Scientia Et Technica
ISSN: 0122-1701
scientia@utp.edu.co
Universidad Tecnológica de Pereira
Colombia

Gaitán Peña, Carlos Alberto
Líneas de Productos Software: Generando Código a Partir de Modelos y Patrones
Scientia Et Technica, vol. 22, núm. 2, junio, 2017, pp. 175-182
Universidad Tecnológica de Pereira
Pereira, Colombia

Disponible en: http://www.redalyc.org/articulo.oa?id=84953103009

Líneas de Productos Software: Generando Código a Partir de Modelos y Patrones

Software Product in Lines: Code Generation from MDA and Patterns.

Carlos Alberto Gaitán Peña
Universidad Nacional de Educación a Distancia - UNED (España), Colombia
Correo-e: inggaitan@yahoo.es

***Resumen*— La Reutilización contribuye al desarrollo de herramientas de software con un alto potencial de flexibilidad, permitiendo el uso de componentes en diversos desarrollos, extendiendo las capacidades, mejoras y funcionalidades del software. Un nuevo paradigma de desarrollo denominado "Líneas de Producto Software", es una tendencia cada vez más utilizada por diseñadores por cuanto permiten el desarrollo de diversos modelos o sistemas software los cuales comparten entre sí características a partir de un núcleo común altamente Reutilizable. Este Artículo es el resultado de una propuesta de diseño de un prototipo para generación de código automatizado a partir de modelos MDA y la implementación de Patrones de diseño como MVC, presentes en la mayoría de generadores de código de tipo comercial y otras herramientas GNU, las cuales transforman códigos para sistemas Transaccionales CRUD en plataformas como JSP, ASP, PHP, Ruby, etc; para entornos Web. El uso del prototipo se planteó a partir de un Lenguaje común, pero su implementación puede extenderse a otros lenguajes o especificaciones debido a su alto grado de Usabilidad y Fiabilidad.**

***Palabras clave*— LSP, Arquitectura Dirigida por Modelos, Patrones de Diseño, Generadores, Usabilidad, Bases de Datos Transaccionales.**

***Abstract*— The reutilization allows the flexible software development that can be used by other developments, extending and improving the functionalities. A new paradigm of development called " Software Product in Lines ", a set of systems of software that share characteristics from a common reusable core. This paper proposes the design of the prototype for code generation from the MDA and the implementation of MVC pattern supported by several code generation commercial and GNU tools, that allow the generation of CRUD interfaces for multiple platforms (JSP, ASP, PHP, Ruby, etc.), in WWW environments. The prototype will adjust to a language specification, but it can be extended to other languages specifications due to this high degree of usability and reliability.**

***Key Word* — LSP, Model Driven Architecture, Software Design Pattern, Code Generation, Usability, CRUD.**

I. INTRODUCCIÓN

Una de las finalidades en todo proceso de ingeniera de software consiste en el desarrollo de artefactos de calidad minimizando costos y mejorando la eficiencia en los procesos a través de objetos altamente flexibles y reutilizables. La reutilización, es una característica promovida por la ingeniería de software que permite el desarrollo de productos de software lo suficientemente flexibles y que pueden ser utilizados con posterioridad en otros desarrollos, ampliando y mejorando sus funcionalidades. En base a esta estrategia, surgió un nuevo paradigma en el desarrollo de software, llamado "Líneas de Productos de Software". De acuerdo al SEI [17], una línea de productos de software se refiere a un conjunto de sistemas de software (producto) que comparten características y que son desarrollados a partir de un conjunto común de bienes núcleo (partes reutilizables).

El presente documento recopila algunos aspectos y funcionalidades de un prototipo generador de código, haciendo uso de patrones de diseño como MVC y modelos de transformación, para generar aplicaciones ágiles en ambientes transaccionales. Este proceso es empleado por diversas herramientas tanto comerciales y de código abierto como Ruby on Rails, desde las cuales se generan interfaces simples tipo CRUD para múltiples plataformas (JSP, ASP, PHP, Ruby, etc.), en entornos web. La implementación del prototipo se basa en especificaciones para el lenguaje PHP por ser un lenguaje flexible, interpretado y con alto grado de usabilidad. Los conceptos y metodología estarán ajustados al proceso MDA y la utilización de modelos para la generación de herramientas software.

II. LÍNEAS DE PRODUCTO SOFTWARE

El Instituto de Ingeniería de Software de la Universidad Carnegie Mellon [1], define a una SPL como: "*…un conjunto de sistemas de software que comparten un conjunto común y gestionado de características que satisfacen las necesidades específicas de un segmento de mercado particular o misión, y que son desarrolladas de forma prescrita a partir de un conjunto común de elementos clave…*". En este contexto, una SPL permite acotar el tiempo de producción de componentes

Fecha de Recepción: 24 de marzo de 2015
Fecha de Aceptación: 03 febrero de 2017

software, con altos niveles de calidad y una alta capacidad de reutilización.

Como enfoque de desarrollo, la SPL utiliza la reutilización como un componente fundamental para el desarrollo de herramientas software flexibles. La reutilización valida los objetos y sus características, realiza abstracciones y a partir de estas validaciones genera nuevos modelos o derivaciones. A partir de este concepto, una SPL puede concebirse a partir de un enfoque guiado por modelos MDD a través de una serie de transformaciones [2].

A. MDD (*Model-Driven Development*)

MDD es una aproximación para solucionar el problema asociado a la complejidad de cada plataforma tecnológica y la inhabilidad que experimentan los lenguajes de propósito general en aliviar esta complejidad [3].

Un enfoque de Desarrollo Dirigido por Modelos (***Model-Driven Development***), establece que para el desarrollo de un componente software, solo habrá de diseñarse un modelo específico que corresponda a la abstracción de la realidad que se desea representar y a partir de este modelo se generarán los componentes, especificaciones y código para diversas plataformas separando los detalles arquitectónicos y de implementación, permitiendo la generación de uno o varios modelos a partir de un mismo PIM (***Platform Independent Model***). Con la implementación de este enfoque se adiciona mayor flexibilidad, interoperabilidad, flexibilidad y reutilización de componentes en una línea de producto software, minimizando el tiempo de desarrollo y ahorrando de manera sustancial la producción de líneas de código. La implementación más conocida de MDD se denomina Model Driven Architecture (MDA).

B. La Usabilidad como factor de desarrollo en una Línea de Producto Software (SPL)

De la Norma ISO:25000 [4], antigua norma ISO/IEC 9126, la ***usabilidad*** se define como "*la capacidad que tiene un software para ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso*". En el proceso de ingeniería para desarrollo de líneas de producto software, la usabilidad se convierte en un factor fundamental que flexibiliza la factorización de componentes software gracias a su capacidad para mantener componentes comunes entre sistemas a través del uso eficiente de modelos y sus transformaciones.

La variabilidad y combinación de modelos y la adaptabilidad de este concepto a cualquier arquitectura, supone que la usabilidad permite la reutilización de bases comunes tanto en la capa de ingeniería de dominio como en la ingeniería de la aplicación. (Fig. 1). Una SPL comprende esencialmente 2 etapas:

- ***La ingeniería de dominio*** es la encargada de diseñar los componentes de software que de acuerdo a su trazabilidad, hacen parte del dominio común y que son sujetos a reutilización. En esta etapa se definen las similitudes y variantes del modelo de dominio en cada una de las fases. Como producto final en esta etapa se considera un componente altamente reusable y flexible que permita a partir de su definición, la consecución de nuevos modelos y artefactos.

- ***La Ingeniería de Aplicación*** se encarga de articular los artefactos definidos en la etapa de dominio a partir de una plataforma específica. El objetivo de esta etapa consiste en lograr la integración de artefactos bajo uno o varios sistemas, logrando con ello un alto grado de reusabilidad de sus componentes a partir de las especificaciones, la definición y desarrollo, documentación e interoperabilidad entre componentes.

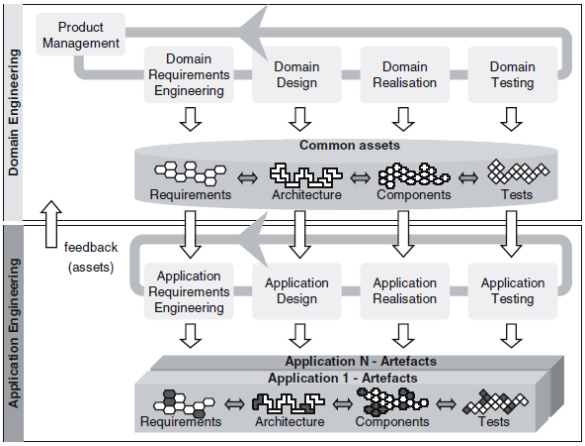

Figura 1. La SPLE consta de dos etapas: Ingeniería de Dominio e Ingeniería de Aplicación. Ambas etapas siguen el esquema de un ciclo tradicional para diseño y desarrollo de software involucrando en sus procesos las fases de: análisis de requerimientos, diseño arquitectónico, implementación y pruebas. Imagen tomada del libro Software Product Lines in Action [5].

C. MDA "*Model Driven Architecture*"

La especificación (MDA) "Model Driven Architecture" es una especialización del desarrollo dirigido por modelos que separa la lógica del negocio del software y las plataformas tecnológicas [6]. ***Model Driven Architecture*** (MDA) según [7] y [8] es una aproximación definida por el *Object Management*

Group (OMG), mediante la cual el diseño de los sistemas se orienta a modelos.

A menudo el término MDA se asocia con MDD (*Model-Driven Development*) y MDE (*Model-Driven Engineering*). MDA en su contexto se refiere a un marco de trabajo promovido por el grupo OMG bajo la cual se desarrolla el paradigma de desarrollo orientado por modelos.[1] MDD permite una alta flexibilidad en la implementación, integración, mantenimiento, prueba y simulación de los sistemas software. En este sentido, MDA se convierte en una solución para el desarrollo de soluciones escalables y flexibles ya que separa las especificaciones del sistema de los detalles de su implementación en una determinada plataforma [9]. El objetivo final de MDA es ofrecer mayor portabilidad, interoperabilidad y reusabilidad en los componentes de software.

MDA define tres tipos de modelos. Los CIM, Computation Independent Model, asociados al dominio del negocio, los PIM, Platform Independent Model, asociados a modelos abstractos del software, y los PSM, Platform Specific Model, relacionados con modelos de software específicos de plataformas tecnológicas [6].

OMG ha definido un conjunto de estándares para el desarrollo de modelos a través de MDA de los cuales se destacan: UML (*Unified Model Language*) [10], MOF (*Meta Object Facility*) [11], QVT (*Query View Transformation*) [12], OCL (*Object Constraint Language*) [13] o XMI (*XML Metadata Interchange*) [14].

La arquitectura dirigida por modelos (MDA) corresponde a una aproximación para el desarrollo de software definido por la OMG [8], donde el desarrollo por modelos y la independencia de herramientas y lenguajes formales, se convierte en una alternativa eficaz para el desarrollo de productos de software.

Según [15], el ciclo de vida de un desarrollo MDA, se fundamenta en la implementación del modelo unificado de desarrollo UP [16]. En este sentido, cada incremento en MDA responde en parte a un proyecto o parte de éste, en el cual se ejecutan las fases de requisitos, análisis, diseño implementación y pruebas, dentro de las cuales se realizan una serie de iteraciones que permitan revisar cada artefacto, hasta que cumpla las condiciones de cada incremento para poder continuar con la siguiente iteración hasta finalizar el producto. Cada incremento corresponde a las etapas de iniciación, elaboración, construcción y transición de la metodología de desarrollo UP.

Lo novedoso en este tipo de arquitectura la cual difiere en la forma en cómo se concibe un artefacto de software, radica en la capacidad que tiene MDA para realizar transformaciones mediante la generación de código automatizado a través de modelos formales. En concreto, se puede afirmar que el aporte de MDA al proceso de la ingeniería de software corresponde a la transformación de modelos PIM en modelos PSM.

El proceso de transformación es el punto clave en la orientación por modelos en MDA. Si bien es cierto que la concepción de este mecanismo es bastante complejo, éste se convierte en un elemento fundamental ya que al realizar un solo modelo PIM, se pueden producir a partir de ésta varias sucesiones o códigos aplicables a diferentes tecnologías de programación de una manera eficiente sin perder ningún detalle inicial, ya que todas las especificaciones están inmersas en cada transformación.

Uno de los aspectos claves para el desarrollo de MDA se centra en el uso de estándares abiertos y en la implementación de métodos ágiles que permiten desarrollos con mayor portabilidad e interoperabilidad entre componentes software, así como la reducción de tiempos y costos de producción de aplicaciones. Como ejemplo de ello, se puede citar la especificación UML que ha demostrado ser un estándar eficaz para el desarrollo de aplicaciones desde un modelado de objetos. En el proceso de desarrollo MDA realiza las siguientes transformaciones:

- **Fase I**. Modelo independiente del cómputo (CIM). incluye el vocabulario del negocio y las funcionalidades del sistema (abstracción del sistema). CIM representa los requisitos y conceptos del sistema. El CIM se centra en el negocio, no en el software.

- **Fase II**. Modelo Independiente de la Plataforma (PIM). PIM es una especificación formal en cierto lenguaje de modelado, como UML bajo el cual se diseñan requisitos, estructuras y funcionalidades del software. Una especificación PIM puede tener uno o varios Modelos Específicos de Plataforma (PSM), (traducción en diferentes lenguajes).

- **Fase III**. Modelo Específico de la Plataforma (PSM). Corresponde a la etapa donde se genera el código fuente a partir de un modelo PSM en un lenguaje específico (Java, .NET, C++, Ruby, etc.). La traducción del código se hace de forma directa, ya que PSM contiene el mapeado del código para cada plataforma de desarrollo.

La fase que realmente aporta la independencia con la plataforma sobre la que se realizará el desarrollo, se concentra en la etapa II, ya que en ésta se traduce el modelo genérico a un modelo específico.

[1] Un ***Modelo*** corresponde a una representación abstracta del conocimiento y las actividades que rigen un dominio de aplicación en particular.

D. Arquitectura MDA

Un enfoque dirigido por modelos requiere de un lenguaje específico que pueda ser utilizado por MDA para transformar modelos. En este sentido el lenguaje adoptado por la OMG que permite la transformación y representación de modelos de forma abstracta, además de proporcionar un conjunto de reglas semánticas para la representación de lenguajes formales se denomina MOF (***Meta Object Facility***).

La especificación MOF se basa en diagramas de clase UML para describir la sintaxis abstracta de un modelo o constructores de modelado, en el cual se definen una serie de reglas para construir Meta-Modelos. Un Meta-Modelo se compone de clases, constructores, propiedades, atributos y relaciones existentes entre los diferentes constructores y sus asociaciones.

A partir de la especificación MOF [11], se ha definido una arquitectura de cuatro niveles para el desarrollo guiado por modelos, pero en este documento se ha fraccionado en cinco niveles (Fig. 2), para una mejor comprensión de la arquitectura.

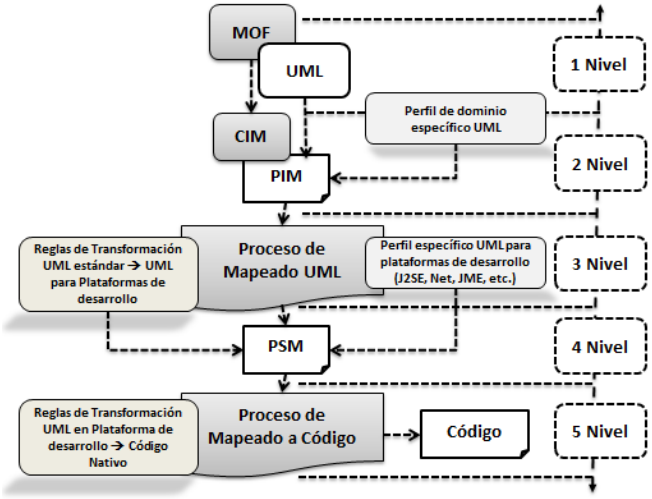

Fig. 2. Con la implementación de la Arquitectura MDA, se pretende dar cobertura a todos los aspectos de diseño y desarrollo de una línea de producto software a lo largo de todo su ciclo de vida, desde su concepción, análisis de requisitos y modelo de negocio, hasta el mantenimiento y generación del código fuente.

- **Nivel 1**: Especificación de Meta-Modelos. En este nivel se definen los Meta-Modelos mediante la especificación MOF. Es el nivel superior de la arquitectura y corresponde a un conjunto de objetos y sus asociaciones. Al ser MOF un lenguaje específico de modelado éste se convierte a sí mismo en un Meta-Modelo. En este nivel UML es un lenguaje apropiado para definir la sintaxis del negocio o modelo de dominio CIM y la lógica de negocio en el PIM.

- **Nivel II**: Todos los objetos (Modelos) producidos en el nivel anterior corresponden a las instancias de MOF. En este nivel se encuentran los elementos que se basan en construcciones UML y las extensiones de los perfiles específicos para cada dominio.

- **Nivel III**: En este nivel se encuentran las instancias del Meta-Modelo. El proceso de mapeado entre PIM y PSM se realiza a través de procesos de transformación y reglas de sintaxis definidas por la especificación MOF y que corresponde a un lenguaje formal para el cual se va a realizar la definición (.Net, Java, C++, PHP, etc.). Cada construcción de PIM a PSM exige de unas reglas de validación y perfiles UML específicos para incorporarlos al diseño del PSM.

- **Nivel IV**: En este nivel se encuentran los objetos y datos que corresponden a las instancias del modelo PIM. Cada PSM es equivalente al PIM que lo generó y contiene los componentes y reglas del lenguaje a transformar y que están descritos por lo general en UML. En este nivel también se encuentran los detalles de la plataforma para la cual se va a realizar el desarrollo, los constructores, asociaciones, extensiones y los perfiles incluidos en el Meta-Modelo.

- **Nivel V**: Mapeado a Código. A partir de este nivel se aplican las reglas de transformación entre el PSM y el código objeto (por ejemplo Java). Este esquema se obtiene a partir de la especificación XMI (XML Metadata Interchange) que es basado en lenguaje XML. XMI, permite representar modelos MOF en ficheros XML, simplificando así la comunicación entre aplicaciones potencializando la reutilización de objetos y componentes. Aparte de XMI, también se utiliza el esquema OCL (*Object Constraint Language*) que restringe el modelo a las reglas semánticas y expresiones que representan *precondiciones*, *postcondiciones*, *inicializaciones*, *reglas de derivación*, así como *consultas* a objetos para determinar sus condiciones de estado. El rol de OCL es el de completar los diferentes artefactos de la notación UML con requerimientos formalmente expresados y los cuales no fueron añadidos por MOF. El proceso de trazabilidad de PSM al código fuente produce en esta etapa lo que se conoce como FCM (Flex-eWare Component Model) que provee un marco común de Meta-Modelos en un contexto orientado a objetos y el cual liga cada elemento PSM a las clases finales de cada lenguaje (por ejemplo Java). FCM se considera como una definición de modelos de librerías o API's las cuales contienen datos referentes a costos y esfuerzos en reglas de producción, ensamblados, técnicas de manufactura y

otros detalles así como la sustentación de procesos en el mecanismo de transformación. El FCM obtenido automáticamente no será la versión final del código objeto, pues necesitará ser refinado y complementado con otros componentes que no fueron generados en el proceso automatizado.

E. GENERACIÓN DE CÓDIGO PARA UNA LSP

En la ingeniería de software y concretamente en el campo de la LSP, la factoría de software está centrada en la reutilización o derivación de ***Activos Base***. Un Activo de software corresponde a un conjunto de artefactos producidos antes y durante el proceso de desarrollo de software. Un Activo Base (***Core Asset***) puede estar compuesto por; arquitecturas, patrones de diseño, componentes, documentación, especificaciones y requisitos del sistema, planes y pruebas entre otras [17].

Las bases de datos se convierten por lo general en la columna vertebral de cualquier desarrollo o sistema de información. Diseñar una adecuada interfaz que permita una interacción eficaz entre usuario y sistema informático depende en gran medida del grado de abstracción y la definición de un modelo que represente de forma concreta el objeto del mundo real. Una de las soluciones que resuelve a la medida este requerimiento en una línea de producto software, corresponde al patrón de arquitectura MVC[2] que permite la separación del modelo de datos, la lógica de negocio y la interacción del usuario con una aplicación informática.

El presente resumen hace referencia al diseño de un prototipo en Ruby on Rails para la generación de operaciones CRUD a partir de la especificación de un modelo y un lenguaje común. El generador automático analiza sintácticamente la sentencia SQL proveída por el usuario. Éste acepta la consulta y el subconjunto de restricciones del lenguaje común, convierte dicha sentencia en estructuras de datos para un PSM específico. En el proceso de transformación, el generador construye la salida a partir de plantillas y especificaciones del lenguaje y procesa dicha información en secuencias binarias que contendrán la lógica y las estructuras tipo CRUD para un sistema transaccional. El generador obtiene la secuencia de caracteres y compara las entradas contra patrones establecidos y "transforma" la secuencia de análisis (Fig.3). El generador en sí mismo no corresponde a una implementación de un sistema de compilación, por cuanto la precisión en los scripts dependerá en gran medida en la introducción de expresiones válidas o sentencias bien formadas para cada script SQL.

La arquitectura del generador está concebida a partir del desarrollo de Activos Base. El proceso involucra la creación de un meta-modelo el cual puede ser aplicado a cualquier dominio[3] o en el caso específico como lo describe la herramienta para la especificación del lenguaje PHP. El proceso de generación incluye las respectivas transformaciones que representan al dominio y las reglas que definidas por el PIM y que serán representadas en el PSM, mapeando los objetos al respectivo lenguaje. Como proceso final de derivación, se aplican las reglas y las plantillas para representar la solución en una plataforma específica, en el caso concreto código nativo para PHP.

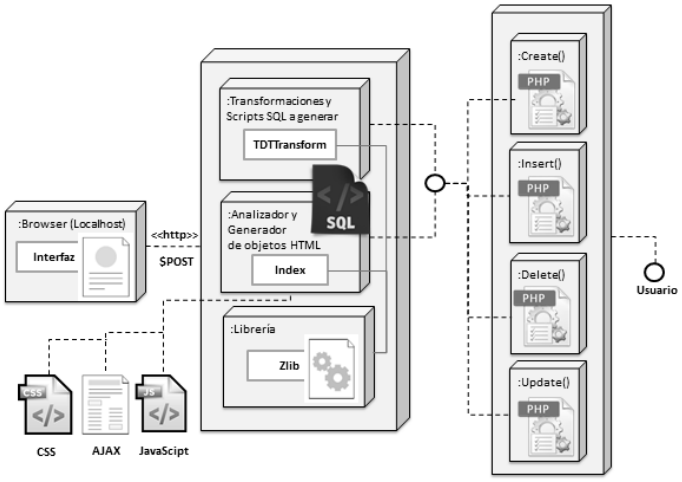

Fig. 3. El generador corresponde a una implementación del patrón MVC. La interfaz se ha definido a partir de un conjunto de objetos tipo web, donde el navegador envía la petición al controlador quien interactúa con el modelo especificado por el lenguaje DDL. Posterior a ello se validan las sentencias y se agregan al contexto la definición de plantillas estáticas para producir la salida. El resultado final contendrá la definición de las operaciones básicas en un sistema transaccional las cuales serán retornadas al usuario a través de la respectiva vista del navegador.

Una de las mayores ventajas en el proceso de generación de interfaces a través de patrones MVC radica en la separación del modelo de datos, la lógica de negocio y la interacción del usuario con la aplicación informática, con la cual se obtiene un alto grado de usabilidad y mantenibilidad del sistema gracias a la separación de componentes. Un generador automático en este sentido debe contemplar:

a) Reglas de validación para la capa de negocio (datos, estructuras y bases de datos).

b) Los valores asociados a los campos de un determinado formulario, deben validarse no solo contra los tipos descritos en la base de datos, sino que también debe validar su funcionalidad implementando mensajes entre la vista y el sistema gestor de base de datos para evitar conflictos entre la capa visual y la lógica.

[2] ***MVC*** : Modelo –Vista – Controlador

[3] Llámese ***dominio*** al ámbito de cualquier lenguaje de programación o sistema de compilación.

c) Un número determinado de validaciones asegura un alto grado de usabilidad de la herramienta en ambas vías tanto del lado del cliente como del lado del servidor.

d) El generador debe contar con funciones que permitan eliminar tags que no serán interpretados por el compilador y procesar aquellos que sean requeridos para armar el esqueleto de los formularios y objetos a generar.

e) El generador debe especificar y validar los parámetros que permitan enlazar los datos (modelo) con la capa gráfica (vista), gestionando la lógica (controles). En este sentido, el generador define parámetros de servidor, motor de gestión, recursos ODBC a utilizar y reglas de consistencia que respaldan la conexión a las fuentes de datos "***Persistencia***".

Estas características se resumen en portabilidad, flexibilidad y consistencia; la razón de ser de todo proceso de Desarrollo Ágil.

III. RESULTADOS OBTENIDOS

En las imágenes 4 y 5 se muestra la línea de producto generada. Es importante considerar que el resultado obtenido no hubiese sido posible sin la reutilización de algunas librerías Open Source y la aplicación de conceptos sobre patrones y modelos.

La técnica de transformación empleada en el desarrollo del prototipo, se asemeja a la utilizada por lenguajes como Ruby on Rails a través del proceso de migraciones. Rails está concebida bajo el esquema MVC, en la cual el programador describe las especificaciones y el sistema reproduce la interfaz para manipular los datos en un sistema relacional. Éste y otros Frameworks de desarrollo permiten ampliar las funcionalidades de las bases de datos CRUD al generar interfaces y códigos con el mínimo esfuerzo de programación. Con la implementación de este artefacto se pretende dar al lector una idea de dicho proceso y la aplicabilidad de conceptos de ingeniería de software aplicados a una LSP.

Fig. 4. El generador valida el flujo de entrada e invoca las rutinas de transformación. Los mecanismos de transformación mapean el código y generan los scripts para la manipulación de los datos CRUD. Esta vista corresponde a las interfaces generadas y los binarios que podrán ser editados por el usuario posteriormente.

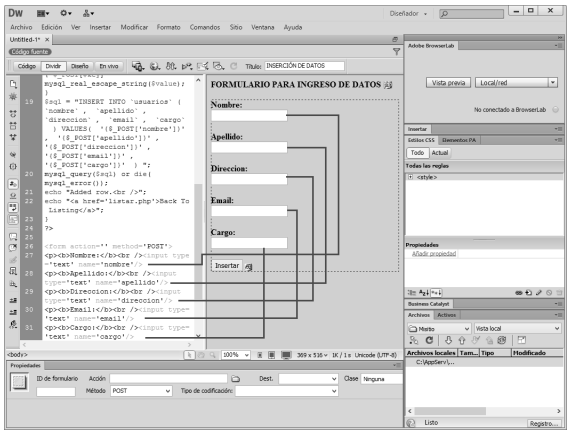

Fig. 5. El producto final corresponde a una interfaz de formulario web desde la cual se puede manipular la información contenida en una base de datos relacional.

IV. CONCLUSIONES

En la ingeniería de software, la LSP corresponde a una implementación eficaz de la Reutilización donde los productos software guiados por artefactos altamente flexibles, hacen artefactos mucho más eficientes y con un alto grado de calidad, donde prima la calidad del producto, de ahí que las líneas de producto se deriven de las Factorías de Software.

Con la implementación de una línea de producto es claro que la reutilización y combinación de paquetes supone una manera más simple y efectiva para gestionar cualquier proyecto de software independientemente de los requerimientos proveídos por los modelos y por los mismos requerimientos del usuario.

La generación automatizada en una línea de producto software se convierte en una herramienta poderosa que facilita la transformación y generación de gran número de artefactos, reduciendo considerablemente el tiempo de desarrollo e incrementando la productividad a partir de desarrollos ágiles.

Aunque el desarrollo del producto (generador PHP desde SQL), propuesto en este documento no incluyó manejo de plantillas especializadas o el uso de objetos embebidos para mejorar las interfaces, es de aclarar que el prototipo permite generar a partir de cualquier esquema de tablas MySQL, las vistas esenciales de un modelo de datos CRUD, dando al lector una idea muy general del proceso de generación que efectúan algunas herramientas como Ruby on Rails, CodeCharge Studio o Hibernate las cuales están basadas en patrones MVC.

La Arquitectura Dirigida por Modelos proporciona una visión más amplia del negocio debido a que su implementación se orienta más hacia la especificidad del dominio y no a la implementación de un modelo en una plataforma específica. Cualquier modelo expresado en función de MDA, puede ser implementado en cualquier lenguaje y plataforma, solo basta con realizar las respectivas transformaciones (división entre modelos PIN y PSM) y la salida dependerá en esencia de las prestaciones que posean las herramientas desde donde se está generando el esquema de transformación.

El proceso de transformación en MDA es un elemento que se encuentra en continuo desarrollo. Para mejorar estos procesos y alcanzar los propósitos ideales de MDA, se requiere de la utilización de lenguajes que interpreten las transformaciones y sus interpretaciones a través de lenguajes comunes como OCL que aunque corresponde a una especificación OMG, no ha sido acogida en su totalidad por la industria del software y a la cual le falta suficiente madurez como para convertirse en un estándar mundial para el desarrollo dirigido por modelos o MDD, y del cual depende una línea de producto software.

REFERENCIAS

[1] SEI, «Software Engineering Institute,» 2012. [En línea]. Available: http://www.sei.cmu.edu/productlines/frame_report/index.html. [Último acceso: 10 Mayo 2013].

[2] H. Arboleda, R. Casallas y J. C. Royer, «Implementing an MDA approach for managing variability in product line construction using the GMF and GME frameworks,» 5th Nordic Workshop on Model Driven Software Engineering, pp. 67-82, 2007.

[3] J. Miller y J. Mukerji, «MDA Guide Version 1.0.1. Object Management Group, Boston - MA, 2003.

[4] ISO 25000, «Software Engineering -- Software product Quality Requirements and Evaluation (SQuaRE),» International Organization for Standarization, Ginebra - Suiza, 2005.

[5] F. Van Der Linden, K. Schmid y E. Rommes, Software Product Lines in Action, P. I. Integra Software Services Pvt. Ltd., Ed., New York, USA: Springer, 2007, p. 340.

[6] B. C. Pelayo García-Bustelo, «TALISMAN: Desarrollo ágil de Software con Arquitecturas Dirigidas por Modelos,» Oviedo - España, 2007.

[7] OMG:MDA, «MDA Guide Version 1.0.1,» Boston, MA, 2003.

[8] S. J. Mellor, S. Kendall, A. Uhl y D. Weise, MDA Distilled – Principles of Model-Driven Architecture, Addison-Wesley, 2004.

[9] T. Gardner y L. Yusuf, «Explore model-driven development (MDD) and related approaches: A closer look at model-driven development and other industry initiatives,» 14 Marzo 2006. [En línea]. Available: http://www.ibm.com/developerworks/library/ar-mdd3/. [Último acceso: 3 Abril 2013].

[10] OMG: UML, «Unified Modeling Language Version 2.2,» Object Management Group, Inc., Boston - MA, 2009.

[11] OMG:MOF, «Meta Object Facility (MOF) Core Specification,» 2011. [En línea]. Available: http://www.omg.org/spec/MOF/. [Último acceso: 3 Abril 2013].

[12] OMG:QVT, «Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT),» Enero 2011. [En línea]. Available: http://www.omg.org/spec/QVT/. [Último acceso: 2 Abril 2013].

[13] OMG:OCL, «Object Constraint Language (OCL),» Enero 2012. [En línea]. Available: http://www.omg.org/spec/OCL/. [Último acceso: 6 Abril 2013].

[14] OMG:XMI, «OMG MOF 2 XMI Mapping Specification. V. 2.4.1,» Agosto 2011. [En línea]. Available: http://www.omg.org/spec/XMI/2.4.1/. [Último acceso: 1 Abril 2013].

[15] A. Kleppe, J. Warmer y W. Bast, MDA Explained – The Model Driven Architecture: Practice And Promise, Addison-Wesley, 2005.

[16] I. Jacobson, G. Booch y J. Rambaugh, El proceso unificado de desarrollo de Software, Madrid - España: Addison Wesley, 2001.

[17] P. Clements y L. Northrop, Software product lines: practices and patterns, Boston - MA.: Addison-Wesley Longman Publishing Co., 2001.

OTRAS REFERENCIAS

[18] AETIC©. (2009). España y las nuevas oportunidades: Las Factorías de Software. Madrid: Asociación de Empresas de Electrónica, Tecnologías de la Información y Telecomunicaciones de España.

[19] APA. (2009). Publication manual of the American Psychological Association (6th. ed.). Washington, DC: APA - American Psychological Association.

[20] Arboleda, H., Casallas, R., & Royer, J. C. (2007). Implementing an MDA approach for managing variability in product line construction using the GMF and GME frameworks. 5th Nordic Workshop on Model Driven Software Engineering, 67-82.

[21] García Molina, J. (2012). Desarrollo de Software Dirigido por Modelos. Murcia - España: Departamento de Informática y Sistemas, Facultad de Informática. Universidad de Murcia.

[22] García, R. (2006). Estudio de herramientas de desarrollo de software basado en modelos: MDA y Factorías de Software. UPCT, Escuela Superior de Ingeniería de Telecomunicación. Murcia - España: Universidad Politécnica de Cartagena.

[23] Gil, R. H. (2007). Metodología de Desarrollo de Software Basada en el Paradigma Generativo. Realización Mediante la Transformación de Ejemplares. Universidad Nacional De Educación A Distancia, Departamento de Ingeniería de Software y Sistemas Informáticos. Madrid - España: Escuela Técnica Superior de Ingeniería Informática.

[24] Gomma, H. (2004). Designing Software Product Lines with UML : From Use Cases to Pattern-Based Software Architectures. New Jersey: Addison-Wesley Professional.

[25] Gonzáles, C., & Henderson, B. (2008). Metamodelling for Software Engineering. New York: Wiley.

[26] Greenfield, J., Short, K., Cook, S., & Kent, S. (2004). Software Factories. New York: Wiley Publising Inc.

[27] Herrington, J. (2003). Code Generation in Action. Greenwich, CT: Manning Publications Co.

[28] Kakola, T., & Dueñas, J. C. (2006). Software Product Lines: Research Issues in Engineering and Management. Springer.

[29] Kleppe, A., Warmer, J., & Bast, W. (2005). MDA Explained – The Model Driven Architecture: Practice And Promise. Addison-Wesley.

[30] Letelier, P., & Panadés, M. C. (2006). Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). (D. d. (DSIC), Ed.) Técnica Administrativa, Vol. 05(No. 26).

[31] Mens, T., & Van Gorp, P. (2006). A taxonomy of model transformation. Electronic Notes in Theoretical Computer Science. Association for Computing Machinery. New York: ACM - Digital Library.

[32] Pressman, R. (2002). Ingeniería del Software. Un enfoquee práctico. Madrid, España: McGraw-Hill Interamericana.

[33] Ruby, S., Thomas, D. & Heinemeier H. D. (2010). Agile Web Development with Rails (4th. ed.), The Pragmatic Bookshelf. ISBN-10: 1-934356-54-9. Raleight N.C - USA: Pragmatic Programmers LLC.

[34] St. Lauren, S., Dumbill, E., & Gruber, E. J. (2012). Learning Rails 3: Rails From the Outside In. California - USA: O'Reilly Media, Inc.