Universidad de las Ciencias Informáticas

"Facultad 8"

Título: Plug-in para Eclipse para la generación de elementos arquitectónicos personalizados

Trabajo de Diploma para optar por el título de Ingeniero Informático

Autor(es): Héctor Luis López González

Tutor(es): Maykell Frómeta Flores.

Mayo 2009, Año del 50 Aniversario del Triunfo de la Revolución.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Héctor Luis López González Maykell Frómeta Flores

Agradecimientos

Les agradezco a todas aquellas personas que hicieron posible la realización de este trabajo, que me apoyaron, me guiaron y orientaron; a mi tutor Maikell Frómeta, Jorge Amado Soria Ramírez y a Yamek Hernández.

Dedicatoria

A mi mamá, por haberme apoyado en todo momento, por haberse consagrado totalmente conmigo desde el día en que nací. Por haberse sacrificado tanto por mí. Hoy estoy donde estoy y soy como soy gracias a su apoyo, cariño y eterna pasión para conmigo. Nunca ha dudado de mí, siempre me ha respaldado y me ha amado, por eso y por la inmensa admiración que siento por ella, es que le dedico no solamente mi tesis sino mi eterno cariño y amor.

A mi abuela, por estar siempre preocupada por mí, por amarme y quererme tanto, por tenerme siempre presente, por estar ahí en los buenos y los malos momentos, por aconsejarme y guiarme por el camino del hombre.

A mi mamá y a mi abuela por haberse quitado hasta lo que no tenían para yo no tener la más mínima preocupación, por eso y más les dedico todos mis logros y sueños.

A mi papá, a mi hermana Yamilet, a Alejandro, a mi abuelo Roberto por haberme ayudado en todo lo que han podido, por acordarse de mí, quererme, amarme, porque me han demostrado que puedo contar con su apoyo en todo momento y por sobre todas las cosas por ser mi familia.

A Dalinda, Idalmis y Ciro por ser más que mi novia y suegros, por haberse ganado un lugar en mi familia y en mi corazón, por haberme apoyado en todo momento, por haber compartido conmigo su cariño y consejo. Especialmente a Dali por haberse sacrificado tanto estando a mi lado, por amarme tanto y entenderme siempre.

Resumen

En el presente trabajo de diploma se propone un nuevo plug-in para Eclipse denominado GEAP, plug-in que permite agilizar el desarrollo mediante la generación de código a partir de las especificaciones arquitectónicas. El plug-in consiste en varios componentes que permiten la generación de elementos arquitectónicos personalizados, e incluye además una ayuda integrada al Eclipse. El sistema pretende automatizar la creación de clases y de paquetes bien estructurados, a través de la creación de plantillas especificadas por la arquitectura. Se utilizó OpenUP como metodología por sus características ágiles y de orientación a proyectos y equipos pequeños. Como resultado de este trabajo se presenta la implementación del plug-in, así como la documentación necesaria para su uso y una validación de su utilidad en un ambiente real.

PALABRAS CLAVE

Plug-in, GEAP, Generación de código.

ÍNDICE DE CONTENIDOS

ÍNDICE DE IMÁGENES

ÍNDICE DE TABLAS

INTRODUCCIÓN

El comandante en jefe Fidel Castro Ruz en el VIII Congreso de la UJC expresó…"con la aspiración de contar con centros de excelencia en la educación superior dio lugar al surgimiento de la Universidad de las Ciencias Informáticas, primera institución de su tipo surgida en la Batalla de Ideas"[1]. La Universidad de las Ciencias informáticas (UCI) tiene como objetivo la formación de jóvenes informáticos que vinculen el estudio con la producción, reflejo de las ideas en torno a la escuela de José Martí. En la misma se desarrollan software para la industria nacional e internacional, lo cual contribuye a la economía nacional y evidencia las altas posibilidades de convertir a Cuba en una potencia productiva en esta rama. En este proceso, innumerables estudiantes con casi ninguna experiencia en Proyectos Productivos de Software, han tenido y siguen desarrollando software con grandes complejidades.

Uno de los espacios más determinantes y muchas veces no suficientemente cubierto en el desarrollo de software es el que tiene que ver con el de proporcionar los elementos necesarios a los programadores para favorecer la agilidad en el desarrollo. Muchas plataformas de desarrollo como Visual Studio .Net ofrecen facilidades en este sentido. Sin embargo, para proyectos que usen plataformas libres o de código abierto, las que abundan para el entorno Java Micro Edition (JME) y Java Enterprise Edition (JEE) (como el Entorno de desarrollo Integrado (IDE) Eclipse) estas facilidades son limitadas. Lo cual provoca una pérdida de productividad por parte de desarrolladores de poca y mediana experiencia.

En la UCI la mayor fuerza para enfrentar el desarrollo de software se encuentra en los estudiantes de 2do a 4to año, además, el 10% de los proyectos de la universidad están desarrollados en Java. Estas dos premisas ofrecen una idea aproximada de hasta dónde puede afectar el no contar con herramientas que agilicen el trabajo de los programadores en el cumplimiento de las metas de los proyectos productivos donde laboran.

De la situación planteada anteriormente surge el siguiente **problema**:

[1] Fragmentos del discurso pronunciado por Fidel Castro Ruz, Primer Secretario del PCC, en la clausura del VIII Congreso de la Unión de Jóvenes Comunistas, La Habana, 5 de diciembre de 2004.

¿Cómo agilizar el desarrollo de software mediante el uso, por parte de los programadores, de una herramienta de generación de código ajustada a las pautas de arquitectura previamente definidas para un proyecto en cualquiera de las plataformas Java?

El **objeto de estudio** es la Arquitectura de Software y las Técnicas de Programación para el desarrollo de Herramientas de Generación de Código.

El **campo de acción** está enmarcado en el desarrollo de herramientas de generación de código para proyectos Java desarrollados con Eclipse IDE.

Como **Idea a Defender** se plantea que con la creación de un plug-in para Eclipse IDE se agilizará el desarrollo de software por parte de los programadores y se asegurará el cumplimiento de un gran porciento de las pautas de Arquitectura de Software definidas por los proyectos que usen Java.

Para darle solución al problema anteriormente planteado, se ha determinado como **objetivo general de la investigación** desarrollar una herramienta de generación de código de fácil manejo por parte de los desarrolladores.

Los **objetivos específicos** son:

- Estudiar y analizar los elementos teóricos para la realización del Plug-in a desarrollar.
- Realizar e implementar un plug-in para Eclipse que permita agilizar el desarrollo mediante la generación de código a partir de las especificaciones arquitectónicas de una aplicación dada.

La siguiente **pregunta científica** tiene como objetivo ser guía del proceso de investigación:

- ¿Cómo y qué agilizar en el desarrollo de la producción enmarcada a una arquitectura en un proyecto sobre JEE?

Para cumplir con el objetivo propuesto y dar solución a la problemática planteada en esta investigación se proponen las siguientes **tareas investigativas**:

- Conocer y analizar la arquitectura orientada a plug-in para Eclipse.
- Investigar acerca de las mejores prácticas para el desarrollo de plug-in de la plataforma Eclipse.
- Realizar el diseño del plug-in.

- Implementar un plug-in para Eclipse, que agilice el proceso de desarrollo en los proyectos productivos y que permita:
    - Crear asistentes que permitan generar elementos arquitectónicos personalizados y que posibiliten la ágil creación de clases que cumplan con los patrones especificados.

En correspondencia con la lógica de la investigación, la tesis se estructura en tres capítulos. El primero titulado: Fundamentación Teórica, donde se introducen los conceptos y las definiciones necesarias para poder desarrollar un plug-in para Eclipse.

En el segundo capítulo titulado: Características del plug-in, se incursiona en las características del plug-in y en las funcionalidades que brindará, así como en su diseño, estructura y principales clases que lo conforman, además de los artefactos generados por la metodología.

En el tercer capítulo: Prueba y validación de la solución, se exponen las pruebas realizadas al plug-in y la validación en ambiente real de la solución dada.

CAPÍTULO 1

FUNDAMENTACIÓN TEÓRICA

Introducción

Este capítulo tiene como objetivo explicar los principales conceptos involucrados en la arquitectura del IDE Eclipse, así como lo relacionado con su elemento fundamental, el plug-in, también se abordan las tecnologías utilizadas para la elaboración del mismo y se realiza un análisis de las principales soluciones enmarcadas en el ámbito del campo de acción de la problemática propuesta.

1.1 Ambiente de desarrollo Integrado:

Un ambiente de desarrollo integrado o IDE, es un programa compuesto por un conjunto de herramientas para agilizarle el trabajo al programador en el proceso de producción de software.

Puede dedicarse a un solo lenguaje de programación como es el caso de Borland C++ o puede estar orientado a varios lenguajes como es el caso de Eclipse.

Un IDE es un entorno de programación que se ha empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador, y un constructor de interfaz gráfica GUI. (1)

Los IDEs brindan un entorno de trabajo amigable para la mayoría de los lenguajes de programación como C++, C#, Java, entre otros. Además pueden ser aplicaciones por si solas o pueden formar parte de aplicaciones que ya existen.

1.2 Eclipse:

Gran parte de la programación de Eclipse fue realizada por IBM (International Business Machines) antes de que se creara el proyecto Eclipse como tal. Con la aparición de Java en la década de los 90, su rápida expansión y sus ventajas con miras a un internet en plena expansión obligaron a IBM a plantearse la construcción de una plataforma basada en Java. Surgió así el proyecto Eclipse.

Al inicio, Eclipse no era muy conocido en la comunidad internacional, por lo que el consorcio de la IBM se mostró renuente a invertir en la plataforma; por lo que a finales de 2001 IBM adoptó una licencia de código abierto para incrementar el desarrollo del mismo y acelerar la acogida por la comunidad de desarrolladores. Para ese entonces IBM puso el proyecto Eclipse en manos de un consorcio (Eclipse.org) de empresas fabricantes de herramientas de software. Originalmente la junta directiva del consorcio incluía a Borland, MERANT, IBM, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft y Webgain. Con el tiempo el número de miembros fue aumentando y ya en el 2003 poesía 80 integrantes.

El 2 de febrero del 2004, el consorcio anunció la reorganización de Eclipse en una corporación sin ánimo de lucro, independientemente de su fundadora original IBM, denominada Fundación Eclipse. Desde ese momento el código fuente y la tecnología desarrollada por la comunidad de Eclipse estuvo disponible gratuitamente a través de la Eclipse Public License. Eclipse se distribuye actualmente bajo la licencia CPL (Common Public License o Licencia Publica Común) versión 1.0 de IBM, aprobada por la organización Open Source Initiative (OSI). (2)

Hasta ahora hemos visto el origen del Eclipse pero aun queda la interrogante de lo que es en realidad el Eclipse.

La web oficial de Eclipse define al mismo como “una plataforma (IDE), abierta para todo y para nada en particular” (3). Eclipse es una plataforma porque no se encuentra acabada en su totalidad, pero está diseñado para que sea extensible indefinidamente con la adecuada implementación de plug-in. Es un ambiente de desarrollo integrado (IDE) porque provee de herramientas que facilitan el trabajo en el desarrollo de software, porque permite administrar el espacio de trabajo o workspace, porque permite compilar, correr y depurar aplicaciones, porque además posee herramientas que permiten compartir elementos y control de versión sobre el código fuente. La característica clave de Eclipse es la extensibilidad. Eclipse es una gran estructura formada por un núcleo y muchos plug-ins que van conformando la funcionalidad final. La forma en que los plug-ins interactúan es mediante interfaces o puntos de extensión; así, las nuevas aportaciones se integran sin dificultad ni conflictos. El Eclipse es neutral y apropiado para todo, puesto que ha sido utilizado para desarrollar todo tipo de aplicaciones, y ha sido altamente probado en cualquier ambiente, ya sea en la construcción de Servicios Web, aplicaciones

desktop, juegos, etcétera. Como se refleja en lo anterior el Eclipse tiene una gran capacidad y versatilidad, lo cual es permitido por su arquitectura.

1.2.1 Arquitectura del Eclipse

Las características que le dan potencia y versatilidad al Eclipse es su bien definida arquitectura orientada a plug-in.

En adición al pequeño núcleo o kernel denominado platform runtime, la Plataforma Eclipse se encuentra formada por plug-in entre los cuales se encuentra el Workbench, el Workspace, el Help, y el Team como se muestra en la Figura 1.

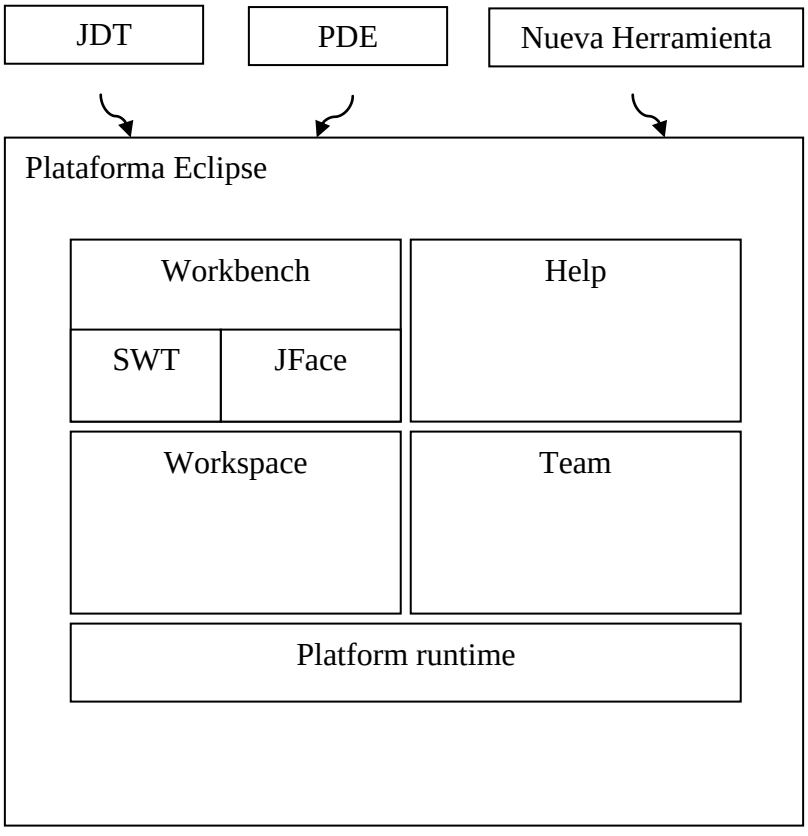


Figura 1: Arquitectura basada en plug-ins

El Workspace es el responsable de manejar los recursos de los usuarios, los cuales están organizados en uno o más proyectos en el nivel superior. Cada proyecto se corresponde a un subdirectorio del directorio

de trabajo de Eclipse. Cada proyecto puede contener archivos y carpetas; normalmente cada carpeta corresponde a un subdirectorio del directorio del proyecto. También proporciona el código para configurar los recursos del proyecto según sea necesario, además contiene puntos de extensión para que otros plug-in interactúen con los recursos de los usuarios.

El Workbench es la interfaz gráfica de usuario (GUI) del Eclipse, siendo el componente de la plataforma responsable de la presentación y de la coordinación de la misma. En adición a mostrar los menús y la barra de herramientas, está organizado en perspectivas conteniendo vistas y editores. También posee puntos de extensión para extender funcionalidades de la GUI para otros plug-in.

Estos dos plug-in son los principales dos componentes esenciales de la plataforma. Pero no son los únicos, además de ellos se encuentran otros como son el plug-in Help y Team.

El plug-in Team facilita el soporte del sistema de control de versiones (o manejo de configuración) para manejar los recursos de usuario en un proyecto y define el flujo de trabajo necesario para salvar y recuperar del repositorio.

El componente Help es un sistema de documentación extensible. Las herramientas provistas pueden adicionar documentación en formato HTML y, usando XML, define la estructura de navegación.

Además de los plug-in anteriormente mencionados existen otros que aunque no sean propiamente dicho parte intrínseca de la plataforma, se encuentran integrados al Eclipse SDK: El Java Develop Tooling (JDT) y el Ambiente de Desarrollo de Plug-in (Plug-in Development Enviroment PDE), el cual facilita el desarrollo de plug-in para poder extender más fácilmente las funcionalidades de la misma plataforma. En la sección 1.4 se abordan más profundamente estos componentes.

De alguna forma se ha hablado de plug-ins y de punto de extensión, pero no se ha dado ninguna definición concreta acerca de ellos.

1.2.2 Plug-ins y puntos de Extensión

Imagine un gigantesco rompecabezas, donde algunas piezas ya han sido conectadas, Esto formaría el núcleo en torno al cual el resto del rompecabezas se construiría. Los límites de las piezas son singularmente cortados para que todas las piezas encajen cómodamente donde van. Si Eclipse es el

rompecabezas, entonces las piezas son plug-in. Un plug-in es la unidad extensible más pequeña en Eclipse. El pude contener código, recursos, o ambos. (4)

La plataforma Eclipse consiste en acerca de 100 plug-ins trabajando juntos. A los límites entre estas piezas que permiten conectar un plug-in a otro se le denomina puntos de extensión. Los puntos de extensión es el mecanismo por el cual un plug-in puede adicionarle funcionalidades a otro.

1.2.3 Arquitectura de un plug-in

Los plug-in son conceptualmente simples. Cada plug-in posee un archivo nombrado MANIFEST.MF y otro plugin.xml definiendo varios aspectos de alto nivel como son los puntos de extensión usados y definidos por el plug-in, las dependencias de otros plug-in, una lista con las librerías requeridas, nombre y versión del mismo, entre otras.

El manifiesto del plug-in está representado por los dos archivos antes mencionados de los cuales se dará una breve descripción:

- MANIFEST.MF: Este archivo se crea dentro de la carpeta META-INF cuando se crea un proyecto plug-in. En él se encuentra las dependencias con otros plug-in y sus atributos como son el nombre, la versión, el proveedor entre otros.
- plug-in.xml: Es un archivo con formato XML que se encuentra en la raíz del proyecto y describe todas las extensiones realizadas por el plug-in y declara los puntos de extensión que el propio plug-in ha realizado.

Estos no son los únicos elementos dentro del directorio de un plug-in, pues se encuentran otros archivos y carpetas como son:

- icons: Directorio para iconos, usualmente en formato GIF.
- abaut.html: localización estándar usada para información de la licencia.
- Otros archivos: según sean necesitados.

1.3 Plug-in existente que permiten la agilización del desarrollo.

Desde el surgimiento de la programación, los desarrolladores han incursionado en del desarrollo de herramientas que agilicen el desarrollo en la producción de software, esta tarea no se ha quedado atrás

en el ambiente de desarrollo integrado Eclipse, la cual posee muchos plug-in encaminados a facilitar y agilizar el trabajo constante del desarrollador.

A continuación se hará un análisis de algunos plug-in los cuales son los más usados en la Universidad de las Ciencias Informáticas y en el mundo; los cuales ofrecen funcionalidades y soporte a distintos framework, agilizando el trabajo con estos y aumentando la calidad del mismo.

1.3.1 Spring IDE

El Spring Framework (también conocido simplemente como Spring) es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java. Por su diseño el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria.

Spring IDE es un plug-in de Eclipse que provee una interfaz gráfica de usuario para la configuración de archivos usados por Spring framework.

El Spring IDE se caracteriza por poseer las siguientes características o rasgos:

Naturaleza Spring: Le adiciona a un proyecto naturaleza de Spring y soporta una lista de archivos de configuración de beanes de Spring.

Constructor Incremental: El cual valida todas las modificaciones realizadas a los archivos de configuración de beanes de Spring en un proyecto de Spring.

Vista: Muestra un árbol con todos los proyectos Spring y sus archivos de configuración de beanes.

Decorador de Imágenes: Decora todos los proyectos de Spring, sus archivos de configuración de beanes y todas las clases Java que son usadas como beanes de Spring.

Grafo: Muestra todos los beanes (y sus relaciones) definidos en un único archivo de configuración o en varios archivos de configuraciones.

Editor de XML: contiene un Editor de XML para los archivos de configuración de beanes de Spring.

Extensiones de Eclipse: Provee facilidades de búsqueda para encontrar beanes definidos en el modelo básico de beanes.

Asistentes: Brinda asistentes para la creación de nuevos proyectos de Spring. (4)

1.3.2 Hibernate Tools

El Hibernate es un framework de Java muy poderoso, con alto rendimiento en la persistencia de objetos relacionales y servicios de consultas. Hibernate permite desarrollar clases persistentes siguiendo el idioma orientado a objetos (incluyendo asociaciones, herencia, polimorfismo, composición, y colecciones). Hibernate permite expresar consultas en su propia extensión portable de SQL (HQL Lenguaje de Consulta de Hibernate), como también en SQL nativo o con Criteria orientado a objeto y Example API.

Hibernate Tools es enteramente un conjunto de herramientas para trabajar con el framework de Hibernate, implementado como una suite integrada de plug-in para Eclipse.

El Hibernate Tools se caracteriza por poseer las siguientes características o rasgos:

Editor de Mapeo: Es un editor para los archivos XML de mapeos, soportando auto-completamiento y resaltado de sintaxis. El editor incluso soporta auto-completamiento semántico para nombre de clases Java, propiedades y nombre de campos, nombre de tablas y nombre de columnas.

Consola: La perspectiva Consola permite configurar la conexión a la base de datos, provee visualización de clases y de sus relaciones y permite ejecutar consultas HQL interactivamente contra la base de datos y buscar el resultado de la consulta.

Ingeniería Inversa: El más poderoso rasgo del plug-in es la ingeniería inversa de la base de datos, la cual puede generar el modelo de clases de dominio y los archivos de mapeo de Hibernate, los EJB 3 entity beans anotados y documentación en formato HTML, en segundos.

Asistentes: Muchos asistentes son provistos, incluyendo asistentes para rápidamente generar la configuración de archivos de Hibernate (hibérnate.cfg), y la configuración de la consola de Hibernate. (5)

Todos los plug-in mencionados anteriormente tienen una característica en común y es que agilizan el desarrollo a través de la generación y soporte de elementos arquitectónicos fuertemente aferrados a una arquitectura fija dictada por el framework que representan, y que no corresponden necesariamente con el alcance del contexto de la solución que se está desarrollando.

En el presente trabajo se propone una alternativa que soluciona los problemas anteriormente planteados, consistentes en un nuevo plug-in para Eclipse denominado GEAP plug-in que permite agilizar el desarrollo mediante la generación de código a partir de las especificaciones arquitectónicas de una aplicación dada.

1.4 Tecnologías utilizadas en el desarrollo del plug-in PGEA.

Como se ha podido apreciar Eclipse es una plataforma la cual no solamente permite desarrollar en distintos lenguajes de programación, sino que también viene con un armazón preparado para agregarle nuevas funcionalidades de cualquier tipo, el Entorno de Desarrollo Integrado Eclipse está preparado para desarrollar y ser desarrollado.

A continuación se dará a conocer un conjunto de herramientas que facilitan el desarrollo de un proyecto Plug-in.

1.4.1 Plug-ins

1.4.1.1 JDT

El proyecto JDT provee los plug-ins necesarios para cualquier IDE de Java que necesite soportar el desarrollo de aplicaciones, incluido plug-ins de Eclipse. Adiciona la naturaleza de proyecto Java y la perspectiva Java al Marco de trabajo Eclipse, además de cierto número de vistas, editores, asistentes, builders y herramientas para mezclar y refactorizar código. El proyecto JDT es la raíz de lo que permite al IDE Eclipse ser un ambiente de desarrollo autosuficiente.

El proyecto JDT está separado en componentes. Cada componente opera como un proyecto aparte, con su propio grupo de contribuyentes, categorías de errores y lista de correo.

Tabla 1: Componentes del JDT

Nombre	**Descripción**
APT	Infraestructura del proceso de anotación Java 5.0
Core	Estructura descentralizada de los IDE Java

Debug	Soporte de debug para Java
Text	Soporte para edición en Java
UI	Interfaz de usuario de los IDE Java

A continuación se darán a conocer algunas de las funcionalidades de estos componentes:

- APT: provee plug-in que añade soporte para el proceso de anotación de Java 5 al Eclipse. Un proceso de anotación de Java es un plug-in compilador que puede coleccionar información acerca del código fuente mientras está siendo compilado, genera tipos Java adicionales u otros archivos de recursos, y muestra advertencias y errores.
- Core: Es la infraestructura de los IDE Java e incluye:
  - Un compilador de Java incremental, implementado como un constructor de Eclipse. En particular, permite correr y depurar código que todavía contiene errores sin resolver.
  - Un Modelo Java que provee un API para navegar el árbol de elementos Java. El árbol de elementos Java define una vista de proyecto centrado en Java. Navega los elementos como fragmentos de paquetes, unidades de compilación, clases binarias, tipos, métodos, campos.
  - Modelo de documentación Java: provee un API para manipular un documento fuente estructurado de Java.
  - Asistente de código y soporte a selección de código.
  - Búsqueda Indexada basada en una infraestructura que es usada para buscar, asistencia de código, determinación de la jerarquía de tipo, y refactorización. El motor de búsqueda de Java puede encontrar con exactitud precisa coincidencias ya sea en fuentes o en binarios.
  - Soporte de evaluación en un contexto de depuración.
  - Formateado de código fuente.

- El Eclipse Debug Project no es un proyecto solo en sí mismo, sino que está dividido de hecho en dos distintos sub-proyectos: Platform Debug y JDT Debug.

  Platform Debug define interfaces para un modelo de depuración independiente del lenguaje con las características comunes de depuración de varios lenguajes, considerando que, JDT Debug provee una implementación de depuración a nivel de plataforma.

- Platform Text provee los elementos básicos para texto y editores de texto dentro del Eclipse y contribuye al editor de texto por defecto del Eclipse. Consta de cinco partes:
  - La infraestructura de texto: provee facilidades para manipulación de texto, manejo de posición, y notificación de cambio.
  - JFace Text: provee componentes de Interfaz de Usuario para editar y presentar texto. Ofrece soporte para estilo basado en regla. Completamiento de contenido y formateo.
  - El Text Editor framework: provee las implementaciones abstractas para un editor de texto de Eclipse.
  - El File Buffers plug-in: introduce búferes de archivo de texto para el acceso compartido al contenido de un archivo de texto.
  - Implementación concreta de un editor: El editor de texto por defecto de Eclipse.
- El JDT UI implementa la interfaz de usuario para el Java IDE. Ofrece varias contribuciones de trabajo para ver y manipular el código Java:
  - Package Explorer View: Muestra el árbol de elementos Java definido por el proyecto.
  - Type Hierarchy View: Muestra el sub y súper tipo de la jerarquía.
  - Java Outline View: Muestra la estructura de una unidad de compilación o archivo de clase.
  - Java Browsing Perspective: Permite navegar el modelo Java usando vistas separadas de proyecto, paquetes, tipos y miembros.
  - Asistentes para la creación de elementos Java: Proyectos Java, paquetes, clases, interfaces.

- Java Editor: coloreado de sintaxis, asistente de código de contenido especifico, resolución de código, asistente para importar, arreglo rápido y asistente rápido. (6)

Todas estas funcionalidades mencionadas y otras, son las que permiten desarrollar plug-in con gran potencial, y son necesarias si se quiere que el plug-in interactúe con aplicaciones Java o con recursos.

1.4.1.2 PDE

El Plug-in Development Enviroment (PDE) ofrece herramientas para crear, desarrollar, probar, depurar, construir y desplegar plug-in de Eclipse, fragmentos, rasgos, sitios de actualización y productos RCP.

PDE también ofrece un conjunto de herramientas OSGI, que lo hacen un ambiente ideal para programación de componentes, no sólo para desarrollar plug-in de Eclipse.

El sub-proyecto PDE está dividido en tres componentes principales, Build, UI y un conjunto de herramientas denominadas PDE API Tools. Cada uno de estos componentes opera como un proyecto independiente, con su propio grupo de contribuyentes, categoría de errores y lista de correo. Existen dos componentes adicionales en PDE, Doc que maneja la documentación de la ayuda e Incubator que desarrolla características que no son propias del SDK.

Tabla 2: Componentes del PDE

Nombre	**Descripción**
PDE Build	Herramienta basada en Ant y scripts para automatizar el proceso de construcción
PDE UI	Modelos, Constructores, Editores y más, para facilitar el desarrollo de plug-in en el Eclipse IDE.
PDE API Tools	El IDE Eclipse y herramientas integradas de construcción de procesos para mantener el API.
PDE Incubator	Desarrollo de nuevas herramientas que no están listas para ser añadidas al

	SDK del Eclipse.
PDE Doc	Documentación de ayuda para el PDE, compartida por los otros componentes.

A continuación se darán a conocer algunas de las funcionalidades de estos componentes:

- PDE Build: Su meta es facilitar la automatización del proceso de construcción de plug-in. Esencialmente, PDE Build produce scripts de Ant basados en información en tiempo de desarrollo provista por varias fuentes, por ejemplo, los archivos plugin.xml y build.properties. El scripts Ant generado, puede extraer los proyectos pertinentes desde un repositorio CVS, construir jars, Javadoc, fuentes Zips, poner todo junto en un formato listo para embarcar y enviar a una locación remota (e.g., una red local o un servidor de descarga).
- PDE UI: Ofrece un conjunto completo de herramientas para crear, desarrollar, probar, depurar y desplegar Eclipse plug-in, fragmentos, características, sitios de actualización y productos RCP.
- A continuación se muestra una lista pequeña de lo que el PDE UI ofrece al SDK del eclipse:
    - Form-Based Manifest Editors – multi-page editors: que centralmente manejan todos los archivos manifest de un plug-in o feature.
    - Herramientas RCP: asistentes y form-based editor que permiten definir, marcas, pruebas y exportar productos a múltiples plataformas.
    - New Project Creation Wizard: crea un nuevo plug-in, fragmento, característica y sitios de actualizaciones.
    - Import Wizard: importa plug-ins y características del archivo del sistema.
    - Export Wizard: asistentes que construyen, paquetes y exportan plug-ins, fragmentos y productos con un simple clic.
    - Launchers: prueban y depuran aplicaciones de Eclipse y paquetes OSGI.
    - Views: ofrece vistas que ayudan a los desarrolladores de plug-in a inspeccionar diferentes aspectos de su ambiente de desarrollo.

- Miscellaneous Tools: asistentes para externalizar y limpiar archivos manifest.
- Conversion Tools: asistentes para convertir un proyecto Java plano o JARs planos e un proyecto plug-in.
- Integracion con JDT: el archivo manifest de un plug-in participa en las búsquedas Java y refactorización.
- User Asistance Tools: Editores y herramientas para desarrollar ayuda a usuarios.
- Declarative Services Tools: Editores y validadores para servicios declarativos OSGI.

- PDE API Tools: asistirán a los desarrolladores en el mantenimiento del API, reportando los defectos del API como incompatibilidades binarias, números incorrectos de versiones de plug-ins y el uso de código no perteneciente al API entre plug-ins. Las herramientas estarán integradas en el SDK del Eclipse y serán usadas en el proceso de construcción automática. Específicamente la herramienta está diseñada para hacer lo siguiente:
  - Identificar problemas de compatibilidad binaria entre dos versiones o componente de software o producto.
  - Actualización del número de versión de plug-ins basado en esquema de versionamiento de Eclipse.
  - Provee nuevas etiquetas para los documentos Java y asistencia de código para anotar tipos con restricciones especiales.
  - Aprovecha la información existente (en el MANIFEST.IMF) para definir la visibilidad de los paquetes entre plug-ins.
  - Identifica el uso de código no perteneciente al API entre plug-ins.
  - Identifica fugas de tipo no pertenecientes al API dentro del API.
- PDE Incubator: Ofrece un lugar para crecer y probar nuevas ideas que caen en el espacio de desarrollo de un plug-in. (7)

1.5 Metodología de Desarrollo de Software

La metodología no es más que un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software. Compuesta por Tareas (actividades en que se dividen los procesos), Procedimientos (Define la forma de ejecutar la tarea), Técnica (herramienta utilizada para aplicar un procedimiento), Herramientas (herramientas de software que automatizan la aplicación) y el Producto (resultado de cada etapa).

Para el presente trabajo analizamos tres metodologías de las existentes en el mundo del software actual: RUP, XP y OpenUP, debido a su relevancia en el medio universitario y su aplicabilidad al problema científico que se trata de resolver.

1.5.1 Extreme Programing (XP)

Es una de las metodologías ágiles de desarrollo de software más exitosas en la actualidad. Se utiliza para proyectos de corto y mediano alcance, pequeño equipo y plazos apretados de entrega. La metodología consiste en un conjunto de prácticas de dudosa efectividad cuando se usan por separado, pero que juntas producen un aumento en la productiva y la habilidad del equipo para enfrentar los cambios inevitables del proceso de desarrollo. La introducción de los clientes como parte del proceso y la necesidad de espacios grandes y personal calificado son las principales desventajas de esta metodología.

1.5.2 RUP

Es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo, a través del UML, y trabajo de muchas metodologías utilizadas por los clientes. En RUP se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo, los 6 primeros de ingeniería y los tres últimos de apoyo. Cada flujo de trabajo cumple con algunas actividades específicas, tiene trabajadores específicos y produce artefactos también definidos. Además RUP presenta el concepto de fase como un estado del proyecto, donde Todos los flujos se aplican en todas las fases, si bien con cargas distintas de trabajo. RUP se caracteriza por ser dirigido por casos de uso, centrado en la arquitectura e iterativo e Incremental, lo que significa que cada fase se desarrolla en iteraciones que involucran actividades de todos los flujos de trabajo, obteniendo un producto que irá creciendo incrementalmente en

cada iteración. RUP es una metodología pesada, cuya duración de fases y requerimientos de documentación son muy grandes para proyectos pequeños o con poco personal.

1.5.3 OpenUP

Se deriva de una metodología basada en RUP llamada BUP, creado por IBM y donado a la fundación Eclipse. Conserva las características esenciales de RUP, pero eliminando varios de sus elementos menos utilizados. El resultado ha sido una metodología mucho más simple y ligera, definiéndose en el campo de proyectos pequeños que abarquen de 3 a 6 personas con un enfoque ágil e iterativo. Como RUP es una metodología con la cual el autor del presente trabajo está muy relacionado debido a que se imparte como parte del plan de estudios de la carrera en la UCI y debido a las características ágiles y de orientación a proyectos y equipos pequeños de OpenUP es que se escoge OpenUP por encima de metodologías como RUP y XP. El proceso propuesto por OpenUP conserva las características esenciales de RUP en el sentido de que se basa en casos de uso, desarrollo iterativo e incremental y es centrado en la arquitectura. Además al ser parte del Eclipse Process Framework ha sido probado en la práctica innumerables veces y presenta antecedentes sólidos para ser seleccionado como metodología de desarrollo para este trabajo.

1.6 Lenguaje Unificado de Modelado (UML)

El UML es un estándar, que tiene como objetivo proveer a los arquitectos del sistema que trabajan con análisis y diseño, de un lenguaje consistente para especificar, visualizar, construir y documentar los artefactos del sistema del software, como también para el modelado del negocio.

Este estándar representa la convergencia de las mejores prácticas en la industria de la tecnología de objetos. UML es el sucesor apropiado del lenguaje de modelado de objetos de tres anteriores métodos orientados a objetos (Booch[2], OMT[3], OOSE[4]). El UML es la unión de estos lenguajes de modelado y

[2] Booch: lenguaje de modelado de objetos y una metodología ampliamente usada en el diseño de software orientado a objetos. Fue desarrollada por Grady Booch mientras trabajaba para Rational Software

3 Object Modeling Technique. es una de las metodologías de análisis y diseño orientadas a objetos, más maduras y eficientes que existen en la actualidad

4 Object Oriented Software Engineering. Metodología de diseño orientado a objetos para la creación de Software creado por Ivar Jacobson en 1992.

además incluye expresiones adicionales para manejar los problemas de modelado que estos métodos no dirigieron totalmente. (9)

1.7 Visual Paradigm

Visual Paradigm es un producto que facilita a las organizaciones diseñar visualmente y con diagramas, integrar y desplegar sus aplicaciones empresariales y sus respectivas bases de datos. La herramienta ayuda al equipo de software a destacarse en el proceso de modelado-construcción-despliegue, maximizando y acelerando las contribuciones del equipo y los individuos. Además, soporta un set de lenguajes para la Generación de Código y la Ingeniería Inversa en Java, C++, PHP, entre otros. (10)

Conclusiones del capítulo

En este capítulo se hizo un análisis de los principales conceptos para lograr comprender el presente estudio. Se explicaron brevemente las tecnologías necesarias para la elaboración del sistema propuesto. Se hizo una breve reseña de la metodología que fue guía del proceso de desarrollo. Se realizó un análisis de los plug-in existentes que brindan funcionalidades similares a las del sistema propuesto.

CAPÍTULO 2

Características del Plug-in

Introducción:

En el presente capítulo se exponen los artefactos necesarios que fueron creados con guía de la metodología OpenUP. Se especifica qué funcionalidades brindará el plug-in para agilizar el proceso de creación de componentes en el desarrollo y a su vez fijarlo a pautas arquitectónicas previamente definidas; así como las características y estructura del plug-in.

El concepto de Elementos Arquitectónicos con el que se trabaja en el documento es el de aquellos elementos definidos por la arquitectura del proyecto, como clases y estructuras de paquetes modulares.

2.1 Funcionalidades a Automatizar:

El plug-in a realizar pretende automatizar el proceso de creación de clases marcada por una arquitectura definida por el proyecto y la creación de un conjunto de paquetes estructurados por la arquitectura, además de agilizar estos procesos.

2.2 Propuesta del Sistema:

El sistema pretende automatizar la creación de clases y de paquetes bien estructurados, a través de la creación de plantilla especificada por la arquitectura, a esta plantilla para la creación de clases se le denomina Tipo de elemento y a la plantilla de creación de paquetes estructurados de le denomina Módulo, el cual contiene un árbol de Paquete.

Contenido de la plantilla de creación de clases Tipo de elementos:

- Nombre de la plantilla.
- Prefijo que tendrá el nombre de la clase.
- Sufijo que tendrá el nombre de la clase.
- Nombre de la clase de la que extiende.

- Listado de interfaces que Implementa la clase.
- Listado de plantillas a las que hace referencia la misma.
- Nombre del paquete que alojará al tipo de clase que representa la plantilla.
- Listado de clases que referencia la clase que representa la plantilla.
- Nombre que recibirá la clase en el momento de ser creada.
- Identificador de instancia de la plantilla.
- Paquete indica en que paquete se generaran las instancias de esta plantilla.

Contenido de Paquete.

- Nombre del paquete en que este se encuentra contenido.
- Nombre del paquete.
- Listado de los paquetes que están alojados dentro del mismo.
- Listado de instancias de TipoElemento que alojara el mismo.

Contenido de la plantilla de creación de Módulo:

- Listado de los paquetes raíces.

2.3 Metodología y artefactos generados:

OpenUP es un Proceso Unificado ligero que aplica una aproximación iterativa incremental dentro de un ciclo de vida estructurado. Posee una filosofía ágil y pragmática que se enfoca en la naturaleza colaborativa del desarrollo de software.

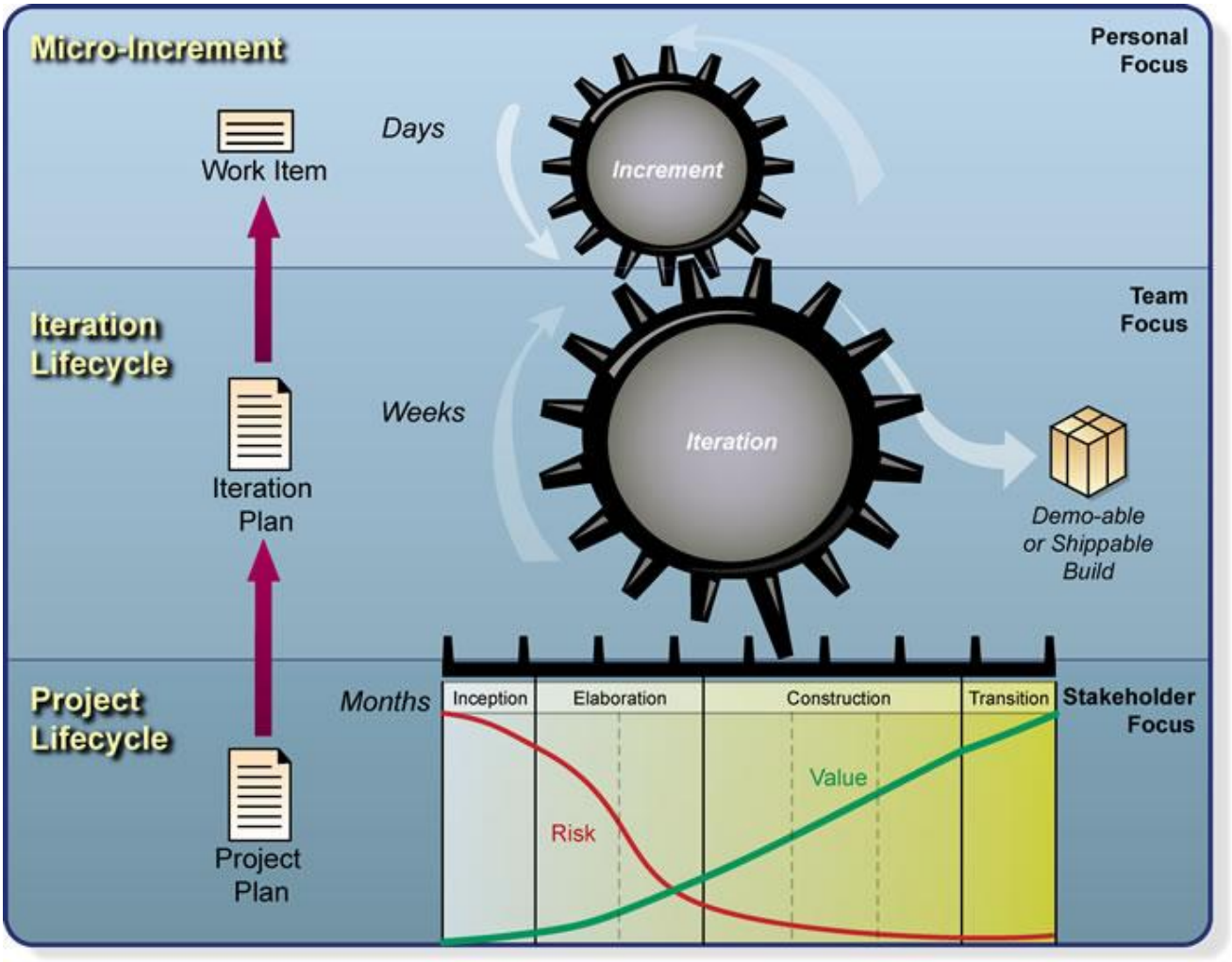


Figura 2: Capas de OpenUP: micro-incrementos, ciclo de vida de las iteraciones y ciclo de vida del proyecto

El esfuerzo personal en un proyecto OpenUP está organizado en **micro-incrementos**. Esto representa pequeñas unidades de trabajo que producen un paso de progreso del proyecto cuantificable y constante.

OpenUP divide el proyecto en iteraciones: planeadas en intervalos de tiempos fijos usualmente medido en semanas. Las iteraciones concentran al equipo en entregas incrementales de valor para los stakeholders en una manera predecible. El plan de iteración define qué debe ser entregado dentro de las iteraciones, y el resultado es un demo. Esto se realiza definiendo finas y bien engranadas tareas de una lista de

elementos de trabajo. OpenUP aplica un **ciclo de vida iterativo** que estructura cómo los micro incrementos son aplicados para entregar construcciones estables y cohesivas del sistema que incrementalmente progresan hacia los objetivos de la iteración.

OpenUP estructura el **ciclo de vida del proyecto** en cuatro fases: Inicio, Elaboración, Construcción y Transición. El ciclo de vida del proyecto provee a los stakeholders y a los miembros del equipo de visibilidad y puntos de decisión a lo largo de todo el proyecto. Esto permite la vigilancia eficaz, y permite tomar decisiones de hacer o no hacer en un tiempo apropiado. (8)

2.3.1 Artefactos generados:

OpenUP plantea la generación de varios artefactos como el documento visión. Este artefacto define la vista para los stakeholders de la solución técnica a desarrollar. Esta definición es especificada en términos de las necesidades claves y características de los stakeholders. El documento visión contiene un contorno de los requisitos centrales del sistema. (9)

2.3.2 Visión GEAP:

Declaración Del Problema.

Tabla 3: Visión GEAP Declaración del Problema

el problema de	Cómo agilizar el desarrollo de software mediante el uso, por parte de los programadores, de una herramienta de generación de código ajustada a las pautas de arquitectura previamente definidas para un proyecto en cualquiera de las plataformas Java
Afecta	Desarrolladores de proyectos en cualquiera de las plataformas Java.
Cuyo impacto es	Provoca una pérdida de productividad por parte de desarrolladores de poca y mediana experiencia
Una solución exitosa seria	Con la creación de un plug-in para Eclipse IDE se agilizará el desarrollo de software por parte de los programadores y se asegurará el cumplimiento de

	un gran porciento de las pautas de Arquitectura de Software definidas por los proyectos que usen Java.

Declaración de Posicionamiento del Producto.

Tabla 4: Visión GEAP Posicionamiento del Producto

Para	Desarrolladores de Java
Quien	Se necesita de una herramienta de generación de código que agilice el proceso de producción.
El nombre del producto	GEAP
Eso	Agiliza el desarrollo por parte de programadores y los rige a la arquitectura propia del proyecto.
Al contrario de	Seam-gen, Hibernate tools, Spring IDE.
Nuestro Producto	Permite la creación de elementos arquitectónicos sin ataduras a los framework que se utilicen en el proyecto.

Ambiente del Usuario

El número de proyectos que usan Eclipse IDE en la Universidad de las Ciencias Informáticas es el 32 por ciento de la totalidad de proyectos en la universidad, y dado el auge de la comunidad de software libre y las características de la Universidad, no se cree que disminuyan estos datos estadísticos.

El tiempo de demora actual de creación de elementos arquitectónicos como las clases, no es muy grande, pero si se analiza con cuidado, este tiempo no varía con el desarrollo del proyecto; pero si la cantidad de clases, que se generan en el transcurso del mismo aumentan considerablemente, este tiempo no es despreciable.

Las restricciones de este plug-in es que tiene que ser instalado en el Eclipse IDE 3.4.

Actualmente en el mundo y en la Universidad de las Ciencias Informáticas la plataforma Java se utiliza muchísimo debido a que la comunidad de software libre ha tomado auge, y por las potencialidades que esta plataforma brinda. Entre ellas es que una aplicación realizada en la plataforma Java puede correr sobre cualquier Sistema Operativo.

Tanto el Sean-gen, como el Hibernate Tools, como El Spring IDE son herramientas usadas frecuentemente por las potencialidades de los framework que representan; Nuestro plug-in no necesita integrase con estas herramientas mencionadas; pero tampoco dificulta el trabajo con las mismas.

Descripción del Producto

Necesidades y Características

Tabla 5: Visión GEAP Necesidades y Características

Necesidad	Prioridad	Características	Liberación planeada
Agilizar el proceso de creación de clases regidas por pautas arquitectónicas definidas	Alta	Se brindarán asistentes, y vistas	14/5/09
Agilizar el proceso de creación de paquetes y restricciones regidas por pautas arquitectónicas definidas	Alta	Se brindarán asistentes y preferencias.	14/6/09

Los requerimientos no funcionales definen los atributos de calidad necesarios del sistema, como también al ser requerimientos funcionales globales no son capturados en los artefactos de requisitos conductuales como los casos de uso.

2.3.3 Requerimientos no funcionales.

Usabilidad.

- Facilidad de uso y de comprensión.

Soporte.

- Fácil de instalar.
- Fácil de actualizar.
- Compatibilidad con Eclipse IDE 3.4.
- Fácil de mantener.

Interfaz de Usuario.

- Legible.
- Simple de usar.

Hardware.

Software.

- JDK 1.5 o superior
- Eclipse IDE 3.4

2.3.4 Modelo de Casos de Uso

Este artefacto captura un modelo de las funciones intencionales y el modelo del sistema y sirve como contrato entre el cliente y el equipo de desarrollo. (10)

Introducción

Identificación de los actores primarios y los casos de uso.

Apreciación Global:

El plug-in GEAP es un plug-in instalado en el Eclipse IDE 3.4, el propósito de este plug-in es agilizar el proceso de desarrollo mediante la generación de elementos arquitectónicos personalizados.

El plug-in requiere que este correctamente instalado en la versión de Eclipse 3.4

2.3.5 Diagrama de Casos de Uso

Figura 3 muestra el diagrama de Casos de Uso de plug-in GEAP.

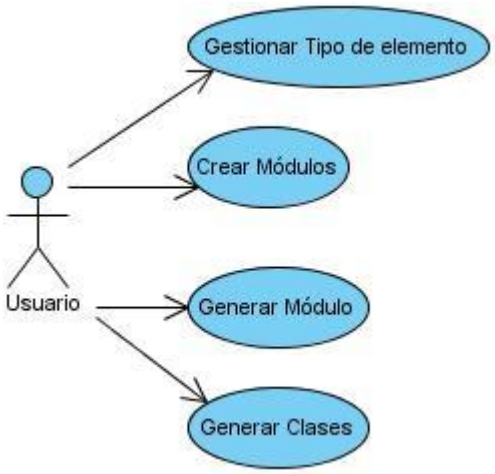


Figura 3: Diagrama de Casos de Uso plug-in GEAP

Actores:

Usuario: Este actor representa a la persona (desarrollador) que va a interactuar con el plug-in.

Casos de Uso:

Gestionar Tipo de Elemento: Este Caso de Uso describe cómo el Usuario gestiona elementos arquitectónicos, a través de operaciones como adicionar, editar y eliminar.

Crear Módulos: Este Caso de Uso describe cómo el Usuario crea una estructura de paquetes y de tipos de elemento, lo cual se considera como la representación de un Módulo.

Generar Módulo: Este Caso de Uso describe como el Usuario genera un módulo a través de las platillas de Módulo que se encuentren previamente definidas.

Generar Clases: Este Caso de Uso describe cómo el Usuario puede generar un árbol clases anidadas a partir de Tipos de Elementos anidados.

2.3.6 Definición de los casos de uso

Caso de Uso: Gestionar Tipo de Elemento

Breve Descripción del caso de uso.

Este Caso de Uso describe cómo el Usuario gestiona elementos arquitectónicos, a través de operaciones como adicionar, editar y eliminar.

Actores

Usuario.

Precondiciones

El Eclipse IDE 3.4 se encuentra abierto

El plug-in GEAP instalado.

Se encuentra creado un proyecto.

Flujo Básico de Eventos.

1. El caso de uso comienza cuando el Usuario, selecciona la vista Element type view
2. El sistema muestra la vista.

3. El Usuario selecciona el botón de la barra de herramienta Project configuration.
4. El sistema muestra un dialogo solicitando la selección de un proyecto.
5. El usuario selecciona un proyecto.
6. El Usuario ejecuta el botón ok.
7. El sistema muestra los Tipos de elementos, de existir alguno.
8. El sistema habilita la opción Add del popup menú.
9. El Usuario ejecuta la acción Add del popup menú.
10. El sistema muestra el asistente Element type wizard.
11. El sistema solicita los siguientes datos.
    a. Nombre del tipo de elemento.
    b. Prefijo que tendrá el nombre de la clase.
    c. Sufijo que tendrá el nombre de la clase.
    d. Nombre de la clase de la que extiende.
    e. Listado de interfaces que implementa la clase.
    f. Listado de los Tipos de elementos a los que hace referencia la misma.
    g. Nombre del paquete que alojara al Tipo de elemento.
    h. Listado de clases que referencia el Tipo de elemento.
12. El usuario introduce los datos.
13. El sistema valida los datos.
14. Se habilita el botón Finish
15. Finaliza el caso de uso Exitosamente.

Flujos Alternos.

Selección Tipo de elemento

- Si en el paso 9 del flujo básico de eventos el Usuario selecciona un Tipo de elemento, entonces
    1. El sistema habilita la opción Edit y Delete del popup menú.
    2. Si el Usuario selecciona la opción Edit ver escenario Edit.
    3. Si el Usuario selecciona la opción Delete ver escenario Delete.
    4. Finaliza el caso de uso con condición de falla.

Cancela Proyecto

- Si en el paso 6 del flujo básico de eventos el Usuario ejecuta el botón cancelar, entonces
    1. Finaliza el caso de uso con condición de falla.

Datos no válidos

Si en el paso 13 del flujo básico de eventos los datos no son válidos

1. El sistema refleja que hay error en los datos de entrada y el botón Finalizar se deshabilita.
2. Si el Usuario decide corregir los datos de entrada se regresa al paso 12 del flujo básico de eventos.
3. El caso de uso termina con condición de falla.

Llaves de Escenario

- Edit
    1. Edit, El sistema muestra el asistente Element type wizard.
    2. Edit, El sistema solicita la actualización de los siguientes datos.
        a. Nombre del tipo de elemento.
        b. Prefijo que tendrá el nombre de la clase.
        c. Sufijo que tendrá el nombre de la clase.
        d. Nombre de la clase de la que extiende.
        e. Listado de interfaces que implementa la clase.

f. Listado de los Tipos de elementos a los que hace referencia la misma.

g. Nombre del paquete que alojara al Tipo de elemento.

h. Listado de clases que referencia el Tipo de elemento.

3. Edit, Se regresa al paso 13 del flujo básico de eventos.

- Delete

1. Delete, El sistema muestra un mensaje de confirmación.
2. Delete, Si el Usuario oprime el botón ok se elimina el Tipo de elemento y se regresa al paso 16 del flujo básico de eventos.
3. Delete, Finaliza el caso de uso con condición de falla.

Post-condiciones

Realización Exitosa.

Se adiciona, edita o elimina un Tipo de elemento

Condición de Falla.

No se adiciona, edita o elimina un Tipo de elemento.

Requerimientos Especiales.

Caso de Uso: Crear Módulos

Breve Descripción del caso de uso.

Este Caso de Uso describe cómo el Usuario crea una estructura de paquetes y tipos de elementos, lo cual se considera como la representación de un Módulo.

Actores

Usuario.

Precondiciones

- El Eclipse IDE 3.4 se encuentra abierto y el plug-in GEAP instalado.

Flujo Básico de Eventos.

1. El caso de uso comienza cuando el Usuario selecciona la página de preferencia Modulo Preference.
2. El sistema muestra la opción de crear:

   Paquetes

   Tipos de elementos
3. El usuario si va a crea un paquete introduce el siguiente dato:

   Nombre del paquete.
4. El usuario si va a adicionar un Tipo de elemento selecciona el tipo de elemento
5. Si los datos son validos, se habilita el botón finalizar
6. Finaliza el caso de uso exitosamente.

Flujos Alternos.

Datos no Válidos.

- Si en el paso 5 del flujo básico los datos no son válidos, entonces
  1. El sistema refleja que hay error en los datos de entrada y el botón Finalizar se deshabilita.
  2. Si el usuario decide corregir los datos de entrada se regresa al paso 3.
  3. El caso de uso termina con una condición de falla.

Post- condiciones

Realización Exitosa.

Se crea el elemento Módulo

Condición de Falla.

No se crea el elemento Módulo.

Requerimientos Especiales.

Caso de Uso: Generar Módulo

Breve Descripción del caso de uso.

Este Caso de Uso describe cómo el Usuario genera un módulo a través de las platillas de módulo que se encuentren previamente definidas.

Actores

Usuario.

Precondiciones

- El Eclipse IDE 3.4 se encuentra abierto y el plug-in GEAP instalado.
- Se encuentre un proyecto.
- Se encuentre creado la configuración del Módulo en el plug-in.

Flujo Básico de Eventos.

1. El caso de uso comienza cuando el Usuario selecciona el wizard Generar Módulo.
2. El sistema solicita el paquete raíz a partir del cual se generará el Módulo.

   Nombre del paquete raíz.
3. El Usuario selecciona la raíz.
4. El sistema solicita los datos para generar los Tipo de Elemento dentro del Módulo

   Nombre de la clase.
5. El Usuario introduce los datos.
6. Si los datos son válidos, el sistema habilita el botón Finalizar.
7. Finaliza el caso de uso exitosamente.

Flujos Alternos.

Datos no Válidos.

- Si en el paso 6 del flujo básico los datos no son válidos, entonces
    1. El sistema refleja que hay error en los datos de entrada y el botón Finalizar se deshabilita
    2. Si el usuario decide corregir los datos de entrada se regresa al paso 5.
    3. El caso de uso termina con una condición de falla.

Post- condiciones

Realización Exitosa.

Se genera el Módulo

Condición de Falla.

No se genera el Módulo.

Requerimientos Especiales.

Caso de Uso: Generar Clases

Breve Descripción del caso de uso.

Este Caso de Uso describe cómo el Usuario puede generar un árbol clases anidadas a partir de Tipos de Elementos anidados.

Actores

Usuario.

Precondiciones

- El Eclipse IDE 3.4 se encuentra abierto y el plug-in GEAP instalado.
- Se encuentre un proyecto.
- Se encuentren Tipos de Elementos Creados.

Flujo Básico de Eventos.

1. El caso de uso comienza cuando el Usuario selecciona el wizard Generar Clases Anidadas.
2. El sistema solicita la selección de un Tipo de Elemento.
3. El Usuario selecciona un Tipo de Elemento, si existe alguno.
4. El sistema muestra un árbol con los Tipos de Elementos que tienen como raíz al Tipo de Elemento seleccionado.
5. El Usuario selecciona cual de los Tipos de elementos quiere generar.
6. El sistema solicita el siguiente dato por cada Tipo de elemento seleccionado.

   Nombre de la clase.

   Listado de los nombre de los objetos de las clases con las que se relaciona.
7. El Usuario introduce los datos.
8. El Sistema valida los datos.
9. Finaliza el caso de uso exitosamente.

Flujos Alternos.

Datos no Válidos.

- Si en el paso 8 del flujo básico los datos no son válidos, entonces
  1. El sistema refleja que hay error en los datos de entrada y el botón Finalizar se deshabilita
  2. Si el usuario decide corregir los datos de entrada se regresa al paso 7.
  3. El caso de uso termina con una condición de falla.

Post-condiciones

Realización Exitosa.

Se genera el árbol de clases seleccionadas.

Condición de Falla.

No se genera el árbol de clases.

Requerimientos Especiales.

2.3.7 Estructura del Diseño:

Este artefacto describe la realización de las funcionalidades requeridas del sistema y sirve como una abstracción del código fuente, es decir que describe los elementos del sistema para que puedan ser examinados y entendidos de una forma que no es posible leyendo el código fuente. (11)

El plug-in se denomina Generador de Elementos Arquitectónicos Personalizados (GEAP) y en la figura 1.5 se muestra su estructura básica. El mismo posee cuatro componentes: Wizard, Views, Preferences y Help. Los tres primeros ofrecen las funcionalidades necesarias para crear los elementos arquitectónicos personalizados y el Help guía al usuario en la utilización del plug-in.

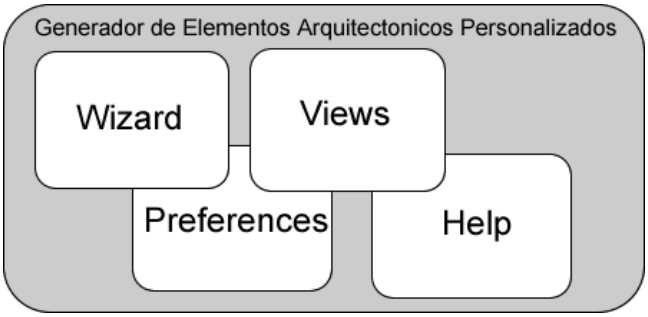


Figura 4: Estructura básica del plug-in GEAP.

2.3.8 Diagrama de paquetes:

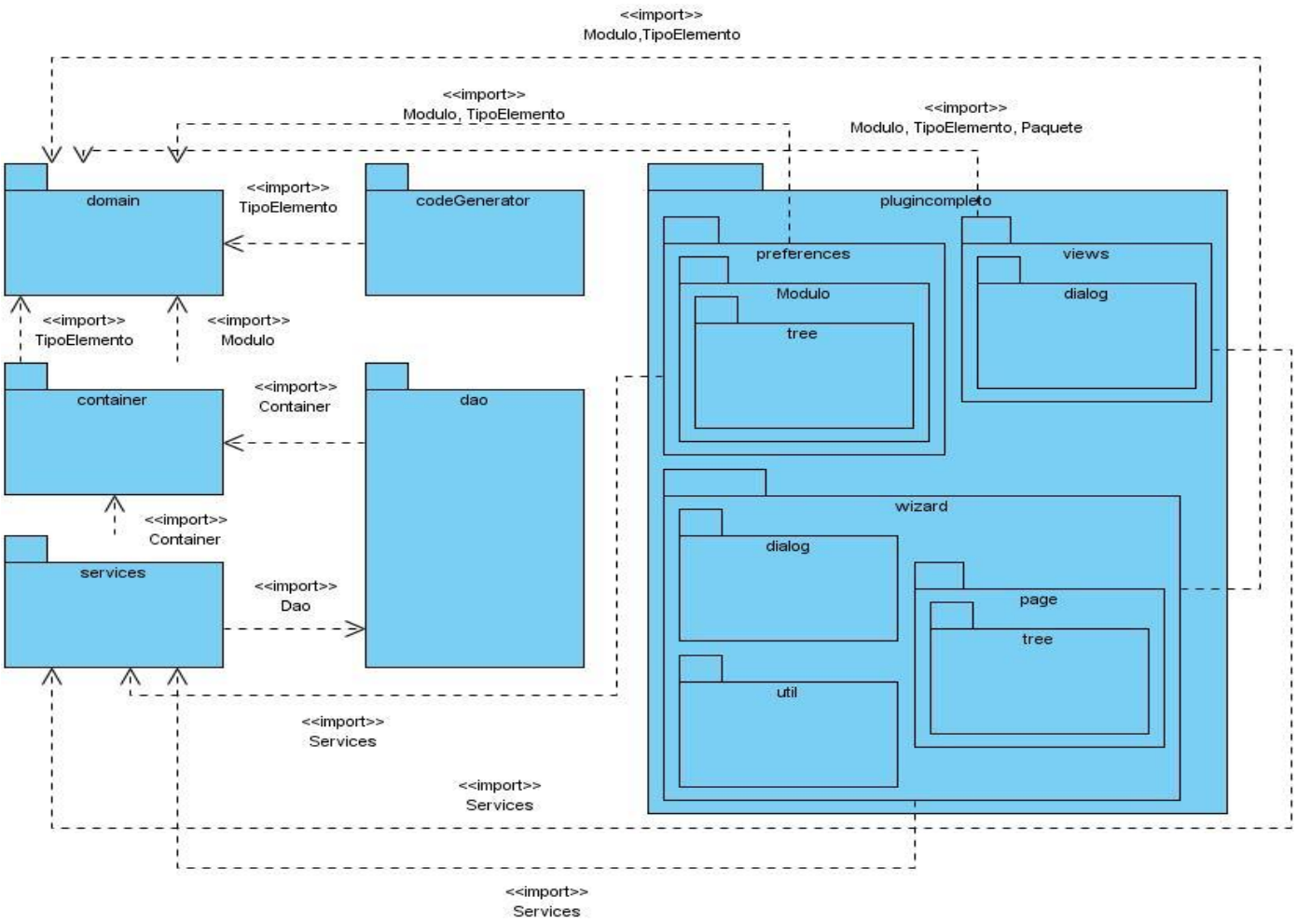


Figura 5: Diagrama de paquetes

El diagrama de paquetes expuesto anteriormente sirve para dar una vista de la organización interna de la estructura dentro del plug-in, donde se denotan tres paquetes fundamentales, tales como preferences, wizard, domain y views:

- El paquete preferences es el contenedor de las funcionalidades que permiten crear una estructura de paquetes para la posterior generación de módulos, a petición de los usuarios, además de establecer restricciones de que tipos de elementos se pueden generar dentro de ellos.
- El paquete wizard es el contenedor de las funcionalidades que permiten crear los tipos de elementos y la generación de código de los mismos.

- El paquete views es el contenedor de la vista del plug-in, la cual permite ver, adicionar, editar y eliminar Tipos de elementos.
- En el paquete domain se encuentran las entidades propias del dominio del sistema, como son los Tipo de elementos y los Módulos.
- El paquete codeGenerator ofrece las funcionalidades de generación de código.
- El paquete services es el encargado de cargar las estructuras almacenadas, ya sea el módulo y los Tipos de elementos a través de su comunicación con el paquete dao.
- El paquete dao es el encargado de la lógica de salva y carga de los Tipos de elementos y el módulo.
- El paquete container es el encargado de agrupar tanto los Tipos de elementos y el módulo para ser salvados o cargados.

Un diagrama de Clases representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas; pero no lo que ocurre cuando interactúan. A continuación se muestra el diagrama de clases.

2.3.9 Diagrama de clases:

Se realizó un diagrama por paquete debido a la gran cantidad de clases que contiene el diseño del plug-in; además se aprecian dos colores en todos los diagramas el color azul verdoso representa a las clases que son propias del paquete y las de color azul son las clases que pertenecen a otros paquetes pero se relacionan con las mismas.

Diagrama de clases del paquete codeGenerator:

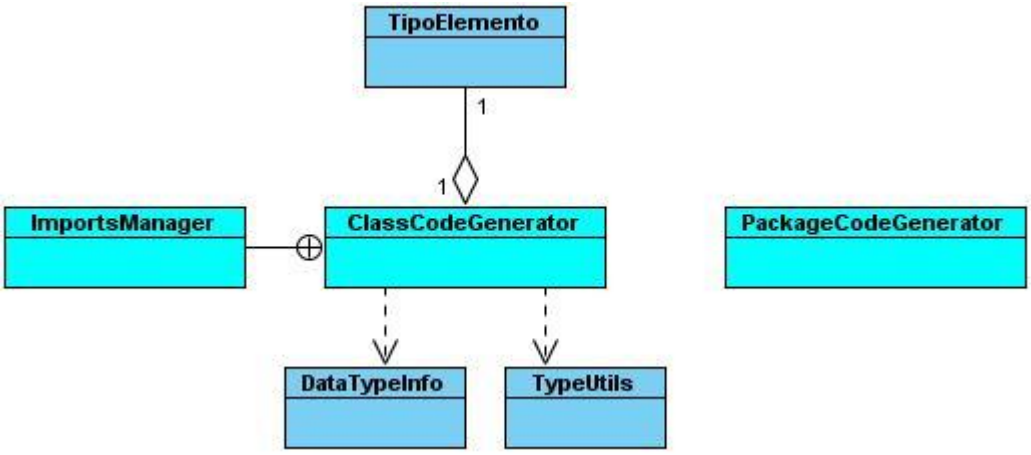


Figura 6: Diagrama de clases del paquete codeGenerator

Clase ClassCodeGenerator: es la entidad que permite la generación de código de las clases que representan Tipos de elementos.

Clase PackageCodeGenerator: es la entidad que permite la generación de paquetes.

Clase ImportsManager: es la entidad que se encuentra dentro de ClassCodeGenerator que permite la correcta importación de clases y paquetes.

Diagrama de clases del paquete container:

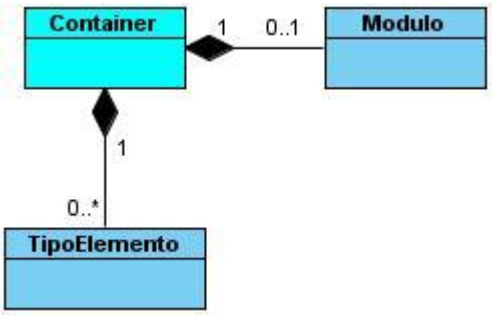


Figura 7: Diagrama de clases del paquete container

Clase Container: entidad que contiene la información de los módulos y los Tipos de elementos.

Diagrama de clases del paquete dao:

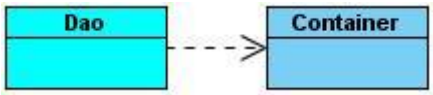


Figura 8: Diagrama de clases del paquete dao

Clase Dao: entidad que se encarga de la persistencia de los módulos y Tipos de elementos.

Diagrama de clases del paquete domain:

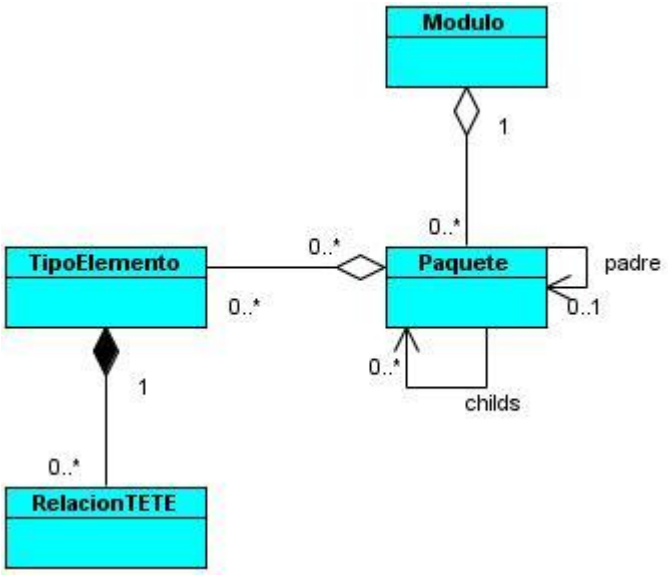


Figura 9: Diagrama de clases del paquete domain

Clase TipoElemento: es la entidad que define los datos de los tipos de elementos que posteriormente permitirán generar las clases.

Clase Modulo: es la entidad que realiza las operaciones sobre el árbol de paquetes que contiene.

Clase Paquete: es la entidad que conforma el árbol de elementos como paquete padre, paquetes hijos y Tipos de elementos que contiene.

Clase RelacionTETE: es la que contiene la información de la relación entre los Tipo de elementos

Diagrama de clases del paquete plugincompleto:

Figura 10: Diagrama de clases del paquete plugincompleto:

Clase Activator: es la entidad central del plug-in.

Diagrama de clases del paquete preferences:

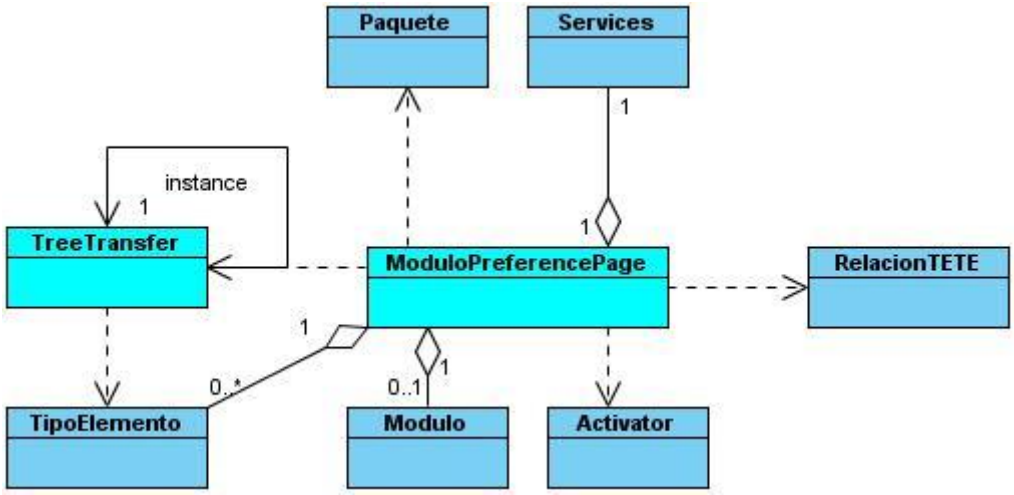


Figura 11: Diagrama de clases del paquete preferences

Clase ModuloPreferencePage: es la entidad que se encarga de manejar la página de preferencia del plug-in y todas las operaciones que permite la misma, como: crear módulos y realizar drag and drop entre los Tipos de elementos.

Clase TreeTransfer: es la entidad que permite obtener la información del elemento al cual se le realiza drag para posteriormente en la operación drop asignársela al otro elemento.

Diagrama de clases del paquete views:

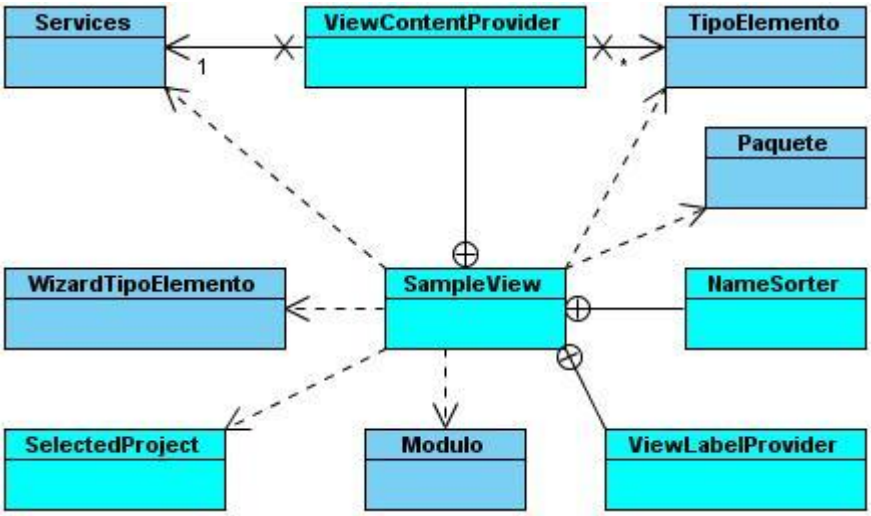


Figura 12: Diagrama de clases del paquete views

Clase SampleView: entidad que permite la creación de la vista y la cual contiene las operaciones que se realizan en la misma como: adicionar, editar y eliminar Tipo de elemento.

Clase ViewContentProvider: entidad que se encuentra dentro de SampleView y es la encargada de facilitarle a la misma el contenido a mostrar.

Clase ViewLabelProvider: es la entidad que se encuentra dentro de SampleView y es la encargada de facilitarle el nombre a los elementos que se mostrarán en la misma

Diagrama de clases del paquete wizard:

Clase NameSorter: es la entidad que se encuentra dentro de SampleView y es la encargada de establecer el orden en que se muestran los elementos de la vista.

Clase SelectedProject: entidad encarga de mostrar un diálogo para escoger el proyecto sobre el cual se realizarán las configuraciones.

Diagrama de clases del paquete wizard:

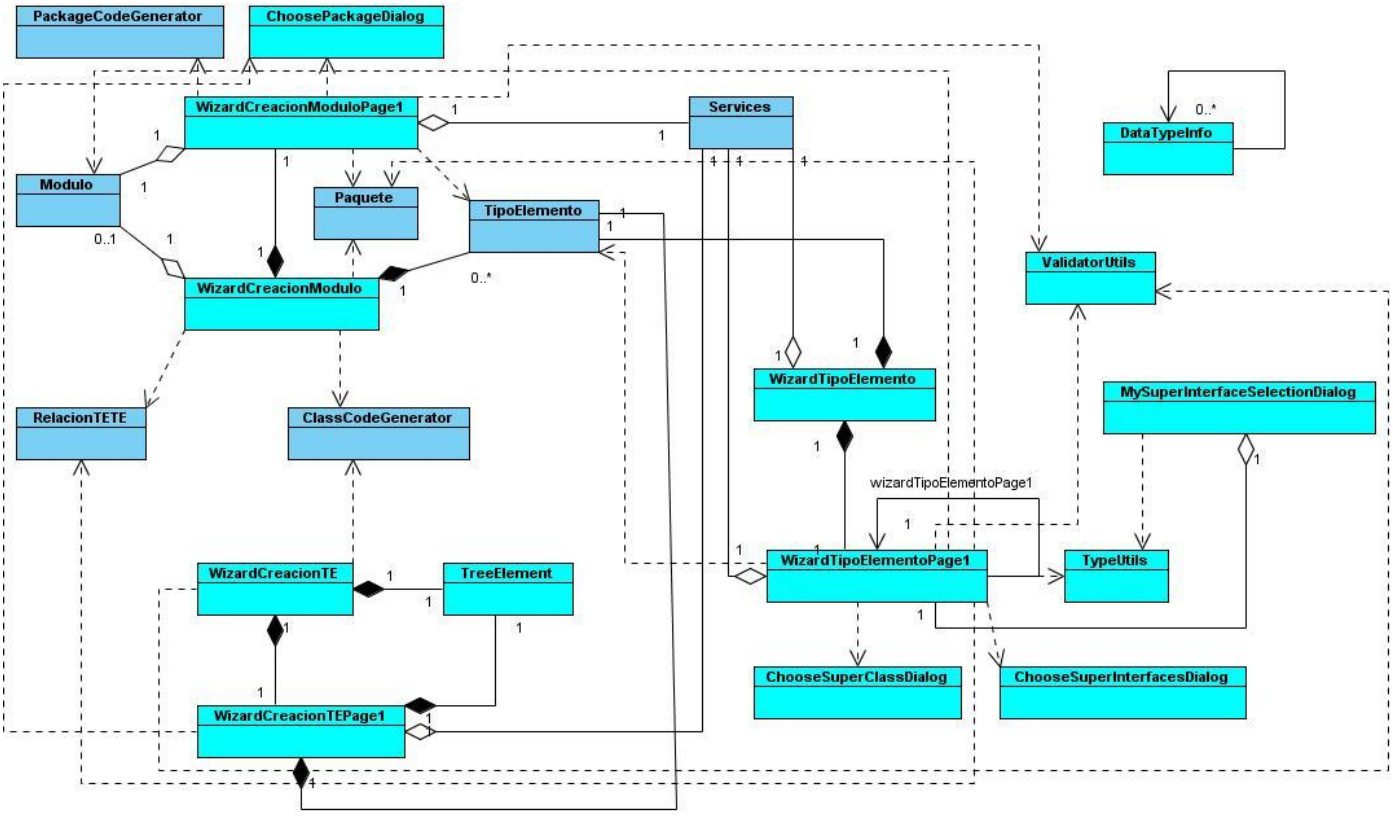


Figura 13: Diagrama de clases del paquete wizard

Clase WizardTipoElemento: Incorpora un asistente a los asistentes del Eclipse IDE 3.4. Esta clase juega el papel de contenedor para las páginas que contenga el asistente y contiene un conjunto de funcionalidades que permiten controlar las mismas.

Clase WizardTipoElementoPage1: constituye una página dentro del asistente WizardTipoElemento que le añade la funcionalidad de crear los Tipos de elementos.

Clase WizardCreacionTE: Incorpora un asistente a los asistentes del Eclipse IDE 3.4. Esta clase juega el papel de contenedor para las páginas que contenga el asistente y contiene un conjunto de funcionalidades que permiten controlar las mismas.

Clase WizardCreacionTEPage1: constituye una página dentro del asistente WizardCreacionTE que le añade la funcionalidad de generar los Tipos de elementos.

Clase WizardCreacionModulo: Incorpora un asistente a los asistentes del Eclipse IDE 3.4. Esta clase juega el papel de contenedor para las páginas que contenga el asistente y contiene un conjunto de funcionalidades que permiten controlar las mismas.

Clase WizardCreacionModuloPage1: constituye una página dentro del asistente WizardCreacionModulo que le añade la funcionalidad de generar los módulos.

Clase TreeElement: entidad que contiene el árbol de Tipos de elementos y las operaciones que se pueden realizar sobre la misma.

Clase ChoosePackageDialog: Dialogo que permite escoger un paquete, ofreciendo sugerencias.

Clase ChooseSuperClassDialog: Dialogo que permite escoger una clase, ofreciendo sugerencias.

Clase ChooseSuperInterfacesDialog: entidad que maneja al dialogo MySuperInterfaceSelectionDialog

Clase MySuperInterfaceSelectionDialog: dialogo que permite la selección de varias interfaces ofreciendo funcionalidades de sugerencia.

Clase TypeUtils: entidad encargada de transformar un tipo de dato a un simple String.

Clase ValidatorUtils: entidad que se encarga de validar los nombres de clases y paquetes.

Diagrama de clases del paquete services:

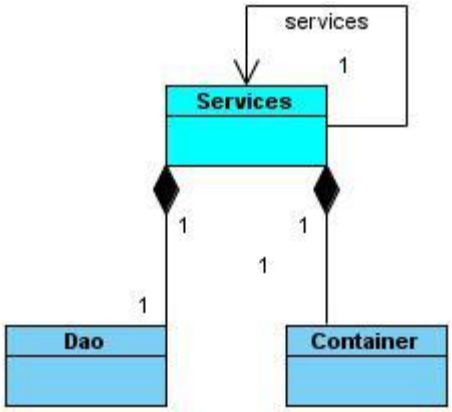


Figura 14: Diagrama de clases del paquete services

Clase Services: entidad encargada de ofrecer los servicios de salva y carga de las configuraciones de los Tipos de elementos y del módulo.

Conclusiones del Capitulo.

En este capítulo se exponen las principales características y funcionalidades del plug-in GEAP, que permitirán el desarrollo ágil y conciso de software. Además se especifican los artefactos generados con la guía de la metodología OpenUP, así como las responsabilidades fundamentales de cada una de las clases y paquetes que componen al plug-in, para lograr una idea de las potencialidades del mismo y de la complejidad de su construcción.

CAPÍTULO 3

Prueba y validación de la solución.

Introducción:

En el presente capítulo se exponen las pruebas realizadas al software así como el impacto en la productividad del plug-in GEAP.

La práctica de pruebas en la construcción de software reduce el tiempo de desarrollo reduciendo la cantidad de tiempo necesario para integrar y estabilizar versiones. Mejora la productividad encontrando y arreglando errores rápidamente, además de incrementar la calidad global del software garantizando que todo el código nuevo ha sido probado, y a todo el código existente se le aplican pruebas de regresión antes de ser integrados a la base central de código.

3.1 Casos de Prueba:

Tabla 6: Prueba Módulo Preference

Nombre de la Prueba	Módulo Preference
Caso de Uso Probado:	Crear Módulos
Descripción de la Prueba:	Se crea la estructura del módulo, adicionando paquetes y tipos de elementos existentes, así como la realización de drag-and drop de un Tipo de Elemento hacia un atributo de un Tipo de Elemento con el que es relacionado.
Pre-condiciones	El Eclipse IDE 3.4 se encuentra abierto y el plug-in GEAP instalado.
Post-condiciones	Se crea el elemento Módulo.

Notas:	Tanto los tipos de Elementos como los paquetes se crean utilizando un popup menú. Se puede renombrar un paquete preciando la tecla F2, se puede eliminar una estructura de paquetes y de Tipos de Elementos preciando la tecla Delete en el elemento deseado
Resultado **(Pasa/Falla/aviso/incompleto)**	

	Pasos de Prueba	**Resultados Esperados de Prueba**	**P**	**F**
1.	Crear un paquete y no darle nombre	Se debe eliminar automáticamente el paquete	x	
2.	Crear un paquete y nombrarlo	Se debe crear el paquete	x	
3.	Crear paquetes anidados	Se deben crear los paquete dentro del paquete seleccionado como padre	x	
4.	Crear tipos de elementos dentro de los paquetes	Se deben crear los tipos de elementos dentro del paquete seleccionado como padre	x	
5.	Crear un tipo de elemento dentro de otro Tipo de elemento	No permite hacerlo	x	
6.	Arrastrar con el mouse un Tipo de Elemento hacia un atributo de igual tipo dentro de otro Tipo de Elemento y soltar el mouse.	Se crea una relación entre El Tipo de elemento y el atributo del otro Tipo de Elemento	x	

Nombre de la Prueba	Módulo Preference
Caso de Uso Probado:	Crear Módulos
Descripción de la Prueba:	Se crea la estructura del módulo, adicionando paquetes y tipos de elementos existentes, así como la realización de drag-and drop de un Tipo de Elemento hacia un atributo de un Tipo de Elemento con el que es relacionado.
Pre-condiciones	El Eclipse IDE 3.4 se encuentra abierto y el plug-in GEAP instalado.
Post-condiciones	Se crea el elemento Módulo.
Notas:	Tanto los tipos de Elementos como los paquetes se crean utilizando un popup menú. Se puede renombrar un paquete preciando la tecla F2, se puede eliminar una estructura de paquetes y de Tipos de Elementos preciando la tecla Delete en el elemento deseado
Resultado **(Pasa/Falla/aviso/incompleto)**	

	Pasos de Prueba	**Resultados Esperados de Prueba**	**P**	**F**
7.	Arrastrar con el mouse un Tipo de Elemento hacia un atributo que no sea de igual tipo dentro de otro Tipo de Elemento y soltar el mouse.	No se crea una relación entre El Tipo de elemento y el atributo del otro Tipo de Elemento	x	
8.	Arrastrar con el mouse un Paquete hacia un atributo de un Tipo de Elemento y soltar el mouse.	No se permite	x	

Nombre de la Prueba	Módulo Preference
Caso de Uso Probado:	Crear Módulos
Descripción de la Prueba:	Se crea la estructura del módulo, adicionando paquetes y tipos de elementos existentes, así como la realización de drag-and drop de un Tipo de Elemento hacia un atributo de un Tipo de Elemento con el que es relacionado.
Pre-condiciones	El Eclipse IDE 3.4 se encuentra abierto y el plug-in GEAP instalado.
Post-condiciones	Se crea el elemento Módulo.
Notas:	Tanto los tipos de Elementos como los paquetes se crean utilizando un popup menú. Se puede renombrar un paquete preciando la tecla F2, se puede eliminar una estructura de paquetes y de Tipos de Elementos preciando la tecla Delete en el elemento deseado
Resultado **(Pasa/Falla/aviso/incompleto)**	

	Pasos de Prueba	**Resultados Esperados de Prueba**	**P**	**F**
9.	Presionar el botón ok para guardar la estructura del Módulo	Se guarda la estructura del Módulo	x	
10.	Presionar el botón cancelar para rechazar los cambios realizados	Se cancela los cambios realizados a la estructura del Módulo.	x	

Tabla 7: Datos de Prueba Módulo Preference

Tabla de Datos de Prueba					
	1	**2**	**3**	**4**	**5**
Nombre del Paquete	raiz	domain	dao	service	facade

Tabla 8: Prueba Crear Tipos de Elementos

Nombre de la Prueba	Crear Tipos de Elementos			
Caso de Uso Probado:	Gestionar Tipo de elemento (flujo principal de eventos)			
Descripción de la Prueba:	Se crea un Tipo de Elemento			
Pre-condiciones	El Eclipse IDE 3.4 se encuentra abierto y el plug-in GEAP instalado. Se encuentra creado un proyecto.			
Post-condiciones	Se crea el Tipo de elemento			
Notas:				
Resultado (Pasa/Falla/aviso/incompleto)				
	Pasos de Prueba	**Resultados Esperados de Prueba**	**P**	**F**
1.	Introducir el nombre del Tipo de Elemento	No se muestra ningún aviso	x	
2.	No introducir el nombre del Tipo de Elemento	Se muestra aviso de error: El nombre es un campo obligatorio Tipo de Elemento	x	
3.	Introducir el prefijo del Tipo de Elemento Con la primera letra en mayúscula y sin caracteres Extraños.	No se muestra ningún aviso	x	
4.	Introducir el prefijo del Tipo de Elemento Con la primera letra en mayúscula y con caracteres Extraños.	Se muestra aviso de error: El Prefijo debe comenzar con letra mayúscula y no puede contener caracteres extraños.	x	

Nombre de la Prueba	Crear Tipos de Elementos
Caso de Uso Probado:	Gestionar Tipo de elemento (flujo principal de eventos)
Descripción de la Prueba:	Se crea un Tipo de Elemento
Pre-condiciones	El Eclipse IDE 3.4 se encuentra abierto y el plug-in GEAP instalado. Se encuentra creado un proyecto.
Post-condiciones	Se crea el Tipo de elemento
Notas:	
Resultado **(Pasa/Falla/aviso/incompleto)**	

	Pasos de Prueba	**Resultados Esperados de Prueba**	**P**	**F**
5.	Introducir el prefijo del Tipo de Elemento Con la primera letra en minúscula y sin caracteres Extraños.	Se muestra aviso de error: El Prefijo debe comenzar con letra mayúscula y no puede contener caracteres extraños.	x	
6.	Introducir el prefijo del Tipo de Elemento Con la primera letra en minúscula y con caracteres Extraños.	Se muestra aviso de error: El Prefijo debe comenzar con letra mayúscula y no puede contener caracteres extraños.	x	
7.	Introducir el sufijo del Tipo de Elemento sin caracteres Extraños.	No se muestra ningún aviso	x	
8.	Introducir el sufijo del Tipo de Elemento con caracteres Extraños.	Se muestra aviso de error: El Sufijo no puede contener caracteres extraños.	x	

Nombre de la Prueba	Crear Tipos de Elementos
Caso de Uso Probado:	Gestionar Tipo de elemento (flujo principal de eventos)
Descripción de la Prueba:	Se crea un Tipo de Elemento
Pre-condiciones	El Eclipse IDE 3.4 se encuentra abierto y el plug-in GEAP instalado. Se encuentra creado un proyecto.
Post-condiciones	Se crea el Tipo de elemento
Notas:	
Resultado **(Pasa/Falla/aviso/incompleto)**	

	Pasos de Prueba	**Resultados Esperados de Prueba**	**P**	**F**
9.	Introducir nombre del paquete contenedor sin caracteres extraños	No se muestra ningún aviso	x	
10.	Introducir nombre del paquete contenedor con caracteres extraños	Se muestra aviso de error: El paquete contenedor no puede contener caracteres extraños.	x	
11.	Escoger la clase base	No se muestra ningún aviso, el campo es opcional	x	
12.	Escoger las Interfaces base	No se muestra ningún aviso, el campo es opcional	x	
13.	Escoger las clases de los atributos por defecto	No se muestra ningún aviso, el campo es opcional	x	

Nombre de la Prueba	Crear Tipos de Elementos
Caso de Uso Probado:	Gestionar Tipo de elemento (flujo principal de eventos)
Descripción de la Prueba:	Se crea un Tipo de Elemento
Pre-condiciones	El Eclipse IDE 3.4 se encuentra abierto y el plug-in GEAP instalado. Se encuentra creado un proyecto.
Post-condiciones	Se crea el Tipo de elemento
Notas:	
Resultado **(Pasa/Falla/aviso/incompleto)**	

	Pasos de Prueba	**Resultados Esperados de Prueba**	**P**	**F**
14.	Introducir el nombre de de los atributos por defecto	Se muestra aviso de error: el nombre de los atributos por defecto no pueden contener caracteres extraños.	x	
15.	Escoger los Tipos de Elementos con los que se relaciona por defecto.	No se muestra ningún aviso, el campo es opcional	x	
16.	Introducir el nombre de de los Tipo de Elemento defecto	Se muestra aviso de error: el nombre de los Tipo Elemento por defecto no pueden contener caracteres extraños.	x	
17.	Se presiona el botón Finish	Se guarda el Tipo de Elemento	x	
18.	Se presiona el botón cancelar	No se guarda el Tipo de Elemento	x	

Tabla 9: Datos de Prueba Crear Tipos de Elementos

Tabla de datos de prueba					
	1	**2**	**3**	**4**	**5**
Nombre	Entidad	Dao	Service	Facade	Nuevo
Prefijo	Mi	mi	M@	m@	
Sufijo	Impl	im@pl	ímpl	iñpl	
Clase Base	AbstractBorder	SAXParser	SAAJResult	D3DContext	
Interfaces Base	Serializable	SaslServer	SavePoint	Scaner	
Clase y nombre del objeto	String obj	Integer var1	Buffer buf	Parser var2	
Tipo de Elemento y nombre de objeto	Entidad	Dao	Service	Facade	

Tabla 10: Prueba Generar Módulos

Nombre de la Prueba	Generar Módulos
Caso de Uso Probado:	Generar Módulos
Descripción de la Prueba:	Se genera un módulo previamente definido, con su estructura de paquetes y clases que lo conforman.

Pre-condiciones	El Eclipse IDE 3.4 se encuentra abierto y el plug-in GEAP instalado. Se encuentre un proyecto. Se encuentre creada la configuración del Módulo en el plug-in.
Post-condiciones	Se genera el Módulo
Notas:	Se le debe poner un nombre de clase a los elementos que se van a generar
Resultado **(Pasa/Falla/aviso/incompleto)**	

	Pasos de Prueba	**Resultado Esperados de Pruebas**	**P**	**F**
19.	Seleccionar el paquete a partir del cual se generara el Módulo.	No se muestra ningún aviso	x	
20.	Introducir el nombre de las clases con la primera letra mayúscula y sin caracteres extraños	No se muestra ningún aviso	x	
21.	Introducir el nombre de las clases con la primera letra minúscula y sin caracteres extraños	Se muestra aviso de error: El nombre de las clases debe comenzar con letra mayúscula y no puede contener caracteres extraños.	x	
22.	Introducir el nombre de las clases con la primera letra minúscula y con caracteres extraños	Se muestra aviso de error: El nombre de las clases debe comenzar con letra mayúscula y no puede contener caracteres extraños.	x	

Nombre de la Prueba	Generar Módulos
Caso de Uso Probado:	Generar Módulos
Descripción de la Prueba:	Se genera un módulo previamente definido, con su estructura de paquetes y clases que lo conforman.
Pre-condiciones	El Eclipse IDE 3.4 se encuentra abierto y el plug-in GEAP instalado. Se encuentre un proyecto. Se encuentre creada la configuración del Módulo en el plug-in.
Post-condiciones	Se genera el Módulo
Notas:	Se le debe poner un nombre de clase a los elementos que se van a generar
Resultado **(Pasa/Falla/aviso/incompleto)**	

	Pasos de Prueba	**Resultado Esperados de Pruebas**	**P**	**F**
23.	Se oprime el botón Finish	Se genera el Módulo con sus paquetes y clases.	x	
24.	Se oprime el botón Cancel	No se genera el Módulo.	x	

Tabla 11: Datos de Prueba Generar Módulos

Tabla de datos de prueba					
	1	**2**	**3**	**4**	**5**

Nombre de clase	Entidad	domain	d@o	F@cade	

3.2 Validación en entorno real:

Para la validación el plug-in se utilizó una muestra de 10 desarrolladores divididos en dos grupos de prueba de 5 cada uno, donde el Grupo #1 utilizó el plug-in GEAP con una previa capacitación del trabajo con el mismo y el Grupo# 2 lo hizo sin la posibilidad de su utilización en el desarrollo del proyecto, del cual se dio una imagen con la estructura inicial del mismo. A cada grupo de desarrolladores se le asignaron dos roles:

- Arquitecto: un miembro.
- Desarrollador: cuatro miembros.

Además se establecieron 3 fases para la prueba y un grupo de tareas por roles para cada una de las fases:

- **Concepción de la arquitectura**: momento en que el arquitecto concibe la arquitectura; dígase framework que se van a utilizar, de utilizarse alguno, si se utilizará arquitectura en capas, y de ser así cuántas, qué patrones de diseño se utilizarán, etc...
- **Conceptualización y configuración de elementos arquitectónicos**: momento en que el arquitecto conceptualiza, configura y decide los principales elementos arquitectónicos, así como concibe el esqueleto del proyecto; dígase configuración de paquetes dentro del proyecto, de archivos XML, tipos de clases, estable como debe ser la comunicación entre módulos, sub-módulos y sub-sistema, entre otras y se reúne con los miembros del proyecto y explica la arquitectura de la misma.
- **Implementación de los elementos arquitectónicos**: momento en que los desarrolladores ya conocen la arquitectura del proyecto y pasan a la implementación.

Para la realización de la prueba se le entrega a los arquitectos un modelo con los elementos arquitectónicos, así como la estructura del proyecto; igualando el tiempo de desarrollo de ambos grupos en la primera fase, siendo entonces este tiempo despreciable para la prueba.

El modelo propone 4 elementos arquitectónicos como son:

- Entidad: define a todas las clases que sean persistentes.
    - Prefijo: no posee.
    - Sufijo: no posee.
    - Clase base: EntidadPersistenteBase.
    - Interfaces base: no posee.
    - Atributos por defecto: String atributo.
    - Elementos arquitectónicos con los que se relaciona: no posee.
- Dao: define a todas las clases que manejan la persistencia de los objetos persistentes.
    - Prefijo: no posee.
    - Sufijo: DaoImpl.
    - Clase base: DaoGenericoImpl.
    - Interfaces base: java.io.Serializable.
    - Atributos por defecto: no posee.
    - Elementos arquitectónicos con los que se relaciona: no posee.
- Service: define a todas las clases que ofrecen servicio sobre una Entidad.
    - Prefijo: no posee.
    - Sufijo: ServiceImpl.
    - Clase base: no posee.
    - Interfaces base: no posee.
    - Atributos por defecto no posee.
    - Elementos arquitectónicos con lo que se relaciona: Dao.
- Facade: define a todas las clases que representan puntos de entrada y salida de un módulo.
    - Prefijo: no posee.

- Sufijo: FacadeImpl.
- Clase base: javax.xml.parsers.SAXParser.
- Interfaces base: no posee.
- Atributos por defecto: Integer num.
- Elementos arquitectónicos con los que se relaciona: Service.

Estructura de clases y paquetes del modelo:

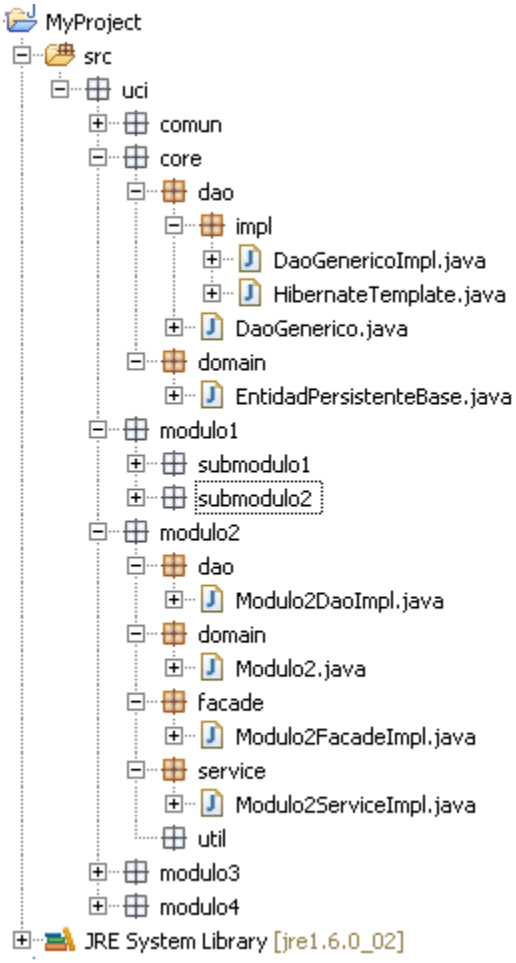


Figura 15: Estructura del proyecto plug-in GEAP

En la figura anterior se puede apreciar la estructura del proyecto. Los paquetes que no se encuentran expandidos, contienen la misma estructura interna que la del módulo “modulo2”.

Clases del paquete core.

EntidadPersistenteBase:

```
package uci.core.domain;

import java.io.Serializable;

public class EntidadPersistenteBase implements Serializable
{
    private static final long serialVersionUID = 8768184480700178479L;
    protected Integer id;

    public Integer getId()
    {
        return id;
    }
    public void setId(Integer id)
    {
        this.id = id;
    }
}
```

DaoGenericoImpl.

```
package uci.core.dao.impl;

import java.util.Collection;

public class DaoGenericoImpl implements DaoGenerico
{
    protected HibernateTemplate hibernateTemplate;
    @Override
    public void actualizar(Object obj){}
    @Override
    public void actualizar(Collection<?> colection){}
    @Override
    public void eliminar(Object obj){}
    @Override
    public void eliminar(Collection<?> colection){}
    @Override
    public void salvar(Object obj){}
    @Override
    public void salvar(Collection<?> colection){}
}
```

DaoGenerico.

```
package uci.core.dao;

import java.util.Collection;

public interface DaoGenerico
{
    public void salvar(Object obj);
    public void salvar(Collection<?> colection);
    public void eliminar(Object obj);
    public void eliminar(Collection<?> colection);
    public void actualizar(Object obj);
    public void actualizar(Collection<?> colection);
}
```

HibernateTemplate:

```
package uci.core.dao.impl;

public class HibernateTemplate {

}
```

En la segunda fase de la prueba, los arquitectos de los dos grupos se reúnen con sus desarrolladores y les explican la arquitectura y sus principales elementos arquitectónicos así como las configuraciones necesarias para desarrollar. Hasta el momento el tiempo transcurrido sigue siendo despreciable para la prueba; pero al acabarse la reunión el arquitecto del Grupo #1 comienza a configurar la estructura de los módulos y a conformar los tipos de elementos arquitectónicamente significativos, mientras que los del Grupo #2 ya comienzan a desarrollar. El tiempo que se demora el arquitecto del Grupo #1 en hacer dicha configuración es de: 2 minutos.

En la tercera fase de la prueba, se apreció que el programador más lento del Grupo #2 se demoró en crear dos módulos 16 minutos. Mientras que en el Grupo #1 el tiempo de creación de módulos es el mismo y a su vez el mínimo, ya que esta tarea se ha automatizado con el plug-in GEAP con un tiempo para la creación de dos módulos de 15 segundos.

Tabla de Resultados por fases:

Tabla 12: Resultados Por Fases

Fases	Grupo#2		Grupo#1	
	Arquitecto	Desarrollador más lento	Arquitecto	Desarrollador más lento
Fase 1	-	-	-	-

Fase 2	-	-	00:02:00	-
Fase 3	-	00:16:00	-	00:00.25
Tiempo acumulado por rol	-	00:16:00	00:02:00	-
Tiempo Total	00:16:00		00:02:25	

La prueba de validación no solamente constó de los procedimientos anteriores, sino que también se le orientó a los dos grupos que aumentaran en cinco el número de elementos arquitectónicos de los módulos y sub-módulos como: Facade, Service y Dao. El Grupo #1 tuvo una demora en la creación de 75 elementos arquitectónicos de 12 minutos mientras que el Grupo #2 tuvo una demora 53 minutos de generando la misma cantidad de elementos.

Conclusiones del Capítulo.

Como se ha podido apreciar en los resultados arrojados por las pruebas realizadas al plug-in GEAP, el mismo no solamente se limita a agilizar el desarrollo, sino que obliga a los desarrolladores a seguir las pautas arquitectónicas definidas por el arquitecto, así como abstrae al desarrollador de pensar en la creación de los tipos de elementos arquitectónicos, y lo concentra netamente en el negocio. Por lo que los objetivos definidos en el presente trabajo se han cumplido satisfactoriamente.

CONCLUSIONES

Como resultado del trabajo se realizó el plug-in GEAP, plug-in de Eclipse para agilizar el proceso de desarrollo, mediante la configuración y generación de elementos arquitectónicos personalizados.

Con la creación de este plug-in se obtienen los siguientes beneficios:

- Garantiza la configuración de elementos arquitectónicos.
- Garantiza la generación de elementos arquitectónicos.
- Todas las funcionalidades son provistas mediante asistentes, vistas y preferencias.
- Posee una ayuda integrada al Eclipse.
- Reduce e iguala los tiempos de creación de elementos arquitectónicos de los desarrolladores.
- Los desarrolladores son obligados a seguir las pautas arquitectónicas establecidas por la arquitectura del proyecto.
- Se reduce la cantidad de errores.
- Concentra a los desarrolladores netamente en el negocio.

Por lo que se puede arribar a la conclusión de que el objetivo de este trabajo y las tareas investigativas han sido cumplidos.

RECOMENDACIONES

- Continuar el desarrollo del plug-in con la creación de Editors que permitan refactorizar elementos que se hayan creado con las plantillas, de forma visual utilizando drag and drop para realizar cambios de referencias.
- Adicionar plantillas que contengan configuraciones en formato XML, HTML, entre otros.
- Adicionar la opción a los Tipos de Elementos para generar los setters y los getters por atributos, de forma opcional.

BIBLIOGRAFÍA

1. www.searchsoftwarequality.techtarget.com. *What is integrated development environment? - a definition from Whatis.com.* [Online] [Cited: 04 12, 2009.] http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci754848,00.html.

2. www.Eclipse.org. *About the Eclipse Foundation.* [Online] [Cited: 04 12, 2009.] http://www.eclipse.org/org/.

3. [book auth.] Dan Rubel Eric Clayberg. *Eclipse: Building Commercial-Quality Plug-ins, Second Edition .* s.l. : Addison Wesley Professional, 2006.

4. **David Gallardo, Ed Burnette and Robert McGovern.** *Eclipse IN ACTION.* s.l. : Addison Wesley, 2003.

5. www.springide.org. *SpringideFeatures - Spring IDE - Trac.* [Online] [Cited: 04 12, 2009.] http://springide.org/project/wiki/SpringideFeatures.

6. www.hibernate.org. *hibernate.org - Hibernate Tools for Eclipse and Ant.* [Online] [Cited: 04 12, 2009.] https://www.hibernate.org/255.html.

7. www.eclipse.org. *Eclipse Java development tools (JDT).* [Online] [Cited: 04 12, 2009.] http://www.eclipse.org/jdt/.

8. www.eclipse.org. *PDE.* [Online] [Cited: 04 12, 2009.] http://www.eclipse.org/pde/.

9. *Unified Modeling Language Specification.* 2001. ISO/IEC 19501..

10. www.visual-paradigm.com. *UML CASE Tools - Free for Learning UML, Cost-Effective for Business Solutions.* [Online] [Cited: 05 18, 2009.] http://www.visual-paradigm.com/product/vpuml/.

11. epf.eclipse.org. *Introduction to OpenUP.* [Online] [Cited: 04 12, 2009.] http://epf.eclipse.org/wikis/openup/index.htm.

12. epf.eclipse.org. *Artifact: Vision.* [Online] [Cited: 04 12, 2009.] http://epf.eclipse.org/wikis/openup/index.htm.

13. epf.eclipse.org. *Artifact: Use-Case Model.* [Online] [Cited: 04 12, 2009.] http://epf.eclipse.org/wikis/openup/index.htm.

14. epf.eclipse.org. *Artifact: Design.* [Online] [Cited: 04 12, 09.] http://epf.eclipse.org/wikis/openup/index.htm.

15. **Erich Gamma, Kent Beck.** *Contributing to Eclipse: Principles, Patterns, and Plug-Ins.* s.l. : Addison Wesley, 2003.

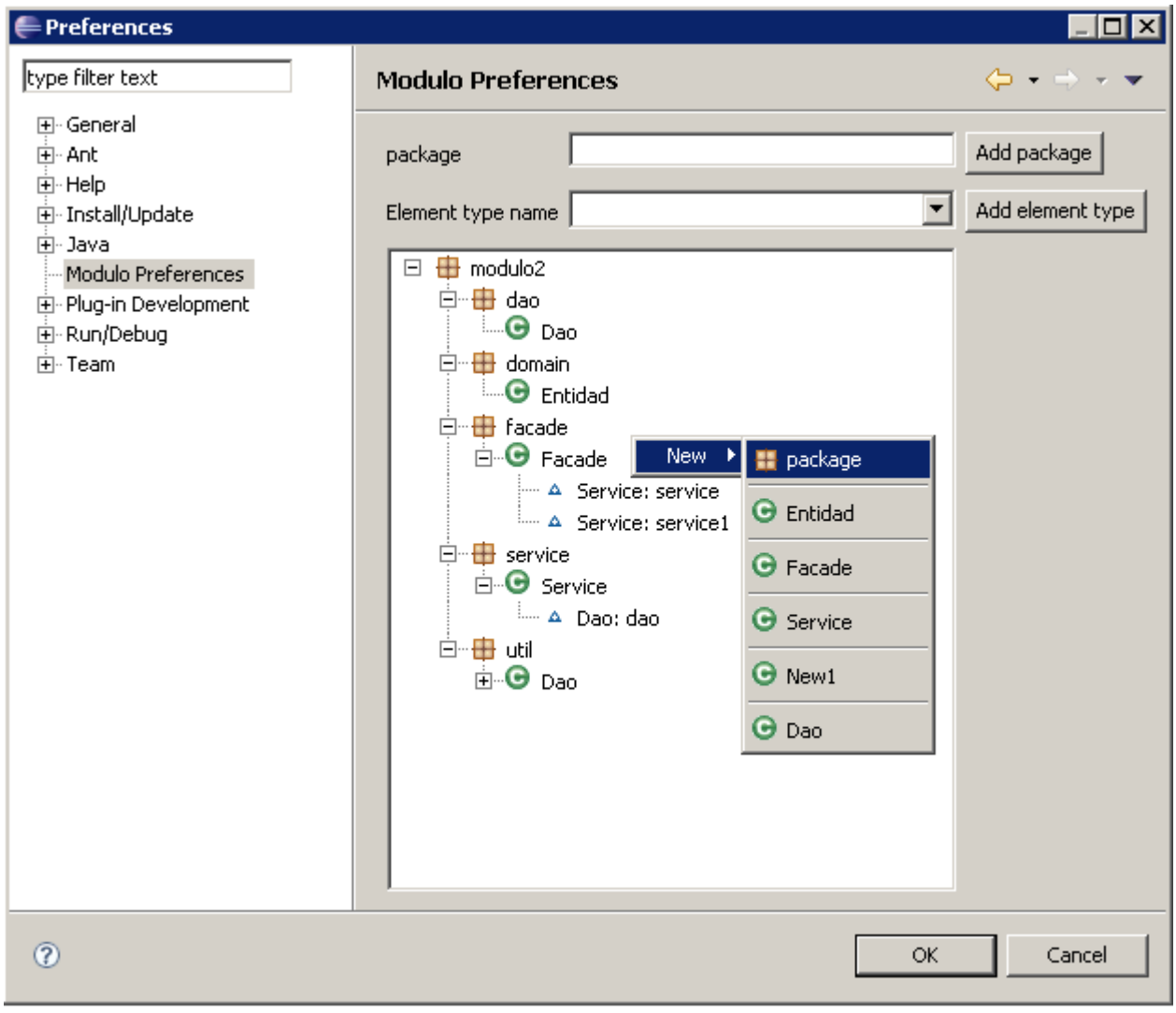
16. **Matthew Scarpino, Stephen Holder,Stanford NG and Laurent Mihalkovic.** *SWT/JFace IN ACTION.* s.l. : Addison Wesley, 2005.

17. gravity.sourceforge.net. *OSGi in a nutshell.* [Online] [Cited: 04 12, 2009.] http://gravity.sourceforge.net/servicebinder/osginutshell.html.

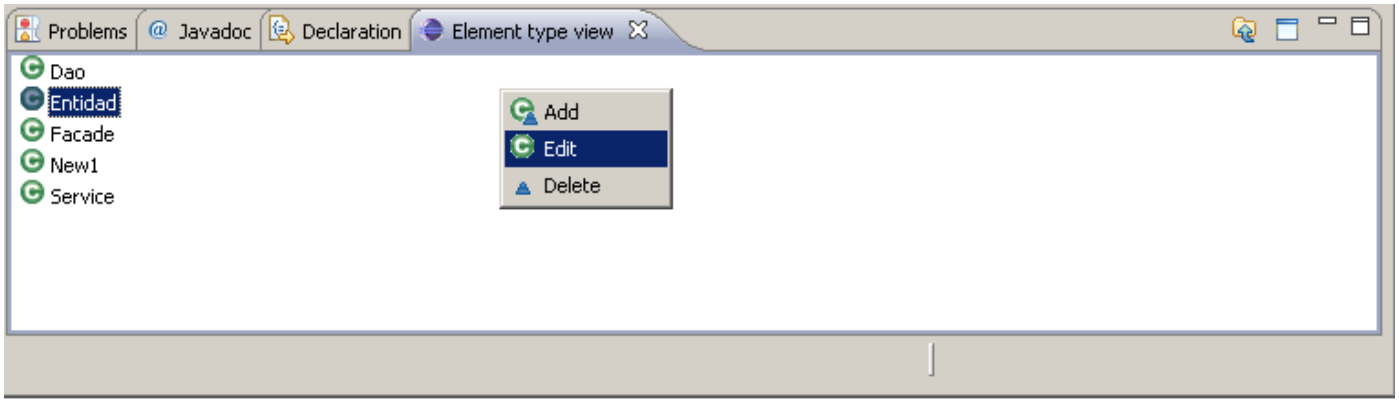
ANEXOS

En esta sección se mostrará las imágenes de las pantallas que corresponden a las funcionalidades del plug-in GEAP planteadas a lo largo de la investigación.

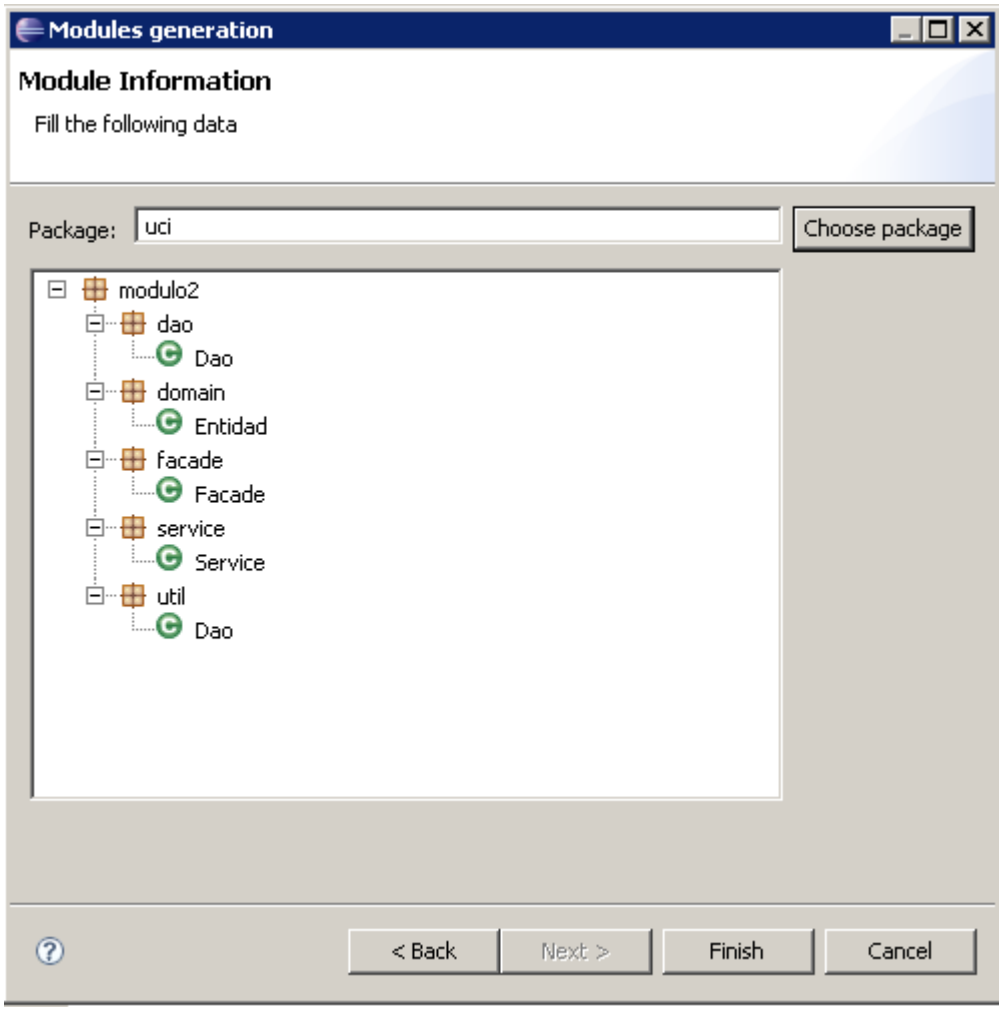
Preference que permite la configuración de un módulo.



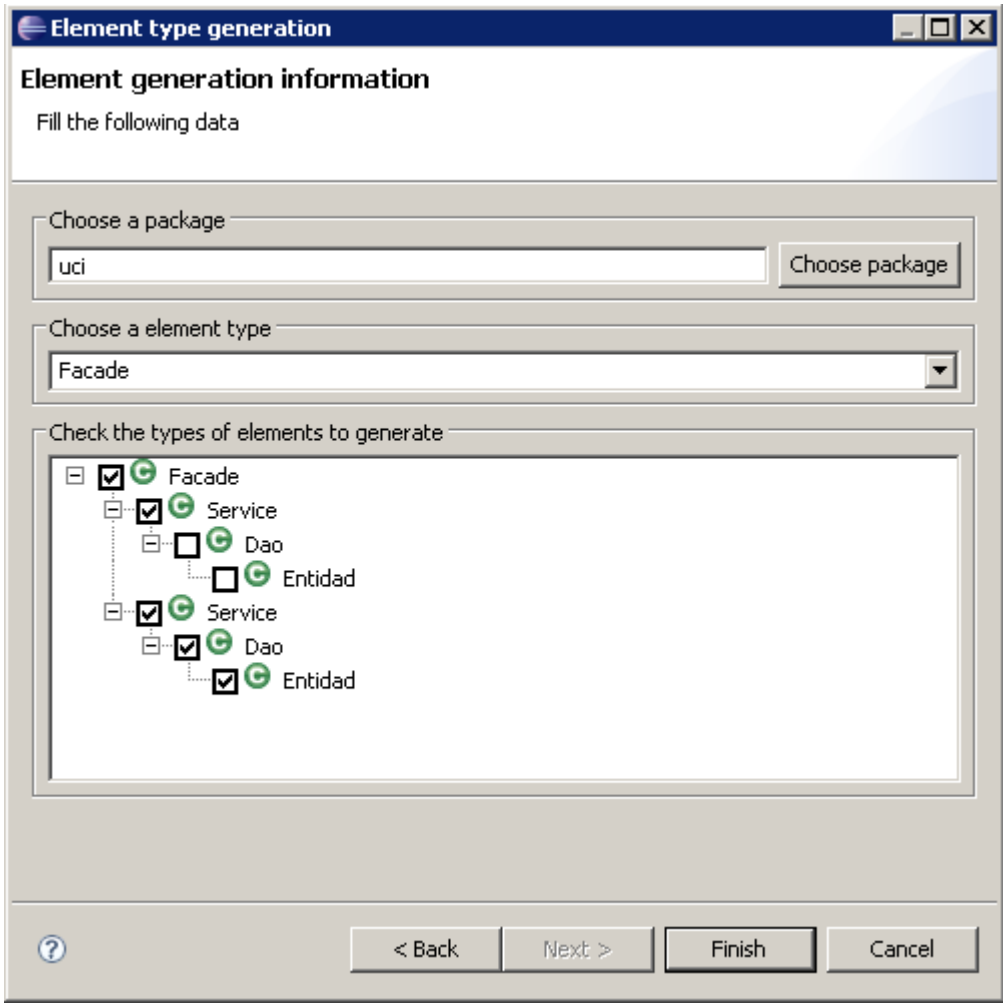
View que muestra todos los Tipos de elementos y permite adicionar, editar y eliminar.



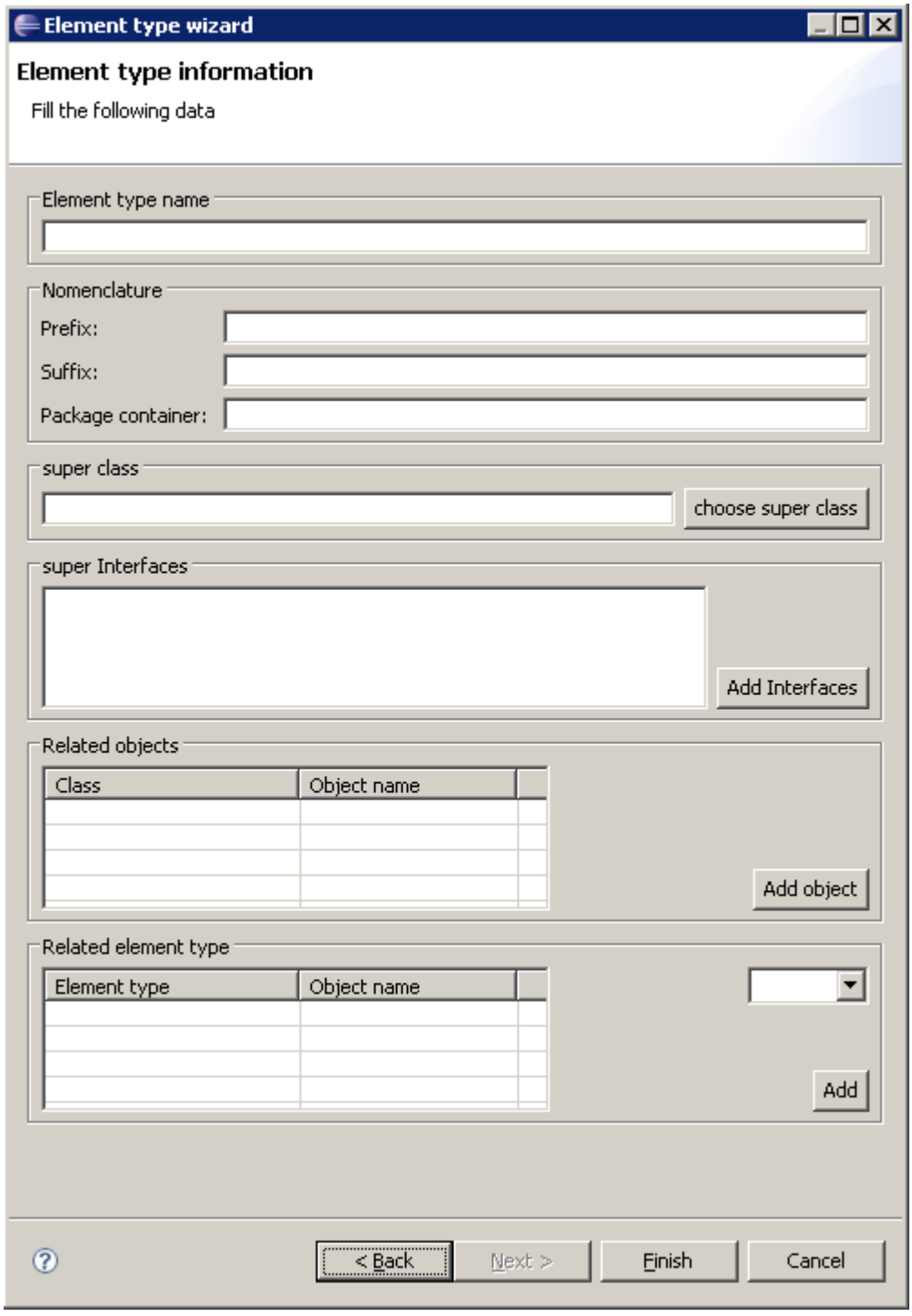
Wizard que permite generar módulos.



Wizard que permite generar Tipos de elementos anidados.



Wizard que permite configurar un Tipo de elemento.



GLOSARIO DE TÉRMINOS

JME: es una especificación de un subconjunto de la plataforma Java orientada a proveer una colección certificada de APIs de desarrollo de software para dispositivos con recursos restringidos. Está orientado a productos de consumo como teléfonos móviles o electrodomésticos.

Web: abreviatura de World Wide Web (WWW); interfaz de comunicación en la Internet, que hace uso de enlaces de hipertexto en el interior de una misma página, o entre distintas páginas.

JEE: La versión empresarial de Java después de J2EE 1.4 es llamada Java EE 5.0; destacando así los cambios significantes de los framework de peso ligero traídos en los estándares empresariales de Java.

Plug-in :(o plug-in -en inglés "enchufar"-, también conocido como addin, add-in, addon o add-on) es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica.

Hibernate Tools: plug-in de Eclipse diseñado para trabajar en proyectos que hagan uso del framework de persistencia Hibernate.

Spring IDE: plug-in de Eclipse diseñado para trabajar en proyectos que hagan uso del framework de persistencia Spring.

IDE: ambiente de desarrollo integrado en la sección 1.4 se desarrolla con mayor amplitud este término.

Eclipse: plataforma de software de código. En el contexto que se utiliza, se ubica como IDE ya que la herramienta se utiliza con el plug-in “Java Development Tooling (JDT)” integrado, que lo convierte en un IDE de java, más adelante en la sección 1.5 se aborda con mayor profundidad el termino.

IBM: empresa que fabrica y comercializa hardware, software y servicios relacionados con la informática.

Eclipse SDK: se denomina también proyecto Eclipse. Es la forma entregable de la plataforma Eclipse, y que al descargarse contiene también además de la Plataforma Eclipse el plug-in JDT y el PDE.

Repositorio: es el lugar en el que se almacenan los datos actualizados e históricos, es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. A veces se le denomina depósito o depot.

OSGi: se refiere a Open Services Gateway Initiative (Iniciativa de enlace de servicios abiertos) , es una corporación independiente, sin ánimo de lucro que trabaja para definir y promover especificaciones

abiertas de software que permitan diseñar plataformas compatibles que puedan proporcionar múltiples servicios.

Naturaleza (nature): el término es utilizado para asociar un proyecto con una funcionalidad, como una herramienta o un proceso. Por ejemplo la naturaleza Java es lo que hace que un proyecto sea un proyecto Java, distinguiéndolo de otros tipos de proyectos.

Constructor incremental (incremental builder): es un término que se utiliza en Eclipse para una herramienta que manipula los recursos de un proyecto de alguna forma. Son siempre usados para aplicar una transformación a un recurso para producir otro diferente o un artefacto de otra índole.

Beans: Son simples clases de Java con sus métodos de acceso.

HQL: lenguaje de consultas definido por Hibernate.

API: es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

classpath: es un argumento que le muestra a la Maquina Virtual de Java donde buscar las clases y paquetes definidos por el usuario en un programa Java.