PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

Análisis, Diseño e Implementación de un Generador MDA de Aplicaciones Java Web

Tesis para optar por el Título de Ingeniero Informático, que presenta el bachiller:

Carlos Humberto Balbuena Palacios

ASESOR: Ing. Luís Flores

Lima, Febrero del 2010

Índice General

E) Patrones de Conversión de objetos a tablas
F) Modelos de Generación de Código
G) Casos de Prueba

Índice de Figuras

Índice de Cuadros

Introducción

Se sabe que en el desarrollo de software un gran porcentaje de las horas de programación están destinadas a la construcción de aplicaciones o módulos que carecen de un modelo de negocio complejo, y que además producen interfaces de usuario relacionadas directamente con las entidades modeladas en etapas anteriores de análisis. Para este tipo de aplicaciones se puede obtener una reducción considerable en los tiempos de desarrollo a través del uso de MDA.

La arquitectura orientada a modelos (MDA) fue formalmente introducida por el OMG (*Object Management Group*) en el año 2001 como un término que abarcaba un gran número de aplicaciones para el modelado y elaboración de especificaciones de arquitectura. Desde entonces, estas especificaciones y su uso han ido evolucionando en forma considerable, y el término MDA es ahora ampliamente reconocido en la industria de tecnologías de información. Sin embargo, MDA va mas allá de ser un conjunto de tecnologías o de un modo específico para generar código, actualmente provee un marco de trabajo para la administración de diferentes maneras de racionalizar y automatizar la especificación, desarrollo, distribución e integración del software.

Por ello, el presente proyecto busca dar soporte a la metodología MDA a través del desarrollo de un marco de trabajo de generación de código *open source* para aquellas aplicaciones que no presenten un modelo de negocio complejo y cuyo desarrollo requiera ser automatizado.

1. Generalidades

En este capítulo se definirán las bases para el presente proyecto de fin de carrera, definiendo primero el problema a resolver, el marco conceptual con los conceptos básicos para el entendimiento del proyecto, el plan de proyecto usado para la gestión del proyecto y el estado del arte, en el cual se hace un análisis comparativo de la situación actual de las herramientas similares a la desarrollada en este proyecto.

1.1 Definición del problema

En estos últimos años han sido desarrollados generadores de código lo suficientemente robustos y versátiles, al punto de generar aplicaciones en cuestión de minutos, utilizando para ello algunos pasos simples de configuración. Dichos generadores, generalmente tienen como punto de partida inicial una fuente de datos previamente generada, tal como una base de datos física, a partir de la cual la aplicación es creada. Sin embargo, para aplicaciones complejas cuyo punto de partida inicial consiste en la creación de un modelo general, por ejemplo: un diagrama de clases, definido en notaciones estándares tales como UML. La gran mayoría de los generadores de código actuales no soportan este tipo de modelos, siendo necesaria la creación de la fuente de datos física primero.

Con el propósito de resolver este tipo de problema es que nació la metodología MDA. Una Arquitectura de Modelo o MDA por sus siglas en ingles, que es capaz de crear código

tomando en consideración únicamente el modelo de la aplicación, sin la necesidad de definir una base de datos física primero.

Este trabajo tiene por objetivo el desarrollo de una herramienta MDA para la generación de aplicaciones Web.

1.2 Marco conceptual del problema

En esta sección se describen las bases teóricas necesarias para el entendimiento del presente proyecto.

1.2.1 MDA - Arquitectura dirigida por Modelos

.

MDA **[15]** es un conjunto de especificaciones y estándares definidos por el OMG(Object Management Group) con el propósito de dar una solución al problema de la adaptación constante entre los modelos de negocio y las tecnologías usadas para la implementación de éstos como sistemas de información.

MDA se apoya sobre los siguientes estándares:

- UML: es un lenguaje visual estandarizado para el modelado de objetos.
- MOF: es un estándar de OMG para la ingeniería orientada a modelos.
- XMI: es un estándar de OMG para el intercambio de *metadata* vía XML.

A continuación se presentan los conceptos claves para entender el funcionamiento de MDA **[15] :**

- **Sistema**
  Se entiende por sistema a un conjunto de componentes o sistemas.
- **Modelo**
  Es la especificación del sistema y su entorno.
- **Dirigido por modelos**
  Se entiende dirigido por modelos debido a que los modelos forman y dirigen todas las etapas del desarrollo del sistema.
- **Arquitectura**

La arquitectura de un sistema es la especificación de las partes del mismo y las interacciones entre ellos.

- **Punto de vista**

  Es una abstracción que hace uso de un conjunto de conceptos de arquitectura y reglas estructurales para centrarse en aspectos particulares del sistema, obteniendo un modelo simplificado.

- **Vista**

  Es una representación del sistema desde un determinado punto de vista.

- **Plataforma**

  Una plataforma es un conjunto de subsistemas y tecnologías que aportan un conjunto coherente de funcionalidades.

- **Aplicación**

  Para MDA una aplicación viene a ser una funcionalidad a ser desarrollada.

- **Independencia de la plataforma**

  Es la característica que debe tener todo modelo en MDA. Consiste en la independencia del modelo sobre cualquier plataforma o tecnología ya sea a nivel de hardware o software.

- **Puntos de vista MDA**
  - ✓ Punto de vista independiente de la computación: se centra en el entorno del sistema y los requisitos para el mismo.
  - ✓ Punto de vista independiente de la plataforma: se centra en las operaciones del sistema, mientras oculta los detalles necesarios para una determinada plataforma.
  - ✓ Punto de vista de plataforma específica: combina el punto de vista independiente de la plataforma con un enfoque específico para su uso en una plataforma específica en un sistema.

- **Modelo independiente de la computación(CIM)**

Un modelo independiente de la computación (CIM) es una vista del sistema desde el punto de vista independiente de la computación. Se puede entender como el modelo del dominio, y se usa vocabulario propio de los expertos en el dominio.

- **Modelo independiente de la plataforma(PIM)**

Es una vista del sistema desde el punto de vista independiente de la plataforma. Expone un carácter independiente de la plataforma sobre la que se desplegará, de modo que pudiera ser empleado en diferentes plataformas de carácter similar como por ejemplo diferentes administradores de bases de datos o lenguajes de programación.

- **Modelo específico de la plataforma(PSM)**

Es una vista de un sistema desde el punto de vista dependiente de la plataforma. Combina las especificaciones del modelo independiente de la plataforma con los detalles que especifican el uso de una plataforma específica por parte del sistema.

- **Transformación de modelos**

  Es la serie de transformaciones de un modelo a otro para un mismo sistema.

- **Mapas de transformación**

  Son el conjunto de reglas para la transformación de un PIM a un PSM para una plataforma en concreto.

Definidos los conceptos se puede proceder a definir el funcionamiento de MDA de manera general:

El proceso de transformación inicia una vez que se tienen claramente definidos los requerimientos. Después de esto se inicia el modelamiento MDA mediante la definición de PIMs, los cuales capturan la lógica del negocio en forma independiente a cualquier tecnología, ya sea a nivel de hardware o software. Luego estos PIMs son traducidos a PSMs, a través de mapas de transformación, en los cuales ya se observan características propias de la plataforma. Finalmente, los PSMs definidos son convertidos a código, los cuales serán ejecutados en la plataforma seleccionada. Este proceso puede ser observado en la figura 1.

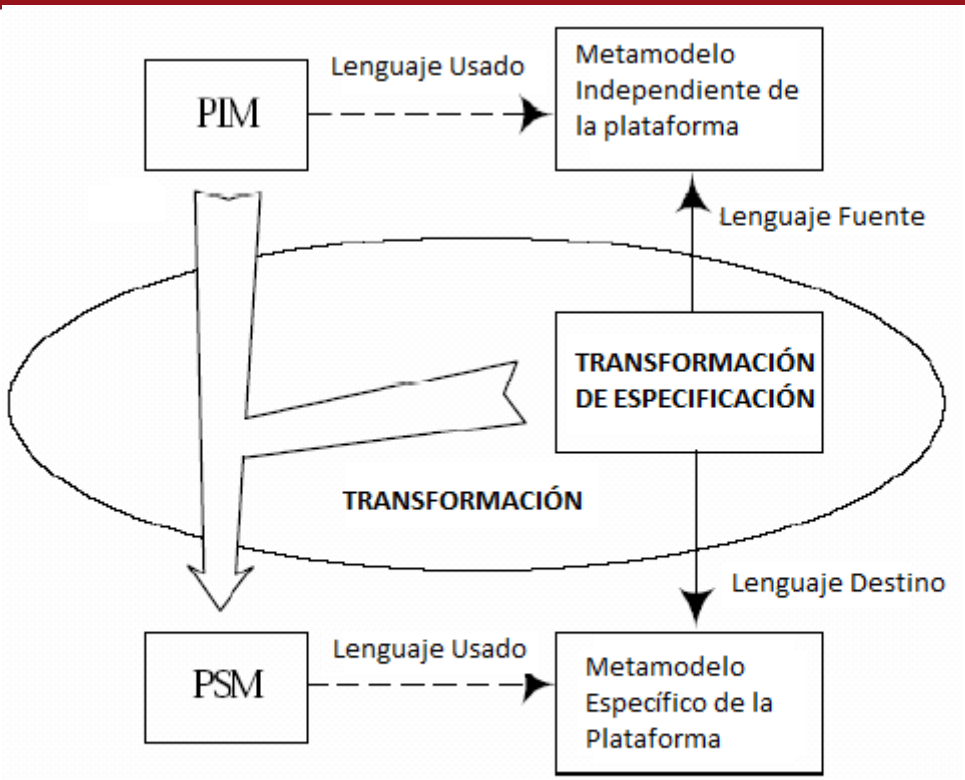


FIGURA 1: TRANSFORMACIONES MDA. (ADAPTADO DE [15])

1.2.2 Herramienta MDA

Se define una herramienta MDA [15] como aquella que provee los medios para la automatización de los procesos(transformaciones) propias del paradigma MDA.

1.2.3 UML – Lenguaje Unificado de Modelado

Permite definir de un modo gráfico y estándar los diferentes procesos y funcionalidades que ofrece un sistema [19] y aspectos más técnicos, tales como la base de datos a usar o los componentes que conforman el sistema, todo esto a través de las diferentes especificaciones que posee.

1.2.4 Modelo de datos relacional

Es la representación a nivel lógico de data estructurada [4] de modo que se tenga un mayor entendimiento del tipo, estructura y características de los datos a ser almacenados en la base de datos.

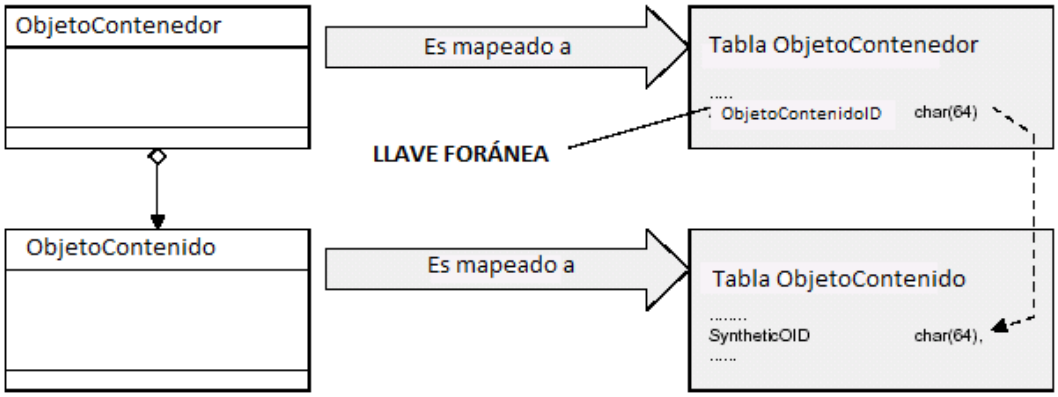
1.2.5 XMI - XML Metadata Interchange

Permite el intercambio de diagramas **[18]**, a través de su especificación con la cual se puede saber como está definido el modelo-. Su principal propósito es el intercambio de información entre herramientas de modelado.

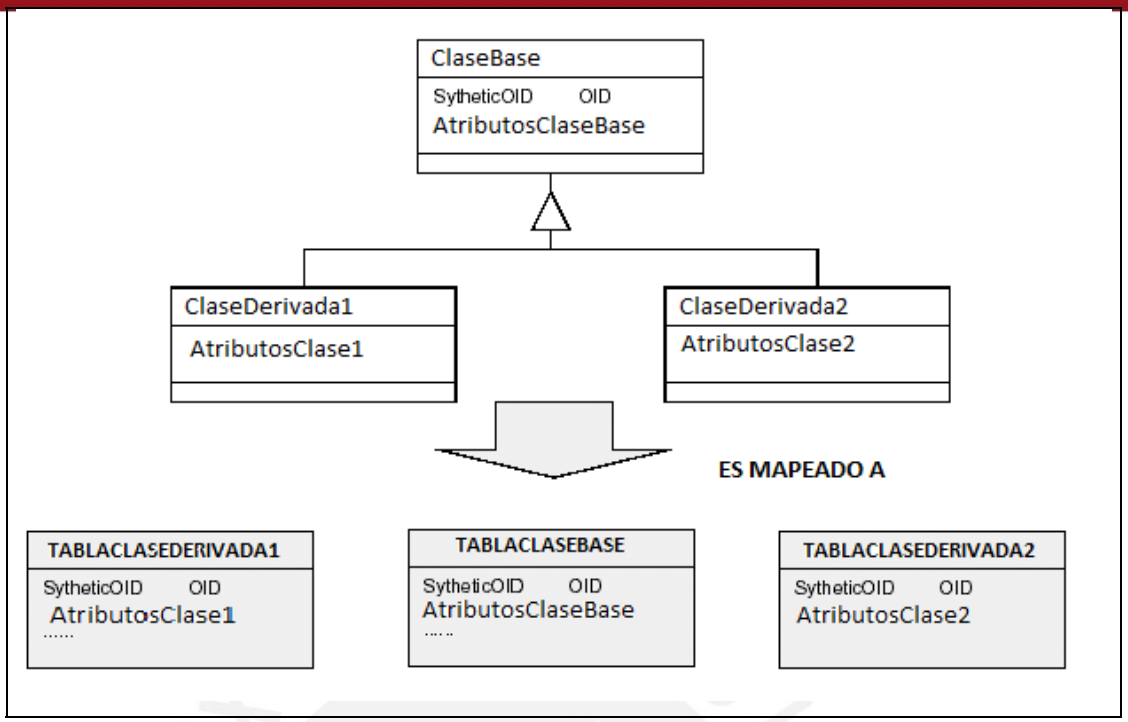
1.2.6 Patrones de conversión de objetos a tablas

A continuación se muestran los patrones principales usados para el mapeo de objetos a tablas **[3]**:

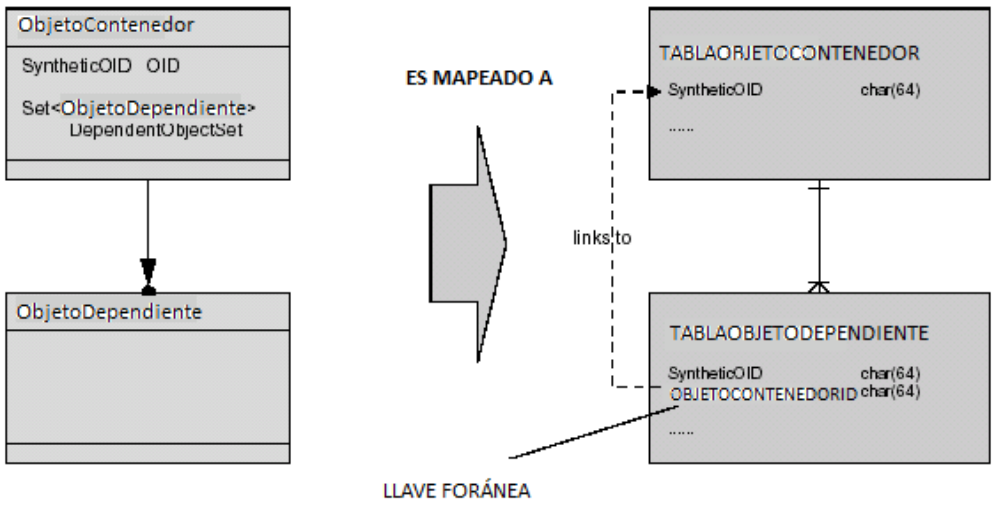
Foreign Key Aggregation – Agregación	
Descripción	En una agregación participan dos objetos: un objeto *contenido* por otro objeto, y un objeto *contenedor* que contiene muchos objetos *contenido*. El patrón consiste en definir dos tablas por separado para cada uno de los objetos que participan en la agregación y luego insertar una llave foránea en la tabla del objeto *contenedor* que hace referencia a una llave definida en el objeto *contenido*.



One Class One Table – Herencia	
Descripción	Consiste en definir una tabla para cada objeto que participa en la relación, y luego poner en cada una de las tablas de los objetos hijos la llave de su tabla padre como llave foránea.



Foreign Key Association – Asociación	
Descripción	En una asociación participan dos objetos: un objeto *dependiente* de otro objeto, y un objeto in*dependiente* que contiene uno o más objetos *dependientes*. El patrón consiste en definir una tabla individual para cada uno de los objetos que participan en la asociación luego se inserta la llave del objeto *independiente* en la tabla de los objetos *dependientes*.



1.2.7 Esquemas de generación de código

A continuación en la figura 2 y 3 se muestran los tipos de generadores de código principales para la construcción de la herramienta **[18]**:

Partial Class Generator

Un generador de código de clases parciales crea un conjunto de clases base para implementar un diseño específico en un archivo de definiciones. Estas clases base son usadas como la plataforma para construir las clases de producción, en las cuales se definen comportamientos específicos.

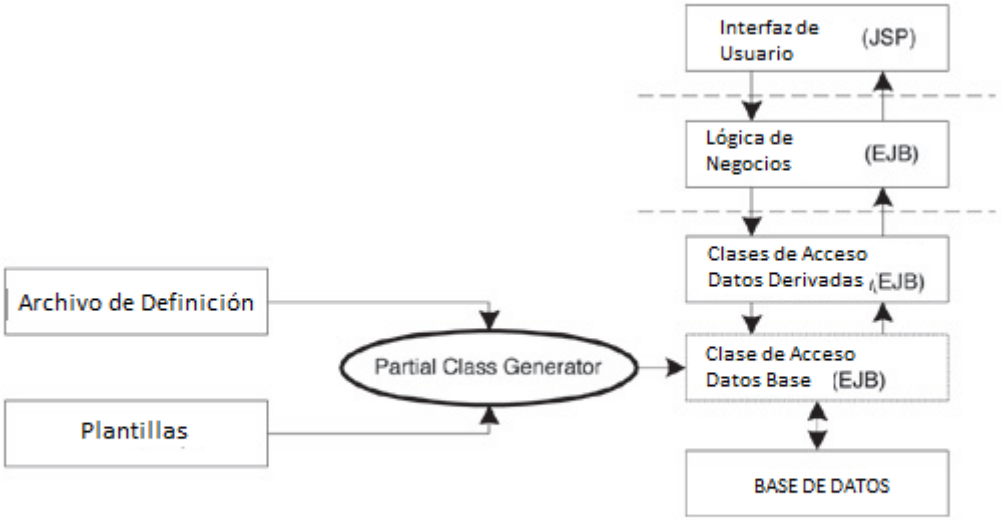


FIGURA 2: GENERACIÓN DE CLASES BASE DE ACCESO A DATOS. (ADAPTADO DE **[17]**)

Tier Generator

Un generador de capas construye y mantiene una capa de una aplicación.

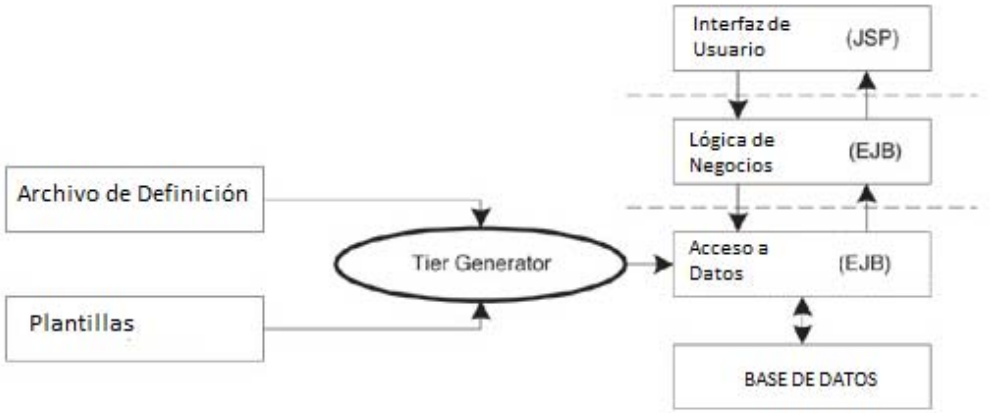


FIGURA 3: GENERACIÓN DE CAPAS DE UNA APLICACIÓN. (ADAPTADO DE [17])

1.2.8 Metamodelo UML

El metamodelo UML [20] es un modelo que describe el lenguaje UML, específicamente sus atributos, asociaciones, paquetes, colaboraciones, casos de uso, actores, mensajes, estados y todos los demás conceptos relacionados con el lenguaje UML. A continuación en la figura 4 se muestra el diagrama uml correspondiente al metamodelo para una clase:

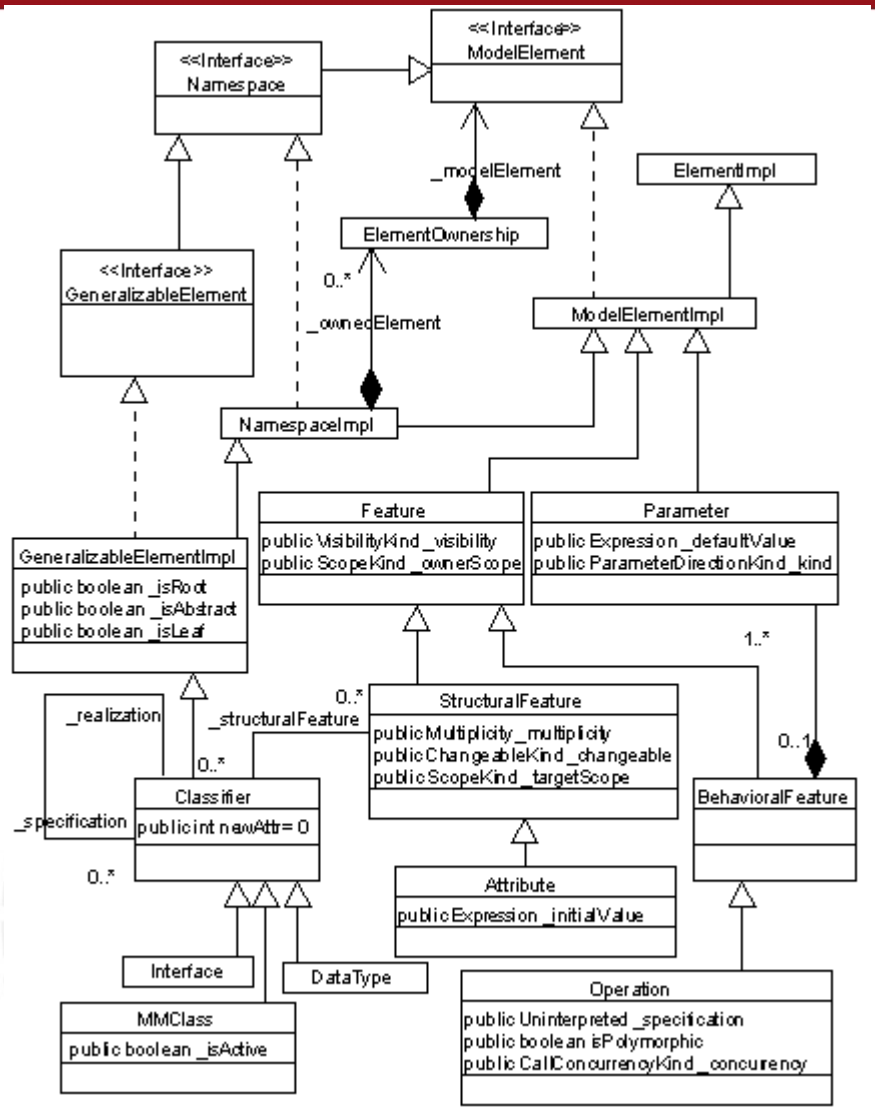


FIGURA 4: METAMODELO PARA LA DEFINICIÓN DE UNA CLASE (TOMADO DE **[19]**)

1.2.9 Arquitectura multicapas

A nivel lógico consiste en la división de la aplicación en componentes reutilizables e independientes. A nivel físico, se tiene la distribución de las capas en varios nodos físicos o procesos **[3]**.

1.2.10 J2EE

Plataforma de programación **[13]** para desarrollar y ejecutar software definido en lenguaje Java, con una arquitectura multicapas, la cual está basada en el uso de componentes de software y el uso de un servidor de aplicaciones.

1.3 Estado del Arte

En esta sección se describe como se resuelve actualmente el problema planteado con base en un enfoque teórico y otro tecnológico, en el cual se hace un análisis comparativo de las soluciones existentes.

1.3.1 Análisis Comparativo de Herramientas MDA

Las herramientas actuales descritas en el cuadro 1, que soportan la metodología MDA proveen las funcionalidades básicas y necesarias para la generación de código a través de un modelo genérico, por lo general definido en formato XMI o formatos propios de la herramienta. Sin embargo, carecen de un flujo completo que de soporte a la generación de una aplicación. Esta carencia es la que intenta satisfacer el presente proyecto en el cual a través de sus tres módulos busca la realización del flujo completo de generación de una aplicación desde el diagramado de clases hasta la generación de una aplicación funcional, sin embargo no restringe el uso de los módulos en forma independiente para la generación de por ejemplo únicamente scripts o de aplicaciones a partir de bases de datos ya existentes.

Herramienta	Descripción
ArcStyler	o Es una herramienta basada en el uso de cartuchos[1] para la descripción de transformaciones que permite generar aplicaciones de n capas codificadas en java/J2EE y c#/.NET a partir de diagramas UML y la especificación de los procesos del negocio. o Permite extender las capacidades de transformación, generando nuevos cartuchos a partir de UML, cuyo objetivo sea cualquier plataforma o lenguaje.
AndroMDA	o Es una herramienta también basada en cartuchos, que admite como entrada descripciones XMI de diagramas UML, y usa XDoclet como tecnología de marcado para el acceso a datos desde las clases Java. o Admite como entrada ficheros XMI versión 1.1.
Codagen Architect 3.2	o Esta herramienta permite la transformación de modelos y la automatización de la generación de código partiendo de los modelos CIM[2]. o Para la integración con herramientas de modelado, Codagen incorpora plugins para Microsoft Visio, Rational Rose y Borland Together Control. o Admite como entrada ficheros XMI versión 1.1 y ficheros MDL de Racional Rose.
IQGen 1.4	o El generador de modelos y código IQGen está desarrollado en Java. o Incorpora un entorno de modelado muy limitado, que se suple con la admisión de entrada de modelos en formato XMI.

[1] Componentes que permiten extender la funcionalidad básica de la herramienta.
[2] Modelo Independiente de la plataforma.

OptimalJ	o Este producto de la compañía Compuware genera aplicaciones J2EE partiendo de los modelos. Implementa completamente la especificación MDA. o Está desarrollado en Java, lo que le hace portable a cualquier plataforma para su ejecución. Dispone de plugins para los entornos de desarrollo integrado Eclipse y NetBeans. o Admite XMI versión 1.1 tanto para la importación de ficheros como para su salida.

CUADRO 1: CUADRO DE HERRAMIENTAS MDA.(ADAPTADO DE **[16]**)

Características/ Herramientas	ArcStyler	AndroMDA	Codagen Architect	IQGen	OptimalJ
Diagramas de Clases	X	X	X	X	X
Diagramas de Casos de Uso	X	X	X	X	x
Diagramas de Actividades	X	X	X	X	X
Diagramas de Estados	X	X	X	X	X
Diagramas de Secuencia	X	X			
Diagramas de Colaboración	X	X		X	X
Diagramas de Componentes		X		X	X
Diagramas de Despliegue		X		X	X
Restricciones OCL	X	X		X	X
Múltiples Lenguajes		X			
Java	X		X	X	X
C#	X		X	X	
C++			X		
Visual Basic			X		
Cobol				X	
XMI		x	x	x	x

CUADRO 2: CUADRO COMPARATIVO DE HERRAMIENTAS MDA.(ADAPTADO DE **[16]**)

1.4 Descripción y Sustento de la Solución

La solución planteada consiste en la implementación de una herramienta con soporte al paradigma MDA a través de tres módulos principales que se muestran en la Figura 5, los cuales cumplen funcionalidades bien definidas, y establecen una serie de etapas en la creación de la aplicación resultante: modelamiento de la aplicación, creación del modelo de base de datos y generación de la aplicación.

FIGURA 5: MÓDULOS DE LA SOLUCIÓN.

- **Modelador**

Este módulo permite al usuario el modelado de la aplicación mediante diagramas de clases con notación UML, el cual una vez completado, es procesado y representado en una estructura basada en la definición de la semántica UML. Con base en esta estructura, se procede a la generación del archivo XMI(XML Metadata Interchange), en el que se define el diagrama UML en formato XML, basado en estándares definidos por el OMG, con todas las definiciones correspondientes al diagrama de clases modelado por el usuario.

- **Conversor**

Este módulo es el encargado de la conversión del modelo en notación UML definido por el usuario, en formato XMI, el cual es procesado y transformado en una estructura de tablas relacionales, es decir, los objetos son mapeados a tablas de una base de datos relacional, y guardados en una estructura de datos. Con el modelo entidad relación definido se proceden a generar los scripts para el sistema de administración de base de datos (DBMS) elegido por el usuario.

- **Generador**

En este módulo se tiene como punto de partida, la base de datos física creada a partir de los scripts generados en el módulo anterior, esta información es procesada y a través de diferentes submódulos se genera código para una aplicación Web J2EE para cada una de las capas lógicas de la cual consistirá la aplicación: capa de presentación, capa de negocio y capa de datos. El código resultante puede ser directamente generado con los parámetros por defecto, o parámetros específicamente configurados como por ejemplo, la tecnología a usar para la persistencia de datos, entre otros. El resultado final de este módulo es una aplicación funcional, que pueda ser adaptada a las necesidades de los usuarios.

La solución planteada espera obtener los siguientes resultados:

- **Portabilidad**

  Aunque la principal plataforma a la cual apunta la solución es J2EE, se tiene que los módulos, los cuales pueden ser usados en forma independiente logran un alto nivel de portabilidad, por ejemplo: el modelo generado puede ser usado por otra

herramienta, p.e. AndroMDA, como entrada o los scripts generados por el módulo conversor pueden apuntar a diferentes base de datos. Esta portabilidad es propia de las herramientas MDA.

- **Calidad en el código generado**

  Debido a que se evitan las inconsistencias y no uniformidad del código, producto de las diferentes técnicas que aplican los programadores en el desarrollo de un software.

- **Mayores tiempos de diseño**

  En vista que la solución reducirá el tiempo de implementación del software en determinados tipos de proyectos, el equipo de desarrollo puede dedicar más tiempo a realizar un adecuado análisis y diseño del software a desarrollar.

- **Uniformidad arquitectónica**

  La solución usada en forma adecuada provee una estructura consistente, que puede servir como base para el entendimiento del funcionamiento de la arquitectura para miembros nuevos que se integren al equipo.

- **Moral alta**

  Los proyectos de larga duración pueden tener un impacto negativo en los equipos de desarrollo, y aún más cuando se tiene grandes cantidades de código repetitivo que escribir. La solución espera reducir los calendarios de los proyectos y mantener a los desarrolladores enfocados en tareas más desafiantes que escribir código monótono y repetitivo.

2. Análisis

En este capítulo se define la metodología de desarrollo de software a usar, los requerimientos funcionales de la herramienta y la viabilidad del sistema.

2.1 Definición de la metodología usada para la solución

La metodología de desarrollo de software seleccionada es una adaptación de RUP(Rational Unified Process) **[1]** , ya que sólo se requieren determinados artifacts **[1]** de RUP y solo se alcanzan las tres primeras etapas debido a la naturaleza del presente proyecto.

El principal motivo para la selección de esta metodología es el de asegurar desde las primeras fases el obtener un producto de calidad que cumpla con las características de funcionabilidad, usabilidad y fiabilidad. Se tiene también que RUP está basado en un enfoque iterativo incremental proporcionando iteraciones tempranas con un prototipo ejecutable que gradualmente evoluciona convirtiéndose en el sistema final y además tiene implícito en su proceso de desarrollo la evaluación continua de la calidad con respecto a los requerimientos de calidad deseados.

En resumen, los criterios para la selección de la metodología fueron :

- Desarrollo iterativo del software
- Administración de requerimientos
- Uso de arquitecturas basadas en componentes
- Modelado visual del software
- Control de la calidad del software
- Control de los cambios del software

2.1.1 Objetivos de Cada Fase

A continuación en el cuadro 3 se presentan los objetivos para cada una de las fases definidas por RUP **[1]**:

Fase	Iteración	Descripción	Puntos de Control Asociados	Riesgos Eliminados
Fase de	Iteración	Se modelan los requerimientos de la	Catálogo de Requisitos del	Prioriza los requisitos

Concepción	Preliminar	herramienta, el cronograma de tareas y los requerimientos del sistema	sistema	de la herramienta. Se delimitan los alcances del proyecto desarrollando un plan de trabajo a seguir. Determina la viabilidad del proyecto desde el punto de vista de los requerimientos establecidos.
Fase de Elaboración	Iteración Desarrollo de Prototipo de Revisión.	Se completa el análisis y diseño de todos los casos de uso identificados en los requerimientos. Desarrolla la arquitectura de los requisitos identificados en la iteración preliminar.	Especificación de Casos de Uso. Documentos de diseño(clases, secuencia).	Se establece la arquitectura del sistema de forma clara. Se reducen los riesgos técnicos. Se realiza un prototipo inicial para que sea probado.
Fase de Construcción	Iteración Desarrollo de la versión de prueba (V1 Beta).	Implementa y prueba los casos de uso de la iteración E1	Software versión de prueba (V1 Beta) : Modelador	Todas las funcionalidades imprescindibles para el usuario y la arquitectura se encuentran implementadas en esta versión beta.
	Iteración Desarrollo de la versión de Software (V1)	Implementar y probar los casos de uso faltantes en la versión de prueba, solucionar los errores identificados en la versión beta e incorporar retroalimentación obtenida a partir de V1 Beta. Desarrollar el sistema V1	Software versión 1 (V1) : Modulador y Conversor	Características de la versión V1 completamente revisadas. Calidad del producto garantizada. Minimización de defectos. Costo de calidad reducido.

	Iteración Desarrollo de la versión de Software (V2)	Diseño, implementación y prueba de Casos de Uso corregidos. Adición de mejoras y solución de imperfecciones detectadas sobre la base de casos de uso anteriores. Desarrollo del sistema V2. Empaquetado del sistema V2.	Software versión 2 (V2) : Modulador, Conversor y Generador Plan de Pruebas	Rápida implementación del software V2 que cuenta con mejoras revisadas. Empaquetado del sistema para su fácil instalación.

CUADRO 3: CUADRO DE FASES RUP DEL PROYECTO

2.1 Identificación de Requerimientos

A continuación se presentan los requerimientos identificados para el presente proyecto, estos requerimientos fueron tomados con el fin de dar soporte al flujo completo de generación de una aplicación java Web en base a la metodología MDA.

Ítem	Características	Prioridad
1	El sistema permitirá el modelado de diagramas de clases usando notación UML.	A
2	El sistema usará UML versión 2.0.	A
3	El sistema permitirá la generación de un archivo XMI a partir del diagrama UML modelado.	A
4	El sistema permitirá administrar las clases (agregar, modificar y eliminar).	A
5	El sistema permitirá administrar las relaciones (agregar, modificar y eliminar).	A
6	El sistema permitirá definir atributos con tipo de datos a las clases.	A
7	El sistema deberá permitir definir las relaciones básicas (agregación, asociación, herencia) entre clases.	A
8	El sistema deberá permitir la definición de cardinalidades entre clases.	A
9	El sistema permitirá definir etiquetas para las relaciones entre clases.	C
10	El sistema permitirá exportar el diagrama a otros formatos: JPG, XML, PNG, HTML.	C
11	El sistema permitirá la definición de propiedades a las relaciones y clases definidas.	A
12	El sistema permitirá el manejo de proyectos.	A
13	El sistema permitirá la persistencia de los proyectos.	A

14	El sistema permitirá el manejo de archivos de logs para el control de errores.	C
15	El sistema permitirá la navegación de los elementos del diagrama.	B
16	El sistema permitirá generar la documentación de las clases definidas en el diagrama de clases realizado.	C
17	El sistema permitirá el uso de diferentes patrones de conversión de UML a ER según el tipo de relación entre clases definidas en el diagrama de clases UML: asociación, agregación y herencia.	A
18	El sistema permitirá exportar el esquema ER a diferentes formatos de salida: JPG, XML, HTML.	C
19	El sistema permitirá el uso de los siguientes motores de base de datos: SQL Server, Oracle y SQLite.	A
20	El sistema permitirá la creación de scripts para la generación manual de la base de datos.	A
21	El sistema permitirá visualizar gráficamente el diagrama ER.	B
22	El sistema permitirá la navegación de los elementos del diagrama ER	B
23	El sistema permitirá la generación de un archivo de log para el manejo de los errores y los resultados de la conversión.	C
24	El sistema permitirá generar la documentación de las tablas en el diagrama ER obtenido.	C
25	El sistema permitirá la generación de una aplicación Web en la plataforma java.	A
26	El sistema permitirá generar aplicaciones con dos patrones arquitectónicos MVC o 3 Capas.	C
27	El sistema permitirá el manejo de los siguientes esquemas de persistencia de datos: DAO, ORM.	C
28	El sistema permitirá la generación de todas las capas o componentes del patrón de arquitectura elegido.	C
29	El sistema permitirá la configuración de la generación de la aplicación a través del uso de plantillas.	B
30	El sistema permitirá la definición de una base de código base y que pueda ser extendida por el usuario.	A
31	El sistema permitirá la modificación del aspecto de la interfaz grafica generada a través del uso de plantillas	C
		A: Alta M: Media C: Baja

2.1.1 Análisis del Sistema

En esta sección se definen las decisiones de análisis de la aplicación, necesarias para satisfacer los requerimientos planteados para el presente proyecto.

2.1.1.1 Casos de Uso

En la figura 6 se presentan los actores del proyecto y en el cuadro 4 la descripción de cada uno de ellos.

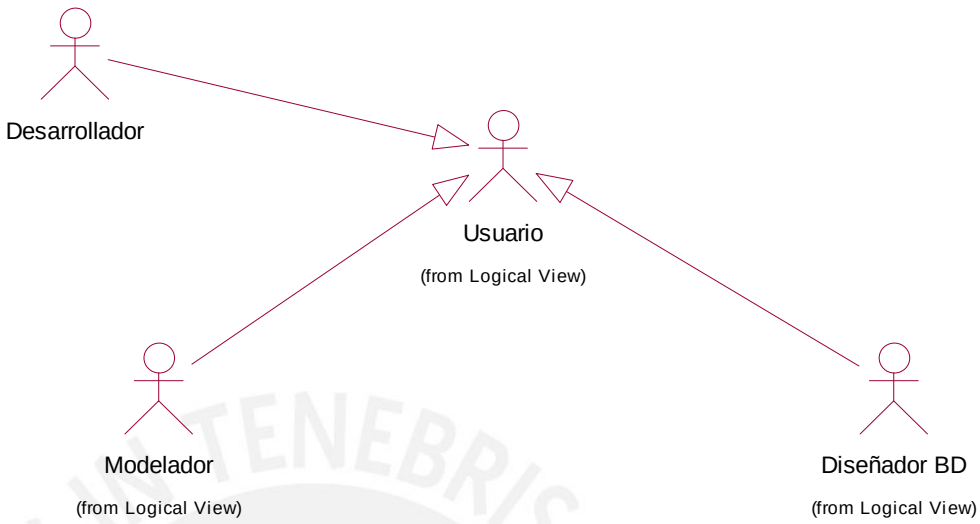


FIGURA 6: CATALOGO DE ACTORES

ACTOR	DESCRIPCION
Modelador	Es el encargado del desarrollo del diagrama de clases en notación UML 2.0, específicamente la definición de las clases, relaciones y propiedades que conforman el diagrama de clases.
Diseñador Base Datos	Es el encargado del diseño de la base de datos de la aplicación.
Desarrollador	Es el encargado de la implementación y mantenimiento de la aplicación.

CUADRO 4: CATALOGO DE ACTORES

A continuación se detallan los casos de usos más relevantes de cada módulo que conforma el proyecto.

Modulo Modelador

En la figura 7 se muestran los casos de uso del modulo Modelador, seguido de las especificaciones de los casos de uso principales.

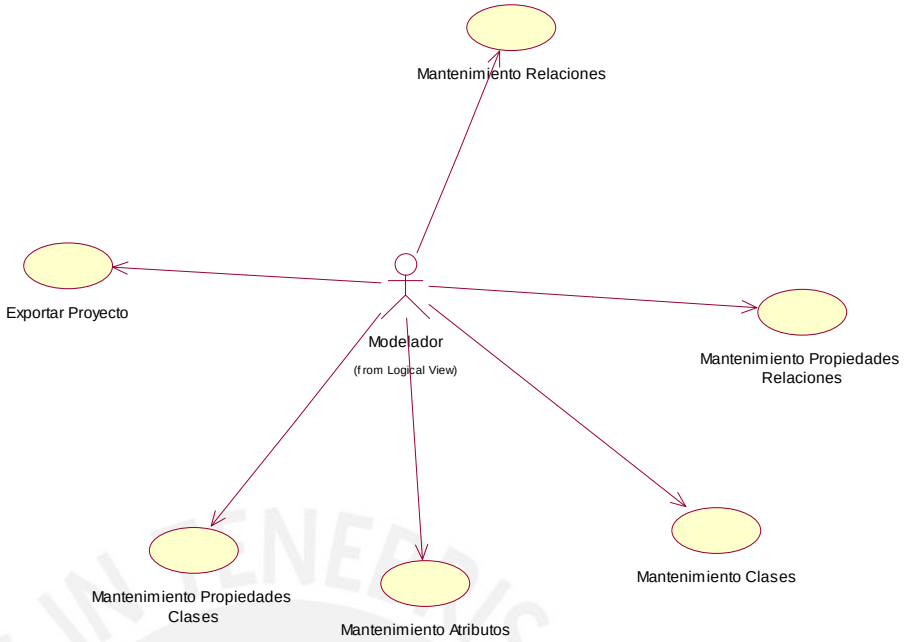


FIGURA 7: CASOS DE USO - MODELADOR

Mantenimiento de Clases

Mantenimiento de Clases	
Descripción	Permite la creación y eliminación de clases.
Actores	Modelador
Requerimientos Especiales	No existen requerimientos especiales.
Precondición	El usuario debe haber ingresado al sistema con el perfil de Modelador.
Flujo Básico	
1. El caso de uso inicia cuando el usuario elige la opción “Nueva Clase”. 2. El usuario indica el punto inicial en el área de dibujo del formulario del diagramado de clases. 3. El sistema muestra las propiedades de la clase. Incluir caso de uso <<Edición de Propiedades de Clases>>. 4. El sistema muestra la nueva clase en el formulario de diagramado de clases, y en el formulario de estructura de proyectos. 5. El caso de uso finaliza.	
Poscondición	Se realizaron los cambios en forma exitosa.
Flujo Alternativo “Eliminar Clase”	
1. El usuario selecciona la clase a eliminar en el formulario de diagramado de clases o del	

formulario de estructura del proyecto.

2. El usuario elige la opción “Eliminar Clase”.
3. El sistema muestra un mensaje de confirmación.
4. En caso de respuesta negativa, Ir al paso 6.
5. En caso de respuesta afirmativa, la clase es eliminada del formulario de diagramado de clases, y del formulario de estructura del proyecto
6. El caso de uso finaliza.

Mantenimiento de Relaciones

Mantenimiento de Relaciones	
Descripción	Permite la creación y eliminación de relaciones entre clases.
Actores	Modelador
Requerimientos Especiales	No existen requerimientos especiales.
Precondición	El usuario debe haber ingresado al sistema con el perfil de Modelador.
Flujo Básico	
1. El caso de uso inicia cuando el usuario elige la opción “Nueva Relación”. 2. El sistema muestra un formulario con el listado de los tipos de relaciones disponibles: 3. “Asociación”, “Herencia”., “Agregación”, y las opciones “Aceptar” y “Cancelar”. 4. Si el usuario elige “Cancelar”, ir al paso 7. 5. El usuario elige la clase inicial y final para el tipo de relación seleccionado., del formulario de diagramado de clases. 6. El sistema muestra las propiedades de la relación. Incluir caso de uso <<Edición de Propiedades de Relaciones>>. 7. El sistema muestra un formulario con la relación diagramada entre las dos clases seleccionadas, y en el formulario de estructura de proyectos. 8. El caso de uso finaliza.	
Poscondición	Se realizaron los cambios en forma exitosa.
Flujo Alternativo : Eliminar Relación	
1. El usuario selecciona la relación a eliminar en el formulario de diagramado de clases. 2. El usuario elige la opción “Eliminar Relación”. 3. El sistema muestra un mensaje de confirmación. 4. En caso de respuesta negativa, Ir al paso 6. 5. En caso de respuesta afirmativa, la relación es eliminada del formulario de diagramado de clases, y del formulario de estructura del proyecto 6. El caso de uso finaliza.	

Exportar Modelo

Exportar Modelo	
Descripción	Permite la exportación del modelo de clases UML a otros formatos : JPG y XMI.
Actores	Modelador
Requerimientos Especiales	No existen requerimientos especiales para este caso de uso.
Precondición	El usuario debe haber ingresado al sistema con el perfil de Modelador.
Flujo Básico	
1. El caso de uso se inicia cuando el usuario del sistema elige la opción “Exportar Modelo”. 2. El sistema muestra un formulario con las opciones : “JPG” y “XMI” . 3. El usuario elige una opción de exportación. 4. El sistema genera como salida un archivo del tipo seleccionado por el usuario, en la ubicación de salida de archivos del sistema. 5. El caso de uso finaliza.	
Poscondición	La exportación del diagrama se realiza exitosamente.

Modulo Conversor

En la figura 8 se muestran los casos de uso del modulo Conversor, seguido de las especificaciones de los casos de uso principales.

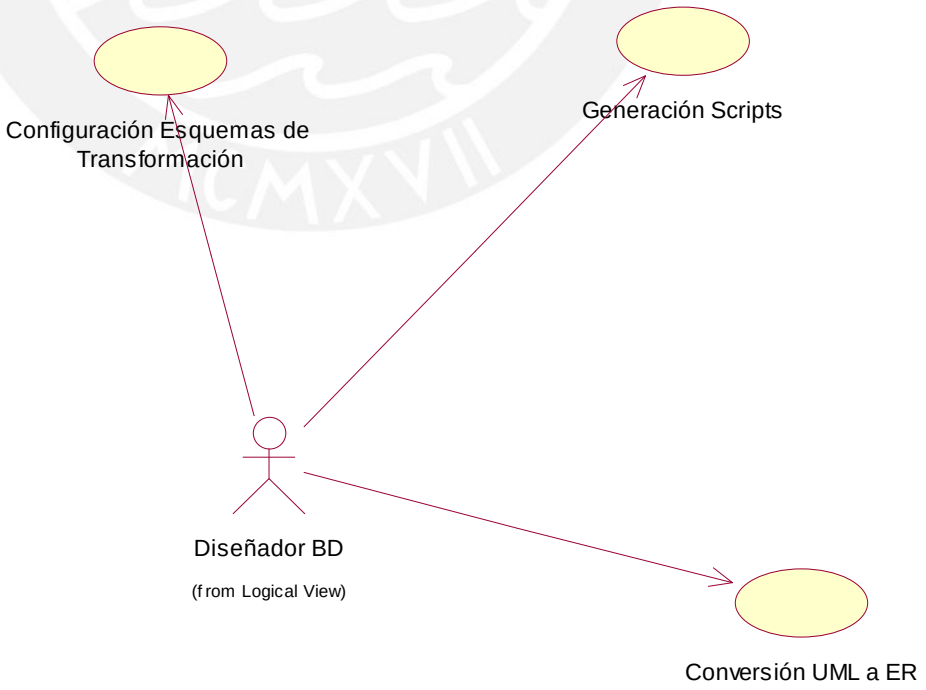


FIGURA 8: CASOS DE USO - CONVERSOR

Configuración Esquemas de Transformación

Configuración Esquemas de Transformación	
Descripción	Permite la selección de los algoritmos de transformación a ser usados en la conversión de uml a er.
Actores	Diseñador Base Datos
Requerimientos Especiales	No existen requerimientos especiales para este caso de uso.
Precondición	El usuario debe haber ingresado al sistema con el perfil de Diseñador BD.
Flujo Básico	
1. El caso de uso se inicia cuando el usuario del sistema elige la opción “Configurar Esquemas”. 2. El sistema muestra un formulario con el listado de los algoritmos de conversión para las relaciones de Asociación, Agregación y Herencia y las opciones “Aceptar” y “Cancelar”. 3. Si el usuario elige “Cancelar”, ir al paso 7. 4. El usuario elige una opción de conversión para cada tipo de relación UML. 5. El usuario elige la opción “Aceptar”. 6. El sistema guarda las opciones de conversión seleccionadas. 7. El caso de uso finaliza.	
Poscondición	Los esquemas de conversión fueron seleccionados exitosamente.

Conversión UML a ER

Conversión UML a ER	
Descripción	Permite la conversión de un esquema UML, definido en un archivo XMI, a un esquema Entidad Relación.
Actores	Diseñador Base Datos
Requerimientos Especiales	No existen requerimientos especiales.
Precondición	El usuario debe haber ingresado al sistema con el perfil de Diseñador BD.
Flujo Básico	
1. El caso de uso inicia cuando el usuario elige la opción “Convertir”. 2. El sistema muestra un formulario con un campo para el ingreso de la ruta del archivo xmi, el listado de los motores de base de datos soportados y las opciones “Aceptar” y “Cancelar”. 3. Si el usuario elige “Cancelar”, ir al paso 7. 4. El usuario elige el motor de base de datos.	

5. El usuario elige la opción “Aceptar”. 6. El sistema muestra en el formulario de <Diagrama> el esquema entidad relación para el motor de base de datos seleccionado en el paso 4. 7. El caso de uso finaliza.	
Poscondición	Se realizo la conversión en forma exitosa.

Generación de Scripts

Generación de Scripts	
Descripción	Permite la creación de los scripts de la base de datos definida en la etapa de conversión.
Actores	Diseñador Base Datos
Requerimientos Especiales	No existen requerimientos especiales.
Precondición	El usuario debe haber ingresado al sistema con el perfil de Diseñador BD. El caso de uso Conversión UML a ER debe haber sido realizado en forma exitosa.
Flujo Básico	
1. El caso de uso inicia cuando el usuario elige la opción “Generar Script”. 2. El sistema muestra en el formulario de <Script> los scripts generados. 3. El caso de uso finaliza.	
Poscondición	Se generaron los scripts en forma exitosa.

Modulo Generador

En la figura 9 se muestran los casos de uso del modulo Generador, seguido de las especificaciones de los casos de uso principales.

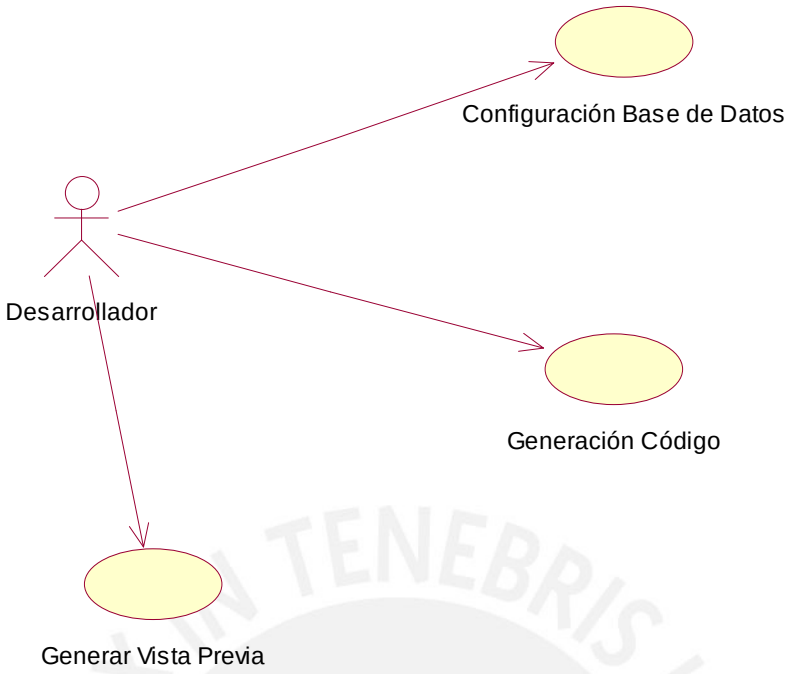


FIGURA 9: CASOS DE USO - GENERADOR

Configuración Base de Datos

Configuración Base de Datos	
Descripción	Permite la configuración de la base de datos para la cual se generara la aplicación.
Actores	Desarrollador
Requerimientos Especiales	No existen requerimientos especiales.
Precondición	El usuario debe haber ingresado al sistema con el rol de Desarrollador.
Flujo Básico	
1. El caso de uso se inicia cuando el usuario selecciona la opción “Configurar Base de Datos”. 2. El sistema muestra un formulario con los campos de conexión a base de datos: base de datos, nombre de base de datos, ubicación, usuario y clave y las opciones “Aceptar” y “Cancelar”. 3. Si el usuario elige “Cancelar”, ir al paso 8. 4. El usuario selecciona los parámetros requeridos. 5. El usuario elige la opción “Aceptar”. 6. El sistema guarda los parámetros de conexión ingresados. 7. El sistema muestra las tablas que conforman la base de datos seleccionada en el formulario de <Tablas> 8. El caso de uso finaliza.	

Poscondición	Se realizó la configuración en forma exitosa.

Generar Código

Generar Código	
Descripción	Genera todos los componentes de una aplicación Web J2EE.
Actores	Desarrollador
Requerimientos Especiales	No existen requerimientos especiales.
Precondición	El usuario debe haber ingresado al sistema con el perfil de Desarrollador. El caso de uso Configuración Base de Datos debe haber sido realizado en forma exitosa El usuario debe haber seleccionado previamente una tabla.
Flujo Básico	
1. El caso de uso inicia cuando el usuario elige la opción “Generar Código”. 2. El sistema muestra un formulario para el ingreso de la ruta donde se generarán los archivos de la aplicación y las opciones “Aceptar” y “Cancelar”. 3. Si el usuario elige la opción “Cancelar”. Ir al paso 5. 4. El usuario selecciona la ruta donde se generarán los archivos de código para la tabla seleccionada. 5. El usuario elige la opción “Aceptar”. 6. El sistema genera los archivos de código para la tabla seleccionada. 7. El caso de uso finaliza.	
Poscondición	Se generó la aplicación en forma exitosa.

2.1.1.2 Diagrama de Paquetes

A continuación se presentan los principales paquetes que conforman la herramienta:

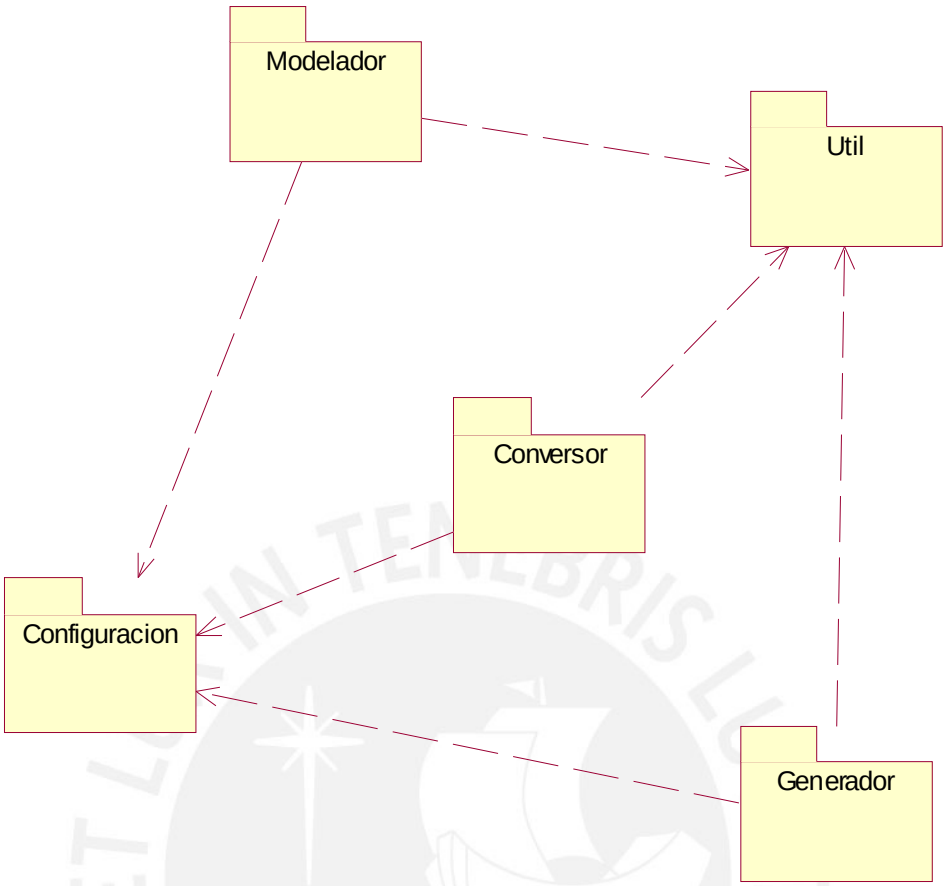


FIGURA 10: DIAGRAMA DE PAQUETES

- **Paquete Modelador**

  Contiene todas las clases necesarias para el modelado de clases y exportación en formato XMI.

- **Paquete Conversor**

  Contiene todas las clases requeridas para la aplicación de los algoritmos de conversión de clases uml a un modelo relacional.

- **Paquete Generador**

  Contiene todas las clases relacionadas a la generación de los archivos que conforman una aplicación java web.

- **Paquete Util**

  Contiene las clases utilitarias usadas por los demás módulos.

- **Paquete Configuración**

  Contiene las clases necesarias para la lectura y escritura de archivos de configuración necesarios para el funcionamiento de la herramienta.

3. Diseño

En este capítulo se definen las decisiones de arquitectura de la aplicación, el diseño gráfico de la interfaz de usuario, y el diseño de las funcionalidades principales de los módulos.

3.1 Arquitectura de la solución

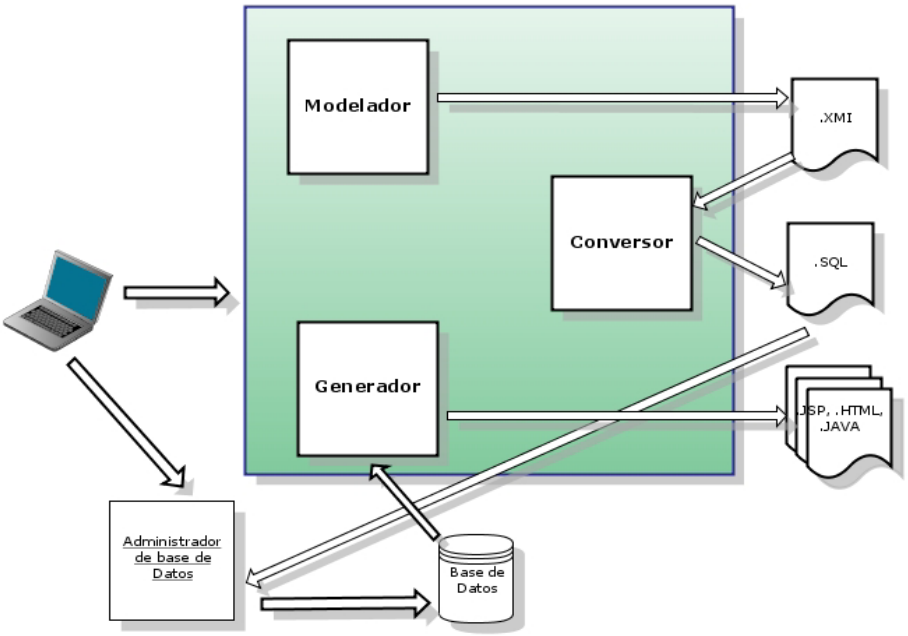


FIGURA 11: ARQUITECTURA DE LA SOLUCIÓN

La arquitectura de la solución está representada por una serie de módulos y componentes como se puede apreciar en la figura 11, a continuación se detallan los componentes que conforman la arquitectura:

- **Modelador**

Permite realizar el diagramado de clases de la aplicación a generar usando el lenguaje de modelado UML.

- **Conversor**

Permite la conversión del modelo definido en UML, en formato XMI, a un modelo relacional usando la notación IDEF1X.

- **Generador**

Permite generar la aplicación Web a partir de la base de datos física generada desde los scripts resultantes del módulo conversor.

- ***Archivo XMI***

Representación XML del modelo UML definido por el modulo modelador.

- ***Script* Base *de Datos***

Archivos de scripts de base de datos generados por el módulo conversor.

- **Aplicación *Web J2EE***

Aplicación Java Web generada por el módulo generador a partir de las tablas creadas por el módulo conversor.

- ***Administrador de Base de Datos***

Administrador de base de datos que permite la creación de la base de datos física a través de la ejecución de los scripts generados por el modulo conversor.

- ***Base de* Datos**

Base de datos resultante de la ejecución de los scripts generados por el módulo Conversor.

3.1.1 Objetivos y limitaciones arquitectónicas

A continuación se listan los requerimientos clave y limitaciones para la herramienta que tiene un impacto significativo en la arquitectura:

- La herramienta será implementada como un sistema *stand-alone*.

- La comunicación entre los distintos módulos de la herramienta se da a través de archivos XML ubicados en la PC donde se ejecute la herramienta.

- La herramienta requiere un sistema operativo Windows 2000 en adelante y/o Linux en la PC donde se instale la herramienta.

- El usuario requerirá contar con permisos de lectura/escritura de archivos en las carpetas a ser usadas por la herramienta.

- Se usará la máquina virtual de Java (JVM) versión 5.0. en adelante.

- La herramienta soportará los administradores de base de datos: SQLite.

- El tiempo requerido para la generación de los archivos XML (Módulo Modelador) y código (Módulo Conversor y Generador) están supeditados al tamaño del modelo desarrollado por el usuario.

- La herramienta requiere de espacio no más de 50 MB en espacio de disco y memoria de 128 MB RAM.

3.1.2 Diagrama de Clases de Diseño

A continuación en la figura 12, 13 y 14 se presentan los diagramas de clase de los módulos que conforman la herramienta y la especificación de las clases principales:

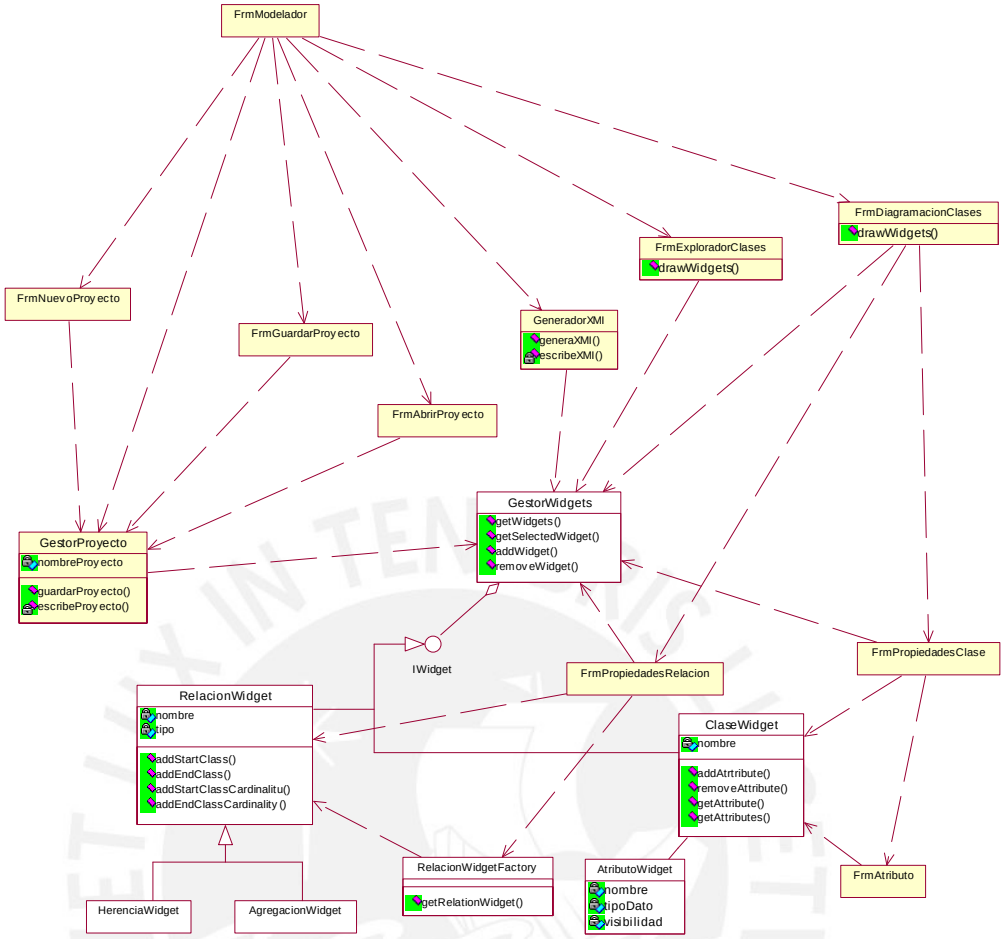


FIGURA 12: DIAGRAMA DE CLASES - MODELADOR

Nombre de la Clase	**GeneradorXMI**
Descripción	Permite la generación de un archivo xmi a partir del diagrama de clases modelado.
Nombre del Atributo	Descripción
Nombre del Operación	Descripción
GeneraXMI	Genere un archivo XMI

Nombre de la Clase	**IWidget**
Descripción	Representación genérica de un elemento UML(Clase o Relación).
Nombre del Atributo	Descripción

Nombre de la Clase	**ClaseWidget**
Descripción	Representación gráfica de una clase UML
Nombre del Atributo	Descripción
Nombre	Nombre de la clase
Nombre del Operación	Descripción

addAttribute	Agrega un atributo de la clase
removeAttribute	Quita un atributo de la clase
getAttributes	Obtiene la lista de atributos de la clase
getAttribute	Obtiene un atributo en especifico

Nombre de la Clase	**AtributoWidget**
Descripción	Representación gráfica de un atributo de una clase UML
Nombre del Atributo	Descripción
Nombre	Nombre del atributo
TipoDato	Tipo de Dato (Integer, String, Boolean, etc..)
Visibilidad	Visibilidad (Private, Public, Protected)

Nombre de la Clase	**RelacionWidget**
Descripción	Representación gráfica de una relación entre dos clases UML
Nombre del Atributo	Descripción
Nombre	Nombre del atributo
Tipo	Identificador del tipo de relación(Asociación, Agregación, Herencia)
Nombre del Operación	Descripción
addStartClass	Agrega clase origen
addEndClass	Agrega clase destino
addStartClassCardinality	Define la cardinalidad de la clase origen
addEndClassCardinality	Define la cardinalidad de la clase destino

Nombre de la Clase	**HerenciaWidget**
Descripción	Representación gráfica de la relación tipo Herencia.
Nombre del Atributo	Descripción

Nombre de la Clase	**AgregacionWidget**
Descripción	Representación gráfica de la relación tipo Agregación.
Nombre del Atributo	Descripción

Nombre de la Clase	**RelacionWidgetFactory**
Descripción	Permite la creación de los tipos de relaciones definidos.
Nombre del Atributo	Descripción
Nombre del Operación	Descripción
getRelacionWidget	Obtiene el tipo de relación solicitado

Nombre de la Clase	**GestorWidgets**
Descripción	Permite la gestión(agregar, editar, eliminar) de objetos tipo widget.
Nombre del Atributo	Descripción
Nombre del Operación	Descripción
addWidget	Agrega un widget
removeWidget	Quitar un widget
getWidgets	Obtiene todos los widgets agregados

getSelectedWidget	Obtiene el widget que ha sido seleccionado***

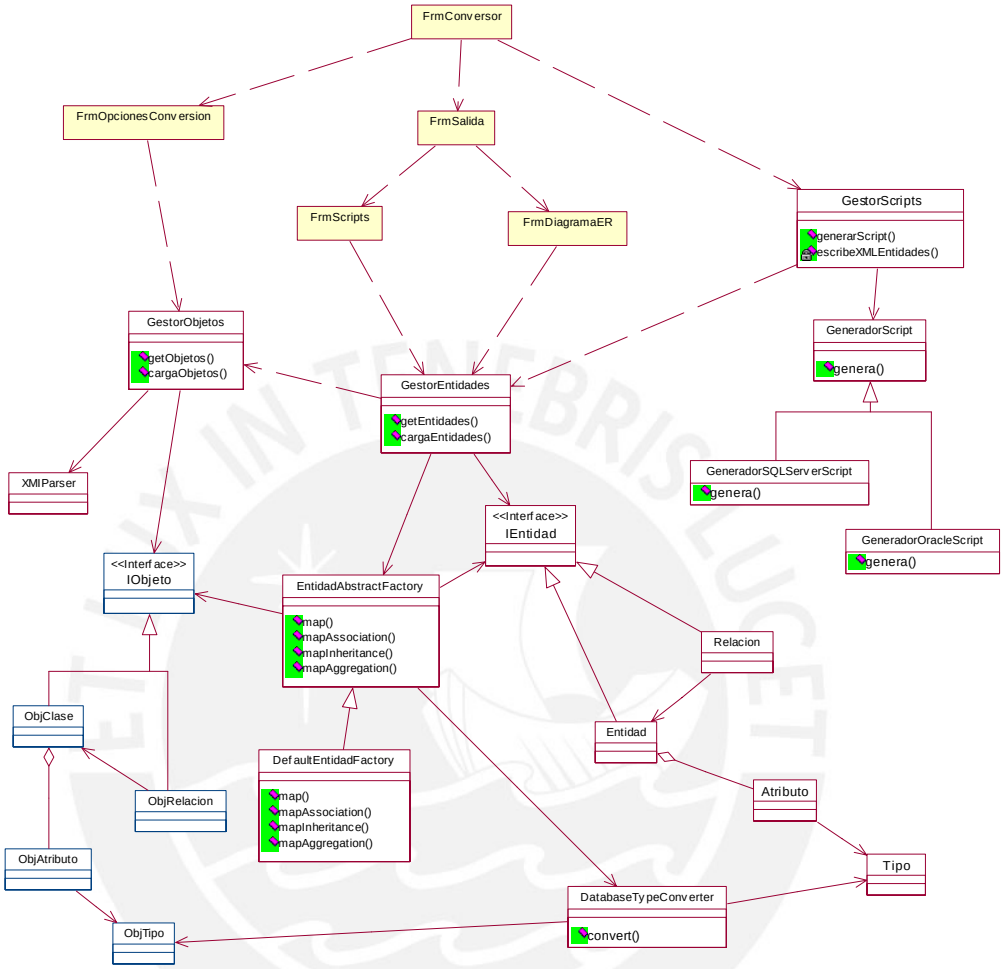


FIGURA 13: DIAGRAMA DE CLASES - CONVERSOR

Nombre de la Clase	**EntidadAbstractFactory**
Descripción	Representación genérica de la factoría para el mapeo de objetos a entidades de un modelo relacional.
Nombre del Atributo	Descripción
Nombre del Operación	Descripción
Map	Realiza el mapeo de objetos a entidades de un modelo relacional.
MapAssociation	Realiza el mapeo de una relación de asociación
MapInheritance	Realiza el mapeo de una relación de herencia
MapAggregation	Realiza el mapeo de una relación de agregación

Nombre de la Clase	**DefaultAbstractFactory**
Descripción	Representación por defecto de la factoría para el mapeo de objetos a entidades de un modelo relacional.
Nombre del Atributo	Descripción

Nombre del Operación	Descripción
Map	Realiza el mapeo de objetos a entidades de un modelo relacional.
MapAssociation	Realiza el mapeo de una relación de asociación
MapInheritance	Realiza el mapeo de una relación de herencia
mapAggregation	Realiza el mapeo de una relación de agregacion

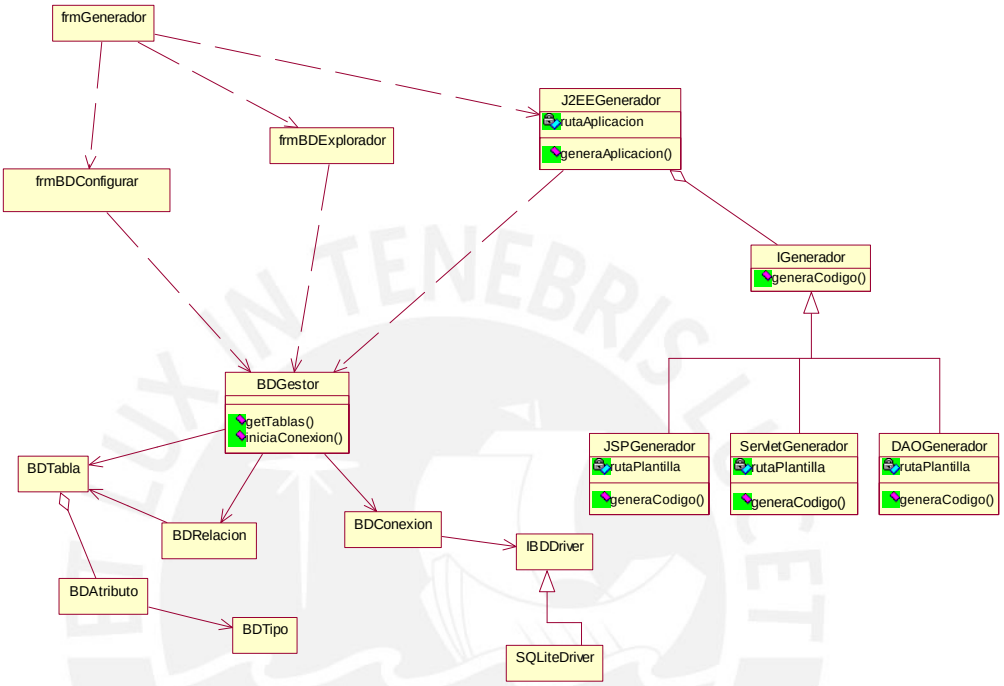


FIGURA 14: DIAGRAMA DE CLASES - GENERADOR

Nombre de la Clase	**J2EEGenerador**
Descripción	Permite generar una aplicación para el lenguaje Java y Arquitectura J2EE.
Nombre del Atributo	Descripción
rutaAplicacion	Ruta donde se generaran los archivos de la aplicación
Nombre del Operación	Descripción
generaAplicacion	Genera la aplicación para el lenguaje y arquitectura seleccionada.

Nombre de la Clase	**IGenerador**
Descripción	
Nombre del Atributo	Descripción
Nombre del Operación	Descripción
generaCodigo	Permite generar código fuente en base a una plantilla.

Nombre de la Clase	**JSPGenerador**
Descripción	
Nombre del Atributo	Descripción
rutaPlantilla	Ruta de plantilla para la capa de presentación.

Nombre del Operación	Descripción
generaCodigo	Permite generar código fuente en base a una plantilla para la capa de Presentación.

Nombre de la Clase	**ServletGenerador**
Descripción	
Nombre del Atributo	Descripción
rutaPlantilla	Ruta de plantilla para la capa de Negocios.
Nombre del Operación	Descripción
generaCodigo	Permite generar código fuente en base a una plantilla para la capa de Negocios.

Nombre de la Clase	**DAOGenerador**
Descripción	
Nombre del Atributo	Descripción
rutaPlantilla	Ruta de plantilla para la capa de Datos.
Nombre del Operación	Descripción
generaCodigo	Permite generar código fuente en base a una plantilla para la capa de Datos.

3.1.3 Diagramas de Secuencia

A continuación se presentan los diagramas de secuencia principales de los módulos que conforman la herramienta:

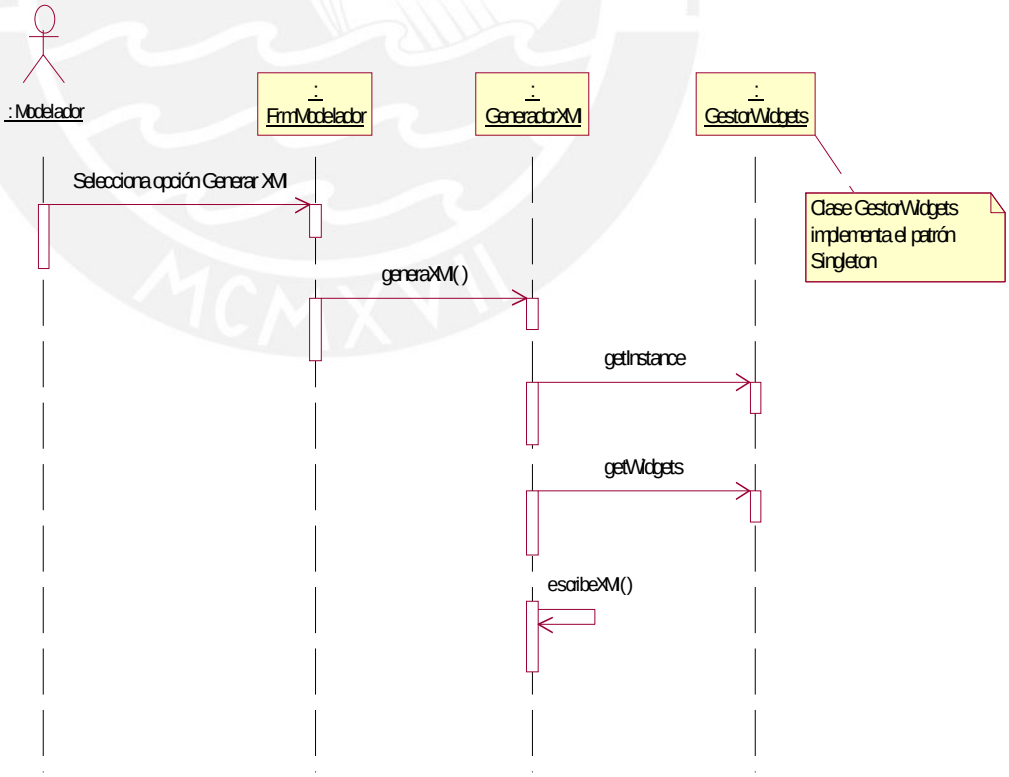


FIGURA 15: EXPORTAR XMI

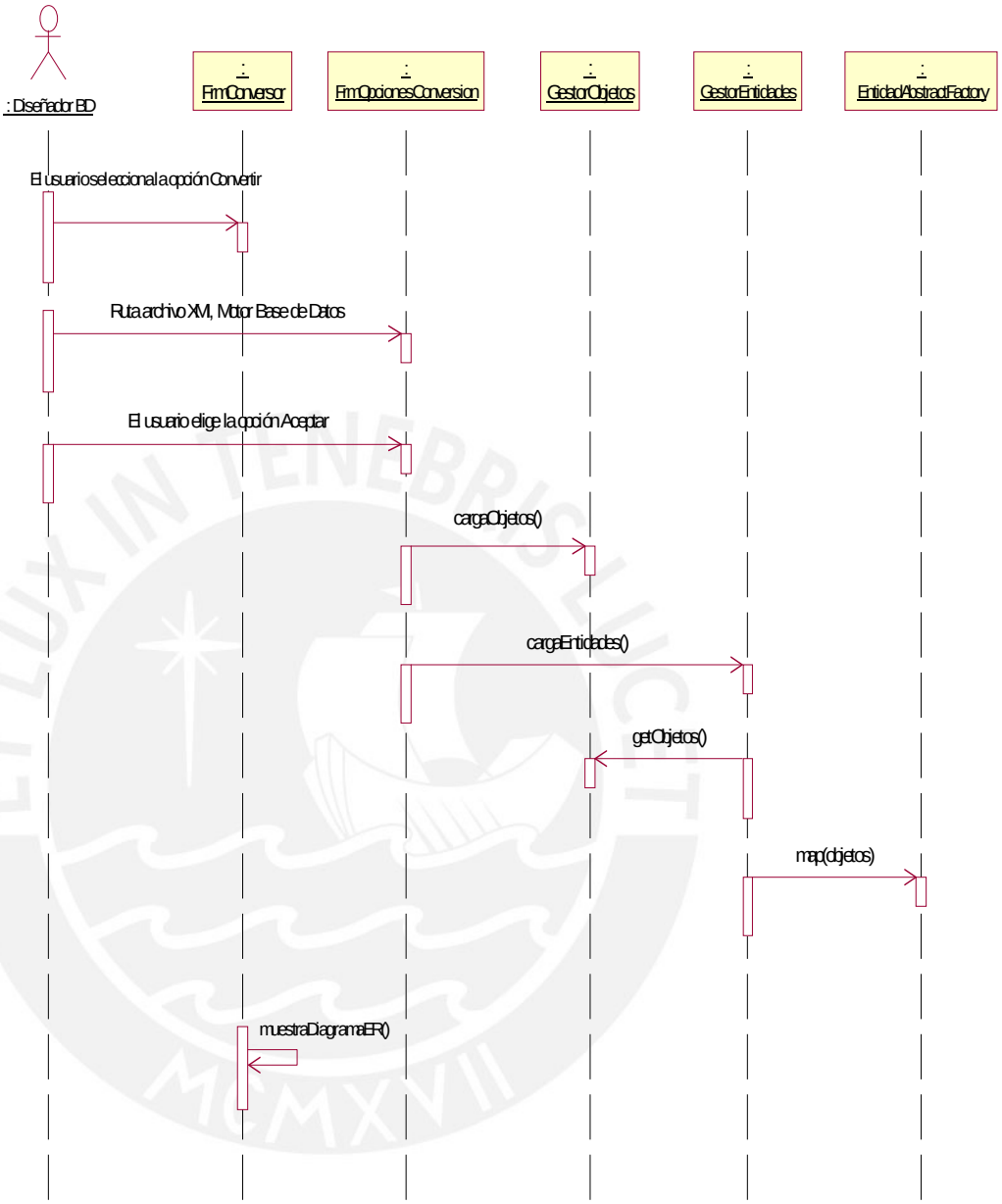


FIGURA 16: CONVERTIR UML A ER

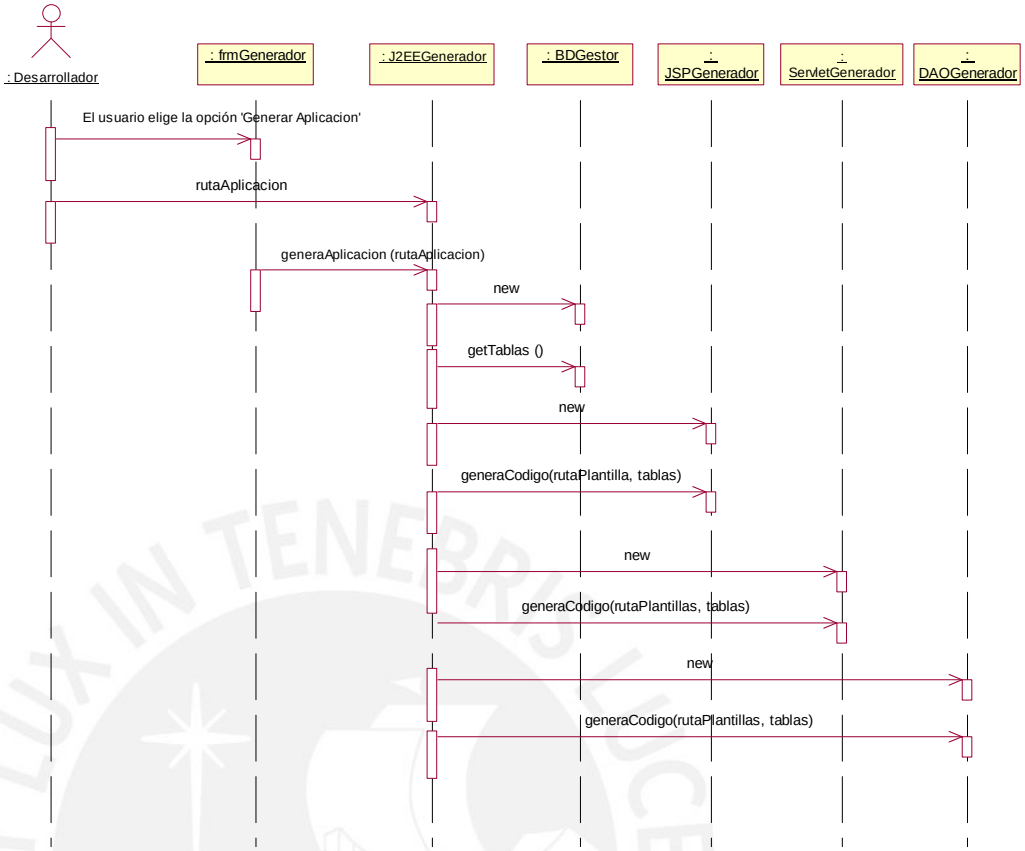
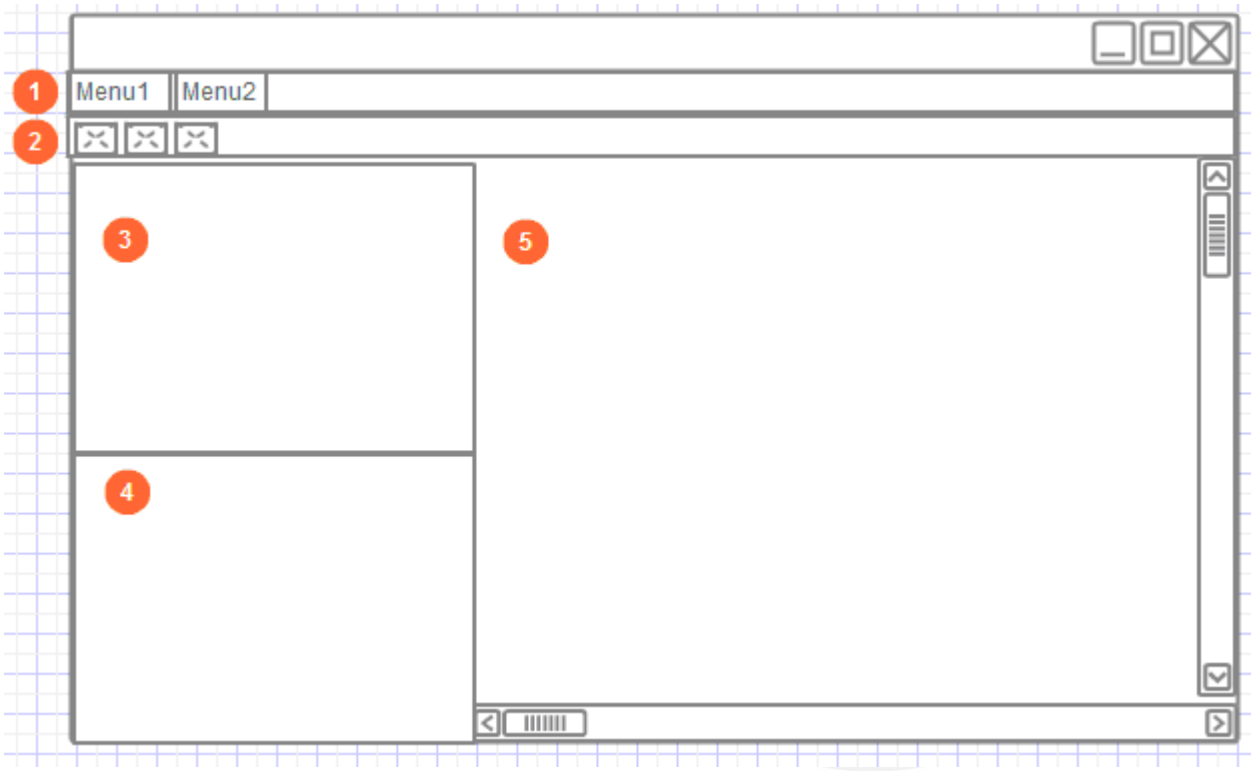


FIGURA 17: GENERAR APLICACION

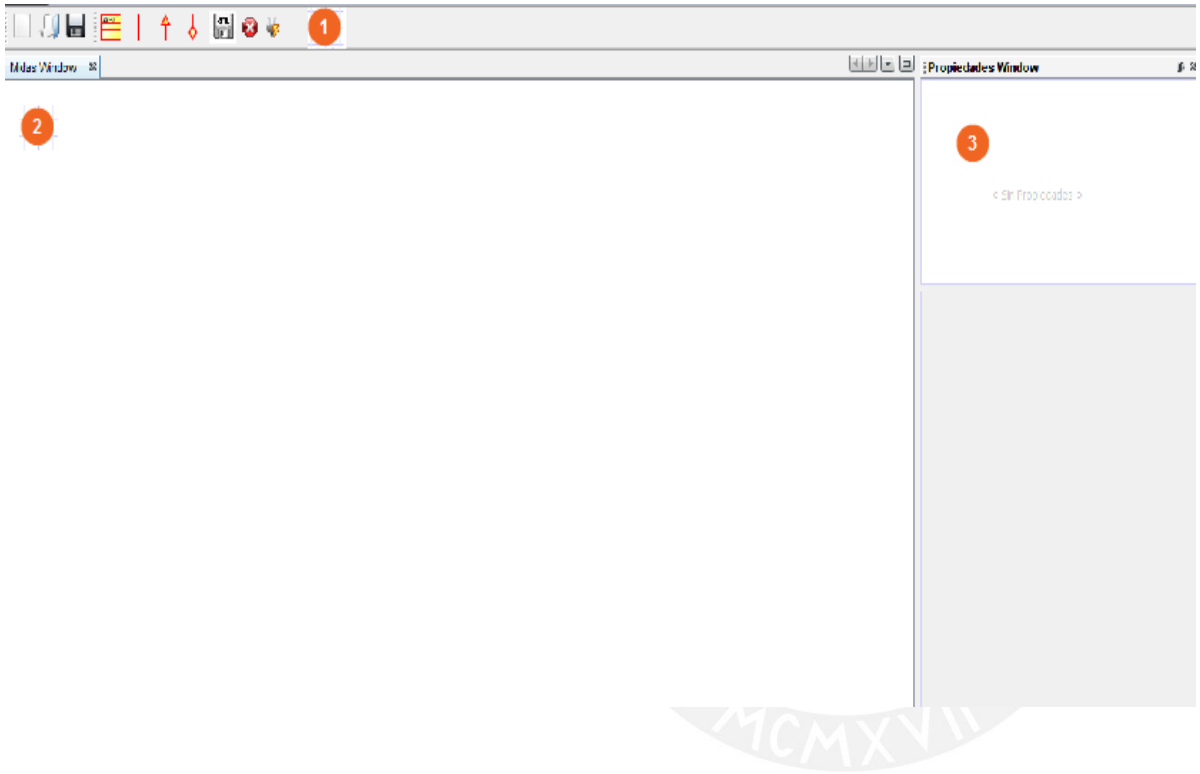
3.2 Diseño de la interfaz gráfica

La interfaz gráfica para los tres módulos que conforman la aplicación ha sido dividida en secciones tal como se muestra en la figura 18, luego en las figuras 19, 20 y 21 se muestran los diseños de las interfaces de cada uno de los módulos respectivamente.



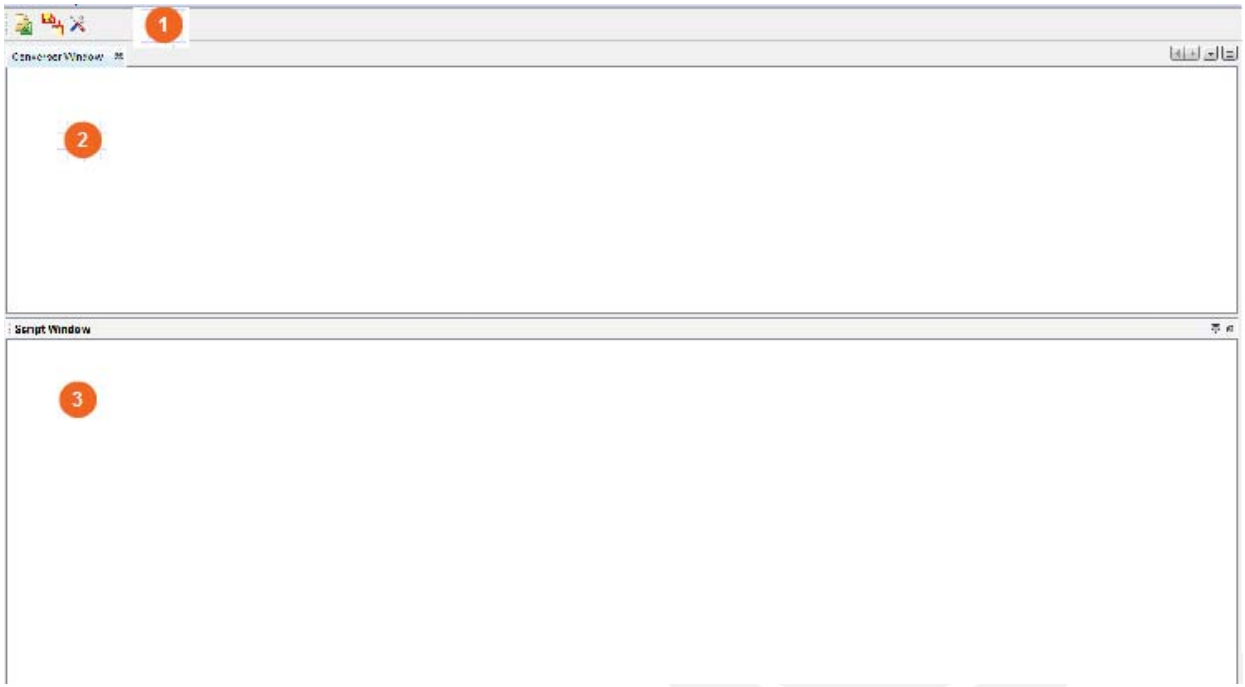
Elemento	Descripción
1	Menú principal.
2	Barra de herramientas.
3	Árbol de navegación de los elementos del módulo : clases, tablas, etc.
4	Propiedades de los elementos del módulo : clases, tablas, etc.
5	Área de Resultados de los procesos del módulo : scripts, diagramas, etc.

FIGURA 18: PLANTILLA DE INTERFAZ DE USUARIO DE UN MÓDULO



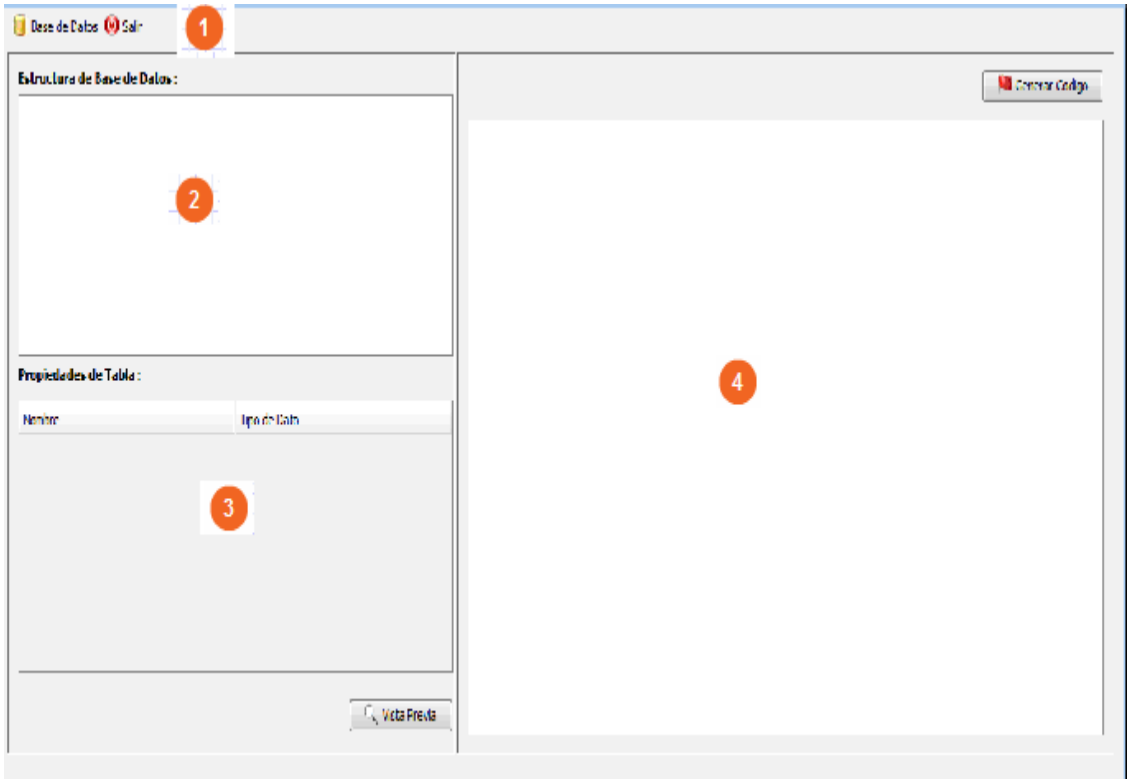
Elemento	Descripción
1	Barra de herramientas.
2	Área de modelado de clases.
3	Ventana de propiedades de clases.

FIGURA 19: INTERFAZ DE USUARIO MODELADOR



Elemento	Descripción
1	Barra de herramientas.
2	Área de modelado de tablas.
3	Ventana de generación de scripts.

FIGURA 20: INTERFAZ DE USUARIO CONVERSOR



Elemento	Descripción
1	Barra de herramientas.
2	Árbol de tablas de base de datos.
3	Ventana de propiedades de tablas.
4	Ventana de vista previa.

FIGURA 21: INTERFAZ DE USUARIO GENERADOR

3.3 Diseño de Archivo XMI

El archivo xmi generado por el módulo modelador se basa en el DTD(*data definition file*) **[20]** de XMI definido por el omg para uml. En el cuadro 5 se muestran las definiciones de los principales elementos usados en la generacion de xmi por la herramienta.

Clase	<!-- _____ --> <!-- --> <!-- CLASS: Foundation.Core.Class --> <!-- _____ --> <!ELEMENT UML:Class.isActive EMPTY > <!ATTLIST UML:Class.isActive xmi.value (false | true) #REQUIRED > <!ELEMENT UML:Class (UML:ModelElement.name | UML:ModelElement.visibility | UML:GeneralizableElement.isRoot | UML:GeneralizableElement.isLeaf | UML:GeneralizableElement.isAbstract | UML:Class.isActive | XMI.extension | UML:ModelElement.binding | UML:ModelElement.template | UML:ModelElement.templateParameter | UML:ModelElement.implementation | UML:ModelElement.view | UML:ModelElement.presentation | UML:ModelElement.namespace | UML:ModelElement.constraint | UML:ModelElement.requirement | UML:ModelElement.provision | UML:ModelElement.stereotype | UML:ModelElement.elementReference | UML:ModelElement.collaboration | UML:ModelElement.behavior | UML:ModelElement.partition | UML:GeneralizableElement.generalization | UML:GeneralizableElement.specialization | UML:Classifier.parameter | UML:Classifier.structuralFeature | UML:Classifier.specification | UML:Classifier.realization | UML:Classifier.associationEnd | UML:Classifier.participant | UML:Classifier.createAction | UML:Classifier.instance | UML:Classifier.collaboration | UML:Classifier.classifierRole | UML:Classifier.classifierInState | UML:ModelElement.taggedValue | UML:Namespace.ownedElement | UML:Classifier.feature)* > <!ATTLIST UML:Class name CDATA #IMPLIED visibility (public | protected | private) #IMPLIED isRoot (false | true) #IMPLIED isLeaf (false | true) #IMPLIED isAbstract (false | true) #IMPLIED isActive (false | true) #IMPLIED binding IDREFS #IMPLIED template IDREFS #IMPLIED templateParameter IDREFS #IMPLIED implementation IDREFS #IMPLIED view IDREFS #IMPLIED

	presentation IDREFS #IMPLIED namespace IDREFS #IMPLIED constraint IDREFS #IMPLIED requirement IDREFS #IMPLIED provision IDREFS #IMPLIED stereotype IDREFS #IMPLIED elementReference IDREFS #IMPLIED ModelElement.collaboration IDREFS #IMPLIED behavior IDREFS #IMPLIED partition IDREFS #IMPLIED generalization IDREFS #IMPLIED specialization IDREFS #IMPLIED parameter IDREFS #IMPLIED structuralFeature IDREFS #IMPLIED specification IDREFS #IMPLIED realization IDREFS #IMPLIED associationEnd IDREFS #IMPLIED participant IDREFS #IMPLIED createAction IDREFS #IMPLIED instance IDREFS #IMPLIED Classifier.collaboration IDREFS #IMPLIED classifierRole IDREFS #IMPLIED classifierInState IDREFS #IMPLIED %XMI.element.att; %XMI.link.att; >
Relación de Asociación	<!ELEMENT UML:Association (UML:ModelElement.name | UML:ModelElement.visibility | UML:GeneralizableElement.isRoot | UML:GeneralizableElement.isLeaf | UML:GeneralizableElement.isAbstract | XMI.extension | UML:ModelElement.binding | UML:ModelElement.template | UML:ModelElement.templateParameter | UML:ModelElement.implementation | UML:ModelElement.view | UML:ModelElement.presentation | UML:ModelElement.namespace | UML:ModelElement.constraint | UML:ModelElement.requirement | UML:ModelElement.provision | UML:ModelElement.stereotype | UML:ModelElement.elementReference | UML:ModelElement.collaboration | UML:ModelElement.behavior | UML:ModelElement.partition | UML:GeneralizableElement.generalization | UML:GeneralizableElement.specialization | UML:Association.link | UML:Association.associationEnd | UML:ModelElement.taggedValue | UML:Namespace.ownedElement | UML:Association.connection)* > <!ATTLIST UML:Association name CDATA #IMPLIED visibility (public | protected | private) #IMPLIED isRoot (false | true) #IMPLIED isLeaf (false | true) #IMPLIED isAbstract (false | true) #IMPLIED binding IDREFS #IMPLIED template IDREFS #IMPLIED templateParameter IDREFS #IMPLIED implementation IDREFS #IMPLIED view IDREFS #IMPLIED presentation IDREFS #IMPLIED namespace IDREFS #IMPLIED constraint IDREFS #IMPLIED requirement IDREFS #IMPLIED provision IDREFS #IMPLIED stereotype IDREFS #IMPLIED elementReference IDREFS #IMPLIED collaboration IDREFS #IMPLIED behavior IDREFS #IMPLIED

	partition IDREFS #IMPLIED generalization IDREFS #IMPLIED specialization IDREFS #IMPLIED link IDREFS #IMPLIED associationEnd IDREFS #IMPLIED %XMI.element.att; %XMI.link.att; >

CUADRO 5: UML XMI DATA DEFINITION FILE

3.4 Estructura de archivos

En la figura 22 se muestra la estructura de archivos de la aplicación la cual consiste en todos los archivos generados y usados por la herramienta.

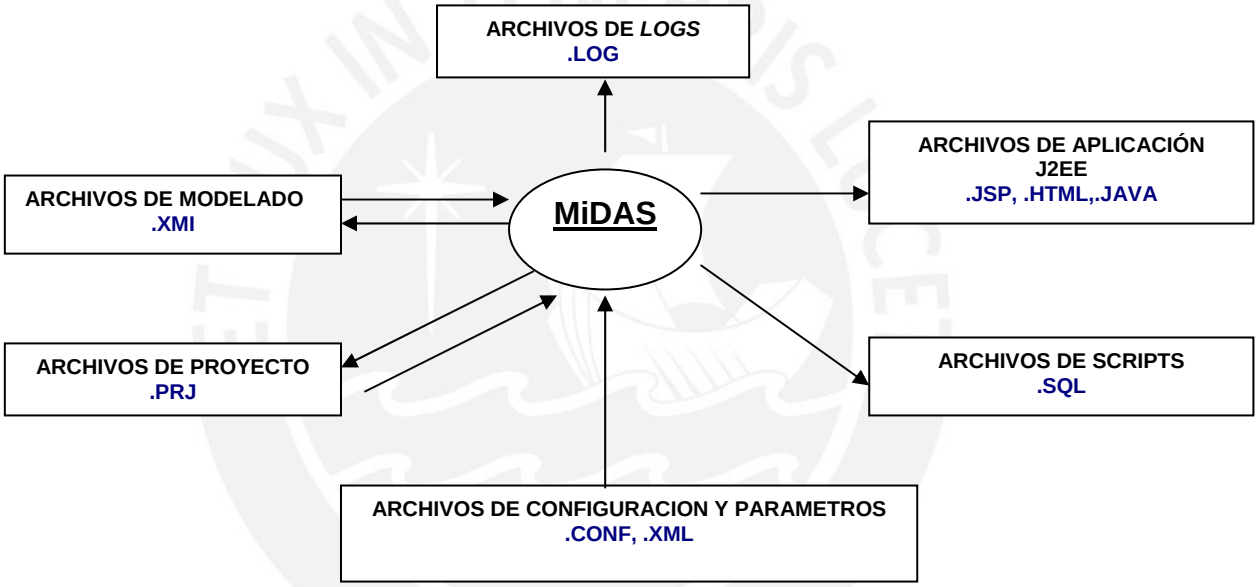


FIGURA 22: ESTRUCTURA DE ARCHIVOS

Archivo	Descripción
.XMI	Representación XML de un diagrama de clases UML en base al estándar XMI resultantes del módulo Modelador.
.JSP, .HTML, .JAVA	Archivos de código fuente de la aplicación Java resultante del módulo Generador.
.SQL	Archivos de *Script* de creación de tablas resultantes del módulo Conversor.
.LOG	Archivo de Logs generados por la herramienta Log4J.
.CONF,.XML	Archivo XML de definición de parámetros de configuración de la herramienta.
.PRJ	Archivo XML de un proyecto de la herramienta

4. Construcción

El presente capitulo define la estructura del proyecto, implementación, tecnología, *frameworks*, patrones, componentes y el plan de pruebas que se utilizarán en la elaboración de la herramienta.

4.1 Estructura de Proyecto

La estructura del proyecto es la siguiente:

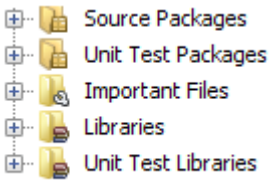


FIGURA 23: ESTRUCTURA DEL PROYECTO

Source Packages	Conjunto de archivos que constituyen el código fuente del proyecto.
Unit Test Packages	Conjunto de archivos que constituyen las pruebas unitarias del proyecto.
Important Files	Archivos de configuración del proyecto.
Libraries	Librerías necesarias para la ejecución del proyecto.
Unit Test Libraries	Librerías usadas para la ejecución de pruebas unitarias del proyecto.

4.2 Tecnologías y frameworks

Un *framework* es aquella estructura de trabajo que está diseñada con el fin de facilitar el desarrollo de software. En el presente proyecto se hará uso de distintos *frameworks* y tecnologías para su desarrollo.

4.2.1 Lenguaje de programación Java

Se usará el lenguaje de programación Java. La elección de este lenguaje se basó principalmente en que el proyecto está orientado hacia la plataforma Java y además las ventajas que ofrece el lenguaje como son:

- Independencia de la plataforma.
- Lenguaje orientado a objetos.
- Buena documentación.
- Soporte por la comunidad.

4.2.2 Netbeans IDE

Es un entorno de desarrollo para la creación de aplicaciones multilenguaje. Su uso en el proyecto es principalmente como entorno para la programación **[14]** en el lenguaje Java y el uso de sus librerías base. El criterio principal para su elección fue el uso de la librería visual library que permite el modelado de clases y tablas.

4.2.3 Netbeans IDE Module Suite

Permite la creación rápida de módulos o aplicaciones *standalone* que usan como base las librerías y componentes de interfaz gráfica propios del netbeans tales como: menús, ventanas de propiedades, barra de herramientas, etc. Se escogió esta librería por la facilidad que aporta en la generación de interfaces gráficas. En la figura 24 se muestra el modulo conversor creado en base al *module suite*.

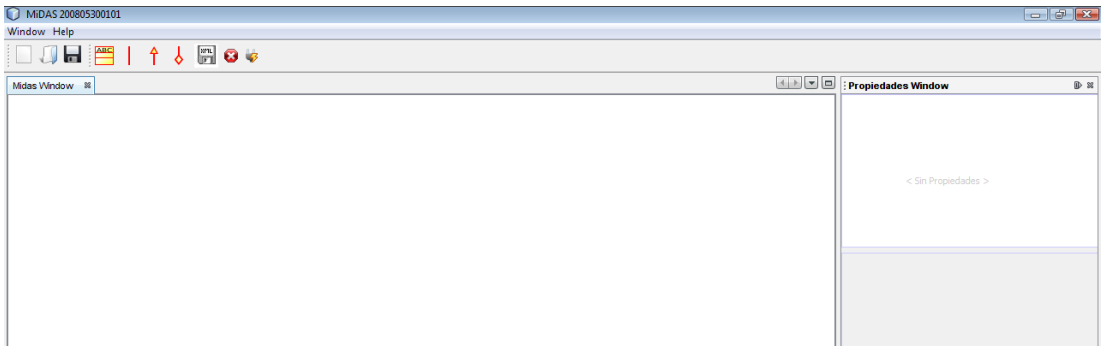


FIGURA 24: USO DEL MODULE SUITE PARA EL MODULO CONVERSOR.

4.2.4 Netbeans Visual Library

Es una librería **[13]** cuyo propósito principal en el proyecto es dar soporte al modelamiento orientado a gráficos, a través de una gama de componentes y elementos visuales. Se usará en el proyecto para el diagramado de las clases y tablas en sus módulos respectivos. El criterio principal para su elección fue la facilidad de creación de elementos visuales y de relaciones entre estos. En la figura 25 se muestra el uso de la librería *visual library* para el modelado de clases del modulo modelador.

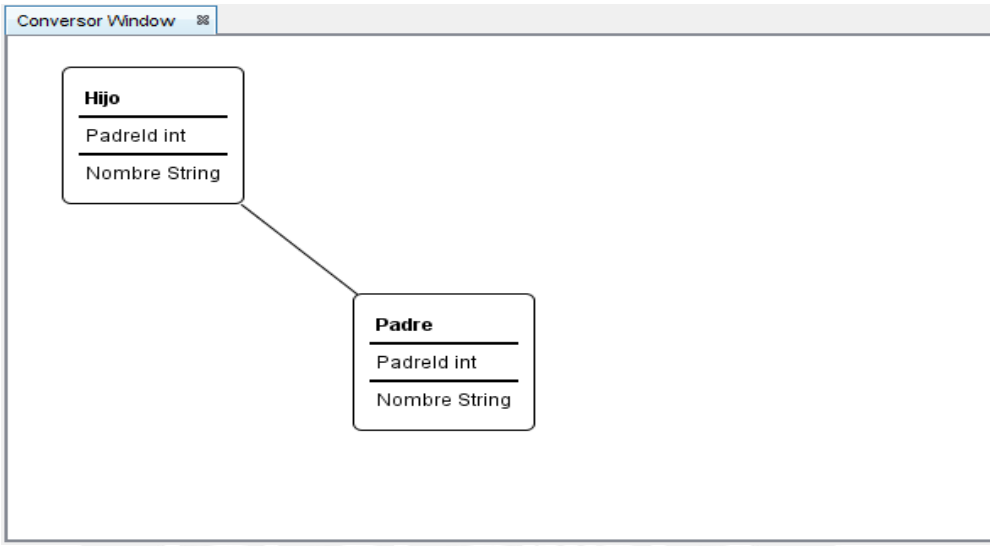


FIGURA 25: USO DE VISUAL LIBRARY EN EL MODULO CONVERSOR.

4.2.5 Eclipse UML2

Es una librería que contiene un conjunto de clases que representan el metamodelo de UML. En el proyecto su uso principal es el de almacenamiento del modelo de clases y para la persistencia de este en archivos xmi. La razón de su elección fue el soporte de la versión 2.0 de UML. En la figura 26 se muestra un archivo xmi generado por el módulo modelador a partir de un diagrama de clases.

```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmlns:uml="http://www.eclipse.org/uml2/2.1.0/UML"
xmi:id="_wXvoMFNsEd60yojya80JhQ" name="DefaultModel">
  <packagedElement xmi:type="uml:PrimitiveType" xmi:id="_wXvoMVNsEd60yojya80JhQ" name="Integer"/>
  <packagedElement xmi:type="uml:PrimitiveType" xmi:id="_wXvoMlNsEd60yojya80JhQ" name="String"/>
  <packagedElement xmi:type="uml:PrimitiveType" xmi:id="_wXvoM1NsEd60yojya80JhQ" name="Char"/>
  <packagedElement xmi:type="uml:PrimitiveType" xmi:id="_wXvoNFNsEd60yojya80JhQ" name="Double"/>
  <packagedElement xmi:type="uml:Class" xmi:id="_wXvoNVNsEd60yojya80JhQ" name="Padre">
    <ownedAttribute xmi:id="_wXvoNlNsEd60yojya80JhQ" name="Nombre" type="_wXvoMlNsEd60yojya80JhQ">
      <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_wXvoN1NsEd60yojya80JhQ" value="10"/>
      <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_wXvoOFNsEd60yojya80JhQ"/>
    </ownedAttribute>
  </packagedElement>
  <packagedElement xmi:type="uml:Class" xmi:id="_wXvoOVNsEd60yojya80JhQ" name="Hijo">
    <generalization xmi:id="_wXvoOlNsEd60yojya80JhQ" general="_wXvoNVNsEd60yojya80JhQ"/>
    <ownedAttribute xmi:id="_wXvoO1NsEd60yojya80JhQ" name="Nombre" type="_wXvoMlNsEd60yojya80JhQ">
      <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="_wXvoPFNsEd60yojya80JhQ" value="10"/>
      <lowerValue xmi:type="uml:LiteralInteger" xmi:id="_wXvoPVNsEd60yojya80JhQ"/>
    </ownedAttribute>
  </packagedElement>
</uml:Model>
```

FIGURA 26: ARCHIVO XMI GENERADO MEDIANTE ECLIPSE UML2.

4.2.6 Velocity

Velocity **[12]** es un motor de plantillas hecho para Java por Apache. Se usará en el proyecto para la generación de *scripts* de creación de base de datos y código fuente (html, jsp, java, etc.). El criterio principal de su elección fue la facilidad que brinda su lenguaje de plantillas para la generación de archivos. En la figura 27 se muestra la plantilla de generación usada por el modulo generador para generar una clase entidad.

```
## Plantilla    :     Generacion de Entidad
## Autor        :     Carlos Balbuena

## Objetos    :
##    tabla                          :    objeto tipo tabla


public class B$nombre
{
#foreach( $atributo in $atributos )
    private $atributo.tipoDato $atributo.nombre;
#end

#foreach( $atributo in $atributos )
    public $atributo.tipoDato get$atributo.nombre ()
    {
        return $atributo.nombre;
    }

    public void set$atributo.nombre ( $atributo.tipoDato $atributo.nombre )
    {
        this.$atributo.nombre = $atributo.nombre;
    }
#end

}
```

FIGURA 27: PLANTILLA VELOCITY PARA LA GENERACIÓN DE CÓDIGO

4.2.7 Log4J

Es una librería **[11]** que permite el manejo de diferentes tipos de *logs* (Error, Información, Depuración), ejecutar filtros por categoría, redireccionar a diferentes destinos de salida(Archivo, Consola, Base de datos), diferentes formatos de visualización y configuración por ficheros. Se usará en el proyecto para la creación de archivos de *logs* de depuración en la etapa de desarrollo y para el control de errores/excepciones que ocurran una vez finalizado el producto.

```
<?xml version="1.0" encoding="UTF-8"?>
<log4j:configuration>
    <appender name="appender" class="org.apache.log4j.FileAppender">
        <param name="File" value="log.txt"/>
        <param name="Append" value="false"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d [%t] %p - %m%n"/>
        </layout>
    </appender>
    <root>
        <priority value="debug"/>
        <appender-ref ref="appender"/>
    </root>
</log4j:configuration>
```

FIGURA 28: ARCHIVO DE CONFIGURACION DE LOG4J

4.2.8 Spring Framework

Spring **[10]** es un *framework* que permite hacer un diseño por medio de interfaces de una manera sencilla, implementa patrones de diseño, posee una documentación extensa, se comunica fácilmente con otros frameworks como lo son: Struts, Hibernate, iBatis, Tapestry,

entre otros. Se usará Spring en el presente proyecto para la configuración de los distintos componentes que requieran ser obtenidos dinámicamente en tiempo de ejecución. El criterio principal para su elección fue la robustez y facilidad de integración que brinda este *framework*. En la figura 29 se muestra la configuración básica de dos objetos en el archivo de configuración de spring.

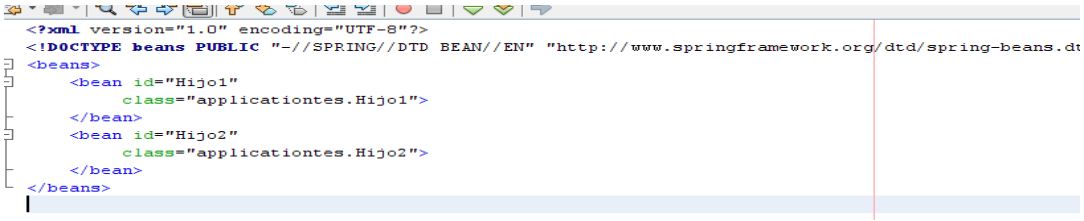


FIGURA 29: ARCHIVO DE CONFIGURACION DE SPRING

4.3 Pruebas

La ejecución de pruebas tiene como propósito principal validar que el software funcione correctamente y cumpla con los objetivos para el cual fue desarrollado.

4.3.1 Plan de pruebas

El propósito de este plan es establecer y documentar la planificación de las pruebas a realizar, así como la estrategia a utilizar para su ejecución.

4.3.1.1 Plan de Pruebas - Módulo Modelador

Acción	Resultado
El usuario inicia la aplicación.	
	El sistema muestra el menú principal de la aplicación.
El usuario selecciona la opción *Modelador.*	
	El sistema muestra la pantalla principal del Modulo Modelador.
El usuario elige la opción *Clase.* 	
	El sistema solicita el nombre de la clase.
El usuario ingresa el nombre de la clase y elige la opción *Aceptar*.	
	El sistema muestra la clase creada.

El usuario selecciona una clase.	
	El sistema muestras las propiedades de la clase seleccionada.
El usuario selecciona la opcion *Atributos.* 	
	El sistema muestra el editor de atributos.
El usuario ingresa los atributos de la clase y elige la opción *OK.*	
	El sistema muestra las clases con los atributos ingresados.
El usuario elige la opción Herencia. 	
El usuario selecciona la clase origen y destino.	
	El sistema muestra la relación creada entre las clases.
El usuario elige la opción Generar XMI. 	
	El sistema muestra la ventana de selección de directorio.
El usuario selecciona la carpeta y elige la opción *Guardar.*	
	El sistema genera un archivo XMI en la carpeta seleccionada.

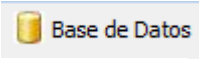
4.3.1.2 Plan de Pruebas - Módulo Conversor

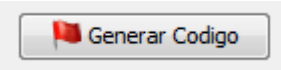
Acción	Resultado
El usuario inicia la aplicación.	
	El sistema muestra el menú principal de la aplicación.
El usuario selecciona la opción *Conversor.*	
	El sistema muestra la pantalla principal del Modulo Conversor.
El usuario elige la opción *Configuración.*	

	
	El sistema muestra el listado de los algoritmos de conversión.
El usuario selecciona los algoritmos de conversión y elige la opción *Aceptar.*	
El usuario selecciona la opción *Abrir XMI.* 	
	El sistema muestra la pantalla de selección de archivo XMI.
El usuario selecciona un archivo XMI.	
	El sistema muestra los resultados de la conversión.
El usuario elige la opción *Script.* 	
	El sistema muestra los scripts generados para la base de datos seleccionada.
El usuario inicia el Administrador de Base de Datos SQLiteManager.	
	El sistema muestra el listado de opciones de administración.
El usuario selecciona la opción *Create a Database*.	
	Él sistema genera un archivo con la nueva base de datos creada.
	El sistema muestra la pantalla de ejecución de SQL para la creación de la base de datos.
El usuario selecciona los scripts generados por el módulo	

Conversor y los ejecuta en la interfaz del SQLiteManager.	
	El sistema muestra las tablas generadas por los scripts ejecutados.

4.3.1.3 Plan de Pruebas - Módulo Generador

Acción	Resultado
El usuario inicia la aplicación.	
	El sistema muestra el menú principal de la aplicación.
El usuario selecciona la opción *Generador.*	
	El sistema muestra la pantalla principal del Modulo Generador.
El usuario selecciona la opción *Base de Datos*. 	
	El sistema muestra la pantalla de selección de base de datos.
El usuario selecciona como Administrador la opción SQLite y elige la opción Archivo.	
	El sistema muestra la ventana de selección de archivos.
El usuario selecciona el archivo de la base de datos y elige la opción *Aceptar.*	
El usuario elige la opción *Probar Conexión*.	
	El sistema muestra un mensaje de éxito de conexión exitosa.
El usuario selecciona la opción *Aceptar*	
	El sistema muestra la información de las tablas en el panel de Estructura de Base de Datos
El usuario selecciona una tabla.	
	El sistema muestra las propiedades de la tabla seleccionada.
El usuario elige la opción *Vista Previa.* 	
El usuario selecciona la	

opción *Generar Código.* 	
	El sistema solicita la carpeta donde se generara el código fuente para la tabla seleccionada.
El usuario selecciona la capeta donde se generara el código de la tabla seleccionada.	
	El sistema genera el código fuente en la carpeta seleccionada

5. Observaciones, conclusiones y recomendaciones

En este capítulo se muestran las observaciones encontradas en el proyecto, las conclusiones a las que se llegaron en su desarrollo y las recomendaciones a futuro en base al proyecto realizado.

5.1 Observaciones

- Se encontró que la metodología MDA aún está madurando, ya que aún no se tiene una definición unánime de que es MDA o no lo es.
- Se tiene que la eficiencia general de la generación de la aplicación está relacionada directamente al tamaño del modelo ingresado, ya que cada unidad (Clase) del modelo determina una unidad o más de código en la aplicación resultante.
- Se tiene que se le debe dar un grado de importancia a cual patrón de transformación seleccionar para la conversión de objetos a tablas, ya que cada uno ofrece mejor eficiencia o no en determinados escenarios.
- Se encontraron diferencias en los archivos XMI para las versiones de UML 1.4 y 2.0.

- Se encontró que el uso de plantillas para la generación de código provee el método más flexible para implementar un generador de código en distintos lenguajes de programación y bases de datos.
- En la ejecución de los algoritmos de conversión de UML a ER se tiene que el orden de ejecución de los algoritmos determina el modelo relacional resultante.

5.2 Conclusiones

- Se tiene que el producto como herramienta MDA reducirá el tiempo en la construcción de una aplicación, además de ofrecer otras ventajas adicionales.
- Se tiene que el producto sólo está limitado a un modelo único (diagrama de clases) para la generación de la aplicación.
- Se tiene que el producto puede ser usado en conjunto o por módulos en base a las funcionalidades y resultados requeridos.
- La herramienta podrá ser usada únicamente para una versión de UML ya que está restringida a una versión de XMI.
- Se tuvo que se debe usar una adaptación de RUP para una herramienta creado por un equipo pequeño, debido a la gran cantidad de entregables que implica el uso de esta metodología.

5.3 Recomendaciones y trabajos futuros

- Se recomienda el uso de la herramienta en conjunto para todo el flujo de desarrollo de la aplicación y por distintos usuarios en base al tipo de usuario definidos para cada modulo.
- Se espera que la herramienta producida pueda ser extendida a plataformas como. Net. u otras, así como diferentes administradores de base de datos, como pueden ser MySql ya que la arquitectura de la herramienta lo permite.
- En caso se desee modificar la interfaz grafica generada se pueden modificar las plantillas en el directorio del modulo generador.

Bibliografía

[1] Krutchen, The Rational Unified Process: An Introduction, Addison Wesley, 2000

[2] Wolfang, Keller. Mapping Objects to Tables, A pattern language, 1997

[3] Martin Fowler, *Patterns of Enterprise Application Architecture* Addison Wesley, 2002

[4] Scott W. Amber, Mapping Objects to Relational Databases, 2002

[5] Scott W. Amber, Building Object Applications that work Cambridge University Press 1998

[6] AndroMDA. MDA Framework

[7] Jack Herrington, Code Generation in Action ,Manning, 2003

[8] GNOME, Estándares de interfaz gráfica, 2004.

[9] Hidalgo Martínez Mario, Utilizando Dependency Injection con Spring Framework, 2006.

[10] Hidalgo Martínez Mario, Trabajando con Log4J, 2006.

[11] Giuseppe Naccarato, Template-Based Code Generation with Apache Velocity, 2004.

[12] Roman Strobl, Building Applications with Netbeans Visual Library, 2008.

[13] Jim D'anjou, The Java Developers Guide to Eclipse, Addison-Wesley Professional, 2004.

[14] Corredera Luis, Arquitectura dirigida por modelos para J2ME, Universidad Pontificia de Salamanca, 2004.

[15] Riley Mike, A Special Guide-MDA and UML Tools, 2006.

[16] Guía de los Fundamentos de la Dirección de Proyectos. Tercera Edición. Guía del PMBOK.

[17] Code Generation Models, Code Generation Network.

[18] Working XML: UML, XMI, and code generation, IBM Developer Works, 2004.

[19] UML 2.1.2 Specification , OMG, 2007.

[20] XMI DTD Specificacion, OMG, 2007.