**CONSTRUCCIÓN DE UNA APLICACIÓN WEB PARA LA ELABORACIÓN DE DIAGRAMAS DE CLASE EN UML 2, USANDO TECNOLOGIAS RICH INTERNET APPLICATION**

**Gabriel Leonardo Díaz Cárdenas**1, 2, **Marco Antonio Adarme Jaimes**1, 3

1 Programa de Ingeniería de Sistemas, Universidad Francisco de Paula Santander. Avenida Gran Colombia No. 12E-96B Colsag. San José de Cúcuta, Norte de Santander, Colombia.

2 Estudiante, gabrielleonardodc@ufps.edu.co.

3 Magister en Computación, Director del Proyecto, madarme@ufps.edu.co.

**RESUMEN**

El presente artículo expone el proceso de construcción de la herramienta CLASS Modeler; una aplicación web basada en RIA que permite diseñar diagramas de clase UML, así como la generación de código fuente en lenguaje Java. La herramienta se crea para dar apoyo al proceso educativo de los estudiantes del Programa de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander.

CLASS Modeler fue desarrollada usando lenguaje Java a nivel empresarial; el estándar J2EE constituyó el ambiente principal de desarrollo de la herramienta, incorporando tecnologías avanzadas en la construcción de aplicaciones con arquitectura basada en capas. Adicionalmente, se fundamenta en la especificación formal del lenguaje UML en su versión 2.4.1 definida por la *Object Management Group* [1]; la generación de código se realiza bajo un metamodelo definido por este estándar, el cual especifica los elementos del diagrama y permite su representación en Java. Para el proceso de ingeniería de software se utiliza un modelo hibrido entre dos metodologías de software, Programación Extrema XP y El Proceso Unificado UP.

CLASS Modeler permite compartir diagramas entre usuarios de la aplicación, lo cual posibilita la edición colaborativa y el trabajo concurrente de diferentes usuarios sobre el mismo diagrama. Esta representa una de las características más importantes de la herramienta donde destacan aspectos como el control de la concurrencia y la sincronización entre la vista de cada usuario y la representación del servidor.

**Palabras Clave:** Aplicación Web, Diagramas de Clase, UML, Herramientas CASE, Desarrollo de Software, Java.

**ABSTRACT**

This article exposes the development process of the tool CLASS Modeler; a web application based on RIA that allows drawing UML class diagrams and generating source code in Java language. This tool was created to support the educational process of students of Systems Engineering Program at the Francisco de Paula Santander University.

CLASS Modeler was developed using Java language Enterprise Edition; J2EE standard was the main development environment tool, incorporating advanced technologies in building layer-based applications. Additionally, it is based on the formal specification of UML in version 2.4.1, defined by the *Object Management Group* [1]; source code generation is performed under the metamodel defined by this standard, which specifies the diagram’s elements and allows its representation in Java. The software engineering process is a hybrid model between two software methodologies, Extreme Programming XP and the Unified Process UP.

CLASS Modeler also allows sharing diagrams between users of the application, which makes possible the collaborative edition and concurrent work of different users on the same diagram. This represents one of the most important features of the tool in which highlights issues like concurrency control and synchronization between user’s view and server representation.

**Index Terms:** Web Application, Class Diagrams, UML, CASE Tools, Software Development, Java.

1. **INTRODUCCIÓN**

Las herramientas CASE son un conjunto de programas que dan asistencia a los analistas, ingenieros de software y desarrolladores durante los pasos del ciclo de vida de desarrollo de un software [2]. CLASS Modeler se considera dentro de este tipo de herramientas, apoyando el desarrollo de software mediante el diseño de diagramas de clase.

En proyectos de investigación realizados anteriormente se determinó que uno de los principales factores de deserción y mortalidad académica en el Programa de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander, es la falta de práctica y comprensión de conceptos como abstracción y modelamiento de sistemas, así como la falta de ambientes para realizar actividades de afianzamiento y refuerzo [3].

Si bien se conocía de antemano la existencia software especializado en el diseño de diagramas UML, se determinó también, en base a observaciones y manifestaciones de estudiantes y docentes, que la tecnología existente no era suficiente en algunos casos. Debido que la mayoría del software se instala de manera nativa, varios requieren configuración y actualización manual, así como de licenciamiento; lo cual resulta en un esfuerzo para adecuar ambientes de trabajo para grupos de estudiantes.

Se planteó el desarrollo de una aplicación RIA[[1]](#footnote-1), que realizara las funciones de una herramienta CASE para el diseño de diagramas de clase UML y generación de código fuente en Java, que proporcionara características propias de una aplicación web como, portabilidad, facilidad de acceso y facilidad de uso.

CLASS Modeler, nombre que se le dio a la aplicación, se clasifica dentro del modelo de servicios *SaaS* establecido por Cloud Computing[[2]](#footnote-2); el cual define el software como un servicio que puede ser accedido bajo demanda desde cualquier lugar y en cualquier momento a través de internet.

CLASS Modeler dispone de un conjunto de características destinadas a la construcción de diagramas de clase, a continuación se mencionan las características principales de la herramienta:

* Generar de código fuente.
* Generar de imágenes PNG.
* Compartir diagramas con otros usuarios.
* Diseñar diagramas de manera concurrente.
* Generar métodos *get* y *set* de un atributo.
* Generar método *constructor* de una clase.

1. **DESARROLLO DEL PROYECTO**
   1. **METODOLOGIA**

En términos de proceso de desarrollo, se utilizó un hibrido entre dos metodologías, XP y UP. Esto permitió enfocar el desarrollo de manera iterativa y adaptable a los cambios, sin dejar de lado los lineamientos un proceso robusto y bien documentado.

A nivel de Extreme Programming se utilizaron conceptos importantes entre los cuales cabe destacar: A) Definición de un proceso basado en iteraciones, en donde el resultado de cada iteración fue una versión utilizable del sistema. B) Cada iteración se conformó por un conjunto de *Historias de Usuario*, las cuales enuncian las funcionalidades a implementar; estas historias fueron producto de un análisis de las herramientas CASE más utilizadas por estudiantes y docentes del programa académico. C) Se destaca un proceso con alta respuesta a los cambios y la posibilidad de realizar refactorización del código, permitiendo una implementación más sencilla y eficaz en las iteraciones finales.

Por otro lado, del Proceso Unificado también se acogieron varias prácticas: A) Se establecieron los actores del sistema y roles del proyecto; B) Se realizó un análisis y diseño basado en UML, donde se definió un *Diagrama de Clases* por cada iteración, así como la especificación de cada funcionalidad en forma de *Diagrama de Secuencia*. C) En fase de análisis y diseño de cada iteración se determinaron los datos persistentes y se constituyó el modelo de datos general de la aplicación.

En última instancia, cabe destacar un proceso apoyado por las pruebas unitarias, que verificaron el funcionamiento de los servicios EJB que contienen la lógica de negocio del sistema en general; la sección 2.5 explica en detalle este aspecto.

* 1. **ARQUITECTURA**

La arquitectura del proyecto CLASS Modeler está basada en capas, en donde cada una de ellas tiene una labor específica y se comunica con las demás mediante interfaces bien definidas.

La siguiente figura muestra las capas de la aplicación:

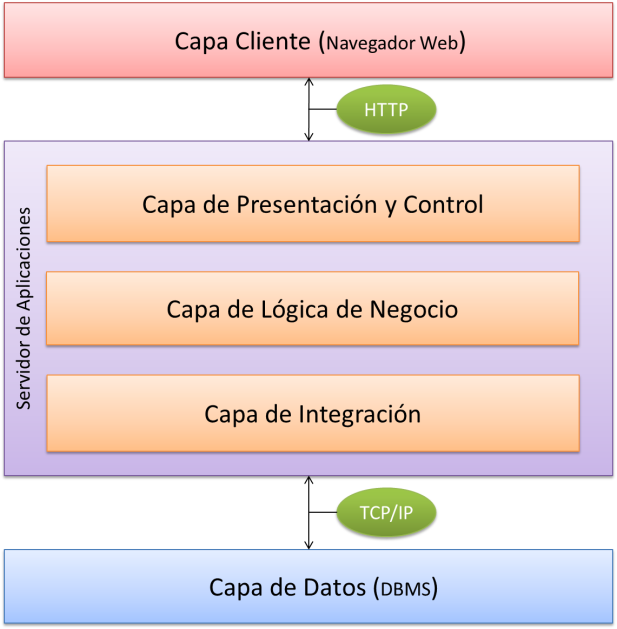


Figura . Arquitectura de CLASS Modeler

Se evidencian claramente 3 ambientes de ejecución: *Navegador Web*, *Servidor de Aplicaciones* y *Servidor DBMS*. Cada ambiente de ejecución representa una plataforma software sobre la cual se ejecuta la aplicación. Las siguientes capas definen la arquitectura de CLASS Modeler:  
  
**Capa Cliente**: Contiene la aplicación web que se ejecuta en el navegador y mediante la cual el usuario interactúa con el sistema. Esta capa es uno de los aspectos más importantes de CLASS Modeler, debido a que representa de manera gráfica un diagrama de clases. Esta representación se hace en formato SVG[[3]](#footnote-3) mediante la ayuda de la librería JavaScript mxGraph [4], en donde cada elemento del diagrama tiene una definición gráfica en formato vectorial y una definición lógica en formato XML; los datos entre el cliente y el servidor viajan en este formato. La visualización, colores, estilos y formatos de texto son manejados comúnmente por medio de hojas de estilo en cascada CSS.  
  
**Capa Presentación y Control**: Se encarga de recibir las peticiones HTTP síncronas y asíncronas, enviadas desde la capa cliente. Estas peticiones siguen en su mayoría el ciclo de vida definido por el Framework JSF [5], el cual se basa en el patrón de arquitectura MVC[[4]](#footnote-4).

Las peticiones son recibidas todas por el Servlet[[5]](#footnote-5) central de JSF, quien direcciona la petición al controlador especifico que se encarga de procesarla, aplicando otro patrón de diseño llamado *Front Controller* [6].

Cada controlador tiene la posibilidad de conectarse con la capa de negocio para realizar verificaciones y mantener consistencia de los datos manejados por el cliente. Por ejemplo, verificar que un usuario tiene privilegios necesarios para poder editar un diagrama específico, o solicitar la generación del código fuente del diagrama.

**Capa de Lógica de Negocio**: Esta capa se encarga del manejo de los datos dominio de la aplicación, es decir, el manejo de los diagramas del usuario, compartir diagramas entre ellos, verificar y controlar privilegios, acceder y escribir en la fuente de datos. Todas estas tareas son llevadas a cabo por servicios EJB[[6]](#footnote-6), los cuales manipulan los datos en base a las reglas definidas por el negocio o dominio del sistema. Se siguió el patrón de diseño *Facade*, ocultando la implementación de los servicios y proporcionando únicamente Interfaces a los controladores JSF a través de *Inyección de Dependencias*[[7]](#footnote-7) utilizando anotaciones del API Java.

La mayoría de los EJBs implementados son *Stateless* [7], es decir, servicios sin estado; debido a que casi todos representan operaciones atómicas de acceso y manipulación de datos que no requerían almacenamiento de información temporal.

Por otro lado se utilizaron servicios externos que proveen funcionalidades importantes para la aplicación, entre ellos destacan las librerías *UML2* y *StringTemplate*, enfocadas en la generación de código fuente a partir del metamodelo del diagrama diseñado.

**Capa de Integración**: Permite la comunicación e interacción con la fuente de datos persistentes. Se utilizó el Framework JPA [7], como mecanismo para facilitar el acceso a datos. Las clases del dominio se mapearon mediante anotaciones del API y se obtuvo una representación objeto-relacional del sistema. Se utilizó una implementación de código abierto del estándar JPA, llamado *EclipseLink* versión 2.4.1.

**Capa de Datos**: Permite almacenar de manera persistente los datos del sistema. Para esta tarea se utilizó un sistema de bases de datos relacional común como lo es MySQL. Almacenando datos básicos como la información de los usuarios y los diagramas que posee cada uno de ellos, sin embargo, quizás la información más critica que debe manejar la aplicación es la representación de los diagramas. Considerando que llevar un diagrama UML a un modelo relacional consiste en un proceso computacional bastante complejo y teniendo en cuenta aspectos de concurrencia, almacenamiento y carga de datos, se decidió almacenar toda la representación del diagrama en formato XML como una cadena de String.

* 1. **TECNOLOGIAS UTILIZADAS**

Las tecnologías utilizadas se clasifican dentro de cada capa de la arquitectura mencionada anteriormente, es así como cada tecnología cumple con su labor específica dentro de su ambiente de ejecución.

La siguiente tabla resume las tecnologías utilizadas para el desarrollo de CLASS Modeler.

|  |  |
| --- | --- |
| **Capa** | **Tecnología** |
| Cliente | jQuery/JavaScript |
| CSS |
| HTML |
| XML |
| mxGraph |
| Presentación y Control | JSF |
| PrimeFaces |
| Facelets |
| XHTML |
| Servlets |
| XML Parser |
| Lógica de Negocio | Enterprise JavaBeans |
| UML2/EMF |
| JavaMail |
| StringTemplate |
| Integración | JPA |
| Datos | MySQL |

Tabla . Tecnologías utilizadas en el desarrollo de CLASS Modeler

* 1. **DE UML A CÓDIGO FUENTE**

La generación de código representó uno de los aspectos más importantes de implementación del proyecto. Como se mencionó anteriormente, CLASS Modeler está basado en una implementación del metamodelo definido por la OMG para el estándar UML. Este metamodelo define una estructura de clases relacionadas que conforman la vista estática del lenguaje.

Esta implementación se llama UML2 [8], la cual es una librería JAR[[8]](#footnote-8) que contiene un conjunto de clases representando cada elemento de los diagramas UML: Clase, Interfaz, Relación, Atributo, Método. Esta librería es parte del *Eclipse Modeling Framework,* que es una suite de componentes para diseño de diagramas para el IDE Eclipse.

En la capa cliente de CLASS Modeler, los diagramas son representados en formato XML. Cada elemento del diagrama contiene un objeto XML y los atributos representan la información básica de dicho elemento: nombre, visibilidad, tipo de dato, entre otros.

La comunicación entre el cliente y el servidor se realiza mediante el envío de mensajes AJAX transportando el XML de los elementos del diagrama.

Al momento de generar el código fuente, todo el XML del diagrama es procesado mediante un *XML Parser* y se obtiene un modelo UML basado en objetos. La transformación de XML a Objetos realizada mediante el Parser, es una implementación propia desarrollada para el proyecto, utilizando las clases del dominio provistas por la librería UML2.

Una vez se obtiene la representación del modelo UML, se realiza la generación del código fuente en lenguaje Java. Para esta labor se implementó un servicio EJB que permite tomar el modelo y en base a un sistema de plantillas de texto, generar cualquier tipo de archivos de texto plano.

La herramienta utilizada para la generación de los archivos de código se llama StringTemplate [9], esta librería de código abierto, permite definir plantillas de texto plano con secciones variables que pueden ser reemplazadas utilizando un modelo de objetos.

La siguiente imagen ilustra el proceso de transformación que sigue la herramienta para generar el código fuente para una clase llamada *Persona*.

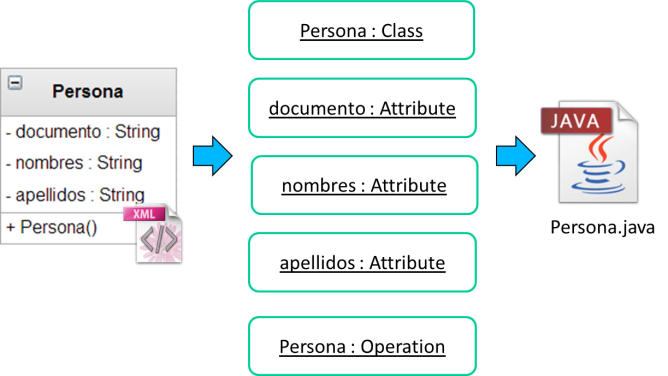


Figura . Transformación de los datos de un diagrama de clases hasta código fuente

* 1. **PRUEBAS DE UNIDAD**

Las pruebas de unidad se definieron al inicio de cada iteración, estas pruebas se enfocaron en la verificación de los servicios EJB que soportan la lógica de negocio de toda la aplicación, incluyendo los servicios que permiten la generación de código y la interacción con la capa de datos.

Para esta labor se dispuso de un Framework para realizar pruebas automáticas, llamado TestNG [10]. Este Framework permite configurar ambientes en los cuales se pueden ejecutar pruebas de software simulando ambientes reales.

El Framework se usó en conjunto con el servidor de aplicaciones Glassfish embebido. Al iniciar la ejecución de las pruebas se lanza de manera automática una instancia del servidor que se encarga de desplegar todos los servicios que serán probados.

1. **RESULTADOS**

Con el desarrollo de CLASS Modeler, se logró construir una herramienta de apoyo para el proceso de aprendizaje de los estudiantes del Programa de Ingeniería de Sistemas de la Universidad Francisco de Paula Santander; muy útil para la realización de prácticas sobre el modelado de diagramas de clase, promoviendo aspectos importantes como el trabajo en equipo y la colaboración.

Se obtuvo una aplicación web que permite generar código fuente en lenguaje Java a partir de los diagramas construidos y que permite a los usuarios trabajar de manera conjunta en la realización de diagramas de clase UML. CLASS Modeler simula la creación de diagramas de clase un ambiente de escritorio tradicional, mediante la incorporación del concepto RIA, buscando mejorar la experiencia de usuario y la productividad, y proporcionando beneficios que resultan atractivos a la hora de utilizar una herramienta de modelado UML, tales como facilidad de acceso, inmediatez y portabilidad de la información. CLASS Modeler solo necesita una conexión a internet y un navegador web para ser usado, y no requiere instalación de complementos como Flash o Shockwave.

Se obtuvo una herramienta con una arquitectura empresarial separada en capas, que permite fácil escalabilidad y mantenimiento.

1. **CONCLUSIONES**

CLASS Modeler se puede clasificar dentro del modelo de servicios SaaS, donde el soporte lógico y los datos de la aplicación se encuentran en un servidor de TI y los clientes acceden bajo demanda mediante un navegador web.

Un aspecto muy importante del proyecto fue la capacidad de convertir el metamodelo de clases de UML a código fuente en lenguaje Java. Esto representó un logro bastante importante debido a la complejidad implícita del proceso y la diferencia que existe entre ambos lenguajes. En algunos casos fue necesario extender el metamodelo para soportar conceptos propios de Java, como son los Tipos Genéricos, utilizados a la hora de crear colecciones de objetos.

La arquitectura de la aplicación representó un reto considerable a la hora de realizar la implementación del sistema, debido a que existen muchas tecnologías enfocadas en aspectos específicos de cada capa. Se logró realizar una integración satisfactoria de un conjunto de tecnologías para el desarrollo web en lenguaje Java. Por otro lado, se aporta todo el conocimiento adquirido en este proyecto para la comunidad del Programa de Ingeniería de Sistemas, de tal manera que los estudiantes se interesen en el desarrollo de software y promuevan el uso de tecnologías de punta en la realización de sus proyectos.

**REFERENCIAS**

1. **OMG Object Management Group.** Unified Modeling Language Superstructure Definition. [En línea] 6 de Agosto de 2011. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/. formal/2011-08-06.

2. **Sommerville, Ian.** *Ingenieria de Software.* Madrid : Pearson Education S.A., 2005. ISBN 8478290741.

3. **Amaya Torrado, Yegny Karina y Herrera Angarita, Lady Torcoroma.** *Identificación de las causas que generan problemas en el aprendizaje de fundamentos de programación de computadores en las facultades de ingenieira de sistemas de las universidades de la ciudad de Cúcuta.* Cúcuta N. de S. : Universidad Francisco de Paula Santander, Biblioteca Eduardo Cote Lamus, 2003. TIS 371.334A489i.

4. **Benson, David y Alder, Gaudenz.** mxGraph JavaScript HTML 5 diagramming library. [En línea] JGraph Company, 26 de 12 de 2012. [Citado el: 23 de 01 de 2013.] http://www.jgraph.com/mxgraph.html.

5. **Geary, David y Horstmann, Cay.** *Core Java Server Faces.* Boston : Prentince Hall, 2010. ISBN 9780137012893.

6. **Gamma, Erich, y otros.** *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison-Wesley, 1994. 0-201-63361-2.

7. **Keith, Mike y Schincariol, Merrick.** *Pro JPA 2 Mastering the Java Persistence API.* New York : Apress, 2009. ISBN 9781430219569.

8. **Eclipse Org.** UML2/MDT Documentation. [En línea] http://wiki.eclipse.org/MDT-UML2.

9. **Parr, Terence.** Documentación de la libreria StringTemplate. [En línea] 2013. https://theantlrguy.atlassian.net/wiki/display/ST4/StringTemplate+4+Documentation.

10. **Beust, Cédric y Suleiman, Hani.** *Next Generation Java Testing: TestNG and Advanced Concepts.* Boston : Addison Wesley, 2008. 978-0-321-50310-7.

1. Rich Internet Applications, son aplicaciones web que tienen la mayoría de características de las aplicaciones de escritorio normales. [↑](#footnote-ref-1)
2. Cloud Computing, es un paradigma que permite ofrecer servicios de computación a través de internet. [↑](#footnote-ref-2)
3. Los SVG *Scalable Vector Graphics*, son una especificación para describir gráficos vectoriales

   bidimensionales tanto estáticos como animados en formato XML. [↑](#footnote-ref-3)
4. MVC *Modelo-Vista-Controlador*, es un patrón de arquitectura que separa los datos y la lógica de negocio de la vista de una aplicación. [↑](#footnote-ref-4)
5. Un Servlet es un componente Java que responde a solicitudes HTTP y se ejecutan en el lado del servidor. [↑](#footnote-ref-5)
6. EJB *Enterprise Java Beans*, son una de las APIs que forman parte del estándar de construcción de aplicaciones empresariales J2EE. [↑](#footnote-ref-6)
7. Inyección de Dependencias, es un patrón de diseño orientado a objetos, en el que se le suministran objetos a una clase en lugar de que ella misma se encargue de crearlos. [↑](#footnote-ref-7)
8. JAR es un tipo de archivo que permite ejecutar aplicaciones escritas en el lenguaje Java. [↑](#footnote-ref-8)