabajo pero se pueden utilizar como lineamientos para fortalecer el plugin de Morphosys y por consiguiente el macro proyecto Metáfora:.

Se debería evaluar la posibilidad de usar e integrar *Epsilon Wizard Languaje* [50]. Con su facilidad de implementación podría absorber funcionalidades básicas e intermedias sin tener la necesidad de realizar desarrollos en el plugin de Morphosys.

Durante el desarrollo del trabajo se realizaron una variedad de actividades relacionadas directamente con los objetivos propuestos para este trabajo.

- Objetivo 1. Realizar una revisión del estado del arte de la transformación de modelos en la ingeniería dirigida por modelos.

  - ✓ Síntesis de actividades realizadas

    - Se realizó una exploración en las fuentes de información científica disponibles en la página de la universidad de Medellín (http://www.udem.edu.co/index.php/2012-11-23-20-25-35/bases-de-datos). Adicionalmente se utilizó google (https://scholar.google.com/).
    - Se utilizó información relevante de libros referenciados en la bibliografía tanto para tener una base teórica de las transformaciones de modelos como para la construcción del plugin de Eclipse.

  - ✓ Síntesis de resultados alcanzados

- Se cumplió con la curva de aprendizaje del plugin de Eclipse Epsilon con el cual se realizan las validaciones, comparaciones, mezcla y transformación de modelos.
- La generación de la línea base del proyecto Java del asistente de generación fue posible como parte del estudio de la plataforma de desarrollo de Eclipse (EPD, por sus siglas en inglés).

- Objetivo 2. Caracterizar las transformaciones de modelo a modelo en un enfoque arquitectónico multivistas en el marco de la Ingeniería Dirigida por Modelos.

  ✓ Síntesis de actividades realizadas

  - Se realizó la clasificación de las características de los elementos de los diagramas a utilizar dentro del proceso iterativo de generación.
  - Para poder establecer los pasos del proceso iterativo, se estableció el orden y la forma como se debían mezclar las instancias de los modelos.
  - Al establecer la forma como se debían realizar las transformaciones se evidencio la necesidad de crear modelos de enlace que hicieran posible el proceso de fusión.

  ✓ Síntesis de resultados alcanzados

  - Se crearon los nuevos modelos Units, UnitsBpmn y Simple_BPMN como parte del proceso para apoyar las transformaciones de modelos.
  - Se generaron las reglas donde se establecía en cada uno de los pasos como se iban a mezclar los modelos.
  - En cada uno de los pasos de los procesos se generaron las reglas de validación, mezcla y transformación de modelos.
  - Se desarrolló en el plugin el comportamiento básico de las transformaciones de acuerdo al proceso de generación previamente establecido.

- Objetivo 3. Proponer un esquema genérico de parametrización para las transformaciones de modelo a modelo.

  ✓ Síntesis de actividades realizadas

  - Se validó la forma de como externalizar y parametrizar el repositorio de arquitecturas, esta característica ofrece la posibilidad al usuario de adicionar, modificar o eliminar arquitecturas de referencia.

- Se especificaron las reglas para poder gestionar los modelos desde el *plugin* de Eclipse. El usuario tiene la posibilidad de adicionar y eliminar modelos al repositorio para crear secuencias de generación totalmente nuevas y de una manera flexible. Los repositorios se deben configurar previamente desde la herramienta como se muestra en la ilustración 43.

✓ Síntesis de resultados alcanzados

- Se logró determinar todos los aspectos funcionales que debía tener el prototipo para garantizar las características genéricas deseadas.

- Objetivo 4. Construir un prototipo que implemente el mecanismo de transformación e integrarlo con un entorno de generación de aplicaciones.

✓ Síntesis de actividades realizadas

- Se afino el plugin de Eclipse con todas las reglas y capacidades antes establecidas para el proceso iterativo de desarrollo.
- Se especificaron todas las reglas de validación y transformación necesarias para todos los pasos en la generación de la aplicación de gestión de incidentes.

✓ Síntesis de resultados alcanzados

- Se logró construir el prototipo que garantiza toda la funcionalidad antes establecida.
- Se plasmaron todas las reglas de validación y transformación para los diferentes modelos en el lenguaje Epsilon que corresponde a la arquitectura de referencia establecida.
- Se realizó la integración del prototipo con el plugin de Epsilon con el propósito de controlar las transformaciones desde el plugin Morphosys.

- Objetivo 5. Validar el mecanismo aplicándolo a un caso de estudio específico.

✓ Síntesis de actividades realizadas

- Se realizó el análisis y diseño del caso de estudio según el proyecto macro Metáfora.
- Se afinaron las plantillas de generación de modelo a texto para mejorar la calidad del código desarrollado.

- ✓ Síntesis de resultados alcanzados

  - Se mejoraron las platillas para garantizar la arquitectura de referencia.
  - Se logró generar el sistema de información de gestión de incidentes por medio de los modelos a través del proceso iterativo de transformación de modelo a modelo y de modelo a texto.
  - Se realizó el afinamiento del plugin de Eclipse con respecto a validaciones de entrada y aspectos de usabilidad.

REFERENCIAS BIBLIOGRÁFICAS

[1] M. D. A. G. Version, A. Kennedy, K. Carter, and W. F. X. Technologies, “MDA Guide Version 1.0.1,” no. June, 2003.

[2] Omg, “Meta Object Facility (MOF) Core Specification,” *Management*, vol. 080907, no. January, pp. 1–76, 2006.

[3] K. H. Russ Miles, *Learning UML 2.0*. O’Reilly Media, 2006.

[4] *Eclipse Rich Client Platform Second Edition*. .

[5] A. Blewitt, “Mastering Eclipse Plug- in Development ‘ Plugging in to JFace and the Common Navigator Framework ’ About the Author,” no. 1.

[6] S. Staab, T. Walter, G. Gr, and F. S. Parreiras, “Model Driven Engineering with Ontology Technologies.”

[7] J. Bernardo-quintero and J. F. Duitama-muñoz, “enfoques centrados en modelos en el desarrollo de software 1 Reflections on the Adoption of Model-Based Approaches for,” vol. 15, no. 1, pp. 219–243, 2011.

[8] J. A. Hincapié, U. De Medellín, A. Ltda, S. A. S. Perceptio, and I. M. S. A. S, “Metáfora : Generación de aplicaciones web y móviles basadas en procesos de negocio , usando ingeniería dirigida por modelos Autores : Juan Bernardo Quintero , Universidad de Medellín Raquel Anaya , Universidad EAFIT Participantes :,” pp. 1–61, 2013.

[9] V. Itil, “ITIL V3 ¿Por dónde empezar?”

[10] R. C. Gronback, *ECLIPSE MODELING PROJECT A Domain-Specific Language Toolkit*. .

[11] M. Salatino and E. Aliverti, *jBPM5 Developer Guide*. 2012.

[12] T. Mens and P. Van Gorp, “A Taxonomy of Model Transformation,” *Electron. Notes Theor. Comput. Sci.*, vol. 152, pp. 125–142, Mar. 2006.

[13] S. Sendall and W. Kozaczynski, “Model Transformation – the Heart and Soul of Model-Driven Software Development,” no. August 2003, pp. 1–12.

[14] A. Vohra and D. Vohra, *Pro XML Development with Java*. 2006.

[15] Y. Martínez, C. Cachero, and S. Meliá, “MDD vs. traditional software development: A practitioner’s subjective perspective,” *Inf. Softw. Technol.*, vol. 55, no. 2, pp. 189–200, Feb. 2013.

[16] R. A, “EL PROCESO DE DESARROLLO DE SOFTWARE,” pp. 131–146, 2007.

[17] N. Aquino, “Adding flexibility in the model-driven engineering of user interfaces,” *Proc. 1st ACM SIGCHI Symp. Eng. Interact. Comput. Syst. - EICS ’09*, p. 329, 2009.

[18] N. Aquino, J. Vanderdonckt, F. Valverde, and O. Pastor, “Using Profiles to Support Model Transformations in the Model-Driven Development of User Interfaces,” pp. 1–12.

[19] O. M. Grassi, C. Pons, F. De Informática, and C. U. Abierta, “Análisis basado en variables para trazabilidad en transformación de modelos,” pp. 134–148, 2012.

[20] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt, “A unifying reference framework for multi-target user interfaces,” pp. 1–21, 1913.

[21] C. De Investigaciones and C. Uda, “DOMINIO Introducción Líneas de investigación y,” pp. 626–630, 2012.

[22] S. Jeary, A. Fouad, and K. Phalp, “Extending the Model Driven Architecture with a pre- CIM level.”

[23] N. Koch, “Transformation techniques in the model-driven development process of UWE,” *Work. Proc. sixth Int. Conf. Web Eng. - ICWE ’06*, p. 3, 2006.

[24] I. Kurtev, J. Bézivin, F. Jouault, P. Valduriez, and A. Inria, “Model-based DSL Frameworks,” pp. 602–615, 2006.

[25] J. Quintero, P. Rucinque, R. Anaya, and G. Piedrahita, “How face the top MDE adoption problems.”

[26] N. Aquino, J. Vanderdonckt, and O. Pastor, “Transformation Templates : Adding Flexibility to Model-Driven Engineering of User Interfaces,” pp. 1195–1202, 2010.

[27] A. G. Sparks and S. Systems, “Una Introducción al UML El Modelo de Proceso de Negocio.”

[28] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, “ATL: A model transformation tool,” *Sci. Comput. Program.*, vol. 72, no. 1–2, pp. 31–39, Jun. 2008.

[29] S. Design, “ITIL Version 3 Service Design.”

[30] C. Larman, “UML y patrones,” pp. 1–265, 2003.

[31] L. Fuentes, “Una Introducción a los Perfiles UML,” pp. 1–13.

[32] H. López, F. Varesi, M. Viñolo, D. Calegari, and C. Luna, “Reporte Técnico RT 09-19 Estado del Arte de Lenguajes y Herramientas de Transformación de Modelos,” 2009.

[33] J. C. JIMENEZ DORADO, “LENGUAJE ESPECÍFICO DE DOMINIO PARA LA DEFINICIÓN DE LA PLATAFORMA EN EL DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS,” p. 156, 2014.

[34] J. M. Vara and E. Marcos, “A framework for model-driven development of information systems: Technical decisions and lessons learned,” *J. Syst. Softw.*, vol. 85, no. 10, pp. 2368–2384, Oct. 2012.

[35] “Facultad de Ciencias y Tecnología Ingeniería Informática,” 2007.

[36] M. Milinkovich, *Praise for Clayberg and Rubel ’ s Eclipse Plug-ins , Third Edition*. .

[37] *Eclipse 4 Plug-in Development by Example Beginner ’ s Guide*. .

[38] D. Kolovos, L. Rose, A. García-domínguez, and R. Paige, “Dimitris Kolovos, Louis Rose, Antonio García-Domínguez, Richard Paige.”

[39] D. S. Kolovos, R. F. Paige, and F. A. C. Polack, “Eclipse Development Tools for Epsilon.”

[40] V. A. Bollati, J. M. Vara, Á. Jiménez, and E. Marcos, “Applying MDE to the (semi-)automatic development of model transformations,” *Inf. Softw. Technol.*, vol. 55, no. 4, pp. 699–718, Apr. 2013.

[41] A. Garc, C. Eric, S. Dupuy-chessa, N. De France, F.- Lille, and F.- Valenciennes, “UsiComp : an Extensible Model-Driven Composer Ga ¨,” pp. 263–268, 2012.

[42] A. R. Reserved and C. Notice, “Spring Roo - Reference Documentation,” 2013.

[43] P. Doyens, A. Stanciulescu, and T. Mens, “Colored Graph Transformation Rules for Model-Driven Engineering of Multi-Target Systems,” pp. 37–44, 2008.

[44] R. France and B. Rumpe, “Model-driven Development of Complex Software : A Research Roadmap Model-driven Development of Complex Software : A Research Roadmap.”

[45] A. Brown, J. Rumbaugh, and B. Selic, “MDA Journal,” pp. 1–9, 2004.

[46] M. López-Sanz and E. Marcos, “ArchiMeDeS: A model-driven framework for the specification of service-oriented architectures,” *Inf. Syst.*, vol. 37, no. 3, pp. 257–268, May 2012.

[47] J. Bernardo and R. Anaya, “Marco de Referencia para la Evaluación de Herramientas Basadas en MDA,” no. c, pp. 1–14.

[48] J. J. Cadavid, D. E. Lopez, J. A. Hincapié, and J. Bernardo, “A Domain Specific Language to Generate Web Applications.”

[49] J. Hutchinson, M. Rouncefield, and J. Whittle, “Model-driven engineering practices in industry,” *Proceeding 33rd Int. Conf. Softw. Eng. - ICSE ’11*, p. 633, 2011.

[50] R. F. Paige, F. A. C. Polack, and L. M. Rose, “Update Transformations in the Small with the Epsilon Wizard Language,” vol. 2, no. 2, 2003.