IMPACTO DE LA TRANSFORMACIÓN DE MODELOS EN EL DESARROLLO DE SOFTWARE

Dr. Severino Feliciano Morales[1], M en C. Edgardo Solís Carmona[2], Dr. José Luis Hernández Hernández[3], Dr. Mario Hernández Hernández[4] y M. en C. Valentín Alvarez Hilario[5]

***Resumen*—En los últimos años, la Ingeniería de software dirigida por modelos (MDSE, Model-Driven Software Engineering, por sus siglas en inglés o simplemente MDE) ha ganado cada vez más aceptación, principalmente por su capacidad para abordar la complejidad del software y mejorar la productividad. Es un paradigma dónde los artefactos principales son los modelos, los que permiten elevar el nivel de abstracción y aumentar el nivel de automatización. Además estos, nos permiten abstraer, representar y automatizar los problemas del mundo real que se pretenden resolver y permiten que el código se pueda generar automáticamente, ya que los modelos pueden ser construidos, compilados, incluso ejecutados. Existe un mecanismo que es vital en MDE, mediante el cual se pueden construir modelos más refinados con pocas líneas de código, si los que tenemos aún no contiene los suficientes elementos para resolver el problema.**

En este artículo, se pretende describir la transformación modelo a modelo (M2M), mediante un caso de estudio y las herramientas de MDE, para mostrar las bondades de este mecanismo en el proceso de desarrollo de software.

***Palabras clave*—Transformación de modelos, MDE, modelo, metamodelo, DSL.**

Introducción

Desde la década pasada, el desarrollo de software dirigido por modelos ha logrado despertar el interés de manera considerable, tanto en el ámbito académico como en el ámbito industrial, se enfoca principalmente en incrementar el nivel de abstracción y automatización en todas las actividades de la Ingeniería de Software y sus principales artefactos son los modelos, los cuales han mostrado su fuerte potencial para mejorar la calidad y productividad de nuevos desarrollos de software, reingenierías y configuración de sistemas dinámicamente en tiempo de ejecución. El uso sistemático de modelos en las diferentes etapas del ciclo de vida ha llegado a ser la base de un conjunto de paradigmas de desarrollo de software que conforman lo que se ha denominado Ingeniería de Software Dirigida por Modelos o simplemente Ingeniería Dirigida por Modelos (MDE).

Esta nueva forma de entender el desarrollo de software, al igual que los lenguajes de programación, también tiene como objetivo principal, elevar el nivel de abstracción y automatización en los procesos, por lo que a través de modelos de alto nivel, que se expresan en algún lenguaje de modelado o lenguaje específico de un dominio (Domain-Specific Language, DSL), se puede generar cualquier código que requiera la aplicación final.

Con el Lenguaje Unificado de Modelado UML(Unified Modeling Language, por sus siglas en inglés), es donde realmente se le empezó a dar una verdadera importancia al modelado, sin embargo, no se explotaba su verdadero potencial de los modelos porque sólo eran utilizados para documentar o por los analistas pero nunca por los programadores, en este contexto UML retoma fuerza y pasa a formar el lenguaje para MDE, donde los modelos pasan a ser un artefacto del desarrollo que permiten generar el código de la aplicación final. Recientemente, MDE está ganando cada vez más aceptación, principalmente debido a su capacidad para abordar la complejidad y mejorar la productividad del software. Las técnicas MDE, en especial el metamodelado y las transformaciones de modelos, han demostrado ser útiles para varias tareas de ingeniería directa e inversa.

En la figura 1, se muestra una representación de base de datos NoSQL, donde se identifican las entidades y sus relaciones entre ellas.

[1] El Dr. Severino Feliciano Morales es Profesor de Programación y Desarrollo de Software en la Universidad Autónoma de Guerrero, México y es Dr. en Informática por la Universidad de Murcia, España. sevefelici@hotmail.com (autor corresponsal).
[2] El M. en C. Edgardo Solís Carmona es Profesor de Base de Datos en la Universidad Autónoma de Guerrero, México. esoliscr@hotmail.com.
[3] El Dr. José Luis Hernández Hernández es Profesor de Inteligencia Artificial en la Universidad Autónoma de Guerrero, México y es Dr. en Informática por la Universidad de Murcia, España. tec_jlhh05@yahoo.com.mx.
[4] El Dr. Mario Hernández Hernández es Profesor de Arquitecturas Heterogéneas en la Universidad Autónoma de México, Guerrero y es Dr. en Informática por la Universidad de Murcia, España. mario.hernandez4@um.es.
[5] El M. en C. Valentín Alvarez Hilario es Profesor de Base de Datos en la Universidad Autónoma de Guerrero, México. valentin_ah@yahoo.com.

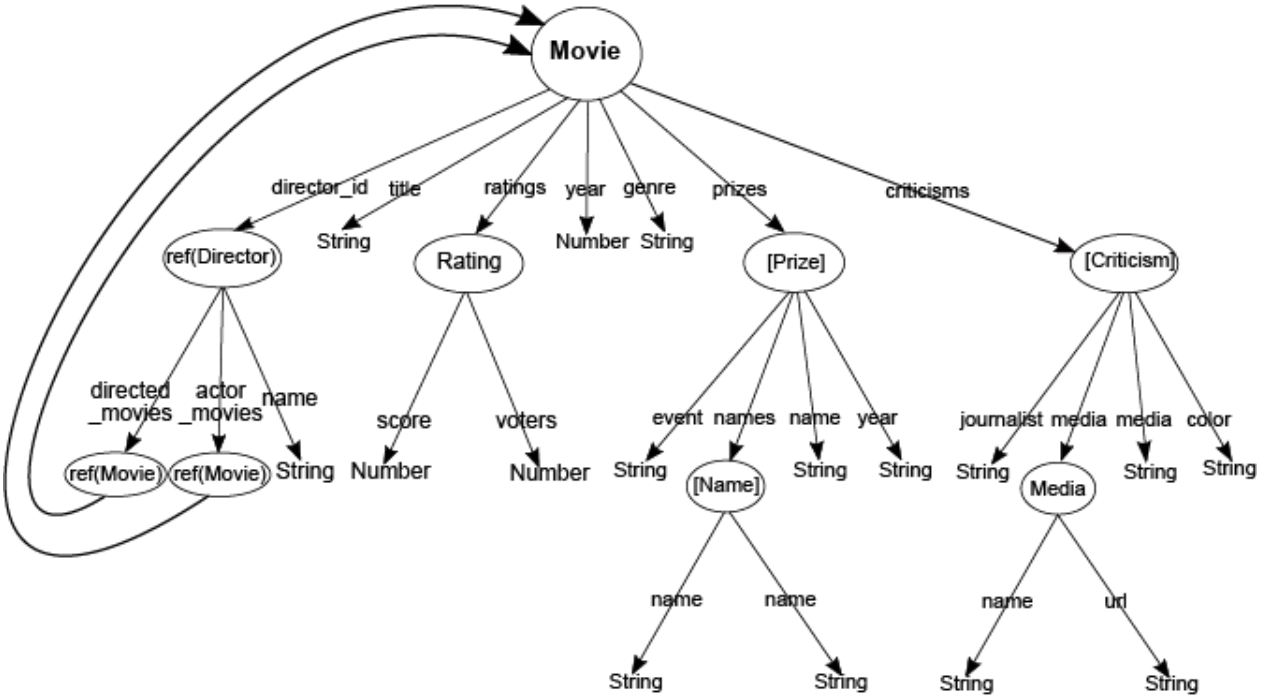

Figura 1. Modelo de una base de datos NoSQL.

Modelos

Los modelos juegan un rol fundamental, ya que permiten que este paradigma pueda elevar el nivel de abstracción y de automatización para atacar el principal problema en la creación de software: el dominio de la complejidad, además de permitir mejorar diferentes aspectos de la calidad del software como la productividad y el mantenimiento.

Un modelo es una extracción simplificada de la realidad, como resultado de un proceso de abstracción, el cual ayuda a comprender y razonar sobre su entorno que lo rodea. Los modelos son expresados mediante alguna notación que depende de su propósito y a quién van dirigidos. Los modelos de software permiten especificar aspectos importantes tales como los requisitos, la estructura y el comportamiento de un sistema.

Un modelo generalmente debe estar expresado formalmente y lo más práctico e idóneo es por medio de un metamodelo.

Las herramientas para expresar modelos formalmente que se ha usado últimamente, se denominan lenguajes específicos de dominio (DSL). Un DSL consta de tres elementos principales: la sintaxis abstracta que define los conceptos del lenguaje (metamodelo), las relaciones entre ellos, así como las reglas que establecen cuando un modelo está bien formado; la sintaxis concreta que establece la notación exacta que debe cumplir cualquier modelo del dominio del DSL y la semántica que normalmente es definida a través de la traducción a conceptos de otro lenguaje destino (por ejemplo, un lenguaje de programación como Java o uno específico).

La sintaxis abstracta de un DSL se define mediante un metamodelo, junto con un conjunto de reglas que definen restricciones adicionales para que un modelo se considere bien formado. Estas reglas se suelen expresan con el lenguaje OCL(Object Constraint Language, por sus siglas en inglés) o vienen implícitas dentro del lenguaje como es el caso de Java a través de la API EMF(Eclipse Modeling Framework, por sus siglas en inglés), Xtend o RubyTL que puede usar el potencial de Ruby, ya que puede embeberse dentro de él.

Para definir un metamodelo, se usa el metamodelo Ecore de EMF, el cual es el elemento central del Eclipse Modeling Framework, que proporciona la infraestructura básica del proyecto de Eclipse para crear herramientas y soluciones MDE. Ecore proporciona herramientas para la definición de un DSL, que incluye los conceptos propios del modelado orientado a objetos que le permite definir un lenguaje: clases para expresar los conceptos, atributos para expresar las propiedades, agregación y referencias.

La figura 2, muestra un metamodelo para representar esquemas de bases de datos NoSQL, donde la figura 1 por ejemplo, puede ser una instancia de este metamodelo, en el cual se puede apreciar que un esquema puede contener

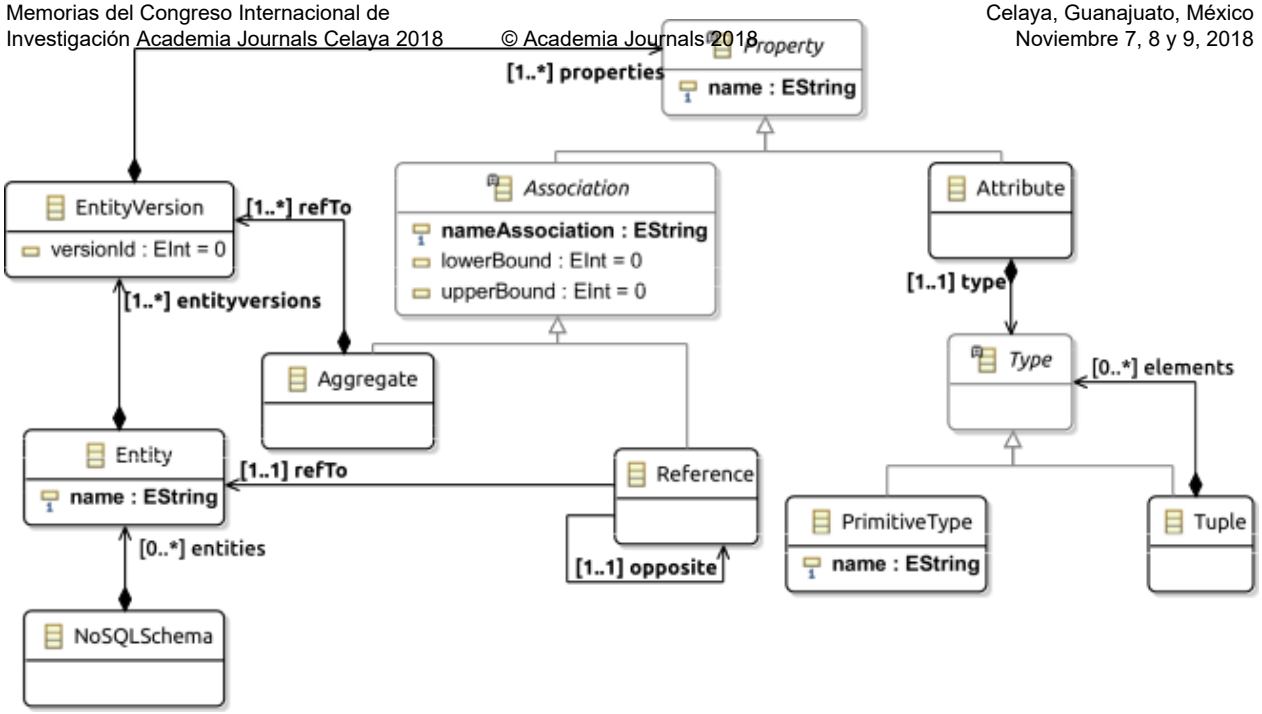

Figura 2. Metamodelo de esquema de una base de datos NoSQL.

cero o más entidades; cada entidad tiene una o más versiones; una versión contiene cero o más propiedades; una propiedad puede ser una asociación o un atributo; una asociación a su vez, puede ser un agregado o una referencia, la cuales contemplan un proceso recursivo para continuar el recorrido por todo el modelo de la base de datos; en caso de que sea un agregado, tiene una o más versiones, por lo que también puede ser una entidad y en caso de que sea una referencia, esta se asocia con una entidad para identificar la relación. Por otro lado, en caso de que sea un atributo, puede ser de tipo primitivo o una tupla y en caso de que sea una tupla, inicia un proceso recursivo para verificar todo el contenido de la tupla.

La notación o sintaxis concreta puede ser textual o gráfica y en el marco de referencia de MDE, han surgido muchas herramientas basadas en el metamodelado que permiten crear tanto DSLs textuales, tales como EMFText y Xtext, como gráficos tales como MetaEdit+ y DSL Tools. Lo idóneo es disponer de herramientas que permitan crear DSLs con una naturaleza híbrida que combine texto y gráfico, aunque hay ocasiones en las cuales no es necesario asociar una notación a un metamodelo, dado que los modelos serán generados como parte de una etapa intermedia hacia la obtención de artefactos de software finales, como parte de una transformación de modelos. Con un DSL es posible expresar soluciones de software a un nivel de abstracción más alto que con lenguajes comúnmente utilizados, tales como los lenguajes de programación de propósito general, XML, HTML, etc. Aunque los modelos llevan ya mucho tiempo siendo usados como parte de la documentación o de razonar sobre el código, en MDE se han convertido en un elemento primordial en el desarrollo de software, de modo que son parte de la solución, de la misma forma que pudiera ser el código Java y configuración XML.

La abstracción que se logra extraer de la los problemas reales, representada en el modelo, puede ser cualquier tipo de sistema, tales como un sistema humano, un sistema mecánico, o un sistema mixto con elementos humanos, mecánicos y de software. Un modelo puede representarse gráficamente mediante diagramas, pero también puede representarse gráficamente en una estructura de árbol (como habitualmente ocurre en el típico panel izquierdo de algunos entornos integrados), o incluso en forma puramente textual como es el caso de XMI o JSON, es decir, el modelo es independiente de su representación y generalmente se expresa en un determinado lenguaje de modelado. Un lenguaje de modelado gráfico es un lenguaje que tiene cierta heurística para hacer inferencias inteligentes que sirven para expresar un modelo con símbolos gráficos, en forma de diagramas. El modelo debe decir implícitamente o explícitamente, lo más importante del problema que se está estudiando y por supuesto conforme a las reglas del lenguaje de modelado. Esto significa que una definición rigurosa del lenguaje de modelado es esencial para poder obtener el mejor resultado posible de los modelos en el proceso de ingeniería.

Análogamente, se puede decir que un metamodelo es la clase, es decir, la definición de todos los posibles modelos que se pueden crear en un determinado lenguaje de modelado. En correspondencia, un modelo es una instancia de su sintaxis abstracta, de su metamodelo.

Finalmente para que el proceso sea completado, se necesita la sintaxis concreta que está estrechamente relacionada

con la sintaxis abstracta definida por el metamodelo, entonces esta sintaxis está basada en EBNF, donde se determina básicamente la estructura de los modelos de entrada, que a su vez existen herramientas que generan un editor, una vez que se tiene definido el metamodelo y su sintaxis concreta, por lo que se pueden capturar los modelos que son instancias del DSL.

Transformación de modelos

La transformación de modelos es un proceso vital para el éxito de los diversos enfoques de desarrollo de software dirigido por modelos. Se han definido varios lenguajes y herramientas de transformación de modelos como resultado de un intenso trabajo de investigación y desarrollo tanto desde la industria como desde el mundo académico. El interés actual está centrado principalmente en la experimentación con los lenguajes existentes a través de la escritura de definiciones de transformaciones para casos reales. En este sentido existen lenguajes muy potentes y de actualidad, algunos específicos como rubyTL, ATL, entre otros o de propósito general tales como Java o Xtend, que es un lenguaje que tiene un generador de código que genera automáticamente su equivalente en lenguaje Java para ejecutarse en su máquina virtual.

Transformaciones Modelo a Modelo (m2m)

Las transformaciones modelo a modelo, son uno de los principales mecanismos de explotación de las arquitecturas MDE. La transformación m2m consiste en obtener a partir de uno o varios modelos, un modelo más refinado, es decir, cuando en el modelo actual aún no se contemplan la suficientes propiedades para abordar el problema en cuestión.

Por ejemplo en la figura 3, se muestra un metamodelo al cual se le denominó *EntityDifferentiation*, el cual se usó para obtener no sólo el esquema de una base de datos NoSQL, sino lograr conseguir todas las versiones de cada Entidad, además con eso también se pudo abordar el problema de esquemas para Object Document Mappers (ODM), ya que este metamodelo logra extraer las propiedades comunes de cada entidad y las propiedades particulares de cada versión particular. Esto se logra a partir de las propiedades del metamodelo de esquema de la figura 2, implementando una *transformación de modelos*, es decir, se obtiene el metamodelo de diferenciación de la figura 3, a partir del metamodelo de esquema de la figura 2, utilizando precisamente técnicas MDE.

Transformaciones Modelo a Texto (m2t)

Existe otro mecanismo dentro de MDE, que es la transformación modelo a texto, en el cual las herramientas toman un modelo como entrada y devuelven una cadena de texto como salida. Los modelos de salida generados tienen un nivel de abstracción muy bajo, pues se trata del código final que necesita el sistema, que puede ser cualquier código como por ejemplo código Java. Existen varios lenguajes para ello, de propósito general como Xtend o específicos como MOFScript, entre otros.

Para el ejemplo de los ODM, se definió una cadena de transformación de modelo de dos pasos que tiene como entrada un modelo de esquema de una base de datos inferido y genera el código Javascript de los artefactos *Mongoose* para el sistema *mongoDB*. Los artefactos generados son principalmente el esquema de la base de datos.

El primer paso de la cadena es una transformación M2M, como se describió anteriormente. Esta transformación genera un modelo que se ajusta al metamodelo EntityDifferentiation.

El segundo paso es una transformación M2T que genera código de Mongoose a partir del modelo obtenido en el paso anterior. El metamodelo EntityDifferentiation se definió para facilitar la escritura de la transformación M2T.

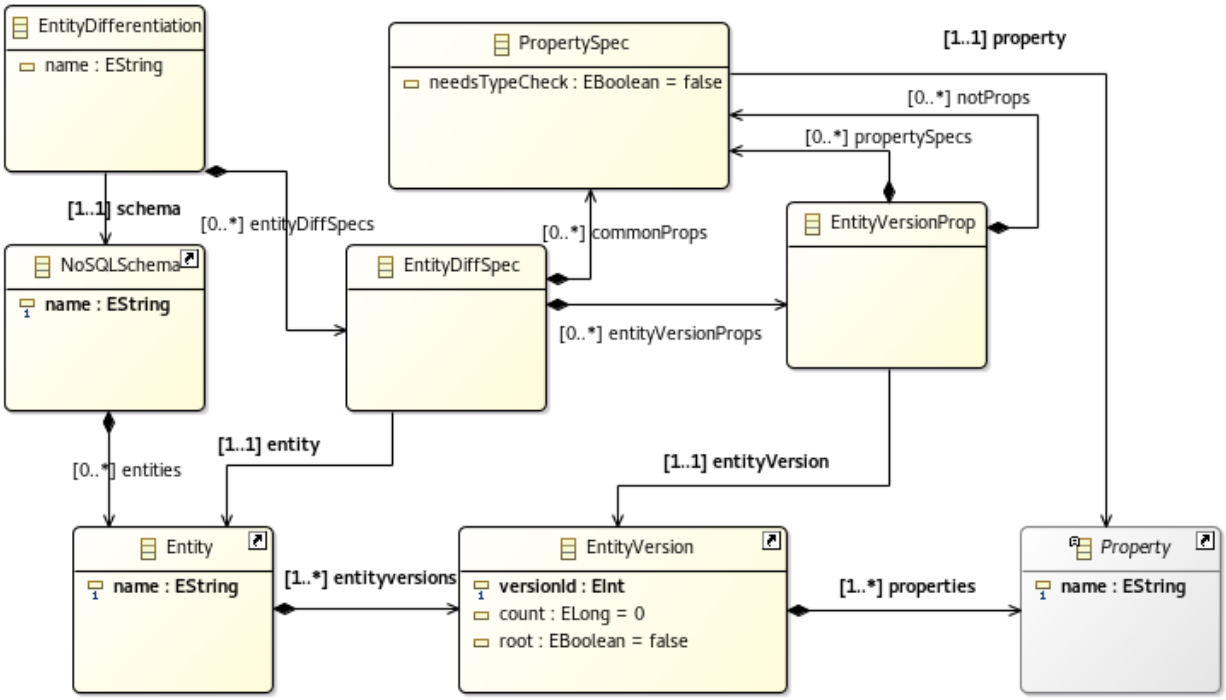

Figura 3. Metamodelo EntityDifferentiation.

Caso Práctico

Para mostrar las bondades de MDE y en particular de la transformación de modelos, se ha implementado un caso práctico en el que si se hiciera tradicionalmente sería muy complejo y con muchas líneas de código. Lo que se ha hecho es una transformación de modelo a modelo para convertir una matriz a un grafo. Como se sabe un grafo nos ayuda a modelar muchas cosas del mundo real, sin embargo para su implementación, toda la información generalmente se almacena en matrices de adyacencia, incluso para algoritmos de búsqueda para inteligencia artificial, también se usan este tipo de matrices. Sin embargo, utilizando técnicas MDE, se ha transformado una matriz a grafo y la implementación se hace directamente sobre el grafo, ya que no se le da tratamiento como grafo, sino como modelo que representa un grafo y se puede recorrer como se requiera.

En la figura 4 se muestran los metamodelos de una matriz y de un grafo, con los cuales se aplica la transformación de modelos. Para ello se escribió una transformación en RubyTL, que es un lenguaje de transformación basado en reglas que ha sido construido como un lenguaje embebido dentro de Ruby. Es un lenguaje de transformación híbrido, cuya parte declarativa está basada al estilo de ATL (Atlas Transformation Language), mientras que la parte imperativa viene dada por las construcciones proporcionadas por el propio lenguaje Ruby. Este proyecto fue resultado de una tesis doctoral del grupo de investigación Modelum de la Universidad de Murcia, España.

También se hizo la transformación de modelo a texto, generando el código java que permite hace una representación gráfica simple, correspondiente al grafo resultante. Este proceso de generación de código, es lo que en MDE, se denomina transformación de modelo a texto y se implementó en el lenguaje Xtend, el cual permite generar texto a partir de un modelo, ya que permite el manejo de plantillas. En la figura 5, se muestra el ejemplo del grafo resultante, dibujado con el código java generado.

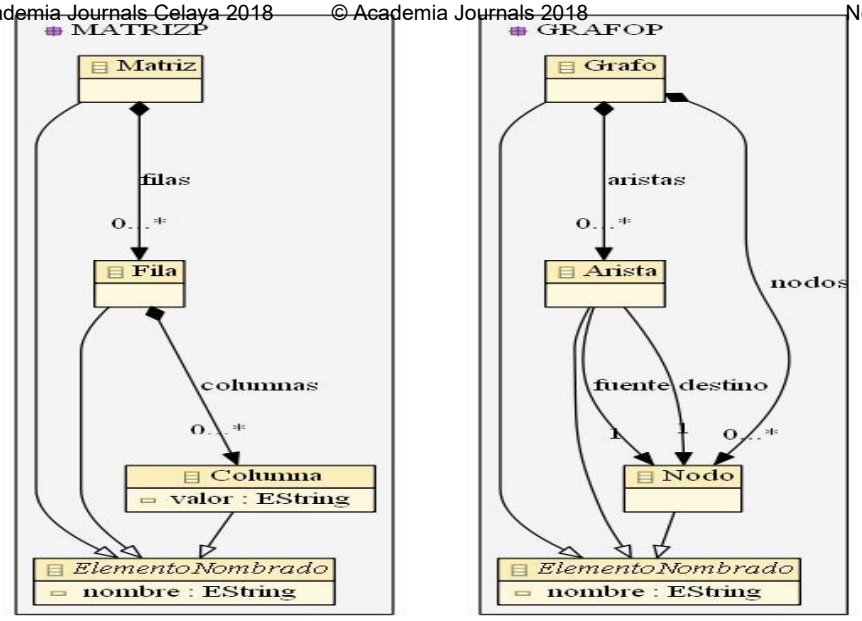

Figura 4. Metamodelos para Matriz y Grafo.

Conclusiones

Al concluir este trabajo, se puede argumentar que con el paradigma MDE, el desarrollo de software mejora la productividad para producir software, ya que no sólo elevan el nivel de abstracción y de automatización, sino que te permite abordar problemas muy complejos de una forma más práctica, rápida y sencilla porque existen herramientas que permiten implementar todo a base de los modelos y su objetivo final es que el código se genere automáticamente. Los artefactos principales para representar los problemas de la realidad, son los metamodelos y modelos en lugar de escribir los miles o millones líneas de código, además estos modelos también pueden ser compilados y ejecutados, esto significa que también se consigue construir un producto terminado, incluso un modelo podemos transformarlo a otro más refinado, si se requiere.

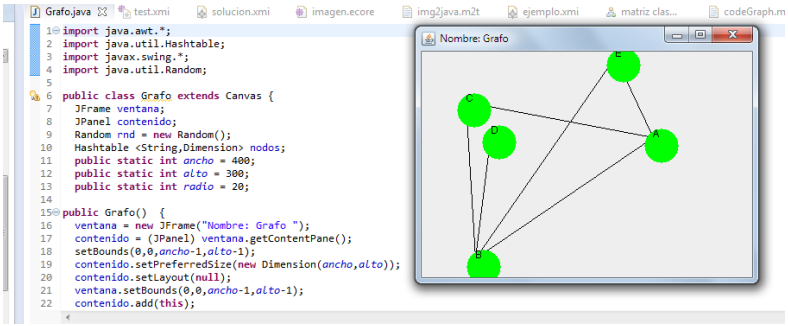

Figura 5. Ejecución del código generado.

Referencias

Feliciano Morales, Severino, et. al. (2017). Inferring NoSQL data schemas with model-driven engineering techniques, Tesis de Doctorado. Universidad de Murcia. Consultada por Internet el 10 de septiembre del 2018. Dirección de internet: http://hdl.handle.net/10201/53472.

García Molina, Jesús et al. (Septiembre 2012). Desarrollo de Software Dirigido por Modelos: Conceptos, Métodos y Herramientas, RA-MA.

Selic, B. (2012). What Will it Take? A View on Adoption of Model-Based Methods in Practice. Software and Systems Modeling, 11(4), 513–526.

Whittle, J., Hutchinson, J., & Rouncefield, M. (2014). The state of practice in model driven engineering. IEEE Software, 31(3), 79–85.

Brambilla, M., Cabot, J., & Wimmer, M. (2012). Model-Driven Software Engineering in Practice. Morgan & Claypool Publishers.

Cánovas Izquierdo, J., Jouault, F y Cabot J y García Molina, J., (2011). API2MoL: Automating the building of bridges between APIs and Model-Driven Engineering. Information and Software Technology 54 (2012) 257–273.

Sánchez Cuadrado J., García-Molina,J.y Menárguez, M., RUBYTL: Un Lenguaje de Transformación de Modelos Extensible, en JISBD 2006, Barcelona, Consultado por Internet el 29 de septiembre del 2018. Dirección de internet http://ceur-ws.org/Vol-227/paper13.pdf.