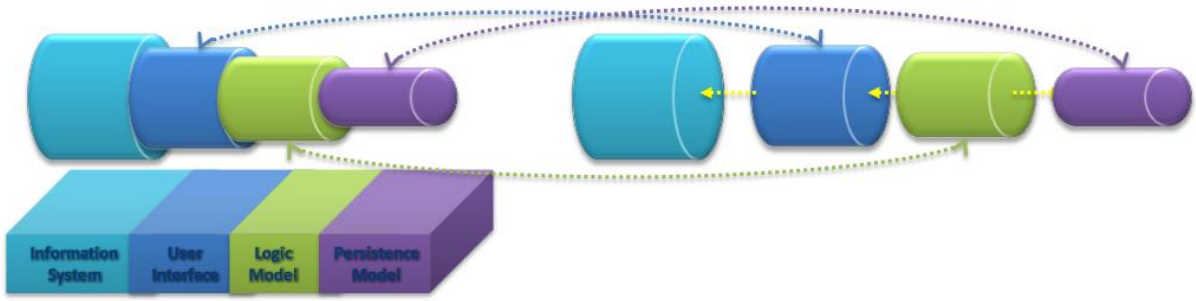
Tesina de
Máster Oficial de Ingeniería del Software,
Métodos formales y Sistemas de Información

Universidad Politécnica de Valencia

Hacia la modernización de aplicaciones corporativas dirigidas por modelos de datos.

Implementación de un round-trip multidimensional orientado a aspectos JEE.



Realizado por: *Víctor García Pau*
Dirigido por: *José Ángel Carsí y Jordi Cabot*

Prefacio del autor

Hacia la modernización de aplicaciones corporativas dirigidas por modelos de datos. Implementación de un round-trip multidimensional orientado a aspectos JEE.

Este trabajo ha estudiado, en el contexto de la evolución de software, como resolver el problema de la sincronización parcial de dimensiones definidas por los aspectos de una aplicación JEE anotada, desde un enfoque dirigido por modelos.

El problema ha evolucionado con la solución, ya que a medida que tomaba forma ha retroalimentado el problema a resolver. El objetivo de partida pretendía ofrecer un round-trip just-in-time entre el modelo, el código fuente y el ejecutable de la aplicación. De forma que un cambio en el código se reflejara en el aplicativo y en su modelo, y un cambio en el modelo se trasladara al código y modificara el aplicativo en ejecución de forma trasparente y automatizada. A esta perspectiva ágil del proceso software se le ha pretendido dar soporte mediante tecnologías dirigidas por modelos, rellenando el hueco entre las metodologías ágiles y las tecnologías model-driven disponibles.

Además de la búsqueda de una solución model-driven al problema, se proporcionan prototipos para los componentes arquitectónicos de la herramienta que da soporte a la automatización del proceso. Aunque no hay una herramienta como resultado de este trabajo, por limitaciones de tiempo y recursos, mientras se escribe esta página todavía se sigue trabajando en los problemas de sincronización y automatización. El trabajo no ha pretendido abarcar el desarrollo de plug-ins comerciales, limitándose a buscar una solución basada en modelos como prototipos de la herramienta diseñada pero siempre desde la perspectiva de lo que necesita ser resuelto para obtener ese efecto ágil en el proceso.

Este proyecto ha sido financiado por el Ministerio de Educación y Ciencia en el marco de las Becas de Movilidad para estudiantes de Máster Oficial durante el curso académico 2010-2011, para poder realizar la tesina de máster en la universidad de la Ecole des Mines de Nantes, bajo la co-dirección de Jordi Cabot, responsable del grupo de investigación AtlanMod, y la co-dirección de José Ángel Carsí desde la Universidad Politécnica de Valencia.

Agradecimientos

A mi familia por su apoyo y esfuerzo desde la distancia.

A Javier Paniza por crear OpenXava y guiarme en su compresión.

A José Ángel Carsí Cubel, por su confianza, sus enseñanzas, su visión e infinita paciencia.

A Jordi Cabot por darme la oportunidad de realizar este proyecto desde l'Ecole de Mines de Nantes, y brindarme la increíble oportunidad de aprender de su experiencia y de la de su grupo.

A Massimo Tisi por criticar mi excesivo optimismo y enseñarme de una vez, A transformar modelos y a jugar a volei.

Hugo Brunelli por guiarme con MoDisco, contrastar mis propuestas y su siempre amable y útil opinión.

A Wolfgang Kling por sus consejos e iniciarme en los plug-ins de Eclipse.

A Cauê Ávila por dar vida a VirtualEMF y compartir su música.

A Marcos Didonet por su soporte a AMW.

A Salvador Martínez por su ayuda con ATL y sus herramientas.

A Carlos Alberto González por estar siempre disponible.

A Valerio Consentino por su opinión sobre el estado del arte.

En general a todos los miembros de AtlanMod y amigos de l'Ecole des Mines de Nantes: Guillum, Hanane y Robert, a mis vecinos de departamento Jurghen, Flavien, Gustavo y Frederico, y mis compañeros de oficina Mauricio, Mohamad y Fred.

Gracias a todos ellos por su colaboración y apoyo.

Contenido

1. Introducción.

1.1. Objetivo.

Este trabajo ha estudiado, en el contexto de la evolución del software, como resolver el problema de la sincronización parcial de dimensiones definidas por los aspectos de una aplicación JEE anotada, desde un enfoque dirigido por modelos.

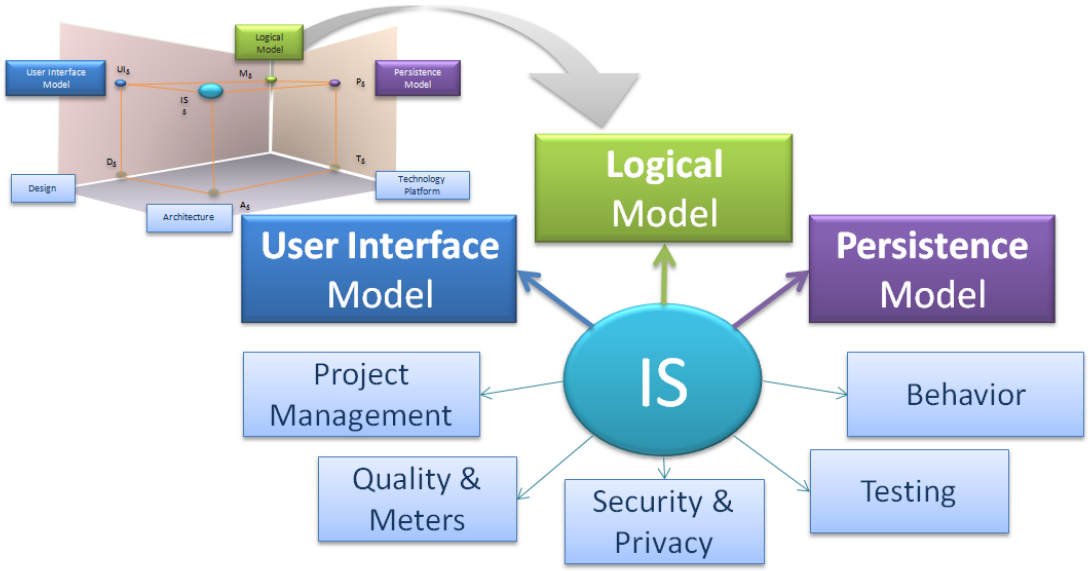


Figura 1.- Dimensiones o aspectos de un sistema de información.

Entendiendo los aspectos o dimensiones del sistema como las vistas o partes que lo definen, y en concreto en el caso de las aplicaciones JEE anotadas, las partes de código y conjuntos de anotaciones que definen una determinada dimensión. Esta solución pretendía inicialmente cubrir la persistencia JPA y la interfaz de usuario, pero si generalizamos la idea podemos aplicarla a cualquier otro aspecto que podamos describir con anotaciones del código JEE: la seguridad, el control de acceso a los recursos, la calidad, la trazabilidad de requisitos o los test unitarios. El concepto de sincronización parcial del código y de los artefactos hace referencia, a la propiedad de sincronizar independientemente las modificaciones sufridas tanto en el código fuente como en el modelo asociado, en cada una de los aspectos de una aplicación JEE anotada. Dividiendo el trabajo o la complejidad del sistema en distintas vistas duales "modelo - código fuente" que permitan utilizar diferentes tipos de editores en cada aspecto. Por ejemplo podríamos separar la estructura de clases junto con las anotaciones JPA de persistencia para editarlas con Dali, el editor estándar de JPA de Eclipse, que trabaja con el código fuente de los atributos anotados definidos en las clases de tipo entidad. Y al mismo tiempo estar editando la interfaz con un editor GMF basado en un modelo abstracto que represente la interfaz de usuario.

El objetivo de partida pretendía ofrecer un round-trip just-in-time entre el modelo completo del código fuente y el ejecutable de la aplicación. De forma que un cambio en el código se reflejara en el aplicativo y en su modelo, y un cambio en el modelo se trasladara al código y modificara el aplicativo en ejecución de forma trasparente

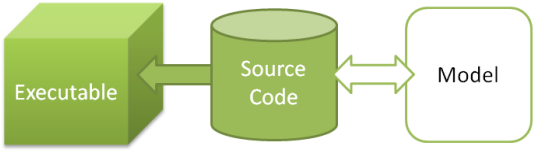


Figura 2.- Dualidad modelo-código fuente.

y automatizada. A esta perspectiva ágil del proceso software se le ha pretendido dar soporte mediante tecnologías dirigidas por modelos, rellenando el hueco entre las metodologías ágiles y las tecnologías model-driven disponibles.

Además de la búsqueda de una solución model-driven al problema, se proporcionan prototipos para los componentes arquitectónicos de la herramienta que da soporte a la automatización del proceso. Aunque no hay una herramienta como resultado de este trabajo, por limitaciones de tiempo y recursos, mientras se escribe esta página todavía se sigue trabajando en los problemas de sincronización y automatización. El trabajo no ha pretendido abarcar el desarrollo de plug-ins comerciales, limitándose a buscar una solución basada en modelos como prototipos de la herramienta diseñada, pero siempre desde la perspectiva de lo que necesita ser resuelto para obtener ese efecto ágil en el proceso. Para ello se partió de un caso de estudio real, al que se le hizo evolucionar hasta cubrir los requisitos. Esto ha permitido comprobar las bondades y debilidades de cada uno de los componentes utilizados así como su complejidad requerida para satisfacer dicho caso de estudio.

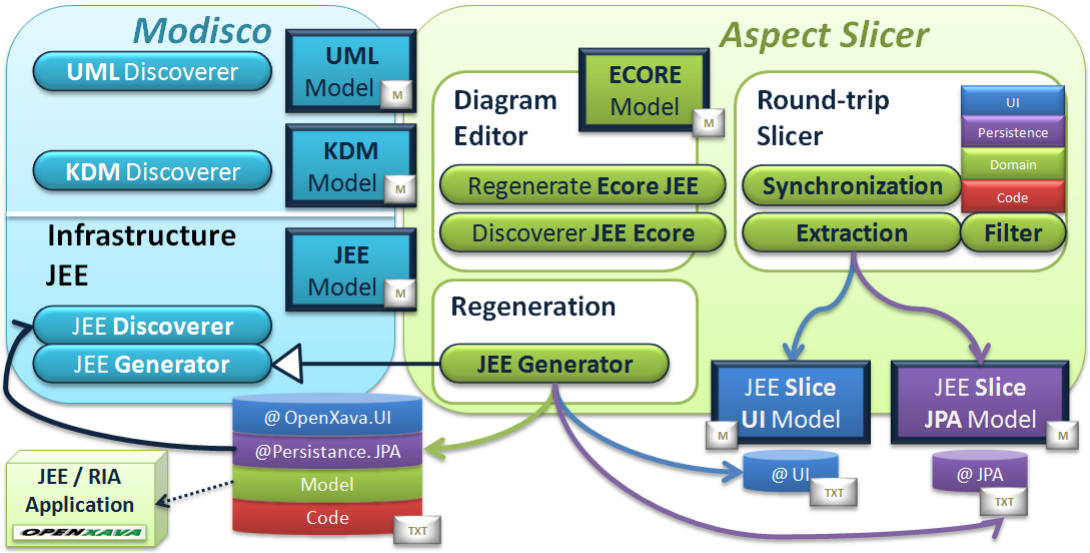


Figura 3.- Arquitectura de la solución.

Este proyecto ha sido financiado por el Ministerio de Educación y Ciencia en el marco de las Becas de Movilidad de Máster Oficial para el curso académico 2010-2011, para poder realizar la tesina de máster en la universidad de la Ecole des Mines de Nantes, bajo la co-dirección de Jordi Cabot, responsable del grupo de investigación AtlanMod, y la co-dirección de José Ángel Carsí desde la Universidad Politécnica de Valencia.

1.2. Situación de partida del proyecto.

Como punto de partida se detectó la necesidad de un round-trip dirigido por modelos de datos en el escenario descrito en trabajos anteriores [1]. La motivación inicial perseguía recuperar la inversión y dar soporte a la evolución del software en determinados escenarios donde los modelos de datos son más utilizados y conocidos que los modelos UML/MOF. Para ello se analizó cómo podía dirigirse la generación de una aplicación JEE en este contexto a partir de su esquema relacional desde un enfoque dirigido por modelos.

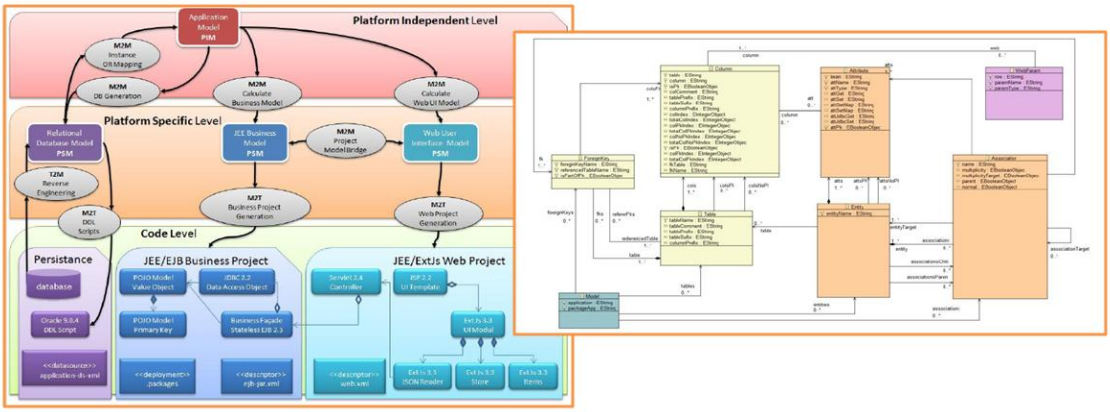


Figura 4.- Un motor de generación de código dirigido por modelos de base de datos, como punto de partida para la implantación de una plataforma MDA en la administración balear. [1] DSDM 2010. Desarrollo de Software Dirigido por Modelos.

Cuando se inició este proyecto aunque la motivación seguía viniendo de la mano de resolver desde la ingeniería inversa dirigida por modelos el desarrollo y mantenimiento de aplicaciones corporativas JEE, el contexto cambió y por tanto también las prioridades. Dado que el problema de inferir el modelo del sistema de información desde el modelo de datos había sido cubierto, ahora el interés recaía en trabajar con un modelo completo conforme al código fuente con el objeto de poder cubrir el apreciado round-trip, y así poder evolucionar no sólo el modelo sino también su código.

Pues bien este cambio de contexto también permitió flexibilizar los requisitos exigidos a la plataforma JEE de destino. Ante la necesidad de seleccionar una plataforma JEE adecuada para el caso de estudio, se seleccionó la que mejor se adaptaba al problema, con el objeto de minimizar costes. De tal forma que sin dejar de ser una solución genérica al problema permitiera abordar un mayor alcance. El framework JEE seleccionado para el caso de estudio, OpenXava, está dirigido por anotaciones; las estándar de JEE y JPA así como otras propias para la interfaz de usuario. De forma que cualquier aspecto o dimensión de la aplicación, queda definida mediante anotaciones. Este framework es una alternativa ágil basada en los estándares de JEE con interfaz AJAX. Tiene la bondad de utilizar el estándar de JPA e Hibernate para la dimensión de la persistencia y de proporcionar una interfaz funcional automática a partir del modelo. Por esta razón, cualquier modelo anotado es directamente un prototipo ejecutable facilitando la productividad y minimizando el coste. Obviamente las reglas de negocio y sus tests requieren ser expresadas mediante código, pero como métodos o servicios del propio modelo, sin desvirtuar la orientación a objetos. La filosofía del framework busca centralizar en el modelo todos los aspectos de la aplicación, en lugar de los clásicos MVC donde se produce una repetición de referencias innecesarias en cada una de las capas, dificultando la trazabilidad del código que queda diseminado en diferentes artefactos no siempre fáciles de depurar.

Esta elección por tanto carece de las virtudes de la separación en capas utilizadas en los clásicos frameworks MVC. Para poder disponer de un modelo de las diferentes vistas de la aplicación como la persistencia o la interfaz de usuario, conviene extraer submodelos (también ejecutables o al menos compilables) del original de forma automática. Pero dado que la solución a este problema sirve para cualquier grupo de anotaciones, podríamos aplicar sobre cualquier vista o dimensión que requiramos ahora o en un futuro (seguridad, calidad, trazabilidad de requisitos…). Es decir, del mismo modo que el framework de la plataforma elegida es configurable y ampliable parece natural que el proceso también lo sea.

Este hecho retroalimentó el problema y su solución. Y de esta forma ya no hablaremos únicamente de las dimensiones de persistencia o interfaz de usuario sino de aspectos corporativos en general.

Fruto de los trabajos anteriores [1] se identificó la plataforma de soporte al proceso software de Moskitt como la más adecuada para dar cobertura a aplicaciones corporativas, en el contexto de la administración pública (aunque no únicamente) ya que proporciona el soporte dirigido por modelos a metodologías configurables, incluida obviamente Métrica-3. En el comienzo del proyecto esta plataforma no disponía ni tenía entre sus prioridades abordar el proceso de round-trip por lo que se consultó al jefe del proyecto si seria de interés una colaboración en este punto.

Por otra parte, el framework JEE elegido, OpenXava, carece de un editor gráfico para Eclipse que se integre just-in-time con el código fuente. Es decir, si modificamos el código vemos su efecto en la aplicación, pero carece de un modelo que proporcione una vista simplificada del sistema, bien de su estructura, bien de su interfaz, bien de su persistencia o de cualquier otro aspecto que requiramos.

1.3. Problema a resolver.

El problema a resolver se ha ceñido a encontrar una solución dirigida por modelos para implementar el round-trip entre el código fuente y el modelo conforme al mismo de forma que nos permita evolucionar el sistema bien desde el código fuente bien desde su modelo.

Teniendo en cuenta que se requiere poder separar las dimensiones del sistema de información, con el objeto de simplificar su complejidad y focalizar las herramientas en sus diferentes aspectos. Esta separación de aspectos implica una posterior sincronización. Además las dimensiones definidas por cada aspecto deben poder evolucionar y configurarse ajustándose a las necesidades de la tarea a realizar, el role del usuario y/o características o limitaciones del editor especializado.

Además necesitamos poder disponer de un visor o diagramador del modelo del sistema configurable, donde podamos seleccionar que aspectos de los definidos queremos visualizar de forma conjunta o independiente.

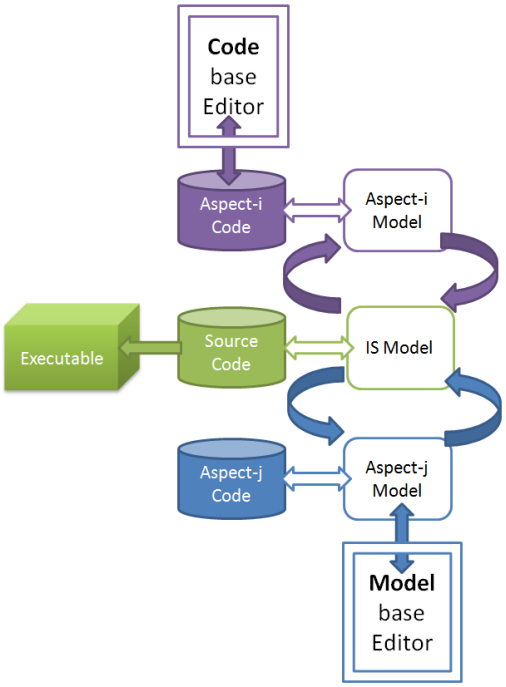


Figura 5.- Posibilitar la evolución desde el código fuente o desde el modelo, en cada aspecto del sistema.

1.4. Organización

En este capítulo se ha introducido el proyecto y se ha definido como objetivo resolver el problema de la sincronización parcial de dimensiones definidas por los aspectos de una aplicación JEE anotada. Se ha descrito la situación de partida y su relación con los trabajos anteriores. Se han explicado las posibles colaboraciones y el interés industrial.

En el siguiente capítulo se aborda la búsqueda del estado del arte, tanto desde la perspectiva de las herramientas disponibles para resolver cada reto, como desde el punto de vista de investigación en el campo de la evolución y de la ingeniería inversa dirigida por modelos.

En el tercer capítulo se aborda y describe la solución propuesta. Cómo se ha encontrado una solución y una descripción de su arquitectura. Se define su alcance y que problemas han sido resueltos. En el cuarto capítulo se describe un caso de estudio de una aplicación real donde poder identificar los escenarios de evolución e inventariar los patrones relevantes.

En los siguientes 4 capítulos, se aborda la fase ingeniería inversa, la separación y sincronización de los aspectos JEE, y se tratan el caso de la persistencia y el de la interfaz de usuario.

En el capítulo noveno se tratan las conclusiones, expectativas y resultados, tanto desde el punto de vista industrial como desde su proyección investigadora y futuros trabajos.

2. Estado del arte.

En la construcción del estado del arte se han contemplado de una parte un análisis de las herramientas disponibles para cubrir los retos del proyecto, y de otra qué aspectos a nivel de investigación componen el estado del arte. Esta búsqueda no sólo ha permitido configurar una propuesta sino que ha permitido al autor recorrer las tecnologías disponibles. Durante este recorrido iremos apuntando las opciones que mejor se alineen con el problema a resolver con el objeto de conformar dicha propuesta.

2.1. Herramientas disponibles.

En la definición del problema se ven involucradas de un forma u otra, todos los tipos de tecnologías dirigidas por modelos. De una parte necesitamos obtener un modelo completo del código para cual requerimos una operación de ingeniería inversa "Text To Model", una vez obtenido el modelo del código necesitamos manipularlo por lo que requerimos al menos un lenguaje de transformación "Model To Model", por último necesitaremos regenerar el código conforme al modelo manipulado mediante un lenguaje de generación "Text To Model". Los modelos del código para poder ser útiles requieren de al menos un visor o editor gráfico con el objeto de cumplir con la función de facilitar la compresión del sistema (no únicamente facilitar su manipulación), por lo que se requiere revisar las tecnologías de editores gráficos disponibles. Además la herramienta necesitará configurarse según las necesidades requeridas por los diferentes roles, requisitos del proyecto y tecnologías utilizadas. Para ello se necesitará configurar y editar modelos textuales que definan la configuración y permitan mantener el estado de la herramienta. Ya que las operaciones aplicadas modifican el estado de la herramienta, parece adecuado estudiar qué herramientas hay disponibles para la construcción de un DSL con el que invocar o guardar secuencias de operaciones.

Respecto a la plataforma de destino nos interesa un framework que utilice estándares JEE y que facilite las tareas de prototipado, descargando de peso a las tareas de construcción para poder focalizar el esfuerzo en las de ingeniería inversa.

2.1.1. Framework JEE: OpenXava.

En la actualidad existen una gran variedad de frameworks JEE, los más populares han sido durante años Struts y JSF, por aplicar el patrón MVC; y Spring como contenedor corporativo ligero con IoC (inversión de control) o EJB como contenedores corporativos posibles. Las virtudes de estos frameworks han sido tratadas y utilizadas ampliamente como estándares de facto de la industria. Pero en cualquier caso la construcción de un aplicativo utilizando estos frameworks implica un despliegue y diseminación de referencias al modelo entre las diferentes capas de la aplicación. Esta diseminación puede dificultar las tareas de depuración al tener referencias del mismo componente en distintos artefactos de distintas capas. Y el hecho de incrementar el número de artefactos también afecta a la mantenibilidad y compresibilidad del sistema, por no hablar del impacto sobre el número de líneas de código fuente.

De un tiempo a esta parte JEE ha incrementado su repertorio de tecnologías y capacidades disponibles pero poco se ha invertido en simplificar el número y complejidad de los artefactos requeridos: " Although JEE 5 is somewhat similar to previous versions, the

simplification is not enough. Ease of development has been ignored in favour of breadth of functionality and flexibility" Richard Monson-Haefel.

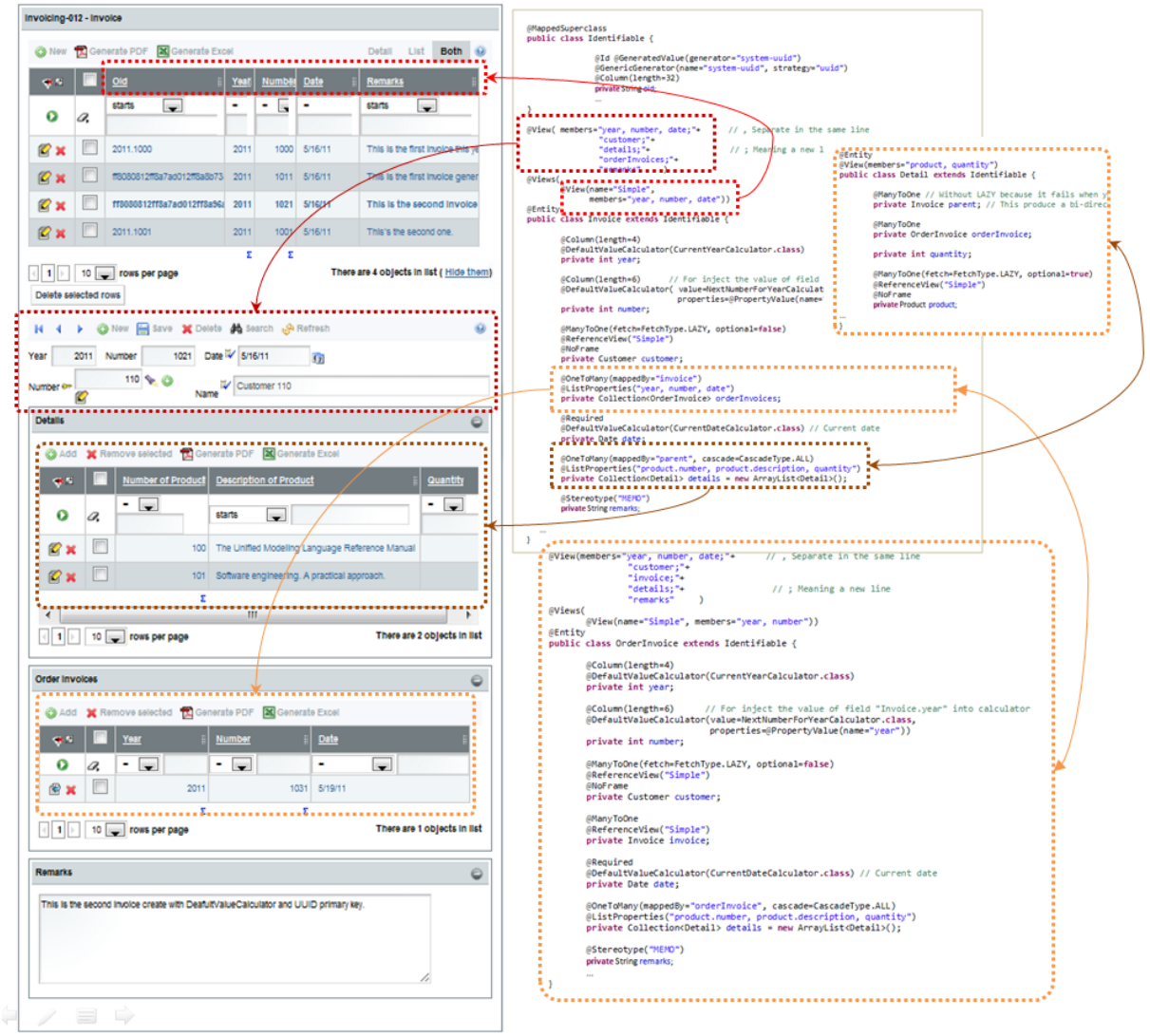


Figura 6.- Interfaz producida por un pequeño ejemplo de código de OpenXava.

Como respuesta a esta situación aparece un nuevo framework que pretende paliar los problemas: diseminación, complejidad y nº de líneas de código requeridas. Este framework open-source es OpenXava una tecnología que utiliza componentes RIA automáticos en la interfaz de usuario capaces de auto configurarse en tiempo de ejecución mediante técnicas de "reflection" sobre el modelo anotado de la aplicación. De forma que cualquier clase mínimamente anotada es de facto un prototipo ejecutable. Delegando a los componentes estándar utilizados por el framework el tratamiento por ejemplo de la persistencia JPA o a los validadores de Hibernate. La interfaz de usuario se construye de forma predeterminada a partir de la meta información estructural y sus anotaciones asociadas, obtenidas por "reflection" de las clases del modelo. Y además se puede configurar su disposición espacial, aspecto y comportamiento también mediante anotaciones. Aunque se puede refinar el comportamiento del controlador de la interfaz de usuario mediante la definición de acciones personalizadas que sustituyan a las predefinidas y un "façade" que proporciona acceso a la meta información utilizada por el framework en tiempo de ejecución con el objeto de poder modificar el comportamiento a partir del estado de la interfaz, o bien crear test unitarios también sobre la interfaz de usuario. El ejemplo de la figura 6 se define con un centenar de

líneas de código diseminadas en únicamente 6 clases java: Identifiable, Customer, Product, Invoice, Detail y OrderInvoice.

2.1.2. Text-to-model, DSL e ingeniería inversa.

Para obtener un modelo del código se dispone de varias alternativas, no todas aptas para el problema a resolver. Es por ello que empezaremos por confeccionar una tabla comparativa con las diferentes herramientas y técnicas relacionadas con la ingeniería inversa, hasta donde conoce el autor, sobre diferentes ámbitos: base de datos, XML, SOA, modelos, gramáticas y código JEE.

Tecnología	Ámbito	MD* GRAMMA	EMF	JPA	XML	WS / SOA	JEE Java	JEE JSP	JEE JAR
Diccionario de datos	Base de datos								
	La meta información que permita acceder de la base de datos vía **SQL** de forma nativa desde su diccionario al usuario. The Data Dictionary – Oracle System Catalogs - PostgreSQL								
JDBC DatabaseMetaData	Base de datos								
	Permite acceder desde Java a cualquier propiedad de los componentes de una base de datos siempre que el driver **JDBC** y la base de datos implementen el acceso. download.oracle.com/javase/6/docs/api/java/sql/DatabaseMetaData								
Hibernate Tools	Persistencia OR-mapping			Sí	Sí				
	Framework de persistencia con mapeo objecto-relactional automatizable con tareas **ANT** y dispone de plug-ins para **Eclipse**, que permite obtener la estructura y generar el código Java/.Net, para diferentes plataformas de persistencia: **JDBC**, **JPA**, EJB3, XML, NoSQL y otras tipos de base de datos. Uno de los motores OR más populares, estándar de facto e implementación JPA de referencia. www.hibernate.org								
Eclipse Link	Persistencia OR-mapping	MD*	Sí	Sí	Sí	Sí			
	Una solución **Java**/SOA de persistencia capaz de manejar : Object-Relational (**JPA**), MOXy: Object-XML (JAXB), DBWS: Database Web Services, XML-Relational (XRM), Service Data Objects (SDO y **EMF/SDO**) y Non-Relational (EIS via JCA). www.eclipse.org/eclipselink/ JPA DBWS MOXy								
MinuteProject	JPA Reverse			Sí	Sí	Sí			
	Es un poderoso **generador de aplicaciones** JEE, JSF, **OpenXava** y Ruby entre otros, basado en tecnologías de Spring e Hibernate principalmente, y alimentado por una herramienta de **ingeniería inversa de JPA** contra base de datos. minuteproject.wikispaces.com/								
Moskitt Ingenieria Inversa de Base de datos	Modeling Tool Software Process DB Reverse Eng.	MD*	Sí	Sí					
	Es una magnífica plataforma MD* con soporte al proceso software y entre muchas otras características con herramientas de Ingenieria Inversa de Base de datos.								

Tabla 1.- Parte 1ª : Tecnologias y herramientas de ingeniería inversa de base datos.

Tecnologia	Ámbito	MD* GRAMMA	EMF	JPA	XML	WS / SOA	JEE Java	JEE JSP	JEE JAR
Teneo	Model-Relational mapping	MD*	Sí	Sí	Sí	Sí			
	Model-Relational mapping and runtime database persistence solution for the Eclipse Modeling Framework using **Hibernate** or **EclipseLink**. It supports automatic creation of **EMF** to Relational Mappings. EMF Objects can be stored and retrieved using advanced queries (HQL or EJB-QL). wiki.eclipse.org/Teneo#teneo								
Service Data Objects (**SDO**)	SOA-mapping	MD*	Sí	Sí	Sí	Sí			
	Simplifica y unifica el desarrollo de aplicaciones en una arquitectura orientada a servicios (**SOA**). Es compatible con XML y se integra con patrones JEE. Incluye una implementación de Service Data Objects basada en EMF. EclipseLink/FAQ/WhatIsSDO								
CDO Model Respository	Database Model Repository	MD*	Sí						
	Connected Data Objects (CDO) es un **repositorio de modelos** distribuidos y un framework de acceso a modelos y metamodelos EMF almacenado en base de datos configurable con Hibernate y EclipseLink contra cualquier fuente de datos. www.eclipse.org/cdo/ wiki.eclipse.org/CDO								
Dynamic EMF	Run-time modeling	MD*	Sí						
	EMF puede generar un metamodelo a partir de XSD, UML, Ecore o Java anotado pero también a partir de la API **reflectiva** de EMF en tiempo de ejecución. Build metamodels with dynamic EMF www.voelter.de - DynamicModelCreationTest.java								
ATL Ant Injector	T2M	MD*	Sí		Sí				
	Permite **inyectar como modelos** de una transformacion ATL diferentes tipos de fichero: xml , texto ATL, ebnf… Existen también sus correspondienes “Extractor”s. wiki.eclipse.org/AM3_Ant_Tasks								

Tabla 1.- Parte 2ª : Tecnologias y herramientas de ingeniería inversa sobre modelos EMF y textos de ATL o XML.

Notas para la interpretación de la clasificación de las tecnologías detalladas:

Ámbito	área sobre la que se enfoca el uso de la herramienta o tecnología.
MD* GRAMMA	indica si utiliza un enfoque model-driven* o dsls o bien gramáticas.
EMF	utiliza o puede utilizar modelos EMF.
JPA	realiza ingeniería inversa de JPA, EJB3 o object/model-relational mapping.
XML	realiza ingeniería inversa de XML.
WS/SOA	realiza ingeniería inversa sobre webservices o componentes soa.
JEE Java	realiza ingeniería inversa sobre código Java.
JEE JSP	realiza ingeniería inversa sobre código JSP y/o JSF.
JEE JAR	realiza ingeniería inversa sobre código compilado o librerías de Java.

Tecnología	Ámbito	MD* GRAMMA	EMF	JPA	XML	WS / SOA	JEE Java	JEE JSP	JEE JAR
Java Reflection	Reflection						Sí		Sí
	Podemos obtener cualquier meta información del código, incluidas las anotaciones java pero **sin acceder al código fuente**. Y además trabaja sobre código **compilado**. Discovering Class Members								
(JDT) Eclipse Java Development Tools	Reflection + Código Compilación Parcial			Sí	Sí		Sí		Sí
	Proporciona soporte complete al IDE de Java para Eclipse: **APT** Annotation Processing Tool. **Core** incluye el compilador incremental, el modelo de Java, el acceso al código, asistentes, refactorizaciones y cálculo de jerarquías de tipos. **Debug** ejecuta en depuración o no, permite evaluar expresiones y recarga de clases.**Text** proporciona el editor de código, acceso al Javadoc y formateo de código. **UI** implementa el Package explorer, Type Hierarchy, Outline y Wizards de Java. Eclipse JDT								

Tabla 1.- Parte 3ª : Tecnologias y herramientas de ingeniería inversa sobre texto Java.

En el siguiente subgrupo, aun no siendo herramientas de ingeniería inversa, posibilitan la creación de analizadores sintácticos, traductores, metamodelos y/o DSLs a partir de gramáticas o de otros metamodelos como es el último caso:

Tecnologia	Ámbito	MD* GRAMMA	EMF	JPA	XML	WS / SOA	JEE Java	JEE JSP	JEE JAR
ANTLR	Gramáticas y compiladores	GRAM		Sí	Sí		Sí	Sí	
	Es un **generador de gramáticas, intérpretes y compiladores** basado en Java y con un plug-in para Eclipse. Tiene una excelente y didáctica documentación principalmente orientada a lenguajes LL(*) . ANTLR 3.x - Concepts								
Textual Concrete Syntax (TCS)		MD* DSL GRAM	Sí						
	Permite definer sintaxis concreta para la sintaxis abstracta de un metamodelo. Una vez definida puede analizar textos y cargarlos en un modelo y permite realce de sintaxis. Está basado en la generación de gramátivas LL(*) para ANTLR v3. www.eclipse.org/gmt/tcs/								
XText	Gramáticas y DSL	MD* DSL GRAM	Sí						
	Un herramienta completa de generación de DSL integrada en el IDE de Eclipse bien a partir de un gramática bien a partir de un metamodelo. Permite crear un DSL con su editor avanzado con asistente, intérprete y/o compilador integrable en Eclipse. . Está basado en la generación de gramátivas LL(*) para ANTLR v3. www.eclipse.org Xtext								

Tabla 1.- Parte 4ª : Tecnologias y herramientas de ingeniería inversa y text-to-model.

Tecnologia	Ámbito	MD* GRAMMA	EMF	JPA	XML	WS / SOA	JEE Java	JEE JSP	JEE JAR
MoDisco	T2M M2T Model Discoverer	MD*	Sí	Sí	Sí	Sí	Sí	Sí	Sí
	La herramienta KDM de referencia para OMG proporciona cobertura completa para sistemas legados. Dispone de una infrastructura completa para Java, JEE y XML, tanto de descubrimiento T2M como de regeneración M2T. Permite obtener un modelo completo, incluido el código fuente, las anotaciones y los comentarios de cualquiere texto Java JEE. Ofrece la posibilidad descubrir el modelo Java, KDM y UML de cualquier proyecto JEE. Eclipse MoDisco KDM · SMM · GASTM · Model Browser · Discovery Manager · Workflow ·Query Manager · Facet Manager · Metrics Visualization Java · JEE · EjbJar · WebApp · XML								
Jar2Uml	T2M Model Discoverer	MD*	Sí						Sí
	Este plug-in importa y covierte ficheros **JAR** a UML y está integrado en MoDisco. soft.vub.ac.be/soft/research/mdd:jar2uml								
Gra2Mol	T2M Model Discoverer Gramáticas	MD* GRAM	Sí						
	Una herramienta para la Extracción de Modelos en Modernización de Software. Javier Luis Cánovas Izquierdo, Jesús García Molina JISBD, 162-165, 2009								
Semantic Designs DMS	T2G G2G G2T Gramáticas Grafos	GRAF GRAM	?	Sí	Sí	Sí	Sí	Sí	Sí
	Herramienta de **re-ingenieria**, cubre todas las facetas: generador de compiladores y analizadors léxicos, análisis del código, transformacdión, refactorización y formateo. Aplicado tanto en lenguajes de programación de software como COBOL, C/C++, Java, Fortran 90, PLSQL, Python, PHP o Perl, así como en lenguajes de especificación de sistemas como VHDL. www.semanticdesigns.com/Products/DMS/DMSToolkit.html								

Tabla 1.- Parte 5ª : Tecnologias y herramientas de ingeniería inversa y text-to-model.

Después de este largo recorrido entre las herramientas referenciadas en la anterior tabla, es obvio que la herramienta que mejor se adapta al problema a resolver es **MoDisco**. Ya que permite un descubrimiento completo de cualquier aplicación JEE y además dispone de la capacidad de regenración del código a partir de su modelo. Se ha comprobado la compatibilidad del metamodelo de Java con los tipos genéricos, las anotaciones y comentarios de un código fuente de Java y todo ello desde un enfoque model-driven.

Por otra parte, aunque no es parte del problema a resolver, la herramienta que mejor se adapataria a la fase previa de obtener el código fuente conforme a un esquema relacional recaeria sobre la herramienta **MinuteProject**. Ya que permite **ingeniería inversa de JPA** y generación de **OpenXava**, cumpliendo perfectamente con el role definido en trabajos anteriores [1]**.**

2.1.3. Model to Model.

Respecto de la transformación de modelos voy a inventariar el conocieminto relevante del que dispongo en el momento de readactar este trabajo. Siendo consciente de estar muy lejos del estado del arte actual. Por lo que me he limitado a utilizar y aprender las tecnologías disponibles a mi alcance para poder resolver con el menor coste el problema planteado.

Tecnologia	Ámbito	Descripción
QVT	M2M Standard OMG	MOF™ Query / View / Transformation (QVT) Permiet la consulta, vista y transformación de modelos MOF.
QVT Procedural	M2M Operational	La versión imperativa del estándar definido por QVT para la transformación de modelos. Model Transformation with Operational QVT EclipseCon 2009
QVT Declarative	M2M Core & Relational	Es la versión declarativa de QVt cuya principal cualidad es la bi-direccionalidad de sus definiciones, entre otras com la trazabilidad, limpieza y propiedades semi-formales. QVT 1.1 OMG specification
MediniQVT	M2M	Plataforma y editor gráfico para la ejecución de transformaciones QVT Relational. mediniQVT
MOMENT	M2M MDE	Es una platforma de gestión de modelos EMF utilizando QVT-Realtional y basado en las propiedades algebraicas proporcionadas por Maude para computar eficiéntemente modelos desde un enfoque formal. **moment.dsic.upv.es**
ATL	M2M Standard de facto	Lenguaje de transformación imperativo y declarativo QVT-like. Disponible desde Eclipse Modeling. ATL - **Overview** - **User Guide** – **Developer Guide**
Ant Tasks ATL Tools	M2M	ATL dispone de un conjunto de tareas ANT con las que automatizar y encadenar transformaciones sobre los diferentes modelos y meta modelos requeridos. **ATL Ant TasksTools**
AMW	M2M	Permite definir weaving de modelos asistiendo enl descubrimiento de similitudes y permitiendo crear bien esquelos de transformaciones bien definiciones con otros propósitos o usos. **AMW**
VirtualEMF	M2M EMF	Herramienta basada en un weaving definido con AMW entre dos meta modelos, proporciona un acceso a un modelo virtual resultado de la composición de los modelos entretejidos en el weaving de AMW siguiendo tres posibles relaciones: merge, asociación y filtrado de clasificadores de los modelos. Las transformaciones realizadas sobre el meta modelo virtual se almacenan en modelos conforme a los meta modelos de entrada permitiendo una definición e interacción de meta modelos sin ser ensuciados con detalles ajenos. **VirtualEMF** - **Paper**

Tabla 2.- Tecnologias model-to-model.

2.1.4. Model to text.

Respecto de la generación de texto a partir de modelo, contamos con 2 grandes grupos de herramientas. De una parte los clásicos formateadores basados en el modelo del **código** como Velocity o FreeMarker (utilizados por las JBoss Tools, Hibernate y otras) o sobre **XML** como XSLT. Y de otra parte las basadas en modelos **EMF**/MOF/UML como JET, XPand y Acceleo.

Tecnología	Ámbito	Descripción
Apache Velocity	Lenguaje de plantillas sobre POJO	Este framework se base en el modelo definido por instancias de POJOs cargados en el contexto y un lenguaje de plantillas formateador del mismo. velocity.apache.org
FreeMarker	Lenguaje de plantillas sobre POJO	Es una mejora del framework anterior con características y uso similar. Actualmente utilizado por los generadores de Hibernate tools. freemarker.sourceforge.net
XSLT	Lenguaje XSL de transformación sobre XML	Este estándar de XML permite generar y transformar a partir de un modelo y una plantilla expresados ambos en XML. www.w3schools.com/xsl
JET	M2T	Lenguaje de pantillas similar a JSP basado en modelos EMF. Eclipse M2T JET
XPand	M2T	Generador incremental basado en modelos EMF, capaz de detectar que partes del modelo han cambiado y generar únicamente los artefactos relacionados. También permite pequeñas transformaciones de modelos. wiki.eclipse.org/Xpand
Acceleo	M2T	Es una implementación del MOF Model to Text Language (MTL) definido por OMG. www.eclipse.org/acceleo Object Management Group (OMG) MOF Model to Text Language (MTL)
ATL writeTo	M2M	ATL disponde de una función de escritura a fichero para cualquier "<String>.**writeTo**(fileName)". Si la utilizamos dentro de una transformación "**refining mode**" de ATL podemos generar y actualizar el modelo utilizado. ATL User_Guide

Tabla 3.- Tecnologías model-to-text.

Respecto al problema a resolver está clara la elección ya que viene influenciada por la anterior, pues la herramienta de regeneración de código Java de MoDisco está implementada con una plantilla de **Acceleo**.

En el caso de requerirse una pequeña generación de texto que además modifique el modelo utilizado, cabe señalar a la función "<String>.**writeTo**(fileName)" de **ATL** utilizada en una transformación "**refining mode**".

2.1.5. Generación de editores gráficos.

Con el objeto de visualizar y editar, si se da el caso, los modelos del código se han revisado las tecnologías de editores gráficos disponibles hasta donde conoce el autor.

De una parte contamos con el procedimiento definido por **GMF (Graphical Modeling Framework)** para la construcción de editores basados en una orientación generativa: la definición de la barra de tareas (**.gmftool**), de los componentes del gráfico (**.gmfgraph**) y el mapeo de ambos (**.gmfmap**) para obeterner un editor gráfico que se ejecute sobre el runtime de GMF. Tecnología dependiente de la plataforma de Eclipse y con ciertas dificultades para daptarse o evolucionar el editor una vez creado. Podemos apoyarnos en las **anotaciones gráficas del meta modelo Ecore** ofrecidas por la herramienta **EuGENia GMF** tool para obtener los tres componentes anteriores.

Tecnología	Ámbito	Descripción
Graphical Modeling Framework (GMF)	GMP Generativo	Este framework se base en la definición de 3 modelos basados en EMF y GEF donde se definen los componentes del editor gráfico basado en el runtime de GMF de Eclipse: .gmfgraph, .gmftool y .gmfmap. **Graphical Modeling Framework (GMF)**
EuGENia (GMF)	GMP Generativo Anotaciones	Genera los modelos .gmfgraph, .gmftool y .gmfmap requeridos por un editor GMF a partir de un meta modelo Ecore con anotaciones @**gmf**. Eclipse GMT Epsilon EuGENia GMF
Graphical Editing Framework (GEF)	GEF	Tecnología para crear editores gráficos y visores para Eclipse. Se compone de: **Draw2d** (org.eclipse.draw2d) **GEF (MVC)** (org.eclipse.gef) **Zest** (org.eclipse.zest) Eclipse Workbench UI. http://www.eclipse.org/gef/
Graphity	EMF GEF	Es una infraestructura para la creación de herramientas gráficas independiente de plataforma con una API Java sencilla y de fácil uso que se integra con EMF y permite extender los algoritmos de renderización para diferentes plataformas. Graphiti
Spray	Graphity DSL	Es un DSL generativo para Graphity disponible después de EclipseCon 2011. Spray 5ise
Graphical Ecore Editor	Ecore Tools	Es el editor gráfico de modelos Ecore, siempre que podamos mapear nuestro meta modelo a Ecore dispondremos de un editor muy ágil. EMFT Ecore Tools EcoreDiagram Editor

Tabla 4.- Tecnologías para la generación de editores gráficos.

De otra parte contamos con la nueva generación de editores independientes de plataforma basados en la infraestructura ofrecida por la API Java de **Graphity** fácilmente

integrable con EMF y con algoritmos de layout extensibles. Una herramienta que promete facilitar todavía más su uso es un DSL, denominado **Spray**, para generar editores basados en esta tecnología.

Por último comentaré una opción muy simple si se puede disponer de Ecore como meta modelo como es el caso. En el corazón de las Ecore Tools se dispone de un Editor Gráfico de Meta modelos Ecore y de un generador **.ecorediagram**. Como Java puede ser representado por un modelo Ecore, incluidas las anotaciones y los tipos genéricos podemos proyectar a Ecore el modelo del código Java y así disponer de un editor a bajo coste. Esta opción ha sido la elección final para visualizar los modelos de Java.

2.2. Architecture-driven Modernization.

Es la iniciativa de Object Management Group para dar soporte a la especificación de estándares aplicables a la modernización de sistemas legados desde una arquitectura dirigida por modelos. El objetivo principal es proporcionar un conjunto de meta modelos sobre los que extraer una representación común para las actividades de modernización: análisis, compresión y transformación del software. El objeto de esta normalización es proporcionar un marco común para el intercambio de meta información, no solo para realizar traducciones del código fuente de una plataforma a otra, sino también permitir rehacer los análisis y diseños del sistema modernizado utilizando como fuente de requisitos el propio sistema legado, salvando costes de reingeniería y evitando perdidas de requisitos. Se compone de los siguientes paquetes:

Abstract Syntax Tree Metamodel (ASTM) define los modelos de los arboles sintácticos abstractos. Este meta modelo está complementado por el Knowledge Discovery Metamodel (KDM) que representa a los modelos de grafos semánticos asociados. En contrasto con KDM o UML, ASTM mantiene una correspondencia directa uno a uno entre cada sentencia del código fuente y un modelos software de muy bajo nivel. Existen 4 subpaquetes dependiendo del grado de abstracción e independencia:

- **Generic Abstract Syntax Tree Metamodel (GASTM)** define un conjunto genérico de elementos comunes de modelado para cualquier lenguaje. En esta norma los modelos GAST son expresados como diagramas de clases UML. Un ejemplo puede encontrarse en la herramienta MoDisco GASTM.
- **Language Specific Abstract Syntax Tree Metamodels (SASTM)** para cada uno de los lenguajes en particular tales como Ada, C, Fortran, Java, etc. son modelados en MOF y expresados conformes a GASTM junto con los elementos de modelado requeridos para poder capturar el lenguaje. Un ejemplo relacionado con este trabajo se puede encontrar en MoDisco Java Infratrucuture Technologies Java 0.9, JEE 0.9 y XML 0.9.
- **Propietary Abstract Syntax Tree Metamodels (PASTM)** expresa un AST de un lenguaje específico como Ada, C, COBOL, etc. modelado en un formato propietario no conforme a MOF, GASTM o SASTM. Pero para el cual el AST propietario define una mínima correspondencia para el soportar del intercambio de modelos.

Knowledge Discovery Metamodel (KDM) define un meta modelo para expresar los activos software y su entornos operacionales (en lo referido al descubrimiento de conocimiento sobre los mismos). Ésta es la primera dentro de la serie de especificaciones definidas por ADM para facilitar el intercambio de información de modernización entre herramientas de

diferentes fabricantes. Aquí se dispone de un enlace a MoDisco KDM la implementación de referencia para OMG.

Software Metrics Metamodel (SMM) define un meta modelo para representar métricas relacionadas con el software, su funcionamiento y su diseño. La especificación es un meta modelo extensible para el intercambio de información relacionada con el software de medición sobre los activos software existentes (sus diseños, sus aplicaciones, u operaciones). Podemos tomar como ejemplo la implementación de MoDisco SMM.

Structured Assurance Case Metamodel (SACM) este meta modelo pretende dar soporte estructurado a los argumentos de calidad, está compuesta de dos meta modelos:

- **Argumentation Metamodel (ARM)** proporciona a los proyectos una comunicación eficiente y satisfactoria de forma estructurada sobre cómo se cumplen los requisitos de calidad de los sistemas y servicios.

- **Software Assurance Evidence Metamodel (SAEM)** establece los modelos de grano fino a las evidencias y análisis de riesgos. Proporciona la base para un diseño lógico que facilita la construcción de herramientas para el almacenamiento, referencia cruzada, evaluación e informe de las evidencias de los sistemas durante el proceso de Calidad del Software.

2.2.1. Model Driven Reverse Engineering.

Trata el problema de la ingeniería inversa de los artefactos software desde un enfoque dirigido por modelos. MoDisco es uno de los frameworks que permite realizar de forma extensible este trabajo y como se ha explicado anteriormente es la implementación de referencia de KDM recomendada por OMG. Para más detalles recomiendo visitar Eclipse-MDT MoDisco project (Model Driven Reverse Engineering framework).

2.2.3. Program Slicing.

Hasta donde conoce el autor esta técnica trata el problema de la extracción de partes de código relacionado con el objeto de calcular, extraer o transformar funcionalidades o aspectos definidos en el código de un programa.

2.2.4. Trabajos relevantes.

Hasta donde conoce el autor y ha podido indagar de expertos en distintos campos. Me aventuro a ofrecer una selección ni sistemática ni exhaustiva de trabajos relevantes.

En el contexto de **Model Driven Reverse Engineering**:

- Model driven reverse engineering. Rugaber, S., Stirewalt, K. - IEEE Software 21(4), p45-53 (2004)

- Foundations of model (driven) (reverse) engineering : Models - episode i: Stories of the fidus papyrus and of the solarus. Favre, J.M. Dagstuhl, Germany (2005)

En el contexto de **Architecture-Driven Modernization** y **MoDisco** debemos señalar:

- How to Deal with your IT Legacy? Reverse Engineering using Models - MoDisco in a Nutshell! . Hugo Brunelière - JavaTech Journal #10

- OMG Architecture Driven Modernization. website (2011). URL http://adm.omg.org

- API2MoL: Automating the Building of Bridges between APIs and Model Driven. Javier Luis Cánovas Izquierdo, Frédéric Jouault, Jordi Cabot, y Jesús García Molina - Information and Software Technology (IST)

- A Domain Specific Language for Extracting Models in Software Modernization. Javier Luis Cánovas Izquierdo, Jesús García Molina. - European Conference on Model Driven Architecture - Foundations and Applications, 82-97, 2009

- Gra2MoL: Una Herramienta para la Extracción de Modelos en Modernización de Software. Javier Luis Cánovas Izquierdo, Jesús García Molina - JISBD, 162-165, 2009

En el contexto de **Program Slicing**:

- A Simple Mathematically Based Framework for Rule Extraction from an Arbitrary. Ramsey, F.V.; Alpigini, J.J.; Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International 2002

- *A brief survey of program slicing. - B Xu, J Qian, X Zhang, Z Wu…; ACM SIGSOFT Software 2005*

- *Context-free slicing of UML class models. - H Kagdi, JI Maletic; IEEE International Conference on Software Maintenance (ICSM'05)*

3. Solución propuesta.

El problema a resolver se ha venido conformando según se revisaba la adecuación al problema de las diferentes herramientas y tecnologías disponibles. Para poder definir una propuesta deberemos ir tomando decisiones de diseño y comprobar su impacto en la solución.

3.1. Búsqueda de la solución.

Para poder encontrar una solución dirigida por modelos del round-trip entre el código fuente JEE y su modelo, se ha elegido a MoDisco ya que nos proporciona un componente bidireccional, tanto para el descubrimiento del modelo como para la regeneración a código, conforme a los estándares ASTM y KDM definidos por OMG para dar soporte a la evolución de software.

Esta herramienta nos permite ver a los artefactos JEE de una forma dual:

- código del artefacto Java
- modelo SASTM conforme a la sintaxis de Java

Ya que el modelo SAST del código Java, en lo sucesivo modelo Java, se puede proyectar a KDM y éste a su vez a UML por la propia herramienta, podemos manipular bien el código, bien sus modelos, con un efecto equivalente, como se apuntaba en la figura 2 de la introducción y aquí se procede a matizar.

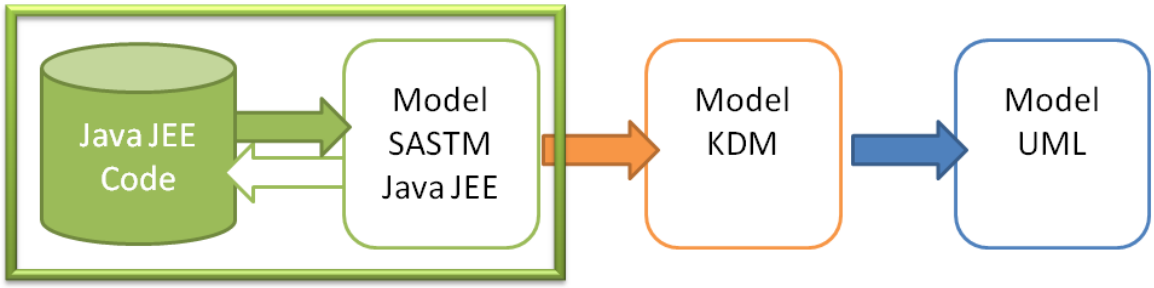


Figura 7.- Dualidad del código Java JEE como modelo SASTM y proyección a KDM y UML.

Ya que MoDisco nos proporciona esta equivalencia código-modelo, al menos hasta nivel AST, podemos manipular bien el código, bien el modelo. Esto resuelve el problema de tener editores que trabajan directamente con el código fuente como los editores JDT de Eclipse o el editor de Dali de JPA para Eclipse. Al mismo tiempo podemos disponer de editores especializados que trabajen a partir del modelo o modelos derivados del código fuente, como podría ser el caso de un diagramador de la interfaz de usuario.

Ya que nos hemos restringido a tratar con JEE anotado, podemos ver los aspectos como conjuntos de anotaciones que describen una semántica extra del código fuente, enriqueciendo semánticamente las estructuras y algoritmos definidos en las clases. Más concretamente en el caso del framework seleccionado, OpenXava, además se centraliza en el modelo todos los aspectos definidos en el código, simplificando el problema a resolver. Estas anotaciones son ancladas al modelo, bien a sus clases, bien a sus atributos o bien a sus métodos.

Como hemos apuntado antes, podemos trabajar diferentes aspectos con diferentes editores, ahora bien si deseamos separar estos aspectos para editarlos nos veremos obligados a mezclarlos tras la modificación. Teniendo que diseñar una estrategia de sincronización.

En este punto parece haber un conflicto pues ¿qué ocurriría si un editor modificara la estructura de las clases donde se anclan las anotaciones? A la hora de devolver modificadas las anotaciones del aspecto al modelo no encontraríamos el anclaje original. Además hay aspectos que están íntimamente relacionados con la estructura de clases como es la persistencia y cuyo editor debería de poder modificar la estructura. En cambio hay otros aspectos que meramente decoran la estructura sin modificarla, como es el caso de la interfaz de usuario. A estos aspectos los debemos de clasificar en dos categorías, unas capaces de modificar la estructura y otras como simples decoradores. A similitud el patrón "maestro/esclavo" o "modelo/decorador" desde el punto de vista de la sincronización. Siendo la persistencia un aspecto maestro y en cambio la interfaz un esclavo o decorador.

De este modo podemos utilizar la información estructural de las clases como índice o esqueleto sobre el que decorar los diferentes aspectos de cada dimensión. Donde los aspectos "maestro" proporcionaran su estructura y anotaciones, mientras que los esclavos o "decoradores" se limitan a añadir sus anotaciones en los componentes estructurales impuestos por el modelo "maestro". Esta opción nos permite actualizar anotaciones sobre componentes comunes a ambos modelos, pero en el caso de modificarse o desaparecer el elemento de anclaje necesitamos definir una estrategia de sincronización. Si analizamos el problema nos encontramos con varias opciones:

- **no hacer nada**: es la opción más sencilla ya que si no existe el elemento de anclaje o no lo podemos encontrar no necesitamos añadirle semántica.
- **comentar el código modificado o eliminado**: nos permite preservar el código y sus anotaciones en caso de eliminar o modificar un componente.
- **refactorizar**: es la mejor opción pero también es la más compleja.

La primera opción puede ser viable si apoyamos al sistema sobre un gestor de versiones de forma que podamos tener una secuencia de las modificaciones de donde recuperar el código si se requiere. Si no disponemos de un gestor de versiones sincronizado con cada cambio del modelo, al menos necesitamos comentar el código eliminado con el objeto de preservar la inversión. La tercera opción sería la ideal si contáramos con una librería de refactorización enfocada al modelo SASTM de Java, haciendo el símil de la proporcionada por JDT para los fuentes Java. Dado que el coste y complejidad de esta solución es mucho mayor, la descartaremos por sobrepasar los recursos y tiempo disponibles.

Dado que la acción de decorar la estructura del modelo implica identificar sus componentes hemos de analizar que identificador utilizaremos para cada elemento de anclaje:

- **nombre y tipo**: el nombre junto con su tipo puede ser un identificar válido siempre que no cambiemos el nombre o el tipo del componente: la clase, el atributo o el método.
- **referencia al elemento del modelo**: esta sería una opción válida siempre que no se cambie el orden de aparición de los elementos en el modelo, ya que si observamos el fichero XMI las referencias son dependientes de su orden de aparición.
- **identificador sintético**: podemos utilizar un identificador sintético o una secuencia independiente del nombre y tipo del objeto. Esta meta información requiere ser recalculada y almacenada en el modelo cada vez que añadidos elementos al modelo y siempre sin reutilizar los huecos generados por los elementos eliminados. Así dispondremos de un identificador único para cada elemento de la estructura a modo de

índice. Esta opción requiere ser almacenado en el modelo junto al elemento de anclaje, preferiblemente como una **EAnnotation** (anotación ECORE al elemento del modelo) con el objeto de no ensuciar el código fuente o verse afectado por manipulaciones inadecuadas.

La última opción, implica poder definir una EAnnotation donde almacenar el índice sintético de cualquier elemento del modelo que deseemos identificar como anclaje, al menos las clases, atributos y métodos. Ahora bien si el meta modelo SASTM no contempla esta posibilidad no lo podemos almacenar de forma trasparente en el modelo, como es el caso del meta modelo SASTM Java utilizado por MoDisco.

De todo esto podemos sintetizar los factores que determinan el algoritmo de sincronización:

Identificación	Estrategia		
	No hacer nada	Comentar	Refactorizar
Nombre y tipo	Apto	Sin renombrado ni cambio de tipo	No apto
Referencia	Apto	Sin reordenación	Sin reordenación
Identificador Sintético	No requerido	EAnnotation	EAnnotation

Tabla 5.- Estrategias de Sincronización e Identificación.

Otro punto a señalar es la generación del comentario de una parte del código, para ello debemos fijar la atención en la estructura del elemento "Comment" del SASTM de Java y vemos que se almacena en forma de texto en el atributo "content". Esto implica que si deseamos realizar una operación tan sencilla como comentar una línea de código necesitamos calcular el texto equivalente generado por el subárbol AST del elemento a comentar. Esta operación no es despreciable ya que supone replicar la plantilla M2T proporcionada por MoDisco dentro de una transformación M2M, lo cual no está justificado ni por coste ni por mantenibilidad. Ya que un cambio en la plantilla o en la transformación debería propagarse a ambos componentes. Una alternativa para resolver esta operación en una transformación M2M es producir un ciclo de regeneración y descubrimiento modificando únicamente la plantilla M2T conforme a un modelo SASTM extendido para almacenar las EAnnotation de los elementos a comentar. Esto requiere una fase de marcado de elementos a comentar con una operación M2M, invocar la generación M2T y posteriormente redescubrir el código regenerado. En cualquier caso se añade complejidad a la herramienta. Una tercera opción es recomendar añadir una EAnnotation de servicio al meta modelo SASTM de MoDisco para Java con el propósito de facilitar la refactorización y el comentado de subárboles SASTM. En este caso dado que no se ha analizado en profundidad las posibles implicaciones de una refactorización el elemento referenciado por la EAnnotation debería extenderse no solo a las clases, sus atributos y métodos, sino a cualquier elemento del árbol. Para ello la EAnnotation debería referirse al clasificador "ASTNode" superclase de todos los elementos del árbol.

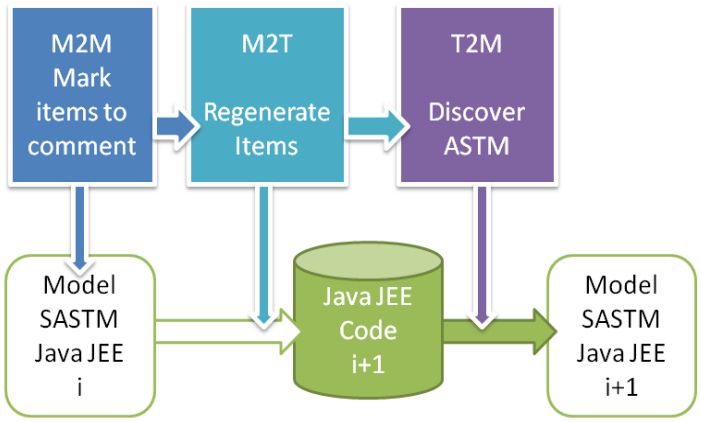


Figura 8.- Regenerar para comentar un elemento.

Otro punto que asoma es la necesidad de invocar diferentes tipos de transformaciones dentro de una secuencia atómica cómo en el caso de la regeneración del código comentado. Este mezclado de transformaciones implica a transformaciones M2M, en general ATL en este contexto y, M2T en forma de plantillas Acceleo y T2M para invocar al plug-in asociado de descubrimiento. En cualquier caso se requiere secuenciar operaciones de distinta naturaleza y compartir información de estado para su sincronización. Dado que ATL no admite parámetros de entrada, toda información le debe llegar en forma de modelo. Cómo las transformaciones M2M y M2T se pueden secuenciar con tareas ANT necesitaremos poder compartir cierta información de estado entre transformaciones vía modelos, y con las tareas ANT, los plug-in y/o ejecutables vía archivos de propiedades. Dado que pretendemos ofrecer una solución dirigida por modelos debemos facilitar la configuración desde el modelo a las transformaciones en primer lugar, y generar los ficheros requeridos tanto para crear las secuencias de tareas, como para alimentar esa configuración a tareas ANT, plug-ins o ejecutables.

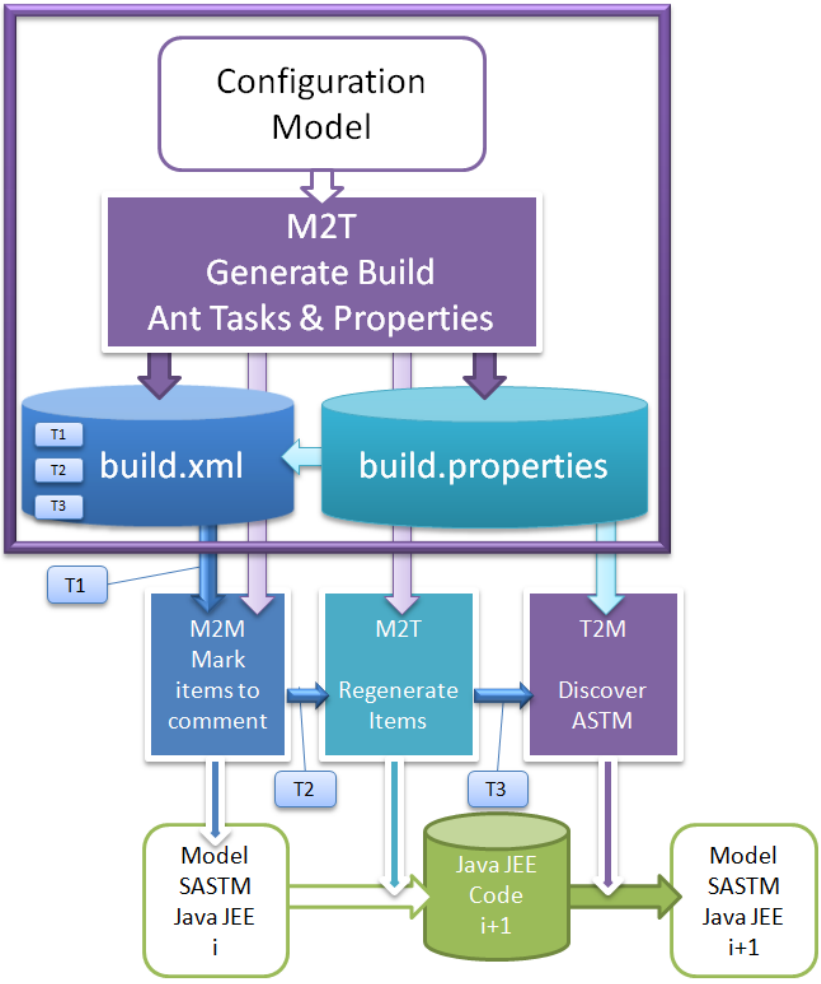


Figura 9.- Compartir información de configuración entre transformaciones y ejecutables.

He de recordar que pretendemos ofrecer una vista ágil de los modelos del código de forma que no se requiera o se minimice las operaciones y manipulaciones del usuario para observar cambios en el modelo, bien en su estructura bien en sus aspectos. Además esta visualización debe ser configurable con el objeto de ayudar a la compresión en lugar de ocultar o dificultar la tarea a realizar.

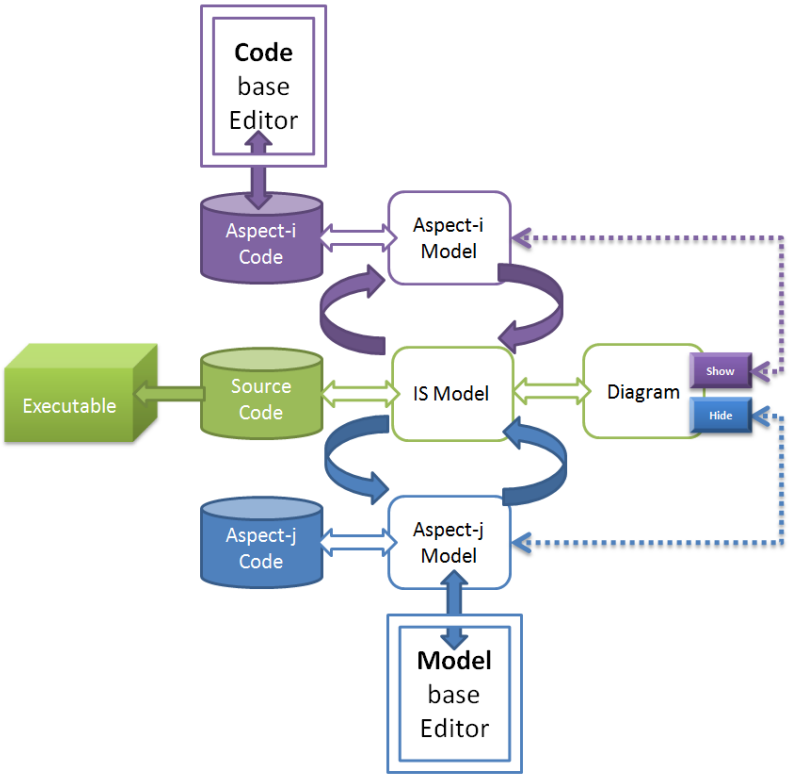


Figura 10.- El diagrama debe de poder mostrar los aspectos adecuados para el usuario.

Esta configuración puede depender del role del usuario, la tarea a realizar y/o limitaciones de diseño de los editores especializados. Además la configuración de cada aspecto puede variar con el tiempo al igual que las tecnologías utilizadas.

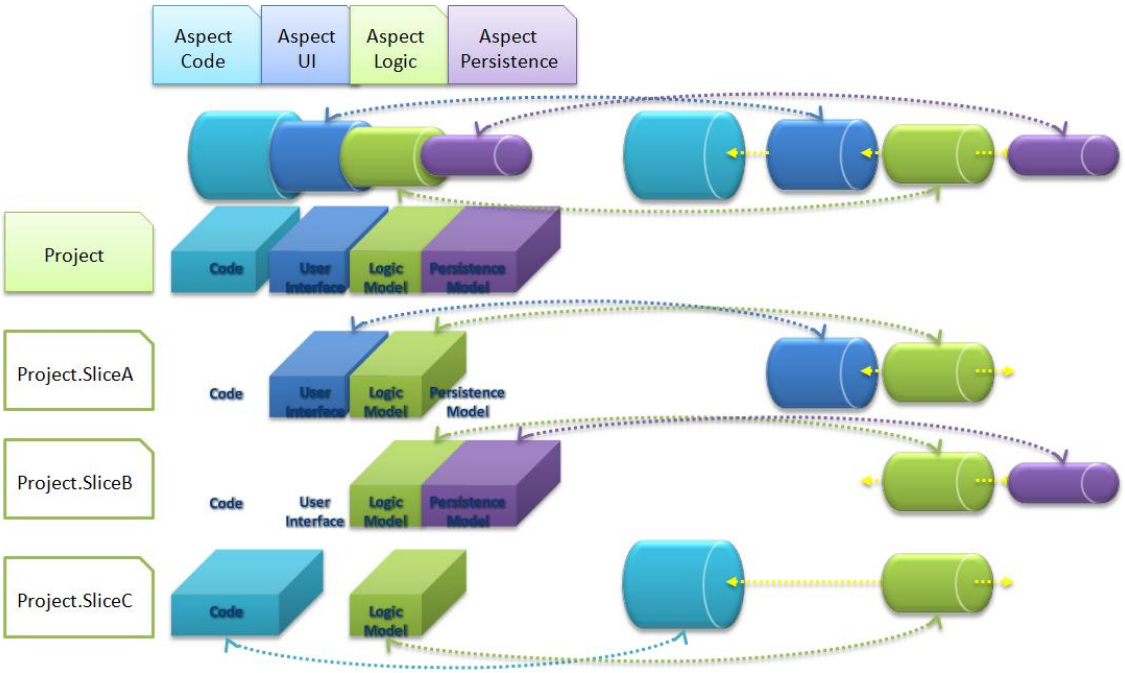


Figura 11.- Configuración de los aspectos y las vistas o "slices".

3.2. *Visión genérica de la arquitectura.*

Uniendo cada uno de estos puntos podemos definir una arquitectura que nos permita especificar los aspectos que trata el sistema, como se agrupan para conformar una vista o "slice" del sistema, donde se almacenan estas definiciones de las vistas y como se crean cada una de estas vistas con su información de estado, las tareas de sincronización y sus ficheros de configuración.

Para resolverlo, se propone crear un proyecto por cada "slice" o vista definida en el modelo de la configuración de la herramienta. Este modelo de configuración común se almacería por ejemplo en la carpeta raíz del proyecto original, el que dispone de todo el código fuente. Y por cada uno de los proyectos se tendría un modelo para configurar las tareas de sincronización encargado de generar las tareas ANT adaptadas al path del proyecto y los ficheros de propiedades con la información de configuración a compartir por las tareas y/o ejecutables.

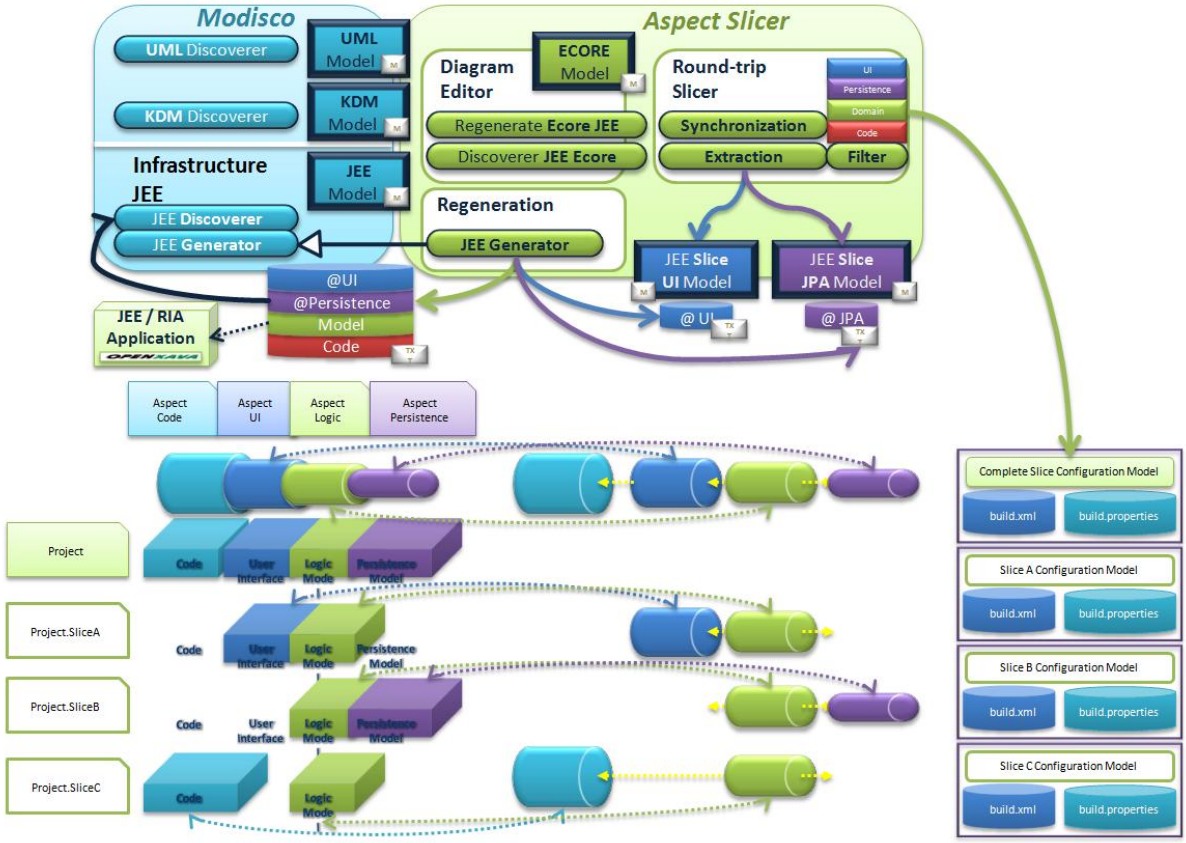


Figura 12.- Visión genérica de la arquitectura.

3.3. *Descripción de la solución.*

La solución propuesta se conforma como un conjunto de proyectos con unos artefactos asociados y transformaciones disponibles, algunos de ellos se encuentran tanto en el proyecto original como en cada uno de los proyectos de las diferentes vistas o "slices". Del conjunto de operaciones detallaremos las diferentes transformaciones requeridas para darle soporte.

El conjunto de artefactos requeridos por el sistema en cada proyecto, se analiza a continuación:

Proyecto	Artefacto	Descripción
Original	**aspectslicer.xmi**	Modelo de configuración común de los aspectos y slices. Aquí se define el inventario de modelos y diagramas junto con sus estados y referencias con el objeto de dirigir el proceso.
Original Slice.XYZ	**aslicer.build.xmi**	Modelo de asistencia a la configuración donde se definen las tareas ANT que deben poder invocarse para: A.- Tareas de infraestructura: 1. crear los proyectos de los slices 2. actualizar la configuración de los slice B.- Tareas comunes: 3. descubrir el código del proyecto 4. regenerar el modelo SASTM Java 5. descubrir el modelo KDM 6. descubrir el modelo UML C.- Tareas de sincronización: 7. extraer un slice 8. sincronizar un slice D.- Tareas de visualización: 9. generar diagrama 10. muestra aspecto 11. oculta aspecto
Original Slice.XYZ	**aslice.build.xml**	Fichero de tareas ANT donde se definen las transformaciones, sus modelos y meta modelos utilizados por cada transformación definida en el "slice".
Original Slice.XYZ	**aslice.build.properties**	Fichero de propiedades donde compartir la información de configuración definida en el modelo del "slice", **aslicer.build.xmi**, con el objeto de alimentar a las tareas ANT, plug-ins o ejecutables requeridos.
Original	aslicer.build.xmi tarea 01 **crearSlices**	Crea un copia del proyecto por cada "slice" definido en el modelo **aspectslicer.xmi** e invoca a la tarea de "configurarslice" una vez creadas.
Original Slice.XYZ	aslicer.build.xmi tarea 02 **configurarSlice**	Actualiza los modelos de configuración de cada unos de los "slices", definido en el modelo **aslicer.build.xmi** almacenado en cada proyecto. E invoca la transformación de generar los artefactos de sincronización de cada "slice": **aslice.build.xml** **aslice.build.properties** (bien sea el proyecto original bien una de sus vistas o "slices").

Tabla 6 Parte 1ª.- Artefactos y tareas del sistema.

Proyecto	Artefacto	Descripción
Original Slice.XYZ	aslicer.build.xmi tarea 03 **descubrirCodigo**	Crea o actualiza el modelo Java conforme al **código Java** y artefactos del proyecto del "slice" (bien sea el proyecto original bien una de sus vistas o "slices").
Original Slice.XYZ	aslicer.build.xmi tarea 04 **regenerarCodigo**	Regenera el código fuente conforme al **modelo Java** y sus artefactos del proyecto del "slice" (bien sea el proyecto original bien una de sus vistas o "slices").
Original Slice.XYZ	aslicer.build.xmi tarea 05 **descubrirKDM**	Crea o actualiza el modelo KDM conforme al **modelo Java** y artefactos del proyecto del "slice" (bien sea el proyecto original bien una de sus vistas o "slices").
Original Slice.XYZ	aslicer.build.xmi tarea 06 **descubrirUML**	Crea o actualiza el modelo UML conforme al **modelo KDM** y artefactos del proyecto del "slice" (bien sea el proyecto original bien una de sus vistas o "slices").
Slice.XYZ	aslicer.build.xmi tarea 07 **extraerSlice**	Crea o actualiza el modelo Java del proyecto conforme al **modelo Java Original filtrado por los aspectos** definidos para la vista o "slice". (bien sea el proyecto original bien una de sus vistas o "slices").
Slice.XYZ	aslicer.build.xmi tarea 08 **sincroSlice**	Actualiza el **modelo Java Original** con el modelo Java del proyecto conforme al **filtrado de los aspectos** definidos en el "slice". (bien sea el proyecto original bien una de sus vistas o "slices").
Original Slice.XYZ	aslicer.build.xmi tarea 09 **generaDiagrama**	Crea o actualiza el modelo Ecore del diagrama a partir del modelo Java del proyecto. (bien sea el proyecto original bien una de sus vistas o "slices").
Original Slice.XYZ	aslicer.build.xmi tarea 10 **mostrarAspecto**	Actualiza el modelo Ecore del diagrama para mostrar un aspecto. (bien sea el proyecto original bien una de sus vistas o "slices").
Original Slice.XYZ	aslicer.build.xmi tarea 11 **ocultarAspecto**	Actualiza el modelo Ecore del diagrama para ocultar un aspecto. (bien sea el proyecto original bien una de sus vistas o "slices").

Tabla 6 Parte 2ª.- Artefactos y tareas del sistema.

Podemos observar que estas tareas se agrupan en 4 funciones relacionadas con las tareas de:
A.- tareas de infraestructura
B.- tareas comunes
C.- tareas de sincronización
D.- tareas de visualización

Para cada una de las tareas definidas identificaremos que transformaciones requieren:

Proyecto	Artefacto	Transformaciones
Original Slice.XYZ	aslicer.build.xmi tarea 02 **configurarSlice**	**T_BuildCreateAnt.atl** Crea los ficheros de configuración del "slice" a partir del modelo **aslicer.build.xmi** almacenado en el proyecto: **aslice.build.xml** **aslice.build.properties**
Original Slice.XYZ	aslicer.build.xmi tarea 03 **descubrirCodigo**	**MoDisco.Discoverer.Java**
Original Slice.XYZ	aslicer.build.xmi tarea 04 **regenerarCodigo**	**MoDisco.Generator.Java** Se requiere modificar su plantilla para incluir el comentado de los elementos marcados. De momento se utilizará sin modificar la plantilla de Acceleo definida en el plug-in.
Original Slice.XYZ	aslicer.build.xmi tarea 05 **descubrirKDM**	**MoDisco.Discoverer.KDM**
Original Slice.XYZ	aslicer.build.xmi tarea 06 **descubrirUML**	**MoDisco.Discoverer.UML**
Slice.XYZ	aslicer.build.xmi tarea 07 **extraerSlice**	Crea o actualiza el modelo Java del proyecto conforme al **modelo Java Original filtrado por los aspectos** definidos para la vista o "slice". (bien sea el proyecto original bien una de sus vistas o "slices").
Slice.XYZ	aslicer.build.xmi tarea 08 **sincroSlice**	Actualiza el **modelo Java Original** con el modelo Java del proyecto conforme al **filtrado de los aspectos** definidos en el "slice". (bien sea el proyecto original bien una de sus vistas o "slices").
Original Slice.XYZ	aslicer.build.xmi tarea 09 **generaDiagrama**	Crea o actualiza el modelo Ecore del diagrama a partir del modelo Java del proyecto. (bien sea el proyecto original bien una de sus vistas o "slices").
Original Slice.XYZ	aslicer.build.xmi tarea 10 **mostrarAspecto**	Actualiza el modelo Ecore del diagrama para mostrar un aspecto. (bien sea el proyecto original bien una de sus vistas o "slices").
Original Slice.XYZ	aslicer.build.xmi tarea 11 **ocultarAspecto**	Actualiza el modelo Ecore del diagrama para ocultar un aspecto. (bien sea el proyecto original bien una de sus vistas o "slices").

Tabla 7.- Transformaciones requeridas por las tareas del sistema.

3.4. Proyecto.

Dado que las tareas a realizar son muchas en relación al tiempo y recursos disponibles se procederá a ir resolviendo los retos con prototipos de los componentes de la herramienta.

Para empezar necesitamos disponer de un corpus de proyectos reales a los que aplicar ingeniería inversa y donde podamos identificar la evolución natural que sufre la aplicación de un sistema de información. Además este conjunto de herramientas no permitirá verificar si el framework proporciona los servicios esperados, mientras aprendemos a utilizarlo. Ya que estos proyectos serán reales, su código reflejará necesidades reales que pueden pasarse por alto en la fase de análisis y diseño de la herramienta.

Una vez construidos los casos de estudio, aplicaremos el descubrimiento de sus modelos Java. A partir de este punto ya contaremos con un modelo completo del código, que podremos regenerar o transformar según las transformaciones definidas en la tabla 7.

Llegado este punto, para poder definir las transformaciones, previamente necesitamos definir los meta modelos y crear modelos de ejemplo, instancias de los mismos donde comprobar que se pueden cubrir los requisitos anteriormente inventariados. Y también los datos, sus modelos de entrada, con los que desarrollar las transformaciones requeridas.

Además dado que el sistema ofrece una cierta complejidad y número importante de transformaciones, en los casos más complejos empezaremos con versiones simplificadas de los meta modelos iterando sobre las soluciones para cubrir los requisitos.

Dado que el trabajo a cubrir excede al tiempo, priorizaremos las transformaciones que no proporciona MoDisco y dejaremos para el final la modificación de la plantilla generadora que aunque simple no deja de suponer trabajo.

Por otra parte para poder comprender y visualizar los cambios aplicados, necesitamos poder disponer de un diagrama que represente al modelo Java. Ya que el número de elementos del modelo de cada proyecto es considerable, incluso en los casos de usos más sencillos empezaremos con las transformaciones de relacionadas con la generación y visualización de los ecorediagram asociados a los modelos Java. Convirtiéndose en un herramienta valiosa para depurar el resto de transformaciones.

El siguiente paso es terminar las transformaciones de visualización, comprobando que podemos mostrar y ocultar los aspectos de forma adecuada para facilitar la comprensión del modelo Java al usuario de la herramienta.

Una vez resuelta la visualización de los diagramas, necesitamos poder extraer los modelos de cada “slice”, con los que comprobar posteriormente la sincronización con el proyecto original.

Además antes de tratar el problema de la sincronización necesitamos proporcionar una infraestructura básica para la automatización y secuenciación de las transformaciones y tareas requeridas por la herramienta.

Finalmente construir los plug-ins que invoquen a las tareas o a las transformaciones según sea más conveniente en cada caso, pero ya adelanto que no cabe dentro de los plazos disponibles para este proyecto.

4. Caso de estudio.

El corpus de proyectos que conforman el caso de estudio ha sido desarrollado para cumplir con las siguientes funciones:

- Conocer cómo utilizar el framework desarrollando un sistema de información.
- Comprobar las capacidades ofrecidas por el mismo.
- Disponer de unos prototipos incrementales ejecutables con un código al que aplicar la herramienta y comprobar sus propiedades.

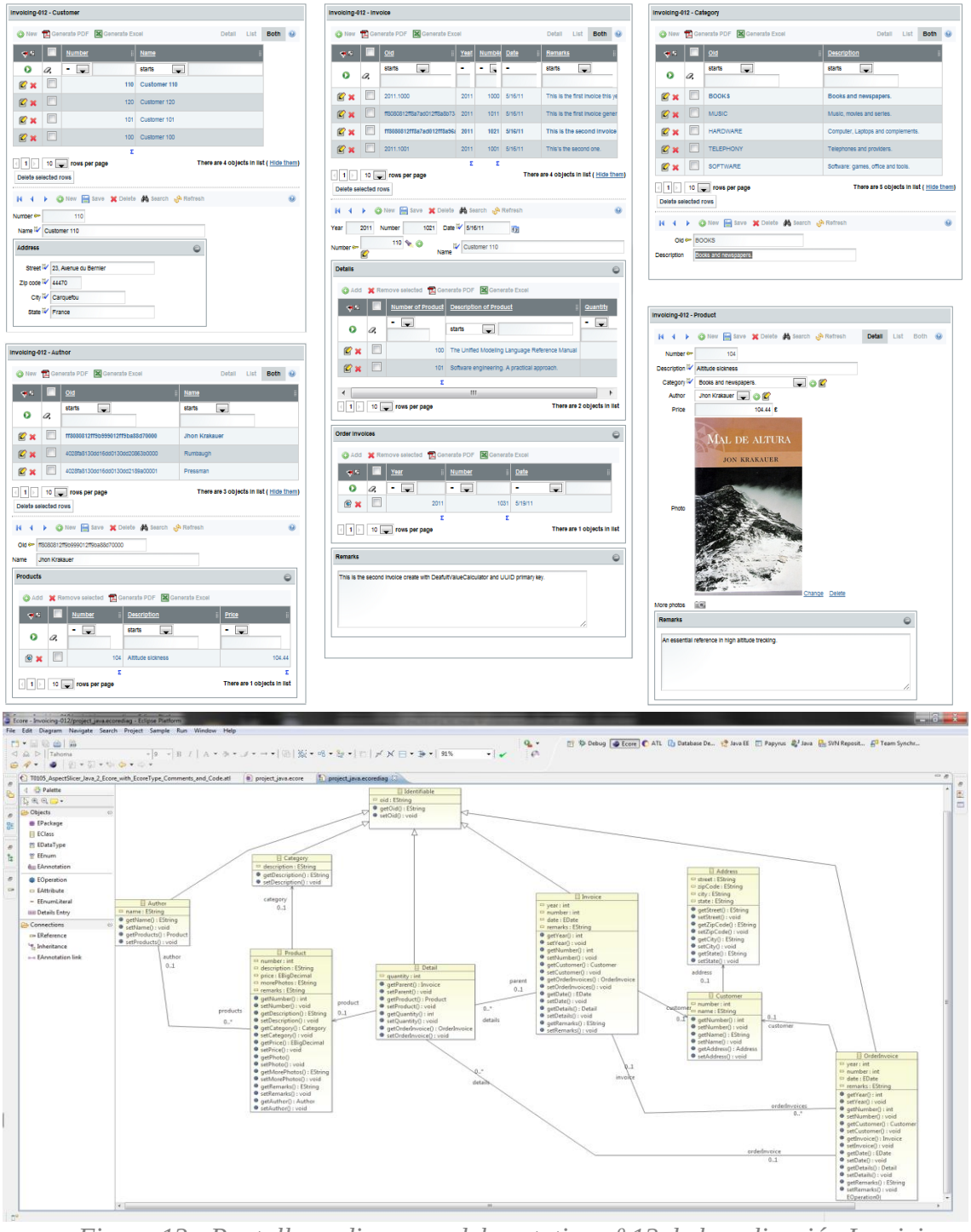


Figura 13.- Pantallas y diagrama del prototipo nº 12 de la aplicación Invoicing.

Como podemos observar en la figura 13 el aspecto de la aplicación final es bastante atractivo y agradable. El número de componentes del modelo es relativamente bajo, ya que con 9 clases podemos definir una aplicación de facturación sencilla.

4.1. Secuencia de evolución.

Para poder dar una idea más detallada de cómo podemos dar soporte a la evolución, se tomo la decisión de crear un prototipo ejecutable por cada cambio significativo requerido por el sistema para su implementación. Se introdujeron, casos típicos, como el cambio de sistema de identificación de las entidades (el frecuente cambio de PK de las aplicaciones reales) y algunos otros problemas relacionados con la herencia y la evolución del sistema.

El primer proyecto Invoicing-000 cuenta con únicamente dos entidades definidas pero cuyo ejecutable ya nos permite editar sus datos.

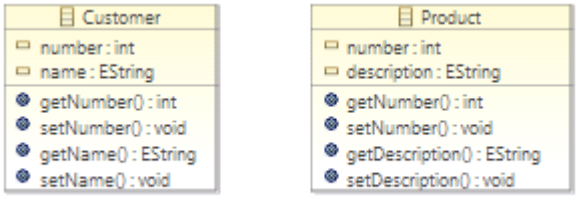


Figura 14.- Prototipo Invoicing-000.

En la siguiente versión Invoicing-001 ya podemos ver diferentes asociaciones entre entidades. En esta ocasión utilizando claves primary key compuestas.

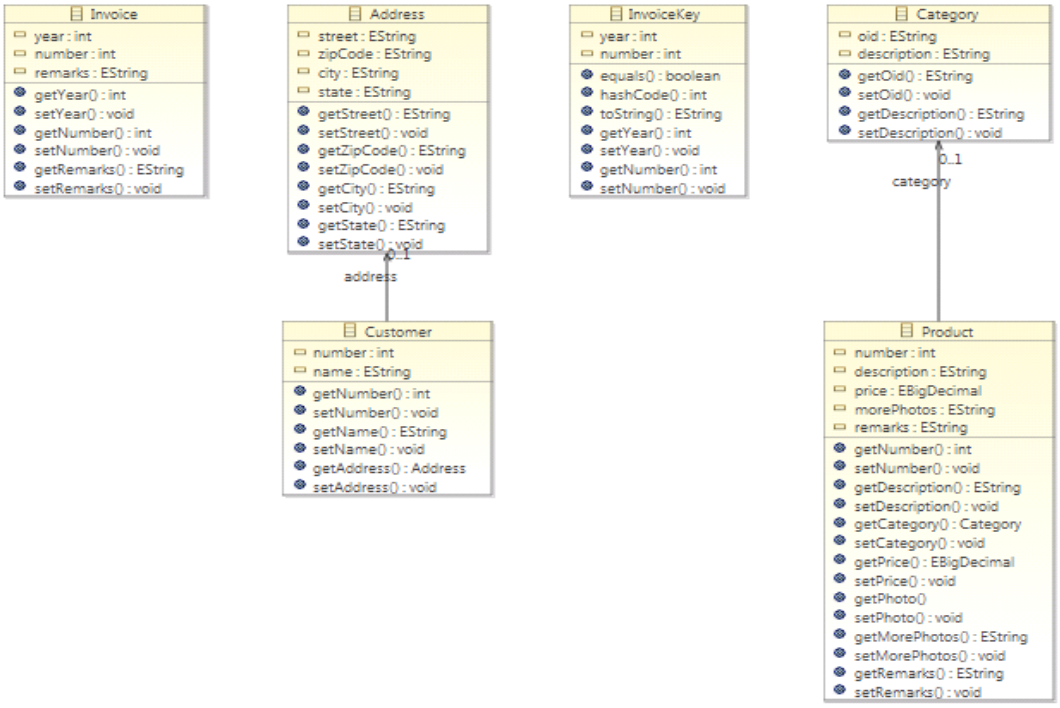


Figura 15.- Prototipo Invoicing-001.

En el siguiente prototipo Invoicing-002 hemos sustituido los identificadores de sintéticos “oid”, por lo que las clases utilizadas para definir la clave compuesta desaparecen.

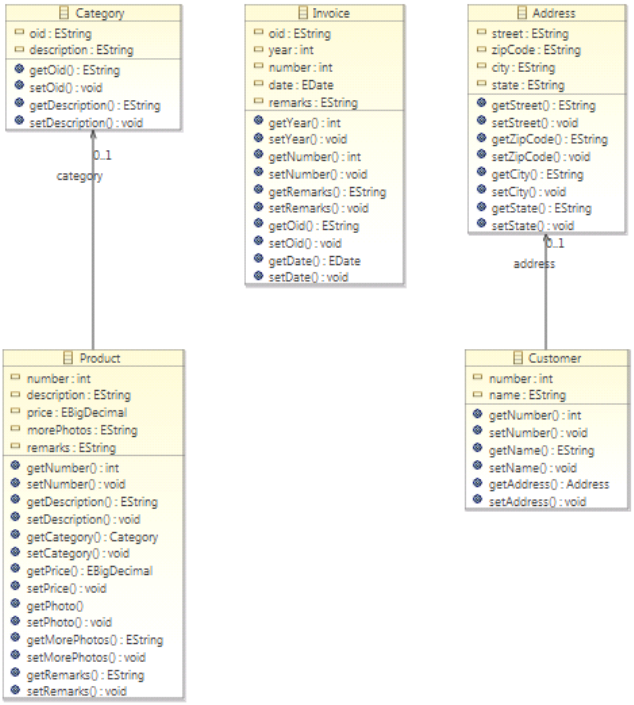


Figura 16.- Prototipo Invoicing-002.

En Invoicing-003 ya tenemos un diagrama de clases más cercano a una solución de facturación con relaciones maestro-detalle (agregados) entre la factura y sus detalles.

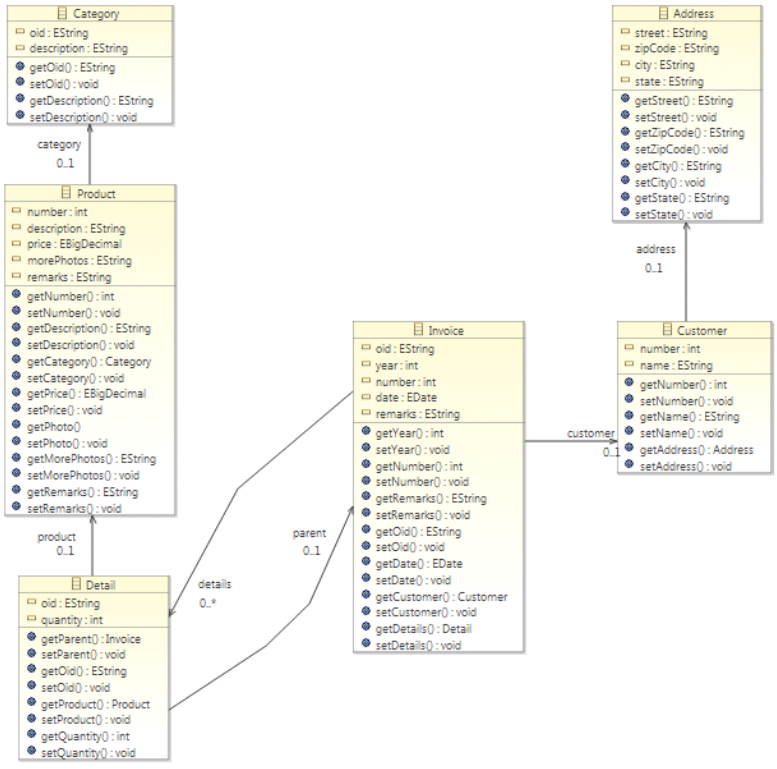


Figura 17.- Prototipo Invoicing-003.

Invoicing-004 añade una segunda relación de agregados para los productos del autor a los productos agrupados por categorías.

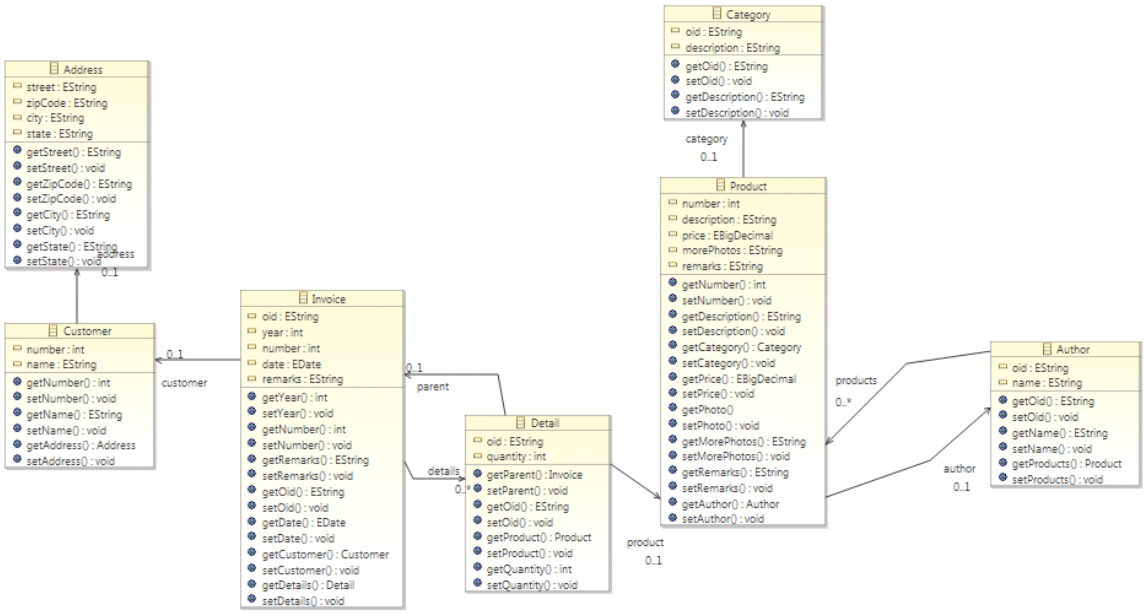


Figura 18.- Prototipo Invoicing-004.

La versión de Invoicing-011 introduce un cambio de identificador al refactorizar las clases: Invoice, Detail, Category y Author haciéndoles derivar de Identifiable. Esta clase abstracta define los identificadores de todas estas subclases.

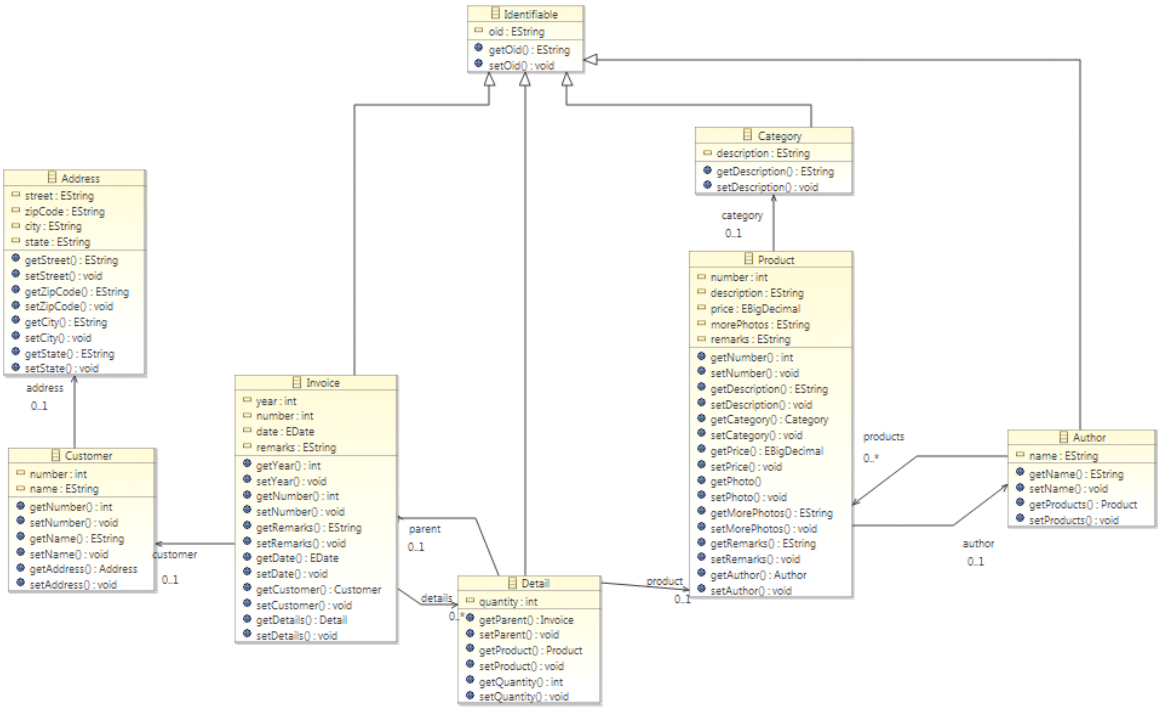


Figura 19.- Prototipo Invoicing-011.

En la versión de Invoicing-012 se introduce la entidad OrderInvoice como agregado de Detail y de la factura Invoice. Este es el paso previo para crear una estructura común a Invoice y Order.

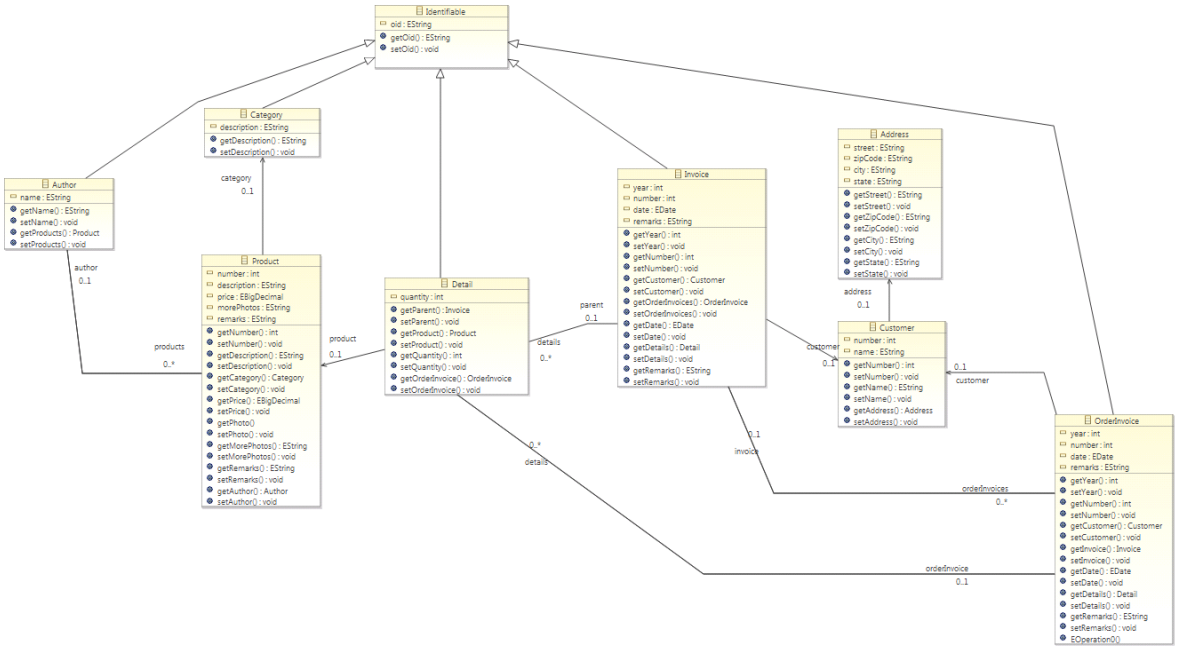


Figura 20.- Prototipo Invoicing-012.

En Invoicing-013 podemos observar cómo se consigue generalizar tanto la factura Invoice como el pedido Order derivando de un ComercialDocument del que cuelgan sus detalles.

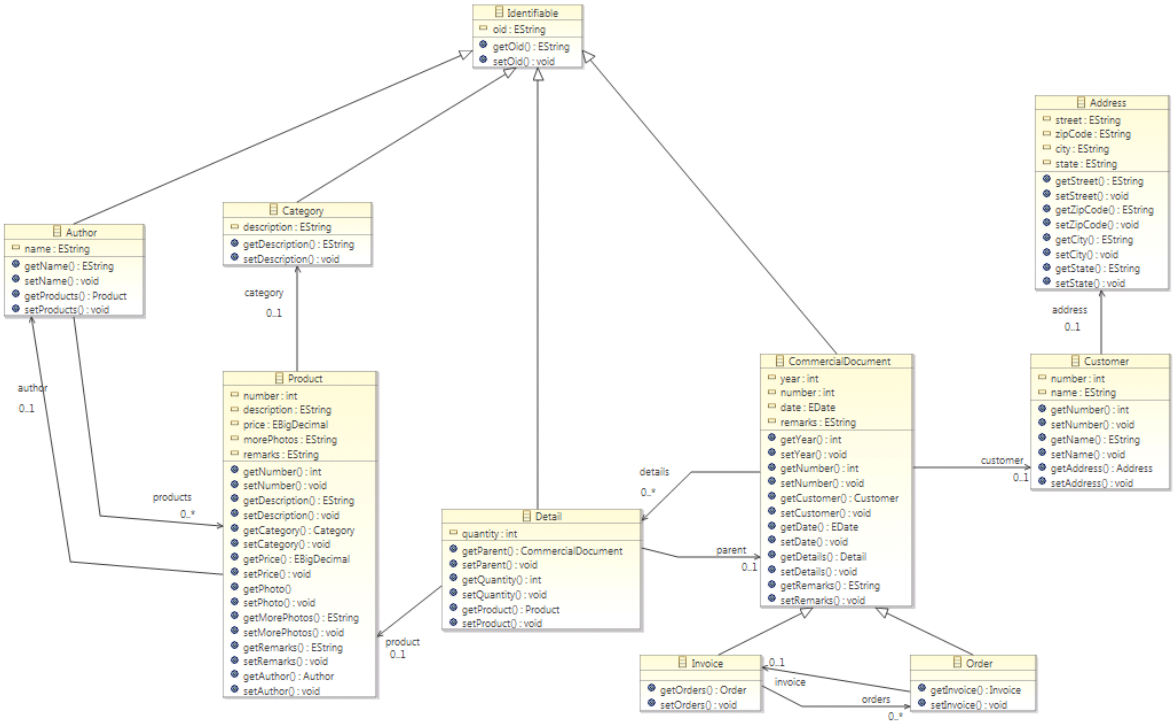


Figura 21.- Prototipo Invoicing-013.

Podríamos seguir añadiendo prototipos ya que se han realizado más de 20 versiones de esta aplicación y además se han utilizado otras dos aplicaciones de ejemplo una muy simple y otra

muy compleja ambas desarrolladas por terceros con objeto de ilustrar las capacidades del framework OpenXava y de testear con otros estilos de programación.

El estudio siguió con las asociaciones reflexivas donde se realizaron pruebas con la interfaz de tipo Tree pero se concluyó que este componente no estaba suficientemente generalizado y no conseguía funcionar correctamente en todos los casos. Se ha diseñado para un caso particular donde la asociación reflexiva ha de estar definida en otra entidad expresa para ello. Ya que el objeto de este trabajo no era resolver bugs al framewok se decidió no seguir con los casos que estudiaban las asociaciones de este tipo.

En cualquier caso la secuencia del caso de estudio ilustra el 90% de los casos posibles en una aplicación web convencional y para nuestro caso el framework cubre las expectativas.

4.2. Identificación de escenarios de evolución: Inventario de patrones.

De los casos del estudio podemos extraer un conjunto de patrones de evolución, o posibles cambios que puedan darse, al menos con cierta probabilidad.

	Renombrar	Crear	Eliminar	Modificar Tipo	Modificar Multiplicidad	Derivar de superclase	Modificar Código	Modificar Signatura
Clase	x	x	x	x		x		x
Atributo	x	x	x	x	x			
Referencia	x	x	x	x	x			
Método	x	x	x	x	x		x	x
Parámetro	x	x	x	x	x			

Tabla 7.- Patrones más comunes de evolución identificados.

Lejos de ser un inventario exhaustivo y formal de patrones de evolución, esta tabla nos permite poner la atención en qué tipo de situaciones se pueden encontrar los anclajes de las anotaciones, cuando se producen cambios estructurales en las clases.

Cabe reseñar que las clases genéricas han sido contempladas pero sus efectos no han quedado del todo analizados en el caso de estudio. Aún así lo reflejamos en la tabla 7 como modificaciones en el tipo o signatura compuesta por los parámetros de la clase.

5. Ingeniería inversa y regeneración del modelo.

He de señalar que cuando se inició este proyecto se pretendía abordar la fase de ingeniería inversa mediante una combinación de reflection, procesamiento de anotaciones vía APT de Java 5 (Annotation Processing Tool) y EMF dinámico para crear los modelos ECORE conformes al código Java; con un resultado similar al que se ha obtenido visualizando los modelos Java como ecorediagrams.

Esta opción de ingeniería inversa fue rápidamente desechada pues el trabajo que ahorraba MoDisco era muy superior al coste de aprender a utilizar y comprender las tecnologías utilizadas por la herramienta. Por no hablar de las ventajas de ofrecer una solución basada en meta modelos estándar de OMG.

5.1 MoDisco

Esta herramienta ha proporcionado el principal componente de la solución, resolviendo los problemas más complejos de ingeniería inversa del código en un modelo completo SASTM de Java. Modelo que incluye código de los métodos, tratamiento de tipos genéricos, anotaciones y comentarios; así como la posibilidad de tratar con artefactos XML, librerías JAR, ficheros JSP/JSF, archivos de configuración web y EJB. En definitiva, una cobertura completa de la infraestructura necesaria para disponer de un modelo conforme al código y artefactos de una aplicación JEE.

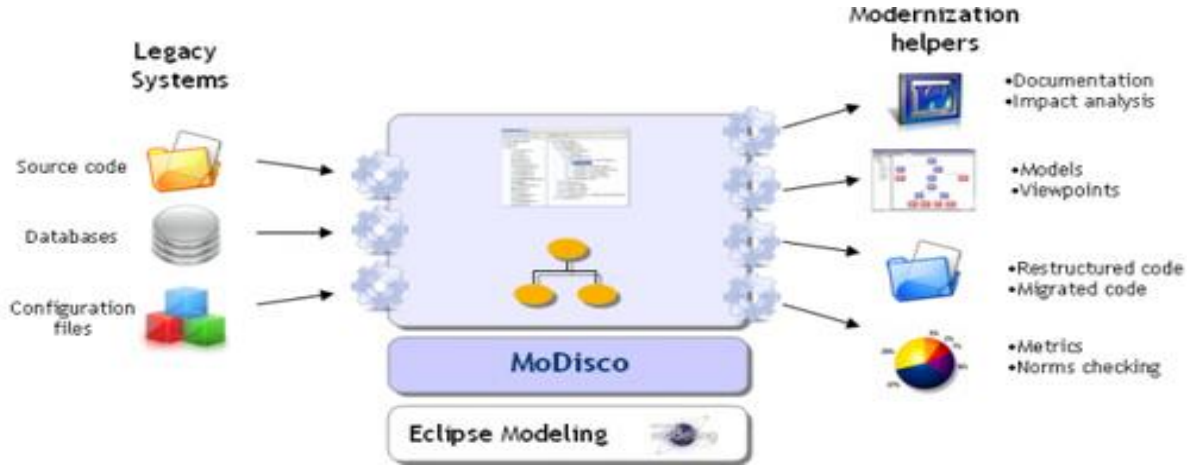
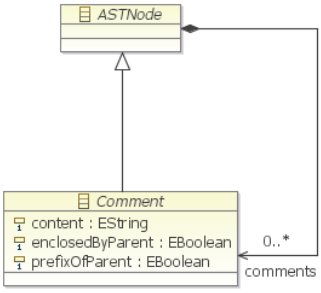


Figura 22.- Características de MoDisco.

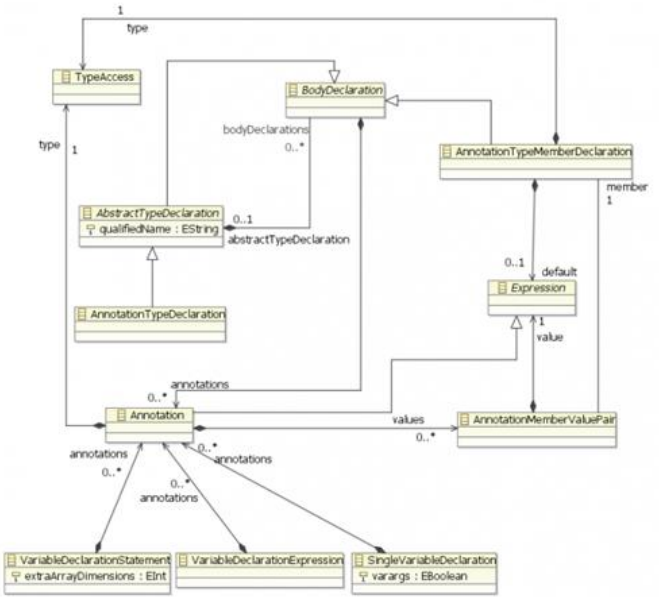
5.1.1. Infraestructura JEE

Como se ha descrito en anteriores puntos aunque la cobertura parecía completa se procedió a comprobar el meta modelo; con el objeto de ver el tratamiento de los puntos clave que afectaban al framework seleccionado:

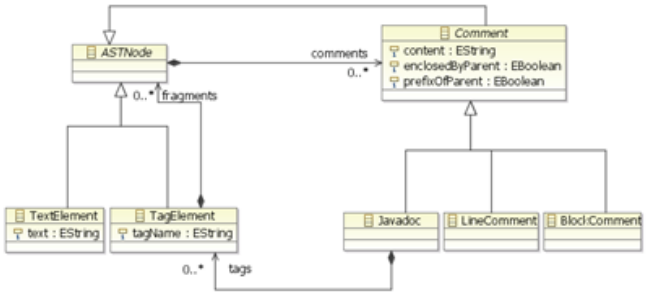
- anotaciones
- comentarios
- tipos genéricos
- referencias sin resolver
- código de los métodos



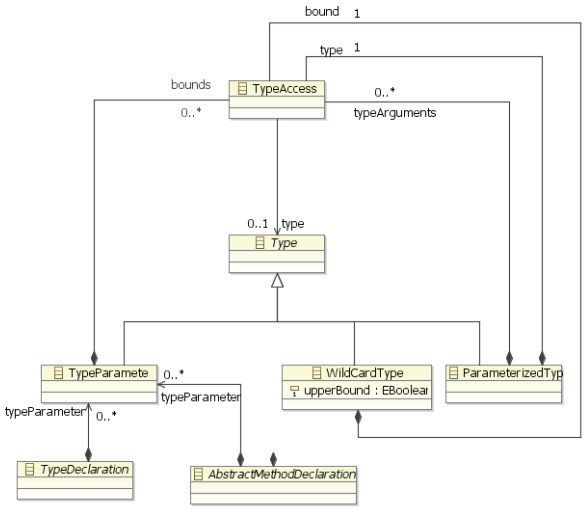
Todo componente del modelo deriva de ASTNode, siendo este clasificador estratégico para definir o extender el meta modelo. Esto también prueba el hecho de que es conforme al estándar definido por OMG para los arboles de sintaxis abstracta, que en este caso define la gramática específica del lenguaje Java.



En este segmento del meta modelo podemos observar cómo son contempladas las anotaciones Java. Si navegamos por el meta modelo podemos comprobar que pueden haber anotaciones en las propias anotaciones, en las declaraciones de las propiedades, los métodos, los parámetros y las clases. Claramente están reflejadas todas las características de las anotaciones como las lista de pares clave/valor y los vectores de miembros.



Los comentarios pueden realizarse en cualquier parte del código como denota el hecho de aparecer como agregado de ASTNode y derivar del mismo.
Se distinguen tres tipos de Comentario: Línea, Bloque y Javadoc. Este último permite incluir los XDoclets utilizados por plataformas como EJB, Struts o Spring.



Otro de los puntos críticos es la cobertura de tipos genéricos, expresamente descrita en esta parte del meta modelo.

Sin embargo se ha encontrado una carencia operativa de este meta modelo. No dispone de ningún elemento EAnnotation con el que etiquetar los elementos del modelo para usos varios como la refactorización.

Figura 23.- Detalles del meta modelo SASTM de Java.

Para ilustrar la cobertura a JEE se pude apreciar los dicoverers disponibles:

- Inventario completo del código JEE con referencias al modelo KDM
- Descubrimiento del código Java
- Descubrimiento del código JSP

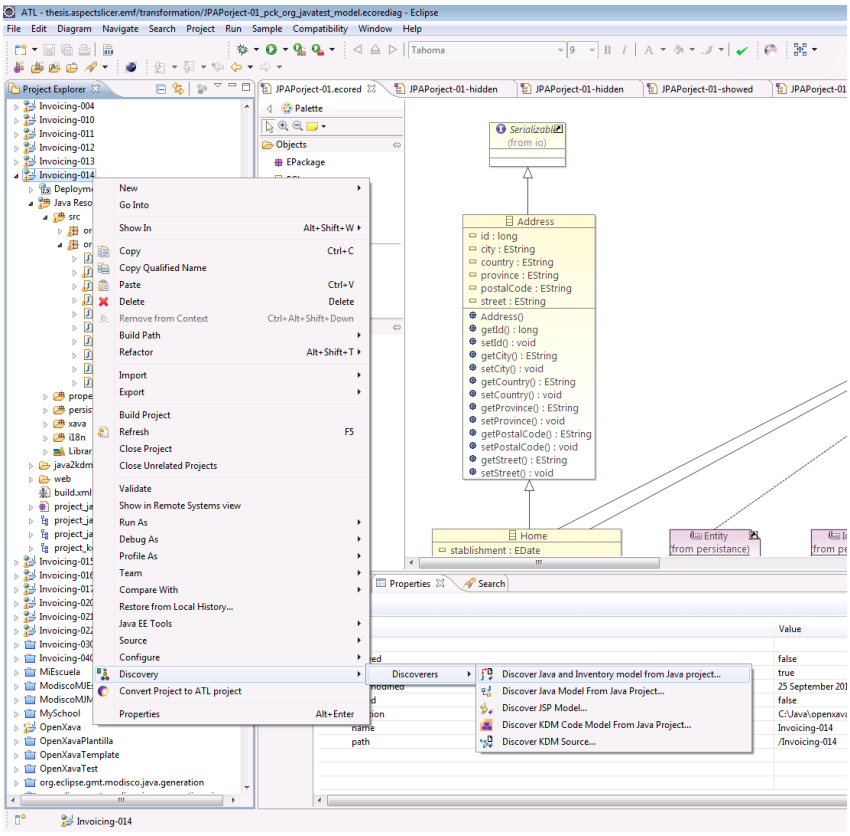


Figura 24.- Discoverers de Java.

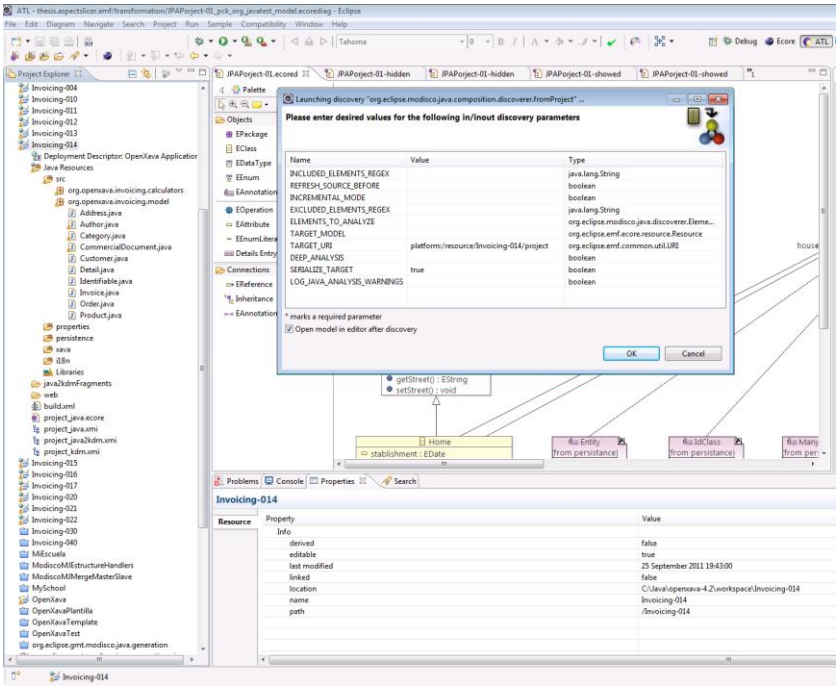


Figura 25.- Dialogo de los discoverers de Java.

Además de los descubridores, MoDisco nos ofrece un servicio de regeneración de código proporcionando así un round-trip completo para JEE.

org.eclipse.gmt.MoDisco.java.generation

```
GenerateJavaExtended javaGenerator = new GenerateJavaExtended(URI.createFileURI("C:/.../my.javaxmi"),
    new File("C:/.../myOutputFolder"), new ArrayList<Object>());
javaGenerator.doGenerate(null);
```

Figura 26.- Regeneración de Java.

Esta fase se ha implementado en MoDisco mediante una plantilla de Acceleo, en la ilustración podemos ver el segmento que genera el texto de un elemento ASTNode.

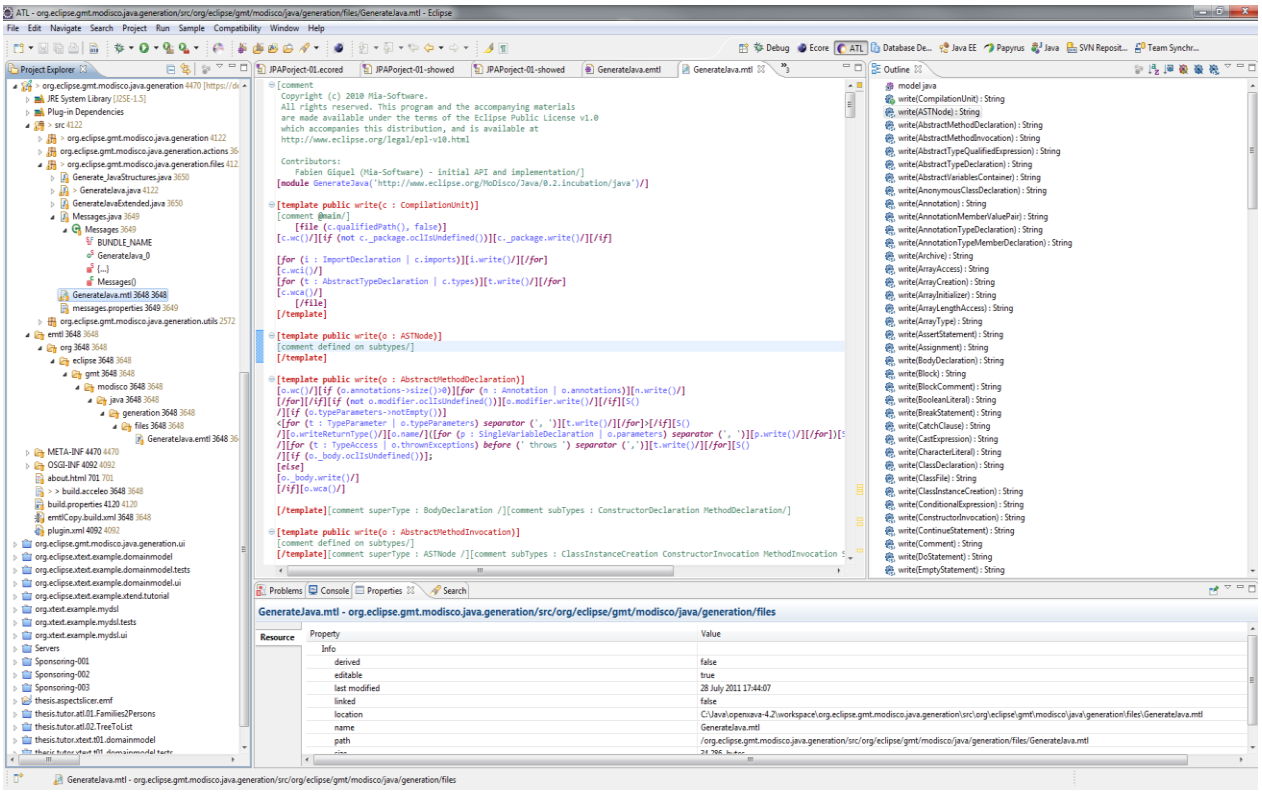


Figura 27.- Plantilla Acceleo M2T para la regeneración del código.

5.1.2. KDM (OMG)

En este punto sólo cabe añadir que la herramienta de referencia elegida por OMG para el soporte a la modernización permite descubrir el modelo KDM de un proyecto JEE de diferentes formas. Y precisamente el hecho de utilizar este componente, proporciona navegabilidad al código desde el browser de MoDisco utilizado para los modelos, como vamos a ilustrar a continuación.

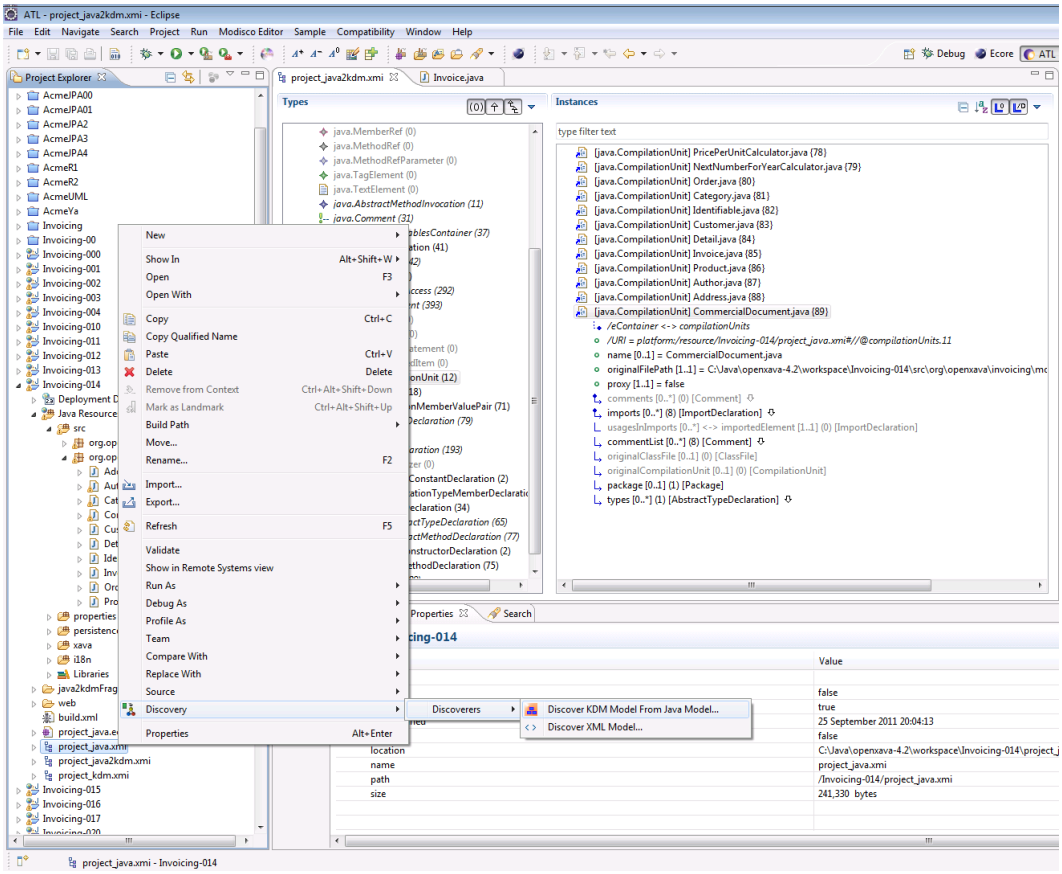


Figura 28.- Discoverers de KDM desde Java.

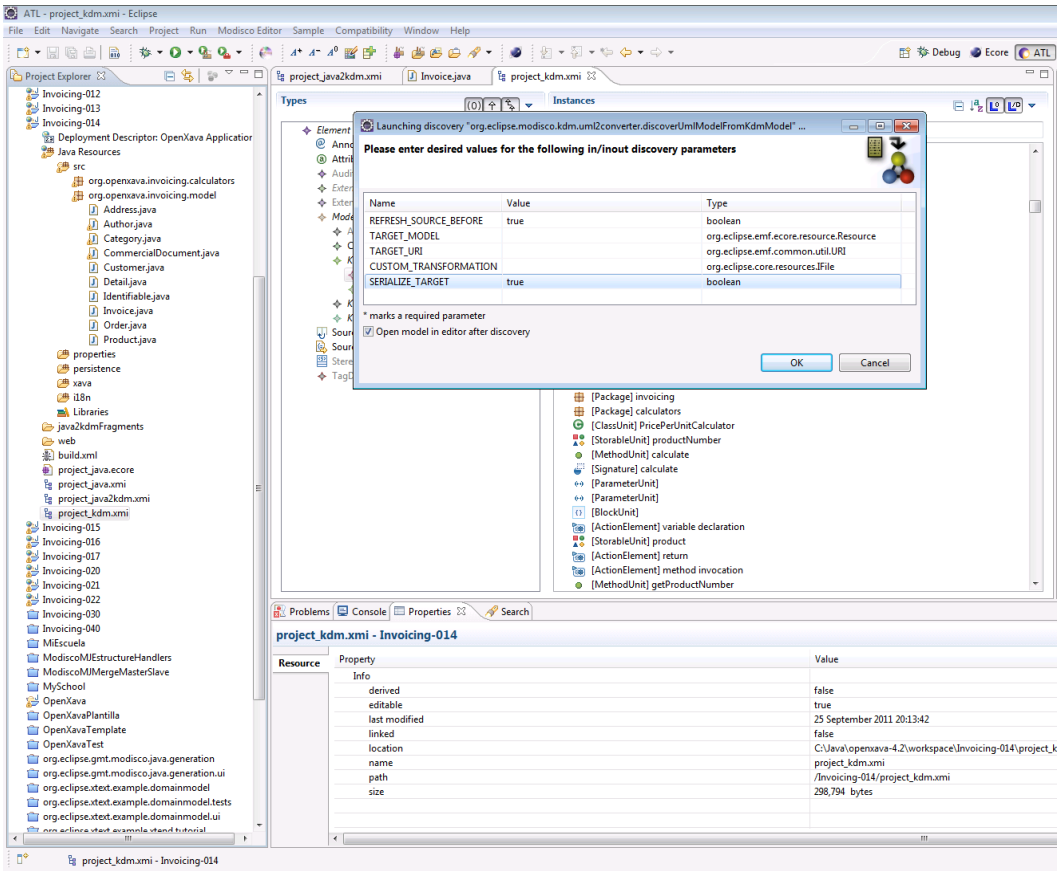


Figura 29.- Dialogo de los discoverers de KDM.

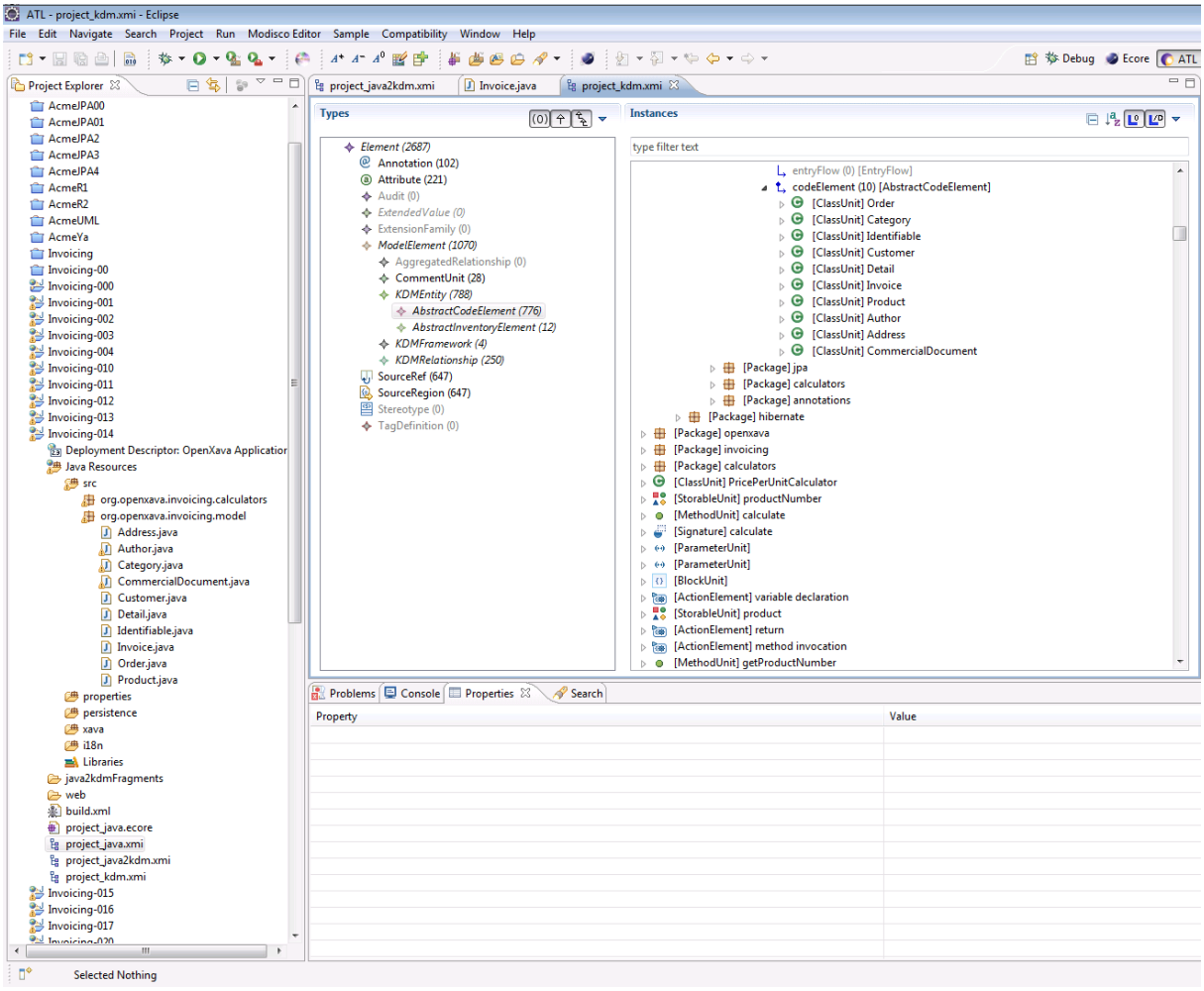


Figura 30.- Browser de MoDisco visualizando el modelo KDM.

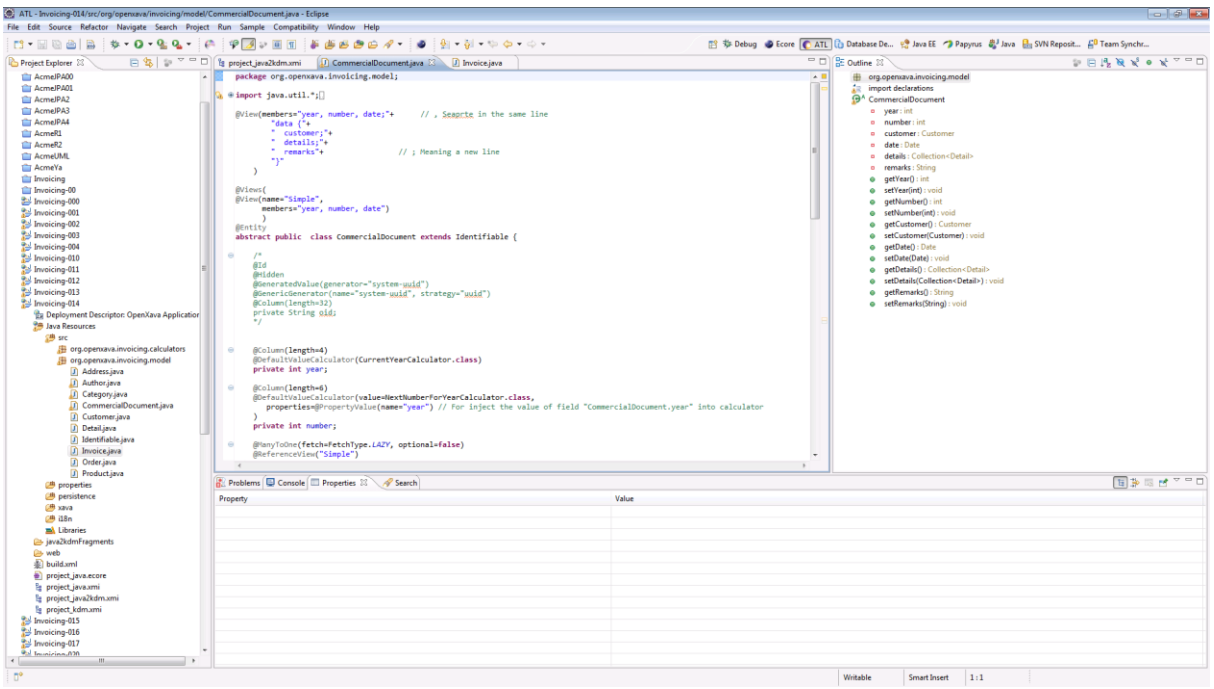


Figura 31.- Navegación al código desde el modelo KDM.

5.1.3 Integración con UML

La herramienta proporciona integración con UML vía una transformación siguiendo el mapeo definido en la tabla 8. Este servicio también está disponible desde los modelos KDM como un discoverer más, como se ilustra en las figuras siguientes.

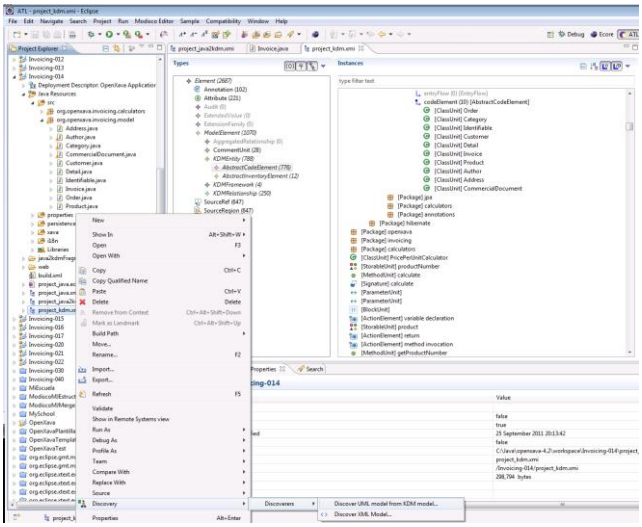


Figura 32.- Discoverer de UML.

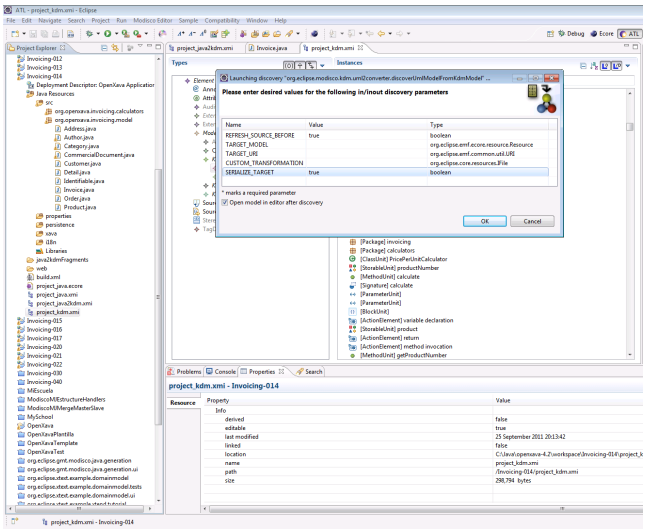


Figura 33.- Dialogo del discoverer de UML.

KDM	UML
LanguageUnit	Package
CodeModel	Model
CodeAssembly	Model
Package	Package
ClassUnit	Class
InterfaceUnit	Interface
MethodUnit	Operation
ParameterUnit	Parameter
Extends, Implements	Generalization
PrimitiveType	PrimitiveType
MemberUnit	Property, Association

Tabla 8.- Mapeo KDM a UML.

Este UML puede ser diagramado por cualquier herramienta UML como Papyrus o Moskitt.

6. Separación y sincronización de aspectos: Implementación de un Round-trip Java Anotado.

6.1. Generalización de la sincronización aspectos.

Para poder generalizar la sincronización de aspectos necesitamos reflejar como se definen y de que se componen. Además estos aspectos pueden evolucionar y deberían de poder organizarse en grupos no disjuntos para conformar vistas especializadas del código-modelo Java.

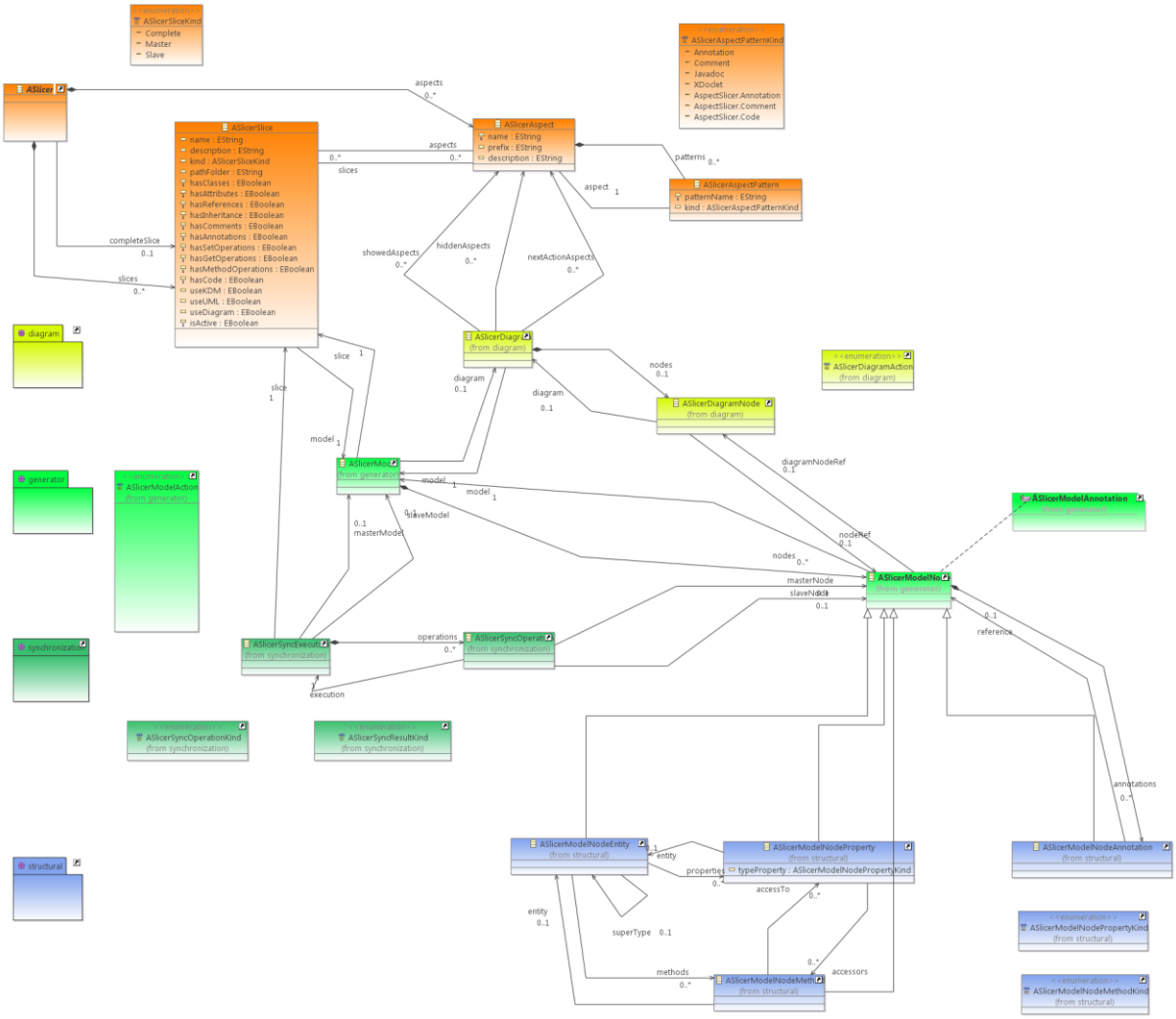


Figura 34.- Metamodelo de AspectSlicer.

6.2. Meta-modelos.

El meta modelo completo, mostrado en la figura 34, está organizado en 5 paquetes: El paquete **Core** donde se definen los aspectos y slices de la instancia. El paquete **Diagram** donde se pretende controlar el estado de los diagramas asociados a cada slice. El paquete **Generator** tiene por misiones de una parte dar soporte al etiquetado de nodos que deben ser comentados en la generación y de otra, mantener el estado de los modelos del slice. El

paquete **Synchronization** proporciona trazabilidad de las operaciones de sincronización**.** Y por último, el paquete **Structural** registra los nodos involucrados en operaciones de la herramienta que actúan como anclajes de anotaciones.

6.2.1.- MetamodeloAspectSlicer

Este meta modelo define que compone cada aspecto y cada slice. Los slices puden ser de tres tipos: el completo o principal, que se corresponde con el proyecto matriz, maestro cuyo role en las operaciones de sincronización es imponer los cambios en la estructura del slice principal, y por último los esclavos que se limitan a decorar al slice principal. Cada slice puede contener o no, diferentes tipos de nodo AST, y además puede estar activado o no. También se indica si genera algún modelo extra asociado (KDM o UML) y si requiere ser visualizado en un diagrama (ecorediagram).

Los aspectos pueden tener un prefijo y una colección de patrones de diversa naturaleza: anotaciones Java, comentarios, Javadoc, XDoclet o bien anotaciones de servicio de la herramienta para marcar anotaciones, comentarios o código. Pero en cualquier caso son un conjunto de patrones de nombres de etiqueta clasificados en diferentes tipos de nodo.

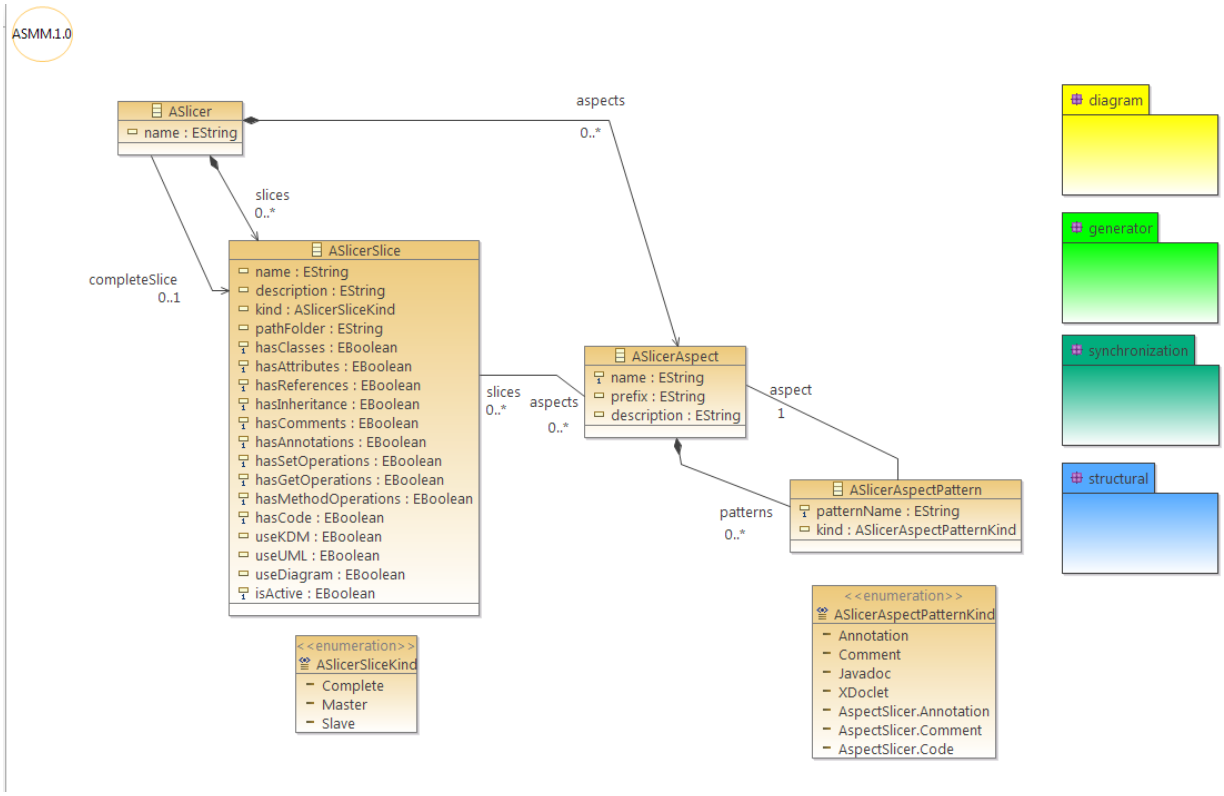


Figura 35.- Metamodelo de AspectSlicer - Core.

6.2.1.1.- MetamodeloAspecSlicer.Diagram

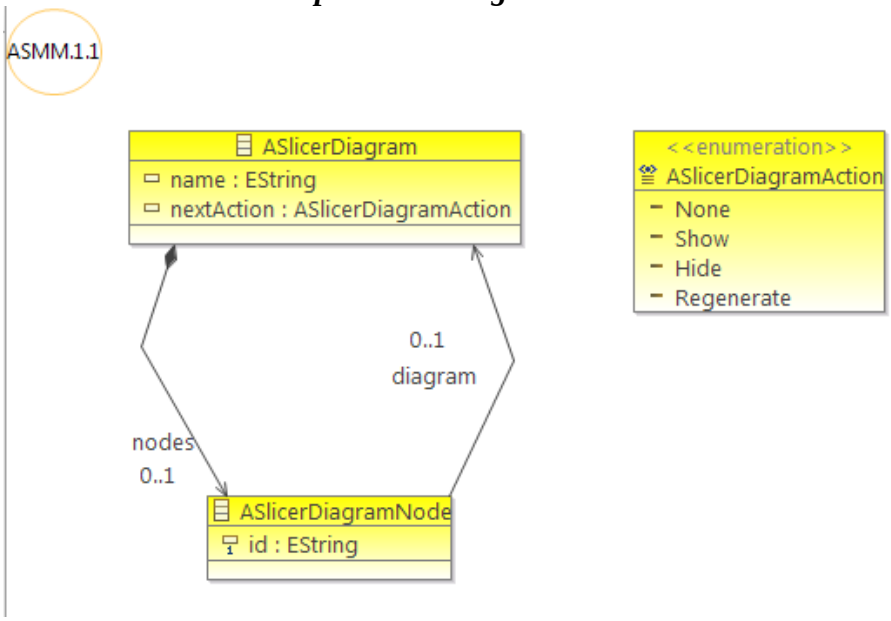


Figura 36.- Metamodelo de AspectSlicer - Diagram.

La función principal es mantener el estado de los diagramas a cada modelo de un slice. Dependiendo del historial de operaciones solicitadas los aspectos asociados a cada slice pueden mostrarse ou ocultarse a criterio del usuario, para ello se mantiene la lista de aspectos mostrados y ocultados.

6.2.1.2.- MetamodeloAspecSlicer.Generator

De un modo similar necesitamos almacenar el estado de los modelos y de sus operaciones. Además cuando generemos utilizaremos los nodos a generar para indicar cuales de ellos han de generarse comentados.

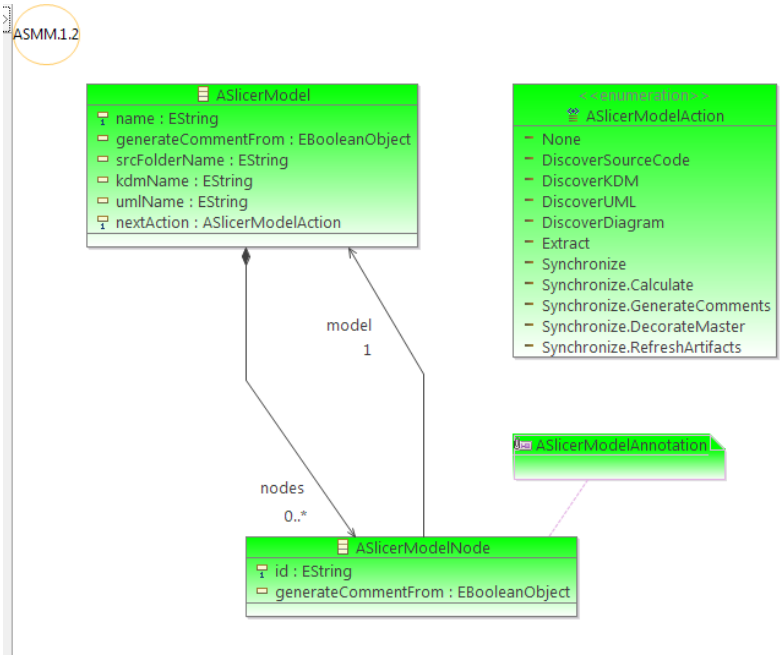


Figura 37.- Metamodelo de AspectSlicer - Generator.

6.2.1.3.- MetamodeloAspecSlicer.Synchronization

En las operaciones de sincronización se procesan infinidad de elementos y necesitamos disponer de una traza de lo ocurrido con cada operación en cada elemento. Si se ha añadido una propiedad o una entidad o bien se ha actualizado. Indicando el resultado de cada operación y en que orden ocurrió.

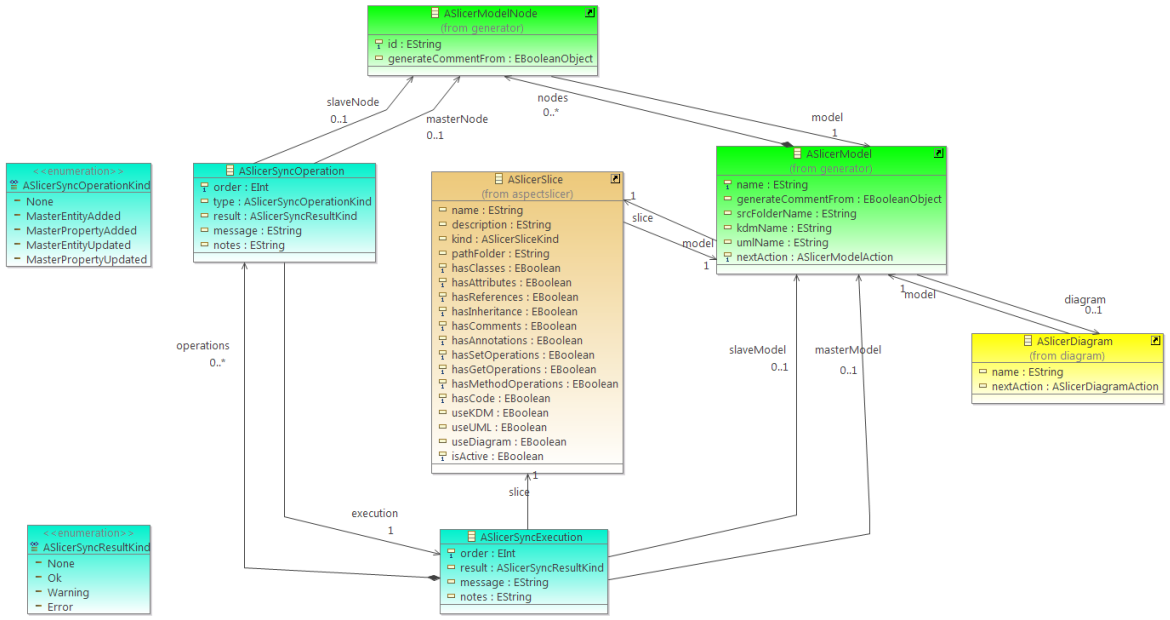


Figura 38.- Metamodelo de AspectSlicer- Synchronization.

6.2.1.4.- MetamodeloAspecSlicer.Estructural

Los nodos afectados en cada operación no solo combiene saber de que modelo proviene sino además que función cumplen en el mismo. Para ello son clasificados según la función que desempeñan en la estructura del código.

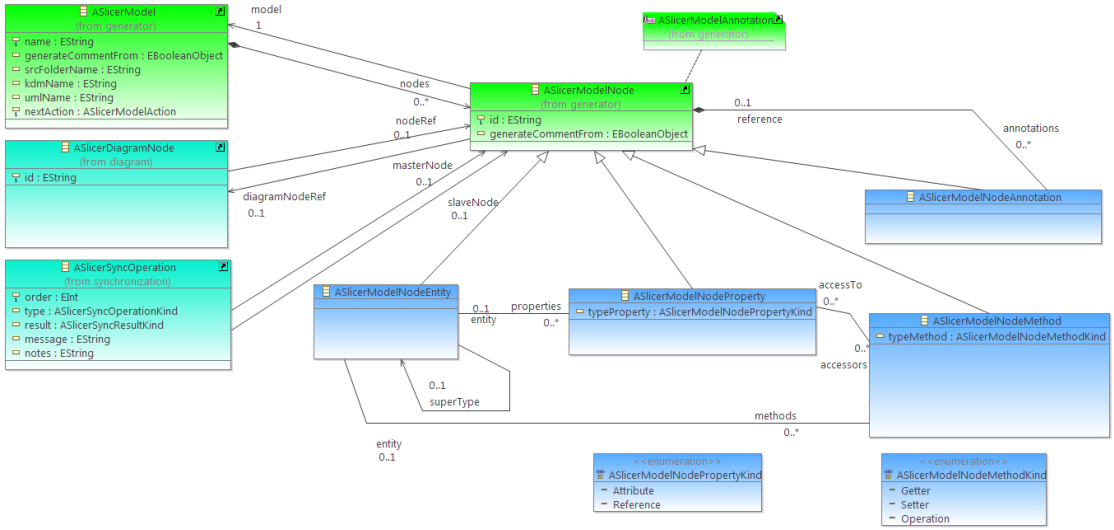


Figura 39.- Metamodelo de AspectSlicer- Estructural.

6.2.2.- Metamodelo Prototipo de AspecSlicer: ListPatterns

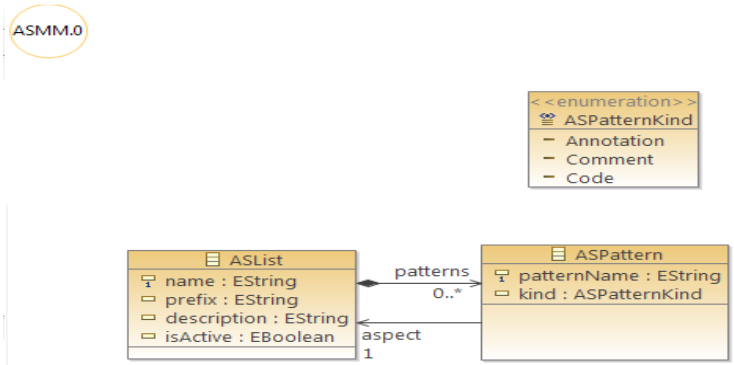


Figura 40.- Metamodelo prototipo de AspectSlicer: ListPatterns.

Con el objeto de poder desarrollar el código de las transformaciones desde una perspectiva sencilla, se diseñó este meta modelo con un subconjunto mínimo de características. Una vez desarrolladas las transformaciones se adaptaron fácilmente al meta modelo inicial.

6.2.3.- MetamodeloAspectSlicer.BuildSlice

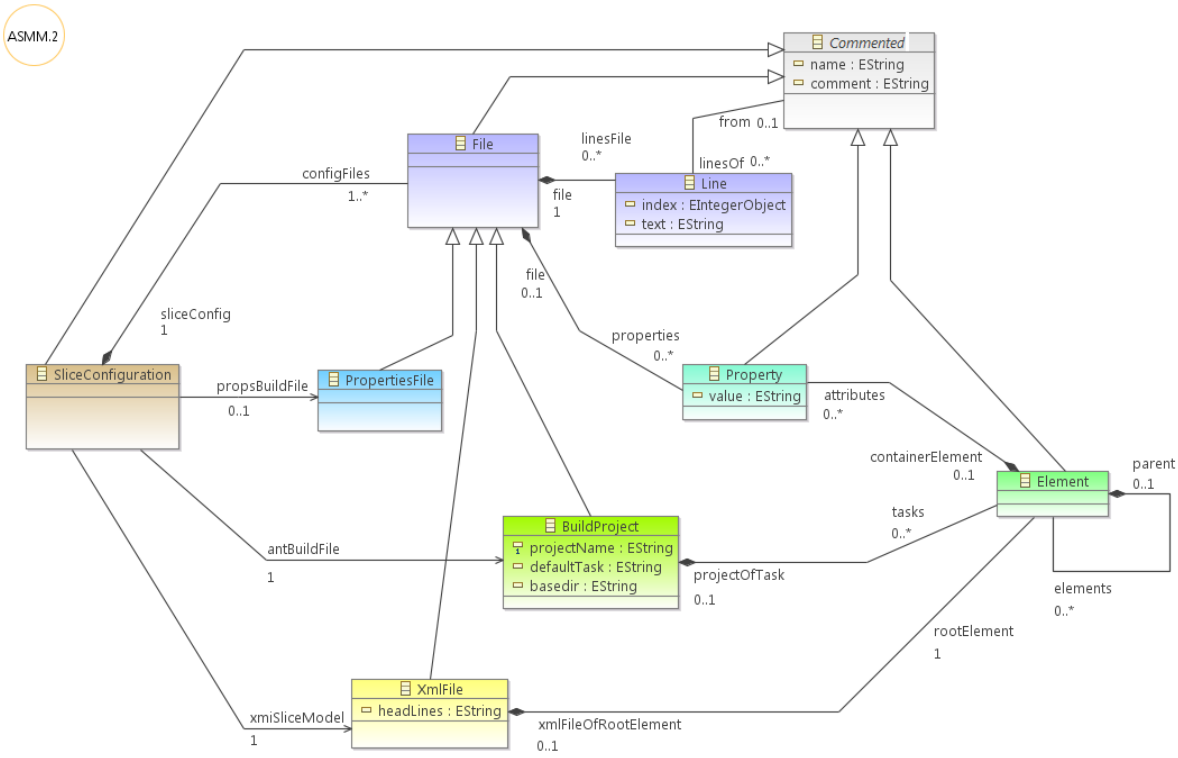


Figura 41.- Meta modelo de AspectSlicer.BuilsSlice.

Con el objetivo de proporcionar datos de configuración de cada slice, en diferentes formatos. Se diseñó este meta modelo donde almacenar los ficheros de configuración necesarios. En primer lugar necesitamos almcenar el fichero de tareas ANT **aslice.build.xml** y de otra su fichero de propiedades asociado **aslice.build.properties**. Ya que pretendemos dar cobertura al intercambio de datos, se contempla la posibilidad de definir ficheros de propósito general de tipo XML y texto plano.

6.3. Transformaciones.

Las transformaciones definidas en la tabla 7 de la solución propuesta, han sido abordadas siguiendo un orden constructivo. Como se indicó entonces, se empezó con la creación de modelos ecorediagram con los que visualizar los modelos Java. A partir de este punto se han ido desarrollando el resto de transformaciones.

6.3.1.- M2M Java To Ecore

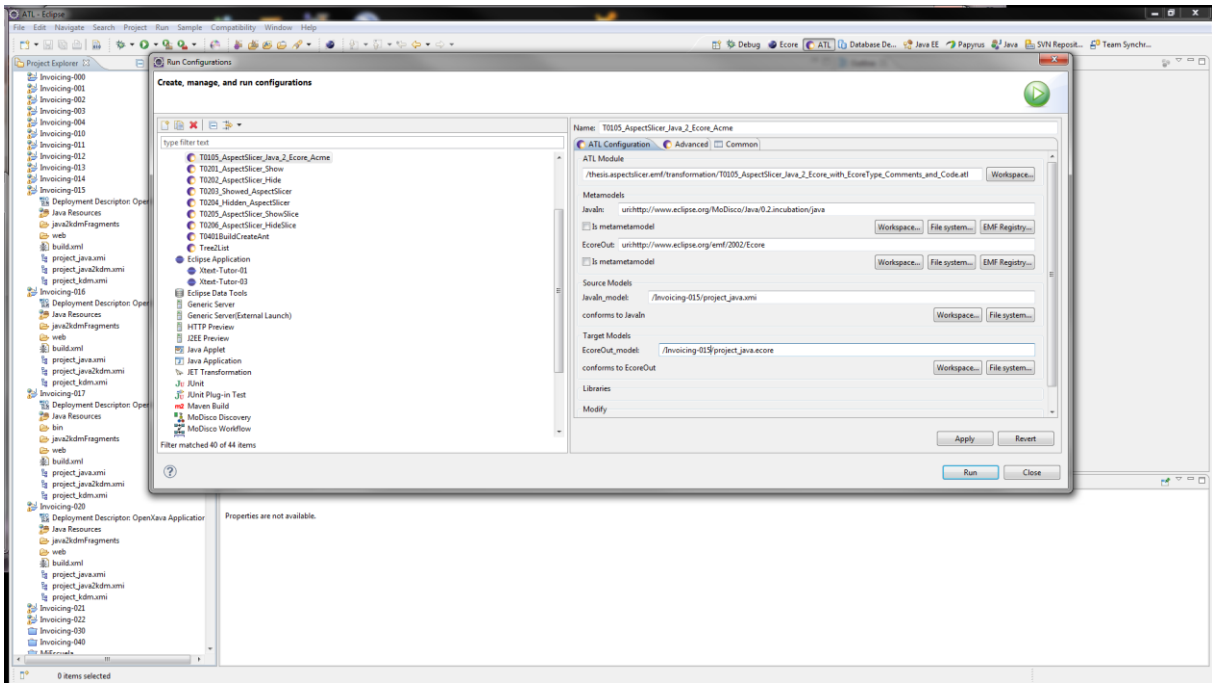


Figura 42.- Ejcución de Java To Ecore.

Para abordar la creación de esta transformación se utilizó la herramienta AMW con la que se definieron las correspondecias entre los meta modelos Java y Ecore. Esta herramienta además proporciona una transformación HOT que nos crea el esqueleto de la transformación ATL que lo implementa.

```
<?xml version="1.0" encoding="ASCII"?>
<Module xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="mmw_qatl" xmi:id="Module1">
  <inputModels xmi:id="InModelRef1" name="mmj" ref="/thesis.aspectslicer.emf/metamodel/java/java.ecore">
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_ClassDeclaration" ref="//ClassDeclaration"/>
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_FieldDeclaration" ref="//FieldDeclaration"/>
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_MethodDeclaration" ref="//MethodDeclaration"/>
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_AnnotationTypeDeclaration" ref="//AnnotationTypeDeclaration"/>
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_Model" ref="//Model"/>
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_name" ref="//Model/name"/>
  </inputModels>
  <outputModels xmi:id="OutModelRef1" name="mme" ref="/thesis.aspectslicer.emf/metamodel/ecore/Ecore.ecore">
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_EClass" ref="//EClass"/>
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_EAttribute" ref="//EAttribute"/>
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_EOperation" ref="//EOperation"/>
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_EReference" ref="//EReference"/>
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_EAnnotation" ref="//EAnnotation"/>
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_EPackage" ref="//EPackage"/>
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_name" ref="//ENamedElement/name"/>
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_nsURI" ref="//EPackage/nsURI"/>
    <ownedElementRef xsi:type="TransformationElementRef" name="ref_nsPrefix" ref="//EPackage/nsPrefix"/>
  </outputModels>
  <rules xmi:id="Rule0" name="Model2EPackageRoot">
    <input xmi:id="InputElement5" name="Model" element="//@inputModels.0/@ownedElementRef.4"/>
    <output xmi:id="OutputElement5" name="EPackage" element="//@outputModels.0/@ownedElementRef.5">
      <bindings xmi:id="Binding2">
        <target xmi:id="ReferredElement4" name="name" element="//@outputModels.0/@ownedElementRef.6"/>
        <source xmi:id="ReferredElement3" name="name" element="//@inputModels.0/@ownedElementRef.5"/>
      </bindings>
      <bindings xmi:id="Binding3">
        <target xmi:id="ReferredElement6" name="nsURI" element="//@outputModels.0/@ownedElementRef.7"/>
        <source xmi:id="ReferredElement5" name="name" element="//@inputModels.0/@ownedElementRef.5"/>
      </bindings>
      <bindings xmi:id="Binding4">
        <target xmi:id="ReferredElement8" name="nsPrefix" element="//@outputModels.0/@ownedElementRef.8"/>
        <source xmi:id="ReferredElement7" name="name" element="//@inputModels.0/@ownedElementRef.5"/>
      </bindings>
    </output>
  </rules>
  <rules xmi:id="Rule1" name="ClassDeclaration2EClass">
    <input xmi:id="InputElement1" name="ClassDeclaration" element="//@inputModels.0/@ownedElementRef.0"/>
```

```
    <output xmi:id="OutputElement1" name="EClass" element="//@outputModels.0/@ownedElementRef.0"/>
  </rules>
  <rules xmi:id="Rule2" name="FieldDeclaration2EReference" description="" isAbstract="false">
    <input xmi:id="InputElement2" name="FieldDeclaration" element="//@inputModels.0/@ownedElementRef.1"/>
    <output xmi:id="OutputElement2" name="EReference" element="//@outputModels.0/@ownedElementRef.3"/>
  </rules>
  <rules xmi:id="Rule4" name="FieldDeclaration2EAttribute">
    <input name="FieldDeclaration" element="//@inputModels.0/@ownedElementRef.1"/>
    <output name="EAttribute" element="//@outputModels.0/@ownedElementRef.1"/>
  </rules>
  <rules xmi:id="Rule3" name="MethodDeclaration2EOperation">
    <input xmi:id="InputElement3" name="MethodDeclaration" element="//@inputModels.0/@ownedElementRef.2"/>
    <output xmi:id="OutputElement3" name="EOperation" element="//@outputModels.0/@ownedElementRef.2"/>
  </rules>
  <rules xmi:id="Rule5" name="Annotation2EAnnotation">
    <input xmi:id="InputElement4" name="AnnotationTypeDeclaration" element="//@inputModels.0/@ownedElementRef.3"/>
    <output xmi:id="OutputElement4" name="EAnnotation" element="//@outputModels.0/@ownedElementRef.4">
      <bindings xmi:id="Binding1" name="annotations">
        <target xmi:id="ReferredElement2"/>
        <source xmi:id="ReferredElement1" name="annotations" description="" element="//@outputModels.0/@ownedElementRef.3"/>
      </bindings>
    </output>
  </rules>
</Module>
```

Figura 43.- Correspondecia AMW de Java To Ecore.

6.3.2.- M2M Show Aspect

Esta transformación está diseñada para mostrar las anotaciones del modelo que se haya seleccionado. El principio utilizado para mostrar y ocultar la EAnnotation se basa en cambiar el contenedor de la anotación sin perder las referencias al elemento propietario. De forma que cuando queremos mostrar una anotación la anclamos al paquete contenedor del elemento, mientras que las anotaciones que no se desean mostrar permanecen ancladas a su elemento propietario original. Para no perder el propietario original se guardan en la referencias de la anotación al elemento contenedor original, seguida por las referencias que ya tenga definidas. De forma que la primera referencia de la anotación siempre sea el elemento propietario.

6.3.2.- M2M Hide Aspect

Esta transformación está diseñada para ocultar las anotaciones del modelo que se haya seleccionado. El principio utilizado para ocultar la EAnnotation, es el complementario al descrito en la transformación anterior. Y el cambio de contenedor de las anotaciones a ocultar pasa por asignarlas de nuevo a la primera referencia de la anotación ya que siempre será el elemento propietario.

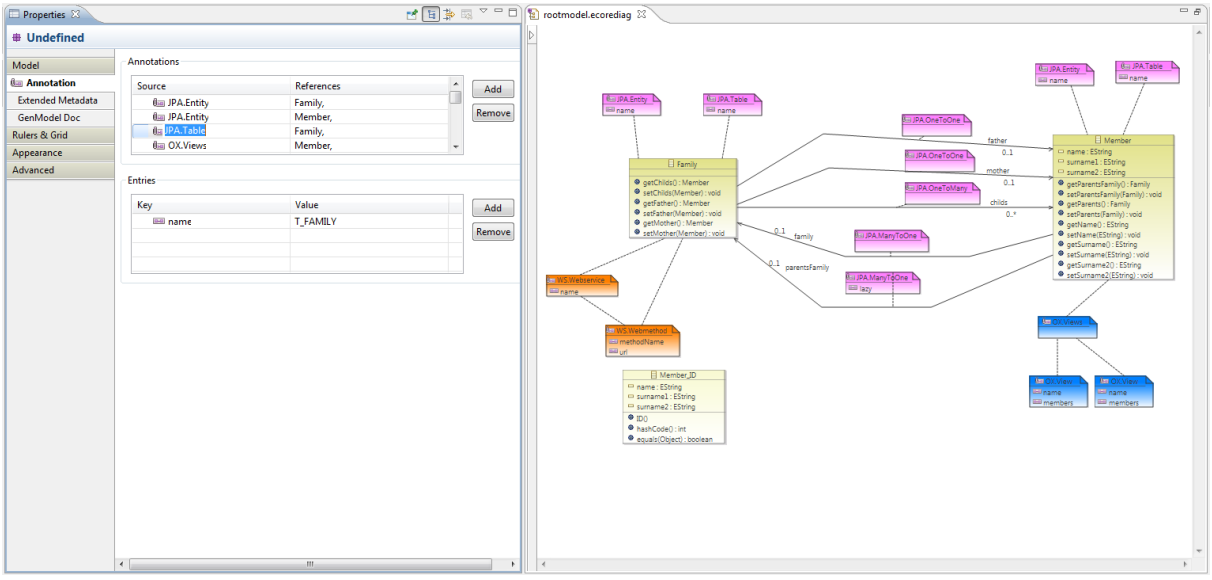


Figura 44.- Detalle de la anotaciones mostradas y ocultadas.

6.3.2.- M2M Extract Slice

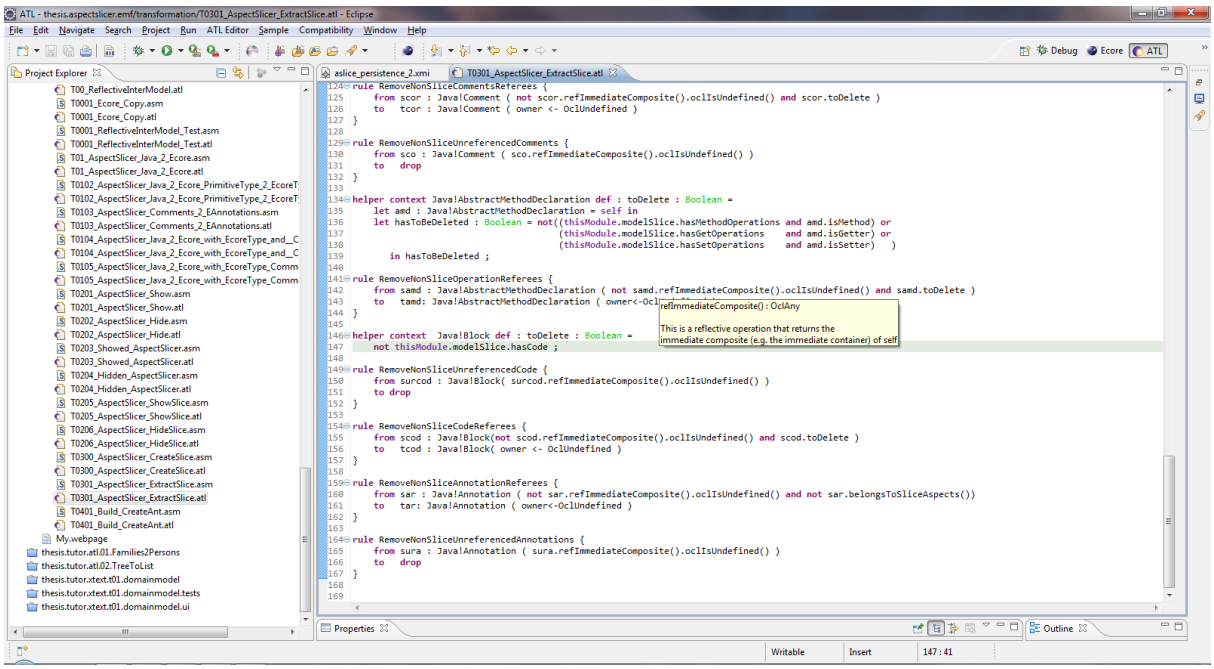


Figura 45.- Detalle de la anotaciones mostradas y ocultadas.

6.3.3.- M2M Synchro Slice

Esta transformación no ha podido ser implementada por falta de tiempo. Pero siguiendo como ejemplo la anterior transformación y aplicando la búsqueda de los elementos de anclaje utilizando un identificador sintético se obtiene la solución.

6.3.4.- M2T Java Generation With Comments

Para la generación de comentarios se ha modificado la plantilla asignada al elemento ASTNode se comprueba si este elemento esta anotado para ser generado como comentario.

```
[template public write(o : ASTNode)]
[if (not o.eAnnotations()->exists( asAnn : EAnnotation | asAnn.oclAsType(AspectSlicerComment)))]/* @@AspectSlicerComment@ [/if]
[comment defined on subtypes/]
[if (not o.eAnnotations()->exists( asAnn : EAnnotation | asAnn.oclAsType(AspectSlicerComment)))]   @AspectSlicerComment@@ */[/if]
[/template]

[template public write(o : AbstractMethodDeclaration)]
[o.wc()/][if (o.annotations->size()>0)][for (n : Annotation | o.annotations)][n.write()/]
[/for][/if][if (not o.modifier.oclIsUndefined())][o.modifier.write()/][/if][S()
```

Figura 46.- Plantilla Acceleo modificada para generar comentarios.

6.3.5.- M2M/M2T Build Slice

Esta transformación es una transformación ATL que produce el texto de los ficheros definidos en el modelo. Básicamente es una M2M en modo refining que vuelca a fichero la concatenación de las líneas calculadas.

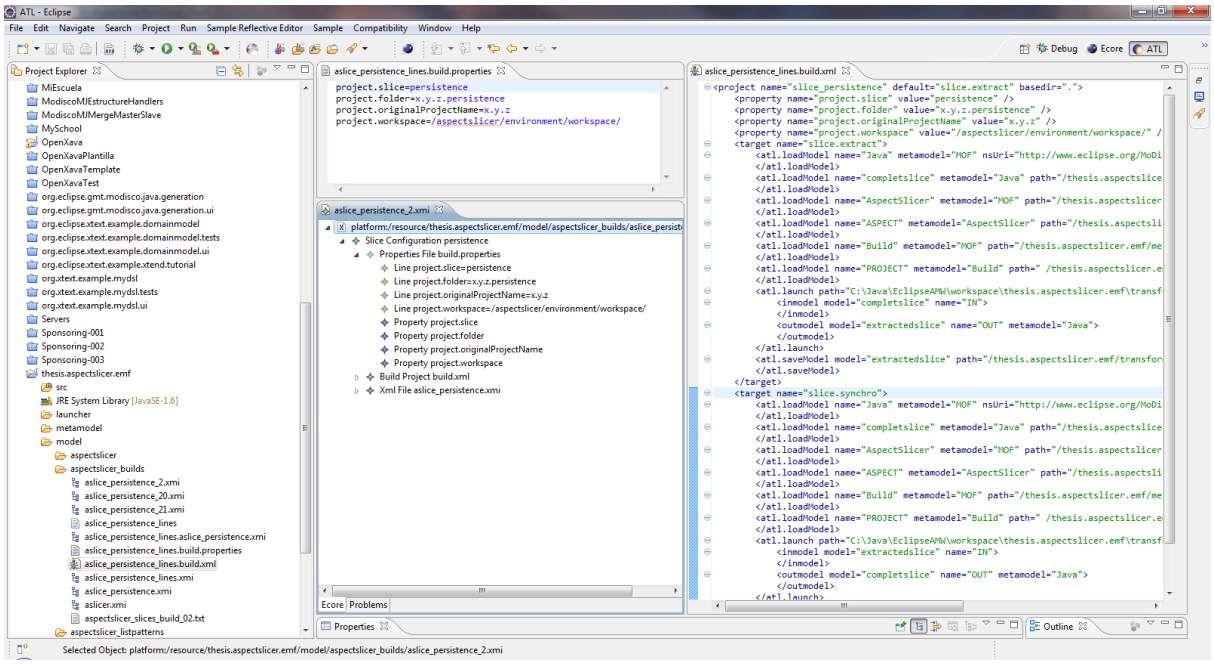


Figura 47.- Build Slice: modelo, ficheros de tareas ANT y propiedades generados.

7. Aspecto de la Persistencia

7.1. Meta-información

La vista de la persistencia está definida como “maestra” ya que puede modificar la estructura de las entidades y sus relaciones. Además en este caso se tiene previsto que el editor utilizado sea Dali, el cual actúa directamente sobre el código fuente.

Desde el AspectSlicer definimos la persistencia con el siguiente conjunto de anotaciones:

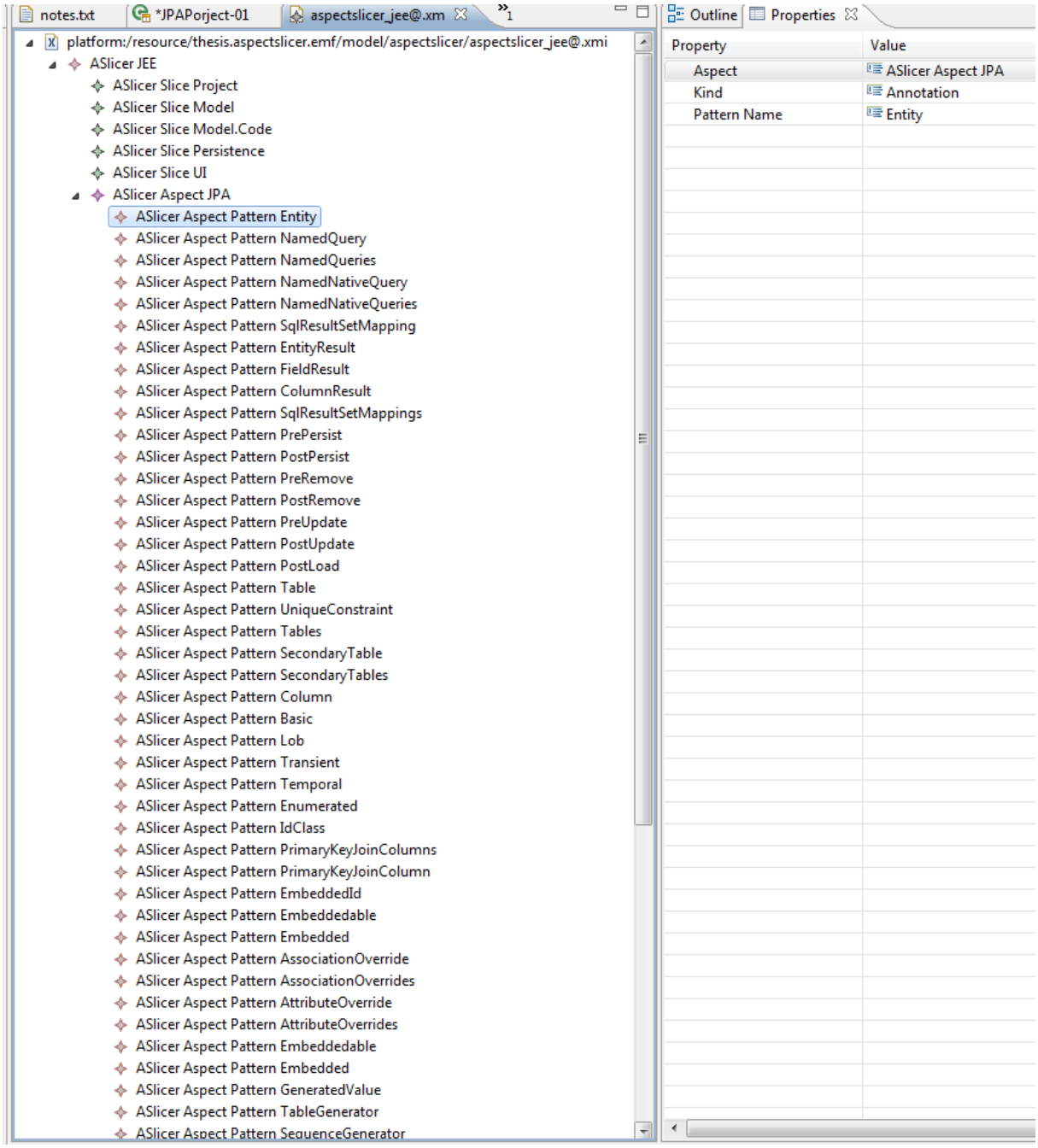


Figura 48.- Anotaciones definidas en el aspecto JPA.

7.2. Solución: Integración con Dali Editor

El editor de JPA proporcionado por Dali para Eclipse cubre todas las necesidades con una excelente calidad y usabilidad. Dispone principalmente de un editor gráfico capaz de ser alimentado con las entidades existentes en la fuente de datos especificada la configuración de persistencia JPA. Cada una de las entidades dispone de una toolbar gráfica enriquecida y un menú contextual que permite refactorizar el código fuente, acceder a los detalles de persistencia, abrir la vista en miniatura e incluso navegar al código fuente.

Es en la vista de detalles de JPA donde se aprecia un editor textual de toda la meta información manejada por JPA tanto para las entidades como para sus propiedades como para con sus asociaciones.

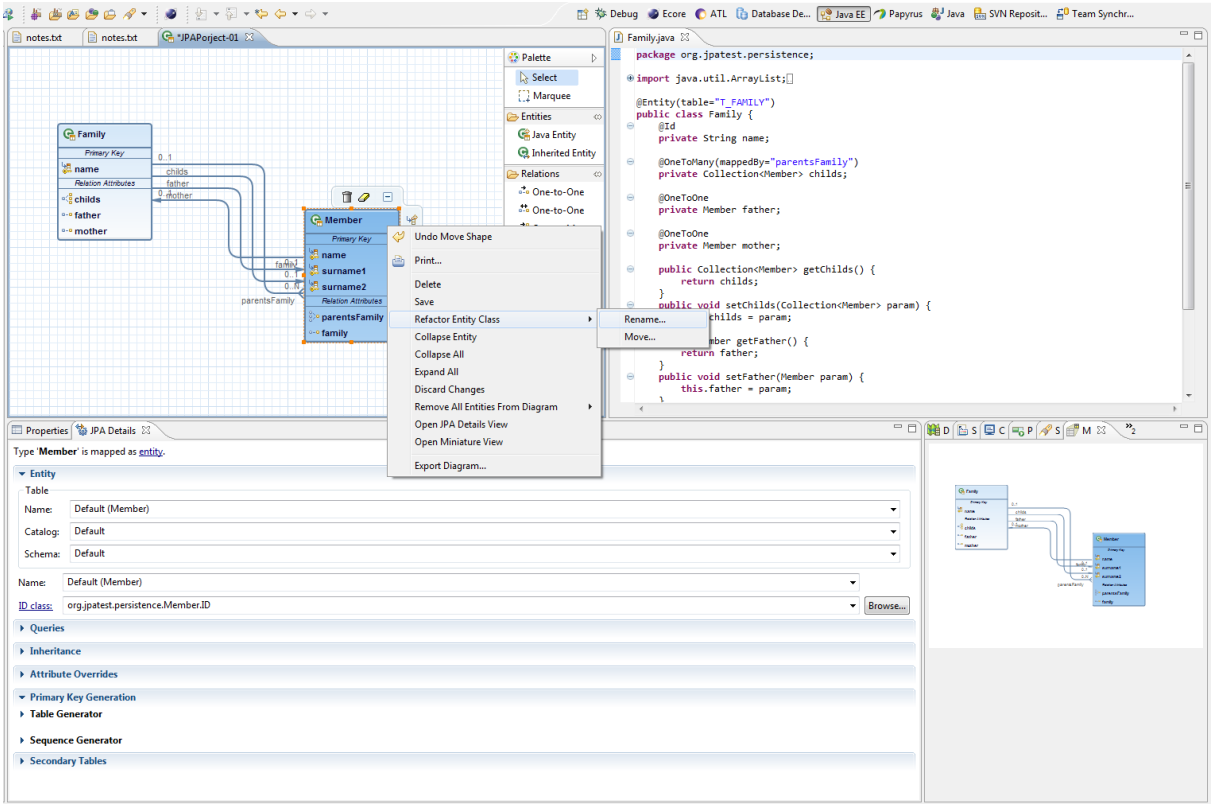


Figura 49.- Editor de JPA: Detalles de JPA, Vista Miniatura, Navegación al código y Menú Contextual.

Este editor es ofrecido con Eclipse desde la distribución de Helios, pero está especialmente integrado a partir de la versión de Indigo. Este editor esta implementado sobre la librería gráfica Graphiti y es un claro ejemplo de su potencia y usabilidad. El editor actúa directamente con el código fuente y con la meta información obtenida de la conexión de persistencia.

8. Aspecto de la Interfaz de Usuario

Este aspecto al contrario que la persistencia es un "esclavo" o "decorador" de la estructura establecida por otras vistas del sistema como es el caso de la persistencia.

8.1. Meta-información

Toda la meta información requerida por OpenXava para distribuir los componentes de la interfaz de usuario vienen definidos por dos pares de anotaciones:

- Views & View y
- Tabs & Tab.

Las primeras definen la distribución especial de cada uno de los componentes de la entidad en base a su orden de aparición dentro de las secciones definidas por el texto contenido en el parámetro @View.members donde se expresa un secuencia de líneas de componentes separadas cada línea de la siguiente por ";". Y donde cada campo o accesor a una propiedad del mismo viene separado del siente por una ",". Además pueden definirse marcos o bloques de campos incluyendo los entre corchetes precedidos por una etiqueta o titular del marco <etiqueta-del-marco> "[" <campo-1>, "[" <campo-2>, … "]". Así mismo también pueden definirse pestañas con una sintaxis similar pero incluyendo los campos entre llaves: <etiqueta-de-la-pestaña> "[" <campo-1>, "[" <campo-2>, … "]". El problema a la hora de construir un analizador para extraer los metadatos, reside en el hecho de que las etiquetas que preceden a los marcos y a las pestañas pueden contener espacios y no llevan

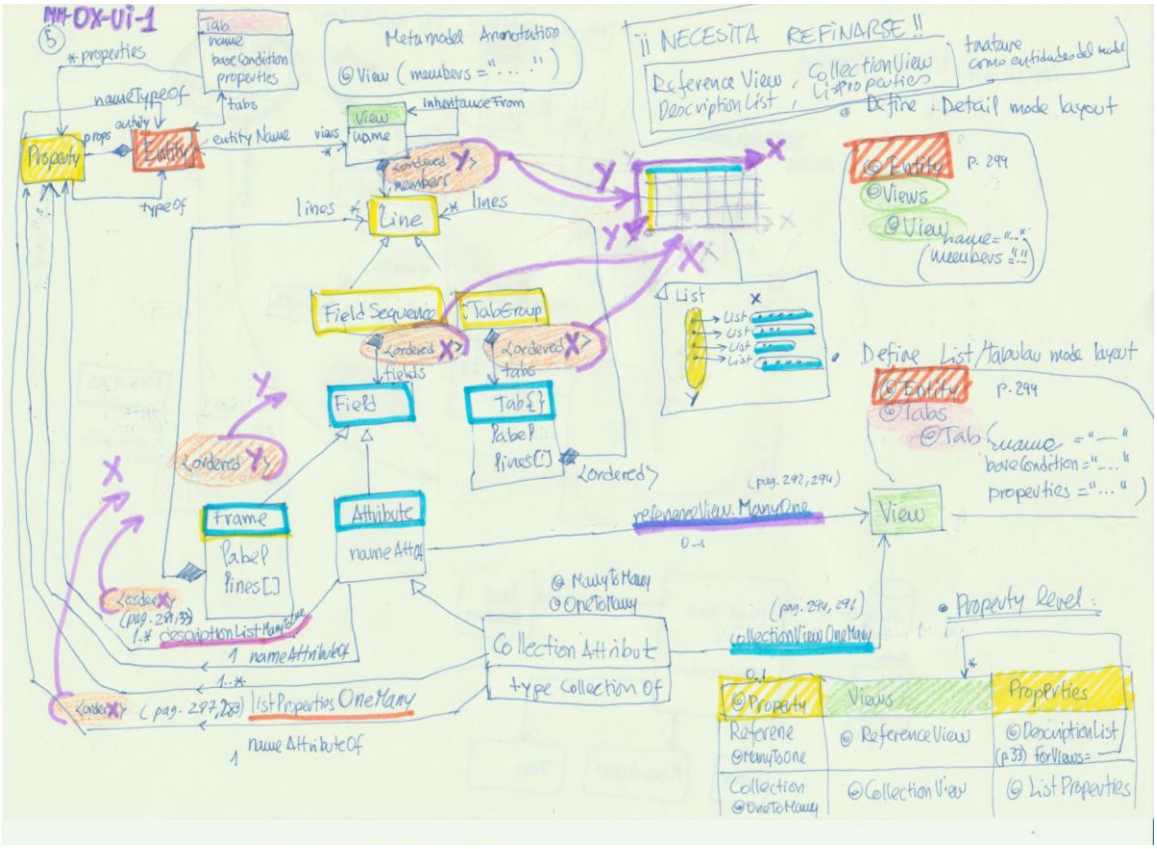


Figura 50.- Borrador de trabajo del meta modelo de la interfaz de usuario de OpenXava.

ningún carácter delimitador por lo que la gramática deja de ser LL(*) y se dificulta el uso de generadores de analizadores sintáctivos como ANTLR o similares.

Aunque no se ha podido abordar el problema por limitaciones de tiempo se ha estudiado cómo resolverlo. Y la solución más sencilla viene por definir un analizador sintáctico con doble puntero de token para eliminar la indeterminación. La implementación del analizador ha de realizarse en una transformación M2M con el objeto de crear el modelo de la interfaz de usuario a partir del texto contenido en la anotación anterior.

Las segundas definen el orden de aparición de los campos seleccionado en las vistas tabulares que se definan para la entidad.

Tanto en un caso como en el otro cada componente tiene suficiente meta información para indicarle al framework cómo debe ser presentado.

8.2. Editor gráfico

El editor gráfico de la interfaz de usuario no es un problema sencillo a resolver por lo que se requiere un esfuerzo que se queda fuera de alcance.

9. Conclusiones.

9.1. Expectativas y resultados.

Este proyecto pretendía integrarse como una herramienta o conjunto de ellas, capaz de automatizar el round-trip de aplicaciones Java anotadas. Desde el punto de vista de la implementación no ha habido tiempo para implementar la transformación de sincronización ni la actualización del Ecore al modelo Java. Pero por otra parte se ha podido recorrer el resto de pasos requeridos por la herramienta. Se han creado alternativas para visualizar y diagramar de forma semi-automática los modelos Java y se ha ideado un mecanismo para compartir información entre transformaciones, tareas ANT, plug-ins y ejecutables.

Desde la perspectiva formativa, este proyecto ha supuesto una toma de contacto real con la producción de software model-driven en el ámbito de la evolución. Y el hecho de tener que integrar herramientas de diferentes fuentes y versiones de Eclipse ha dificultado el normal proceso del proyecto.

Por otra parte el estudio del arte a nivel de herramientas ha servido de oportunidad para actualizar su conocimiento y probarlas, dando la oportunidad de adquirir una experiencia que de otro modo no se hubiera podido dar.

El meta modelo SASTM de Java podría disponer de una EAnnotation de servicio con el objeto de facilitar tareas de manipulación de sus elementos. Extendiendo la idea sería conveniente en cualquiera de los meta modelos utilizados por la herramienta ADM. De esta forma podrían aplicarse un etiquetado de sus elementos facilitando las operaciones de evolución.

Otra mejora no tan obvia es el hecho de que en el ámbito de la evolución, no existen fronteras bien delimitadas entre M2M, M2T o T2M. Por lo que convendría poder invocar a una función M2T desde una M2M por ejemplo para comentar un elemento, o bien invocar una operación T2M como podría ser un analizador ANTLR con el objeto de extraer meta información de cadenas textuales, como es el caso del texto contenido en la anotación @Views.members que determinan el layaout de los componentes de las entidades.

9.2. Aplicación industrial.

En el comienzo del proyecto se contacto con el equipo de Moskitt para averiguar si tenían planificado abordar la fase de ingeniería inversa para esta plataforma. En ese momento la herramienta no disponía ni tenía entre sus prioridades abordar el proceso de round-trip por lo que tras consultarse a la jefe del proyectome indicó que no tenían planificado abordarlo y seria de interesante una colaboración en este punto.

Por otra parte, el framework JEE elegido, OpenXava, carece de un editor gráfico para Eclipse que se integre just-in-time con el código fuente. Es decir, si modificamos el código vemos su efecto en la aplicación, pero carece de un modelo que proporcione una vista simplificada del sistema, bien de su estructura, bien de su interfaz, bien de su persistencia o de cualquier otro aspecto que requiramos.

9.3. Proyección investigadora.

A falta de un estudio sistemático del estado del arte no aparecen demasiadas herramientas de modernización relacionadas con las palabras claves "aspect slicing" por lo que parece no estar demasiado tratado. Esto no indica que no haya herramientas que lo soporten pero al menos no se han encontrado ni demasiadas publicaciones ni referencias desde Eclipse.

Otros caminos que quedan abiertos con este trabajo:

- construir un plug-in que pueda ofrecer esta herramienta como componente de MoDisco.
- estudiar en profundidad las opciones e impacto de la refactorización del código en las declaraciones de variables, parámetros y propiedades definidas en una clase.
- extender el problema y la herramienta en el campo de "Aspect Slicing" desde un enfoque dirigido por modelos.
- integrar la herramienta con otros frameworks generativos como MinuteProject, Texeo y/o Moskitt.
- trabajar en la integración de plataformas M2T y T2M desde transformaciones M2M.

10. Bibliografía.

Model driven reverse engineering. Rugaber, S., Stirewalt, K. - IEEE Software 21(4), p45-53 (2004)

Foundations of model (driven) (reverse) engineering : Models - episode i: Stories of the fidus papyrus and of the solarus. Favre, J.M. Dagstuhl, Germany (2005)

How to Deal with your IT Legacy? Reverse Engineering using Models - MoDisco in a Nutshell! . Hugo Brunelière - JavaTech Journal #10

OMG Architecture Driven Modernization. website (2011). URL http://adm.omg.org

API2MoL: Automating the Building of Bridges between APIs and Model Driven.

Javier Luis Cánovas Izquierdo, Frédéric Jouault, Jordi Cabot, y Jesús García Molina - Information and Software Technology (IST)

A Domain Specific Language for Extracting Models in Software Modernization. Javier Luis Cánovas Izquierdo, Jesús García Molina. - European Conference on Model Driven Architecture - Foundations and Applications, 82-97, 2009

Gra2MoL: Una Herramienta para la Extracción de Modelos en Modernización de Software. Javier Luis Cánovas Izquierdo, Jesús García Molina - JISBD, 162-165, 2009

A Simple Mathematically Based Framework for Rule Extraction from an Arbitrary. Ramsey, F.V.; Alpigini, J.J.; Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International 2002

A brief survey of program slicing. - B Xu, J Qian, X Zhang, Z Wu…; ACM SIGSOFT Software 2005

Context-free slicing of UML class models. - H Kagdi, JI Maletic; IEEE International Conference on Software Maintenance (ICSM'05)

Eclipse Plug-ins. Third Edition Eric Clayberg Dan Rubel

Aprende OPenXava con ejemplos. Javier Paniza

Generación de un plan de migración de datos entre esquemas conceptuales OASIS generados con OO-Method/CASE. Jennifer Pérez Benedí.