Universidad de las Ciencias Informáticas

Facultad 3

"Xaphiro: plugin del Eclipse para la creación de interfaces web con Dojo Toolkit"

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

Autor(es): Eduardo Rojas Escobar

Adrian Jardinez Calderón

Tutor: Ing. Leosvel Pérez Espinosa.

La Habana, Cuba

"En la ciencia no existen calzadas reales, y quien aspire a remontar sus luminosas cumbres; ha de estar dispuesto a escalar la montaña por senderos escabrosos".

Karl Marx

Declaramos que somos los únicos autores del trabajo “Xaphiro: plugin del Eclipse para la creación de interfaces web con Dojo Toolkit” y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio. Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

_____ _____

Adrian Jardinez Calderón

Autor

Eduardo Rojas Escobar

Autor

_____

Ing. Leosvel Pérez Espinosa

Tutor

A mi novia Claudia por acompañarme en buenas y malas ocasiones, tratos, decisiones, a su familia por la acogida, a mi familión de los Calderones en especial a Eider, Luis, Mimi y Sandra, de todos he aprendido mucho. A mis amigos de todas las edades, a los de cinco años que valen por cien, a Will mi hermano. A mi compañero de tesis por esta corta e intensa travesía, por ser capitán y a veces marinero. A Todos los que de alguna forma hayan compartido su tiempo conmigo, disculpen por no mencionar nombres, valoro más los recuerdos, infinitas gracias.

Adrian

A todas las personas que de una forma u otra ayudaron a convertirme en lo que soy, a mis padres, a mis profes, a mis amigos. A mi tía Kire por ayudarme tanto en estos 5 años. A las personas que compartieron su vida conmigo. A una persona pequeñita que, aunque llegó al final, me brindó toda su amistad de forma incondicional, me ayudó en todo lo que pudo y gracias a ella pude salir bien en la predefensa, a ti Gleidis. A mi tutor por su apoyo y por toda la ayuda brindada. A mi compañero de tesis, por su dedicación durante todos estos meses. A mis enemigos, por hacer que la vida no siempre fuera tan fácil. A mis compañeros de grupo.

Eduardo

A Marcia, por cada segundo de mi vida, a Cuti por ser mi luz.

Adrian

A toda mi familia, a ellos le debo lo que soy y todo lo que he vivido. Gracias por su apoyo y por brindarme todo cuanto han podido. Los quiero mucho.

Eduardo

RESUMEN

El uso de tecnologías asíncronas basadas en JavaScript ha tomado carácter esencial en el impulso que hoy poseen los sistemas web en Cuba. El proyecto Sistema Automatizado para la Gestión Bancaria (SAGEB) hace uso de Dojo para la creación de interfaces apoyándose en el editor de código JavaScript que posee el Entorno de Desarrollo Integrado Eclipse. Las particularidades de este proceso, carente de herramientas que permitan sacar ventaja del uso de dichas tecnologías web, provocan un lento progreso en esta etapa de desarrollo del producto. Xaphiro, la solución que se presenta en este trabajo, pretende dar respuesta a esta problemática a través de una paleta de componentes de Dojo integrada al Eclipse. Durante la realización del mismo se estudiaron tecnologías existentes que hacen uso de paleta de componentes así como las principales características de la arquitectura de Eclipse, se describen las principales tecnologías, metodologías y patrones de desarrollo utilizados y se muestran los principales artefactos y pruebas generados durante el proceso de creación del producto. El resultado final es un componente de Eclipse tan extensible y multiplataforma como el sistema en sí que permite la fácil creación de elementos web con Dojo, acelerándose, de esta manera, la creación de interfaces en el proyecto SAGEB y permitiendo su uso a todos los usuarios con similares necesidades.

Palabras Claves: Plugin, Ambiente de desarrollo integrado, Widget

INTRODUCCIÓN

Un objetivo fundamental en el desarrollo moderno de aplicaciones Web[1] lo constituye la utilización de interfaces ricas e interactivas haciendo uso de AJAX[2], acrónimo de Asynchronous Javascript and XML (JavaScript[3] asíncrono y XML[4]), debido fundamentalmente a que las tecnologías que este combina (dígase JavaScript, DOM[5], etc.) poseen estándares abiertos y son aplicables a múltiples plataformas de desarrollo, Sistemas Operativos y navegadores web.

Ante el auge y la fortaleza de este tipo de tecnologías los desarrolladores han comenzado a surtirse de numerosos frameworks [6]JavaScript que les proporcionan una variada gama de funcionalidades para facilitar el proceso creativo, los que han comenzado a crecer en potencia y complejidad, convirtiéndose en poderosas y a la vez muy intuitivas herramientas de desarrollo. Entre los más reconocidos por la comunidad internacional podemos mencionar JQuery, Mootools, Dojo y ExtJS.

La Universidad de las Ciencias Informáticas (UCI), en su empeño de informatizar la sociedad cubana, implementa soluciones web en muchos de los proyectos que lleva a cabo, utilizando en las mismas tecnologías libres y de punta, como las antes expuestas, ante el imperativo de la excelencia en los servicios: La modernización del Sistema Bancario Cubano es una de las misiones que comienza a enfrentar la UCI y es el Banco Nacional de Cuba (BNC) una de las entidades pioneras en este proceso.

El Centro de Informatización de Entidades (CEIGE) acordó con el BNC la creación de un nuevo sistema que sustituyera al obsoleto Sistema Automatizado para la Banca Internacional de Comercio (SABIC) que está basado en tecnología MS-DOS y exento de las facilidades que brindan los sistemas informáticos actuales.

---

[1] 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.

[2] AJAX: técnica de desarrollo web para crear aplicaciones interactivas.

[3] JavaScript: lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas, más adelante en la sección 1.7 se aborda con mayor profundidad el término.

[4] XML: Lenguaje de Marcado Extensible (Extensible Markup Language), es un lenguaje de marcado que puede usar para crear sus propias etiquetas.

[5] DOM: Document Object Model, permite a los programadores web acceder y manipular los documentos XHTML como si fueran documentos XML.

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

De este acuerdo surge el Sistema Automatizado para la Gestión Bancaria (SAGEB) que se encuentra actualmente en despliegue de su Fase 1 y en implementación de su Fase 2. La solución que el CEIGE desarrolla para el Banco Nacional, adopta como parte de su arquitectura el uso de Dojo como framework para generar las interfaces que interactúan con el usuario en la Capa de Presentación. Dojo es una caja de herramientas muy completa que provee gran variedad de componentes visuales con múltiples funcionalidades, mediante su uso los desarrolladores ahorran mucho tiempo en el diseño de interfaces complicadas y en el manejo de eventos complejos. El proyecto emplea además el Entorno de Desarrollo Integrado (IDE [7]por sus siglas en inglés) Eclipse [8] gracias al soporte que este brinda a la utilización de frameworks como Spring e Hibernate y al manejo de versiones.

En el proyecto SAGEB el diseño de interfaces web con Dojo se realiza de forma poco sofisticada: los desarrolladores deben escribir el código JavaScript necesario para generar un componente de Dojo haciendo uso del editor de código JavaScript que provee el Eclipse el cual posee características similares a un editor de texto convencional. Los programadores deben poseer, para el cumplimiento cabal de dicha tarea, conocimientos del framework Dojo y del lenguaje JavaScript, además de ejemplos funcionales de los elementos que deben utilizarse.

Las principales afectaciones derivadas de las características de desarrollo antes expuestas se mencionan a continuación:

Capacitación: Los desarrolladores adquieren conocimientos de Dojo y lenguaje JavaScript como parte de Cursos Optativos, de aproximadamente 4 semanas de duración, impartidos por personal capacitado dentro del centro de producción. La capacitación resulta por ende un proceso que consume recursos importantes del proyecto como tiempo, alrededor de 4 semanas, medios, para realizar actividades dentro de los cursos y personal que los imparte.

Lento desarrollo: Para lograr una interfaz web de óptima calidad los programadores deben asociar cada uno de los componentes de la vista, cajas de texto, botones, tablas de datos, con su equivalente de Dojo. Las interfaces del sistema Quarxo poseen decenas de elementos en cada una de sus páginas por lo que esta labor, en cada interfaz, puede tomar varias horas de trabajo debido además a errores lógicos o de sintaxis al escribir el código JavaScript.

[7] IDE: ambiente de desarrollo integrado en la sección 1.1 se desarrolla con mayor amplitud este término.

[8] Eclipse: plataforma de software de código. En el contexto que se utiliza, se ubica como IDE ya que la herramienta se utiliza con el plugin “Java Development Tooling (JDT)” integrado, que lo convierte en un IDE de java, más adelante en la sección 1.2 se aborda con mayor profundidad el término.

Como se puede apreciar las características del proceso de creación de interfaces repercuten con mayor intensidad sobre el tiempo de desarrollo, factor este que, históricamente, ha incidido de forma negativa en la calidad y resultado de los proyectos informáticos.

De todo lo anteriormente expuesto se deriva el siguiente **problema:**

¿Cómo agilizar el desarrollo de interfaces visuales con el framework Dojo en el Eclipse?

Teniendo esto en cuenta, se define como **objeto de estudio** el desarrollo de plugins[9] para el Eclipse.

Quedando enmarcado el **campo de acción** en el desarrollo de plugins para el Eclipse que provean una paleta de componentes.

El **objetivo general** se define como:

Realizar el diseño e implementación de un plugin para Eclipse que provea una paleta de componentes de Dojo.

Para llevar a cabo el objetivo propuesto se llevan a cabo las siguientes **tareas investigativas**:

- Caracterizar la arquitectura orientada a plugins del Eclipse y cómo desarrollar nuevos plugins para esa herramienta de desarrollo.
- Caracterizar los componentes de Dojo más usados en el proyecto para incluirlos en la paleta de componentes.
- Caracterizar los distintos componentes desarrollados para la aplicación Quarxo para incluirlos en la paleta de componentes.
- Realizar el diseño de clases de un plugin para el Eclipse que consista en una paleta de componentes de Dojo.
- Implementar la vista de la Paleta de Componentes del plugin.
- Implementar la vista del Editor Visual del plugin.
- Implementar la vista de Propiedades del plugin.
- Implementar la carga e interpretación de ficheros previamente creados y que sean reconocidos y dibujados en el Editor Visual del plugin.

[9] Plugin: Plugin 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.

- Validar la solución midiendo el tiempo empleado en el desarrollo de interfaces visuales con Dojo usando el plugin y sin usarlo.

El **aporte práctico** de esta investigación es brindar la implementación de una paleta de componentes de Dojo para Eclipse.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se enuncian conceptos y definiciones necesarias para llevar a cabo el desarrollo de un plugin de Eclipse. Se hace un análisis sobre las características y funcionalidades de los plugins más utilizados, tanto en la universidad como en el mundo, que provean al Eclipse de una paleta de componentes, además de las tecnologías a utilizar en la construcción de un plugin.

1.1 Ambiente de desarrollo integrado

Un ambiente de desarrollo integrado o IDE, es un programa compuesto por un conjunto de herramientas que proveen facilidades a los programadores para agilizar el proceso de desarrollo de software. (1)

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación. Consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI[10]), algunas veces contiene también un sistema para llevar a cabo el control de versiones. Muchos de los IDEs modernos llevan integrados además un navegador de clases, un inspector de objetos y diagramas de herencia de clases para ser usados en el desarrollo orientado a objetos. (1)

Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Estos proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación, tales como, C++, C#, Java Object Pascal, Visual Basic, entre otros.

En muchos casos se puede dedicar a un sólo lenguaje de programación, por ejemplo, Borland C++ o puede utilizarse para varios como es el caso de Eclipse.

1.2 Eclipse

En la web oficial de Eclipse, el mismo se define como “una plataforma (IDE), abierta para todo y para nada en particular”. Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados, como el IDE de Java llamado Java Development

[10] GUI: Interfaz Gráfica de Usuario, actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

Toolkit y el compilador que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo). (2)

Como se muestra en este concepto, Eclipse es un producto singular por la versatilidad que ofrece. Pero, ¿qué es lo que le permite a Eclipse tener estas características? Todo esto es posible debido a la poderosa arquitectura basada en plugins con la cual fue diseñado.

1.2.1 Arquitectura de Eclipse

Eclipse es considerado un producto inigualable por su arquitectura madura, bien diseñada y extensible basada en plugins. (3)

Con la excepción de un pequeño núcleo o kernel denominado Platform Runtime, todos los demás componentes que conforman la plataforma Eclipse son plugins (3). Entre ellos se encuentran el Workbench, el Workspace, el Help y el Team como se muestra en la Figura 1.

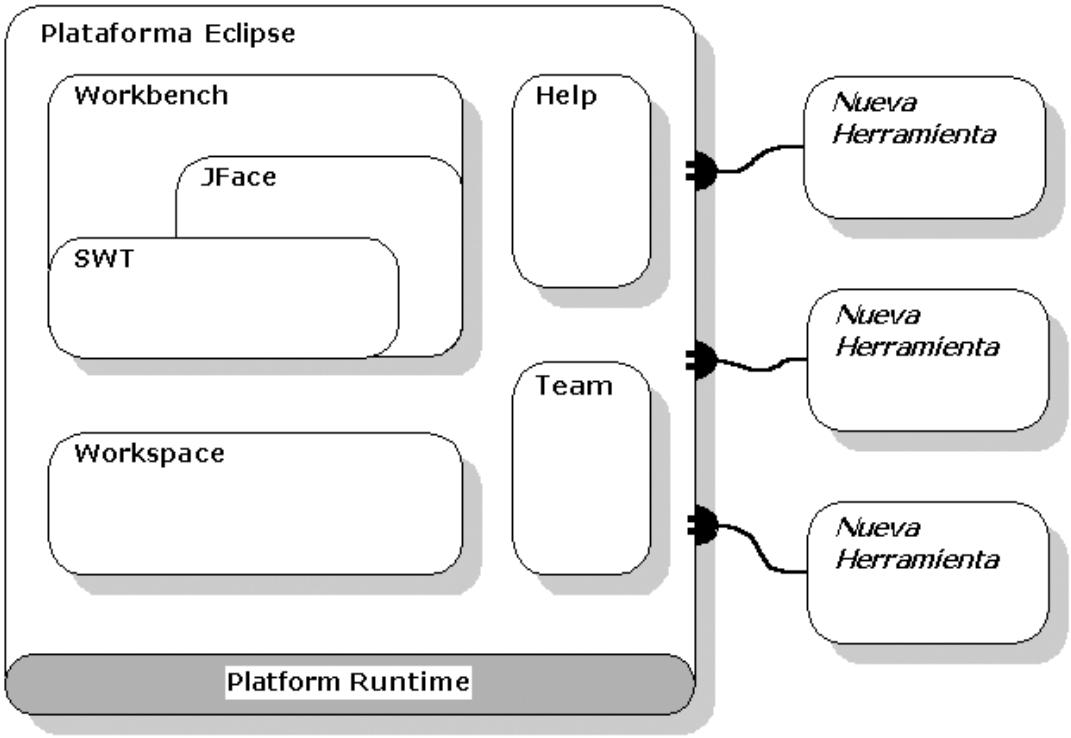

Figura 1: Arquitectura basada en plugins (2).

El Workbench es el componente de la plataforma responsable de la presentación y la coordinación de la interfaz gráfica de usuario (GUI), además de que ofrece la estructura básica de la misma para Eclipse y los puntos de extensión necesarios para extender las funcionalidades de la GUI por otros

plugins. Es el encargado, en fin, de mostrar todos los menús, editores de textos, vistas y perspectivas de Eclipse. (3)

El Workspace se caracteriza por ser el responsable de manejar los recursos de los usuarios, los cuales se organizan en uno o más proyectos a un nivel superior. Cada proyecto corresponde a un subdirectorio del directorio de trabajo de Eclipse y en él se pueden encontrar carpetas y ficheros. También puede proporcionar el código para configurar los recursos del proyecto según sea necesario y como en el caso del Workbench, contiene además los puntos de extensión para que otros plugins interactúen con los recursos de los usuarios. (3)

Los plugins anteriormente mencionados son los componentes esenciales de la plataforma. Pero además de ellos, la misma contiene integrado otros como el plugin Team y el Help. El plugin Team facilita el uso del control de versiones (o gestión de configuración) de los sistemas para administrar los recursos en los proyectos, además de que define el flujo de trabajo necesario para salvar y recuperar datos de un repositorio[11]. El componente Help, no es más que un sistema de documentación, extensible como la misma plataforma Eclipse, al cual mediante el uso de herramientas provistas por la plataforma, se le puede añadir documentación en formato HTML y usando XML puede definir una estructura de navegación. (3)

Además de los plugins ya referidos se pueden encontrar otros que aunque no son parte integral de la plataforma, se encuentran integrados en el Eclipse SDK[12], ellos son el Java Development Tools (JDT) y el Ambiente de Desarrollo de Plugin o en inglés Plugin Development Environment (PDE), que como describe su nombre, es un plugin que facilita el desarrollo de plugins para de esta forma extender las funcionalidades de la misma plataforma (3). En la sección 1.5 se abordan con más profundidad estos componentes. Hasta el momento se ha hablado de plugins y puntos de extensión, pero no se han definido concretamente los conceptos anteriores, a continuación se explicarán los mismos.

1.2.2 Plugins y puntos de extensión

Imagine un gigantesco rompecabezas, al principio algunas de las piezas cuando se comienza a armar el mismo, ya han sido conectadas. Si Ud. se percata, los bordes de las piezas están especialmente

[11] 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.

[12] 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 plugin JDT y el PDE.

conformados para que cada una pueda conectarse a otra perfectamente. Si Eclipse fuera el rompecabezas, los plugins serían las piezas que lo conforman. Un plugin es la unidad más atómica y extensible de Eclipse.

Está escrito puramente en el lenguaje de programación Java y típicamente consiste en código Java empaquetado en un archivo de Java (JAR), con algunos archivos de solo lectura, y otros recursos como imágenes, catálogo de mensajes, entre otros.

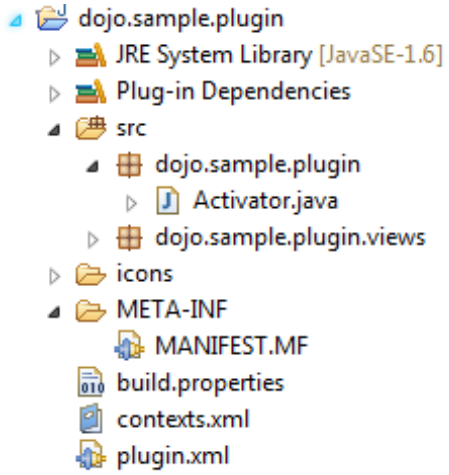

Figura 2: Estructura simple de un plugin de Eclipse.

Al analizar la Figura 1 se percibe que cada pieza tenía bordes especiales que permitían a otras piezas conectarse perfectamente entre sí, estos bordes o conexiones en el mundo de Eclipse se denominan puntos de extensión. Un punto de extensión es el mecanismo por el cual un plugin puede añadir funcionalidades específicas a otro.

Anatomía de un plugin

Cada plugin posee un manifiesto del plugin (en inglés, plugin manifest) en el cual se describe su interconexión con otros plugins, o sea, se declara un número determinado de puntos de extensión y a su vez se exponen las extensiones que se han realizado de otros puntos de extensión definidos por otros plugins. El manifiesto del plugin está representado por un par de ficheros (Ver en la figura 2). El MANIFEST.MF y el plugin.xml (4). A continuación se hace una breve descripción de los mismos:

- MANIFEST.MF: este fichero es generado dentro del directorio META-INF cuando es creado un proyecto plugin. Es un manifiesto del paquete OSGi [13]en donde se describen las dependencias que posee el plugin con otros plugins en tiempo de ejecución y sus atributos como son: el nombre que posee el plugin, el proveedor, la versión del mismo, entre otros. (4)
- plugin.xml: se encuentra en la misma raíz del proyecto y consiste en un archivo de formato XML que describe todas las extensiones realizadas por el plugin y declara además los puntos de extensión que el propio plugin ha definido. (4)

Además de estos elementos en el directorio de un plugin se encuentran otros archivos y carpetas como se muestra a continuación:

- icons: directorio en el cual se encuentran las imágenes o iconos que va a utilizar el plugin. Usualmente son archivos en formato GIF.
- plugin.properties: contiene cadenas de caracteres externalizadas que son referenciadas en el plugin.xml.
- about.html: archivo en formato HTML que es utilizado para mostrar informaciones relacionadas con licencias.

1.3 Librerías de software

Las bibliotecas o librerías de componentes de software son repositorios donde se guardan o conservan documentos, ideas, fragmentos de código y todos aquellos componentes (widgets) que participan en el desarrollo de un software, que puedan ser reutilizados, transformados o consultados. Se puede decir que tienen como objetivo asegurar la disponibilidad de activos, para apoyar el desarrollo de un producto de software. (5)

Mantienen información relevante de cada componente tales como especificación técnica, clasificación, documentación, entre otros. Constituyen además una forma de acelerar los tiempos de programación, implementación, bajar costos y agregar funcionalidades utilizando soluciones ya probadas. (5)

En todos los lenguajes de programación existen librerías de funciones que sirven para hacer diversas cosas a la hora de programar. Las librerías de los lenguajes de programación ahorran la tarea de

[13] OSGi: se refiere a **O**pen **S**ervices **G**ateway **I**nitiative (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.

escribir las funciones comunes que por lo general pueden necesitar los programadores. En ocasiones es más complicado conocer bien todas las librerías que aprender a programar en el lenguaje. (6)

1.4 Widget

Un widget es un componente gráfico con el cual el usuario interactúa; usualmente presentado en archivos o ficheros pequeños, como por ejemplo, una ventana, una barra de tareas o una caja de texto. Los widgets se combinan para construir las interfaces gráficas de usuario. El programador los adapta según sus necesidades sin tener que escribir más código que el necesario para definir los nuevos valores de las propiedades de los widgets. (5)

1.5 Toolkits

Existen herramientas que facilitan la creación de interfaces gráficas de usuario. Estas son conocidas como toolkits. Un Toolkit consiste en una biblioteca de utilidades, las cuales pueden también incorporar componentes de interfaz de usuario (IU) o widgets. (7)

Según Cambiaso (7), el uso de toolkits tiene grandes beneficios, como los que se mencionan a continuación:

- Independencia de la IU: Separa la parte correspondiente al diseño de la IU de las complejidades de la programación. Esta separación le permite a los diseñadores el realizar prototipos rápida y fácilmente, hacer revisiones en minutos, y de esta manera agilizar los procedimientos de prueba. La programación necesaria para el sistema se empieza a escribir una vez que el diseño de la IU se ha estabilizado.
- Código más confiable: El código generado por los toolkits es frecuentemente más confiable que el código creado por un implementador humano, por lo que requiere una menor cantidad de depuraciones[14]. Esto se debe a que el código generado automáticamente tiene un alto nivel de especificación.
- Elaboración rápida de prototipos: Con un toolkit las IU's pueden ser construidas y modificadas rápidamente. Gran parte del trabajo de diseño y construcción de un programa se debe a la creación de la IU. Al utilizar un toolkit para su elaboración, el tiempo de fabricación se reduce de una manera considerable. El tiempo ganado puede ser utilizado para hacer una mayor cantidad de pruebas de evaluación, y tener al final un producto apropiado para el usuario final.

[14] Depuraciones: es el proceso de identificar y corregir errores de programación.

- Interfaces de usuarios más fáciles de crear y de mantener: Esto es porque las especificaciones de la interfaz pueden ser representadas, validadas y evaluadas fácilmente gracias al empleo de los widgets incluidos en el toolkit, la cantidad de código a escribir es mucho menor.
- Facilidad para hacer modificaciones: El empleo de un toolkit permite que la incorporación de cambios a la interfaz después de haber hecho las pruebas correspondientes sea más sencilla.

1.6 Dojo Toolkit

Dojo es un toolkit open source DHTML[15] escrito en JavaScript destinado a facilitar el rápido desarrollo de JavaScript o de las aplicaciones basadas en AJAX y sitios web. Fue iniciado por Alex Russell en el año 2004 y es doble licenciado bajo la licencia BSD [16] y la Licencia Libre Académica. Su idea es la de abstraer al desarrollador de las complejidades del DHTML y de las discrepancias existentes entre navegadores, que hacen que el código JavaScript a utilizar sea diferente. Dojo provee gran cantidad de funcionalidades las cuales divide en tres grandes proyectos, Dojo Core, Dijit y DojoX. Dijit y DojoX están basados en la sólida base que Dojo Core brinda. Dijit incorpora un conjunto de widgets con los cuales se pueden crear amigables interfaces en muy poco tiempo, y escribiendo casi o ningún código JavaScript. DojoX es donde se desarrollan extensiones para Dojo Toolkit, sirve de cuna para nuevas ideas y pruebas de funcionalidades adicionales para el Toolkit principal. (8)

Según la Dojo Foundation (8), las principales funcionalidades que brinda Dojo Toolkit son:

- **Clases de ayuda para la programación orientada a objetos**: Javascript es basado en prototipos, no basado en clases. Dojo esencialmente construye un sistema de clases adicionando grandes funcionalidades como la herencia, encapsulamiento y polimorfismo entre clases.
- **Módulos**: Al tiempo que el código del cliente crece, es difícil de mantener los nombres de variables globales, por eso Dojo brinda un sistema de módulos como parte de su sistema de clases, similares a los paquetes en Java.
- **Métodos de Dijit**: Permite crear componentes de Dijit programáticamente. También permite crear widgets propios y extender las clases existentes en Dijit.

[15] DHTML: Dynamic HTML, designa el conjunto de técnicas que permiten crear sitios web interactivos utilizando una combinación de lenguaje HTML estático, un lenguaje interpretado en el lado del cliente (como JavaScript), el lenguaje de hojas de estilo en cascada (CSS) y la jerarquía de objetos de un DOM.

[16] BSD: Berkeley Software Distribution.

- **Eventos**: Dojo generaliza el sistema de eventos existente en Javascript. Permite conectar eventos en código y la aplicación resultante funciona en Firefox, Internet Explorer y Safari sin modificación alguna.
- **XHR (AJAX)**: Dojo adiciona una buena fachada para los servicios nativos de XmlHttpRequest, con el uso de una sola función se pueden pasar parámetros a una petición vía AJAX y obtener la respuesta en texto plano, en XML o en un objeto Javascript vía JSON[17].
- **Drag and Drop:** Los servicios de Drag and Drop[18] son esenciales para una interacción fácil con el usuario. La capa de Drag and Drop que Dojo ofrece es rápida y funciona en múltiples navegadores.
- **Dojo.data**: El módulo de datos es una capa abstracta que hace el acceso a datos desde la base de datos o un servicio web de una forma consistente. Permite crear tus módulos de datos propios desde diversas fuentes.
- **Dojo.query:** Permite buscar y manipular nodos HTML tan fácil como lo es trabajar con selectores CSS.
- **i18n:** Los servicios de internacionalización hacen todo el código globalmente accesible con fechas entendibles, formatos de números y monedas y recursos traducidos al idioma elegido.
- **Manejo del botón de atrás del navegador:** dojo.back salva la aplicación de usuarios nerviosos tratando de ir una página atrás. Aplicaciones de una sola página pueden perder datos al ir atrás por el navegador, dojo.back cambia el comportamiento del botón de atrás para evitar la pérdida de datos.

1.7 JavaScript

JavaScript es un lenguaje de programación que ha permitido el desarrollo de la Web, ha sido el avance más significativo en el logro de páginas Web dinámicas y exactas en cuanto a posición y presentación de su contenido, es un lenguaje robusto y a la vez ligero, el cual a pesar de ser considerado por muchos como un lenguaje no orientado a objetos permite implementar varias de las características de este paradigma de programación, basado en prototipos, las nuevas clases se generan clonando las clases bases y extendiendo su funcionalidad, cualquier navegador puede interpretar el código JavaScript dentro de las páginas Web, permite la máxima interactividad entre el usuario y la página,

[17] JSON: Javascript Object Notation, es un formato que representa a los objetos de JavaScript.

[18] Drag and Drop: Arrastrar y soltar elementos de interfaz, muy conocido del inglés Drag and Drop

además de la verificación de los datos introducidos por el usuario antes de enviar el formulario al servidor, el manejo de applets[19] y plugins dentro de múltiples marcos de HTML. (9)

Según Eguíluz Pérez (9), este lenguaje de programación tiene algunas limitaciones:

- JavaScript por definición no es un lenguaje orientado a objetos aunque debido a la potencia de esta programación se ha conseguido un funcionamiento similar, intuitivo y potente.
- No se precompila.
- No es obligatorio declarar las variables.
- Verifica las referencias en tiempo de ejecución.
- No tiene protección del código, ya que se descarga en texto claro.
- Es un lenguaje interpretado, o sea, que el sistema lo lee y traduce al mismo tiempo que lo va ejecutando, no confundir con Java.

Los efectos con JavaScript refuerzan la usabilidad de una página web y la comodidad del usuario queda garantizada con las potencialidades del lenguaje sobre cada uno de sus componentes.

1.8 Herramientas existentes

Como evento previo al diseño e implementación del plugin, se realizó una minuciosa búsqueda que tenía por objetivo encontrar, tanto dentro del ámbito nacional como internacional, herramientas que brindasen funcionalidades que pudieran dar solución al objetivo planteado en la investigación. Se analizaron además, otras herramientas que proveyesen al entorno de desarrollo Eclipse de paletas de componentes que permitan la creación de interfaces de cualquier índole para comprobar la efectividad de las mismas.

1.8.1 WaveMaker Studio

WaveMaker Studio, un editor WYSIWYG[20] que se ejecuta en un navegador web y permite el arrastrado y soltado siguiendo el modelo MVC[21]. WaveMaker Studio está basado en TurnoAjax Studio y es desarrollado por Scott Miles y Steve Orvell. (10)

---

[19] Applet: es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web.

[20] WYSIWYG: acrónimo de **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et (en inglés, "lo que ves es lo que obtienes").

[21] MVC: Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Características y Funciones

1. Facilidad de uso, mediante una interfaz con propiedades Drag and Drop, permite insertar todo tipo de elementos con tan solo llevarlos hasta la interfaz del programa.
2. Integra diversos widgets de todo tipo, ya sea de servicios web, o algunos más avanzados como consultas a bases de datos, que permiten integrar diversas funciones en un proyecto con tan solo hacer clic sobre ellos.
3. Permite una rápida visualización del camino recorrido en muy pocos segundos y de forma muy sencilla.
4. Es una aplicación totalmente gratuita. (10)

Desventajas

1. No se integra con el entorno de desarrollo Eclipse.
2. No incluye los componentes desarrollados para el Sistema de Gestión Bancaria Quarxo.
3. No responde a la arquitectura definida para el sistema Quarxo.

1.8.2 Maqetta

Es una herramienta útil de diseño de IU, la cual provee un editor WYSIWYG para HTML5 y está concebido para utilizar cualquier librería JavaScript o tema de estilo CSS todo ejecutándose desde el navegador. Al ser una donación de IBM para la Dojo Foundation el proyecto es de código abierto y entre las primeras librerías a las que brinda soporte es a Dojo. (11)

Esta herramienta está dirigida a fases tempranas del diseño para proyectos de aplicaciones web, en las que el diseñador crea maquetas de interfaces, y persigue como principal objetivo agilizar dicho proceso, reducir los costos totales del proyecto y aumentar la calidad del producto. Aún en versiones de prueba Maqetta promete introducirse rápidamente en el mercado, debido a la flexibilidad ante cambios de tecnología que supone ejecutarse en el navegador y la variedad de librerías JavaScript a las que dará soporte. Maqetta no puede considerarse como solución de la problemática planteada al encontrarse en fase de prueba y ser un proyecto de comunidad que a corto plazo no ofrece posibilidades de resultados concretos. (12)

Aunque el futuro se muestra prometedor la realidad demuestra que en la actualidad no existe un producto, sea aplicación de escritorio, de la web o componente de estos, que provea a los desarrolladores de una paleta de componentes de Dojo que pudiera utilizarse en el entorno de trabajo UCI, específicamente para desarrolladores de la aplicación Quarxo.

1.8.3 Plugins existentes que proveen paletas de componentes

Eclipse por su carácter multipropósito provee plugins para una variada gama de necesidades entre las que se encuentra el diseño visual de páginas web, aplicaciones de escritorio, diagramas de clases, de flujos, de UML[22] y otros. A continuación se hará mención de algunos de los plugins que proveen paleta de componentes.

WindowBuilder: Es un diseñador de interfaz gráfico bidireccional, potente y de fácil uso. Está compuesto por el diseñador de SWT[23], el de Swing[24] y el de GWT[25]. Entre sus principales ventajas se puede mencionar que su carácter bidireccional permite que cambios en el código surtan inmediato efecto en el diseñador gráfico y viceversa, no adiciona librerías al proyecto debido a que usa solo clases de SWT y Swing y despliega un árbol de componentes que permite facilidades en la navegación. (13)

WTP: Web Tools Platform, en español Plataforma de Herramientas Web. Extiende la plataforma Eclipse con herramientas para el desarrollo de aplicaciones web y Java EE[26]. Incluye editores gráficos y de código para varios lenguajes y útiles para apoyar el despliegue, la explotación y las pruebas de una aplicación. (13)

Dentro de las principales ventajas que ofrece una paleta de componentes a la hora de realizar una tarea de cualquier índole se encuentran:

1. Reduce considerablemente el tiempo de desarrollo.
2. Reduce el riesgo de aparición de errores dentro la aplicación.
3. No requiere de desarrolladores con grandes conocimientos en la materia.

Es válido destacar que ninguno de los plugins mencionados anteriormente provee una paleta de componentes para Dojo.

[22] UML: Lenguaje Unificado de Modelado, ver epígrafe 1.11

[23] SWT: Standard Widget Toolkit, una librería de widgets de IU estrechamente ligada al Sistema Operativo.

[24] Swing: Librería gráfica de Sun para Java emulada sobre la Máquina Virtual.

[25] GWT: Google Widget Toolkit. Framework para aplicaciones AJAX que permite generar código JavaScript a partir de código en Java.

[26] Java EE: Java Platform, Enterprise Edition es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

1.9 Tecnologías utilizadas

Como se ha analizado en los epígrafes anteriores Eclipse no solo es una plataforma que permite desarrollar todo tipo de lenguajes (Java, C++, PHP, Python), sino que además está equipada con todas las herramientas necesarias para extender sus funcionalidades. A continuación se abordarán las principales tecnologías utilizadas para lograr la construcción de un plugin de Eclipse, de forma eficiente y con calidad. Es objetivo de este acápite además, definir herramientas que faciliten la administración de los recursos de un proyecto Plugin.

1.9.1 Plugins

1.9.1.1 JDT

El JDT es uno de los principales proyectos de Eclipse y es el que hace que la plataforma Eclipse se transforme en un potente IDE de Java, uno de los más reconocidos del mundo. Es el conjunto de plugin más avanzado y elaborado que posee Eclipse. El JDT asiste a los usuarios en los procesos de escribir, compilar, probar, depurar y editar programas escritos en el lenguaje de programación Java, incluyendo plugins de Eclipse. (13)

Según Clayberg y Dan (13), la plataforma Eclipse combinada con el plugin JDT, provee una enorme cantidad de propiedades y funcionalidades. A continuación solo se mencionan algunas de ellas.

- Editores: ofrece editores para todos los archivos de un proyecto Java, brindando funcionalidades como asistentes de completamiento de código, corrección de problemas a través de su funcionalidad denominada “Quick Fix” y formateo de código fuente, entre otros.
- Vistas: provee un conjunto de vistas para navegar y administrar proyectos Java. Entre ellas se tiene el Explorador de Paquetes (Package Explorer), que es una de las piedras angulares de la perspectiva Java; así como la perspectiva Buscador Java o Java Browsing, que está especializada en asistir a los desarrolladores en la comprensión y navegación de grandes aplicaciones con varios proyectos.
- Configuración de Proyectos: incluye un amplio soporte para configurar el classpath[27], las dependencias, librerías, opciones del compilador y muchas otras características de un proyecto Java.

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

- Búsquedas: permite realizar búsquedas complejas de métodos, variables, ficheros e incluso realizar búsquedas de plugins.
- Asistentes: contiene una gran cantidad de asistentes o wizards que permiten crear diversos elementos Java como proyectos Java, paquetes, clases e interfaces.
- Depuradores: provee un ambiente enriquecido para depurar errores, en donde se puede insertar puntos de detención o breakpoints, tracear la ejecución de un programa, inspeccionar y modificar el valor de una variable e incluso cambiar el código de un método durante la depuración del programa.

El plugin JDT dota a la plataforma Eclipse de un IDE de Java completo, como se ha visto hasta el momento; pero además de esto, JDT provee también un API [28] y varios puntos de extensión. Llegados hasta este punto, después de haber visto un conjunto de las numerosas características que aporta JDT a Eclipse, cabría preguntar ¿para qué se necesita utilizar el API de JDT? Si se desea construir un plugin que entre sus funciones se encuentre la de interactuar con aplicaciones Java o con recursos, entonces el API de JDT es indispensable (13). Posee entre muchas otras capacidades la posibilidad de realizar las siguientes tareas:

- Manipular programáticamente recursos Java posibilitando realizar acciones como: crear proyectos, generar clases, interfaces o incluso detectar errores en el código.
- Levantar programáticamente una aplicación Java desde la plataforma.
- Agregar nuevas funcionalidades y extensiones al IDE de Java. (13)

El JDT está estructurado en tres componentes principales que contienen los paquetes del API:

- JDT Core: define el núcleo de los elementos Java y provee clases útiles para manipular archivos de extensión .class y el modelo de elementos Java.
- JDT UI: contiene la interfaz de usuario que provee el IDE.
- JDT Debug: brinda un conjunto de clases que permiten correr y depurar código Java. (13)

1.9.1.2 PDE

El Plugin Development Enviroment o PDE, no es más que el Ambiente de Desarrollo para Plugin que provee Eclipse y uno de los principales subproyectos del mismo, junto con JDT. Está compuesto por

[28] API: Application Programming Interface, es el conjunto de funciones y procedimientos o métodos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

un conjunto de herramientas que cubren todo el ciclo de vida completo del desarrollo de un plugin. Facilita todo el proceso de crear, desarrollar, probar, depurar, construir y desplegar plugins de Eclipse (13). Algunas de estas herramientas de PDE incluyen:

- Editores del Manifiesto: ofrece editores multipáginas basados en formularios, que administran centralmente todos los ficheros del manifiesto del plugin, o sea, el MANIFEST.MF y/o el plugin.xml.
- Asistentes para la creación de nuevos proyectos: facilitan todo el proceso de creación de un proyecto plugin.
- Asistentes para importar: provee un conjunto de asistentes que simplifican el proceso de importar plugins del sistema de archivos o file system.
- Asistentes para exportar: provee un conjunto de asistentes que construyen, empaquetan y exportan plugins con un simple clic a un formato de despliegue.
- Launchers (Probadores): permiten probar y depurar aplicaciones de Eclipse, así como paquetes OSGi.
- Vistas: PDE provee cuatro vistas y una perspectiva de desarrollo, denominada Plugin Development, que ayudan a los desarrolladores de plugins a inspeccionar los diferentes aspectos de su ambiente de desarrollo.
- Herramientas Combinadas: posee asistentes para externalizar y purificar los ficheros del manifiesto del plugin.
- Herramientas de Conversión: consiste en asistentes para facilitar el proceso de convertir un proyecto simple de Java o un archivo JAR simple, en un proyecto plugin.
- Integración con el JDT: los ficheros del manifiesto del plugin participan en búsquedas Java y en la reestructuración del código. (13)

1.9.1.3 Subclipse

Subclipse es un plugin que agrega soporte para Subversion (SVN) al Entorno de desarrollo Eclipse y que permite realizar entre otras tareas las de sincronización y actualización de los elementos de un proyecto. Provee una perspectiva denominada “SVN Repository Exploring” para trabajar con varios repositorios SVN al mismo tiempo. Dispone de varias vistas, que permiten mostrar al usuario los diferentes repositorios con que se está trabajando, o el historial de los cambios realizados en un recurso determinado. (14)

1.9.2 Control de versiones

Se denomina Control de Versiones, a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas. (14)

1.9.2.1 Subversion

Subversion (SVN) es un sistema de control de versiones libre y de código fuente abierto. Uno de los reemplazos más populares de Concurrent Versions System (CVS), ofreciendo mejoras en muchas de las dificultades que presenta este último. Subversion maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central, esto permite que se recuperen versiones antiguas de los datos o examinar el historial de cambios de los mismos. Subversion es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros ya sea código fuente o de otro tipo. (14)

Es válido destacar que, aunque existen otras herramientas que permiten llevar a cabo el control de versiones de un proyecto, solo se analizó Subversion debido a que:

- Ha demostrado ser una potente herramienta para el control de versiones, hecho que se evidencia en los más de 3 años de explotación del mismo dentro del proyecto Quarxo.
- Se integra fácilmente con el Eclipse a través de plugins.
- Hacer uso del servidor de Subversion existente en el proyecto, y no uno nuevo, representa un ahorro de recursos, tanto para el proyecto como para la Universidad.

1.10 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) (15). Para el presente trabajo se analizaron cuatro de las metodologías existentes en el mundo del software actual: RUP[29], XP,

[29] RUP: Rational Unified Process (Proceso Unificado Racional en español) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

Scrum y OpenUP[30], debido a su relevancia en el medio universitario y su aplicabilidad al problema científico que se trata de resolver.

1.10.1 Metodologías Ágiles

Las metodologías ágiles están especialmente orientadas para entornos variables, proyectos pequeños donde los individuos y las interacciones entre ellos, son más importantes que las herramientas y los procesos empleados y donde se exige reducir drásticamente los tiempos de desarrollo manteniendo una alta calidad. Las metodologías ágiles dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. (15)

La prioridad de este enfoque es satisfacer al cliente mediante tempranas y continuas entregas que le aporte un valor, por lo que es más importante crear un producto software que funcione sin tener que escribir documentación exhaustiva, tributando con una elevada simplificación pero sin renunciar a las prácticas esenciales para asegurar la calidad del producto. La colaboración con el cliente debe prevalecer sobre la negociación de contratos por lo que se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Las metodologías ágiles están revolucionando la manera de producir software y a su vez, generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales. (15)

1.10.2 ¿Por qué usar Metodologías Ágiles?

Las metodologías ágiles están especialmente indicadas en proyectos con requisitos poco definidos o cambiantes. Estas metodologías se aplican bien en equipos pequeños que resuelven problemas concretos. Las metodologías ágiles presentan diversas ventajas, entre las que se pueden destacar:

- Capacidad de respuesta a cambios de requisitos a lo largo del desarrollo.
- Entrega continua y en plazos breves de software funcional.
- Trabajo conjunto entre el cliente y el equipo de desarrollo.
- Importancia de la simplicidad, eliminado el trabajo innecesario.
- Atención continua a la excelencia técnica y al buen diseño.
- Mejora continua de los procesos y el equipo de desarrollo. (15)

[30] XP, Scrum y OpenUP: Constituyen procesos ágiles para el desarrollo de software. En próximos epígrafes se abordarán cada uno de ellos.

Para seleccionar una metodología ágil se analizaron tres de las metodologías ágiles más empleadas dentro de la Universidad de las Ciencias Informáticas.

eXtreme Programing

La programación extrema o eXtreme Programing (XP) es una metodología reciente en el desarrollo de software. La filosofía de XP es satisfacer al completo las necesidades del cliente, por eso lo integra como una parte más del equipo de desarrollo. Fue inicialmente creada para el desarrollo de aplicaciones donde el cliente no sabe muy bien lo que quiere, lo que provoca un cambio constante en los requisitos que debe cumplir la aplicación. XP está diseñada para el desarrollo de aplicaciones que requieran un grupo de programadores pequeño, donde la comunicación sea más factible que en grupos grandes de desarrollo. La comunicación es un punto importante y debe realizarse entre los programadores, los jefes de proyecto y los clientes. (15)

Según Kniberg (16), las características esenciales de esta metodología son las siguientes:

- Comunicación: Los programadores están en constante comunicación con los clientes para satisfacer sus requisitos y responder rápidamente a los cambios de los mismos. Muchos problemas que surgen en los proyectos se deben a que después de concretar los requisitos que debe cumplir el programa no hay una revisión de los mismos, pudiendo dejar olvidados puntos importantes.
- Simplicidad: Codificación y diseños simples y claros. Muchos diseños son tan complicados que cuando se quieren ampliar resulta imposible hacerlo, por lo que se tienen que desechar y partir de cero.
- Realimentación (Feedback): Mediante la realimentación se ofrece al cliente la posibilidad de conseguir un sistema apto a sus necesidades, ya que se le va mostrando el proyecto a tiempo para poder ser cambiado y poder retroceder a una fase anterior para rediseñarlo a su gusto.

Scrum

Scrum fue presentado en 1996 por sus creadores Jeff Sutherland y Ken Schwaber. Es una metodología fácil que apoya su gestión de proyecto en la adaptación continua a las circunstancias en que evoluciona el mismo. Scrum es un modo de desarrollo adaptable a tal punto que según uno de sus creadores, Ken Schwaber, es solo un marco de trabajo. Es orientado a las personas y basado en el principio ágil de desarrollo iterativo e incremental. Es una metodología ágil muy potente ya que posee altos niveles de abstracción, que la vuelven ideal para proyectos complejos, y factible para grupos de trabajo de muchos o pocos integrantes, factores que la diferencian del resto de metodologías ágiles

enfocadas solo en grupos pequeños y en proyectos no muy complejos. Scrum además emplea las buenas prácticas de otras metodologías ágiles como la interacción continua con el cliente y el enfoque continuo en la calidad. (16)

OpenUP

Es una versión más ágil de lo que es el Rational Unified Process (RUP), es un proceso unificado que aplica propuestas iterativas e incrementales dentro del ciclo de vida, tratando de ser manejable en relación con RUP. Plantea que se debe tener un software ya funcional, o lo que es lo mismo, un proyecto ejecutable en poco tiempo, para esto se debe utilizar sólo los procesos que sean necesarios, sin demasiados artefactos y sobre todo que el proyecto debe acoplarse a las necesidades del usuario, pudiendo ser éste modificado, mejorado y extendido. OpenUP tiene dos ventajas importantes, este tipo de método disminuye los riesgos y además puede utilizarse tanto en proyectos pequeños como en proyectos grandes, aunque está concebida para proyectos pequeños. Si se maneja con cuidado y con profesionalismo se puede desarrollar un software de gran calidad, a pesar de que se le diseñe en poco tiempo y con poca documentación. Se utiliza preferentemente en proyectos pequeños, dígase proyectos de 3 a 6 personas. OpenUP contiene las características esenciales de RUP, que incluye el desarrollo iterativo de casos de uso, además de proporcionar un acercamiento a la arquitectura central del sistema. El resultado es un proceso mucho más simple que sigue fielmente los principios de RUP. (17)

Según la Eclipse Foundation (17), OpenUP se organiza en dos dimensiones diferentes: Método y Proceso.

- *Método*: Los roles, las tareas y los artefactos están definidos, sin tener en cuenta cómo son aplicados en el ciclo de vida del proyecto.
- *Proceso*: Es cuando los elementos del proceso son aplicados en el sentido conductual, donde el mismo brinde los roles, las tareas y los artefactos. Pueden crearse ciclos de vida diferentes para proyectos diferentes.

OpenUP propone la generación de 13 artefactos fundamentales, los cuales permiten realizar una detallada descripción, organización y construcción de software por grupos pequeños de desarrollo. Los artefactos a generar (los 13 artefactos no son de uso obligatorio) dependen de las necesidades del equipo de desarrollo, el número de integrantes del mismo y los conocimientos que se tengan sobre el producto a desarrollar. La generación o no de un artefacto no incurre en la modificación de la

metodología, OpenUP los propone y el equipo decide. El ***¡Error! No se encuentra el origen de la referencia.*** muestra cada uno de estos artefactos.

Después de analizar estas tres metodologías se ha decidido utilizar OpenUP.

1.10.3 ¿Por qué OpenUP?

Preserva la esencia de RUP y al igual que él, posee un modelo de desarrollo iterativo e incremental, pero con la diferencia de que al ser un proceso mucho más ligero permite micro incrementos en breves períodos de tiempo, lo que posibilita ir creando liberaciones del producto mientras se desarrolla y efectuar modificaciones a los requerimientos sin altos costos en cuanto a tiempo, precio e implementación. Es apropiado para proyectos pequeños y de bajos recursos, permite detectar errores tempranos a través de un ciclo iterativo. Además, evita la elaboración de documentación, diagramas e iteraciones innecesarias requeridas por RUP y puesto que es una metodología ágil, tiene un enfoque centrado en el cliente con iteraciones muy cortas.

1.12 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. (18)

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[31], OMT[32], OOSE[33]). El UML es la unión de estos lenguajes de modelado y además incluye expresiones adicionales para manejar los problemas de modelado que estos métodos no dirigieron totalmente. (18)

El uso de UML como lenguaje de modelado se fundamenta en que la metodología OpenUP, empleada para guiar el proceso de desarrollo del plugin, propone el uso del mismo para la realización de todos los diagramas que se generen.

[31] 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.

[32] OMT: Object Modeling Technique es una de las metodologías de análisis y diseño orientadas a objetos, más maduras y eficientes que existen en la actualidad

[33] OOSE: Object Oriented Software Engineering Metodología de diseño orientado a objetos para la creación de Software creado por Ivar Jacobson en 1992.

1.13 Herramientas de modelado

Existen varios software para el modelado UML, son las denominadas herramientas CASE[34]. Una herramienta CASE es la aplicación de tecnología informática a las metodologías de desarrollo, su objetivo es acelerar el proceso para el que han sido diseñadas y automatizar o apoyar una o más fases del ciclo de vida del desarrollo de software. Entre las herramientas CASE para el modelado están Visual Paradigm y Rational Rose.

1.13.1 Rational Rose

Rational Rose es la herramienta CASE que comercializan los desarrolladores de UML y que soporta de forma completa la especificación del UML. Esta herramienta propone la utilización de cuatro tipos de modelo para realizar un diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema, uno lógico y otro físico. Permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software. (19)

Rational Rose utiliza un proceso de desarrollo iterativo controlado (controlled iterative process development), donde el desarrollo se lleva a cabo en una secuencia de iteraciones. Cada iteración comienza con una primera aproximación del análisis, diseño e implementación para identificar los riesgos del diseño, los cuales se utilizan para conducir la iteración, primero se identifican los riesgos y después se prueba la aplicación para que éstos se hagan mínimos. (19)

Rational Rose proporciona mecanismos para realizar la denominada Ingeniería Inversa, es decir, a partir del código de un programa, se puede obtener información sobre su diseño. Permite completar una gran parte de las disciplinas (flujos fundamentales) del proceso unificado de Rational (RUP), en concreto:

- Modelado del negocio.
- Captura de requisitos.
- Análisis y diseño.
- Implementación.
- Control de cambios y gestión de configuración. (19)

[34] CASE: Ingeniería de Software asistida por computadora o Computer Aided Software Engineering por su significado en inglés.

1.13.2 Visual Paradigm

Es una herramienta CASE, utilizada en un ambiente de software libre, debido a la posibilidad de ejecutarse sobre cualquier sistema operativo, por lo que se convierte en una herramienta multiplataforma, permite crear diferentes tipos de diagramas en un ambiente totalmente visual. Soporta además el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite graficar todos los tipos de diagramas de clases, generar documentación, generar código desde diagramas y código inverso (20). También proporciona tutoriales, demostraciones interactivas y proyectos UML. Como principales características se encuentran:

Beneficios:

- Navegación intuitiva entre el modelo visual y el código.
- Poderosa herramienta de generación de PDF/HTML a partir de diagramas UML.
- Sincronización entre el código fuente y el modelo en tiempo real o bajo demanda.
- Entorno visual de modelado superior.
- Soporte para toda la notación UML.
- Análisis de textos.
- Sofisticados y automáticos diagramas de capas.
- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de integrarse en los principales IDEs.
- Disponibilidad en múltiples plataformas. (20)

1.13.3 Visual Paradigm como herramienta a emplear

Los beneficios que se consiguen al utilizar Visual Paradigm son varios, por un lado el uso de lenguajes visuales facilita su asimilación y entendimiento por parte del equipo de desarrollo y el tiempo invertido

en la definición de la arquitectura se minimiza. Una ventaja que destaca sobre todas las demás es la notable efectividad y productividad que se consigue en labores de diseño arquitectónico y mantenimiento haciendo uso de esta herramienta.

Visual Paradigm se puede integrar con el IDE Eclipse lo que permite la realización de ingeniería inversa a Java, es multiplataforma por tanto apoya la campaña de migración a software libre y permite el desarrollo sobre sistemas operativos basados en Linux. Las razones anteriormente definidas han favorecido su elección como herramienta de modelado para la realización del diseño del plugin.

Conclusiones del capítulo

La investigación de las principales tecnologías empleadas para la realización de un proyecto de plugin, el estado del arte y las modernas metodologías de desarrollo permitió arribar a las siguientes conclusiones.

- En la actualidad no existe un plugin para Eclipse que permita la creación de interfaces web haciendo uso del framework de JavaScript Dojo, a pesar de ser este último uno de los más empleados en el mundo.
- El estudio de algunas herramientas que emplean paletas de componentes para crear interfaces demuestra la efectividad de las mismas en cuanto a reducción de tiempo de desarrollo y corrección de errores.
- La metodología de desarrollo de software OpenUP provee a los desarrolladores de una guía para el enfrentamiento a tipos de proyectos como el que se muestra avalándose esto al ser la metodología propuesta y a la que da soporte la Fundación Eclipse.
- El conocimiento de la herramienta Visual Paradigm por parte del equipo de desarrollo, su carácter multiplataforma y la capacidad de aplicar ingeniería inversa así como el manejo del ciclo completo de desarrollo la hacen ideal para el presente trabajo y presumiblemente para la UCI en su camino hacia la implantación de sistemas libres en todas las etapas de desarrollo de software.

CAPÍTULO 2: CARACTERÍSTICAS DEL PLUGIN

Introducción

En el presente capítulo se exponen los artefactos necesarios que fueron creados con la guía de la metodología OpenUP. Se especifica qué funcionalidades brindará el plugin para agilizar el proceso de creación de Interfaces Web empleando Dojo y Eclipse, y a su vez fijarlo a pautas arquitectónicas previamente definidas. Se exponen también las características y estructura del plugin.

2.1 Propuesta del Sistema

El plugin tiene como objetivo agilizar el proceso de creación de interfaces web haciendo uso de la herramienta de desarrollo Eclipse IDE y el framework de JavaScript Dojo, siguiendo los estándares definidos en el proyecto para el desarrollo de las mismas. La principal funcionalidad a agilizar es el decorado de componentes HTML (tablas, campos de texto, listas desplegables) empleando código JavaScript.

2.2 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. (17)

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. (17)

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[35] en una manera predecible. El plan de iteración define que 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 microincrementos son aplicados para entregar construcciones estables y cohesivas del sistema que incrementalmente progresan hacia los objetivos de la iteración. (17)

[35] Stakeholder: es aquella persona o entidad que está interesada en la realización de un proyecto o tarea, auspiciando el mismo ya sea mediante su poder de decisión o de financiamiento, o a través de su propio esfuerzo

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. (17)

2.2.1 Artefactos generados

Dentro de los artefactos que plantea generar OpenUP se encuentra el documento visión. Este artefacto contiene la definición de la mirada de los Stakeholders del producto a desarrollar, especificado en términos de las necesidades y características claves de los Stakeholders. Proporciona una visión completa del sistema de software en desarrollo y los soportes del contrato entre los clientes y la organización de desarrollo.

2.2.2 Visión Xaphiro:

Declaración Del Problema.

El problema de	¿Cómo agilizar el desarrollo de interfaces visuales con el framework Dojo en el Eclipse?
Afecta	Desarrolladores de interfaces web en el proyecto SAGEB.
Cuyo impacto es	Provoca una pérdida de productividad por parte de desarrolladores de poca y mediana experiencia.
Una solución exitosa sería	Con la creación de una paleta de componentes de Dojo como plugin para el Eclipse se agilizará el desarrollo de interfaces web por parte de los programadores.

Tabla 1: Visión Xaphiro Declaración del Problema

Declaración de Posicionamiento del Producto.

Para	Desarrolladores de interfaces web a través de Dojo.
Quien	Necesita de una herramienta de generación de código que agilice el proceso de producción y reduzca la aparición de errores.
El nombre del producto	Xaphiro
Eso	Agiliza el desarrollo por parte de programadores y los rige a la arquitectura propia del proyecto.
Al contrario de	WaveMaker y Maqetta.
Nuestro Producto	Se integra por completo al Eclipse, soporta la arquitectura definida en el proyecto, e incluye los nuevos componentes de Dojo generados en el proyecto.

Tabla 2: Visión Xaphiro Posicionamiento del Producto

Ambiente del Usuario

Actualmente el número de proyectos y desarrolladores que emplean Eclipse IDE dentro de la Universidad de las Ciencias Informáticas es bastante elevado, 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 que demora actualmente la creación de interfaces web no es muy grande, generalmente este tiempo no varía con el desarrollo del proyecto; pero si la cantidad de interfaces que se generan en el transcurso del mismo aumentan considerablemente, este tiempo no es despreciable.

Las restricciones de este plugin son:

1. Debe ser instalado en el Eclipse IDE 3.6 o posterior.
2. Debe estar previamente instalado el plugin WTP.

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, como por ejemplo que una aplicación realizada en la plataforma Java puede correr sobre cualquier Sistema Operativo.

Necesidades y Características

Necesidad	Prioridad	Características	Liberación planeada
Agilizar el proceso de creación de interfaces web.	Alta	Se brinda una paleta de componentes de Dojo, y una vista con las propiedades de cada componente.	11/05/2012

Tabla 3: Visión Xaphiro Necesidades y Características

2.2.3 Principales artefactos del Análisis

Para lograr una mejor comprensión del Diseño e Implementación del sistema a desarrollar se muestran a continuación los principales artefactos obtenidos en la fase de Análisis, previa a la realización del Diseño de la solución.

2.2.3.1 Descripción de los requisitos funcionales y no funcionales

Los requisitos para un software son la descripción de los servicios proporcionados por el sistema y sus restricciones. Los requerimientos funcionales son los que describen lo que el sistema debe hacer, y los requisitos no funcionales vienen dados de las características requeridas por el software (requerimientos del producto), de la organización que desarrolla el software (requerimientos organizacionales) o de fuentes externas. (21)

2.2.3.1.1 Requisitos funcionales

En la tabla 4 se muestran los requisitos funcionales identificados.

Requisitos funcionales	
RF1: Crear fichero JSP[36]	**RF2:** Crear componente de Dojo
RF3: Modificar componente de Dojo	**RF4:** Eliminar componente de Dojo
RF5: Modificar código HTML	**RF6:** Modificar código JavaScript
RF7: Configurar Xaphiro	

Tabla 4: Requisitos funcionales

Crear fichero JSP

Consiste en brindar la posibilidad de crear un nuevo fichero JSP teniendo en cuenta la estructura definida dentro del proyecto SAGEB para la creación de los mismos.

Crear componente de Dojo

Consiste en proveer a la aplicación de una Paleta de Componentes que contenga cada uno de los componentes que brinda el framework de JavaScript Dojo así como los desarrollados dentro del proyecto SAGEB. Debe permitir al desarrollador seleccionar un componente y soltarlo en la posición deseada, la aplicación, instantáneamente, debe generar el código HTML y JavaScript perteneciente al componente creado.

Modificar componente de Dojo

Consiste en la creación de una Vista de Propiedades que permita al desarrollador la redefinición de un conjunto de propiedades pertenecientes a los componentes de Dojo que se encuentran dentro del JSP que está siendo decorado.

Eliminar componente de Dojo

[36] JSP: Java Server Pages es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.

Consiste en la eliminación de uno o varios componentes, previamente seleccionados, del JSP que está siendo decorado. Una vez eliminado el componente, el plugin debe ser capaz de actualizar el código HTML y JavaScript asociado al elemento eliminado.

Modificar código HTML

Esta funcionalidad permite la creación de nuevos elementos dentro de un JSP introduciendo de forma manual el código HTML asociado. Una vez creado el nuevo componente, el plugin debe actualizar la información que se muestra en el editor web del plugin.

Modificar código JavaScript

Esta funcionalidad permite la decoración de forma manual de los elementos de un JSP, así como la creación y/o modificación de funciones dentro de un fichero JavaScript. El sistema debe ser capaz de reconocer la adición, modificación o supresión de propiedades a los componentes existentes.

Configurar Xaphiro

Consiste en brindar la posibilidad de ocultar y mostrar las categorías y elementos a la Paleta de Componentes del plugin, definir las etiquetas de Spring necesarias para el funcionamiento del editor web.

2.2.3.1.2 Requerimientos no funcionales

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.

Usabilidad.

- Facilidad de uso y de comprensión

Soporte.

- Fácil de instalar
- Fácil de actualizar

Interfaz de Usuario.

- Legible
- Simple de usar

Software.

- JDK 1.6 o superior
- Eclipse IDE 3.6 o posterior

2.2.3.3 Diagrama de Casos de Uso

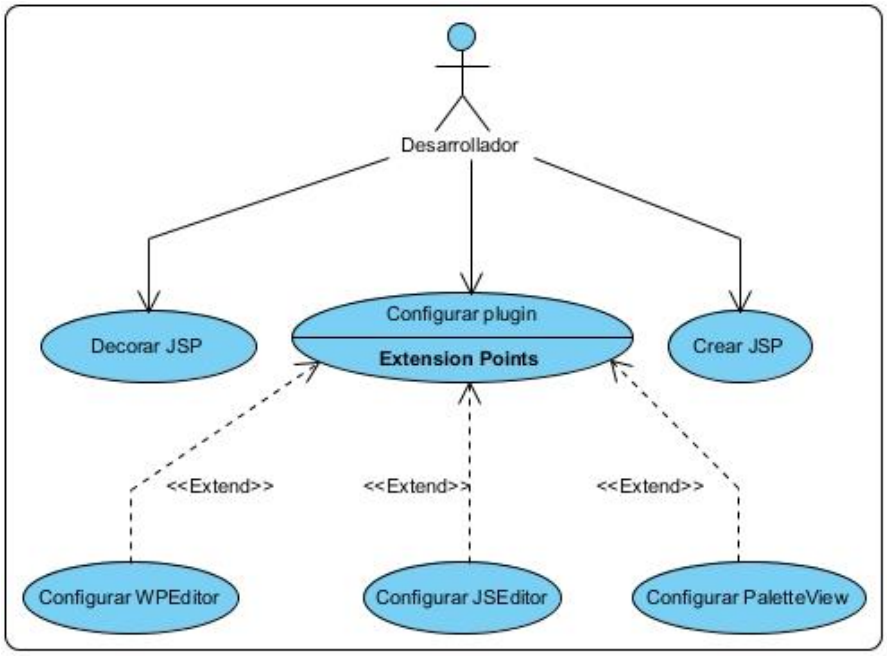

Figura 3: Diagrama de Casos de Uso del sistema

Actores:

Desarrollador: Es el encargado de llevar a cabo el proceso de creación de las Interfaces Web de la aplicación Quarxo.

Casos de Uso:

- *Crear JSP:* Describe cómo el desarrollador crea un nuevo fichero JSP acorde a la estructura definida para este dentro del proyecto SAGEB.
- *Decorar JSP:* Describe cómo el desarrollador lleva a cabo el proceso de decoración de un fichero JSP. Incluye la creación, modificación y eliminación de componentes visuales dentro del fichero en edición.
- *Configurar plugin:* Describe el proceso de configuración de cada uno de los componentes (WebPageEditor, PaletteView, PropertiesView) del plugin.

2.2.4 Diseño del plugin

El diseño que se propone se fundamenta en las características expuestas anteriormente en este capítulo. El plugin se denomina Xaphiro.

En la figura 5 se muestra la estructura básica que posee Xaphiro. El mismo contiene 5 componentes: WebPageEditor, PropertiesView, PaletteView, XaphCore y Help. Los 4 primeros ofrecen las funcionalidades necesarias para decorar las páginas HTML y el componente Help guía al usuario en la utilización del plugin.

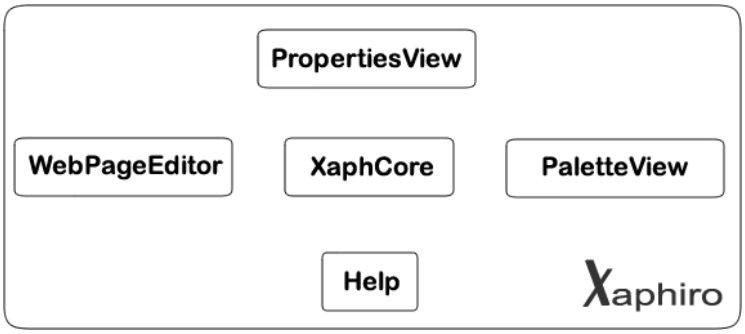

Figura 4: Estructura básica de Xaphiro

El componente **XaphCore** es el componente núcleo del plugin y no hace cambios en la IU del Xaphiro. Es el encargado de la gestión programática del Xaphiro: puntos de extensión usados y generados, clases genéricas, clase Activator, entre otros elementos comunes.

El componente **WebPageEditor** permite que las páginas JSP sean interpretadas por el Xaphiro mostrando además los cambios generados por el decorado con Dojo y por cambios en su estructura HTML. Posee casi las mismas características de la vista web del plugin Web Tools Plataform (WTP) y es el editor central en la perspectiva del Xaphiro.

El componente **PaletteView** genera una paleta con los componentes de Dojo a los que el usuario puede acceder, permitiendo el arrastrado y soltado dentro del WebPageEditor. Hace identificable a cada componente mediante un icono y un nombre.

El componente **PropertiesView** permite la gestión de los componentes de Dojo que ya están en la página JSP. Genera una vista con las propiedades principales de cada elemento en uso permitiendo que estas sean alteradas por el usuario.

El componente **Help** es el elemento que se responsabiliza de ofrecer la documentación necesaria para desarrollar con Xaphiro, muestra mediante un manual de usuario integrado a la ayuda del Eclipse todos los pasos e instrucciones a seguir.

2.2.5 Diagrama de Paquetes

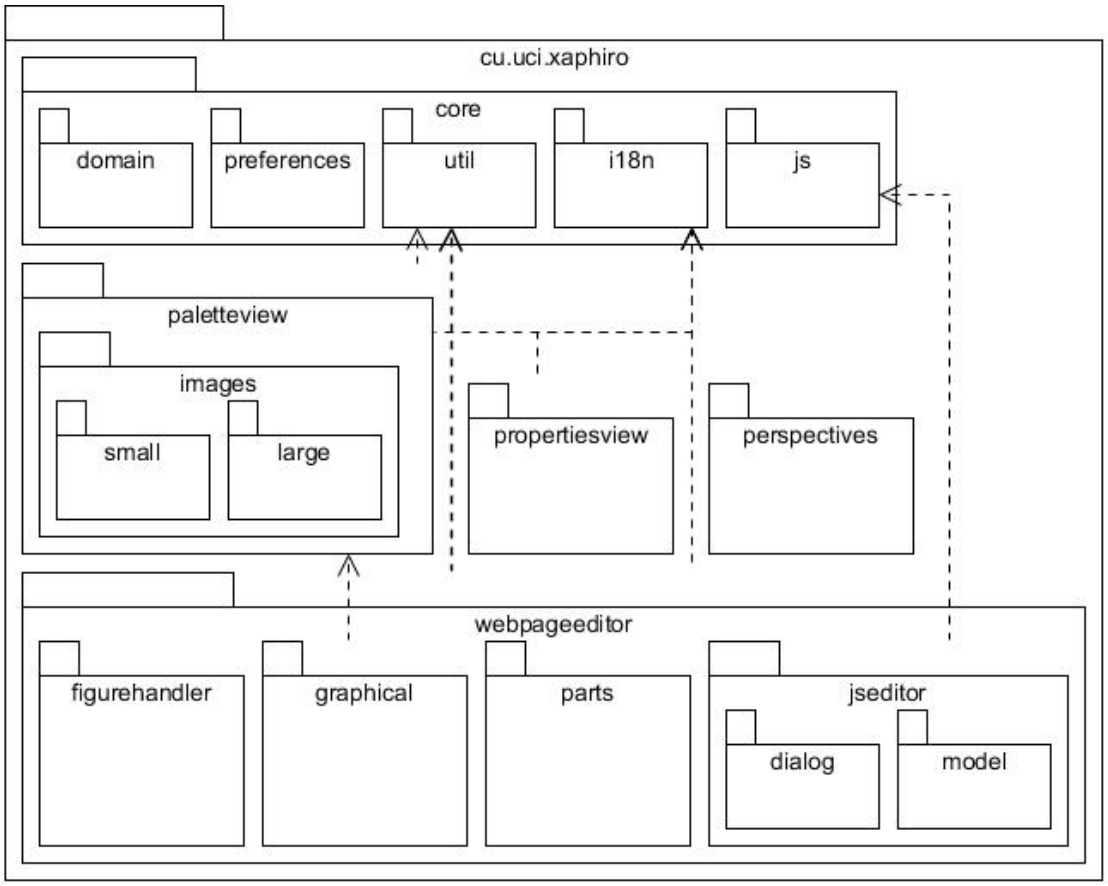

Figura 5: Diagrama de paquetes

El diagrama de paquetes expuesto anteriormente sirve para dar una vista de la organización interna de la estructura del plugin. En el mismo se denotan cuatro paquetes principales: core, webpageeditor, paletteview y propertiesview nomenclatura que no por casualidad es casi idéntica al diseño del plugin previamente mostrado, y es que la estructura de paquetes guarda ciertas dependencias con respecto al diseño general. Es por ello que no se mostrará el propósito de cada paquete sino especificaciones de cada uno de estos en pos de su mejor entendimiento.

El paquete core contiene las clases del componente XaphCore repartidas en cinco paquetes según su propósito:

- El paquete *domain* contiene el equivalente en Java de cada uno de los componentes de Dojo empleados en el proyecto para facilitar su gestión.
- El paquete *js* contiene las clases para la gestión de ficheros JavaScript (.js) y entre sus funcionalidades estarán las de interpretación de código.
- El paquete util es contenedor de funcionalidades de uso más general como carga de ficheros de cualquier formato, manejo de estos ficheros cargados, leer y escribir de los mismos, manejo de excepciones y todo fragmento de código que no cumpla con ninguna clasificación para otros paquetes.
- El paquete *i18n* permite la internacionalización de la aplicación.
- El paquete *preferences* permite la gestión de la configuración del plugin.

El paquete **webpageeditor** es el contenedor de las funcionalidades que permiten generar la vista del editor web para el plugin, su configuración y manejo. Contiene 4 paquetes internos que cumplen funciones específicas dentro del módulo webpageeditor:

- El paquete *figurehandler* contiene las clases encargadas de dibujar cada uno de los elementos dentro del editor web.
- El paquete *graphical* es el encargado de proveer al plugin de un editor web para ficheros JSP.
- El paquete *parts* contiene las clases encargadas de actualizar la información que muestra el editor de ficheros JSP cada vez que este es modificado.
- El paquete *jseditor* es el encargado de proveer al plugin de un editor de ficheros JavaScript.

El paquete **paletteview** es el encargado de generar la paleta de componentes del plugin y el manejo de eventos. Contiene tres paquetes internos, *images, small y large*, que permiten la gestión de las imágenes de cada uno de los elementos de la paleta.

El paquete **propertiesview** es el contenedor de las funcionalidades que permiten la configuración de propiedades, al igual que el componente homónimo, y la validación de las entradas del usuario.

El paquete **perspectives** maneja las perspectivas, componentes más genéricos de la interfaz de usuario del plugin.

2.2.6 Diagrama de Clases

Se aprecian dos colores en todos los diagramas el color azul representa a las clases que son propias del paquete y las de color gris son las clases que pertenecen a otros paquetes pero se relacionan con las mismas.

Diagrama de clases del componente WebPageEditor:

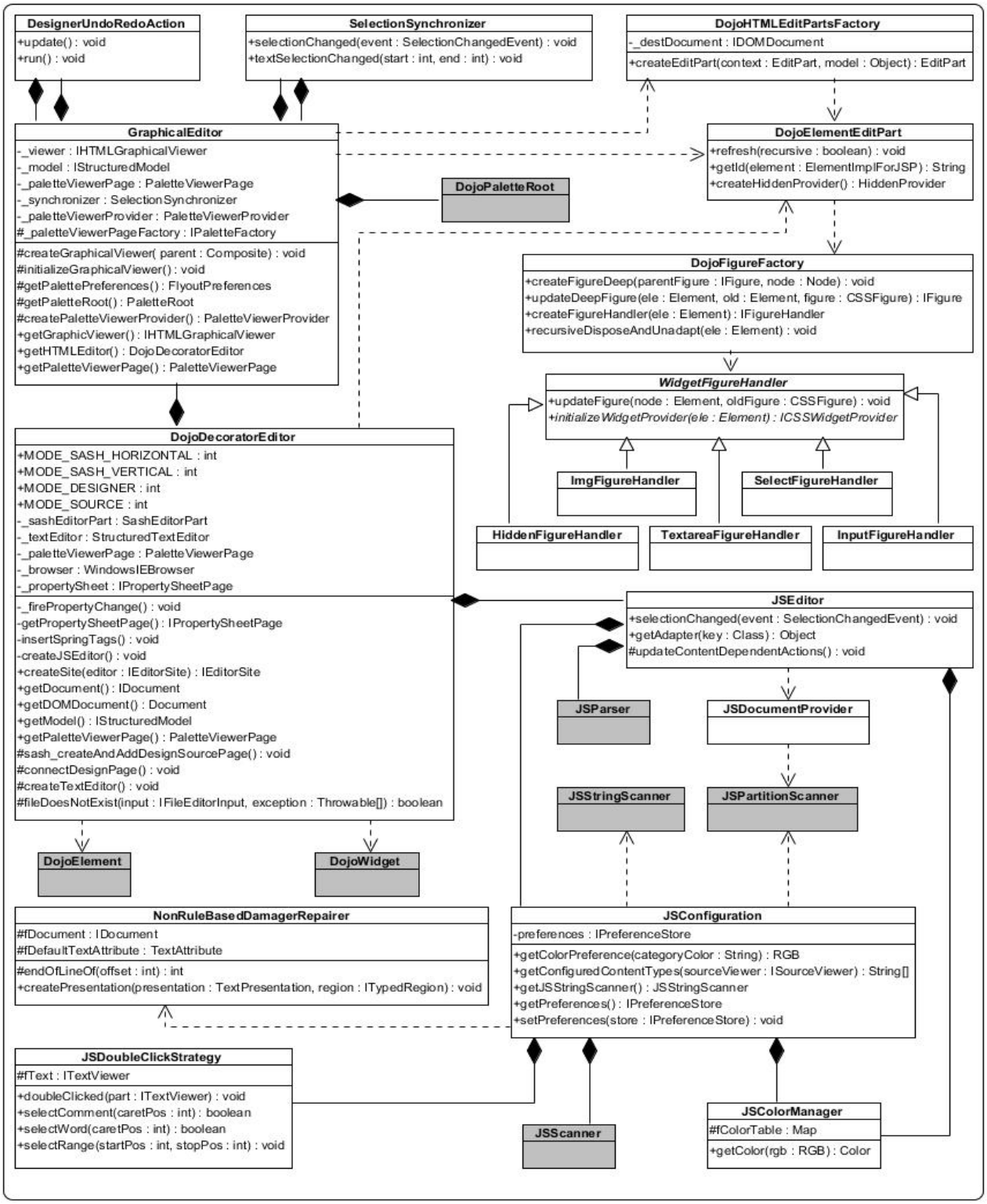

Figura 6: Diagrama de clases del componente WebPageEditor

Nombre: DojoDecoratorEditor	
Descripción: Es la entidad encargada de crear la vista que contiene el editor HTML, el editor de código JavaScript y la paleta de componentes.	
Para cada responsabilidad:	
Nombre	**Descripción**
getPropertySheetPage	Es la funcionalidad encargada de crear la vista Propiedades asociada al editor web.
createJSEditor	Es la funcionalidad encargada de crear el editor JavaScript y cargar el fichero asociado al JSP en edición.
createTextEditor	Es la funcionalidad encargada de crear el editor HTML asociando al editor web del plugin.
getPaletteViewerPage	Es la funcionalidad encargada de crear la vista que contiene la paleta de componentes.
connectDesignPage	Es la funcionalidad encargada de conectar todos los editores del plugin. En caso de ocurrir un cambio en uno, todos los demás actualizarán su información en caso de ser necesario.
insertSpringTags	Es la funcionalidad encargada de insertar dentro de un JSP un conjunto de etiquetas, necesarias para el correcto funcionamiento del mismo.
Nombre: GraphicalEditor	
Descripción: Es la entidad encargada de crear el editor HTML.	
Para cada responsabilidad:	
Nombre	**Descripción**
createPaletteViewerProvider	Es la funcionalidad, unida a *getPaletteRoot,* encargada de generar la paleta de componentes dentro de la vista creada por la clase *DojoDecoratorEditor.*
initializeGraphicalViewer	Es la funcionalidad encargada de dibujar cada uno de los elementos de un JSP dentro del editor web del plugin.

getGraphicViewer	Es la funcionalidad encargada de inicializar el editor gráfico donde se dibujan cada uno de los elementos del JSP.
Nombre: DojoFigureFactory	
Descripción: Es la entidad encargada de dibujar cada uno de los elementos de un JSP dentro del editor web del plugin.	
Para cada responsabilidad:	
Nombre	**Descripción**
createFigureHandler	Es la funcionalidad encargada de crear el objeto FigureHandler asociado a una etiqueta HTML o Spring.
createFigureDeep	Es la funcionalidad encargada de decidir, para cada elemento de un JSP, si este debe dibujar, mostrar como texto o no mostrar.
recursiveDisposeAndUnadapt	Es la funcionalidad encargada de, una vez cerrado el editor web, eliminar cada uno de los elementos previamente creados dentro del editor web.
Nombre: WidgetFigureHandler	
Descripción: Es la entidad encargada de dibujar un elemento específico de un JSP. Cada especificación de esta clase está asociada a un elemento.	
Para cada responsabilidad:	
Nombre	**Descripción**
initializeWidgetProvider	Es la funcionalidad encargada de dibujar un elemento, cada especificación de la clase WidgetFigureHandler implementa esta funcionalidad de manera diferente.
updateFigure	Es la funcionalidad encargada de actualizar el elemento dibujado teniendo en cuenta nuevos cambios introducidos por el desarrollador.

Tabla 5: Descripción de las principales clases y funcionalidades del componente WebPageEditor

Diagrama de clases del componente PaletteView:

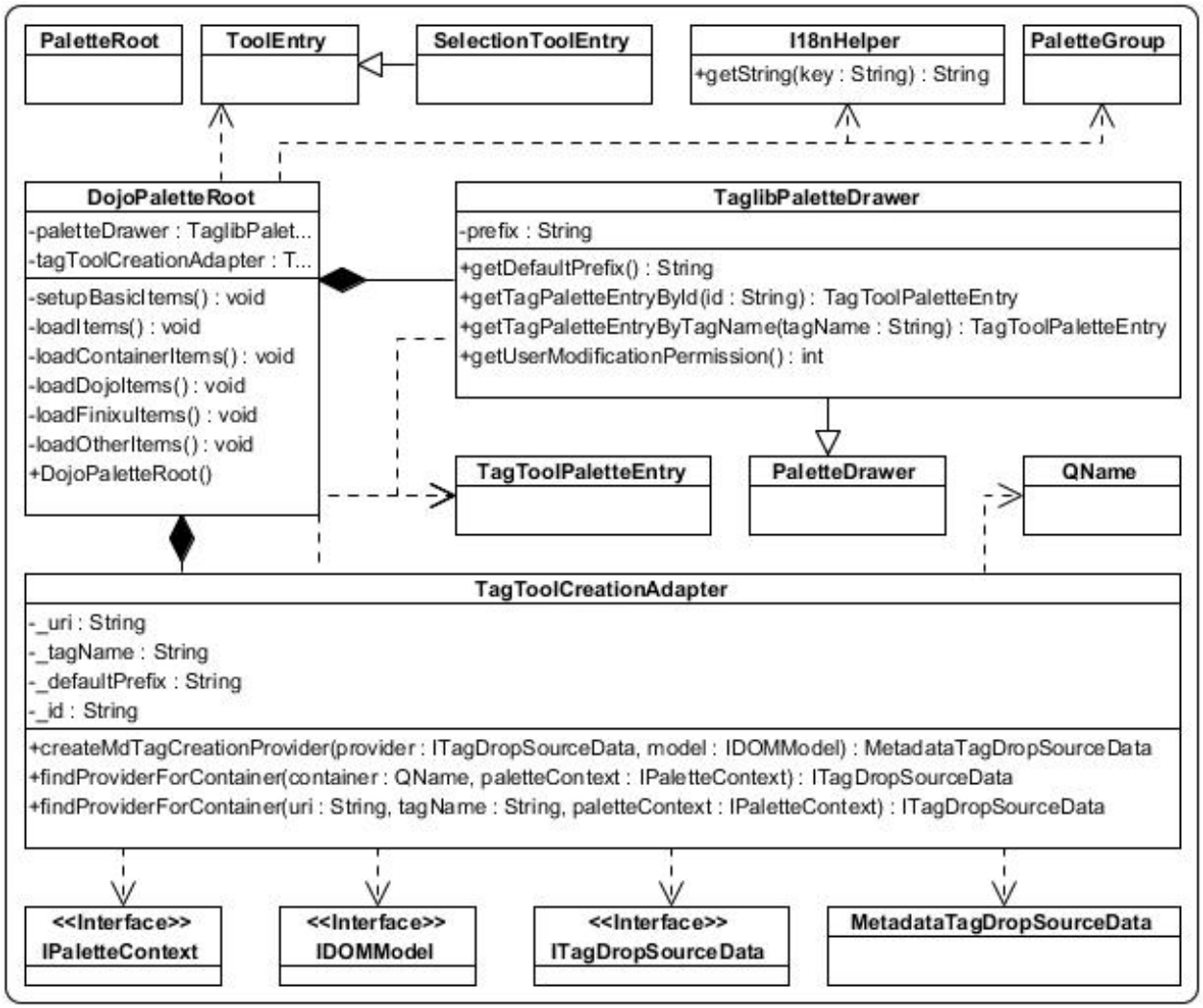

Figura 7: Diagrama de clases del componente PaletteView

Nombre: DojoPaletteRoot	
Descripción: Es la entidad encargada de crear cada una de las categorías y componentes de la paleta.	
Para cada responsabilidad:	
Nombre	**Descripción**
setupBasicItems	Es la funcionalidad encargada de añadir a la paleta herramientas para la selección de componentes dentro de un JSP.

loadItems	Es la funcionalidad encargada de crear cada una de las categorías y componentes de la paleta, haciendo uso de las demás funcionalidades de la clase que la contiene.
Nombre: TagLibPaletteDrawer	
Descripción: Es la entidad empleada para definir las categorías dentro de la paleta de componentes.	
Para cada responsabilidad:	
Nombre	**Descripción**
getTagPaletteEntryById	Es la funcionalidad encargada de devolver un elemento perteneciente a una categoría dado su identificador.
getTagPaletteEntryByTagName	Es la funcionalidad encargada de devolver un elemento perteneciente a una categoría dado el nombre de la etiqueta que genera.
getUserModificationPermission	Es la funcionalidad encargada de definir el nivel de permisos que tiene el desarrollador sobre un elemento de la paleta a la hora de personalizar la misma.
Nombre: TagToolCreationAdapter	
Descripción: Es la entidad encargada de proveer la información necesaria para la creación de una etiqueta al ser arrastrada desde la paleta al editor.	
Para cada responsabilidad:	
Nombre	**Descripción**
createMdTagCreationProvider	Es la funcionalidad encargada de crear la información que describe a los componentes de la paleta (identificador, tipo de etiqueta).
findProviderForContainer	Es la funcionalidad que permite obtener la información necesaria para la generación de una etiqueta dado un parámetro de búsqueda.

Tabla 6: Descripción de las principales clases y funcionalidades del componente PaletteView

Diagrama de clases del componente PropertiesView:

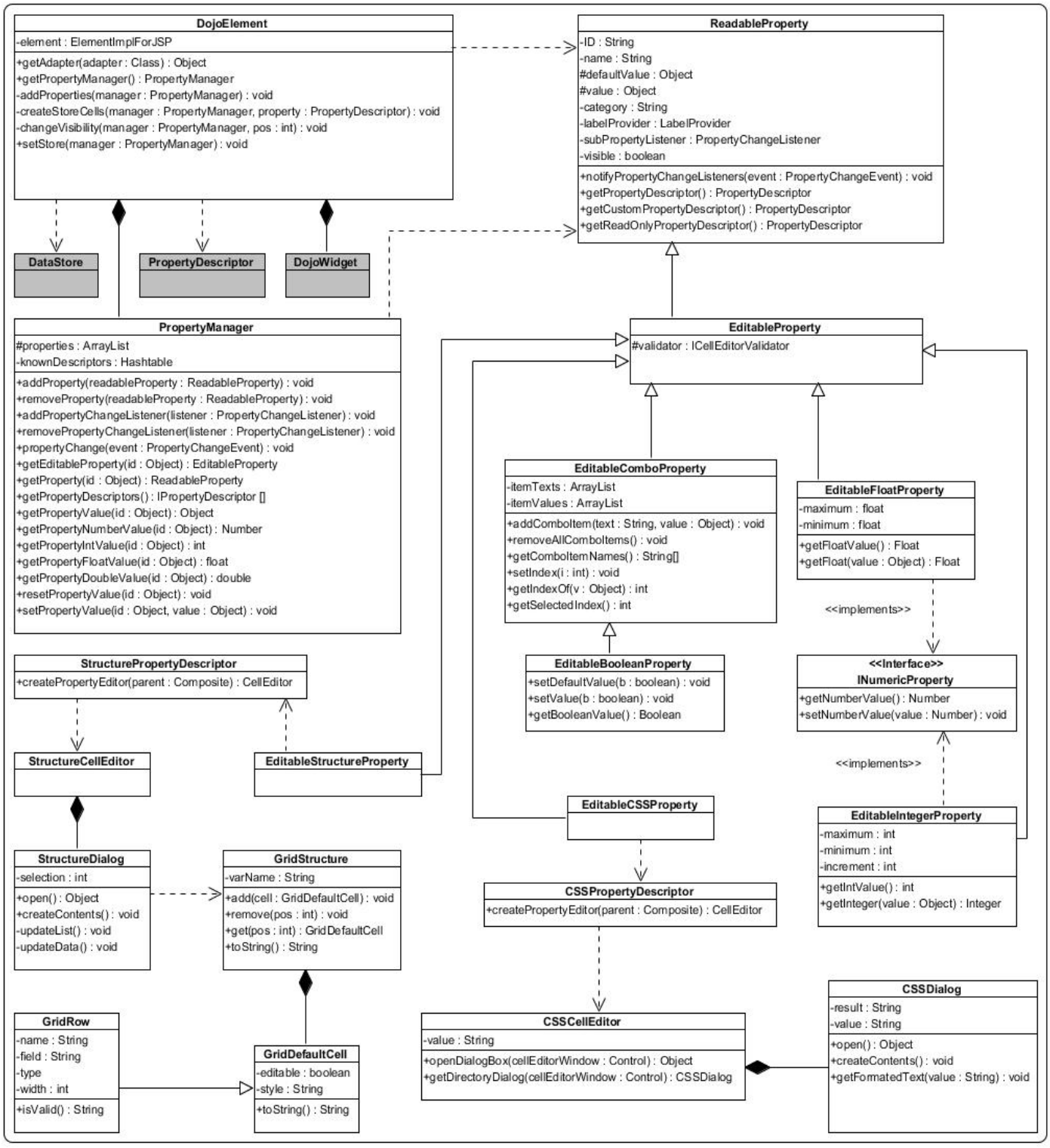

Figura 8: Diagrama de clases del componente PropertiesView

Nombre: DojoElement	
Descripción: Es la entidad encargada de manejar las propiedades asociadas a los elementos contenidos dentro del JSP que está siendo decorado.	
Para cada responsabilidad:	
Nombre	**Descripción**
addProperties	Es la funcionalidad encargada de adicionar las propiedades de un elemento a la vista Propiedades del Eclipse.
createStoreCells	Es la funcionalidad encargada de generar las propiedades asociadas a la propiedad "store" de determinados componentes de Dojo.
setStore	Es la funcionalidad encargada de generar un objeto DataStore a partir de los valores introducidos por el desarrollador.
Nombre: PropertyManager	
Descripción: Es la entidad encargada de facilitar la integración con la vista propiedades. Los objetos de tipo ReadableProperties son adicionados al PropertyManager y este se encarga de manejar la interacción entre la vista Propiedades y las propiedades.	
Para cada responsabilidad:	
Nombre	**Descripción**
addProperty	Es la funcionalidad encargada de adicionar los objetos ReadableProperty al PropertyManager.
getEditableProperty	Es la funcionalidad encargada de devolver un objeto EditableProperty dado su identificador.
getPropertyDescriptors	Es la funcionalidad encargada de generar un arreglo de objetos PropertyDescriptor, el cual será usado por la vista Propiedades.
addPropertyChangeListener	Es la funcionalidad encargada de asignar un evento PropertyChangeListener a un objeto ReadableProperty que se encargará de notificar cuando dicho objeto cambie de valor.

Tabla 7: Descripción de las principales clases y funcionalidades del componente PropertiesView

Diagrama de clases del componente XaphCore:

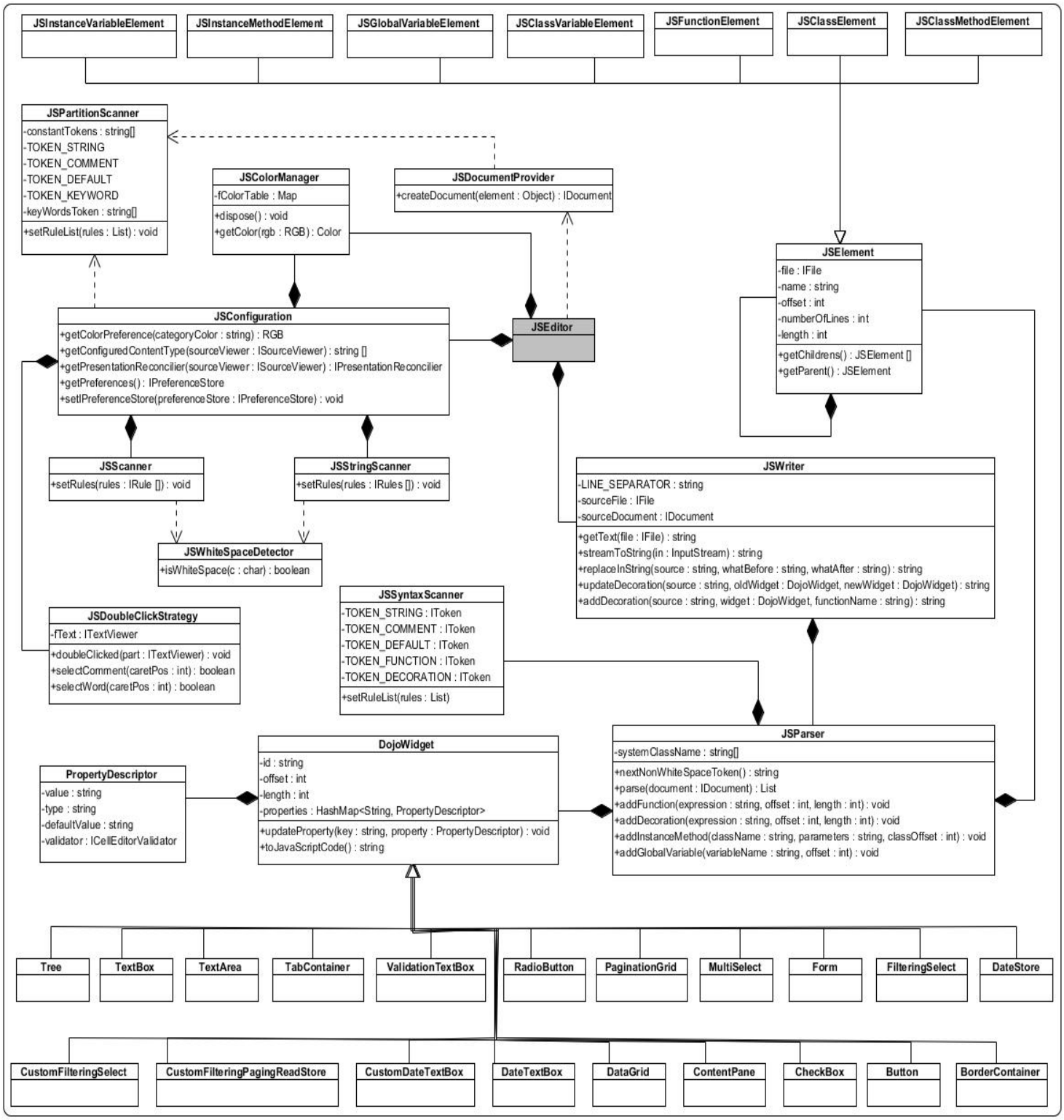

Figura 9: Diagrama de clases del componente XaphCore

Nombre: JSWriter	
Descripción: Es la entidad encargada de gestionar la lectura-escritura en el archivo JavaScript desde los restantes componentes del plugin.	
Para cada responsabilidad:	
Nombre	**Descripción**
getText, streamToString, replaceInString.	Métodos para la gestión de ficheros y String útiles para la recogida y transformación de datos desde el archivo.
updateDecoration	Modifica el contenido de un DojoWidget a nivel de fichero JavaScript.
addDecoration	Adiciona un nuevo DojoWidget al final de una función dentro del fichero.
Nombre: JSParser	
Descripción: Entidad encargada del análisis gramatical del archivo JavaScript según las necesidades del plugin.	
Para cada responsabilidad:	
Nombre	**Descripción**
nextNonWhiteSpaceToken	Retorna el siguiente token que represente un símbolo legible del archivo.
parse	Analiza todo el fichero JavaScript guardando los tokens reconocidos útiles para el plugin.
addFunction	Analiza un segmento de fichero que posiblemente representa a una función, generando el elemento y recogiéndolo para su posterior uso.
addDecoration	Ídem que addFunction.
addGlobalVariable	Ídem que addFunction.
addInstanceMethod	Ídem que addFunction.
Nombre: DojoWidget	
Descripción: Es la entidad que representa al componente de Dojo como objeto del dominio.	
Para cada responsabilidad:	

Nombre	Descripción
updateProperty	Actualiza una propiedad dentro del widget al valor dado.
toJavaScriptCode	Genera el String equivalente al componente, que puede insertarse en el archivo JavaScript.
Nombre: JSElement	
Descripción: Es la entidad que representa los componentes del archivo JavaScript como funciones, variables y métodos.	
Para cada responsabilidad:	
Nombre	**Descripción**
getChildrens	Devuelve los elementos contenidos dentro del componente JavaScript.
getParent	Devuelve el elemento que engloba al componente JavaScript.

Tabla 8: Descripción de las principales clases y funcionalidades del componente XaphCore

2.2.7 Patrones de Diseño

Los **Patrones de Diseño** son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. (22)

Según Metsker (23), un patrón de diseño es:

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

Un desarrollador debe buscar un equilibrio entre las ventajas y las desventajas a la hora de decidir que patrón utilizar.

A continuación se muestran algunos ejemplos donde la solución dada a un problema en específico puede convertirse en un patrón que puede ser empleado para evitar la ocurrencia de estos errores

durante el desarrollo de aplicaciones. Estos patrones son independientes del lenguaje de programación que se emplea lo que aumenta las posibilidades de uso en la vida diaria.

Encapsulación (ocultación de datos)

Problema: los campos externos pueden ser manipulados directamente a partir del código externo, lo que conduce a violaciones del invariante de representación o a dependencias indeseables que impiden modificaciones en la implementación.

Solución: esconda algunos componentes, permitiendo sólo accesos estilizados al objeto.

Subclase (herencia)

Problema: abstracciones similares poseen miembros similares (campos y métodos). Esta repetición es tediosa, propensa a errores y un quebradero de cabeza durante el mantenimiento.

Solución: herede miembros por defecto de una superclase, seleccione la implementación correcta a través de resoluciones sobre qué implementación debe ser ejecutada.

Iteración

Problema: los clientes que desean acceder a todos los miembros de una colección deben realizar un transversal especializado para cada estructura de datos, lo que introduce dependencias indeseables que impiden la ampliación del código a otras colecciones.

Solución: las implementaciones, realizadas con conocimiento de la representación, realizan transversales y registran el proceso de iteración. El cliente recibe los resultados a través de una interfaz estándar.

Excepciones

Problema: los problemas que ocurren en una parte del código normalmente han de ser manipulados en otro lugar. El código no debe desordenarse con rutinas de manipulación de error, ni con valores de retorno para identificación de errores.

Solución: introducir estructuras de lenguaje para arrojar e interceptar excepciones.

2.2.7.1 Patrones creacionales

Singleton (instancia única): Está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. (23)

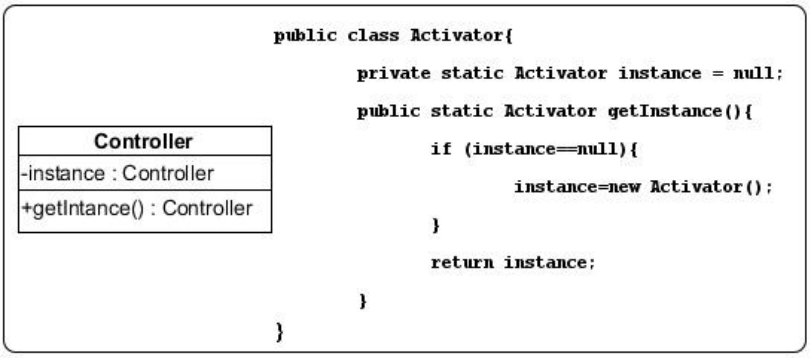

Figura 10: Ejemplo de aplicación del patrón Singleton

Factory Method: Consiste en utilizar una clase constructora abstracta con unos cuantos métodos definidos y otro(s) abstracto(s): el dedicado a la construcción de objetos de un subtipo de un tipo determinado. (23)

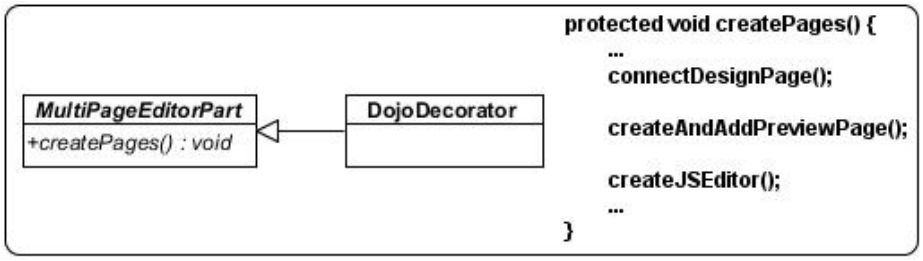

Figura 11: Ejemplo de aplicación del patrón Factory Method

2.2.7.2 Patrones estructurales

Adapter (Adaptador): es un patrón de diseño que se utiliza para transformar una interfaz en otra, de tal modo que una clase que no pudiera utilizar la primera, haga uso de ella a través de la segunda. (23)

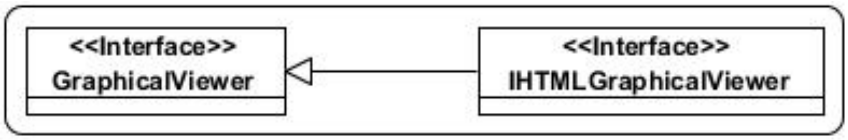

Figura 12: Ejemplo de aplicación del patrón Adapter

2.2.7.3 Patrones de comportamiento

Interpreter (Intérprete): Es un patrón de diseño que, dado un lenguaje, define una representación para su gramática junto con un intérprete del lenguaje. Se usa para definir un lenguaje para representar expresiones regulares que representen cadenas a buscar dentro de otras cadenas. (23)

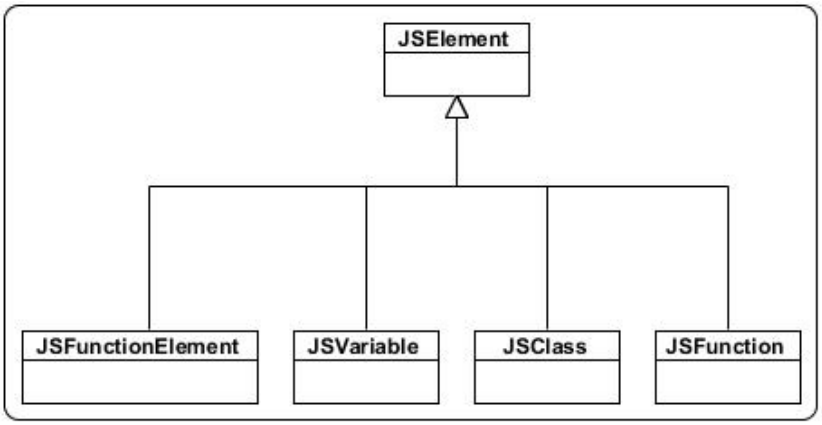

Figura 13: Ejemplo de aplicación del patrón Interpreter

2.2.8 Métricas de validación del diseño

El IEEE Standard Glossary of Software Engineering Terms define métrica como una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado. (24)

Las métricas del software permiten medir de forma cuantitativa la calidad de los atributos internos del producto, esto permite al ingeniero evaluar la calidad durante el desarrollo del sistema. Son varios los puntos de vista relacionados con la calidad del software. Las métricas de diseño a nivel de componentes se concentran en las características internas de los componentes del software con medidas que pueden ayudar al desarrollador a juzgar la calidad de un diseño a nivel de componente. (24)

Un aspecto importante a tener en cuenta en la evaluación del diseño, es la creación de métricas básicas inspiradas en el estudio de la calidad del diseño orientado a objetos. Los atributos de calidad que se tienen en cuenta son: (24)

- **Responsabilidad:** Es la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- **Complejidad de implementación:** Es la complejidad de implementación que posee una estructura de diseño de clases.
- **Reutilización:** Es el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento:** Es el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.

- **Complejidad del mantenimiento:** Es el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
- **Cantidad de pruebas:** Es el número o el grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (componente, módulo, clase, conjunto de clases, etc.) diseñado.

Las métricas seleccionadas como instrumento para evaluar la calidad del diseño del plugin son las siguientes:

Tamaño Operacional de Clase (TOC): Está dado por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad:

Atributo de calidad	Modo en que lo afecta
Responsabilidad	Aumento del TOC provoca aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Aumento del TOC provoca aumento de la complejidad de implementación de la clase.
Reutilización	Aumento del TOC provoca disminución del grado de reutilización de la clase.

Tabla 9: Atributos de calidad evaluados por la métrica TOC.

Se definieron los siguientes criterios y categorías de evaluación para los atributos de calidad anteriores:

Atributo	Categoría	Criterio
Responsabilidad	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Complejidad de implementación	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Reutilización	Baja	>2*Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	<=Promedio

Tabla 10: Criterios de evaluación para la métrica TOC.

Relaciones entre Clases (RC): Está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad:

Atributo de calidad	Modo en que lo afecta
Acoplamiento	Aumento del RC provoca aumento del Acoplamiento de la clase.
Complejidad de mantenimiento	Aumento del RC provoca aumento de la complejidad del mantenimiento de la clase.
Reutilización	Aumento del RC provoca disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Aumento del RC provoca aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 11: Atributos de calidad evaluados por la métrica RC.

Se definieron los siguientes criterios y categorías de evaluación para los atributos de calidad anteriores:

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
Complejidad de mantenimiento	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Reutilización	Baja	>2*Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	<=Promedio
Cantidad de pruebas	Baja	<=Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio

Tabla 12: Criterios de evaluación para la métrica RC.

2.2.8.1 Resultados obtenidos de la aplicación de la métrica TOC

Luego de aplicarse la métrica de diseño TOC se obtuvieron resultados que permiten evaluar el diseño propuesto de calidad aceptable teniendo en cuenta que el 70% de las clases empleadas en el sistema poseen 6 operaciones o menos lo que conlleva a evaluaciones positivas de los atributos de calidad involucrados (responsabilidad, complejidad de implementación y reutilización).

Responsabilidad y Complejidad de implementación:

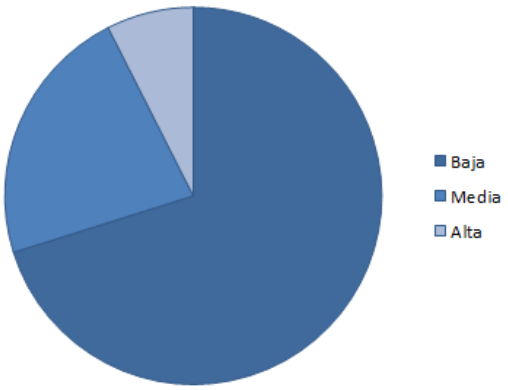

Figura 14: Resultados de la métrica TOC para los atributos Responsabilidad y Complejidad.

Reutilización:

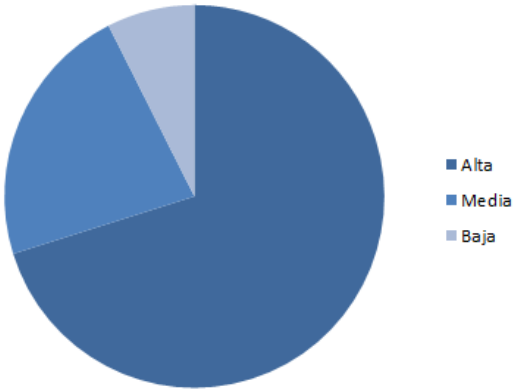

Figura 15: Resultados de la métrica TOC para el atributo Reutilización.

2.2.8.2 Resultados obtenidos de la aplicación de la métrica RC

Luego de aplicarse la métrica de diseño RC se obtuvieron resultados que permiten evaluar el diseño propuesto de calidad aceptable teniendo en cuenta que más del 80% de las clases empleadas en el sistema poseen 1 o menos dependencias con otras clases lo que conlleva a evaluaciones positivas de

los atributos de calidad involucrados (acoplamiento, complejidad de mantenimiento, cantidad de pruebas y reutilización).

Acoplamiento:

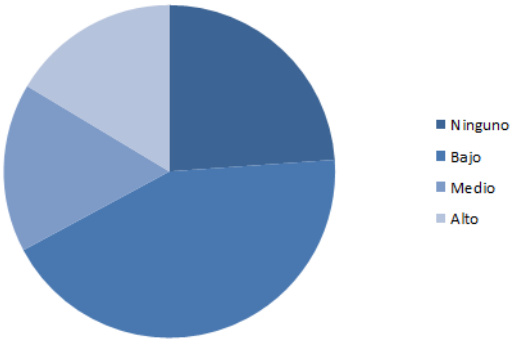

Figura 16: Resultados de la métrica RC para el atributo Acoplamiento.

Complejidad de mantenimiento y Cantidad de pruebas:

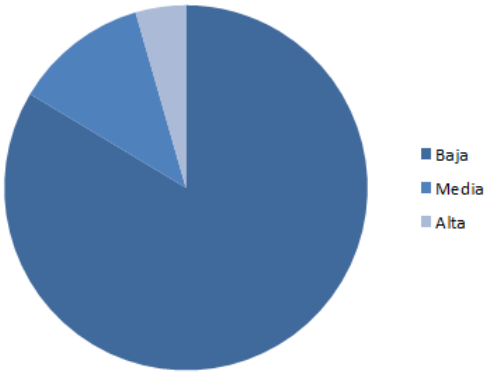

Figura 17: Resultados de la métrica RC para los atributos Complejidad de mantenimiento y Cantidad de pruebas.

Reutilización:

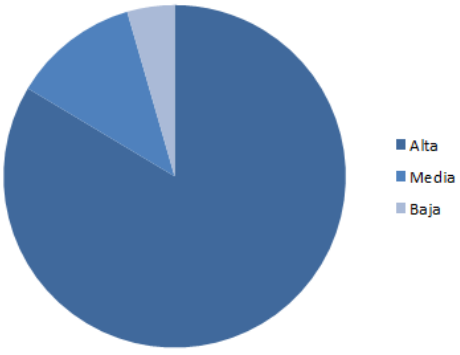

Figura 18: Resultados de la evaluación de la métrica RC para el atributo Reutilización.

Conclusiones del Capítulo

Durante la creación del plugin Xaphiro en base a la metodología OpenUP y los patrones arquitectónicos y de diseño propuestos se pudo concluir que:

- La flexibilidad de la metodología OpenUP permitió un desarrollo centrado en la implementación, a partir de la creación de solo los artefactos necesarios e ilustrativos del proceso, muy necesario a causa de un cronograma apretado y un equipo de trabajo pequeño.
- La validación del diseño, junto a la utilización de claros estándares de código y Patrones de Diseño reconocidos, evidenció la construcción de un plugin legible, reutilizable y de fácil mantenimiento, elementos que soportan la posibilidad de incrementar el alcance del proyecto para futuras versiones y garantizar el aporte tecnológico que representa Xaphiro para el centro y la universidad.

CAPÍTULO 3: IMPLEMENTACIÓN, PRUEBA Y VALIDACIÓN DE LA SOLUCIÓN

Introducción

En el presente capítulo se describen las principales funcionalidades implementadas así como las pruebas realizadas al software desarrollado, con el objetivo de comprobar las funcionalidades del plugin en los diferentes escenarios, verificando en todos los casos que los resultados de las pruebas sean los esperados.

3.1 Descripción de las funcionalidades de Xaphiro

3.1.1 Crear JSP

La funcionalidad Crear JSP permite al desarrollador crear un nuevo fichero JSP teniendo en cuenta la estructura definida para los mismos dentro del proyecto SAGEB y haciendo uso de una ventana de diálogo que proporciona el Eclipse.

3.1.2 Asociar fichero JS

Permite al desarrollador asociar a un fichero JSP un fichero JS el cual contendrá todo el código JavaScript correspondiente a la decoración de los componentes HTML existentes dentro del JSP.

Para llevar a cabo el proceso de asociación el plugin brinda una ventana de diálogo que permite al desarrollador asociar un JS existente o crear uno nuevo y asociarlo.

3.1.3 Decorar JSP

Permite al desarrollador la creación, modificación y eliminación de componentes dentro de un JSP. Para la creación de un componente, el desarrollador selecciona el elemento deseado desde la paleta de componentes, luego lo suelta dentro del editor web del plugin. El plugin genera el código asociado, tanto HTML como JavaScript. Una vez creado el componente, las propiedades del mismo pueden ser modificadas tanto de forma manual como a través de la vista Propiedades del Eclipse.

Para la eliminación de componentes, solo es necesario seleccionar los elementos a eliminar y presionar la tecla DEL, el sistema se encarga de eliminar el código asociado a cada componente.

3.1.4 Configurar Xaphiro

Permite al desarrollador configurar parámetros necesarios para el correcto funcionamiento del plugin. Para ello, Xaphiro añade una nueva página de configuración a la ventana *Preferencias* del Eclipse.

Se configuran dos parámetros fundamentales:

1. Las etiquetas de Spring empleadas para la generación del código dentro del JSP. Estas etiquetas son insertadas al inicio de cada JSP.
2. Las relaciones existentes entre ficheros JSP y JavaScript, se pueden asociar nuevos ficheros o eliminar las relaciones existentes.

3.2 Pruebas de Software

Las pruebas tienen gran importancia en el desarrollo de un software, ya que mediante éstas se pueden detectar y corregir errores tempranamente. Antes de entregar el producto al cliente final se debe garantizar que el software cumpla con todos los requerimientos y se han corregido todos los errores, pues cada vez que el programa se ejecuta el cliente lo está probando, por tanto, se debe hacer un esfuerzo en garantizar la calidad del producto y con ello que el cliente final se sienta satisfecho. Con el objetivo de encontrar el mayor número posible de errores, las pruebas deben conducirse sistemáticamente y los casos de prueba deben diseñarse utilizando técnicas definidas.

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. (24)

3.2.1 Pruebas de Caja Blanca

Las pruebas de caja blanca: denominadas pruebas de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. En estas pruebas se examina el programa en varios puntos sobre el código para determinar si el estado real coincide con el esperado. Así como las pruebas de caja negra ejercitan los requisitos funcionales desde el exterior del módulo, las de caja blanca están dirigidas a las funciones internas. Estas pruebas se llevan a cabo en primer lugar sobre un módulo concreto, para luego realizar las de caja negra sobre varios subsistemas. (24)

Objetivo

El objetivo de realizar este tipo de prueba al sistema es que se garantice que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo o método, todos los bucles en sus límites operacionales así como las estructuras internas de datos para asegurar su validez. (24)

Alcance

El proceso de pruebas de caja blanca se va a concentrar principalmente en validar a través del framework de software libre JUnit, que cada uno de los módulos o segmentos de códigos funcione apropiadamente.

Descripción

La prueba de caja blanca es considerada como uno de los tipos de pruebas más importantes que se le aplican a los software, logrando como resultado que disminuya en un gran por ciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad. (24)

Para el desarrollo de estas pruebas unitarias se utilizó framework JUnit, ya que ofrece las funcionalidades necesarias para implementar pruebas en un proyecto desarrollado en Java. Además cuenta con una interface simple que informa si cada una de las pruebas realizadas o conjunto de pruebas falló, pasó o fue ignorada.

Las pruebas se realizaron a las clases que contienen funcionalidades de determinan el correcto funcionamiento del plugin, entre ellas están: *JSParser, JSWriter, DojoElement, DojoFigureFactory, SimpleGraphicalEditor, DojoDecoratorEditor y DojoElementEditPart.*

Para explicar el proceso de realización de las pruebas de Caja Blanca con JUnit se toma como objeto de estudio el método *parse*(IDocument aSourceDocument) que se encuentra en la clase JSParser, el cual se muestra a continuación. Los resultados de las pruebas aplicadas a las demás clases aparecen plasmados en los anexos del presente trabajo.

La realización de pruebas haciendo uso de JUnit permite comprobar si el resultado obtenido luego de la ejecución de la funcionalidad a probar es el esperado. Para la realización de pruebas al método previamente seleccionado primeramente se creó una clase la cual debe extender la clase TestCase brindada por el framework JUnit, en este caso dicha clase se nombra ParserTester en la cual se realizan una serie de pasos los cuales se muestran y describen a continuación.

```
public List parser(IDocument aSourceDocument) {
      sourceDocument = aSourceDocument;
      scanner.setRange(sourceDocument, 0, sourceDocument.getLength());
      IToken token = scanner.nextToken();
      while (!token.isEOF()) {
            int offset = scanner.getTokenOffset();
            int length = scanner.getTokenLength();
            String expression = getExpression(offset, length);

            if (token.equals(JSSyntaxScanner.TOKEN_FUNCTION))
                  addFunction(expression, offset, length);

            else if (token.equals(JSSyntaxScanner.TOKEN_DECORATION))
                  addDecoration(expression, offset, length);

            else if(token.equals(JSSyntaxScanner.TOKEN_DOJO_CONNECT))
                  addDojoConnect(expression, offset, length);

            else if(token.equals(JSSyntaxScanner.TOKEN_DOJO_STYLE))
                  addDojoStyle(expression, offset, length);

            else if(token.equals(JSSyntaxScanner.TOKEN_VARIABLE) &&
                    expression.contains("="))
                  addVariable(expression, offset, length);

            token = scanner.nextToken();
      }

      return elementList;
}
```

Figura 19: Método empleado en las pruebas de Caja Blanca con JUnit

```
public class ParserTester extends TestCase{

    private JSParser parser;
    private IDocument aSourceDocument;

    @Before
    public void setUp() throws Exception {
        parser = new JSParser();
        aSourceDocument = new Document(TestUtil.getText(new File("D:\\WSpaces\\junit.js")));
    }
```

Figura 20: Clase ParserTester empleada para la realización de las pruebas

Primeramente se crea un objeto de la clase en la cual se encuentra el método que se va a probar, luego se definen cada uno de los parámetros que recibe el método. Posteriormente en el método setUP () se inicializan todos los atributos anteriormente creados.

```
@Test
public void testParseIDocument() {
    List result = parser.parse(aSourceDocument);
    assertNotNull(result);
    assertEquals(result.size(), 3);
    assertEquals(((JSFunctionElement) result.get(0)).getName(), "init");
    assertEquals(((JSFunctionElement) result.get(0)).getArguments(), "()");

    assertEquals(((Form) result.get(1)).getId(), "form1");
    assertEquals(((PropertyDescriptor) ((Form) result.get(1))
            .getProperties().get("method")).getValue(), "'POST'");

    assertEquals(((TextBox) result.get(2)).getId(), "input1");
    assertEquals(((PropertyDescriptor) ((TextBox) result.get(2))
            .getProperties().get("style")).getValue(), "'margin: 10px'");
    assertEquals(((PropertyDescriptor) ((TextBox) result.get(2))
            .getProperties().get("required")).getValue(), "true");
}
```

Figura 21: Método testParseIDocument de la clase ParserTester

Para la realización de las pruebas al método *parse*(IDocument aSourceDocument), se hace uso del método *testParseIDocument ()* perteneciente a la clase ParserTester, primeramente se ejecuta la funcionalidad a probar y los resultados son almacenados dentro de una lista. Luego, haciendo uso del método *assertEquals (Object expected, Object actual)* el framework JUnit comprueba que cada uno de los resultados obtenidos coincide con los resultados esperados y muestra en una ventana los resultados obtenidos, en caso de obtener resultados satisfactorios para todas las pruebas realizadas se observa una línea verde en la ventana, en caso contrario aparece una línea roja. En la figura 25 aparecen los resultados de las pruebas realizadas.

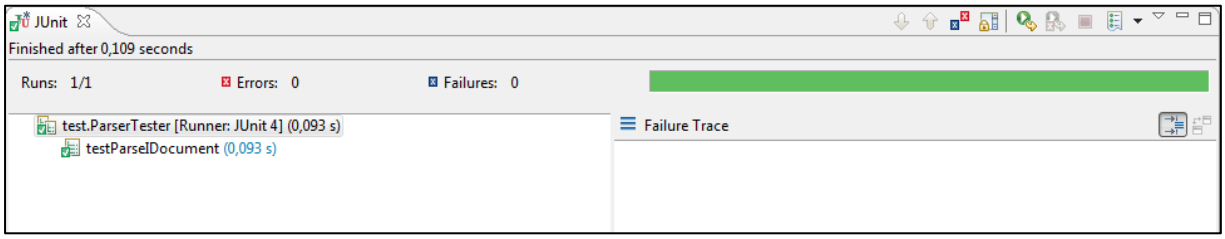

Figura 22: Resultados de las pruebas con JUnit

3.2.2 Pruebas de Caja Negra

Las pruebas de caja negra: verifican las especificaciones funcionales sin tener en cuenta la estructura interna del programa y son realizadas sin el conocimiento interno del producto. El objetivo es demostrar que las funciones del software son operativas, que las entradas se aceptan de forma

adecuada y se produce un resultado correcto, además que la integridad de la información externa se mantiene, saber qué es lo que hace el software pero sin entrar en detalles de código, es decir, que es lo que hace, y no cómo lo hace. Por ello se realizan sobre la interfaz del sistema controlando los datos de entrada y de salida. (24)

Objetivo

El objetivo de realizar este tipo de prueba al sistema es para detectar el incorrecto o incompleto funcionamiento de este, así como los errores de interfaces, rendimiento y errores de inicialización y terminación. (24)

Alcance

El proceso de pruebas de caja negra se va a centrar principalmente en los requisitos funcionales del software para verificar el comportamiento de la unidad observable externamente y la calidad funcional.

Descripción

Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales del programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

La prueba de caja negra no es una alternativa a las técnicas de prueba de la caja blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la caja blanca. (25)

Existen varias técnicas para desarrollar la prueba de caja negra, entre ellas están: (26)

- Técnica de la Partición de Equivalencia: divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. En esencia esta técnica se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.

- Técnica del Análisis de Valores Límites: los errores tienden a darse más en los límites del campo de entrada que en el centro. Esta técnica lleva a una elección de casos de prueba que ejerciten los valores límites.

- ✓ Condiciones sublímite: las condiciones límite normales son las más obvias de descubrir. Estas son definidas en la especificación o son evidentes al momento de utilizar el software. Algunos límites, sin embargo, son internos al software, no son necesariamente aparentes al usuario final pero aun así deben ser probadas por el probador. Estas son conocidas como condiciones sublímite o condiciones límite internas.

- Técnica de prueba basada en grafos: permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

  - ✓ Modelado del flujo de transacción: los nodos representan los pasos de alguna transacción, y los enlaces representan las conexiones lógicas entre los pasos.

  - ✓ Modelado de estado finito: los nodos representan diferentes estados del software observables por el usuario, y los enlaces representan las transiciones que ocurren para moverse de estado a estado.

  - ✓ Gráfica Causa-efecto: la gráfica Causa-efecto representa una ayuda gráfica en seleccionar, de una manera sistemática, un gran conjunto de casos de prueba. Tiene un efecto secundario beneficioso en precisar estados incompletos y ambigüedades en la especificación.

Dentro de las técnicas de prueba de caja negra se utilizó la de Partición Equivalente. Esta técnica permite obtener un conjunto de pruebas que reducen el número de casos de prueba que se deben realizar para evaluar correctamente el software, esto se debe a que se centra en la evaluación de clases de equivalencia que representan un conjunto de estados válidos o no válidos para condiciones de entradas existentes en el software.

Las pruebas se realizaron sobre los requisitos funcionales planteados que requieren de la interacción de los componentes del plugin, la gestión del componente de Dojo y la modificación del código HTML y JavaScript, concentrándose en cada una de las principales debilidades que pudieran presentarse:

- En Crear Componente se concentran en los eventos de Drag and Drop.
- En Eliminar y Actualizar Componente se concentran en el carácter bidireccional de los cambios desde y hasta los principales componentes del plugin: el editor de código JavaScript, el editor de código HTML y la vista de diseño del JSP.
- En el Caso de uso crear JSP se enfoca en la creación del nuevo fichero.

Nombre de la Prueba:	Crear Componente		
Caso de Uso Probado:	Decorar JSP		
Descripción de la Prueba:	Se crea el componente de Dojo seleccionado dentro de la vista, haciendo Drag and Drop desde la Paleta de componentes del plugin hacia la vista de diseño del JSP o hacia el editor de código HTML, reflejándose dicha acción en el archivo JavaScript asociado al JSP y en su propio código HTML.		
Pre-condiciones:	Interfaz web abierta con el plugin Xaphiro en el Eclipse 3.6		
Post-condiciones:	Se crea el componente de Dojo		
Notas:	Requisito funcional Crear componente de Dojo		
Resultado: (Pasa/Falla/aviso/incompleto)	Pasa		
Pasos de Prueba	**Resultados Esperados de Prueba**	**P**	**F**
1. Se selecciona el elemento de la Paleta y se suelta fuera de la vista de diseño	No suceden cambios	X	
2. Se selecciona el elemento de la Paleta y se suelta fuera del editor HTML	No suceden cambios	X	
3. Se selecciona el elemento de la Paleta y se suelta dentro de la vista de diseño JSP o dentro del código HTML	Se genera el elemento dentro del JSP, código y diseño, y en el archivo JavaScript asociado	X	
Nombre de la Prueba:	Actualizar Componente desde Paleta		
Caso de Uso Probado:	Decorar JSP		
Descripción de la Prueba:	Se selecciona un componente del JSP en su vista de diseño o en su código HTML, se le adicionan y modifican sus propiedades a través de la paleta del Xaphiro		
Pre-condiciones:	Interfaz web abierta con el plugin Xaphiro en el Eclipse 3.6		
Post-condiciones:	Se actualiza el componente de Dojo		

Notas:	Requisito funcional Modificar Componente de Dojo		
Resultado: (Pasa/Falla/aviso/incompleto)	Pasa		
Pasos de Prueba	**Resultados Esperados de Prueba**	**P**	**F**
1. Se selecciona un componente del JSP que no está decorado	La paleta de propiedades no muestra nada	X	
2. Se selecciona un componente del JSP decorado	La paleta muestra las propiedades actuales de dicho componente	X	
3. Se escribe un valor dentro del campo de entrada de la propiedad	El componente acepta los cambios mostrando los nuevos valores en la paleta de propiedades y en el archivo JavaScript asociado	X	
Nombre de la Prueba:	Actualizar Componente desde JavaScript		
Caso de Uso Probado:	Decorar JSP		
Descripción de la Prueba:	Se selecciona la vista del editor JavaScript del plugin y se modifica el texto que representa a un componente de Dojo, transformándose el contenido que este muestra en la Vista de Propiedades.		
Pre-condiciones:	Interfaz web abierta con el plugin Xaphiro en el Eclipse 3.6		
Post-condiciones:	Se actualiza el componente de Dojo		
Notas:	Requisito funcional Modificar Componente de Dojo		
Resultado: (Pasa/Falla/aviso/incompleto)	Pasa		
Pasos de Prueba	**Resultados Esperados de Prueba**	**P**	**F**
1. Se modifica texto JavaScript ajeno a la decoración de un componente	No suceden cambios visuales	X	
2. Se modifica el texto correspondiente a un elemento de Dojo	Se modifican las propiedades del componente que muestra la vista de propiedades del plugin	X	
Nombre de la Prueba:	Actualizar código HTML		

Caso de Uso Probado:	Decorar JSP		
Descripción de la Prueba:	Se modifica el código HTML del JSP al adicionar elementos HTML, eliminarlos o variar sus propiedades		
Pre-condiciones:	Interfaz web abierta con el plugin Xaphiro en el Eclipse 3.6		
Post-condiciones:	Se modifica el elemento HTML		
Notas:	Requisito Funcional Modificar código HTML		
Resultado: (Pasa/Falla/aviso/incompleto)	Pasa		
Pasos de Prueba	**Resultados Esperados de Prueba**	**P**	**F**
1. Se escribe código HTML incorrecto	El editor de código HTML del plugin señala la línea del error	X	
2. Se escribe código HTML que genera un nuevo elemento en la página	La vista de diseño JSP muestra el nuevo componente HTML	X	
3. Se modifica el código HTML perteneciente a un elemento de la página JSP	La vista de diseño JSP del plugin muestra los cambios visuales, de ocurrir alguno, en el elemento modificado	X	
Nombre de la Prueba:	Eliminar Componente		
Caso de Uso Probado:	Decorar JSP		
Descripción de la Prueba:	Se elimina el elemento de la vista de diseño JSP y se elimina de forma automática el código HTML y JavaScript asociados		
Pre-condiciones:	Interfaz web abierta con el plugin Xaphiro en el Eclipse 3.6		
Post-condiciones:	Se elimina el componente de Dojo		
Notas:	Requisito Funcional Eliminar componente de Dojo		
Resultado: (Pasa/Falla/aviso/incompleto)	Pasa		
Pasos de Prueba	**Resultados Esperados de Prueba**	**P**	**F**
1. Se selecciona y elimina un elemento de la vista de diseño no decorado	Se elimina el elemento dentro de la vista del JSP y en el código HTML	X	
2. Se selecciona y elimina un elemento decorado con	Se elimina el elemento en su código HTML y en el JavaScript asociado	X	

Dojo de la vista de diseño			
Nombre de la Prueba:	Crear fichero JSP		
Caso de Uso Probado:	Crear JSP		
Descripción de la Prueba:	Se crea un fichero JSP con el encabezado definido para la aplicación Quarxo y el nombre determinado por el usuario en el directorio seleccionado		
Pre-condiciones:	Eclipse 3.6		
Post-condiciones:	Se crea el fichero JSP		
Notas:	Requisito Funcional Crear fichero JSP		
Resultado: (Pasa/Falla/aviso/incompleto)	Pasa		
Pasos de Prueba	**Resultados Esperados de Prueba**		
1. Se selecciona un paquete o carpeta dentro del proyecto y se crea un nuevo Xaphiro JSP	El Eclipse genera el wizard para el Xaphiro JSP		
2. Se genera un nombre incorrecto para el Xaphiro JSP, comenzando con números o incluyendo símbolos extraños	El wizard suspende la opción de terminar la creación del fichero		
3. Se genera un nombre existente en el directorio seleccionado para el fichero	El wizard suspende la opción de terminar la creación del fichero		
4. Se genera, para el fichero, un nombre correcto y nuevo en el directorio	El wizard habilita la opción de terminado y acepta la creación del fichero, que se genera en la dirección señalada		

Tabla 13: Pruebas realizadas al plugin Xaphiro

3.2 Validación

Para validar el plugin Xaphiro se crearon dos grupos de programadores, grupo A y grupo B, formados por tres estudiantes. Uno de los estudiantes seleccionados de cada grupo posee registrado un Curso

Optativo de Dojo mientras que los otros 2 tendrán poco o ningún conocimiento sobre el framework o el lenguaje JavaScript.

El grupo A realizó una serie de tareas referentes al diseño de presentación haciendo uso de editores de JavaScript y HTML del Eclipse, vía plugin WTP, mientras que el grupo B lo hizo a través del plugin Xaphiro, previa capacitación para su uso.

Tarea 1: Decorar una vista desde cero. La vista a decorar deberá poseer los componentes principales usados en el proyecto. Esta tarea incluye la creación del fichero JSP a decorar.

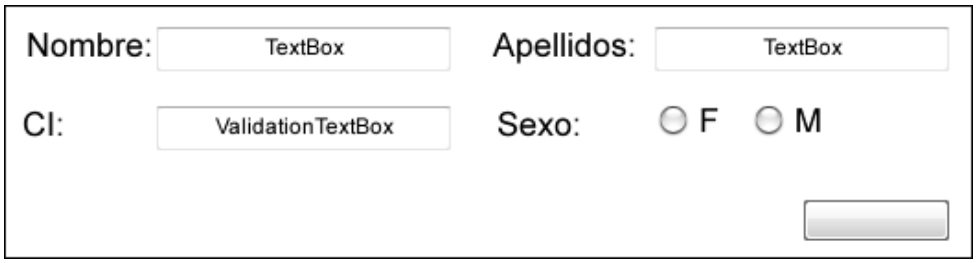

Figura 23: Boceto de la vista a crear por los desarrolladores

Tarea 2: Cambiar propiedades de varios elementos de una vista previamente decorada. Los cambios serán sobre propiedades de algunos componentes de la vista y la adición de nuevas propiedades a dichos componentes.

Elemento	TextBox *Nombre*	**Elemento**	RadioButton *F*	**Elemento**	ValidationTextBox *Carnet*
Propiedad	**Valor**	**Propiedad**	**Valor**	**Propiedad**	**Valor**
id	nombre	id	sexoF	id	ci
required	true	checked	true	required	true
alt	Entre su nombre	alt	Sexo femenino	alt	Entre su CI
				regExp	/^d{11}$/

Tabla 14: Elementos y propiedades a modificar en la tarea 2

Luego de cumplidas las tareas por cada grupo se midieron los tiempos totales de cumplimiento y se contó el número de errores en que incurrieron sus miembros.

Grupo	**Número**	**Curso de Dojo (Si/No)**	**Tarea 1**		**Tarea 2**	
			Tiempo	**Errores**	**Tiempo**	**Errores**
A	1	Si	**00:07:10**	1	**00:02:15**	0
	2	No	00:12:20	3	00:03:23	2

	3	No	00:14:06	6	00:04:01	3
B	1	Si	00:03:35	0	**00:01:00**	0
	2	No	**00:03:32**	0	00:01:01	0
	3	No	00:03:45	0	00:01:09	1

Tabla 15: Resultados obtenidos tras la realización de las tareas

Teniendo en cuenta los resultados obtenidos (el texto en negrita representa el mejor tiempo de cada equipo para cada tarea, mientras que el texto subrayado indica el peor tiempo), se puede apreciar que el uso del plugin Xaphiro agiliza en un tiempo considerable el desarrollo de aplicaciones Web que hagan uso de Dojo para la Capa de Presentación y reduce la necesidad de contar con expertos conocedores del framework para la creación de interfaces, ya que el tiempo empleado por cualquier desarrollador, haciendo uso de la herramienta, es casi idéntico. También se manifiesta una reducción en los errores cometidos por los desarrolladores que emplearon Xaphiro en comparación con los que utilizaron WTP, lo cual evidencia la confiabilidad del código generado con la primera herramienta y reduce el tiempo de revisión del mismo.

Conclusiones del Capítulo

Las pruebas realizadas y la validación de la solución son el colofón de este arduo trabajo. Con la confección de este último capítulo se arribó a las conclusiones siguientes:

- El framework JUnit aumenta las posibilidades que brinda el lenguaje Java para las pruebas de caja blanca y de manera sencilla .demuestra la calidad del código generado.
- Los resultados de las pruebas realizadas lograron su objetivo principal: Demostrar la calidad de la solución propuesta a través de un código limpio altamente funcional y un funcionamiento general exitoso. Se constató la eficacia de la herramienta JUnit para pruebas de caja blanca y de la técnica Partición de Equivalencia para el funcionamiento de la aplicación.

CONCLUSIONES

Después de la investigación y posterior implementación, reflejadas en este documento, se ha llegado al siguiente conjunto de conclusiones generales.

El estudio de los sistemas que proveen paleta de componentes y la posterior creación de Xaphiro permitió culminar, basándose en los resultados de las pruebas realizadas, que la utilización de una paleta de componentes, en este caso específico de Dojo, disminuye en gran medida el tiempo de confección de interfaces web.

La caracterización de la arquitectura del Eclipse y su posterior puesta en práctica, a través de la creación de un plugin, denotó la efectividad y extensibilidad de la misma, las ventajas de trabajo que ofrece al desarrollador y las ilimitadas ventajas de trabajo con este Entorno de Desarrollo Integrado.

La metodología OpenUP dirige sus esfuerzos a proyectos pequeños y permite a su vez ser extendida a trabajos de mayor complejidad, sus características le permiten poseer lo mejor de ambos mundos, el de las metodologías ágiles y el de las tradicionales, en pos de un resultado en tiempo y acorde a las necesidades.

Xaphiro al ser parte del Eclipse se extiende a diferentes sistemas operativos, por ser de código abierto su aporte se abre a un variado rango de aplicaciones y por estar orientado a necesidades sobre la web representa una solución práctica a una problemática creciente.

RECOMENDACIONES

- La construcción de plugins es un tema poco explotado en la UCI, por lo que se recomienda que se amplíe su uso. Entre las principales ramas que se beneficiarían está la Programación de compiladores e intérpretes que, al igual que Xaphiro, se vale de Escáner y reconocedores de lenguajes para su funcionamiento.
- Asociada a esta primera recomendación está la de aumentar el número de cursos optativos sobre Eclipse para profundizar su conocimiento en la comunidad.
- A partir de Xaphiro, y tomándolo como base, se recomienda la creación de otros plugins para versiones más avanzadas de Dojo, para otros frameworks JavaScript y la creación de un plugin genérico que brinde la posibilidad de especificar el código a generar para cada componente de la paleta.
- Realizar pruebas de calidad al plugin en el centro Calisoft de la Universidad.

BIBLIOGRAFÍA

1. www.searchsoftwarequality.techtarget.com. *What is integrated development environment? – a definition from Whatis.com.* [En línea] [Citado el: 8 de diciembre de 2011.] http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci754848,00.html.

2. www.Eclipse.org. *About the Eclipse Foundation.* [En línea] [Citado el: 6 de diciembre de 2011.] http://www.eclipse.org/org.

3. **C. L., John Arthone.** Official Eclipse 3.0 Faqs. Addison-Wesley Professional. s.l. : Eclipse Series, 2004.

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

5. **Álvarez, Miguel Ángel.** www.desarrolloweb.com. [En línea] [Citado el: 8 de diciembre de 2011.] http://www.desarrolloweb.com/articulos/391.php.

6. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El proceso unificado de desarrollo de software. Addison-Wesley.* Madrid : s.n., 2000.

7. **Cambiaso, Diego.** Pixelco. *www.pixelco.us.* [En línea] [Citado el: 9 de diciembre de 2011.] http://pixelcoblog.com/recopilacion-de-frameworks-bibliotecas-y-otros-recursos-para-desarrollo-web.

8. **The Dojo Foundation.** The Dojo Foundation. [En línea] [Citado el: 15 de marzo de 2012.] http://dojofoundation.org.

9. **Eguíluz Pérez, Javier.** *Introducción a JavaScript.* junio : 7, 2008.

10. www.wavemaker.com. *WaveMaker.* [En línea] [Citado el: 12 de diciembre de 2011.] http://www.wavemaker.com.

11. **The Dojo Foundation.** The Dojo Foundation. [En línea] The Dojo Foundation. [Citado el: 15 de marzo de 2012.] http://dojofoundation.org/projects/maqetta.

12. —. Maqetta. [En línea] The Dojo Foundation. [Citado el: 15 de marzo de 2012.] http://maqetta.org/.

13. **Clayberg, Eric y Rubel, Dan.** *Eclipse plug-ins.* s.l. : The Eclipse Series, 2008.

14. **Nagel, William.** *Subversion Version Control.* s.l. : Pearson Education, 2005. 0-13-185518-2.

15. **Figueroa, Roberth G. y Solís, Camilo J.** Metodologías Tradicionales vs. Metodologías Ágiles. [En línea] [Citado el: 10 de diciembre de 2011.] http://www.mygnet.net/manuales/software/metodologias_tradicionales_vs_dot_metodologias_agiles.

16. **Kniberg, Henrik.** *Scrum y XP desde las trincheras.* s.l. : C4Media, 2007. 978-1-4303-2264-1.

17. **Eclipse Fundation.** OpenUP. [En línea] [Citado el: 20 de febrero de 2012.] http://epf.eclipse.org/wikis/openup/.

18. **Booch, G, Rumbaugh, J y Jacobson, I.** *The unified modeling language. User guide.* 1999.

19. **Meyer, Jeremy y Thomas, Dunstan.** *Comparison of UML Modelling Tools: Sparx Systems "Enterprise Architect" and IBM Rational "Rose".* 2005.

20. www.visual-paradigm.com. *UML CASE Tools - Free for Learning UML, Cost-Effective for Business Solutions.* [En línea] [Citado el: 6 de diciembre de 2011.] http://www.visual-paradigm.com/product/vpuml.

21. **Pello, Javier.** SoftQaNetwork. [En línea] 11 de abril de 2009. [Citado el: 6 de febrero de 2012.] http://www.softqanetwork.com/requisitos-no-funcionales-nfr.

22. Patrones GRASP. [En línea] [Citado el: 10 de Noviembre de 2011.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.

23. **John Metsker, Steven.** *Design Patterns Java Workbook.* s.l. : Addison Wesley, 2002.

24. **Pressman, Roger S.** *Ingeniería del Software: Un enfoque práctico.* 2005.

25. **Grupo Alarcos.** Grupo Alarcos - Universidad de Castilla-La Mancha. [En línea] [Citado el: 2 de abril de 2012.] http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema09.pdf.

26. **Moreno Álvarez, José Luis.** Colección de Tesis Digitales, Universidad de las Américas Puebla. [En línea] http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/moreno_a_jl/portada.html.

27. Toolkits . *www.fismat.umich.mx.* [En línea] [Citado el: 9 de diciembre de 2011.] http://www.fismat.umich.mx/~crivera/tesis/node20.html.

28. Ecured. [En línea] [Citado el: 20 de febrero de 2012.] http://www.ecured.cu/index.php/Validaci%C3%B3n_de_Requisitos.

29. **Gutiérrez Rosales, Rolando y Cabrera Fleites, Dayana.** *Adaptación de OpenUp/Basic para el Polo Productivo de BioInformática.* 2009.

30. **Reynoso, Carlos y Kiccillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004.

31. **Torres Flores, Carmina Lizeth .** *Establecimiento de una Metodología de Desarrollo de Software para la Universidad de Navojoa Usando OpenUP.* 2008.

32. **Kroll, Per y Macisaac, Bruce.** *Agility and Discipline Made Easy: Practices from OpenUP and RUP.* s.l. : Addison-Wesley, 2006.

33. **López González, Héctor Luis.** *Plug-in para Eclipse para la generación de elementos arquitectónicos personalizados.* 2009.

34. **Scarpino, Matthew.** *SWT/JFace in Action.* s.l. : Manning Publications, 2005.

35. **Li Guojie, Jackwind.** *Professional Java Interfaces with SWT/JFace.* s.l. : John Wiley & Sons, 2005.

36. **Massol, Vincent y Husted, Ted.** *JUnit in Action.* s.l. : Manning Publications, 2004.