# Universidad de las Ciencias Informáticas

## Facultad # 3

**Título:** Diseño e Implementación del módulo Entidades del plugins Génesis.

**Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.**

**Autores:** Roberto Amaury Castro Santiesteban.

**Tutor:** Ing. Yulier Matías León.

**Co-Tutor:** Ing. Reinier Lugo Díaz

**Ciudad de La Habana, Junio de 2011.**

**"Año 53 de la Revolución."**

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los _____ días del mes de _____ del año _____.

| | |
|---|---|
| Roberto Amaury Castro Santiesteban | Ing. Yulier Matías León |
| _____ | _____ |
| Firma del Autor | Firma del Tutor |

**DATOS DEL CONTACTO:**

Tutor: Ing. Yulier Matías León.

Clasificación: Profesional.

Clasificación del área de desarrollo: CEIGE.

Síntesis de la Tutor: Graduado en la Universidad de las Ciencias Informáticas con título de oro.

Actualmente es Analista y Diseñador del proyecto SAGEB perteneciente al Dpto. Soluciones Financieras.

Categoría Científica: Ingeniero

Categoría Docente: Instructor

Co-Tutor: Ing. Reinier Lugo Díaz.

Clasificación: Profesional.

Clasificación del área de desarrollo: CEGEL

Síntesis del Co-Tutor: Ingeniero Informático, dos años de graduado.

Actualmente es Analista Principal y Desarrollador del sub-proyecto Encuadernación perteneciente al proyecto de Registros y Notarías Fase III.

Categoría Científica: Ingeniero

# Agradecimientos

Agradezco ante todo a las personas que me han impulsado y no me han dejado claudicar en el logro de mis sueños, mis padres, que son mi ejemplo a seguir, mis guías, son las personas más importantes para mí, por apoyarme siempre y por brindarme todo lo necesario para mi superación, especialmente por su amor, dedicación, comprensión y ayudarme a lograr este sueño de poder regarles esta satisfacción de verme graduado espero no haberles fallado.

A toda mi familia en general, que los quiero, y son el soporte espiritual de mi vida.

A todas mis amistades, los que más tiempo pasaron conmigo en esta etapa, además han estado a mi lado en el transcurso de esta carrera, a todos en general, de todos aprendí algo y siempre los recordaré.

Al tutor, por su ayuda y empeño, para que este trabajo saliera con éxito.

A la Revolución por brindarme la posibilidad de estudiar en una universidad como esta.

Roberto A. Castro Santiesteban

# *Dedicatoria*

*A mis padres; los cuales me han apoyado mucho en estos años lejos de ellos, a quien les debo todo lo que tengo y lo que soy en esta vida.*

*A mi familia, en general los cuales aportaron su granito de arena para que hoy pueda incorporarme a la sociedad como un ciudadano útil.*

*A mis profesores quienes me guiaron en el aprendizaje, dándome los últimos conocimientos para un buen desenvolvimiento en la sociedad.*

*A todas mis amistades que han compartido conmigo esta larga y difícil tarea, siempre los tendré presente.*

*Roberto A. Castro Santiesteban*

# *Pensamientos*

*"…el mundo camina hacia la era electrónica…todo indica que esta ciencia se convertirá en algo como una medida de desarrollo; quien la domine será un país vanguardia. Vamos a colocar nuestros esfuerzos en este sentido con audacia revolucionaria".*

*"…aquí está una de las tareas de la juventud: empujar, dirigir con el ejemplo la producción del hombre de mañana. Y en esta producción, en esta dirección, está comprendida la producción de sí mismos…"*

# *Resumen*

Actualmente en la Universidad de las Ciencias Informáticas se están desarrollando un gran número de proyectos. Muchos de ellos poseen características comunes tales como el desarrollo en sus aplicaciones de módulos que se encargan de gestionar entidades del sistema.

En las aplicaciones de gestión de entidades, un alto por ciento de los módulos presenta el patrón CRUD en parte o en su totalidad. Para desarrollar estos módulos de una manera más rápida y sencilla, se propone crear modelos que recojan toda la información necesaria sobre el sistema.

La ingeniería dirigida por modelos (MDE) es un nuevo paradigma de software, la misma promete una mejora de la productividad y de la calidad del software generado ya que reduce el salto semántico entre el dominio del problema y la solución.

En el presente trabajo se propone el desarrollo de una herramienta utilizando el entorno de desarrollo Eclipse, con el propósito de llevar a cabo un desarrollo dirigido por modelos que contribuya a agilizar el proceso de desarrollo en los proyectos productivos.

Se ha realizado un análisis de las principales herramientas MDA disponibles actualmente, algunas utilizan estándares, como MOF, XML, UML mientras que otras definen sus propios lenguajes. Se sigue la metodología y los procesos que RUP propone, empleándose patrones dentro del flujo de desarrollo del software.

**Palabras Claves:** Plugin, MDD, MDE, MDA, entidades, desarrollo de software.

# Índice

# Índice de Figuras

# Índice de Tablas

# Introducción

Poco a poco con el devenir de los años la Universidad de las Ciencias Informanticas ha avanzado en el proceso de desarrollo de software, lo que ha conllevado a que sean cada vez más las empresas que solicitan de sus servicios informáticos, provocando un aumento cada vez mayor de la producción. Para poder acometer en tiempo y con calidad este proceso de producción, la universidad se plantea el uso de metodologías, frameworks[1] y herramientas que agilicen dicho proceso.

Según la metodología RUP, el proceso de desarrollo de software está dividido en 4 fases. De estas, la fase de construcción es una de las más complejas, en la que se lleva a cabo la implementación del sistema como flujo de trabajo de mayor peso (1). En los proyectos productivos, el tiempo dedicado a esta fase se extiende significativamente, por lo que los equipos de proyecto preparan el personal suficiente y los capacitan para enfrentar esta tarea al máximo y reducir el tiempo de desarrollo.

Las entidades[2] de un sistema son las bases sobre las cuales descansa un sistema y constituyen el punto de partida para llevar a cabo el desarrollo. Si estas se destruyen todo lo relacionado con la misma también deja de existir, por lo que la relación que se establece entre ellas, conceptualmente es fuerte. Generalmente para la gestión y acceso a ellas se utiliza el patrón CRUD[3], el cual se refiere a un conjunto de operaciones que permiten crear, consultar, actualizar y eliminar una entidad y aportándole un resultado final al cliente.

Una entidad puede ser persistente en una base de datos, esto significa que una clase ligera de Java representa típicamente una tabla en la base de datos emparentada. Los casos de uso de la entidad corresponden a las filas individuales en la tabla. Las entidades tienen típicamente relaciones con otras entidades, y estas relaciones se expresan con objeto/meta datos emparentados. El objeto/los meta datos

---

[1] ***Framework:*** *un esquema (un esqueleto, un patrón) para el desarrollo y/o la implementación de una aplicación.*

[2] ***Entidad*** *o* ***ente*** *es todo aquello cuya existencia es reconocida por algún sistema de ontología, por lo tanto puede ser concreta, abstracta, particular o universal*

[3] ***Acrónimo de Create-Read-Update-Delete****. Conocido como el padre de todos los patrones de capa de acceso. Describe que cada objeto debe ser creado en la base de datos para que sea persistente. Una vez creado, la capa de acceso debe tener una forma de leerlo para poder actualizarlo o simplemente borrarlo*

emparentados se puede especificar directamente en el archivo de la clase de la entidad usando anotaciones, o en un separado XML archivo del descriptor distribuido con el uso.

Las tecnologías actuales como ORM[4] permiten el mapeo entre las entidades del sistema y sus tablas correspondientes en una base de datos. Esto crea una abstracción significativa para el desarrollador que no necesita realizar transformaciones manualmente en el código entre el modelo relacional y el objetual y viceversa, pero el proceso de creación de la base datos y los ficheros ORM requieren de tiempo y cuidado para garantizar el éxito y si ocurren cambios en las entidades estas deben actualizarse manualmente, así como los ficheros ORM y las tablas de la base datos.

Las herramientas actuales de generación de ficheros ORM y script de base de datos tienen separado estos procesos de forma tal que la generación se hace independiente y no de forma integrada lo que no permite que cualquier cambio en las entidades del sistema se refleje automáticamente en los ficheros ORM y las tablas de la base de datos. También a estos sistemas de generación le falta la información suficiente para que la generación obtenida sea la deseada por el desarrollador, lo que implica que el desarrollador tenga que arreglar el código generado.

Teniendo en cuenta lo anterior se decidió realizar un sistema que permita llevar a cabo un desarrollo dirigido por modelos que posibilite crear un modelo único y flexible, en el que se guarden los datos de las entidades según los metadatos identificados. Este modelo sería independiente de la plataforma o arquitectura que se utilice para el desarrollo de la aplicación final, aminorando la complejidad asociada con aplicaciones desarrolladas en diferentes tecnologías.

Dada la situación antes planteada surge el siguiente **problema a resolver:**

¿Cómo lograr la gestión de los modelos de entidades y la transformación de estos hacia una arquitectura específica que permita desarrollar software de forma ágil?

El **objeto de estudio** lo constituyen las herramientas de gestión de modelos de entidades, donde el **campo de acción** queda enmarcado en la gestión de modelos de entidades en el plugin[5] de Génesis.

Con el propósito de dar solución al problema anteriormente planteado, se ha trazado como **objetivo general** el desarrollo del módulo Entidades de Génesis para el entorno de desarrollo de Eclipse, obteniéndose los siguientes objetivos específicos:

---

[4] ***ORM**: (Del Inglés Object-Relational Mapping), Mapeo del modelo de objetos al modelo relacional*

[5] ***Plugin:** programa que interactúa con una aplicación principal para proveerle de cierta función normalmente muy específica.*

- ✓ Estudiar el marco teórico de la investigación sobre la gestión de entidades.
- ✓ Realizar el diseño del módulo entidades del plugin Génesis.
- ✓ Diseñar e implementar el módulo de entidades del plugin Génesis.
- ✓ Llevar a cabo la fase de pruebas.

Para dar solución a lo antes planteado se definen las siguientes **tareas de investigación:**

1. Caracterización de algunos sistemas que implementan un desarrollo dirigido por modelos.
2. Realización del meta-modelo de entidades del plugins Génesis.
3. Realización del diseño de clases visuales y auxiliares del módulo Entidades del plugins Génesis.
4. Realización del modelo de componentes del módulo Entidades del plugins Génesis
5. Implementación del modelo de componentes del módulo Entidades del plugins Génesis.
6. Realización de pruebas unitarias a la implementación del módulo Entidades del plugins Génesis.

Partiendo de los objetivos específicos antes mencionados, se obtendrá como **posible resultado** el módulo Entidades del plugin Génesis para el entorno de desarrollo Eclipse.

Para desarrollar este trabajo se ha planteado la siguiente **idea a defender:** si se logra la gestión de entidades del negocio teniendo en cuenta los requisitos propuestos por Génesis, se acorte el tiempo invertido por los desarrolladores actualmente en la implementación, esto contribuye a agilizar el proceso de desarrollo de software en los proyectos productivos.

Los siguientes **métodos teóricos** sustentan el trabajo de investigación:

- ➢ Histórico-Lógico: Su empleo permitió el desarrollo evolutivo y coherente en el estudio de la metodología orientada a objetos, patrones de diseño, herramientas CASE[6] y estudio de los plugins.
- ➢ Analítico-Sintético: Permitió integrar y descomponer el conocimiento, resultó importante a la hora de fundamentar los elementos más significativos de la investigación determinando los aspectos esenciales y el arribo a conclusiones prácticas y teóricas.
- ➢ Inductivo-deductivo: Permitió pasar de la particularidad del proceso a lo general y viceversa beneficiando objetivamente el vínculo que se establece en la realidad entre lo singular que ocurre en cada entidad nacional y lo general que esta estandarizado para ellas, ya que ambas se complementan mutuamente en el proceso de desarrollo.

[6] ***CASE:*** *Ingeniería de Software Asistida por Ordenador (del inglés Computer Aided Software Engineering).*

- Modelación: Pues se crean abstracciones que explican la realidad, por ejemplo, todos los modelos y diagramas presentados.

La estructura del documento está definida de la siguiente manera:

- **En el Capítulo 1. Fundamentación teórica:** Se exponen los conceptos fundamentales relacionados con el tema de investigación y se describen los principales aspectos de los lenguajes, herramientas y metodologías a utilizar para el desarrollo de un plugin de Eclipse.

- **En el Capítulo 2. Características del sistema:** El propósito de este capítulo es la descripción del objeto de estudio del presente trabajo. Se describirá la propuesta del sistema como solución a los problemas que dan paso a esta investigación, así como el entorno donde se desarrolla el plugin mediante el modelo de dominio, el cual se representa mediante una herramienta. Se realiza el diseño del mismo partiendo de los casos de uso elaborado por el analista.
- **En el Capítulo 3. Prueba y validación de la solución propuesta:** Se realiza todo lo relacionado con los flujos de trabajo de implementación y prueba, se desarrolla el diagrama de implementación para dar una visión de cómo las clases, artefactos y otros elementos de bajo nivel, se unen para formar componentes de alto nivel así como las conexiones entre ellos y se realizan los casos de pruebas para garantizar buena calidad al software.

# Capítulo 1: Fundamentación Teórica

## 1.1. Introducción

Las tecnologías en la sociedad de hoy se encuentran en constante avance, desarrollo y evolución. Nuestro país apuesta por tener una independencia tecnológica del software y aplicaciones libres, por los grandes beneficios que esto traería a la economía nacional, destacándose el ahorro de miles de dólares en compra de licencias. En el presente capítulo se describen las tendencias, herramientas y tecnologías usadas para el desarrollo de un plugin de Eclipse. También se realiza un análisis sobre las características y funcionalidades de los plugins de Eclipse más utilizados, tanto en la universidad como en el mundo, los cuales facilitan el proceso de configuración de frameworks, y que propicien un mejor entendimiento para una propuesta de solución.

## 1.2. Conceptos relacionados con el dominio del problema.

### 1.2.1. Modelo.

**¿Qué es un modelo?** Un modelo es una representación, en cierto medio, de algo en el mismo u otro medio. El modelo capta los aspectos importantes de lo que estamos modelando, desde cierto punto de vista, y simplifica u omite el resto. La ingeniería, la arquitectura y muchos otros campos creativos usan modelos.

Un modelo de un sistema software está construido en un lenguaje de modelado, como UML. El modelo tiene semántica y notación y puede adoptar varios formatos que incluyen texto y gráficos. El modelo pretende ser más fácil de usar para ciertos propósitos que el sistema final. (5)

Un modelo es una representación abstracta de alguna cosa que puede o no existir en la realidad; es una simplificación que muestra ciertos aspectos de lo que se desea modelar, escondiendo aquellos elementos que no son de interés a los que usaran el modelo. (6)

Los modelos pueden ser expresados de muchas formas; ya que pueden ser representados en todas las formas del lenguaje gráficas o textuales existentes. Así puedo tener un modelo de un sistema representado por Casos de Uso escritos en lenguaje UML, o puedo tener el código en lenguaje Java del

sistema, que también, para sorpresa de algunos, es un modelo del sistema (recordemos que el sistema final es un conjunto de 0s y 1s, de los que el desarrollador del sistema no se entera, es decir hay una abstracción de estos detalles); la diferencia entre los dos modelos mencionados radica en el nivel de abstracción.

De forma específica, han sido aprovechados dentro la ingeniería para representar los problemas planteados y sus posibles soluciones; ya que la abstracción realizada en el modelo se ha utilizado como base para una mejor comunicación entre los diferentes participantes del problema, así como para una mejor compresión del mismo. Sin embargo, más allá del modelo en sí, lo que realmente interesa a todas las ramas de la ingeniería es la construcción de métodos, herramientas, etc. que sirvan de base para la construcción de modelos que sean realmente útiles; es decir el tema de estudio es el modelamiento en sí.

### 1.2.2. Metamodelo.

Un metamodelo es un modelo que define el lenguaje para expresar un modelo.

El metamodelo de un lenguaje es una descripción de todos los conceptos que pueden usarse en el mismo. Por ejemplo,

- ✓ los conceptos de package, clase , atributo y operaciones aparecen en UML;
- ✓ los conceptos de métodos, constructores e interfaces en JAVA;
- ✓ los conceptos de tabla, columna, clave son parte de SQL

Cada elemento de un modelo es una instancia de una metaclase en el metamodelo.

Una clase define a sus objetos y una metaclase define a los elementos del modelo.

Los metamodelos son herramientas de Control, sin embargo, como veremos más adelante, también nos permiten Generar ocurrencias, en forma automática, y establecer Comportamiento. Esta clasificación, sin embargo no es excluyente, ya que un metamodelo puede ser de Control y Comportamiento o de Control y Generativo.

*Metamodelos de Control:* están destinados a verificar que el ingreso de un Objeto este supeditado a la existencia de su Tipo.

*Metamodelos Generativos:* no solamente permiten controlar el ingreso o modificación de datos en la base, estos pueden generar en forma automática ocurrencias en la base de datos sin necesidad que la aplicación lo haga. (15)

### 1.2.3. Ingeniería basada en Modelos (MDE).

La ingeniería basada en modelos (MDE) hace referencia al uso sistemático de modelos, como los elementos principales en la ingeniería de software, que se centra en la creación de modelos, o abstracciones, más cerca de algunos conceptos de dominio particular, en lugar de la informática (o algoritmos) conceptos. Tiene el propósito de aumentar la productividad mediante la maximización de la compatibilidad entre los sistemas, simplificando el proceso de diseño, y promover la comunicación entre los individuos y equipos que trabajan en el sistema.

Un paradigma de modelado para MDE se considera eficaz si sus modelos tienen sentido desde el punto de vista del usuario y puede servir como base para la implementación de sistemas. Los modelos son desarrollados a través de una amplia comunicación entre los gerentes de producto, diseñadores y miembros del equipo de desarrollo. Como la realización método de modelos, que permiten el desarrollo de software y sistemas.

La mejor iniciativa de MDE conocido es el Object Management Group (OMG) iniciativa Model-Driven Architecture (MDA), que es una marca registrada de OMG. (2)

¿MDE cómo se utiliza en ingeniería de software?

Lo que se refiere al desarrollo de software, ingeniería basada en modelos se refiere a una serie de enfoques de desarrollo que se basan en el uso de software de modelado como una forma principal de expresión. A veces los modelos se construyen para un determinado nivel de detalle. En ocasiones se construyen modelos completos incluyendo las acciones ejecutables. El código puede ser generado a partir de los modelos, que van desde el esqueleto del sistema para completar los productos de despliegue. Con la introducción del Lenguaje Unificado de Modelado (UML), MDE se ha vuelto muy popular hoy en día con un amplio cuerpo de profesionales y herramientas de apoyo. Tipos más avanzados de la MDE se han ampliado para permitir que los estándares industriales que permiten la aplicación coherente y resultados. La continua evolución del MDE ha añadido un mayor enfoque en la arquitectura y la automatización.

Tecnologías de MDE con un mayor enfoque en la automatización de la arquitectura y el correspondiente rendimiento de más alto nivel de abstracción en el desarrollo de software. Esta abstracción promueve modelos más simples con un mayor énfasis en espacio del problema. Combinado con la semántica ejecutable este eleva el nivel total de automatización posible. El Object Management Group (OMG) ha

desarrollado un conjunto de normas llamada arquitectura dirigida por modelos (MDA), la creación de una base de este enfoque centrado en la arquitectura avanzada. (3)

### 1.2.4. Desarrollo basado en Modelos (MDD).

Uno de los principios básicos de la Ingeniería del Software es la **abstracción**, para separar lo esencial de lo no esencial, en términos del desarrollo de software, lo esencial es la **funcionalidad**, y lo no esencial es la plataforma tecnológica, estas abstracciones nos las provee los modelos, donde el modelado y la transformación de modelos hasta el nivel de abstracción requerido, constituye el núcleo del Desarrollo Dirigidos por los Modelos (MDD). MDD es un estilo de desarrollo de software donde el artefacto primario son los modelos, a partir de los cuales se obtiene el código y otros artefactos, los modelos se usan para pensar en el dominio del problema y en el dominio de la solución.

Para poder integrar sistemas de software con tecnologías diversas (EJB/J2EE, .NET, XML, SOAP, etc.) y que puedan adaptarse rápidamente a los negocios cambiantes, la tendencia es apostar al desarrollo dirigido por modelos (Model Driven Development, MDD).

El principio básico detrás del desarrollo basado en modelos es que si sabemos que algo va a cambiar en una forma predecible, modelemos la descripción del cambio de tal forma que sea fácil modificarlo. En otras palabras: si algo va a cambiar mucho, hazlo fácil de cambiar. Al desarrollo basado en modelos, también se la conoce como Ingeniería Basada en Modelos (Model Driven Engineering, MDE) de las cuales la Arquitectura Dirigida por Modelos (Model Driven Architecture, MDA) propuesta por el Object Management Group (OMG) es la más avanzada. Sin embargo, este no es el único enfoque al respecto, también está la propuesta de Modelos de Objetos Adaptables (Adaptive Object Models) y Metamodelos que intentan facilitar el construir sistemas dinámicos y adaptables, que está altamente relacionado con la investigación de reglas de negocio, específicamente cuando se necesitan medios para describir las reglas de negocio y automáticamente generar implementaciones.

### 1.2.5. Arquitectura basada en Modelos (MDA).

MDA es el acrónimo de Model Driven Architecture (arquitectura dirigida por modelos), que propone basar el desarrollo de software en modelos especificados utilizando UML para que, a partir de ellos, se realicen transformaciones que generen código u otro modelo, con características de una tecnología particular (o con menor nivel de abstracción). Suele escucharse decir que MDA es la evolución natural de UML, ya que tiende a incrementar la cantidad de código generado, a partir de especificaciones detalladas en UML.

MDA, es un enfoque de diseño de software para el desarrollo de sistemas de software. Proporciona un conjunto de directrices para la estructuración de las especificaciones, que se expresan como modelos. Arquitectura basada en modelos es un tipo de ingeniería de dominio, y apoya la ingeniería basada en modelos de sistemas de software. Fue lanzado por el Object Management Group (OMG) en 2001. (4)

Se debe tener en cuenta que el término "arquitectura" de la arquitectura basada en modelos no se refiere a la arquitectura del sistema que está siendo modelado, sino más bien a la arquitectura de las diversas normas y modelos de formularios que sirven de base tecnológica para la MDA.UML ejecutable es otro enfoque específico para aplicar la MDA.

Uno de los principales objetivos de la MDA es separar el diseño de arquitectura. Como los conceptos y las tecnologías utilizadas para realizar los diseños y los conceptos y las tecnologías utilizadas para realizar arquitecturas han cambiado a su propio ritmo, la disociación que permite a los desarrolladores del sistema para elegir la mejor y más adecuada en ambos dominios. El diseño se refiere a la funcional (caso de uso), mientras que los requisitos de la arquitectura proporcionan la infraestructura a través del cual los requisitos no funcionales como escalabilidad, fiabilidad y rendimiento se realizan. MDA prevé que el modelo independiente de la plataforma (PIM), lo que representa un diseño conceptual de la realización de los requisitos funcionales, a sobrevivir a los cambios en las tecnologías de la realización y arquitecturas de software.

### 1.2.6. Transformación de Modelos.

En MDD (Model Driven Development) los conceptos claves son los modelos y las transformaciones de modelos. MDD puede ser visto en forma genérica como un proceso de desarrollo de software implementado mediante una red de transformaciones entre modelos que se combinan o componen en modos diversos, haciendo al desarrollo dirigido por modelos mucho más abierto y flexible. La composición de transformaciones requiere contar con operadores para composición dentro de un lenguaje simple de transformación entre modelos. (16)

### 1.3. Sistemas que gestionan modelos de Entidades dirigidos por modelos.


**Visual Paradigm**

Es una herramienta CASE que utiliza "UML": como *lenguaje* de modelaje. Se integra con las siguientes herramientas Java:

- ✓ Eclipse/IBM WebSphere
- ✓ JBuldier
- ✓ NetBeans IDE
- ✓ Oracle JDeveloper
- ✓ BEA Weblogic

El software de modelado ayuda a una rápida construcción de aplicaciones de calidad, mejores y con menor costo. Está disponible en varias ediciones, cada una destinada a unas necesidades: Enterprise, Professional, Community, Standard, Modeler y Personal, donde la version *Community* solo se puede utilizar para aplicaciones no comerciales, una de las limitaciones con las que cuenta Visual Paradigm es a la hora de realizar cualquier actualización en los diagramas de dominios generados, esto se torna un poco engorroso cuando se sincroniza el cambio con la base de datos. (13)


**AndroMDA**

AndroMDA es un framework MDA Open Source, su función principal es tomar cualquier número de modelos generalmente en UML en formato XMI producidos por herramientas de diagramación, combinándolos con plugins de AndroMDA que producen un buen número de componentes (código) personalizable. Se pueden generar cualquier cantidad de componentes en gran número de lenguajes entre los que se encuentra JAVA, PHP, .NET, HTML solo con utilizar, adaptar y realizar nuevos o existentes plugins. (17)


**ArcStyler**

Herramienta comercial para la transformación de modelo a modelo, generación de código, y generación de archivos para pruebas y despliegue, 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. Usa Rose como herramienta de modelado UML, soportando directamente los ficheros MDL. No soporta diagramas de componentes ni diagramas de despliegue, pero admite código propio para la generación de código. (19)

**OptimalJ**

OptimalJ de *Compuware* es una herramienta MDA que utiliza MOF para soportar estándares como UML y XMI. Se trata de un entorno de desarrollo que permite generar aplicaciones J2EE[7] completas a partir de un PIM. 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. Admite XMI tanto para la importación de ficheros como para su salida. (20)

**GeneXus**

GeneXus es una poderosa herramienta multiplataforma para el desarrollo de aplicaciones. Permite el desarrollo incremental de aplicaciones de negocios. Genera el 100% de la aplicación, y mantiene automáticamente el modelo de datos, la información y las aplicaciones. GeneXus es usado para desarrollar complejas aplicaciones de misión crítica con extensas bases de datos, que incluyen tanto aplicaciones centralizadas como distribuidas y basadas en Web. Valida sus requerimientos en la etapa de diseño, a través de prototipos 100% funcionales. Genera la base de datos y cuando cambian los requerimientos, automáticamente realiza un análisis de impacto y propaga los cambios. GeneXus es un sistema muy completo pero es privativo y de pago. (18)

**Codagen Architect 3.2**

Permite la transformación de modelos y la automatización de la generación de código partiendo de los modelos CIM. Como integración con herramientas de modelado, Codagen incorpora plugins para Microsoft Visio, Rational Rose y Borland Together Control. Admite como entrada ficheros XMI y ficheros MDL de Rational Rose. Entre los estándares libres soportados tenemos únicamente Struts, pero como contrapartida, los lenguajes de salida que admite son

[7] ***J2EE**: Java Platform, Enterprise Edition o Java EE es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en lenguaje JAVA con arquitectura de N niveles distribuidos, basándose en componentes de software modulares se ejecuta sobre servidores aplicaciones.*

Java, C#, C++ y Visual Basic, admitiendo también la incorporación de código propio al código generado. (22)

**IQGen 1.4**

El generador de modelos y código IQGen, de la empresa InnoQ está desarrollado en Java. Incorpora un entorno de modelado muy pobre, que se suple con la admisión de entrada de modelos en formato XMI. Puede generar salida en cualquier lenguaje (J2EE/EJB, C#, Cobol, entre otros). (23)

## 1.4. Metodología empleada en el desarrollo.

### 1.4.1. Metodología (RUP)

Hoy existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo; estas engloban procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un producto de software. Una metodología debe ser lo suficientemente precisa como para que todo el mundo la pueda seguir y sea de utilidad como pauta común, pero también debe ser lo suficientemente adaptable como para poder aplicarse en distintos proyectos, y lo suficientemente sencilla como para que no resulte muy gravosa su utilización, pero lo suficientemente completa y compleja como para que la utilización por parte del equipo sea provechosa. RUP no es un sistema de pasos rígidamente establecidos, sino una metodología adaptable al contexto y necesidades de cada organización y que se compone de fases y disciplinas como se muestra en la figura 2, lo cual permite guiar las actividades de un proyecto y contar con una documentación adecuada a lo largo del desarrollo de un proyecto.

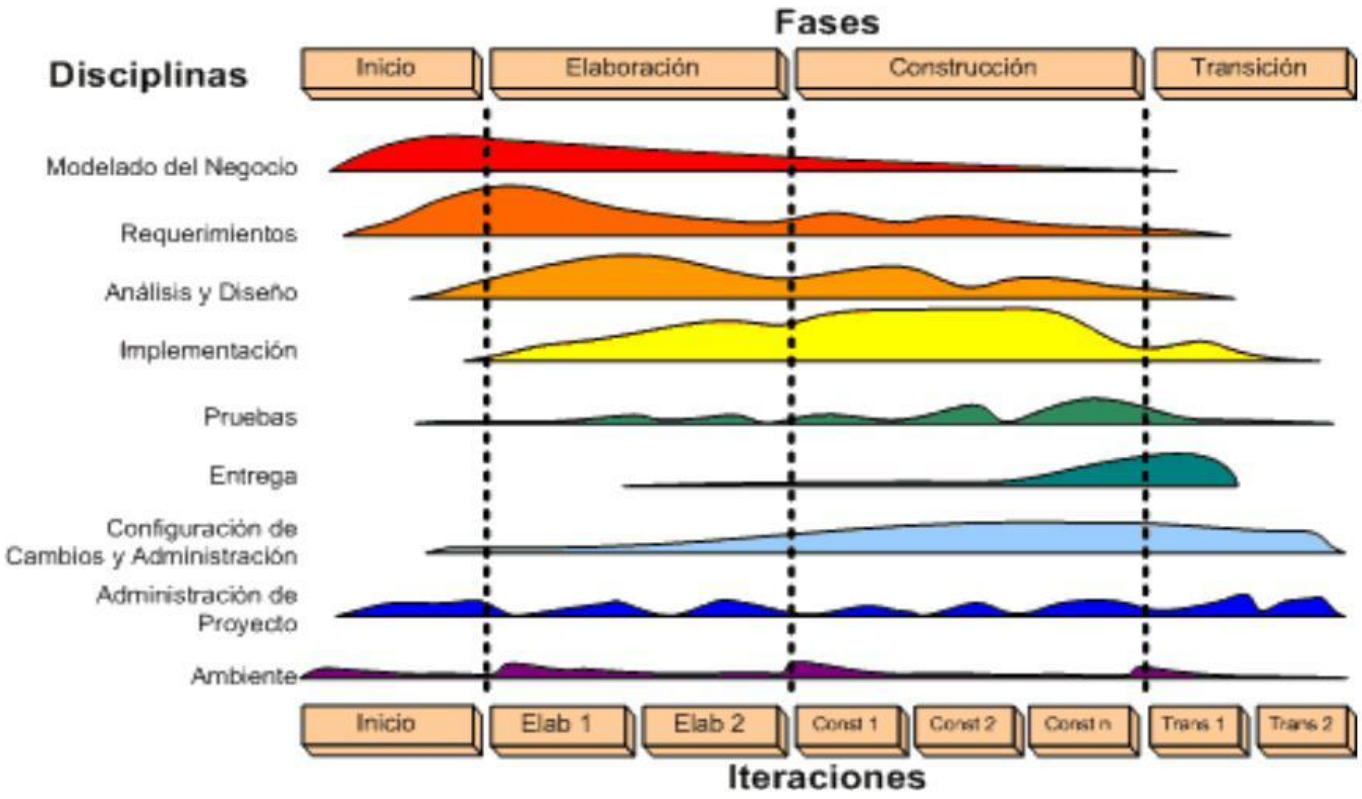



**Figura 1 Faces y disciplinas de la metodología RUP**

## 1.5. Lenguajes y herramientas utilizadas en el desarrollo.

### 1.5.1. Java

Java es un lenguaje de desarrollo orientado a objetos, multiplataforma. Proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas. Es un lenguaje interpretado y compilado, esto quiere decir que su código fuente se transforma en algo similar al código máquina, los bytecodes. A su vez estos bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real (run-time). Además, es considerado un lenguaje robusto, fiable y seguro, para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución.

Soporta la sincronización múltiple de hilos de ejecución (multithreading), lo cual da la posibilidad que un hilo puede ocuparse de interactuar con el cliente y otro de realizar una operación. Otra de las tantas características que lo hacen idóneo para su utilización, es que se utiliza para dos tipos de programas, aplicaciones independientes y applets.

### 1.5.2. Lenguaje de anotación extensible (XML)

XML[8] no es sólo un lenguaje, es una forma de especificar lenguajes, de ahí lo de extensible, por otra parte es una tecnología en realidad muy sencilla que tiene a su alrededor otras tecnologías que la complementan y la hacen mucho más grande y con posibilidades mucho mayores.

XML, con todas las tecnologías relacionadas, representa una manera distinta de hacer las cosas, más avanzada, cuya principal novedad consiste en permitir compartir los datos con los que se trabaja a todos los niveles, por todas las aplicaciones y soportes. El XML juega un papel fundamental en este mundo actual, que tiende a la globalización y la compatibilidad entre los sistemas, ya que es la tecnología que permitirá compartir la información de una manera segura, fiable y fácil. Además permite al programador y los soportes dedicar sus esfuerzos a las tareas importantes cuando trabaja con los datos, ya que algunas tareas tediosas como la validación de estos o el recorrido de las estructuras corre a cargo del lenguaje y está especificado por el estándar, de modo que el programador no tiene que preocuparse por ello.

Ventajas de XML:

- Las aplicaciones se pueden generar rápidamente y su mantenimiento es más sencillo.
- Separa los datos de la presentación y del proceso, lo que permite mostrar y procesar los datos al gusto deseado con sólo aplicar distintas hojas de estilo y aplicaciones.
- La información es más accesible y reutilizable, por la flexibilidad de las etiquetas de XML que permiten su utilización sin tener que amoldarse a reglas específicas de un fabricante. (10)

[8] ***XML:*** *Lenguaje de marcado extensible (Del Inglés Extensible Markup Language).*

### 1.5.3. Lenguaje Unificado de Modelado (Unified Modeling Language, UML)

UML es ante todo un lenguaje. Un lenguaje proporciona un vocabulario y una regla para permitir una comunicación. En este caso, este lenguaje se centra en la representación gráfica de un sistema.

Este lenguaje nos indica cómo crear y leer los modelos, pero no dice cómo crearlos. Esto último es el objetivo de las metodologías de desarrollo.

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones:

- Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Aunque UML está pensado para modelar sistemas complejos con gran cantidad de software, el lenguaje es los suficientemente expresivo como para modelar sistemas que no son informáticos, como flujos de trabajo (workflow ) en una empresa, diseño de la estructura de una organización y por supuesto, en el diseño de hardware.

Está compuesto por tres clases de bloques de construcción:

- Elementos: Los elementos son abstracciones de cosas reales o ficticias (objetos, acciones, etc.)
- Relaciones: relacionan los elementos entre sí.
- Diagramas: Son colecciones de elementos con sus relaciones. (11)

### 1.5.4. Eclipse 3.6

*¿Qué es Eclipse?* En la web oficial de Eclipse, el mismo se define como “una plataforma (IDE), abierta para todo y para nada en particular”. Eclipse es un plataforma porque no es una aplicación acabada de por sí, pero está diseñada para ser extendida indefinidamente con herramientas sofisticadas. Es un ambiente de desarrollo integrado (IDE) porque provee herramientas para administrar áreas de trabajo (o workspaces

en inglés), para construir, lanzar y depurar aplicaciones, para compartir artefactos con un equipo y versionar el código fuente. Eclipse es abierto porque su diseño le permite ser extendido fácilmente por terceras partes. Es apropiado para todo, ya que ha sido utilizado exitosamente para construir ambientes para un amplio rango de temas, como el desarrollo en lenguaje de programación Java, Servicios Web, certámenes de programación de juegos. Eclipse no es *para nada en particular*, pues no se enfoca en ningún dominio específico. (12)

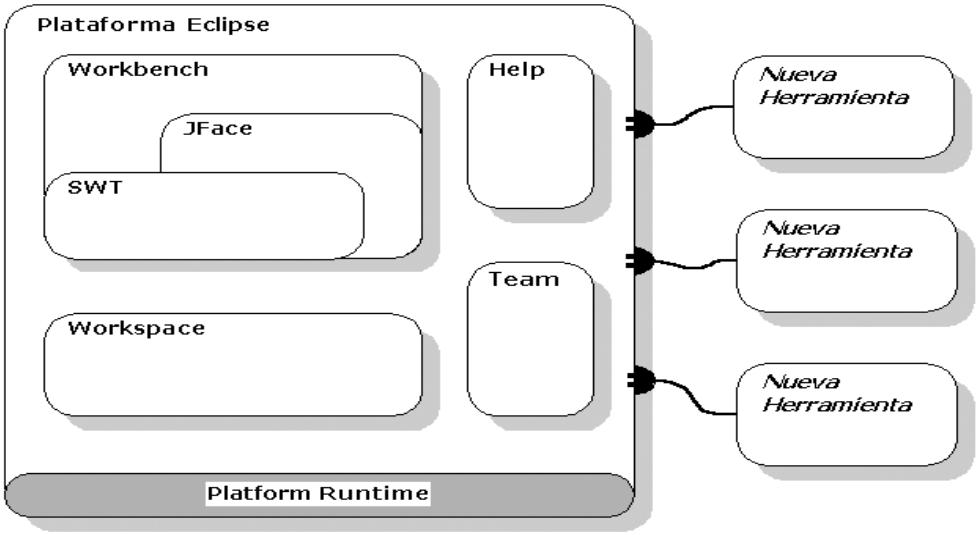



**Figura 2 Arquitectura basada en plugin.**

Un *plugin* es la unidad más atómica y extensible de Eclipse. Está escrito puramente en el lenguaje de programación Java y típicamente consiste en código Java empaquetado en un archivo de Java (JAR), con algunos archivos de solo lectura, y otros recursos como imágenes, catálogo de mensajes, etc.

**SWT**

El Standard Widget Toolkit es un conjunto de componentes para construir interfaces gráficas en Java, (widgets) desarrollados por el proyecto Eclipse. Recupera la idea original de la biblioteca AWT de utilizar componentes nativos, con lo que adopta un estilo más consistente en todas las plataformas.

SWT es un framework que crea a través de JNI (Java Native Interface) interfaces gráficas nativas del Sistema Operativo en donde ejecutemos nuestra aplicación SWT. Esto quiere decir que con el mismo

código visualizaremos en cada Sistema Operativo nuestras ventanas como si hubieran sido creadas para ese SO en específico.

La interfaz del workbench de Eclipse (vistas, editores, perspectivas, asistentes) depende de una capa intermedia de interfaz gráfica de usuario (GUI) llamada JFace que simplifica la construcción de aplicaciones basadas en SWT.

**JFace**

Es una librería que usa SWT para facilitar ciertas labores en la creación de GUIs o incluir nuevos widgets de más alto nivel. Básicamente añade funcionalidad a la librería básica de SWT. Es independiente del SO. Sin embargo SWT no depende de ningún código JFace o plataforma en absoluto.

JFace está diseñado para proporcionar la funcionalidad de interfaz de usuario de aplicación común en la parte superior de la biblioteca SWT. JFace no intenta "ocultar" SWT o reemplazar su función. Proporciona clases e interfaces que se encargan de muchas de las tareas comunes asociadas con la programación de una dinámica interfaz de usuario mediante SWT.

**PDE**

El Plugin Development Enviroment o PDE, no es más que el Ambiente de Desarrollo para Plugin que provee Eclipse y uno de los principales subproyectos del mismo, junto con JDT. Está compuesto por un conjunto de herramientas que cubren todo el ciclo de vida completo del desarrollo de un plugin. Facilita todo el proceso de crear, desarrollar, probar, depurar, construir y desplegar plugins de Eclipse.

Algunas de estas herramientas de PDE incluyen:

- ✓ *Editores del Manifiesto*: ofrece editores multipáginas basados en formularios, que administran centralmente todos los ficheros del manifiesto del plugin, o sea, el MANIFEST.MF y/o el plugin.xml.
- ✓ *Asistentes para la creación de nuevos proyectos*: facilitan todo el proceso de creación de un proyecto plugin.
- ✓ *Asistentes para importar*: provee un conjunto de asistentes que simplifican el proceso de importar plugins del sistema de archivos o file system.
- ✓ *Asistentes para exportar*: provee un conjunto de asistentes que construyen, empaquetan y exportan plugins con un simple clic a un formato de despliegue.

- ✓ *Launchers (Probadores)*: permiten probar y depurar aplicaciones de Eclipse, así como paquetes OSGi.
- ✓ *Vistas:* PDE provee cuatro vistas y una perspectiva de desarrollo, denominada Plug-in Development, que ayudan a los desarrolladores de plugins a inspeccionar los diferentes aspectos de su ambiente de desarrollo.
- ✓ *Herramientas Combinadas*: posee asistentes para externalizar y purificar los ficheros del manifiesto del plugin.
- ✓ *Herramientas de Conversión*: consiste en asistentes para facilitar el proceso de convertir un proyecto simple de Java o un archivo JARs simple, en un proyecto plugin.
- ✓ *Integración con el JDT*: los ficheros del manifiesto del plugin participan en búsquedas Java y en la reestructuración del código.

### 1.5.5. Visual Paradigm 6.3

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Visual Paradigm forma parte del IDE[9] de Eclipse. Está diseñado para desarrollar software con Programación Orientada a Objetos, examina y permite reducir la duración del ciclo de desarrollo brindando ayuda tanto a arquitectos, analistas, diseñadores y desarrolladores. Busca también automatizar tareas tediosas que pueden distraer a los desarrolladores.

Dentro de sus características fundamentales están:

- ➢ Multiplataforma: Soportada en plataformas Java para Sistemas Operativos Windows, Linux y Mac OS X

[9] ***IDE:*** *Entorno de Desarrollo Integrado del inglés Integrated Development Environment.*

- Interoperabilidad: Intercambia diagramas UML y modelos con otras herramientas. Soporta exportar e importar a XMI, XML y archivos Excel. Importa archivos de proyectos de Rational Rose. Integración con Microsoft Office Visio.
- Modelamiento de los Requisitos: Captura de requisitos con diagrama de requisitos, modelamiento de casos de uso y análisis textual.
- Colaboración de Equipo: Realiza el modelado en colaboración y simultáneamente con el Visual Paradigm TeamWork Server y Subversion.
- Generación de Documentación: Comparte y genera los diagramas y diseños en formatos como PDF, HTML y Microsoft Word.
- Editor de Detalles de Casos de Uso: Entorno todo en uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Ingeniería de Código: Permite generación de código e ingeniería inversa en lenguajes como Java, C++, CORBA, IDL, PHP, XML Schema, Ada, Python, C#, VB .NET, ODL, Flash ActionScript, Delphi, Perl y Rugby. También permite ingeniería inversa.
- Modelado de Procesos de Negocio: Visualiza, comprende y mejora los procesos de negocio con la herramienta muy completa para estos procesos.
- Integración con Entornos de Desarrollo: Apoyo al ciclo de vida completo de desarrollo del software: análisis, diseño e implementación, en IDE como Eclipse, Microsoft Visual Studio, NetBeans, Sun ONE, Oracle JDeveloper, JBuilder y otros.
- Modelamiento de Bases de Datos: Generación de bases de datos, conversión de diagramas entidad-relación a tablas de base de datos, mapeos de objetos y relaciones, ingeniería inversa desde gestores de bases de datos. (13)

### 1.5.6. SVN (SubVersion)

**Subversion** es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS, el cual posee varias deficiencias. Es software libre bajo una licencia de tipo Apache/BSD y se lo conoce también como **svn** por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de

versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo. (14)

### 1.5.7. Sub Eclipse.

Subeclipse es un plugin para **Eclipse IDE** que permite soportar subversion. Este software está bajo la Licencia Pública de Eclipse (EPL), lo cual significa que es gratuito.

Para los que no saben subverion es un sistema que permite el manejo de versiones y es altamente útil para el desarrollo web.

Cosas que puedes hacer con subeclipse:

- ✓ sincronizar tu proyecto con un repositorio
- ✓ hacer commint
- ✓ realizar update
- ✓ manejo de branch
- ✓ ignorar contenido
- ✓ realizar reverts

Eclipse trae integrado CVS como sistema de control de versiones porque tradicionalmente ha sido uno de los sistemas más populares. No obstante, las ventajas de Subversion frente a CVS hacen de SVN una alternativa cada vez más usada. Eclipse ofrece una interfaz bastante buena para trabajar con proyectos remotos colaborativos mediante CVS, y aunque no incorpore SVN de fábrica, era de esperar que gracias a su fortísima orientación a framework haya extensiones que ofrezcan la misma funcionalidad para SVN. En concreto Tigris.org, la misma comunidad que desarrolla SVN, también mantiene Subclipse, un plugin que integra SVN en Eclipse.

Subclipse incluye un conector opcional Mylyn que permite crear de cambios automática basada en las tareas que están trabajando. También permite a los enlaces a las tareas que la historia la visión de Subversion se compromete.

Por último, Subclipse incluye una función de revisión gráfica de gran alcance que se construye con Eclipse GEF/Draw2D. Esto le permite visualizar compromete y se fusiona a través de las ramas de Subversion.

### 1.5.8. Herramienta Ecore


EMF[10] es un marco de trabajo de Eclipse que unifica Java, XML y UML, permitiendo a los desarrolladores construir rápidamente aplicaciones robustas basadas en modelos simples. El metamodelo usado para representar modelos en EMF se llama Ecore. Ecore es en sí mismo, un metamodelo EMF, y así es su propio metamodelo, es decir, un metamodelo. (26).

Ecore es un subconjunto pequeño y simplificado de UML (se utiliza un subconjunto de UML en vez del mismo porque éste es demasiado ambicioso modelando, más de lo que el núcleo de EMF puede soportar). Las herramientas de componente Ecore proporciona un completo entorno para crear, editar y mantener Ecore modelos. Este componente facilita la manipulación de los modelos Ecore con un editor gráfico y puentes a otras herramientas existentes Ecore (Emfatic, generadores...). La gráfica Ecore Editor implementa soporte multi-diagrama, una vista personalizada propiedades con fichas, comentarios de validación, las capacidades de refactorización. El objetivo a largo plazo es proporcionar el mismo nivel de servicios al igual que JDT para Java. (25)

## 1.6. Patrones de diseño.

Los patrones de diseño son soluciones estándares a problemas específicos y comunes del diseño orientado a objetos, una manera más práctica de describir ciertos aspectos de la organización de un programa, además son soluciones basadas en la experiencia y una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios.

### 1.6.1. Patrones de diseño GRASP (General Responsibility Asignment Software Patterns).

Los Patrones de Software para la Asignación General de Responsabilidad (GRASP) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones, aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software

Existen nueve patrones GRASP los cuales son: Experto, Creador, Alta Cohesión, Bajo Acoplamiento, Controlador, Fabricación Pura, Indirección y No Hables con Extraños, pero solo serán referenciados aquellos que estén vinculados directamente con el desarrollo del plugins Génesis.

[10] ***EMF**:* siglas del inglés *Eclipse Modelling Framework.*

- ✓ **Experto.**

La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos), donde una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada.

- ✓ **Creador.**

Este patrón como su nombre lo indica es el que crea, el guía la asignación de responsabilidades relacionadas con la creación de objetos, se asigna la responsabilidad de que una clase B cree un Objeto de la clase A solamente cuando:

1) B contiene a A
2) B es una agregación (o composición) de A
3) B almacena a A
4) B tiene los datos de inicialización de A (datos que requiere su constructor)
5) B usa a A.

- ✓ **Alta cohesión.**

La cohesión es una medida de la fuerza con la que se relacionan las clases y el grado de focalización de las responsabilidades de un elemento, Cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable, una clase con baja cohesión hace muchas cosas no relacionadas o hace demasiado trabajo.

- ✓ **Bajo acoplamiento.**

El acoplamiento es una medida de fuerza con que un elemento está a, tiene conocimiento de, confía en, otros elementos. Este patrón es un principio que asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad, etc.). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. Un error muy común es asignarle demasiada responsabilidad y alto nivel de acoplamiento con el resto de los componentes del sistema.

- ✓ **Controlador.**

Es un evento generado por actores externos, donde se asocian con operaciones del sistema como respuestas a los eventos del sistema, tal como se relacionan los mensajes y los métodos. Normalmente un controlador delega en otros objetos el trabajo que se necesita hacer; coordina o controla la actividad y no realiza mucho trabajo por sí mismo.

### 1.6.2. Patrones de diseño Gof (“Pandilla de los cuatro”)

Los patrones de diseño el grupo de GoF clasifican en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

- **Creacionales**: tratan con las formas de crear instancias de objetos. El objetivo de estos patrones es de abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.
  - ✓ **Singleton** (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.
- **Estructurales**: Los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.
  - ✓ **Facade** (Fachada): Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.
- **Comportamiento**: Los patrones de comportamiento ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.

## 1.7. Conclusiones parciales

Con el estudio de algunos sistemas que permiten la gestión de entidades existentes en el mundo y en Cuba se llega a la conclusión de que a pesar de la gran ventaja que representa el desarrollo de un sistema basado en modelos con la caracterización de algunos de estos sistemas se identificaron ciertas limitaciones lo cual hace de nuestra propuesta una solución más viable para algunas de estas dificultades; entre las que se encuentran:

- ✓ Muchos de estos sistemas solo cubren una parte del desarrollo del software ya que solo pueden utilizarse en las fases iniciales del desarrollo de los mismos, algunos de estos sistemas son por ejemplo Enterprise Architect, el cual es una herramienta comercial que se especializa en la gestión de requisitos pero solo se puede utilizar durante las fases de análisis y diseño, otros casos similares ocurre con BOUML, Poseidon, OpenAmeos.
- ✓ No se encontraron sistemas que cumplan y se adapten satisfactoriamente a la arquitectura propuesta por nuestro plugin, lo cual viene dado porque cada vez son más las herramientas MDA que se están

desarrollando pero cada una de ellas define su propia arquitectura de trabajo a la hora de especificar nuevas transformaciones.

- ✓ Muchos de estos sistemas tienen un alto nivel de dependencia con la plataforma o arquitectura para la cual se cree, esto imposibilita la reutilización de los mismos para dar solución a problemáticas similares en un ambiente diferente.

Por tanto se decidió elaborar un sistema multiplataforma, que facilite el proceso de desarrollo de modelos, que brinde facilidades de uso a los usuarios, este trabajo se considera un valioso aporte en el área del desarrollo basado en modelos.

En la confección del mismo se utilizará como metodología de desarrollo RUP, como lenguaje de programación Java, con auxilio del Eclipse para el desarrollo del módulo y el Visual Paradimg para el modelado de los diagramas de componente.

# Capítulo 2: Propuesta de Solución.

## 2.1. Introducción

En este capítulo se tratan los elementos inmersos en el proceso de construcción de software, guiado por la arquitectura y haciendo énfasis en sus características esenciales, así como la forma de dar cumplimiento a las funcionalidades identificadas. Se realiza una valoración de la propuesta del diseño del sistema exponiendo las principales ventajas y deficiencias del mismo. Provee además el conjunto de artefactos obtenidos a través de la aplicación de la metodología de desarrollo seleccionada, la interrelación entre cada uno de estos componentes y sus funciones específicas. Este diseño del software permite traducir los requisitos analizados de un sistema, tanto funcionales como no funcionales, en una representación del software, que inicialmente da una visión del mismo y tras posteriores refinamientos sirve como esquema para la implementación del sistema y es, de ese modo, utilizado como una entrada fundamental de las actividades de implementación.

## 2.2. Modelado del Sistema

La parte más difícil de construir un sistema es precisamente saber qué construir. Ninguna otra parte del trabajo conceptual es tan difícil como establecer los requerimientos técnicos detallados, incluyendo todas las interfaces con gente, máquinas y otros sistemas. Ninguna otra parte del trabajo afecta tanto el sistema si es hecha mal. Ninguna es tan difícil de corregir más adelante. Entonces, la tarea más importante que el ingeniero de software hace para el cliente es la extracción iterativa y el refinamiento de los requerimientos del producto. [Frederick P. Brooks, 1987].

### 2.2.1. Requisitos Funcionales.

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir, no alteran la funcionalidad del producto, o sea, se mantienen invariables sin importarle con que propiedades o cualidades se relacionen. Es decir, especifican el comportamiento de entrada y salida del sistema y surgen de la razón fundamental de la existencia del producto.

1. **Crear Entidad**

El sistema debe brindar la posibilidad de crear una entidad del sistema en cuestión. Para ello se muestra una interfaz con los datos iniciales necesarios que la identifican. La entidad creada debe ser salvada y debe ser posible editar dicha entidad en cualquier momento (Anexo1).

2. **Editar Entidad**

En esta opción el sistema debe permitir editar una entidad del sistema. Las entidades pueden ser o no persistentes y esto se explica debido que algunas no son utilizadas para almacenar datos en al BD sino para participar en la intercambio de datos entre el cliente y el servidor. En este CU se muestra una interfaz con los datos necesarios y además se explican las reglas de validación que deben cumplirse en toda entidad. El objeto editado debe ser entregado al generador de código para que genere o modifique las tablas de la BD implicadas y la clase persistente en el lenguaje deseado. **(**Anexo2**).**

3. **Crear Método**

El propósito del CU es crear un método en el sistema. Este es un caso de uso extendido que lo utilizan otros casos de uso de ahí su separación hacia en un lugar común. Ejemplos de uso de este CU pueden ser cuando estamos creando una entidad del sistema y se necesita asignarle un método. O cuando necesitamos el cumplimiento de una regla genérica o crear un método en la capa negocio. En el caso de uso se describen los datos de un método y las validaciones necesarias. (Anexo3)

### 2.2.2. Requisitos no funcionales.

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Las propiedades son las características que hacen al producto atractivo, usable, rápido y confiable. Normalmente, están vinculados a requisitos funcionales. Una vez que se conoce lo que el sistema debe hacer, se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser. Los requisitos no funcionales son fundamentales en el éxito del producto.

*Requisitos de Software.*

- ✓ Los requisitos mínimos de software necesarios son: una computadora personal con plataforma del sistema operativo Microsoft Windows 2000 o superior.
- ✓ Para la ejecución de la aplicación, debe instalarse la Máquina Virtual de Java.

*Requisitos de Hardware.*

- ✓ Las condiciones mínimas de hardware son: que las computadoras tengan procesador Intel Celeron a 3.06 GHz, superior o equivalente, disco duro de 40 GB y 256 MB de RAM.
- ✓ Se requiere de una impresora estándar para la impresión de los reportes que solicite el usuario.

*Requisitos en el diseño y la implementación.*

Para organizar el Análisis y Diseño del sistema se utilizará:

- ✓ La metodología RUP.
- ✓ UML como lenguaje de modelado.
- ✓ Visual Paradigm como herramienta CASE.
- ✓ Java será el lenguaje de programación a ser usado para la implementación.
- ✓ Los nombres de los métodos y de las clases deben ser lo más sencillo posible para un mejor entendimiento entre el equipo de desarrolladores.

*Requisitos de apariencia o interfaz externa.*

- ✓ El sistema debe contar con una interfaz fácil, amigable y sencilla que permita a los usuarios finales interactuar con el sistema, aun teniendo conocimientos básicos.
- ✓ El tamaño y el tipo de letra deberá ser uniforme para todas las interfaces.
- ✓ Las interfaces evitarán cargar diálogos, paneles o ventanas innecesarias.
- ✓ Los iconos y botones que se utilicen en las interfaces, deben de estar relacionados con la función a realizar.
- ✓ La información a brindar debe estar vinculada totalmente con el tema a tratar, a fin de evitar el entretenimiento innecesario del usuario.
- ✓ El tamaño de los campos debe corresponderse con el tamaño de la información a tratar.

*Requisitos de usabilidad.*

- ✓ Los usuarios que van a utilizar el producto deberán tener algún conocimiento previo del manejo de una computadora personal.
- ✓ Se impartirán clases a los usuarios finales, con el fin de explicar cómo se debe trabajar en el sistema.

- ✓ El software brindará siempre la posibilidad de consultar la ayuda disponible para el usuario, permitiendo un avance considerable en el trabajo con la aplicación.

*Requisitos de rendimiento.*

- ✓ Los tiempos de respuesta y velocidad de procesamiento de la información serán rápidos, para lograr un producto eficiente y que cumpla con la gestión de la información que se necesita.

*Requisitos de soporte.*

- ✓ Realizar pruebas a la aplicación cuando se haya concluido.
- ✓ La aplicación será portable a diferentes Sistemas Operativos (SO).

*Requisitos legales.*

- ✓ La herramienta CASE seleccionada: Visual Paradigm, aunque es propietaria, en estos momentos la UCI tiene la licencia para su uso.
- ✓ El lenguaje seleccionado para la implementación: Java es software libre.

## 2.3. Arquitectura de la solución

La arquitectura de software es la organización fundamental de un sistema descrita en sus componentes, relación entre ellos y con el ambiente y principios que guían su diseño y evolución, donde se establecen los fundamentos para que analistas, diseñadores, programadores, etc. trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades.

La arquitectura del módulo está estructurada por un patrón de n-capas, el cual cuenta con 4 capas lógicas que dan un alto nivel de encapsulamiento de las responsabilidades, permitiendo reducir al máximo el acoplamiento y aumentar la reutilización entre las mismas. Una vez que estas estén bien definidas la comunicación entre ellas se realizará solo a nivel de interfaces que permiten trabajar de manera transparente a las instancias reales; gracias a esto se realiza una arquitectura escalable.

### 2.3.1. Vista lógica del Módulo Entidades

El modelo de capas de una arquitectura organiza el sistema en capas, cada una de las cuales proporciona un conjunto de servicios. Este tipo de arquitectura soporta el desarrollo incremental de sistemas y evita que los cambios en una de las capas afecten directamente al resto.

La idea de hacer una separación en capas es que cada una de las mismas cumpla con un rol y tenga responsabilidades bien definidas. Una aplicación puede tener una, dos, tres o N capas lógicas dentro de

un mismo equipo físico, esto queda a criterio de la arquitectura que se decida utilizar, en nuestro caso el módulo Entidades opto por combinar 4 capas esenciales, donde cada nivel se especializa en una actividad específica para evitar la dependencia a una sola tecnología o aplicación. Aunque MVC se manifiesta habitualmente en aplicaciones web, nuestro plugin hace referencia a este tipo de arquitectura.

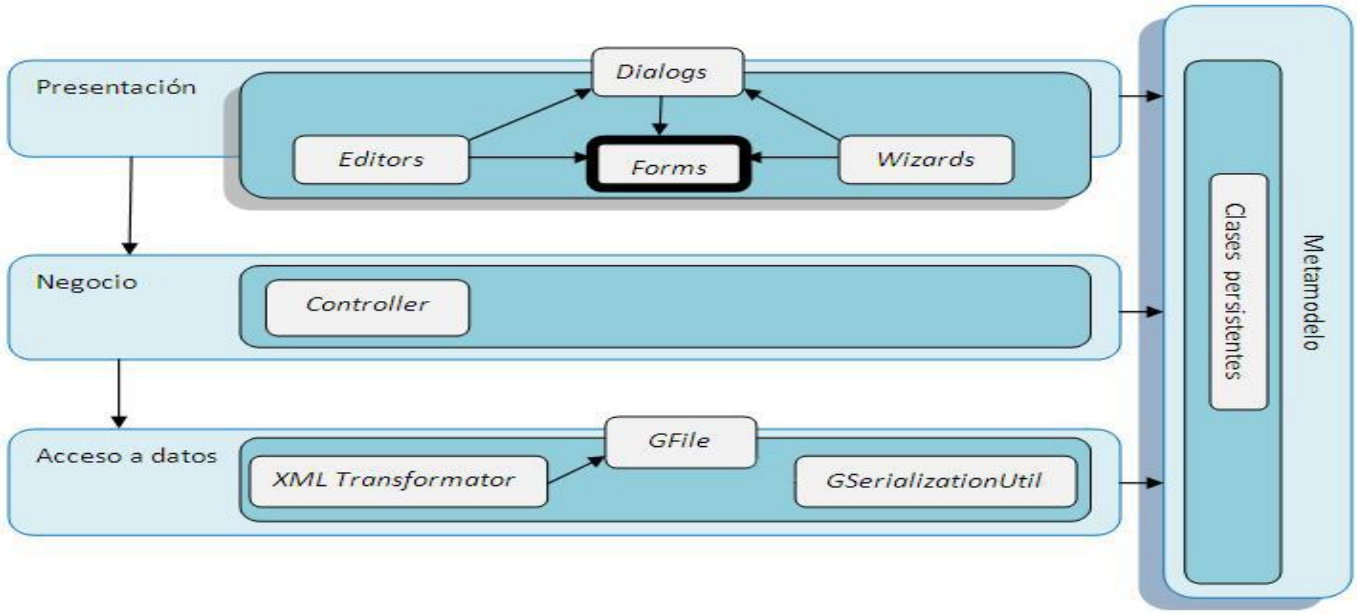



**Figura 3 Capas del módulo Entidades.**

A continuación se da a conocer algunas de las características más esenciales de cada capa:

- ✓ **Presentación:** trata sobre los aspectos gráficos de la aplicación, es donde se realiza la interacción con del usuario con el sistema. Contiene los requerimientos y la lógica de la interfaz de usuario dentro de una capa separada que permite tanto su reutilización como su independencia de la lógica de negocio, permitiendo que se pueda modificar la interfaz sin que esto afecte al negocio y viceversa. Está compuesta por todas las interfaces de usuario y los componentes necesarios para su correcto funcionamiento.
- ✓ **Negocio**: esta capa reúne todos los aspectos del software que automatizan o apoyan los procesos de negocio que llevan a cabo los usuarios. Estos aspectos típicamente incluyen las tareas que forman parte de los procesos, las reglas y restricciones que aplican. (Es donde se ejecutan todas las reglas de negocio). Por otra parte en esta capa se recogen todos los servicios necesarios para darle solución a los requisitos de negocio que no pueden ser satisfechos por el metamodelo. Tienen la responsabilidad

de manejar todas las operaciones sobre una entidad de negocio en específico, así como todas las entidades que por conceptos de composición se encuentran relacionadas con esta; además de contener las clases controladoras, que son las que manejan todas las operaciones sobre las entidades de negocio definidas.

- ✓ **Metamodelo:** Ofrece servicios de persistencia y recuperación de información a las capas superiores; la persistencia permite al programador almacenar, transferir y recuperar el estado de los objetos; el metamodelo describe las entidades que participan en el sistema, las relaciones y el flujo de datos que existe entre ellas, dichas entidades se mapean en clases que se componen de propiedades y métodos. Los objetos del modelo de dominio no deben contener código para cargar el estado de base de datos ni tampoco para guardarlo, dicho código debe declararse en otras clases de la capa de negocio, una buena práctica para unificar este código es usar el patrón repositorio, el cual consiste en crear una capa de abstracción que actúe de puente entre la capa de datos y la capa de negocio. Las clases repositorio, deben poder invocarse desde la capa de presentación, y normalmente tenemos una por cada objeto.
- ✓ **Acceso a datos:** esta capa reúne todos los aspectos del software que tienen que ver con el manejo de los datos persistentes o sea, se ocupa de obtener y persistir los datos. La capa de acceso a datos está directamente relacionada con los servicios definidos en el negocio. Su principal función es realizar una implementación de las interfaces definidas en la capa de negocio y al mismo tiempo trabajar directamente con la fuentes de datos establecidas en el metamodelo. Contiene las clases entidades que representan lo que se gestiona en la aplicación.

### 2.3.2. Patrón Modelo-Vista-Controlador (MVC)

Este patrón arquitectónico separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes, como se representa en la figura 7:

- **Modelo:** Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- **Vista:** Maneja la visualización de la información.
- **Controlador:** Controla el flujo entre la vista y el modelo (los datos).

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo, independientemente de la representación visual.

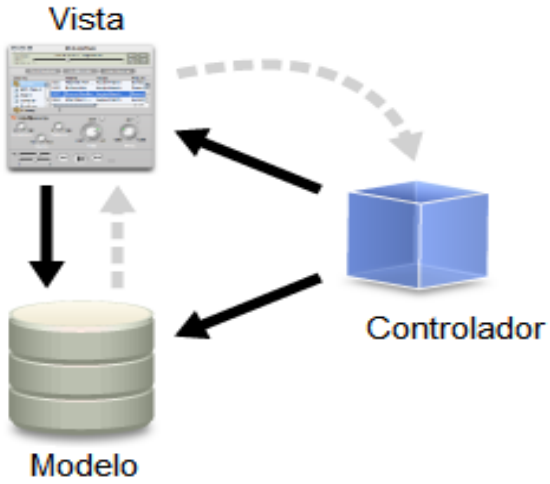



**Figura 4 Representación del patrón MVC.**

Entre las ventajas del estilo Modelo-Vista-Controlador están las siguientes:

Soporte de múltiples vistas: Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Por ejemplo, múltiples páginas de una aplicación Web pueden utilizar el mismo modelo de objetos mostrado de maneras diferentes.

Adaptación al cambio: Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares o PDAs[11]. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo.

Una desventaja que tiene este modelo es el costo de actualizaciones frecuentes: Si el modelo experimenta cambios frecuentes, por ejemplo, podría desbordar las vistas con una lluvia de requerimientos de actualización.

### 2.4. Modelo de Diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de usos centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tiene un impacto en el sistema a considerar. Además, el

[11] ***PDAs:*** *Asistente Digital Personal (del inglés Personal Digital Assistant).*

modelo de diseño sirve de abstracción de la implementación del sistema y es, de ese modo, utilizada como entrada fundamental de las actividades de implementación.

### 2.4.1. Estructura

El plugin Génesis se ha desarrollado con una estructura que permite el encapsulamiento en paquetes que se encuentran relacionadas para dar solución a un requerimiento en particular. Se caracteriza por contener elementos que permiten una implementación de forma gráfica lo más cercana posible al modelo del proceso de negocio.

### 2.4.2. Modelo de paquetes

Un paquete de diseño es usado para estructurar el modelo de diseño mediante su división en partes más pequeñas. Los paquetes son usados principalmente para administración de configuración y organización del modelo. Es de gran ayuda a la hora de realizar el diseño de un software el organizar por paquetes las clases y ficheros de un módulo.

El modelo de paquete del módulo Entidades del plugin Génesis se encuentra estructurado de la siguiente forma:

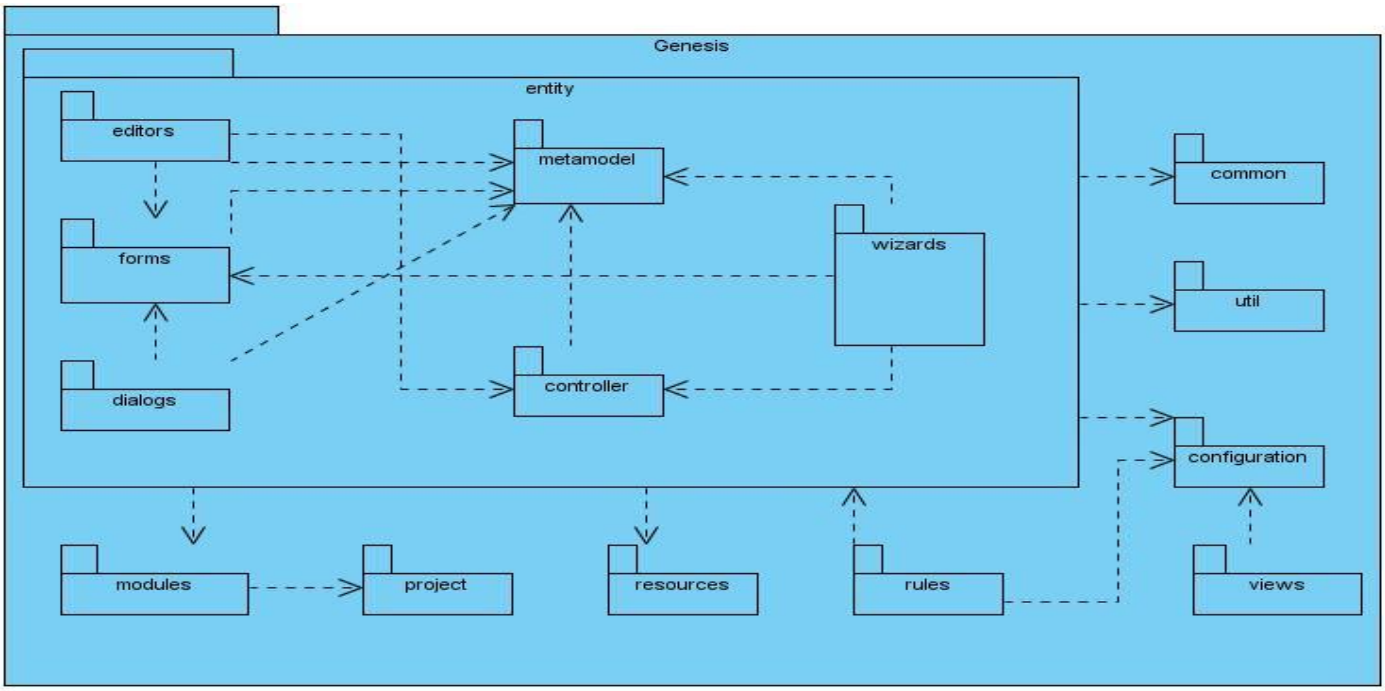



**Figura 5 Diagrama de paquetes del módulo Entidad del plugin Génesis**

*Paquete common:* contiene las clases que será utilizada indistintamente por cada módulo del sistema.

*Paquete configuración:* contiene las clases que describen los tipos de datos de las clases del dominio.

*Paquete entity*: contiene los paquetes necesarios para gestionar la creación de una entidad.

*Paquete rules*: contiene los paquetes necesarios para gestionar los diferentes tipos de validación que se pueden realizar sobre una entidad.

*Paquete views*: contiene los paquetes necesarios que actúan sobre una entidad para gestionar la creación de las vistas correspondientes.

*Paquete modules*: contiene las clases necesarias gestionar los editores en cada módulo.

*Paquete project:* contiene las clases necesarias para crear un proyecto.

*Paquete forms:* en este paquete se encuentran las interfaces e implementaciones del módulo correspondiente

*Paquete metamodel:* en este paquete están las clases del dominio del módulo.

*Paquete editors:* contiene clases que representan objetos a manipular en los formularios.

*Paquete wizards:* contiene las clases necesarias para gestionar la creación de un objeto por etapas.

*Paquete dialogs:* implementa las clases que hacen referencia al objeto a editar en el formulario que contienen.

*Paquete útil:* en este paquete están las clases encargadas de manejar el objeto creado

- GSerializationUtils: clona el objeto a modificar hasta terminar la operación y entonces persiste los datos.
- GFile: contiene el objeto persistido a transformar en un fichero xml y salvar.
- XML Transformator: transforma el objeto a un fichero xml.

*Paquete resources:* contiene las imágenes utilizadas en las interfaces de los diferentes módulos.
*Paquete Controller:* contiene las clases necesarias que controlan la creación de un objeto.

### 2.4.3. Diseño de clases

En el diseño de clases se resume la definición de las clases que se pueden implementar en el software, se visualizan las relaciones entre ellas, y se muestra gráficamente la interacción de los objetos para comunicarse entre sí.

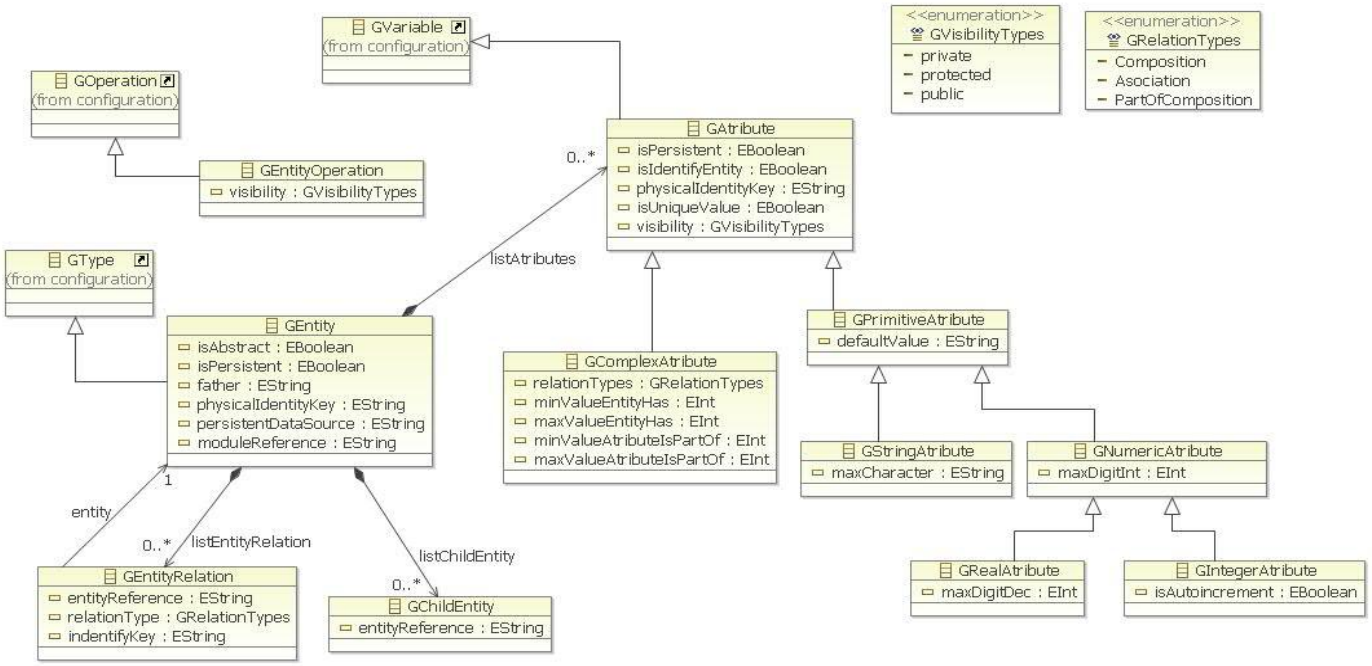



**Figura 6 Diagrama de clases del diseño del módulo Entidades del plugin Génesis.**

*Diagrama de Clases*

El diagrama de clases contiene una definición parcial de software de las clases y de las interfaces en una aplicación, además de las asociaciones y de los atributos básicos, el siguiente diagrama se ha ampliado para incluir los métodos de cada clase, la información sobre el tipo de dato de los atributos, su visibilidad y la navegación entre objetos.

### 2.4.4. Diagrama de interacción: diagrama de secuencia

Los diagramas de interacción se utilizan para modelar los aspectos dinámicos de un sistema, lo que conlleva modelar instancias concretas o prototípicas de clases interfaces, componentes y nodos, junto con los mensajes enviados entre ellos, todo en el escenario que ilustra un comportamiento determinado. En el contexto de las clases se describe la forma en que grupos de objetos colaboran para proveer un

comportamiento. Mientras que un diagrama de casos de uso presenta una visión externa del sistema, la funcionalidad de dichos casos de uso se recoge como un flujo de eventos utilizando para ello interacciones entre sociedades de objetos.

Los diagramas de interacción son diagramas que describen cómo grupo de objetos colaboran para conseguir algún fin. Estos diagramas muestran objetos, así como los mensajes que pasan entre ello dentro del caso de uso

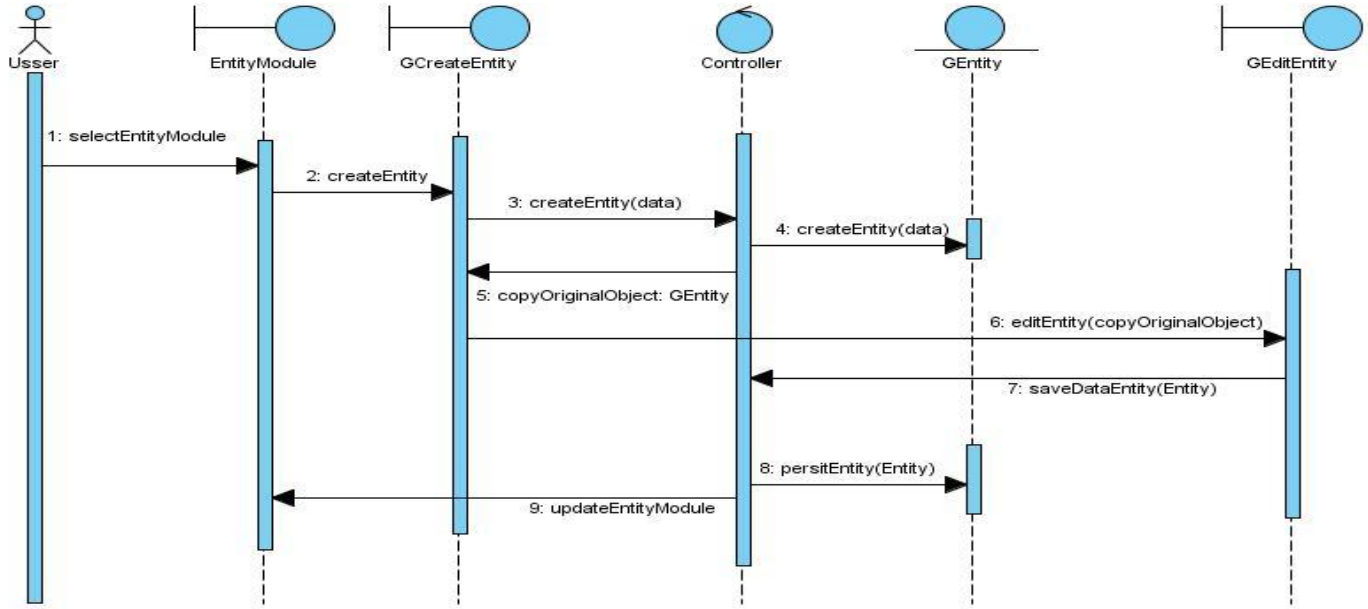



**Figura 7 Diagrama de secuencia del CU Crear Entidad.**

### 2.4.5. Diagrama de clases persistentes (Metamodelo)

La persistencia es la capacidad de un objeto de mantener su valor en el espacio y en el tiempo. Las clases persistentes por lo general tienen como origen las clases entidad que modelan la información y el comportamiento asociado de algún fenómeno o concepto, como un objeto del mundo real o un suceso. El diagrama de clases persistentes cuenta con todas la clases del modelo de datos incluyendo sus atributos, métodos y las relaciones que se establecen entre ellos. A continuación se muestra el diagrama de clases persistentes correspondientes a cada módulo.

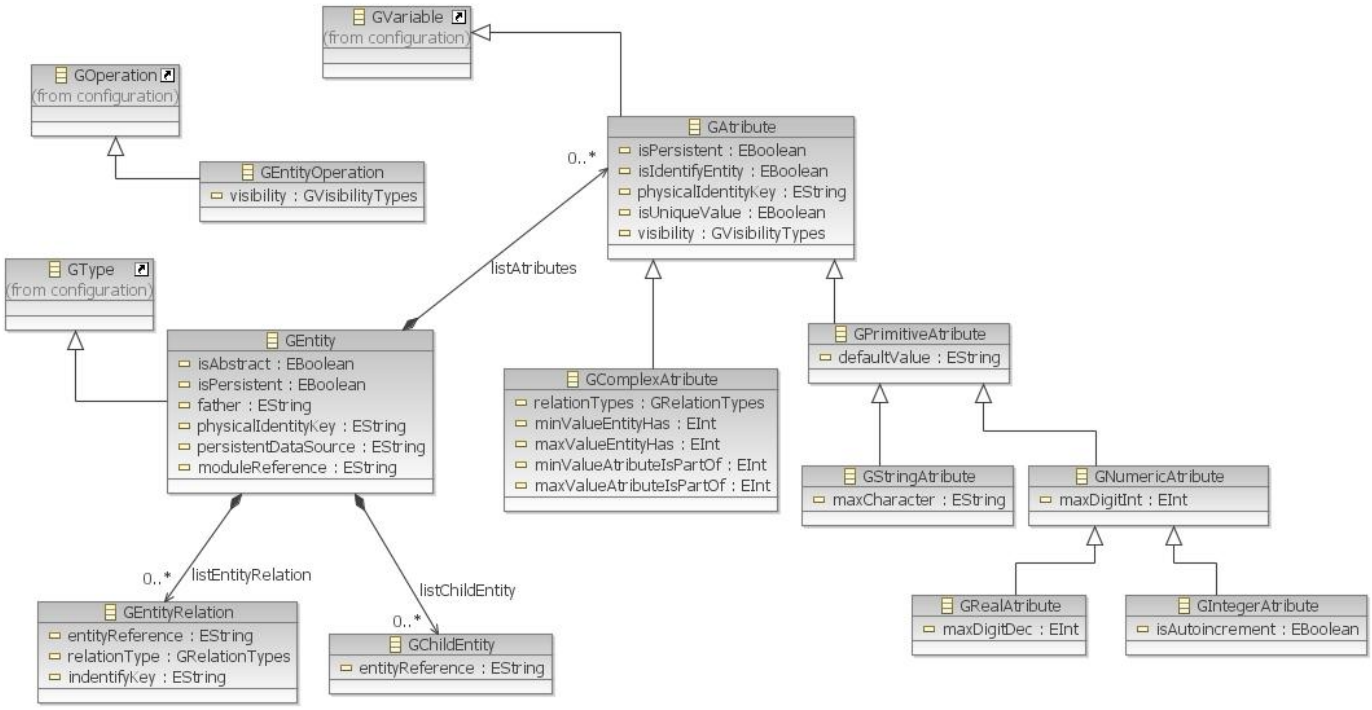



**Figura 8 Diagrama de clases del dominio del módulo Entidades. (Metamodelo).**

### 2.5. Patrones de diseño utilizados.

Se puede entender como un patrón de diseño a:

- una solución estándar para un problema común de programación
- una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios
- un proyecto o estructura de implementación que logra una finalidad determinada
- un lenguaje de programación de alto nivel
- una manera más práctica de describir ciertos aspectos de la organización de un programa
- las conexiones entre componentes de programas
- la forma de un diagrama de objeto o de un modelo de objeto.

Para el desarrollo del sistema en cuestión se han tenido en cuenta un conjunto de patrones que permiten darle flexibilidad y no constituyen grandes cambios en el rendimiento del mismo, además se identificaron

un conjunto de patrones específicos para el desarrollo de Workflows que brindan claridad y fortaleza a los diseños de estos.

### 2.5.1. Patrones de diseño GRASP.

- ✓ **Experto**

Este patrón se encarga de asignar una responsabilidad al experto en información, es decir, la clase que cuenta con la información necesaria para cumplir la responsabilidad, por lo que fue utilizado en el diseño del módulo de manera general ya que cada método se implementa en la clase que conoce toda la información relacionado con este.

- ✓ **Creador**

Se brinda soporte a un bajo acoplamiento; esto supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización.

- ✓ **Bajo Acoplamiento**

Un Alto Acoplamiento significa que una clase recurre a muchas otras, esto conlleva a que los cambios de las clases afines ocasionan cambios locales, por lo tanto, Bajo Acoplamiento significa asignar una responsabilidad para mantener pocas dependencias entre las clases, un ejemplo de esto se puede ver en la clase GEntityController.

- ✓ **Alta cohesión**

Este patrón fue utilizado en el diseño del módulo de manera general ya que cada clase cuenta con la cantidad mínima de operaciones necesarias para realizar su funcionalidad.

- ✓ **Controlador**

Este patrón se encarga de asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas, un ejemplo de la aplicación de este patrón lo constituye la clase controladora del módulo, la cual tendrá la responsabilidad de escuchar y responder a las peticiones realizadas por la capa de presentación y de comunicarse con la capa de negocio.

### 2.5.2. Patrones de Creación.

- ✓ **Singleton**

Se asegura que solo se pueda crear una instancia de la clase y ofrece un punto global de acceso a esta instancia. El uso de este patrón permite que los servicios puedan ser creados solo una vez, un ejemplo del uso de dicho patrón se puede ver en la clase GEntityController.

## 2.6. Conclusiones parciales.

En este capítulo se describió como la arquitectura de software es uno de los elementos más importantes en el desarrollo de un sistema informático debido a que establece la estructura que debe tener el módulo a desarrollar y guía todas las actividades de implementación del mismo. La descripción de clases importantes permitió tener una visión de la implementación realizada. De forma general se logró implementar los requerimientos definidos por los analistas. Al finalizar queda implementado el sistema y descritas todas las clases utilizadas, cumpliendo con los objetivos específicos y gran parte de las tareas propuestas para la elaboración de este, dándole una respuesta satisfactoria al problema científico planteado, esto influye de manera positiva, pues se logra a la hora de implementar la solución propuesta, una mayor organización.

# Capítulo 3: Implementación y pruebas

## 3.1. Introducción

La implementación es el flujo de trabajo en el que se implementa el sistema en términos de componentes: ejecutables, ficheros de código fuente, scripts, entre otros. Tiene como objetivo principal desarrollar la arquitectura y el sistema como un todo, así como definir la organización del código, en este capítulo se definirá cómo quedará dispuesta la misma; se implementarán las clases y servicios encontrados durante el diseño y se obtendrán los artefactos correspondientes como son el diagrama de componentes y de despliegue; además de diseñarse los casos de las pruebas que se le harán al sistema, en los que las posibilidades de que aparezcan fallos humanos son muy grandes. Los errores pueden presentarse debido a especificaciones erróneas e imperfectas de los requisitos, uso indebido de las estructuras de datos, errores al integrar módulos, entre otras causas. Dado a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo del software ha de ir acompañado de una actividad que garantice la calidad.

## 3.2. Estándares de codificación

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código (24).

El usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento. Además, si se aplica de forma continua un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas, y, posteriormente, se efectúan revisiones del código de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

### 3.2.1. Estilos para la capitalización

Se podrán utilizar los siguientes tres convenios para la capitalización de los identificadores:

- ✓ **Pascal**

La primera letra en el identificador y la primera letra de cada subsiguiente palabra concatenada se capitalizan. Puede utilizar los identificadores de Pascal en caso de tres o más caracteres.
Por ejemplo: *GEntityForm.*

- ✓ **Camello**

La primera letra en el identificador está en minúscula y la primera letra de cada subsiguiente palabra concatenada es mayúscula. Por ejemplo: *gParameters*

- ✓ **Mayúscula**

Todas las letras en el identificador se capitalizan. Esta convención se utilizará solo para los identificadores que constan de dos o menos letras. Por ejemplo: *GEntity, GAtribute*

- ✓ **Sensibilidad a mayúsculas**
  - No se deberá utilizar nombres o identificadores que requieran ser *case sensitivity*[12]
  - No se deberá crear dos *namespaces*[13] que se diferencien solo en el uso de las mayúsculas. Por ejemplo:

    *Report.entities.Reportcontrols*

    *Report.Entities.ReportControls*
  - No crear funciones con nombres de parámetros que se diferencian solo en el uso de la mayúscula. Por ejemplo:

    *createGObject(GEntity GEntity,GEntity gEntity)*
  - No se deberá crear *namespaces* con nombres de clases que se diferencien solo en el uso de las mayúsculas. Por ejemplo:

    *Report.Entities*

    *Report.ENTITIES*
  - No crear clases con propiedades que se diferencien solo en el uso de las mayúsculas. Por ejemplo:

    *string Description{get, set}*

    *string DESCRIPTION{get, set}*

[12] *Sensible a las mayúsculas o minúsculas.*

[13] *De la definición en inglés de espacio de nombres*

- No crear clases con métodos que se diferencien solo en el uso de las mayúsculas.

✓ **Evitando confusión de nombre y tipo**

- Utiliza nombres que describan a sus identificadores en vez de nombres que describen el tipo de identificador.

  Por ejemplo: void Write(float value)

Con el objetivo de incrementar la legibilidad del código también se emplearon comentarios en todas las declaraciones de clases y funciones más complejas. También se organizó el código de forma estructurada, en bloques de código, para una mejor lectura del mismo.

### 3.3. Tratamiento de errores

El diseño de un sistema no solo debe tener en cuenta lo que debe ocurrir, sino que además debe realizar un análisis profundo de las diferentes situaciones que se puedan presentar y que constituyen algún tipo de violación o de situación en particular que provocaría un error dentro del sistema.

Con el fin de lograr una mayor integridad y confiabilidad en los datos que se introducen en el sistema se adoptaron estrategias para el tratamiento de errores tales como:

- ✓ Se Comprueba que no se deja en blanco un campo obligatorio como se muestra en la figura 9.
- ✓ Se comprueba que la entrada de datos siga un patrón definido como una expresión regular, este tipo de validación nos permite comprobar secuencias predecibles de caracteres, ya sean cadenas de caracteres como valores numéricos, entre otros como se muestra en la figura 10.
- ✓ No se permite ejecutar ninguna operación mientras existan errores en la página.

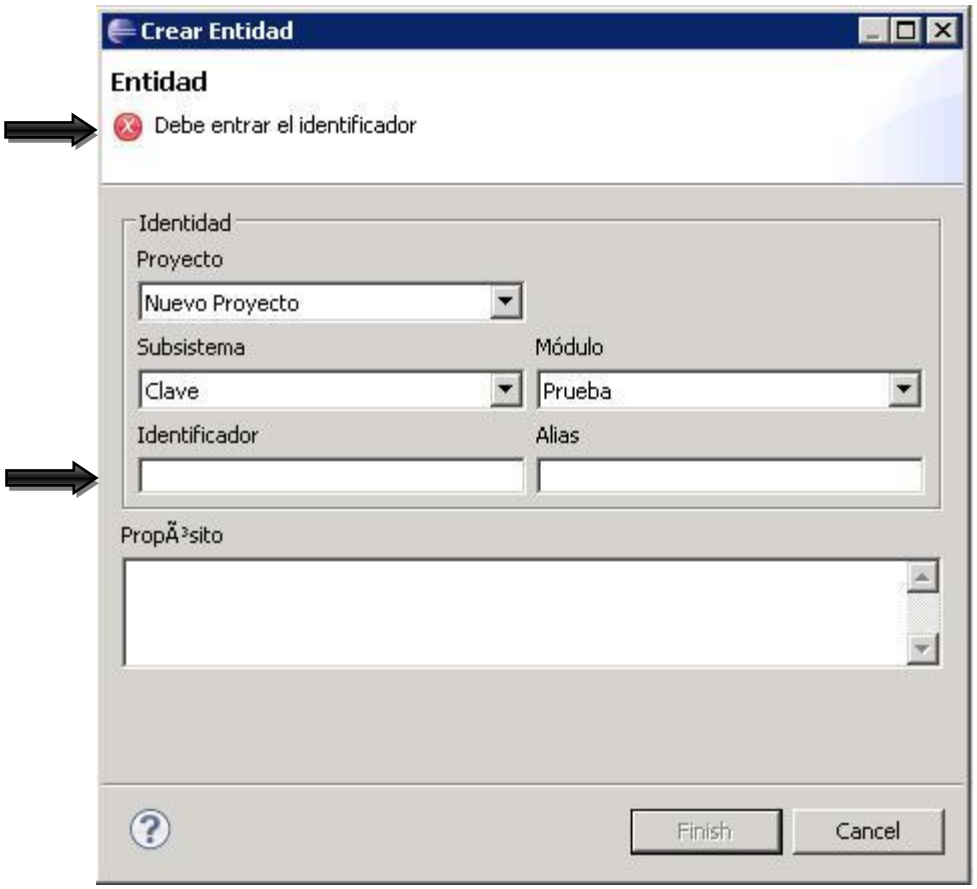



**Figura 9 Dejar en blanco campos obligatorios.**

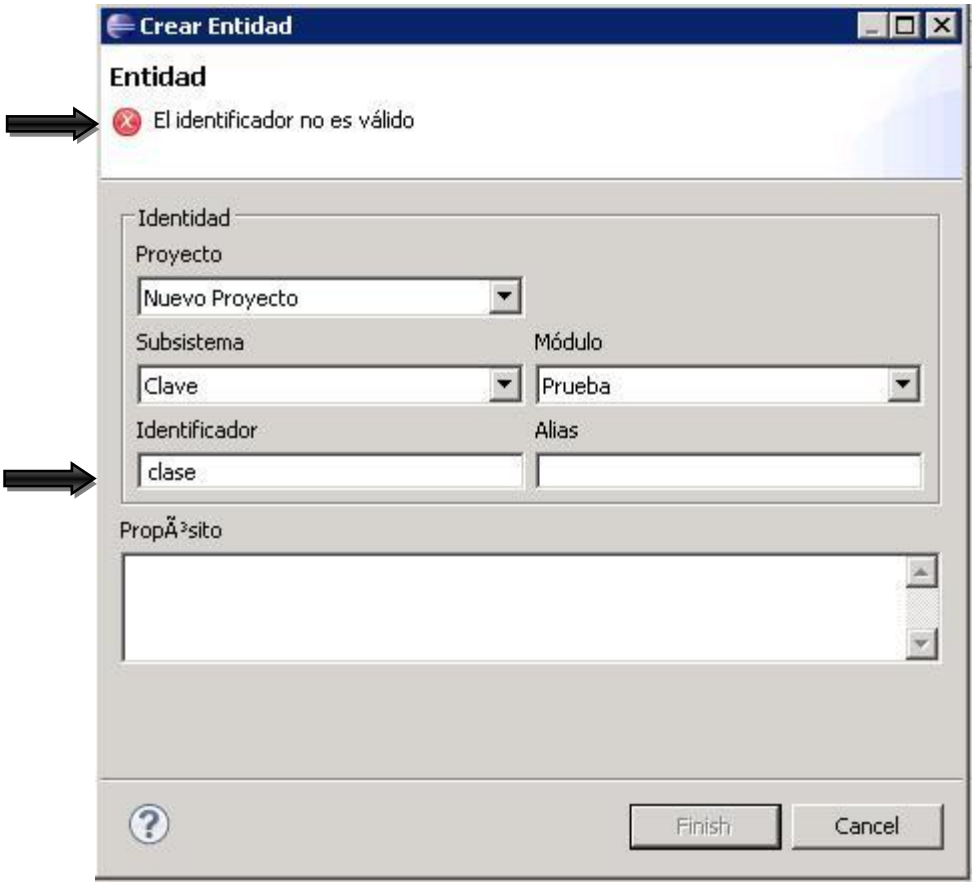



**Figura 10 Comprobar entrada de datos.**

### 3.4. Diagrama de componente

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes. Los diagramas de Componentes prevalecen en el campo de la arquitectura de software pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema. (21)

*¿Por qué utilizar un Diagrama de Componentes?*

- ✓ Nos permite ver el modelado de un sistema o subsistema
- ✓ permite especificar un componente con interfaces bien definidas.

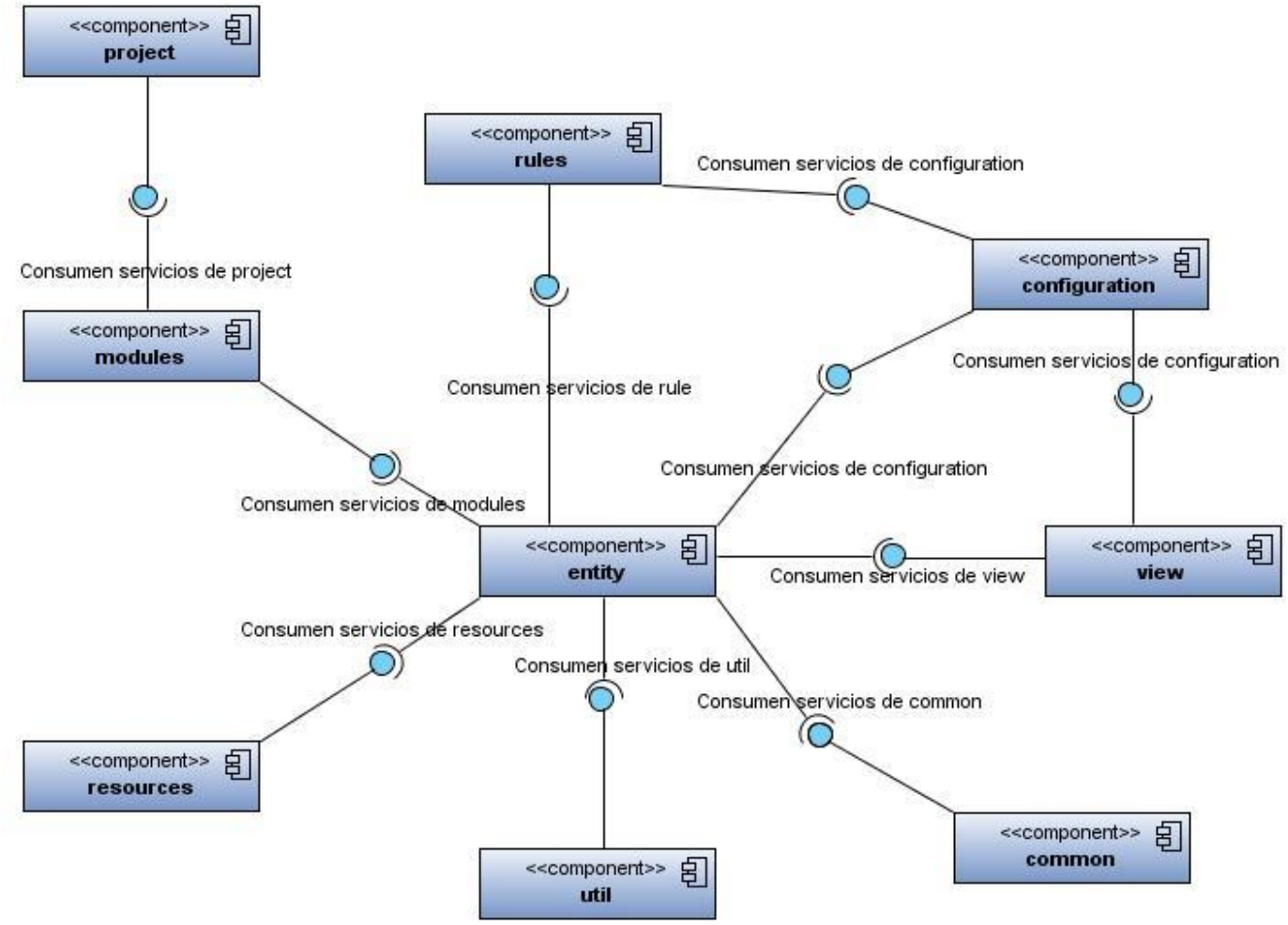



**Figura 11 Diagrama de componentes del módulo Entidades del plugin Génesis.**

### 3.5. Diagrama de despliegue

El diagrama de despliegue es un tipo de diagrama del Lenguaje Unificado de Modelado que se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes. (21)

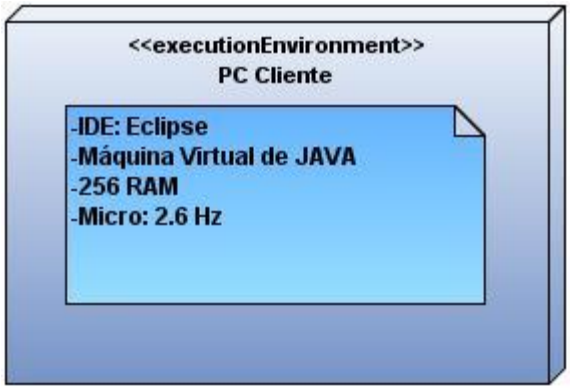



**Figura 12 Diagrama de despliegue del plugin Génesis.**

### 3.6. Pruebas de Software

Las pruebas de software son el conjunto de técnicas que permiten determinar la calidad de un producto. Estas se integran dentro de las diferentes fases del ciclo de vida del software dentro de la Ingeniería. Así se ejecuta un programa y mediante técnicas experimentales se trata de descubrir que errores presenta. La calidad de un sistema informático es algo subjetivo que depende del contexto y del objeto que se pretenda conseguir. Para determinar dicho nivel de calidad se deben efectuar medidas o pruebas que permitan comprobar el grado de cumplimiento con respecto a las especificaciones iniciales del sistema.

#### 3.6.1. Objetivos de las Pruebas

El objetivo de las pruebas de un programa es el de detectar todo posible mal funcionamiento, un error puede ser costoso de reparar mientras más se avanza en las etapas del ciclo de vida del software. Para minimizar estos riesgos se crean baterías de pruebas que serán de mayor calidad en la medida de cuantos menos errores queden por descubrir tras haberla pasado y viceversa, si un programa aún tiene muchos fallos tras haberla superado, se dirá que esta es de poca calidad. Si se pudiera probar un programa con todos los posibles datos de entrada, se tendría una batería de pruebas perfecta, pues no hay lugar para las sorpresas. Lamentablemente, casi nunca es posible probar con todos los casos. En consecuencia se necesita un criterio para elegir qué casos se prueban.

#### 3.6.2. Técnicas de Diseño de Pruebas

Existen tres enfoques principales para el diseño de casos:

1. El enfoque estructural o de caja blanca: que se basan en un minucioso examen de los detalles procedimentales del código a evaluar, por lo que es necesario conocer la lógica del programa.

2. El enfoque funcional o de caja negra: que realizan pruebas sobre la interfaz del programa a probar, entendiendo por interfaz las entradas y salidas de dicho programa. No es necesario conocer la lógica del programa, únicamente la funcionalidad que debe realizar.
3. El enfoque aleatorio: consiste en utilizar modelos (en muchas ocasiones estadísticos) que representen las posibles entradas al programa para crear a partir de ellos los casos de prueba caja negra.
La Figura 10 representa gráficamente la filosofía de las pruebas de caja blanca y caja negra. En este caso profundizaremos en las pruebas de caja blanca teniendo en cuenta que fue seleccionada esta técnica para validar la implementación del módulo.

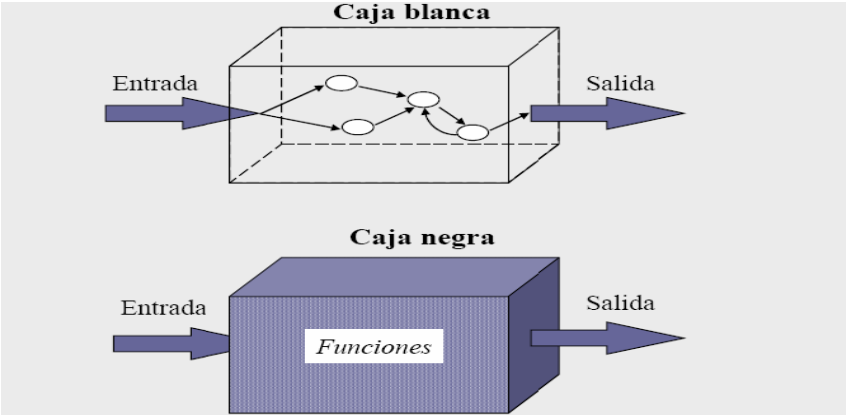



**Figura 13 Representación de pruebas de Caja Blanca y Caja Negra.**

### 3.6.3. Prueba de Caja Negra o Funcionales

El Método de Caja Negra: se refiere a las pruebas que se llevan a cabo sobre la interfaz del *software*. O sea, los casos de prueba pretenden demostrar que las funciones del *software* son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene. La prueba de caja negra son aquellas que se llevan a cabo sobre la interfaz del *software*. Estos casos de prueba pretenden:

- Demostrar las funciones del *software* son operativas.
- Que las entradas se aceptan de la forma adecuada y que se produce el resultado correcto.

A nuestra aplicación no se le realizaron las pruebas utilizando el método de caja blanca porque casi todos los Requisitos Funcionales son Gestionar y los métodos no tienen un nivel de complejidad alto para realizarle dicha prueba.

#### 3.6.3.1. Casos de pruebas

| Nombre de la sección | Escenarios de la sección | Descripción de la funcionalidad | Flujo central |
|---|---|---|---|
| SC1: Crear Entidad | EC1.1: Crear entidad correctamente | Se crea una nueva entidad | - Seleccionar la opción “Nueva Entidad”<br>- Llenar los campos<br>- Seleccionar “Finish” |
| | EC1.2: Crear una entidad con campos en blanco | No se crea una nueva entidad | - Llenar los campos<br>- Se muestra un mensaje en el campo donde se cometió el error.<br>- Seleccionar “Finish” |
| | EC1.3: Crear entidad incorrectamente | No se crea una nueva entidad | - Seleccionar la opción “Nueva Entidad”<br>- Llenar los campos con datos incorrectos<br>- Se muestra un mensaje indicando que los campos están incorrectos.<br>- Seleccionar “Finish” |
| | EC1.4: Cancelar la operación | No se crea una nueva entidad | - Seleccionar la opción “Nueva Entidad”<br>- Llenar los campos<br>- Seleccionar “Cancelar” |

**Tabla 1 CP Crear Entidad.**

| Nombre de la sección | Escenarios de la sección | Descripción de la funcionalidad | Flujo central |
|---|---|---|---|
| SC1: Editar Entidad | EC1.1: Editar entidad correctamente | Se edita una entidad del sistema | - Seleccionar la entidad que se desea editar<br>- Luego hacer doble click en dicha entidad o presionar “Enter”<br>- Llenar los campos<br>- Seleccionar “Aceptar” |
| | EC1.2: Editar una entidad con campos en blanco | No se edita una entidad del sistema | - Llenar los campos<br>- Se muestra un mensaje en el campo donde se cometió el error.<br>- Seleccionar “Aceptar” |

| | | | |
|---|---|---|---|
| | EC1.3: Editar entidad incorrectamente | No se edita una entidad del sistema | - Llenar los campos con datos incorrectos<br>- Se muestra un mensaje indicando que los campos están incorrectos.<br>- Seleccionar “Aceptar” |
| | EC1.4: Cancelar la operación | No se edita una entidad del sistema | - Llenar los campos<br>- Seleccionar “Cancelar” |

**Tabla 2 CP Editar Entidad.**

| Nombre de la sección | Escenarios de la sección | Descripción de la funcionalidad | Flujo central |
|---|---|---|---|
| SC1: Crear Atributos | EC1.1: Crear atributos correctamente | Se crea un atributo de la entidad | - Seleccionar la entidad que se desea editar<br>- Luego hacer doble click en dicha entidad o presionar “Enter”<br>- Seleccionar la opción de adicionar atributos<br>- Llenar los campos<br>- Seleccionar “Aceptar” |
| | EC1.2: Crear atributos con campos en blanco | No se crea un atributo de la entidad | - Llenar los campos<br>- Se muestra un mensaje en el campo donde se cometió el error.<br>- Seleccionar “Aceptar” |
| | EC1.3: Crear atributos incorrectamente | No se crea un atributo de la entidad | - Llenar los campos con datos incorrectos<br>- Se muestra un mensaje indicando que los campos están incorrectos.<br>- Seleccionar “Aceptar” |
| | EC1.4: Cancelar la operación | No se crea un atributo de la entidad | - Llenar los campos<br>- Seleccionar “Cancelar” |

**Tabla 3 CP Crear Atriobuto.**

| Nombre de la sección | Escenarios de la sección | Descripción de la funcionalidad | Flujo central |
|---|---|---|---|
| SC1: Crear Métodos | EC1.1: Crear métodos correctamente | Se crea un método de la entidad | - Seleccionar la entidad que se desea editar<br>- Luego hacer doble click en dicha entidad o presionar “Enter”<br>- Seleccionar la opción de adicionar métodos<br>- Llenar los campos<br>- Seleccionar “Aceptar” |
| | EC1.2: Crear métodos con campos en blanco | No se crea un método de la entidad | - Llenar los campos<br>- Se muestra un mensaje en el campo donde se cometió el error.<br>- Seleccionar “Aceptar” |
| | EC1.3: Crear métodos incorrectamente | No se crea un método de la entidad | - Llenar los campos con datos incorrectos<br>- Se muestra un mensaje indicando que los campos están incorrectos.<br>- Seleccionar “Aceptar” |
| | EC1.4: Cancelar la operación | No se crea un método de la entidad | - Llenar los campos<br>- Seleccionar “Cancelar” |

**Tabla 4 CP Crear Métodos.**

| Nombre de la sección | Escenarios de la sección | Descripción de la funcionalidad | Flujo central |
|---|---|---|---|
| SC1: Crear Parámetros | EC1.1: Crear parámetros correctamente | Se crea un parámetro del método | - Seleccionar la entidad que se desea editar<br>- Luego hacer doble click en dicha entidad o presionar “Enter”<br>- Seleccionar la opción de adicionar atributos<br>- Llenar los campos<br>- Seleccionar “Aceptar” |
| | EC1.2: Crear | No se crea un | - Llenar los campos |

| | | | |
|---|---|---|---|
| | parámetros con campos en blanco | parámetro del método | - Se muestra un mensaje en el campo donde se cometió el error.<br>- Seleccionar “Aceptar” |
| | EC1.3: Crear parámetros incorrectamente | No se crea un parámetro del método | - Llenar los campos con datos incorrectos<br>- Se muestra un mensaje indicando que los campos están incorrectos.<br>- Seleccionar “Aceptar” |
| | EC1.4: Cancelar la operación | No se crea un parámetro del método | - Llenar los campos<br>- Seleccionar “Cancelar” |

**Tabla 5 CP Crear Parámetros.**

### 3.7. Evaluación de la solución obtenida mediante métricas

Numerosos son los puntos de vista relacionados con la calidad de un software. Desde las disimiles metodologías existentes hasta las normas de calidad vigentes en la actualidad. Inspirados en los estudios que realizara Pressman sobre la calidad del diseño orientado a objeto se crearon un grupo de métricas básicas, referenciadas en (37).

Las mismas son otro aspecto muy importante a la hora de realizar la evaluación de la solución obtenida teniendo en cuenta que estas cubren los principales atributos de calidad que un software debe requerir.

Atributos de calidad que se abarcan:

1. **Responsabilidad:**

Su esencia radica fundamentalmente en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.

2. **Complejidad de implementación:**

Su esencia radica fundamentalmente en el grado de dificultad que tiene implementar un diseño de clases determinado.

3. **Reutilización:**

Su esencia radica fundamentalmente en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

**4. Acoplamiento:**

Su esencia radica fundamentalmente en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.

**5. Complejidad del mantenimiento:**

Su esencia radica fundamentalmente en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.

**6. Cantidad de pruebas:**

Su esencia radica fundamentalmente en el número o el grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (Componente, módulo, clase, conjunto de clases, etc.) implementado.

**7. Nivel de Cohesión:**

Su esencia radica fundamentalmente en el grado de especialización de las clases concebidas para modelar un dominio o concepto específico.

**8. Abstracción del diseño:**

Su esencia radica fundamentalmente en capacidad de modelar lo más cercano posible a la realidad un concepto o dominio determinado. Luego de concebir estas métricas, las cuales se muestran a continuación y que servirán de instrumento para realizar una evaluación crítica de lo implementado se realizó un análisis de las tablas de resultados obtenidos en la evaluación de los instrumentos de medición.

### 3.7.1. Métricas aplicadas

**1- Tamaño operacional de clase (TOC):** Está dado por el número de métodos asignados a una clase. La cual afecta un grupo de atributos como se muestra en la tabla 6.

| Atributo que afecta | Modo en que lo afecta |
|---|---|
| Responsabilidad | Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase. |
| Complejidad de implementación | Un aumento del TOC implica un aumento de la complejidad de implementación de la clase. |
| Reutilización | Un aumento del TOC implica una disminución en el grado de reutilización de la clase. |

**Tabla 6 Tamaño operacional de clase (TOC).**

**2- Relaciones entre clases (RC):** Está dado por el número de relaciones de uso de una clase con otras. La cual afecta un grupo de atributos de calidad como se muestra en la tabla 7.

| Atributo que afecta | Modo en que lo afecta |
|---|---|
| Acoplamiento | Un aumento del RC implica un aumento del acoplamiento de la clase. |
| Complejidad del Mantenimiento | Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase. |
| Reutilización | Un aumento del RC implica una disminución en el grado de reutilización de la clase. |
| Cantidad de pruebas | Un aumento del RC implica un aumento de la cantidad de pruebas de unidad necesarias para probar una clase. |

**Tabla 7 Relaciones entre clases (RC).**

### 3.7.2. Resultados de los instrumentos de evaluación de las métrica

#### 3.7.2.1. Resultados del instrumento de evaluación de la métrica Tamaño operacional de clase (TOC)

En las figuras 20 y 21 se muestran los resultados obtenidos por el instrumento de medición agrupados en intervalos definidos. También se mostraran en las figuras 22, 23 y 24, todos los resultados obtenidos en el instrumento de medición TOC para los atributos de calidad responsabilidad, complejidad y reutilización. Para más información ver instrumentos y tabla de resultados en (Anexo 4 Instrumento de medición de la métrica Tamaño operacional de clase (TOC)).

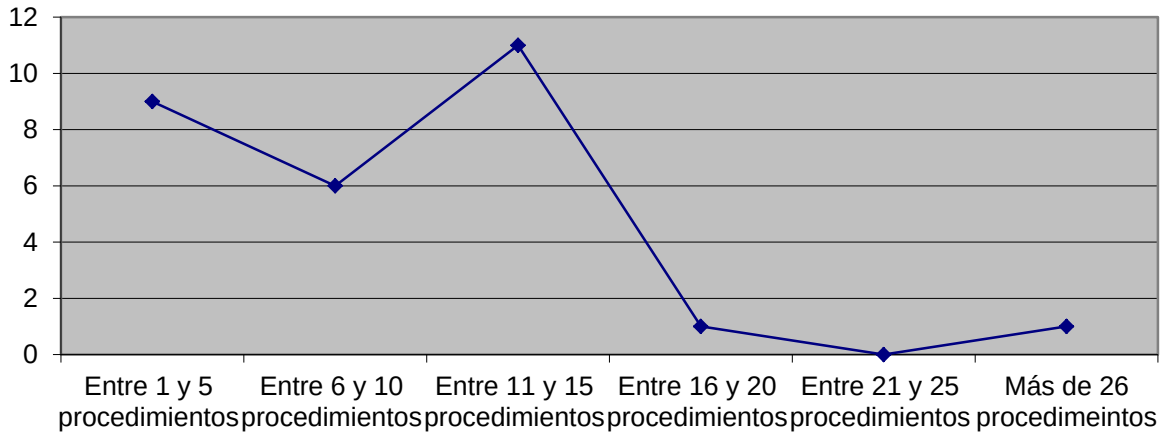



**Figura 14 Representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.**

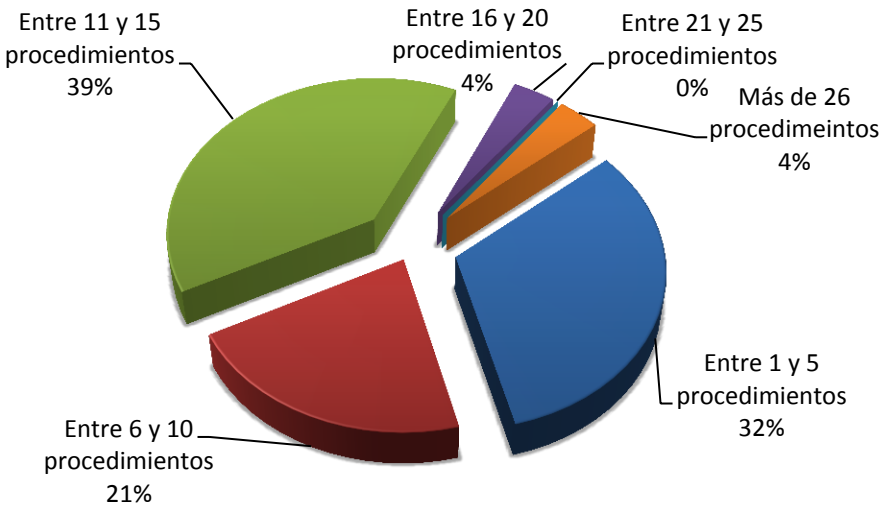



**Figura 15 Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.**

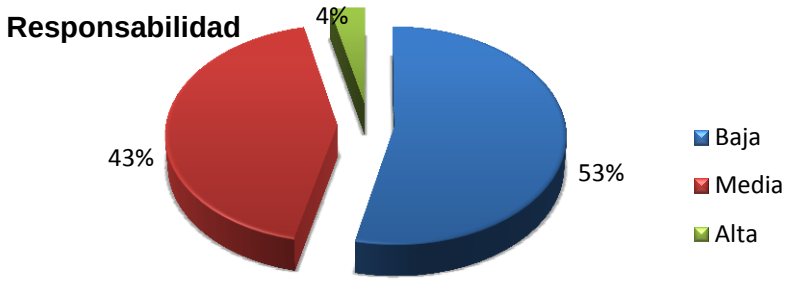



**Figura 16 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad.**

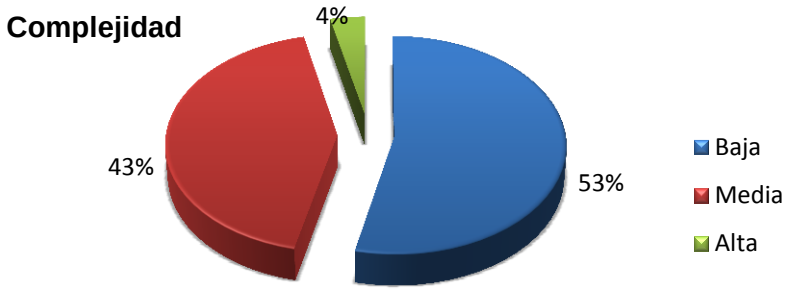



**Figura 17 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación.**

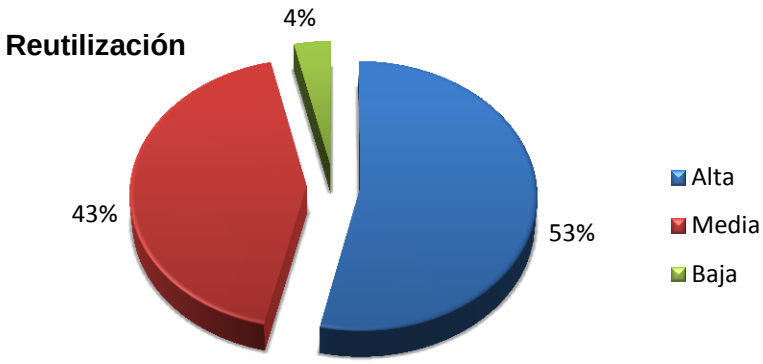



**Figura 18 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.**

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC, se pudo observar que la implementación de las clases necesarias para dar solución a la problemática tiene una calidad aceptable, teniendo en cuenta que el 55 % de las clases utilizadas, incluidas en la solución posee menos cantidad de operaciones que la mitad del valor máximo registrado en las mediciones. Además el 100% de las clases poseen evaluaciones positivas en los atributos de calidad (Responsabilidad, Complejidad de Implementación y Reutilización).

#### 3.7.2.2. Resultados del instrumento de evaluación de la métrica Relaciones entre Clases (RC)

En las figuras 21 y 22 se muestran los resultados obtenidos por el instrumento de medición agrupados por la tendencia de los valores. También se mostraran en las figuras 23, 24, 25 y 26, todos los resultados obtenidos en el instrumento de medición RC para los atributos de calidad acoplamiento, complejidad cantidad de pruebas y reutilización.

Para más información ver instrumentos y tabla de resultados en (Anexo 5 Instrumento de medición de la métrica Relaciones entre clases (RC)).

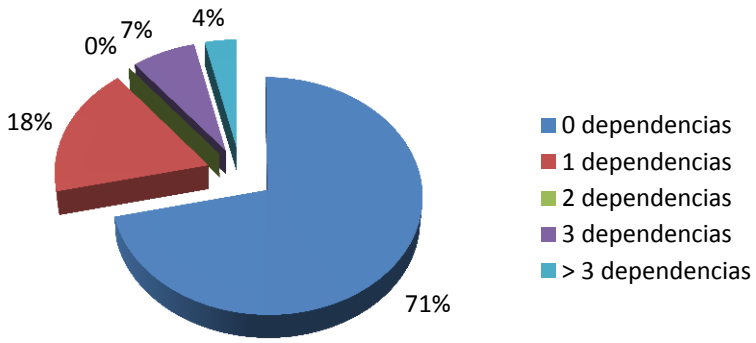



**Figura 19 Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.**

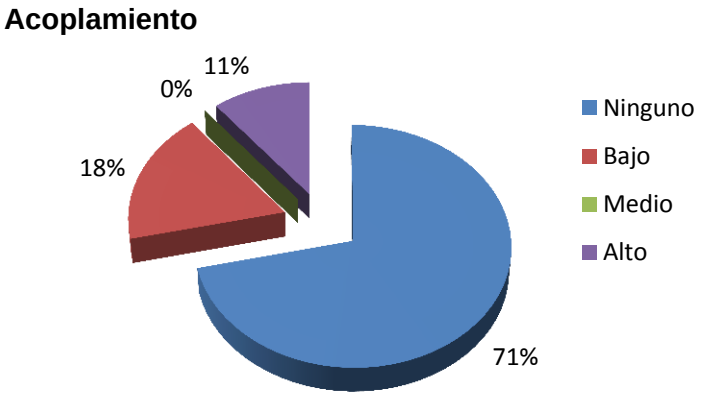



**Figura 20 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.**

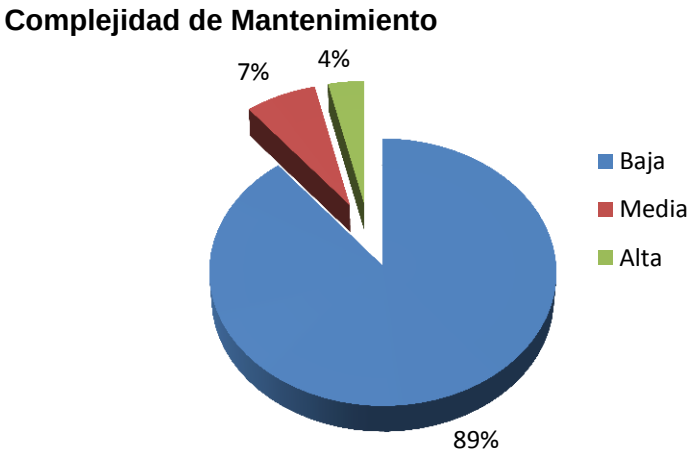



**Figura 21 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento.**

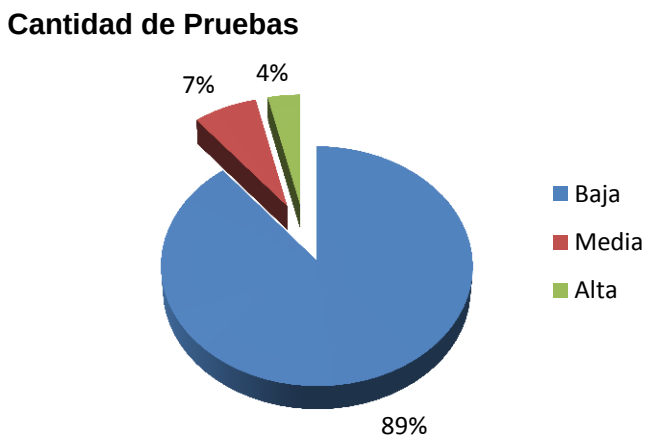



**Figura 22 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.**

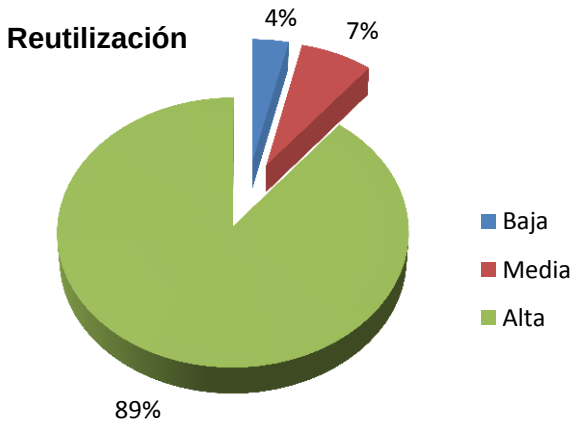



**Figura 23 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización.**

Al hacer un análisis de los resultados obtenidos al evaluar la métrica Relaciones entre clases (RC) con el instrumento de evaluación, se pudo observar que la implementación de las clases necesarias para dar solución a la problemática tiene una calidad aceptable teniendo en cuenta que de las clases implementadas el 70 % de las clases desarrolladas posee 3 o menos dependencias de otras clases. Además de que el 55 % de las clases no posee acoplamiento alguno y el 70% posee índices de calidad

aceptables respecto al mismo atributo de calidad. A la vez que los atributos de calidad Complejidad de Mantenimiento, Cantidad de Pruebas y Reutilización se comportan satisfactoriamente en un 85 % de las clases.

### 3.7.3. Matriz de cubrimiento de los parámetros de calidad evaluados con las métricas propuestas.

La matriz de cubrimiento o matriz de inferencia de los indicadores de calidad es un resumen de los resultados conseguidos al aplicar las métricas referidas en el epígrafe anterior. Esta matriz es una representación estructurada de los atributos de calidad y métricas utilizadas para evaluar la calidad de la solución obtenida como se muestra en la tabla 10. La misma permite conocer si el resultado obtenido de la relación atributo/métricas es positivo o negativo. Al llevar estos resultados a una escala numérica se le asignará el valor 1 si los resultados son positivos, en caso de sean negativos el valor 0 y si no existe relación alguna tomará valor -1, tal y como se muestra en la tabla 8.

| Calificativo | Valor |
|---|---|
| Positivo | 1 |
| Negativo | 0 |
| Nulo | -1 |

**Tabla 8 Clasificación de los parámetros de calidad según impacto en el diseño propuesto**

Una vez completado los datos de dicha relación se determina el promedio de los valores obtenidos de la relación atributo/métrica (solo se toman en consideración las que arrojan un resultado distinto de -1). Este valor representa el impacto que tiene cada atributo en la implementación de la solución determinando si su impacto fue bueno, regular o malo atendiendo al rango en que se encuentre dicho valor, tal y como se muestra en la tabla 9. Al desarrollar este procedimiento con los resultados que se obtuvieron una vez aplicadas las métricas en la implementación realizada se obtuvo la matriz de cubrimiento que se expone a continuación en la tabla 12.

| Calificativo | Rango |
|---|---|
| Malo | <= 0.4 |
| Regular | >0.4 y <=0.7 |
| Bueno | >0.7 |

**Tabla 9 Rangos para evaluar el impacto de los parámetros de calidad en la solución.**

| Atributos de calidad evaluados | Métricas aplicadas al diseño de la solución propuesta | | |
|---|---|---|---|
| | TOC | RC | PROMEDIO |
| Responsabilidad | 1 | -1 | 1 |
| Complejidad Implementación | 1 | -1 | 1 |
| Reutilización | 1 | 1 | 1 |
| Acoplamiento | -1 | 1 | 1 |
| Complejidad Mantenimiento | -1 | 1 | 1 |
| Cantidad Pruebas | -1 | 1 | 1 |

**Tabla 10 Matriz de cubrimiento para los parámetros de calidad evaluados con las métricas aplicadas a la solución.**

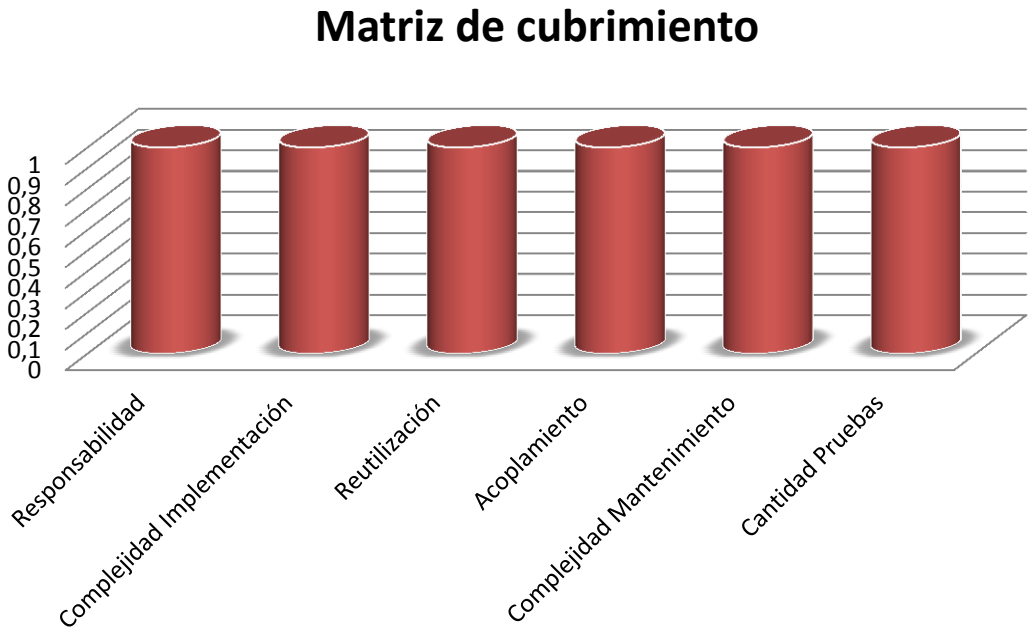



**Figura 24 Resultados obtenidos de la evaluación de los atributos de calidad**

Como se evidencia en los datos visualizados en la matriz se obtuvieron resultados positivos en todos los atributos de calidad evaluados, exceptuando el atributo de la Reutilización, el cual al aplicar la métrica de PH se ve afectado por las características propias del sistema, donde las clases hijas desarrolladas poseen un nivel muy bajo de herencia, provocando a una pobre reutilización. Todo lo anteriormente expuesto

permitió demostrar que las características funcionales más importantes definidas fueron cubiertas en la solución obtenida

### 3.8. Conclusiones parciales

En el desarrollo de este capítulo se analizaron diferentes aspectos como el significado de las pruebas de software y sus objetivos. Se realizó una descripción de las pruebas de unidad, dentro de los cuales se analizó y aplicó el tipo de prueba Caja Negra, verificando la funcionalidad del sistema, y todas las condiciones tanto en su vertiente verdadera como falsa, obteniéndose resultados satisfactorios, que permitieron validar el correcto funcionamiento de las funciones implementadas. Además se elaboraron instrumentos inspirados en métricas para calidad del diseño que permitieron evaluar distintos criterios de calidad arrojando como resultado, valores positivos de su comportamiento, ratificando que la implementación realizada puede catalogarse de aceptable.

# Conclusiones generales

El presente trabajo recoge todo el proceso de implementación del Módulo de Entidades del plugin Génesis quedando como colofón del mismo:

El análisis realizado sobre aplicaciones existentes a nivel internacional relacionadas con desarrollo de sistemas basados en modelos, se obtuvo una visión de cómo se desarrolla el tema en la actualidad.

Se estudiaron las tecnologías, metodologías, lenguajes y herramientas necesarias para realizar el análisis, diseño y posterior desarrollo del sistema.

Se realizó un estudio de la arquitectura propuesta.

La descripción de los aspectos más significativos de la solución como son, la reutilización de componentes, el cálculo de la complejidad ciclomática de algoritmos no triviales así como las principales clases y estándares de codificación utilizados en la implementación de los requerimientos específicos.

La evaluación de la implementación a partir de la aplicación de métricas que permitieron analizar el comportamiento de los atributos de calidad de reutilización, facilidad de mantenimiento, complejidad del diseño, complejidad de implementación, cohesión, acoplamiento y cantidad de pruebas; lo cual arrojó resultados positivos que permiten catalogar de aceptable la implementación realizada.

Por todo lo anteriormente expresado se puede concluir que se cumplió con el objetivo general propuesto: Diseño e Implementación del módulo de entidades del plugin Génesis.

# Recomendaciones

Existen una serie de factores que permitirán la mejora de los procesos de Gestión de Entidades y que deben tenerse en consideración para versiones posteriores de este producto, entre ellos están:

- ✓ Corregir las no conformidades detectadas en las pruebas pilotos con el objetivo de refinar la solución.
- ✓ Ampliar las funcionalidades del módulo con los nuevos requerimientos que surjan por necesidades del cliente.
- ✓ Desarrollar otras versiones de este sistema incorporando nuevas funcionalidades que se crea puedan ser útiles para lograr un producto de mayor calidad.
- ✓ Desarrollar el manual de usuario del módulo implementado para capacitar al personal que lo utilizará.
- ✓ Continuar el estudio con el objetivo de añadir nuevas funcionalidades al módulo.

# Referencias bibliográficas


1. **Jacobson, I, Booch, G y Rumbaugh, J.** *El Proceso Unificado de Desarrollo de Software.* Madrid : s.n., 2000.

2. Object Management Group (2006-05-24). *"OMG Trademarks".* [En línea] 26 de 02 de 2008. http://www.omg.org/legal/tm_list.htm.

3. **Schmidt, D C.** *"Model-Driven Engineering". IEEE Computer.* 2006-05-16. http://www.cs.wustl.edu/~schmidt/PDF/GEI.pdf..

4. Object Management Group. [En línea] http://www.omg.org.

5. **Jacobson , I, Booch, G y Rumbaugh, J.** *El lenguaje unificado de modelado. Manual de Referencia.* 1998.

6. **Evan, Erick .** *Domain-Driver Design: Talckling complexity in the heart of software.* 2004.

7. **Olivares Rojas, Mc. Juan Carlos .** *Patrones de Diseño.* s.l. : Dirección General de Educación Superior Tecnológica.

8. **Gamma , E, y otros, y otros.** *Design Patterns: Elements of Reusable Object Oriented Software, Addison Wesley.* 1995.

9. **Welicki, León.** *Patrones y Antipatrones: una introducción, Revista MTJ .Net.* 2005.

10. Qué es XML. [En línea] http://www.desarrolloweb.com/articulos/449.php.

11. **Hernández Orallo., Enrique.** El Lenguaje Unificado de Modelado (UML). [En línea] http://www.acta.es/articulos_mf/26067.pdf..

12. **JOHN ARTHORNE, C.** *Official Eclipse 3.0 Faqs. Addison-Wesley Professional.* 2004. 384 p. Eclipse Series..

13. *Visual Paradigm for UML.* http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_(M%C3%8D)_14720_p/.

14. *Guia Ubuntu. .* 2009. http://www.guia-ubuntu.org/index.php?title=Subversion..

15. **Pardo Brown, Marcelo .** *Metamodelos, Ing. Civil, Master en informática.*

16. **Giandini, Roxana , Pérez, Gabriela y Pons, Claudia .** Composición de Transformaciones de Modelos en MDD. [En línea] http://www.lifia.info.unlp.edu.ar/papers/2007/Roxana2007.pdf.

17. **Aranda, Andres y Gonzalez Rodriguez, Leidy Natalia.** Grupo MDA. Herramientas. . [En línea] http://ingenieriasoftwaredos.wikispaces.com/Grupo+MDA..

18. **Molina, Pedro J.** GeneXus community. Reporte del Encuentro GeneXus 2009. [En línea] 2009. http://www.genexus.com/..

19. Facebook. [En línea] http://www.facebook.com/note.php?note_id=339752216495.

20. Manuales de OptimalJ 3.0. Compuware. [En línea] 2003. http://www.compuware.com/products/optimalj/.

21. Wikipedia. [En línea] http://es.wikipedia.org.

22. Codagen Architecturect. [En línea] http://www.omg.org/mda/mda_files/Codagen2004.pdf.

23. The Model Driven Software Generator. [En línea] http://www.innoq.com/iqgen/.

24. ESTÁNDAR PARA CODIFICACIÓN EN LENGUAJE C ++. [En línea] http://progra.iteso.mxestandares/estandar%20codificacion%20/c++/estandarcodificacion.pdf..

25. Eclipse Modeling Framework Technology (EMFT). Ecore Tools. [En línea] [Citado el: 30 de 3 de 2011.] http://www.eclipse.org/modeling/emft/?project=ecoretools..

26. **Budinsky, Frank.** *Addison Wesley Profesional. Eclipse Modeling Framework.* 2003.

# Glosario de términos

**Aplicación web**

En la ingeniería software se denomina aplicación web a aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores web (HTML, JavaScript, Java, etc.) en la que se confía la ejecución al navegador.

**Código abierto** (*en inglés Open Source*)

Término con el que se conoce al software distribuido y desarrollado libremente.

**Componente**

Un componente es una clase de uso específico, que puede ser configurada o utilizada de forma visual desde el entorno de desarrollo.

**Framework**

Es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

**Herramientas**

Es un objeto elaborado a fin de facilitar la realización de una tarea mecánica que requiere de una aplicación correcta de energía. 8. Lenguaje de marcado: Es una forma de codificar un documento que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto o su presentación.

**Metodología**

Se refiere a los métodos de investigación que se siguen para alcanzar una gama de objetivos.

**Arquitectura**

La arquitectura de un sistema es la especificación de las partes del mismo, las conexiones entre ellos, y las normas de interacción entre las partes del sistema haciendo uso de las conexiones especificadas.

**Módulo**

Es un software que agrupa un conjunto de subprogramas y estructuras de datos.

**Plataforma**

En informática, una plataforma es precisamente el principio en el cual se constituye un hardware, sobre el cual un software puede ejecutarse/desarrollarse. Una plataforma es un conjunto de subsistemas y tecnologías que aportan un conjunto coherente de funcionalidades a través de interfaces y determinados patrones de uso, que cualquier aplicación que se construya para esa plataforma puede usar sin preocuparse por los detalles de la implementación o cómo se lleva a cabo la misma dentro de la plataforma.

**Independencia de la plataforma**

La independencia de la plataforma es una cualidad que tienen que presentar los modelos. Lo que significa que un modelo es independiente de las facilidades o características que implementan las plataformas, de cualquier tipo (Kleppe Anneke et al., 2003; Arlow Jim, Neustadt Ila, 2004).

**CIM** (*Modelo Independiente de la computación*)

Este modelo es independiente de la manera en la que el sistema está implementado. Es un modelo que muestra el sistema en el entorno en el cual operará y así ayuda a representar exactamente que se espera que el sistema haga.

**PIM** (*Modelo Independiente de Plataforma*)

Describe el sistema, pero no muestra los detalles de plataforma. En este sentido independiente de plataforma es un término relativo. Cuando se indica que un lenguaje o modelo es independiente de plataforma, se debe especificar de qué plataforma tecnológica es independiente.

**PSM** (*Modelo específico de Plataforma*)

Es un modelo computacional que especifica información o formato de tecnología, ya sea un lenguaje de programación, un componente middleware distribuido, etc. Igualmente al caso anterior, un modelo específico se trata de un término relativo. Se dice que un modelo es más específico de plataforma que otro cuando representa conceptos más concretos de la plataforma destino.

**IQGen**

El generador de modelos y código IQGen, de la empresa InnoQ está desarrollado en Java. Incorpora un entorno de modelado muy pobre, que se suple con la admisión de entrada de modelos en formato XMI. Puede generar salida en cualquier lenguaje (J2EE/EJB, C#, Cobol, entre otros).

**EMF - Eclipse Modeling Framework Project (EMF)**

El proyecto EMF es un framework de modelado y facilidades de generación de código para la creación de herramientas de instalación y otras aplicaciones basadas en un modelo de datos estructurados. Desde una especificación de modelo descrito en XMI, EMF ofrece herramientas y soporte en tiempo de ejecución para producir un conjunto de clases Java para el modelo, junto con un conjunto de clases adaptadoras que permiten la visualización y la edición basada en comandos, del modelo, y un editor básico.

**MDE**

Acrónimo inglés de **Model Driven Engeenering,** en español se traduce como Ingeniería de Software Conducida por Modelos. El paradigma MDE tiene dos ejes principales: - por un lado hace énfasis en la separación entre la especificación de la funcionalidad esencial del sistema y la implementación de dicha funcionalidad usando plataformas tecnológicas específicas. Por otro lado, en MDE los modelos son considerados los conductores primarios en todos los aspectos del desarrollo de software. MDE identifica dos tipos principales de modelos: modelos con alto nivel de abstracción e independientes de cualquier tecnología de implementación, llamados PIM y modelos que especifican el sistema en términos de construcciones de implementación disponibles en alguna tecnología específica, conocidos como PSM. Un PIM es transformado en uno o más PSMs, es decir que para cada plataforma tecnológica específica se genera un PSM específico.

**MDD**

Otro acrónimo relacionado a MDE es **Model-Driven Development** *(MDD),* que en español se traduce como Desarrollo de Software Conducida por Modelos. Es visto como un sinónimo de MDE, ambos describen la misma metodología de desarrollo de Software.

**MDA**

En español, **Arquitectura conducida por modelos**. Han surgido varios enfoques dentro del ámbito de MDE, pero sin duda la iniciativa más conocida y extendida es la MDA, acrónimo de Model Driven Architecture, presentada por el consorcio OMG (Object Management Group) en noviembre de 2000 con el objetivo de abordar los desafíos de integración de aplicaciones y los continuos cambios tecnológicos. MDA propone el uso de un conjunto de estándares (descriptos en este Glosario) como MOF, UML, JMI o XMI. Su objetivo es separar la especificación de la funcionalidad del sistema de su implementación sobre una plataforma concreta, por lo que se hace una distinción entre modelos PIM y modelos PSM.

**EMF**

**Eclipse Modeling Framework Project** (EMF).El proyecto EMF es un framework de modelado y facilidades de generación de código para la creación de herramientas de instalación y otras aplicaciones basadas en un modelo de datos estructurados. Desde una especificación de modelo descrito en XMI, EMF ofrece herramientas y soporte en tiempo de ejecución para producir un conjunto de clases Java para el modelo, junto con un conjunto de clases adaptadoras que permiten la visualización y la edición basada en comandos, del modelo, y un editor básico.

**OMG**

El **Object Management Group** u OMG *(de su sigla en inglés Grupo de Gestión de Objetos)* es un consorcio dedicado a la gestión y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA. Es una organización no lucrativa que promueve el uso de tecnología orientada a objetos mediante guías y documentos de especificación de estándares. El grupo está formado por compañías y organizaciones de software como lo son: Hewlett-Packard (HP), IBM, Sun Microsystems, Apple Computer.

**UML 2.0, Infrastructure**

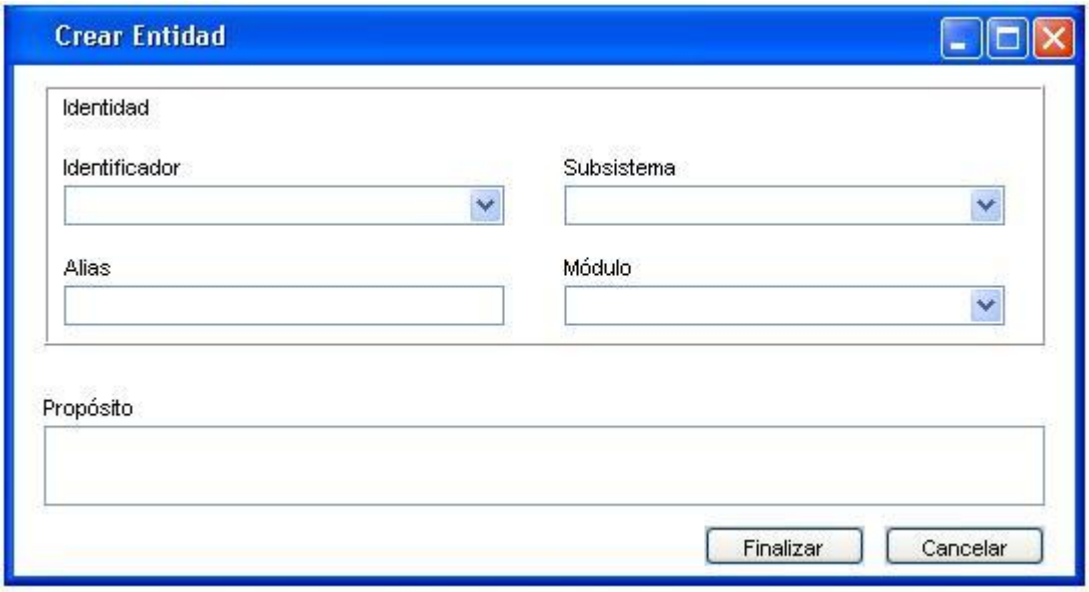
La **Infraestructura UML 2.0** es la primera de dos especificaciones complementarias que representan la principal revisión para el UML de OMG. La segunda especificación es la Superestructura, la cual usa la base arquitectural provista por la primera. La Infraestructura UML 2.0 provee los constructores básicos elementales requeridos para definir lenguajes de modelado, en particular para definir UML 2.0, pero también es útil como base para la definición de otros lenguajes. En el caso de UML, se complementa con la Superstructure.

**UML 2.0, Superstructure**

La **Superestructura UML 2.0** es la especificación que complementa a la Infraestructura definiendo los constructores a nivel usuario requeridos por UML 2.0. Las dos especificaciones complementarias constituyen una especificación completa del lenguaje de modelado UML 2.0.

# Anexos

**Anexo1. Descripción extendida del Caso de Uso Crear Entidad.**

| **Caso de Uso:** | **Crear Entidad** |
|---|---|
| **Actores:** | **Usuario** |
| **Flujo Normal de Eventos** | |
| **Sección "Crear Entidad"** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| 1. El usuario selecciona la opción "Crear Entidad". | 2. El sistema muestra un wizard con los datos que identifican la entidad. |
| ***Prototipo de interfaz***   | |
| *3.* El usuario introduce los datos y selecciona la opción "Finalizar". | *4.* El sistema valida los datos de entrada.<br>*a) En caso de datos incorrectos. Ver sección "Datos Incorrectos".* |

| | |
|---|---|
| | |
| | 5. El sistema almacena la entidad y finaliza el caso de uso. |
| **Flujos Alternos** | |
| **Sección "Datos Incorrectos"** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 1. El sistema señala los datos incorrectos y muestra el mensaje de error correspondiente. |
| 2. El usuario corrige los datos de entrada. | 3. El sistema pasa a la acción 4 del flujo normal de eventos. |

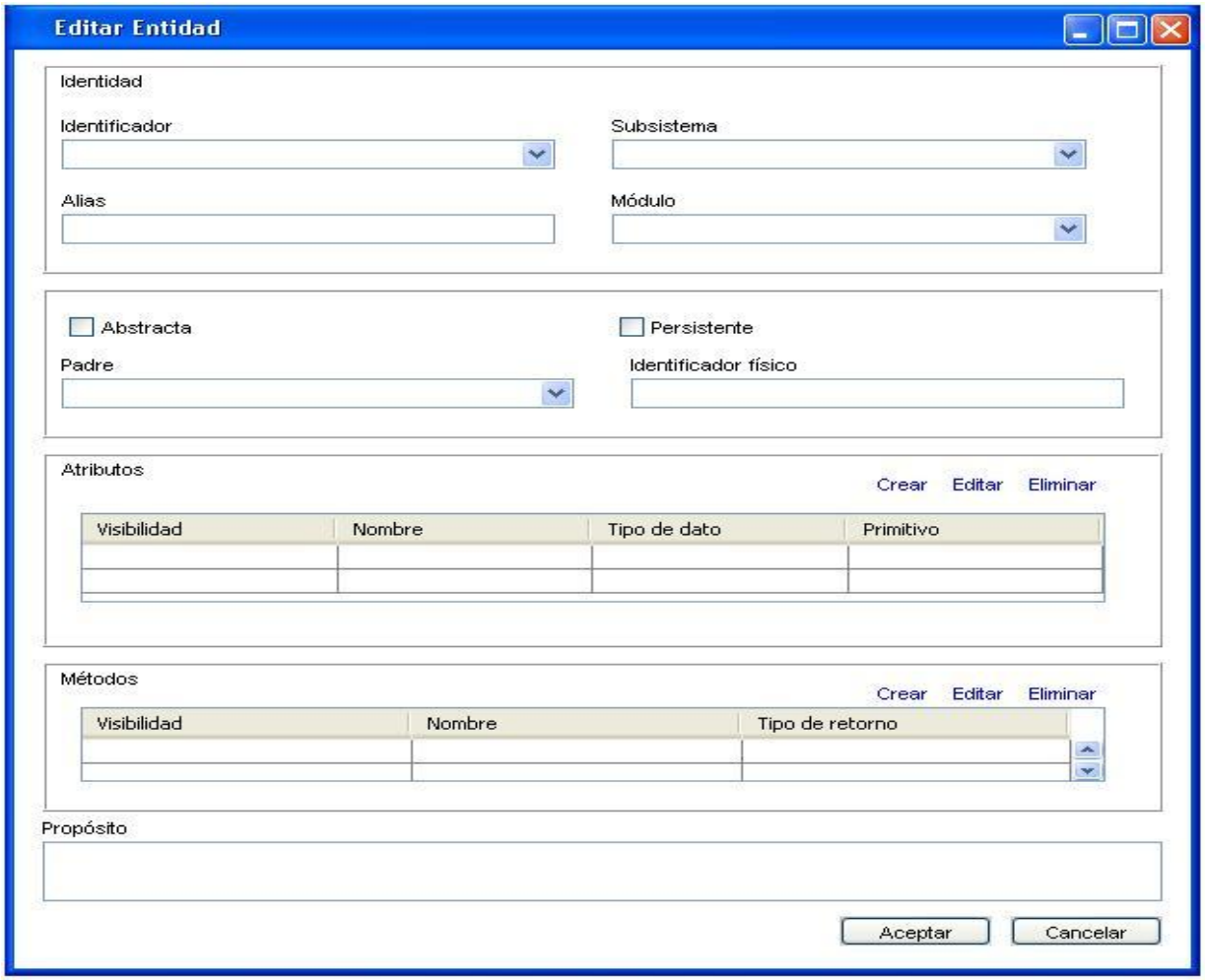
*Validación de campos*

| # | Nombre | Ob. | Validación |
|---|---|---|---|
| 1 | Identificador | X | Debe cumplir la expresión regular siguiente [A-Z][0-9a-zA-Z_]{1,18}. Además no puede existir otro igual en el módulo. |
| 2 | Propósito | X | Puede almacenar hasta 300 caracteres. |
| 3 | Alias | X | Debe cumplir el siguiente formato [0-9a-zA-Z ]{100} |

**Anexo2. Descripción extendida del Caso de Uso Editar Entidad.**

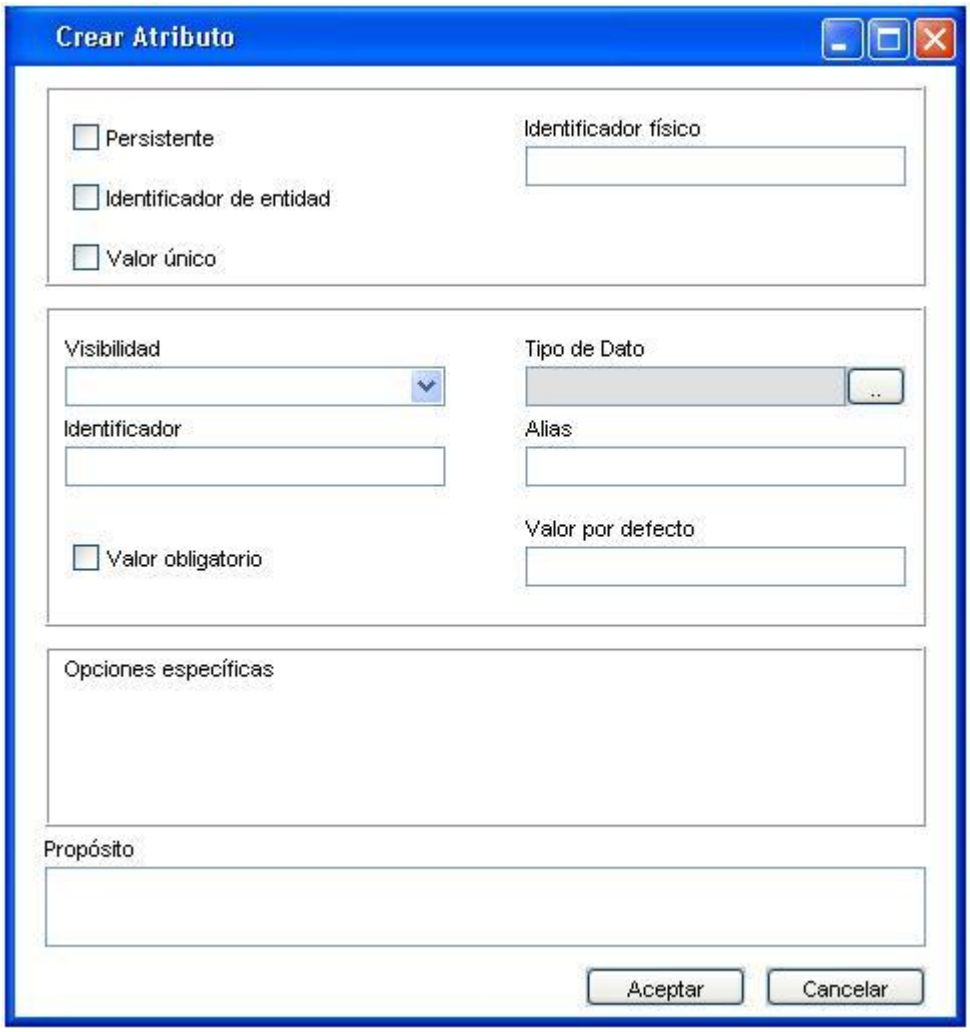
| **Caso de Uso:** | **Editar Entidad** |
|---|---|
| **Actores:** | **Usuario** |
| **Flujo Normal de Eventos** | |
| **Sección "Editar Entidad"** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| 1. El usuario selecciona la opción "Editar Entidad". | 2. El sistema muestra un editor con todos los datos editables según como se muestra en la Figura 1. |

***Figura 1***



| | |
|---|---|
| 3. El usuario introduce los datos y selecciona la opción “Aceptar”.<br>*a) En el caso de seleccionar en la sección Atributos, la opción “Crear”. Ver sección “Crear Atributo”.*<br>*b) En el caso de seleccionar en la sección Atributos, la opción “Editar”. Ver sección “Editar Atributo”.*<br>*c) En el caso de seleccionar en la tabla Atributos, la opción “Eliminar”. Ver sección “Eliminar Atributo”.*<br>*d) En el caso de seleccionar en la sección Métodos, la opción “Crear”. Ver descripción de CU “Comunes/DCU Crear Método”.*<br>*e) En el caso de seleccionar en la sección Métodos, la opción “Editar”. Ver descripción CU “Comunes/DCU Editar Método”.*<br>*f) En el caso de seleccionar en la sección Métodos, la opción “Eliminar”. Ver sección “Eliminar Método”.* | *4.* El sistema valida los datos de entrada.<br>*b) En caso de datos incorrectos. Ver sección “Datos Incorrectos”.* |
| | 5. El sistema almacena los datos y finaliza el caso de uso. |
| **Sección “Crear Atributo”** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 6. El sistema muestra una interfaz con los campos necesarios para crear el atributo como se muestra en la Figura 2. |

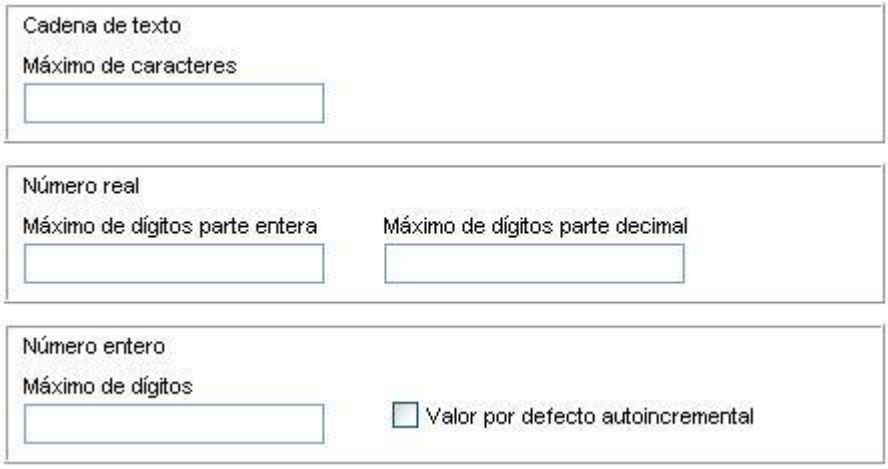
***Figura 2***




7. El usuario introduce los datos.
    a. En caso de seleccionar un tipo de dato primitivo ver sección “Tipo de dato primitivo”.
    b. En caso de seleccionar un tipo de dato complejo ver sección “Tipo de dato complejo”.

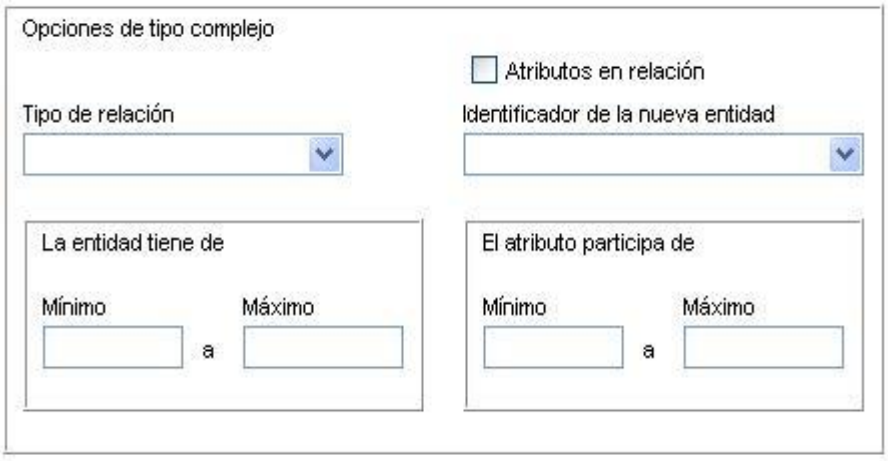
| | |
|---|---|
| 8. El usuario selecciona la opción “Aceptar”. | 9. El sistema valida los datos de entrada y pasa a la acción 4 del flujo normal de eventos.<br>*a) En caso de datos incorrectos. Ver sección “Datos Incorrectos”.* |
| **Sección “Editar Atributo”** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 10. El sistema muestra una interfaz con los campos necesarios para editar el atributo similar a la Figura 2. |
| 11. El usuario introduce los datos y selecciona la opción “Aceptar”. | 12. El sistema valida los datos de entrada y pasa a la acción 4 del flujo normal de eventos.<br>*b) En caso de datos incorrectos. Ver sección “Datos Incorrectos”.* |
| **Sección “Eliminar Atributo”** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 13. El sistema elimina el atributo y pasa a la acción 4 del flujo normal de eventos. |
| **Sección “Eliminar Método”** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 14. El sistema elimina el método y pasa a la acción 4 del flujo normal de eventos. |
| **Flujos Alternos** | |
| **Sección “Tipo de dato primitivo”** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 15. El sistema añade al panel “Opciones específicas” las opciones correspondientes al tipo primitivo seleccionado como se muestra en la Figura 3. |

***Figura 3***




| | |
|---|---|
| 16. El usuario introduce los datos en uno de los paneles mostrados en la Figura 3. | 17. El sistema pasa a la acción 4 del flujo normal de eventos. |
| **Sección "Tipo de dato complejo"** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 18. El sistema añade al panel "Opciones específicas" las opciones correspondientes al tipo complejo seleccionado como se muestra en la Figura 4. |

***Figura 4***



| | |
|---|---|
| 19. El usuario introduce los datos en el panel mostrado. | 20. El sistema pasa a la acción 4 del flujo normal de eventos. |
| **Sección "Datos Incorrectos"** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 21. El sistema muestra el mensaje de error. |
| 22. El usuario corrige los datos de entrada. | 23. El sistema pasa a la acción 4 del flujo normal de eventos. |

**Validación de campos**

*Validación de la entidad*

| # | Nombre | Ob. | Validación |
|---|---|---|---|
| 1 | Alias | X | Debe cumplir el siguiente formato [0-9a-zA-Z ]{100} |
| 2 | Propósito | X | Puede almacenar hasta 300 caracteres. |
| 3 | Padre | | No puede haber en la lista a escoger una entidad hija de la entidad editada. |
| 4 | Identificador físico | ¿? | Si el campo persistente esta seleccionado, el valor de este campo es obligatorio y se debe validar igual que 1. De no estar seleccionado Persistente, se debe deshabilitar este campo. |

*Validación del Atributo*.

| # | Nombre | Ob. | Validación |
|---|---|---|---|
| 5 | Identificador | X | Debe cumplir la expresión regular siguiente [a-z][0-9a-zA-Z_]{49}.<br>Además no puede existir otro igual en la entidad. |
| 6 | Alias | X | Debe cumplir el siguiente formato [0-9a-zA-Z ]{1,100} |
| 7 | Propósito | X | Puede almacenar hasta 300 caracteres. |
| 8 | Persistente | ¿? | Si la entidad no es persistente se debe inhabilitar este atributo. |
| 9 | Identificador físico | ¿? | Si el campo 4 esta seleccionado, el valor de este campo es obligatorio y se debe validar igual que 1. De no estar seleccionado Persistente, se debe deshabilitar este campo. |

| | | | |
|---|---|---|---|
| 10 | Valor por defecto | | Su validación depende del tipo primitivo seleccionado. |
| 11 | Valor obligatorio | ¿? | Si el campo identificador esta seleccionado se debe deshabilitar este campo marcándolo antes. |

*Validación del atributo tipo primitivo cadena de texto*

| # | Nombre | Ob. | Validación |
|---|---|---|---|
| 12 | Máximo de caracteres | | Número entero. |

*Validación del atributo tipo primitivo número real*

| # | Nombre | Ob. | Validación |
|---|---|---|---|
| 13 | Máximo de dígitos parte entera. | X | Número entero. |
| 14 | Máximo de dígitos parte decimal. | X | Número entero. |

*Validación del atributo tipo primitivo número entero*

| # | Nombre | Ob. | Validación |
|---|---|---|---|
| 15 | Máximo de dígitos. | X | Número entero. |

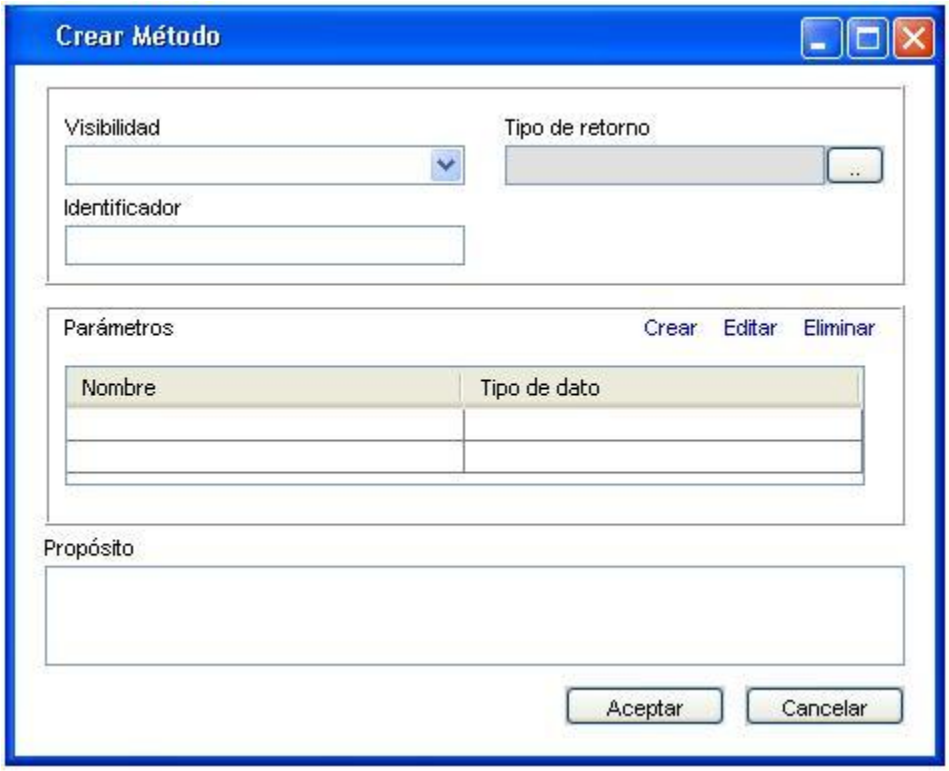
*Validación del atributo tipo complejo*

| # | Nombre | Ob. | Validación |
|---|---|---|---|
| 16 | Identificador de la nueva entidad. | ¿? | Si el campo “Atributos en la relación” esta seleccionado, el valor de este campo es obligatorio y se debe validar igual que 1. De no estar seleccionado “Atributos en la relación”, se debe deshabilitar este campo. |
| 17 | Mínimo | ¿? | Número entero. Según se haya seleccionado “Tipo de relación” estará o no habilitado. Se explica en reglas de negocio. |
| 18 | Máximo | ¿? | Número entero. Según se haya seleccionado “Tipo de relación” estará o no habilitado. Se explica en reglas de negocio. |

**Anexo 3 Descripción extendida del Caso de Uso Crear Método.**

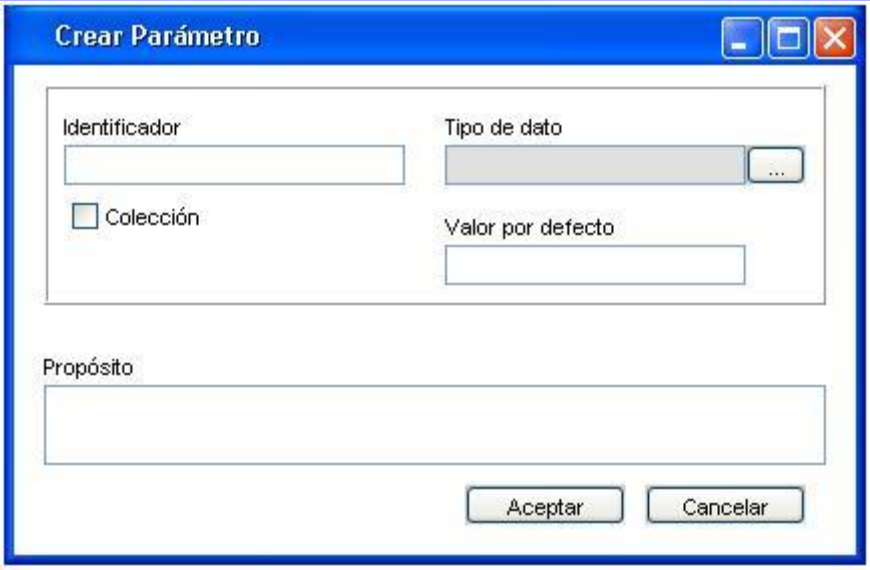
| **Caso de Uso:** | **Crear Método** |
|---|---|
| **Actores:** | **Usuario** |
| **Flujo Normal de Eventos** | |
| **Sección "Crear Método"** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| 6. El usuario selecciona la opción "Crear Método". | 7. El sistema muestra una interfaz con los datos del método. |

***Figura 1***




| | |
|---|---|
| *8.* El usuario introduce los datos y selecciona la opción "Aceptar".<br>*a. Si selecciona la opción Crear parámetro, ver sección "Crear* | *9.* El sistema valida los datos de entrada.<br>*c) En caso de datos incorrectos. Ver sección "Datos Incorrectos".* |

| | |
|---|---|
| *parámetro".*<br>b. *Si selecciona la opción editar parámetro, ver sección "Editar parámetro".*<br>c. *Si selecciona la opción eliminar parámetro, ver sección "Eliminar parámetro".* | |
| | 10. El sistema almacena la entidad y finaliza el caso de uso. |
| **Sección "Crear Parámetro"** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 11. El sistema muestra una interfaz con los campos necesarios para crear el parámetro como se muestra en la Figura 2. |
| ***Figura 2*** | |




| | |
|---|---|
| 12. El usuario introduce los datos y selecciona la opción “Aceptar”.<br>a. *Si selecciona la opción Tipo de dato, ver “Comunes/CU Seleccionar Tipo de Dato”.* | 13. El sistema valida los datos de entrada y pasa a la acción 4 del flujo normal de eventos.<br>c) *En caso de datos incorrectos. Ver sección “Datos Incorrectos”.* |
| **Sección “Editar Parámetro”** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 14. El sistema muestra una interfaz con los campos necesarios para editar el parámetro similar a la Figura 2. |
| 15. El usuario introduce los datos y selecciona la opción “Aceptar”. | 16. El sistema valida los datos de entrada y pasa a la acción 4 del flujo normal de eventos.<br>d) *En caso de datos incorrectos. Ver sección “Datos Incorrectos”.* |

| | |
|---|---|
| **Sección "Eliminar Parámetro"** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 17. El sistema elimina el parámetro y pasa a la acción 4 del flujo normal de eventos. |
| **Flujos Alternos** | |
| **Sección "Datos Incorrectos"** | |
| **Acción del Actor** | **Respuesta del Sistema** |
| | 18. El sistema señala los datos incorrectos y muestra el mensaje de error correspondiente. |
| 19. El usuario corrige los datos de entrada. | 20. El sistema pasa a la acción 4 del flujo normal de eventos. |

*Validación del Método*.

| # | Nombre | Ob. | Validación |
|---|---|---|---|
| 1 | Identificador | X | Debe cumplir la expresión regular siguiente [A-Z][0-9a-zA-Z_]{1,49}. Además no puede existir otro igual en el módulo. |
| 2 | Propósito | X | Puede almacenar hasta 300 caracteres. |

*Validación del Parámetro*.

| # | Nombre | Ob. | Validación |
|---|---|---|---|
| 1 | Identificador | X | Debe cumplir la expresión regular siguiente [A-Z][0-9a-zA-Z_]{1,49}. Además no puede existir otro igual en el módulo. |
| 2 | Propósito | X | Puede almacenar hasta 300 caracteres. |
| 2 | Valor por defecto | | Depende del tipo primitivo seleccionado. |

**Anexo 4 Instrumento de medición de la métrica Tamaño Operacional de Clase (TOC).**

| | Categoría | Criterio |
|---|---|---|
| Responsabilidad | Baja | < = Prom. |
| | Media | Entre Prom y 2* Prom. |
| | Alta | > 2* Prom. |
| | **Categoría** | **Criterio** |
| Complejidad | Baja | < = Prom. |
| | Media | Entre Prom y 2* Prom. |
| | Alta | > 2* Prom. |
| | **Categoría** | **Criterio** |
| Reutilización | Baja | > 2* Prom. |
| | Media | Entre Prom y 2* Prom. |
| | Alta | < = Prom. |

**Tabla 11 Rango de valores de para la evaluación técnica de los atributos de calidad (Responsabilidad, Complejidad de Implementación y Reutilización) relacionados con la métrica TOC.**

| Nº | Subsistema | Clase | Cantidad de Procedimientos | Responsabilidad | Complejidad | Reutilización |
|---|---|---|---|---|---|---|
| 1 | Entidades del plugin Génesis | GEntityController | 4 | Baja | Baja | Alta |
| 2 | Entidades del plugin Génesis | GAtributeDialog | 8 | Baja | Baja | Alta |
| 3 | Entidades del plugin Génesis | GOperationDialog | 8 | Baja | Baja | Alta |
| 4 | Entidades del plugin Génesis | GParameterDialog | 8 | Baja | Baja | Alta |
| 5 | Entidades del plugin Génesis | GAtribute | 13 | Media | Media | Media |

| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | Entidades del plugin Génesis | GChildEntity | 5 | Baja | Baja | Alta |
| 7 | Entidades del plugin Génesis | GComplexAtribute | 13 | Media | Media | Media |
| 8 | Entidades del plugin Génesis | GEntity | 34 | Alta | Alta | Baja |
| 9 | Entidades del plugin Génesis | GEntityRelation | 7 | Baja | Baja | Alta |
| 10 | Entidades del plugin Génesis | GIntegerAtribute | 5 | Baja | Baja | Alta |
| 11 | Entidades del plugin Génesis | GNumericAtribute | 5 | Baja | Baja | Alta |
| 12 | Entidades del plugin Génesis | GPrimitiveAtribute | 5 | Baja | Baja | Alta |
| 13 | Entidades del plugin Génesis | GRealAtribute | 5 | Baja | Baja | Alta |
| 14 | Entidades del plugin Génesis | GStringAtribute | 5 | Baja | Baja | Alta |
| 15 | Entidades del plugin Génesis | GEntityEditor | 11 | Media | Media | Media |
| 16 | Entidades del plugin Génesis | GEntityEditorContributor | 6 | Baja | Baja | Alta |
| 17 | Entidades del plugin Génesis | GEntityEditorInput | 11 | Media | Media | Media |
| 18 | Entidades del plugin Génesis | GAtributeForm | 14 | Media | Media | Media |
| 19 | Entidades del plugin Génesis | GComplexAtributeForm | 12 | Media | Media | Media |
| 20 | Entidades del | GEntityEditForm | 20 | Media | Media | Media |

| | | | | | | |
|---|---|---|---|---|---|---|
| | plugin Génesis | | | | | |
| 21 | Entidades del plugin Génesis | GEntityForm | 13 | Media | Media | Media |
| 22 | Entidades del plugin Génesis | GIntegerAtributeForm | 14 | Media | Media | Media |
| 23 | Entidades del plugin Génesis | GOperationForm | 14 | Media | Media | Media |
| 24 | Entidades del plugin Génesis | GParameterForm | 9 | Baja | Baja | Alta |
| 25 | Entidades del plugin Génesis | GRealAtributeForm | 14 | Media | Media | Media |
| 26 | Entidades del plugin Génesis | GStringAtributeForm | 14 | Media | Media | Media |
| 27 | Entidades del plugin Génesis | NewEntityWizard | 5 | Baja | Baja | Alta |
| 28 | Entidades del plugin Génesis | NewEntityWizardPage | 4 | Baja | Baja | Alta |

**Tabla 12 Resultados de la evaluación de la métrica TOC y su influencia en los atributos de calidad (Responsabilidad, Complejidad de Implementación y Reutilización).**

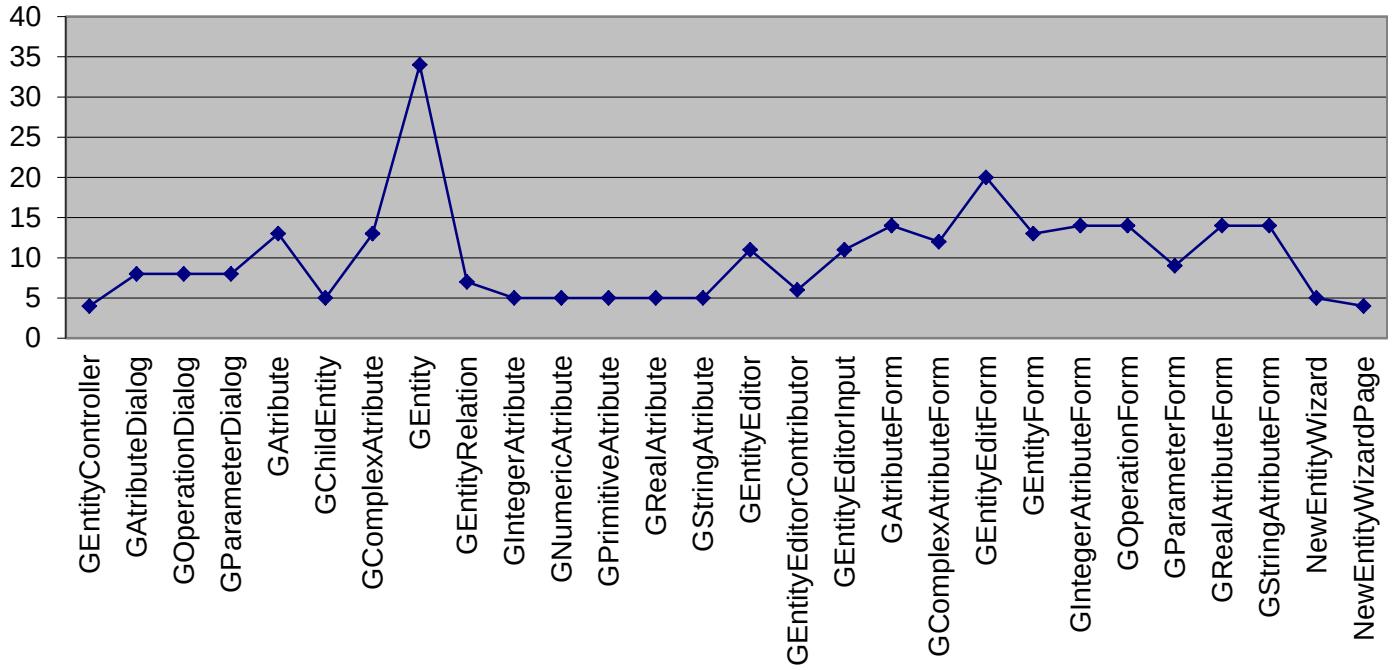



**Figura 25 Gráfica de los resultados de la evaluación de la métrica TOC y su influencia en los atributos de calidad (Responsabilidad, Complejidad de Implementación y Reutilización).**

**Anexo 5 Instrumento de medición de la métrica Relaciones entre Clase (RC).**

| | Categoría | Criterio |
|---|---|---|
| Acoplamiento | Ninguno | 0 |
| | Bajo | 1 |
| | Medio | 2 |
| | Alto | >2 |
| | **Categoría** | **Criterio** |
| Reutilización | Baja | > 2* Prom. |
| | Media | Entre Prom y 2* Prom. |
| | Alta | < = Prom. |
| | **Categoría** | **Criterio** |
| Complejidad | Baja | < = Prom. |
| | Media | Entre Prom y 2* Prom. |
| | Alta | > 2* Prom. |
| | **Categoría** | **Criterio** |
| Cantidad de pruebas | Baja | < = Prom. |
| | Media | Entre Prom y 2* Prom. |
| | Alta | > 2* Prom. |

**Tabla 13 Rango de valores de para la evaluación técnica de los atributos de calidad (Acoplamiento, Reutilización, Complejidad de Implementación y Cantidad de pruebas) relacionados con la métrica RC.**

| Nº | Subsistema | Clase | Cantidad de Relaciones de uso | Acoplamiento | Reutilización | Complejidad | Cantidad de pruebas |
|---|---|---|---|---|---|---|---|
| 1 | Entidades del plugin Génesis | GEntityController | 0 | Ninguno | Baja | Alta | Baja |
| 2 | Entidades del plugin Génesis | GAtributeDialog | 0 | Ninguno | Baja | Alta | Baja |
| 3 | Entidades del plugin Génesis | GOperationDialog | 0 | Ninguno | Baja | Alta | Baja |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | Entidades del plugin Génesis | GParameterDialog | 0 | Ninguno | Baja | Alta | Baja |
| 5 | Entidades del plugin Génesis | GRuleDialog | 3 | Alto | Media | Media | Media |
| 6 | Entidades del plugin Génesis | GAtribute | 1 | Bajo | Baja | Alta | Baja |
| 7 | Entidades del plugin Génesis | GChildEntity | 1 | Bajo | Baja | Alta | Baja |
| 8 | Entidades del plugin Génesis | GComplexAtribute | 7 | Alto | Alta | Baja | Alta |
| 9 | Entidades del plugin Génesis | GEntity | 1 | Bajo | Baja | Alta | Baja |
| 10 | Entidades del plugin Génesis | GEntityRelation | 0 | Ninguno | Baja | Alta | Baja |
| 11 | Entidades del plugin Génesis | GIntegerAtribute | 0 | Ninguno | Baja | Alta | Baja |
| 12 | Entidades del plugin Génesis | GNumericAtribute | 3 | Alto | Media | Media | Media |
| 13 | Entidades del plugin Génesis | GPrimitiveAtribute | 0 | Ninguno | Baja | Alta | Baja |
| 14 | Entidades del plugin Génesis | GRealAtribute | 0 | Ninguno | Baja | Alta | Baja |
| 15 | Entidades del plugin Génesis | GRelationTypes | 0 | Ninguno | Baja | Alta | Baja |
| 16 | Entidades del plugin Génesis | GStringAtribute | 0 | Ninguno | Baja | Alta | Baja |
| 17 | Entidades del plugin Génesis | GEntityEditor | 0 | Ninguno | Baja | Alta | Baja |
| 18 | Entidades del plugin Génesis | GEntityEditorContributor | 0 | Ninguno | Baja | Alta | Baja |
| 19 | Entidades del plugin Génesis | GEntityEditorInput | 0 | Ninguno | Baja | Alta | Baja |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 20 | Entidades del plugin Génesis | GAtributeForm | 0 | Ninguno | Baja | Alta | Baja |
| 21 | Entidades del plugin Génesis | GComplexAtributeForm | 1 | Bajo | Baja | Alta | Baja |
| 22 | Entidades del plugin Génesis | GEntityEditForm | 0 | Ninguno | Baja | Alta | Baja |
| 23 | Entidades del plugin Génesis | GEntityForm | 1 | Bajo | Baja | Alta | Baja |
| 24 | Entidades del plugin Génesis | GIntegerAtributeForm | 0 | Ninguno | Baja | Alta | Baja |
| 25 | Entidades del plugin Génesis | GOperationForm | 0 | Ninguno | Baja | Alta | Baja |
| 26 | Entidades del plugin Génesis | GParameterForm | 0 | Ninguno | Baja | Alta | Baja |
| 27 | Entidades del plugin Génesis | GRealAtributeForm | 0 | Ninguno | Baja | Alta | Baja |
| 28 | Entidades del plugin Génesis | GStringAtributeForm | 0 | Ninguno | Baja | Alta | Baja |
| 29 | Entidades del plugin Génesis | NewEntityWizard | 0 | Ninguno | Baja | Alta | Baja |
| 30 | Entidades del plugin Génesis | NewEntityWizardPage | 0 | Ninguno | Baja | Alta | Baja |

**Tabla 14 Resultados de la evaluación de la métrica RC y su influencia en los atributos de calidad (Acoplamiento, Reutilización, Complejidad de Implementación y Cantidad de pruebas).**

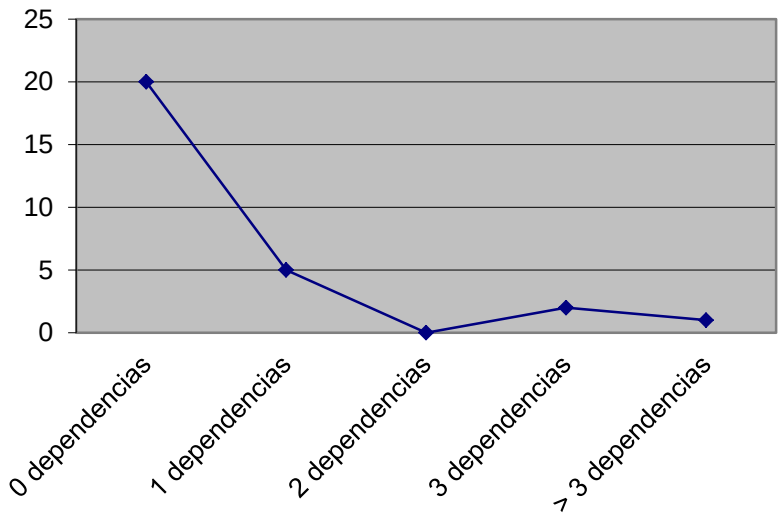



**Figura 26 Gráfica de los resultados de la evaluación de la métrica RC y su influencia en los atributos de calidad (Acoplamiento, Reutilización, Complejidad de Implementación y Cantidad de pruebas).**

**Anexo 6 Herramientas MDA (AndroMDA, ArcStyler, OptimalJ)**

| Aspectos | AndroMDA | ArcStyler | OptimalJ |
|---|---|---|---|
| **Licencia** | Open Source. | Comercial. | Comercial. |
| **Configuración mínima del equipo** | CPU 1,2 GHz.<br>512 MB de Memoria.<br>400 MB de espacio libre en disco. | CPU 1,4 GHz.<br>512 MB de Memoria.<br>400 MB de espacio libre en disco (800 MB durante la instalación), dependiendo del paquete de MDA Cartridges que se instale. | CPU 1 GHz.<br>512 MB de Memoria.<br>450 MB de espacio libre en disco.<br>Software de requisito previo requerido: JDK 1.4.2 o 5.0.<br>Resolución de Pantalla de 1280 x 1024 pixel. |
| **Sistemas Operativos Soportados** | Windows 2000.<br>Windows XP.<br>Red Hat 8.0.<br>Windows 2003 Server. | Windows NT 4.0.<br>Windows 2000.<br>Windows XP.<br>Windows 2003 Server.<br>Linux (Probado en SuSE 9.2 y GenToo), | Windows 2000.<br>Windows XP.<br>Red Hat 8.0. |
| **Ediciones** | AndroMDA. | ArcStyler Enterprise Edition o ArcStyler Architect Edition. | OptimalJ Architecture Edition, OptimalJ Professional Edition y OptimalJ Developer. |
| **Plataforma** | J2EE, Spring, .NET | J2EE y .NET | J2EE |
| **Lenguaje de Modelado** | UML (MagicDraw, Poseidon, ArgoUML entre otras). | UML (MagicDraw). | UML (MagicDraw). |
| **Documentación/Ayuda** | Dispone de numerosos manuales para el manejo del aplicativo | Dispone de numerosos manuales tanto para el manejo del aplicativo como para el uso de MDA | Dispone de numerosos manuales para el manejo del aplicativo (en su gran mayoría en idiomas diferentes al español). |

| | | | |
|---|---|---|---|
| | (en su gran mayoría en inglés). | Cartridge (en su gran mayoría en idiomas diferentes al español). | |
| **Ciclo de vida** | Soporta casi todo el ciclo de vida (excepto Soporte para el manejo de requisitos). | Soporta casi todo el ciclo de vida (excepto Soporte para codificación y despliegue). | Soporta casi todo el ciclo de vida (excepto Soporte para el manejo de requisitos). |
| **Permite varias implementaciones** | Incorpora implementaciones para generar código Java (en particular J2EE), y gracias a su sistema de cartuchos puede ampliarse a cualquier otra plataforma. | SI (Java2, EJB, Servicios Web, Corba, .NET y gracias a CARAT permite definir nuevos MDA-Cartridge). | Genera tres tipos distintos de PSM's a partir del mismo PIM, pero todas van dirigidos a la misma plataforma J2EE. |
| **Interoperabilidad** | SI (Importa y Exporta modelos a través de XMI). | SI (Importa y Exporta modelos a través de XMI). | SI (Importa y Exporta modelos a través de XMI). |
| **Trazabilidad** | La herramienta ofrece poco soporte en este aspecto. | No cuenta con un registro que permita conocer qué elemento del PIM corresponde al del PSM. | Permite conocer qué elemento del PIM le corresponde a un elemento del PSM y el sitio donde está ubicado en el código. |
| **Calidad del código generado** | Bien documentado y legible. | Bien documentado y legible. | Bien documentado pero poco legible ya que utiliza varios patrones en el código. |
| **Facilidad en la creación de aplicaciones.** | Se requiere de un buen conocimiento de la herramienta | Requiere de un mayor esfuerzo de desarrollo, el programador está en la | En poco tiempo permite crear una aplicación a partir de un simple modelo de clases, |

| | | | |
|---|---|---|---|
| | para poder desarrollar. | obligación de escribir el código de integración de modelos. | generando código de calidad y aplicando patrones de diseño. |
| **Qué elementos UML permite.** | Diagramas de clases.<br>Restricciones OCL.<br>Casos de uso.<br>Diagramas de actividad.<br>Diagramas de estado.<br>Diagramas de secuencia.<br>Diagramas de colaboración.<br>Diagramas de componentes.<br>Diagramas de despliegue. | Diagramas de clases.<br>Restricciones.<br>OCL.<br>Casos de uso.<br>Diagramas de actividad.<br>Diagramas de estado.<br>Diagramas de secuencia.<br>Diagramas de colaboración. | Diagramas de clases.<br>Restricciones OCL.<br>Casos de uso.<br>Diagramas de actividad.<br>Diagramas de estado.<br>Diagramas de secuencia.<br>Diagramas de colaboración.<br>Diagramas de componentes.<br>Diagramas de despliegue. |

**Tabla 15 Tabla comparativa de herramientas MDA (AndroMDA, ArcStyler, OptimalJ).**