Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No.2, Mes Febrero, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 1-15
http://publicaciones.uci.cu

Tipo de artículo: Artículo original
Temática: soluciones informáticas
Recibido: 11/10/2018 | Aceptado: 22/12/2018 | Publicado: 28/01/2019

Módulo de diseño de mapas de impresión para la plataforma ULTRON

Print map design module for the ULTRON platform

Yelena Vento Diaz[1], Gerdys Ernesto Jiménez Moya[2], Reinier Suarez Estevez[3]

[1] Facultad de Ciencias y Tecnologías Computacionales, Universidad de las Ciencias Informáticas, yvento@estudiantes.uci.cu
[2] Facultad de Ciencias y Tecnologías Computacionales, Universidad de las Ciencias Informáticas, gejimenez@uci.cu
[3] Facultad de Ciencias y Tecnologías Computacionales, Universidad de las Ciencias Informáticas, restevez@uci.cu

[*] Autor para correspondencia: yvento@estudiantes.uci.cu

Resumen

Los Sistemas de Información Geográfica constituyen una herramienta de gestión y análisis de datos espaciales con capacidades de producción cartográfica. Estas capacidades permiten la generación de documentos de acuerdo con un diseño determinado. La presente investigación tiene como objetivo desarrollar el Módulo de diseño de mapas de impresión para la plataforma ULTRON del centro de Geoinformática y Señales Digitales (GEYSED), que permita a los usuarios/clientes capacitados realizar el proceso de diseño y personalización cartográfica de los mapas y sus elementos. Se realizó un análisis a un conjunto de sistemas homólogos que permitió determinar varios aspectos importantes a tener en cuenta en el desarrollo del módulo. Para la implementación de la solución se optó por la utilización de las tecnologías y lenguajes de programación: HTML5, CSS3, JavaScript (tanto del lado del cliente como del lado del servidor) y como framework de desarrollo SEAN.JS. Además, para guiar el proceso de desarrollo se utilizó la metodología AUP-UCI, adaptada a los procesos productivos de la universidad. Como resultado se obtuvo el desarrollo de la aplicación

Palabras claves: diseño cartográfico, mapas de impresión, sistemas de información geográfica.

Abstract

The Geographic Information Systems are a spatial data management and analysis tool with cartographic production capabilities. These capabilities allow the generation of documents according to a specific design. The objective of this research is to develop the Print Map Design Module for the ULTRON platform of the Geoinformatics and Digital Signals Center (GEYSED), which allows trained users / clients to carry out the design and cartographic customization of the maps and its elements. An analysis was made to a set of homologous systems that allowed to determine several important aspects to be taken into account in the development of the module. For the implementation of the solution we chose the use of programming languages and technologies: HTML5, CSS3, JavaScript (both on the client side and on the server side) and as a development framework SEAN.JS. In addition, to

guide the development process, the AUP-UCI methodology was used, adapted to the productive processes of the university. As a result, the development of the application was obtained.

Keywords: *cartographic design, printing maps, geographic information systems*

Introducción

La mayoría de los Sistemas de Información Geográfica (SIG) incorporan capacidades de creación de cartografía impresa, generando un documento cartográfico que posteriormente puede imprimirse y emplearse como un mapa clásico. Fundamentalmente, estas capacidades permiten la composición de documentos cartográficos de acuerdo con un diseño dado. En la elaboración de dicho diseño, pueden emplearse los elementos que habitualmente se encuentran en un mapa: el propio mapa en sí, leyenda, título, escala, entre otros. Con estos elementos, se crea una versión personalizada de la información geográfica, que puede ya emplearse de modo independiente del SIG. (Olaya, 2014)

Estos mapas constituyen una personalización que contiene y está enriquecido con la información que es de interés para cada negocio o sector al que está dirigido. La carencia de estos se refleja directamente en la toma de decisiones de una institución o empresa por parte de sus directivos, ya que al no contar con la información detallada y necesaria para realizar un correcto análisis, se puede incurrir en la pérdida de tiempo y de recursos.

En la Universidad de las Ciencias Informáticas (UCI), en el centro de Geoinformática y Señales Digitales (GEYSED), se desarrolla la plataforma web para el desarrollo de Sistemas de Información Geográfica denominada ULTRON (Ultimate Run Of Nodejs). Este sistema está implementado sobre tecnologías libres, haciendo uso de SEAN-JS, un Full-Stack de JavaScript que integra las librerías de AngularJS, Express, Sequelize, corriendo sobre el servidor NodeJS.

Actualmente, la plataforma ULTRON cuenta con un componente para exportar en una imagen de formato PNG el área del mapa que se está visualizando. Sin embargo, existen limitaciones que afectan el resultado final y por ende los análisis para los que son objetos los documentos generados. Estas limitaciones se reflejan al no permitir la inclusión al documento de impresión de elementos tales como: la leyenda, la escala gráfica y el mapa de referencia correspondiente. Además, de forma específica el componente actual no permite localizar el mapa en el ubicación ni la orientación deseada en el documento, así como la configuración de elementos del mapa que se está consultando. Todo esto constituye un obstáculo para el enriquecimiento de la información a imprimir.

Estas limitaciones no posibilitan la generación de un mapa de impresión personalizado, obteniéndose un documento con carencia en la descripción completa y detallada de los objetos espaciales que en él se visualizan, lo que impide

obtener salidas similares a los mapas convencionales impresos. A causa de esto, los usuarios no cuentan con un documento con todos los detalles gráficos necesarios para una correcta toma de decisiones, lo que afecta el análisis realizado sobre la información geoespacial y por tanto influye de forma negativa sobre los resultados provenientes de dichos análisis.

Materiales y métodos o Metodología computacional

La presente sesión se describe la solución propuesta mediante la identificación de los requisitos funcionales y no funcionales con los que debe cumplir el módulo. Para ello se hace necesario seguir los pasos de la metodología propuesta en el capítulo anterior (AUP-UCI) en su escenario 4, y elaborar los artefactos fundamentales del proceso de desarrollo. También se presenta el diseño arquitectónico y se ejemplifica el empleo de los patrones del diseño.

Requisitos de Software

La ingeniería de requisitos del software es un proceso de descubrimiento, refinamiento, modelado y especificación. Se refinan en detalle los requisitos del sistema y el papel asignado al software. (Pressman, et al., 2015)

Requisitos Funcionales

Los requisitos funcionales son los que definen las funciones que el sistema será capaz de realizar, ya que describen las transformaciones sobre las entradas para producir salidas. Es importante que se describa el qué y no el cómo se deben hacer esas transformaciones. Estos requisitos, al mismo tiempo que avanza el proyecto de software, se convierten en los algoritmos, la lógica y gran parte del código del sistema. (García Velázquez, 2016)

Los requisitos funcionales identificados para el desarrollo del módulo de diseño de mapas de impresión son los siguientes:

RF1: Adicionar documento

El módulo debe permitir adicionar un nuevo documento de impresión.

RF2: Guardar documento

El módulo debe permitir guardar el documento de impresión diseñado en el formato seleccionado.

RF3: Importar plantilla

El módulo debe permitir importar una plantilla de impresión al sistema desde una fuente externa.

RF4: Exportar plantilla

El módulo debe permitir exportar una plantilla de impresión hacia una fuente externa de almacenamiento.

RF5: Seleccionar área del mapa

El módulo debe permitir seleccionar el área del mapa para su diseño.

RF6: Modificar área del mapa

El módulo debe permitir modificar el área de mapa seleccionada anteriormente.

RF7: Adicionar etiqueta de texto

El módulo debe permitir insertar una etiqueta de texto en un documento de impresión.

RF8: Adicionar escala

El módulo debe permitir insertar en el documento la escala del mapa seleccionado.

RF9: Adicionar leyenda

El módulo debe permitir insertar la leyenda del mapa seleccionado.

RF10: Modificar leyenda

El módulo debe permitir modificar la leyenda del documento de impresión adicionada.

RF11: Editar propiedades del documento

El módulo debe permitir modificar las propiedades de salida del documento de impresión como el formato y la resolución del mismo.

RF12: Adicionar mapa de referencia

El módulo debe permitir insertar el mapa de referencia seleccionado al documento de impresión.

Requisitos no funcionales

Los requisitos no funcionales son restricciones en los servicios o funciones ofrecido por el sistema, los cuales incluyen restricciones de tiempo, restricciones en el proceso de desarrollo y restricciones impuestas por los estándares. Estos a menudo se aplican al sistema como un todo en lugar de características o servicios individuales del sistema. Suelen ser más críticos que los requisitos funcionales ya que los usuarios de generalmente pueden encontrar formas de evitar una función del sistema que realmente no satisface sus necesidades. A pesar de ello, no cumplir con un requisito no funcional puede significar que todo el sistema sea inutilizable. (Sommerville, 2016)

Para el desarrollo del módulo de diseño de mapas de impresión fueron definidos los requisitos no funcionales que se muestran a continuación:

Usabilidad:

RNF1. El módulo debe proporcionar una interfaz de usuario intuitiva, con facilidad de uso que permita a los usuarios tener una mejor interacción con las funcionalidades que el mismo brinde.

Rendimiento:

RNF2. El módulo debe responder en un máximo de 6 segundos las solicitudes de los usuarios.

Interfaz:

RNF3. El módulo debe poseer una interfaz acorde a los principios del diseño, sin saturación de colores ni imágenes.

RNF4. El módulo debe poseer una interfaz de usuario basada en Angular Material.

Requerimientos de Hardware:

RNF5. La PC cliente es recomendable que cumpla con los siguientes requisitos mínimos que se especifican a continuación:

- Tarjeta de red, una RAM de 512 MB o superior, procesador a 512 MHz o superior y almacenamiento de 40 GB.

RNF6. La PC servidor de Mapas debe tener como mínimo 2 GB de RAM, 40 GB de disco duro y un procesador de 2 GHz como mínimo.

RNF7. La PC servidor de base de datos debe tener como mínimo 2 GB de RAM, 40 GB de disco duro y un procesador de 2 GHz como mínimo.

RNF8. La PC servidor de aplicaciones es recomendable que cumpla con los siguientes requisitos mínimos que se especifican a continuación: RAM de 4 GB, 40 GB de disco duro y un procesador de 3 GHz.

Requerimientos de Software:

RNF9. La PC Cliente debe poseer un Navegador Web que cumpla con los estándares de la World Wide Web Consortium (W3C).

RNF10. La PC Servidor debe tener servidor Node-JS 4.4 en adelante.

RNF11. La PC servidor debe tener PostgreSQL 9.4 o superior como Sistema Gestor de Base de Datos y PostGIS 2.0 o superior como extensión de PostgreSQL como soporte de datos espaciales.

RNF12. La PC Servidor debe tener MapServer 6.4 o superior.

RNF13. La PC Servidor debe tener Redis-Server 2.2 o superior.

Descripción de los actores que interactúan con el sistema

Un actor es una entidad externa al sistema y que puede interactuar con él. Puede ser una persona o un grupo de personas homogéneas, otro sistema, o una máquina. El actor representa un papel, no a un usuario individual del sistema. El conjunto de funcionalidades a las que un actor tiene acceso define un rol en el sistema y el alcance de su acción. (Mediavilla, 2018)

A continuación, se menciona el actor que va a interactuar con el módulo a desarrollar, definiendo el rol que le ocupa dentro del mismo y su descripción.

Tabla 1: Descripción de los actores del sistema.

Actor	**Descripción**
Especialista	Es el encargado de realizar todo el proceso en la gestión de diseño de mapas de impresión en el módulo.

Resultados y discusiones

Descripción de los requisitos funcionales

Las historias de usuarios son parte del desarrollo ágil de software que ayuda a enfocarse en hablar de los requerimientos en lugar de escribir acerca de ellos. Estas son cortas descripciones de una funcionalidad desde la perspectiva de la persona que la desea, usualmente un usuario o cliente. (Cohn, 2018)

La propuesta de solución presenta un total de 12 requisitos funcionales. En correspondencia con la selección del escenario número cuatro de la metodología empleada se procede a modelar el sistema con historias de usuario, donde se define una por cada requisito funcional. A continuación, se presenta la descripción de los requisitos del Módulo de diseño de mapas de impresión a partir del modelo propuesto por la Historia de usuarios.

Tabla 2: Historia de usuario “Editar propiedades del documento”.

Número: 11	**Nombre del Requisito:** Editar propiedades del documento
Programador: Yelena Vento Diaz	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo estimado:** 9 días
Riesgo en desarrollo: • Planificación irreal. • Interrupción del servicio eléctrico. • Afectaciones externas al personal de trabajo.	**Tiempo Real:** 9 días

Descripción:
Se deben mostrar las propiedades del documento, posibilitando la edición de los siguientes campos:

- **Tamaño de hoja:** Campo que permite seleccionar el formato de hoja en el que se va a exportar el documento, la opción por defecto de este campo es A2.
- **Tipo de documento:** Campo que permite seleccionar el formato del documento. Este puede ser PDF o una imagen PNG, siendo esta última la que se encuentra seleccionada por defecto.
- **Título:** Campo que permite introducir el título del documento de impresión.

Las propiedades del documento se van actualizando automáticamente a medida que se modifican los campos anteriormente descritos.

- **Resolución:** Campo que permite seleccionar la resolución de la imagen del mapa. La opción por defecto de este campo es 72 dpi.

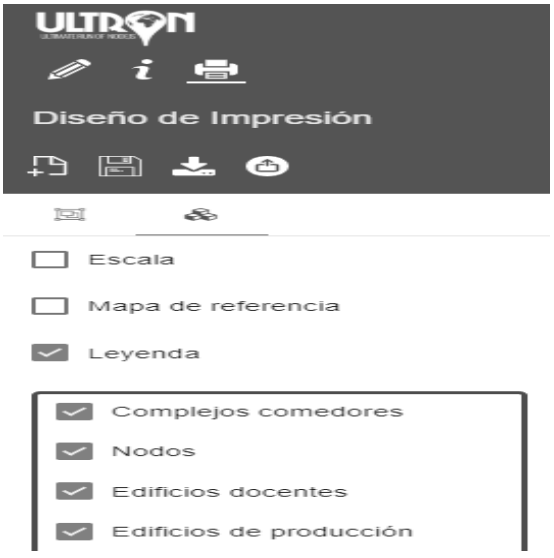
Observaciones:

- El Título es un campo obligatorio, admite valores alfanuméricos.
- A medida que se selecciona un formato de hoja en el campo Formato, se actualizan automáticamente el ancho y altura de la hoja seleccionada.

Prototipo de Interfaz:

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No.2, Mes Febrero, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 1-15
http://publicaciones.uci.cu

Tabla 3: Historia de usuario "Modificar leyenda".

Número: 10	**Nombre del Requisito:** Modificar leyenda
Programador: Yelena Vento Diaz	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo estimado:** 7 días
Riesgo en desarrollo: • Planificación irreal. • Interrupción del servicio eléctrico. • Afectaciones externas al personal de trabajo.	**Tiempo Real:** 6 días
Descripción: Se debe mostrar la interfaz que contiene un árbol con los datos de la leyenda de del mapa. Se marca los Checkbox de los datos de la leyenda que se desean mostrar en la misma. Se da clic en el botón Aceptar de la interfaz, quedando guardado los datos seleccionados. Si se desean cancelar las acciones se da clic sobre el botón Cancelar.	
Observaciones:	
Prototipo de Interfaz: 	

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No.2, Mes Febrero, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 1-15
http://publicaciones.uci.cu

Patrón arquitectónico

La arquitectura de software se refiere a la estructuración del sistema que, idealmente, se crea en etapas tempranas del desarrollo. Representa un diseño de alto nivel que tiene dos propósitos primarios: satisfacer los atributos de calidad y servir como guía en el desarrollo. (Cervantes, 2018)

Según (Huaman, 2018) un patrón arquitectónico es una solución general y reutilizable a un problema común en la arquitectura de software dentro de un contexto dado. Son similares al patrón de diseño de software, pero tienen un alcance más amplio. Además, constituye una colección de decisiones de diseño arquitectónico, que tiene un nombre específico y son parametrizadas para tener en cuenta diferentes situaciones durante el desarrollo de software.

Para el desarrollo del módulo se decide hacer uso del Patrón Arquitectónico Modelo - Vista – Controlador (MVC). Esta arquitectura es empleada en el sistema ULTRON y es la que sigue el framework de JavaScript Angular.JS.

Las principales ventajas del uso del patrón MVC son:

1. La separación del Modelo y la Vista, lo cual logra separar los datos, de su representación visual.
2. Facilita el manejo de errores.
3. Permite que el sistema sea escalable si es requerido.
4. Es posible agregar múltiples representaciones de los datos.

La arquitectura Modelo Vista Controlador (Model-View-Controller) es un patrón de diseño de software en torno a la interconexión de los tres tipos de componentes principales en un lenguaje de programación. A menudo con un fuerte enfoque en la programación orientada a objetos (POO). Estos tres tipos de componentes son llamados modelos, vistas y controladores. (Gómez, 2015)

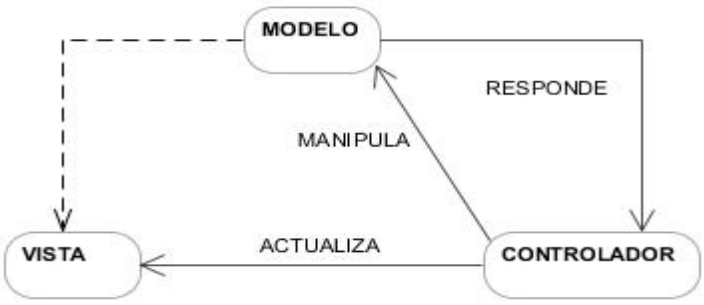


Fig. 1: Patrón arquitectónico. Fuente: (Suarez, y otros, 2018)

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 12, No.2, Mes Febrero, 2019
ISSN: 2306-2495 | RNPS: 2343
Pág. 1-15
http://publicaciones.uci.cu

Diagrama de clases del diseño

El diagrama de clases del diseño muestra los bloques de construcción de cualquier sistema orientado a objetos. Permite representar gráficamente y de manera estática la estructura general de un sistema, mostrando cada una de las clases y sus interacciones (como herencias y asociaciones), representadas en forma de bloques, los cuales son unidos mediante líneas y arcos. (Pointeau, 2018). A continuación, se representa el diagrama de clases del diseño correspondiente al módulo de diseño de mapas de impresión para la plataforma ULTRON.

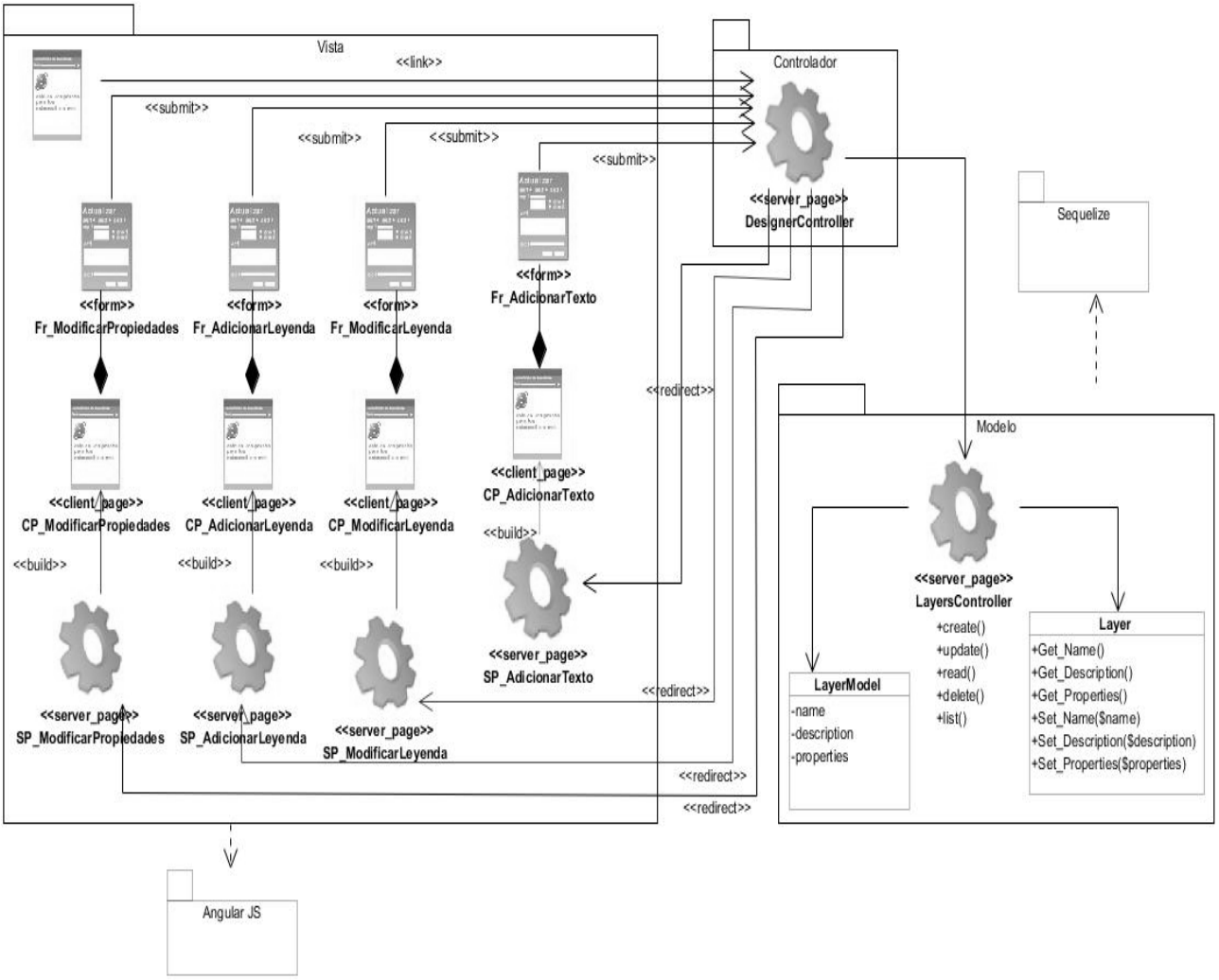


Fig. 2: Diagrama de clases del diseño.

Descripción de las clases del diseño.

Las clases contenidas en el **paquete vista** son las encargadas de recibir y procesar las peticiones del usuario. Estas tienen la responsabilidad de mostrar los formularios para realizar funcionalidades, tales como: adicionar etiquetas de texto, adicionar y modificar elementos de la leyenda, así como la edición de las propiedades del documento de impresión.

La clase **DesignerController** tiene la responsabilidad de interpretar todas las peticiones del usuario, redireccionarlas al modelo y enviar la respuesta a la vista. Esta es una clase servidora que se encarga de manejar el flujo de eventos e información del módulo.

La clase **LayersController** contiene todos los métodos que van a ser necesarios para crear o eliminar capas y modificar o listar sus atributos. La clase **Layer** es la encargada de contener toda la información referente a una capa. La clase **LayerModel** posee los atributos necesarios para brindar al controlador un modelo para administrar esta información.

Patrones de diseño de software empleados en el desarrollo del módulo.

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Se utilizan para crear propuestas de soluciones reusables y con documentación ya probada, partiendo de técnicas que se han aplicado en problemas similares. Estableciendo un lenguaje estándar entre todos los miembros de un equipo. (Sánchez, 2017)

Patrones para la asignación de responsabilidades (GRASP). Según (Torre, 2017) los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos. Son un método de enseñanza que ayuda a entender el Diseño de POO y aplica el análisis y diseño de objetos de un modo sistemático, racional y explicable. Para el desarrollo del módulo se hizo uso de los patrones siguientes:

El patrón **Experto (Expert)** es el encargado de asignar responsabilidades a las clases que tiene la información necesaria para cumplir dicha responsabilidad. Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. Hay que tener en cuenta que esto es aplicable mientras estemos considerando los mismos aspectos del sistema. La utilización de dicho patrón se ve aplicada en clases como: LayerModel y Layer, debido que cada una de ellas es responsable de manejar su información.

El patrón **Creador (Creator)** es quien guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Está aplicado en la clase LayersController, la cual utiliza la información de la clase LayerModel para crear objetos de ella.

El patrón **Controlador (Controller**) ofrece una guía para tomar decisiones apropiadas que generalmente se aceptan. La misma clase controlador debería utilizarse con todos los eventos sistémicos, de modo que se pueda conservar la información referente al estado del caso. El mismo se refleja en la clase DesignerController, la cual controla el flujo de eventos del módulo.

El patrón **Bajo Acoplamiento (Low Coupling)** estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento, tanto que produzca los resultados negativos propios de un alto acoplamiento. Soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecienten la oportunidad de una mayor productividad. Se ve empleado en todo el diseño del módulo ya que cada clase solo se relaciona con otras que necesite, para evitar que si se realiza algún cambio en algunas de ellas se vean afectadas las otras clases lo menos posible.

El patrón **Alta Cohesión (High Cohesion)** es el encargado de que la información que contiene una clase debe de ser coherente, altamente relacionada con dicha clase y que la misma no realice una excesiva cantidad de trabajo. Posee un número relativamente pequeño, con una importante funcionalidad relacionada y poco trabajo que hacer. Colabora con otros objetos para compartir el esfuerzo si la tarea es grande. Se refleja en las clases Layer y LayerModel ya que cada clase contiene la información y los métodos correspondientes exclusivamente a ellas, logrando que no se sobrecarguen con información de otras clases.

Patrones GOF.

Los patrones GoF describe soluciones simples y elegantes a problemas específicos en el diseño de software orientado a objetos. Permiten enfrentarse a la programación de software propiciando reutilización y extensibilidad de soluciones que han funcionado en el pasado (Parra, 2014). Se clasifican según el propósito para el que han sido definidos en:

- **Patrones creacionales:** Son aquellos que aplican soluciones para crear instancias de objetos de manera determinada. De este modo buscan encapsular la instanciación de objetos dentro de los métodos de otros objetos, ocultándola al usuario del código. (Oliveira, 2018)

- **Prototipo**: El patrón de prototipo se usa cuando la creación de objetos es costosa, requiere mucho tiempo y recursos y ya poseemos un objeto similar. Este patrón proporciona un mecanismo para copiar el objeto original a un nuevo objeto y luego modificarlo de acuerdo a nuestras necesidades. Es utilizado por la clase DesignerController al disponer del Árbol de Capas como un prototipo para la elaboración de la leyenda y permitir su posterior modificación en dependencia de las capas habilitadas.

- **Singleton**: El patrón de diseño 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. Se evidencia en la clase DesignerController, la cual define varias variables globales de instancia única que permiten la comunicación con el modelo y demás clases.

❖ **Patrones estructurales:** Abstraen la composición de interfaces a través de otros recursos, como la herencia o la agregación. Se encargan de proporcionar al usuario del código un objeto de firma simple que agrupa múltiples objetos

- **Proxy**: Es una clase que funciona como interfaz hacia cualquier otra cosa: una conexión a Internet, un archivo en disco o cualquier otro recurso que sea costoso o imposible de duplicar. Este patrón es empleado por la clase DesignerController para tener acceso a las imágenes del OSM[1] de la capa base del mapa.

- **Decorator**: Permite añadir funcionalidad extra a un objeto (de forma dinámica o estática) sin modificar el comportamiento del resto de objetos del mismo tipo. Se ve reflejado en las clases cliente de la vista al heredar el código que es común en todas las páginas del sistema, para no tener que repetirlo en cada una de estas.

❖ **Patrones de comportamiento:** Buscan proporcionar al usuario del código objetos que aporten una funcionalidad determinada**.** De este modo los objetos presentan una firma cuyos métodos resuelven un problema específico sin exponer su funcionamiento. (Leiva, 2017)

- **Observer**: Los objetos son capaces de suscribirse a una serie de eventos que otro objetivo va a emitir, y serán avisados cuando esto ocurra. Este patrón se utiliza en la clase LayersController para controlar las

[1] Open Street Map

llamadas sincrónicas a la base de datos; así como en las clases del paquete vista para realizar las acciones deseadas luego de concluir una interacción del mapa.

Conclusiones

Se logró la identificación de los requisitos funcionales y no funcionales del sistema dando paso a una mejor comprensión, por parte de la autora. Fueron definidas un total de 12 HU generadas a partir de la metodología seleccionada. La definición de la arquitectura y los patrones de diseño a utilizar, permitieron establecer las bases para fomentar la reutilización y las buenas prácticas de programación, así como disminuir el impacto de los cambios futuros en el código fuente. El diagrama de clases permitió identificar las relaciones entre las clases del sistema, además de brindar una visión más exacta del sistema en términos de implementación.

Referencias

Olaya, Víctor. 2014. Sistemas de Información Geográfica. 2014.

Pressman, Roger S. y Maxim, Bruce R. 2015. Software engineering: a practitioner's approach, Eighth Edition. New York : McGraw-Hill Education, 2015. ISBN 978-0-07-802212-8.

García Velázquez, Luis Ambrosio. 2016. RECI Revista Iberoamericana de las Ciencias Computacionales e Informática. [En línea] Julio de 2016. [Citado el: 07 de Marzo de 2018.] http://reci.org.mx/index.php/reci/article/view/49/221.

Sommerville, Ian. 2016. Software engineering: Tenth Edition. s.l. : Pearson Education, 2016. 978-0-13-394303-0.

Mediavilla, Elena. 2018. Jojooa.com. [En línea] 05 de Agosto de 2018. [Citado el: 24 de Enero de 2019.] http://jojooa.com/analisis-y-diseno-de-sistemas/actor-sistema/.

Cohn, Mike. 2018. User Stories and User Story Examples by Mike Cohn. [En línea] Mountain Goat Software, 26 de Febrero de 2018. [Citado el: 24 de Enero de 2019.] https://www.mountaingoatsoftware.com/agile/user-stories.

Cervantes, Humberto. 2018. SG.com. [En línea] 7 de Mayo de 2018. [Citado el: 5 de Febrero de 2019.] https://sg.com.mx/revista/27/arquitectura-software .

Huaman, Wilber Ccori. 2018. Medium.com. [En línea] 7 de Septiembre de 2018. [Citado el: 4 de Febrero de 2019.] https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b.

Gómez, Rodrigo. 2015. Modelo Vista Controlador. [En línea] 11 de Noviembre de 2015. [Citado el: 5 de Febrero de 2019.] http://rodrigogr.com/blog/modelo-vista-controlador/.

Pointeau, Arthur. 2018. Openclassrooms. [En línea] 30 de Marzo de 2018. https://openclassrooms.com/en/courses/4990961-planea-tu-proyecto-con-uml/4996676-diagrama-de-clase.

Sánchez, Miguel Ángel. 2017. Medium. [En línea] 22 de Noviembre de 2017. [Citado el: 12 de Febrero de 2019.] https://medium.com/all-you-need-is-clean-code/patrones-de-dise%C3%B1o-b7a99b8525e.

Torre, Fernando Alfonso Casas de la. 2017. Slideshare. [En línea] 24 de Mayo de 2017. [Citado el: 13 de Febrero de 2019.] https://es.slideshare.net/Indiana_1969/patrones-grasp-76283581.

Parra, Wilmer Eduardo Valdés. 2014. Integración de patrones de seguridad y patrones de diseño. Madrid : s.n., 2014.

Oliveira, Helder De. 2018. Linkedin. [En línea] 07 de Mayo de 2018. [Citado el: 14 de Febrero de 2019.] https://www.linkedin.com/pulse/patrones-de-dise%C3%B1o-creacionales-helder-de-oliveira.

Leiva, Antonio. 2017. Devexperto. [En línea] 6 de Marzo de 2017. [Citado el: 13 de Febrero de 2019.] https://devexperto.com/patrones-de-diseno-software/.