**Universidad de las Ciencias Informáticas**

**Facultad 6**


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

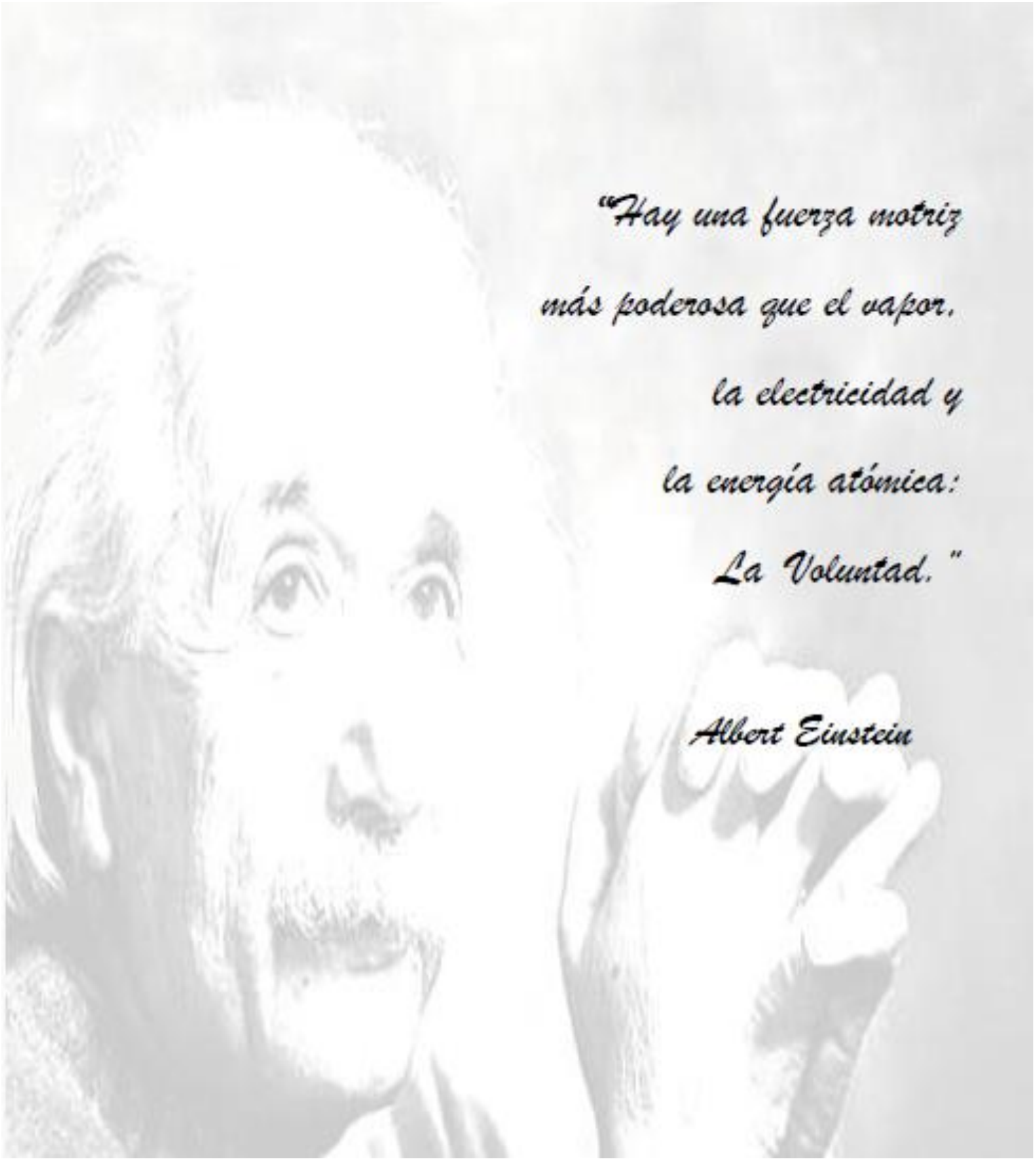
# Título: "Componente para la detección de errores topológicos en la plataforma GeneSIG sobre capas de tipo línea y polígono"

**Autor:**
Manuel de Jesús Jiménez Espinosa

**Tutor:**

Ing. Ariel Labrada Delgado

**La Habana, julio del 2016**
**"Año 58 de la Revolución"**



*"Hay una fuerza motriz*

*más poderosa que el vapor,*

*la electricidad y*

*la energía atómica:*

*La Voluntad."*

*Albert Einstein*

## Declaración de autoría

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

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

_____

Manuel de Jesús Jiménez Espinosa

Autor

_____

Ing**.** Ariel Labrada Delgado

Tutor

## Datos del contacto

**Autor:** Manuel de Jesús Jiménez Espinosa
**Correo Electrónico:** mjjimenez@estudiantes.uci.cu

**Tutor:** Ing. Ariel Labrada Delgado
**Formación Académica:** Ingeniero en Ciencias Informáticas (junio/2012).
**Centro Laboral:** Universidad de las Ciencias Informáticas (UCI).
**Correo Electrónico:** alabradad@uci.cu

## Dedicatoria

***Dedico esta tesis a mi mamá Ana, por todo su sacrificio dedicado a mi durante estos años.***

***A mi papá Jesús, que donde quiera que este sé que está muy orgulloso de mí, porque siempre lo estuvo desde día que ingrese en esta universidad.***

***A mis hermanos Annie, Michel y Mariam y a toda mi familia por todo el apoyo que me han dado.***

# Agradecimientos

***Agradezco a mi mamá por todo lo que me ha brindado en la vida, por desear este título más que yo y por confiar en mi a pesar de las dificultades.***

***A mis hermanos por ser estar pendientes a mi todo este tiempo y por todo su cariño brindado. A mis abuelos y a mis tíos por todo el apoyo que me dieron cuando más lo necesite y haber ocupado el lugar de mi papa.***

***A mi familia de aquí de la habana, Patricia, Alicia, Alejandra, Deysi y Ramírez por haberme acogido en sus casas durante estos 5 años.***

***A toda mi familia en general por todo su apoyo brindado para poder llegar a este momento.***

***A mi tutor Ariel que más que un tutor ha sido un amigo, te mereces el premio al mejor tutor, no solo por la ayuda brindada a mí, sino a todos los tesista de GeneSIG.***

***A los profesores del proyecto GeneSIG, como Pedro, Celia y otros que me ayudaron cuando tuve alguna duda.***

***Al tribunal y a mi oponente por todas las sugerencias brindadas, han sido de mucha ayuda y enseñanza.***

***A Laura por haber compartido dos años juntos y brindarme todo su apoyo.***

***A mis amigas más cercanas dentro de la universidad María, Teresita, Ilianet, Dayana, Jennifer, Kielam Daynelis, a Ivon que ha sido una hermanita durante estos cinco años, a Yeline por la buena amistad en estos últimos meses.***

***A todos mis amigos del apartamento del aula y de 5to año. A Darian y Eddy más que amigos han sido mis hermanos durante estos cinco años, hemos estado siempre en el mismo apartamento en la misma brigada creo que me llevo muchos recuerdos que nunca voy a olvidar todas las travesuras que hicimos, todas esas horas que dedicamos al estudio.***

***A los demás del piquete que estamos igual desde primer año Yosbanys, Joyce, Orlando, Lijandy, a los que se incorporaron luego Bermúdez, Marlon a Jony que no puede estar hoy presente, pero sé que estuviera como el primero si pudiera.***

***Al piquete del balonmano Eliober Nelson Asley, a los demás amigos que he hecho durante estos cinco años y muchas gracias a todos los presentes hoy aquí.***

## Resumen

Los sistemas de información geográfica trabajan con fuentes cartográficas. En ocasiones estas fuentes presentan errores topológicos, estos traen como consecuencia dificultades en el resultado final de las soluciones que se desarrollan afectando la calidad de los datos geométricos. La plataforma GeneSIG perteneciente al centro GEYSED de la facultad 6 se encarga del desarrollo de estos tipos de sistemas, en ella estos errores no son detectados. Por ello fue necesario crear un mecanismo para detectar los mismos, de manera que fuera posible corregirlos posteriormente. A partir de la necesidad planteada se determinó elaborar una solución al problema planteado.

Durante la investigación se trazó como objetivo desarrollar un componente para la detección de errores topológicos sobre capas de tipo línea y polígono. Este facilitó, mediante la implementación de varias reglas aplicadas a estas primitivas geométricas, favorecer la información cartográfica que se muestra en el resultado final de un proyecto.

El desarrollo del componente comprobador topológico fue guiado por la metodología Prodesoft. Se utilizó el lenguaje de modelado UML y la herramienta CASE Visual Paradigm para el diseño de los artefactos. La implementación se realizó con el lenguaje de programación php y NetBeans como entorno de desarrollo. Una vez concluido el componente se aplicaron pruebas de caja negra y caja blanca para validar funcionalmente la solución. Teniendo como resultado una aplicación que es capaz de detectar errores topológicos en la plataforma GeneSIG.

**Palabras clave:** capas, componente, errores topológicos, geometrías.

## Abstract

Geographic information systems work with cartographic sources. Sometimes these sources have topological errors, so these difficulties bring about some complications in the final result of the developed solutions affecting the quality of the geometric data. The center belongs to the faculty GEYSED platform number 6 and it is responsible of the development of these types of systems, but these errors are not detected yet. That's why it was necessary to create a mechanism to detect them, so it could be possible to correct them later. Taking into account this need We decide to give a solution to this problem.

During the investigation the objective is to develop a component for the detection of topological errors on type layers' line and polygon. This ease through the implementation of various rules applied to these geometric primitives, to promote cartographic information displayed on the outcome of a project.

The development of topological component tester was guided by the Prodesoft methodology. UML language modeling and CASE tool Visual Paradigm for the design of the devices were also used. The implementation was done with the php programming language and NetBeans development environment. Once the component was completed some tests to black box and white box were applied to validate the solution functionally. Consequently, now there is an application that is able to detect the topological errors in GeneSIG platform.


**Key words:** component, geometries, layers, topological errors.

# Tabla de Contenido

# Índice de figuras

# Índice de tablas

# Introducción

Los Sistemas de Información Geográfica (SIG) se definen como un conjunto que integra el hardware, el software y los datos geográficos, con el fin de resolver problemas complejos de planificación, análisis, manipulación, gestión, modelado, y exposición de datos espacialmente referenciados (NCGIA, 2008). Una de las características que poseen los SIG que los diferencian de otros sistemas de gestión de la información es la capacidad para editar y generar cartografía. La cartografía (del griego chartis = mapa y graphein = escrito) es la ciencia que se encarga del estudio y elaboración de los mapas geográficos, territoriales y de diferentes dimensiones lineales. También se denomina cartografía a un conjunto de documentos territoriales referidos a un ámbito concreto de estudio.

La cartografía constituye un conjunto de operaciones que permiten a partir de observaciones y mediciones, la representación de una parte o la totalidad de la Tierra (León, 2014). La representación de toda o parte de la superficie terrestre en una superficie plana constituye un mapa. La topología no es un concepto de moda dentro la cartografía, por el contrario, desempeña un papel fundamental en el trabajo con un SIG, por lo que resulta imprescindible considerarla y valorarla para el cumplimiento de las expectativas propuestas en los SIG.

De este modo, en la cartografía se entiende como topología a las relaciones espaciales entre los diferentes elementos (puntos, líneas y polígonos) que forman el mapa: si están cerca, si se tocan, si se superponen, etc. Estas relaciones, que para el ser humano pueden ser obvias a simple vista, no lo son tanto para el software informático, que las debe establecer mediante un lenguaje y reglas de geometría matemática definidas. La topología permite encontrar geometrías coincidentes o comunes tanto en entidades de puntos, líneas y polígonos, así como comprobar la integridad de la información, además permite la validación de las representaciones dentro de una geodatabase, muy útil en elementos hidrográficos, red vial, parcelas, entre otros (Team Quantum GIS Development, 2016).

Actualmente en Cuba se potencia el uso de SIG, evidenciándose en el desarrollo de la plataforma soberana para el desarrollo de sistemas de información geográfica (GeneSIG), la cual forma parte de un proyecto colaborativo entre el grupo Empresarial GeoCuba, la Empresa de Tecnología de Información para la Defensa (XETID) y la Universidad de las Ciencias Informáticas (UCI). Tiene como objetivo poner a disposición de usuarios y desarrolladores de soluciones geoespaciales, un sistema de tecnología libre, de código abierto y orientado a la Web, para solucionar demandas básicas de información espacial, en cuanto a su almacenamiento, análisis, y representación. Presenta una arquitectura distribuida, que cuenta con una

base cartográfica, y sobre ella un conjunto de objetos representados geoespacialmente, que contienen información asociada, ya que se identifican a partir de las necesidades particulares de los clientes finales.

Las instituciones que desarrollan soluciones SIG a partir de la plataforma GeneSIG utilizan fuentes cartográficas que contienen errores topológicos, estos, encontrados en capas de tipo polígono, línea y puntos son muy comunes. Los mismos pueden darse de forma involuntaria y son muy frecuentes encontrarlos como: superposiciones, bucles, nodos colgantes, entre otros. Estos traen como consecuencia dificultades a la hora de realizar análisis en fuentes cartográficas dependiendo del uso que tengan, ejemplo en el cálculo de rutas óptimas en aplicaciones de enrutamiento, medición de superficies, planificación y ordenamiento urbano. En áreas, como la agricultura, se hace factible utilizar la cartografía para determinar aspectos tales como uso del suelo, tipos de suelo, pertenencia de la tierra, entre otros. Por estos aspectos citados se hace necesario su comprobación.

Aun cuando la plataforma cuenta con varios años de uso estos errores no son detectados, por lo que deben ser comprobados con otras herramientas para su posterior corrección, afectando el trabajo de los usuarios finales y el equipo de trabajo. Desde el punto de vista espacial se dificulta la implementación de condiciones y restricciones. La calidad de los datos geométricos se ve afectada ya que no existen de antemano condiciones que permitan su comprobación. Los errores de datos existentes y los errores introducidos por el usuario son difíciles de encontrar a simple vista. Existe ausencia de mecanismos que describan las relaciones entre puntos, líneas y polígonos que representan los objetos espaciales de una región geográfica.

Dada la problemática expuesta anteriormente surge como **problema a resolver** ¿Cómo contribuir a mejorar la calidad de la cartografía en la plataforma GeneSIG?

Para darle solución al problema se tiene como **objetivo general:** Desarrollar un componente para la detección de errores topológicos, que permita contribuir con la calidad de la cartografía en la plataforma GeneSIG.

El **objeto de estudio** de la investigación se define como: análisis topológico en los SIG, enmarcado en el **campo de acción** detección de errores topológicos en la plataforma GeneSIG.

Para dar cumplimiento a los objetivos se plantean las siguientes **tareas de investigación:**

1. Elaborar el marco teórico de la investigación asociado al análisis de la topología en los SIG.
2. Realizar el estado del arte de los sistemas homólogos a la solución.
3. Caracterizar las herramientas, metodología y tecnologías que se utilizan en el desarrollo del componente.

4. Elaborar la lista de requisitos del componente.
5. Diseñar la propuesta de solución basados en los artefactos de la metodología Prodesoft.
6. Implementar la propuesta de solución.
7. Validar la solución mediante pruebas de caja negra y caja blanca.

Como **posibles resultados** de la investigación se esperan:

1. Un componente que permita la detección de errores topológicos en capas de tipo línea y polígono en la plataforma GeneSIG.
2. La documentación asociada al desarrollo del componente.

Los **métodos científicos** que se utilizan en el desarrollo de la investigación son los siguientes:

**Métodos teóricos**

- ✓ **Analítico-sintético**: permitió la extracción de los elementos más importantes del estudio y análisis de diversas fuentes bibliográficas que se relacionan con el desarrollo del componente para la detección de errores topológicos integrado a la plataforma GeneSIG.
- ✓ **Modelación:** se utilizó en la modelación de los diagramas correspondientes al diseño del componente.

**Métodos empíricos**

- ✓ **Análisis documental:** permitió el análisis documental para obtener los datos necesarios de la bibliografía consultada, para el desarrollo del componente comprobador topológico.

El presente trabajo está estructurado en tres capítulos:

**Capítulo 1. Fundamentación teórica sobre la detección de errores topológicos**

Acumula aspectos concernientes al objeto de estudio, son expuestos los principales conceptos asociados a la topología. Se expone una síntesis de algunas soluciones existentes para el problema identificado. Se realiza una breve descripción de las herramientas definidas para la construcción de la solución propuesta.

**Capítulo 2. Características y diseño del componente comprobador topológico**

Se modela el dominio, se realiza la especificación de los requisitos de software. Se describe la arquitectura, los patrones arquitectónicos y de diseño a emplear. Además, se lleva a cabo la creación de los artefactos ingenieriles fundamentales relacionados con el diseño.

**Capítulo 3. Implementación y prueba del componente comprobador topológico**

En este capítulo se incluyen los diagramas de despliegue y de componentes, el flujo de trabajo de pruebas con la descripción de los casos de prueba y la validación de la solución propuesta.

# Capítulo I. Fundamentación teórica sobre la detección de errores topológicos.

## 1.1 Introducción

Con el objetivo de facilitar la comprensión del alcance de la investigación, en el presente capítulo se abordan los aspectos teóricos fundamentales para el desarrollo de la solución. Se hace mención de las relaciones topológicas más importantes y se explican las reglas topológicas más utilizadas para la detección de errores topológicos. Se analizan diversos sistemas informáticos que pueden servir de referencia o punto de partida de la presente investigación y se describe la metodología, las tecnologías y herramientas de software definidas por la línea base del proyecto GeneSIG.

## 1.2 Sistemas de Información Geográfica

Un SIG abarca tecnología de la información, gestión de la información, asuntos legales y de negocios, y conceptos específicos de materias de un gran abanico de disciplinas, pero es implícito la idea de SIG como una tecnología usada para tomar decisiones en la solución de problemas que tengan al menos una parte de componente espacial (Miller, 2012).

Se puede definir que un SIG es la unión del software, hardware y datos geográficos, que permite a los usuarios manipular, analizar, integrar y desplegar información geográficamente referenciada, asociada a un territorio.

Las características principales de un SIG según (Miller, 2012) son:

- La capacidad de visualización de información geográfica compleja a través de mapas.
- La funcionalidad de los SIG como una base de datos sofisticada, en la que se mantiene y relaciona información espacial y temática.
- La diferencia con las bases de datos convencionales estriba en que toda la información contenida en un SIG está unida a entidades geográficamente localizadas. Por ello en un SIG la posición de las entidades constituye el eje del almacenamiento, recuperación y análisis de los datos.
- Son una tecnología de integración de información.
- Se han desarrollado a partir de innovaciones tecnológicas ocurridas en campos especializados, de la geografía y otras ciencias (tratamiento de imágenes, análisis fotogramétricos, cartografía automática, etc.), para constituir un sistema único.
- Permiten unificar la información en estructuras coherentes.

- Este carácter integrador y abierto, hace de los SIG un área de contacto entre variados tipos de aplicaciones informáticas, destinadas al manejo de información con propósitos y formas diversas; por ejemplo: programas estadísticos, gestores de bases de datos, programas gráficos, hojas de cálculo, procesadores de texto, etc.

## 1.3 Objeto Geográfico

La representación primaria de los datos en un SIG está basada en algunos tipos de objetos geográficos que se refieren al punto, línea y polígono. Los objetos geográficos son abstracciones de elementos del mundo real que están asociadas a una posición geográfica y temporal definida, respectivamente, en un sistema de referencia espacial y temporal. Estos objetos presentes en el mundo real tienen dos tipos básicos de abstracción: **ocurrencias** y **tipos** (ICDE, 2013).

**Ocurrencia o instancia**

Es cualquier objeto que es diferenciado de los demás teniendo en cuenta sus características propias, es decir, se particulariza o se vuelve único. Por ejemplo: Río Magdalena, Río Amazonas, Río Orinoco, etc. Se puede identificar cada objeto de manera inequívoca y con una ubicación espacial específica.

**Tipo**

Cuando un conjunto de elementos posee características similares y estos son abstraídos como una clase de elemento, se les llama tipos. Esta forma de abstracción es comúnmente utilizada para la definición de catálogos. Deben ser documentados de tal manera que no dependan del producto o escala.

**Características de un objeto geográfico**

- Ubicación absoluta en el espacio:
  Su carácter geográfico exige que siempre estén relacionados a un sistema de coordenadas referido a la superficie terrestre. Por tanto, siempre tendrán una posición absoluta al origen de coordenadas para su ubicación.
- Ubicación relativa en el espacio:
  Los objetos también pueden ser ubicados con base en la posición de otros objetos, de manera que tanto la distancia como la orientación sean fácilmente precisadas por el carácter geográfico de otro objeto.

## 1.4 Topología

La topología ha sido durante mucho tiempo un requisito clave en los SIG para la administración y la integridad de los objetos geográficos. Es una rama de las matemáticas que estudia las propiedades cualitativas intrínsecas de las configuraciones espaciales que son independientes de la forma y el tamaño.

Desde la topología se conserva: la contigüidad, contenencia, no contenencia, conectividad y coincidencia espacial, las cuales son denominadas en los SIG como invariantes topológicas; aquellas que no se conservan como el área, perímetro, forma y distancias reciben el nombre de variantes topológicas. En este orden de ideas, es la topología implícita o de mapa la que permite construir las relaciones espaciales de los objetos geográficos almacenados en un SIG, de modo tal que permite validar las invariantes topológicas garantizando la integridad (considere como continuidad en el espacio) de la información geográfica que es almacenada en una base de datos espacial. Esta información, con base en lo anterior, es almacenada en tablas cuyos nombres son: arco-nodo, arco-polígono y contigüidad (Vargas, 2015).

Las relaciones topológicas más importantes según (Alonso, 2015) son:

- Adyacencia (entre polígonos).
- Contigüidad (entre línea y polígono).
- Pertenencia (arcos a polígonos).
- Conectividad (entre arcos, en redes).
- Inclusión (punto en polígono, línea en polígono, polígono en polígono).

## 1.5 Análisis topológico

El análisis de los datos geográficos ha cobrado una nueva dimensión desde la aparición de los SIG, surgiendo nuevos planteamientos y mejorándose los ya existentes. A lo largo de toda su historia, el análisis ha sido uno de los elementos más importantes de un SIG, y al día de hoy existen formulaciones que cubren casi todo el abanico posible de necesidades.

En los SIG se recurre al uso del análisis topológico para definir las relaciones espaciales de los elementos geográficos. Las consultas hechas a las capas de datos espaciales pueden tener relación no solo con su posición sino con la relación de otros elementos de la misma capa. La existencia de topología puede emplearse para la realización de consultas que respondan a cuestiones como, entre otras, las siguientes (Olaya, 2011):

- ¿Cómo llegar desde mi posición actual hasta una coordenada concreta por la red viaria existente?

- ¿Qué comunidades autónomas comparten límite con una Provincia?

### *1.5.1 Reglas topológicas*

Las reglas que se definen para una topología controlan las relaciones entre las entidades de una clase de entidad, entre las entidades de diferentes clases de entidad o entre los subtipos de las entidades. También se pueden definir reglas topológicas entre los subtipos de las clases de entidad.

Por ejemplo, supongamos que se tiene dos subtipos de entidades de línea de calle: calles normales (aquellas que se conectan con otras calles en ambos nodos) y calles sin salida (aquellas que no tienen salida en un nodo). Una regla topológica puede exigir que las entidades de calle estén conectadas con otras entidades de calle en ambos extremos, excepto en el caso de calles que pertenecen al subtipo de calle sin salida (Inc Research Institute ArcGIS, 2016).

Existen diferentes tipos de reglas para las primitivas geométricas, a continuación, se explican una algunas de ellas:

**Reglas topológicas para capas de tipo punto**

**Debe coincidir con:** Requiere que los puntos en una clase (o subtipo) de entidad coincidan con los puntos de otra clase (o subtipo) de entidad. Esto es útil para los casos en los cuales los puntos deben estar cubiertos por otros puntos.

**Debe estar separado:** Requiere que los puntos se encuentren separados espacialmente de otros puntos en la misma clase (o subtipo) de entidad. Los puntos que se superpongan son errores. Esto resulta útil para asegurarse de que los puntos no coincidan ni se dupliquen dentro de la misma clase de entidad, tal como en capas de ciudades, pozos o postes de luz.

**Debe estar cubierto por el límite de:** Requiere que los puntos se encuentren en los límites de las entidades de área. Esto resulta ventajoso cuando las entidades de punto facilitan un sistema de límites, tal como marcadores de límites, los que deben encontrarse en los bordes de determinadas áreas.

**Debe estar incluida correctamente:** Requiere que los puntos se encuentren dentro de las entidades de área. Esto resulta útil cuando las entidades de punto están relacionadas con polígonos, tales como pozos y rellenos de pozos o puntos de dirección y parcelas.

**Debe estar cubierto por el extremo de:** Requiere que los puntos en una clase de entidad deben cubrirse con los puntos finales de líneas en otra clase de entidad.

**El punto debe estar cubierto por la línea:** Requiere que los puntos en una clase de entidad deben cubrirse con las líneas en otra clase de entidad. No contiene la parte de cobertura de la línea para ser un punto final. Esta regla resulta conveniente para puntos que se encuentran a lo largo de un conjunto de líneas, tales como carteles de carreteras.

Las siguientes reglas correspondiente a capas de tipo polígono y línea, estarán presentes en la solución.

**Reglas topológicas para capas de tipo polígono**

**No se deben superponer:** Los polígonos adyacentes no deben de compartir un área en común.

**No deben superponerse con:** Los polígonos adyacentes de una capa no deben compartir un área con los polígonos de otra capa.

**Contiene un punto:** Requiere que cada polígono contenga una entidad de puntos y que cada entidad de puntos se encuentre dentro de un único polígono. Esto se utiliza cuando debe haber una correspondencia uno a uno entre las entidades de una clase de entidad poligonal y las entidades de una clase de entidad de puntos, como los límites administrativos y sus capitales. Cada punto debe estar perfectamente dentro de un polígono y cada polígono debe contener exactamente un punto. Los puntos deben encontrarse en el interior del polígono, no en el límite.

**Contiene anillos:** Esta regla precisa que no haya vacíos dentro de un polígono simple o entre polígonos adyacentes. Todos los polígonos deben formar una superficie continua. Siempre existirá un error en el perímetro de la superficie. Se puede ignorar este error o agregarse como una excepción.

**Reglas topológicas para capas de tipo líneas**

**Los puntos finales deben estar cubiertos:** Requiere que los extremos de las entidades de línea estén cubiertos por entidades de puntos en otra clase de entidad. Esto es útil para modelar los casos en los que se debe conectar un ajuste con dos canalizaciones o que se debe encontrar un cruce de calle en la unión de dos calles.

**No debe tener extremos sueltos:** Requiere que una entidad de línea deba tocar las líneas desde la misma clase (o subtipo) de entidad en ambos extremos. Un extremo que no esté conectado con otra línea se llama nodo colgado *(dangle)*. Esta regla se utiliza cuando las entidades de línea deben formar bucles cerrados, como cuando definen los límites de las entidades poligonales. También se podría utilizar en los casos en los que las líneas se conectan generalmente con otras líneas, como con calles. En este caso, las excepciones se pueden utilizar allí donde la regla se viola ocasionalmente, como con una calle sin salida.

**No debe tener pseudos:** Un punto final de geometría de línea debe estar conectado a los extremos de otras geometrías. Si el punto final está conectado al punto final de otra geometría, el punto final se denomina un nodo pseudo.

A continuación, se mencionan las reglas que son comunes para estas capas:

**No debe tener geometrías multi-parte:** A veces, una geometría es en realidad una colección geometrías sencillas (parte sencilla). Si contiene sólo un tipo de geometría simple, lo llamamos multi-punto, multi-líneas o multi-polígono. Por ejemplo, un país que consta de múltiples islas se puede representar como un multi-polígono.

**No debe tener duplicados:** Requiere que una geometría no este repetida dentro de la capa.

**No debe tener geometrías no validas:** Comprobar si las geometrías son válidas.

### *1.5.2 Errores de topológicos*

La calidad de los objetos geográficos es hoy día un aspecto clave para el trabajo con SIG. Crear y tener datos topológicos correctos no solo es importante para el propio análisis en un SIG, sino también para los usuarios que reciben esos datos. Los sistemas que realizan análisis topológico implementan un conjunto de reglas topológicas para detectar errores topológicos y así contribuir a la calidad de sus datos espaciales.

En los SIG se pueden encontrar distintos tipos de errores topológicos, pueden agruparse dependiendo del tipo de capa vectorial (polígonos, líneas o puntos). Los errores topológicos rompen la relación entre los elementos, durante los procesos de digitalización pueden surgir errores involuntarios muy comunes (nodos colgantes, bucles, superposiciones, etc.). Es necesario solucionar estos errores para poder analizar los datos vectoriales con procedimientos como el análisis de red (por ejemplo, encontrar la mejor ruta por una red de carreteras o descubrir la longitud de un río (Affairs, 2009).

Es importante ser consciente de los errores que contienen los datos y de la posible aparición de estos a medida que se realizan tareas con ellos, con el objetivo de minimizar dicha aparición y limitar la presencia e influencia de los errores en los resultados finales de las soluciones SIG.

Un error muy común en las capas de tipo línea, es cuando presentan alguno de sus extremos sueltos, ejemplo cuando una línea no conecta en su extremo con otra línea. Encontrar estos tipos de errores es importante cuando las líneas son una parte de una red (por ejemplo, las líneas que representan los ríos se necesitan conectar). La siguiente figura muestra este tipo de error (1).


**Figura 1:** Error de línea con extremo suelto *(Team Quantum GIS Development, 2016)*.

Por otra parte, un error muy común en los polígonos, es la superposición de los límites de los polígonos. Reconocer estos tipos de errores es importante, (por ejemplo, resulta útil para modelar límites administrativos, como códigos postales o distritos electorales, y clasificaciones de área mutuamente exclusivas, como cobertura de suelo o tipo de forma de suelo). La siguiente figura muestra este tipo de error.

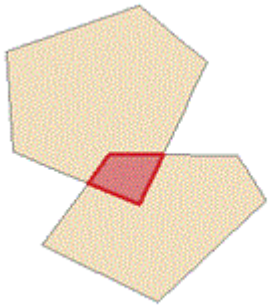


**Figura 2:** Error de polígono de tipo superposición *(Inc Research Institute ArcGIS, 2016).*

### *1.5.3 Funciones de PostGIS*

Para la implementación de reglas topológicas es conveniente utilizar funciones de PostGIS. PostGIS es una extensión que convierte el sistema de base de datos PostgreSQL en una base de datos espacial. (Arévalo, 2012). Presenta un conjunto de funciones estrechamente relacionadas con relaciones topológicas. Dichas funciones por si solas o en conjunto pueden dar solución una regla topológica. Se enfocan en los siguientes aspectos:

- ✓ Conversión: Funciones que convierten las geometrías a otros formatos externos.
- ✓ Recuperación: Obtienen propiedades y medidas de las geometrías.
- ✓ Comparación: Comparan dos geometrías y obtienen información sobre su relación espacial.

A continuación, se mencionan algunas de las funciones más utilizadas y se brinda una breve descripción (Ramsey, 2015).

- ✓ **ST_IsClosed:** Devuelve verdadero si el punto final es igual punto inicial de una línea.
- ✓ **ST_Equals:** Devuelve verdadero si dos geometrías son iguales.
- ✓ **ST_Contains:** Devuelve verdadero si la geometría A está completamente dentro de la geometría B.
- ✓ **ST_Overlap:** Devuelve verdadero si las geometrías se superponen, esto quiere decir que se cruzan, pero una geometría no contiene a la otra completamente.
- ✓ **ST_isValid:** Prueba si el valor de una geometría está bien formado.
- ✓ **ST_Crosses:** Toma dos objetos de geometría y devuelve verdadero si las geometrías tienen puntos interiores en común, pero no, en toda su totalidad.
- ✓ **ST_EndPoint:** Devuelve el último punto de una geometría lineal.
- ✓ **ST_Intersects:** Devuelve verdadero si las geometrías espacialmente se cruzan y comparten cualquier porción de espacio, devuelve falso si no lo hacen.

## 1.6 Análisis de soluciones existentes

En el mundo actual existen distintos SIG que implementan soluciones para la detección de errores topológicos, a continuación, se presentan las características más relevantes de algunos de ellos:

### *1.6.1 QGIS*

Anteriormente conocido como Quantum GIS el mismo es un SIG de código libre multiplataforma. Es un sistema desarrollado en *QT Toolkit* y C++. Se publica bajo Licencia Pública General (GNU *General Public License*). Ofrece muchas características SIG comunes, proporcionadas por las funciones del núcleo y los complementos. Permite manejar formatos raster y vectoriales a través de las bibliotecas GDAL y OGR, así como bases de datos.

Es capaz de diseñar mapas y explorar datos espaciales de forma interactiva con una interfaz amigable. Permite además crear, editar, administrar y exportar mapas vectoriales en varios formatos.

Presenta un componente para comprobar topología, que permite revisar las capas vectoriales y verificar la topología con varias reglas topológicas. Estas reglas comprueban mediante relaciones espaciales si un objeto espacial es *'Equal', 'Contain', 'Cover', 'CoveredBy', 'Cross'*, o son *'Disjoint', 'Intersect', 'Overlap', 'Touch'* o *'Within'* el uno al otro (Team Quantum GIS Development, 2016).

### *1.6.2 ArcGIS*

ArcGIS Desktop es el nombre que recibe una colección de productos de software en la esfera de los SIG. Elaborado y comercializado por ESRI (*Enviromental Systems Research Institute*) una empresa que, aunque tiene ramificaciones en muchos países, radica en California; *ArcGIS Desktop* está compuesto por varias

aplicaciones para la captura, edición, análisis, tratamiento, diseño, publicación e impresión de información geográfica. El *ArcGis Desktop* contiene un grupo de aplicaciones SIG integradas, en las cuales se encuentran ArcMap, ArcToolbox, ArcCatalog, ArcScene y ArcGlobe. Se utiliza para crear, importar, revisar, preguntar, trazar, analizar, y publicar datos geográficos.

Todas las aplicaciones de *ArcGIS Desktop* tienen la misma arquitectura con el objetivo de que cualquier usuario que use uno de estos SIG *Desktop* tenga la posibilidad de compartir su trabajo con otros usuarios. Los usuarios pueden compartir los mapas, la simbología, las capas, las herramientas personalizadas, las interfaces, los informes, evitando así la necesidad de desplegar diversos productos desiguales.

Para acceder al *ArcGIS Desktop* se utilizan cuatro productos de software y cada uno de ellos suministra un nivel superior de funcionalidad. Dichas aplicaciones se encierran en grupos temáticos como ArcGIS Server, para la publicación y gestión Web, o ArcGIS Móvil para la captura y gestión de información. ArcGIS presenta dentro de sus componentes un editor para la topología, con el mismo se detectan errores encontrados en las capas analizadas para darle su debida solución (Inc Research Institute ArcGIS, 2016).

### *1.6.3 gvSIG*

Es una herramienta SIG sobre software libre que surge por el año 2002 a partir de la necesidad de la Consellería de Infraestructuras y Transporte (CIT, Valencia España) de migrar toda la informática de su organización a software libre.

La herramienta gvSIG está orientada al manejo de información geográfica. Su interfaz es amigable y de fácil uso, es capaz de acceder a los formatos raster y vectoriales de forma rápida. Integra en una vista datos tanto locales como remotos a través de un origen WMS (*Web Map Service*), WCS (*Web Coverage Service*) o WFS (*Web Feature Service*). Cuando se construyó se tuvo en cuenta la extensibilidad del proyecto para permitir la incorporación de nuevas funcionalidades a la aplicación, así como desarrollar aplicaciones totalmente nuevas a partir de las bibliotecas utilizadas en gvSIG.

Vale destacar que gvSIG es uno de los SIG libres más completos de la actualidad por el amplio conjunto de funcionalidades que contiene y además la posibilidad de poder agregar más funciones a las ya existentes. Este framework está desarrollado sobre JAVA. Permite cargar datos locales y remotos, entre los formatos que usa están: .SHP, .DXF, DWG, y DGN todos estos de tipo vectorial y, ECW, el MrSID, el GeoTIFF y otros de tipo raster, gvSIG posibilita cargar datos remotos a través de un origen WMS, WCS o WFS, y también de bases de datos espaciales como PostGIS y MySQL. Al igual que las soluciones anteriores gvSIG presenta un componente para detectar errores topológicos en las soluciones que realiza, utilizando reglas para la detección de estos y de esta forma alcanzar mayor calidad en el desarrollo de sus soluciones (Grupo gvSIG, 2016).

Después de realizar un estudio de las soluciones más cercanas a la solución propuesta se decide desarrollar un componente para la detección de errores topológicos en la plataforma GeneSIG y no hacer uso de estas herramientas debido a limitaciones que se precisan a continuación:

- ✓ ArcGIS es una solución privativa por lo que se tendría que pagar por una licencia para su uso.
- ✓ La utilización de QGIS y gvSIG no sería factible. La integración de alguna de estas herramientas con la plataforma GeneSIG crea una dependencia innecesaria entre ambos sistemas, de manera que se tendría que instalar ambas plataformas para la comunicación simultánea entre ellas a través de servicios.

Las soluciones estudiadas proporcionan elementos que son necesarios para la implementación del componente:

- ✓ Se realiza una selección de las reglas topológicas más utilizadas por estas herramientas.
- ✓ Los errores que se detectan en estos sistemas son visualizados en el mapa y se detalla una descripción de los mismos.
- ✓ Se verifican los errores para una extensión determinada del mapa y para toda la extensión de la capa seleccionada.
- ✓ Se aplica una tolerancia específica a una regla para detectar errores con mayor exactitud.

A diferencia de estos sistemas en la solución que se propone se facilita la visualización de los errores topológicos correspondientes a cada regla, aplicando un color aleatorio a cada geometría. Además, es posible mostrar y ocultar cada geometría independientemente del resto.

## 1.7 Tendencias y tecnologías

Para el desarrollo del sistema se usaron las herramientas y tecnologías definidas por la plataforma GeneSIG, para lograr una fácil integración con la misma.

### *1.7.1 Metodología de desarrollo*

“Una metodología de desarrollo de software, es aquella que hace posible la planificación, organización y construcción de un sistema o proyecto, con independencia de su temática o complejidad. Actualmente estas metodologías son una guía en el proceso de desarrollo de las aplicaciones informáticas, permitiendo que se obtengan resultados con la mayor calidad, rapidez y eficiencia posible, para evitar cometer errores futuros” (Pressman, 2005).

En correspondencia con el proyecto “GeneSIG” se define Prodesoft (Proceso de Desarrollo de Software) como metodología de desarrollo de software a emplear en la solución que se propone. Prodesoft se basa

en una combinación de varios modelos de procesos: el basado en componentes, el iterativo y el incremental. También, es una combinación entre ágil y robusta.

Este proceso se encuentra disponible en su versión 1.5 y según su especificación (UCID, 2012), plantea que el ciclo de vida de un proyecto está compuesto por 5 fases: inicio, modelación, construcción, explotación experimental y despliegue (ver Figura 3).

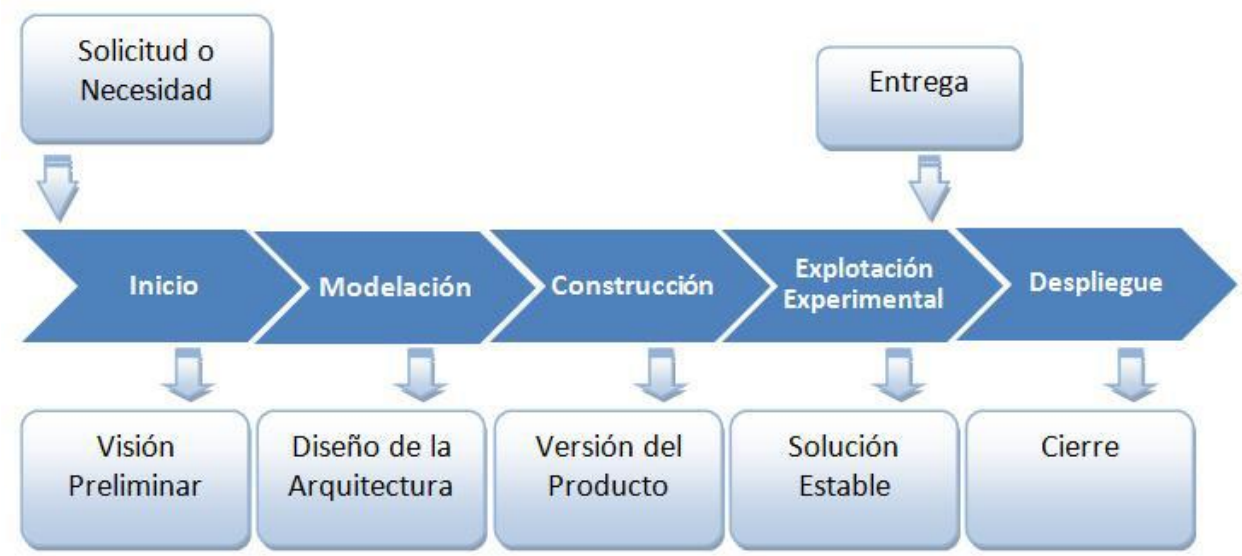



**Figura 3:** Ciclo de vida de un proyecto, metodología Prodesoft *(UCID, 2012)*.

Durante la fase de **Inicio** se logra una visión preliminar de la problemática a resolver y se definen los recursos relevantes para la ejecución del proyecto. Es decir, se describen los objetivos y el alcance del proyecto, se identifican los involucrados y ejecutores (entidades involucradas).

Se estima de manera general las actividades a realizar durante todo el ciclo de desarrollo del proyecto (Cronograma General), se establece la estrategia a seguir para realizar la modelación del negocio y la captura de requisitos y de ser necesario se estiman los recursos materiales que deberán ser adquiridos.

En la fase de **Modelación** se capturan las partes esenciales del sistema, donde se identifican los procesos de negocio fundamentales y se aceptan los requisitos funcionales, obteniéndose la línea base de la arquitectura y una estrategia de construcción de la aplicación aprobada por los implicados en el proyecto. El hito fundamental de esta fase es la liberación de la arquitectura de sistema, datos y despliegue.

En la fase de **Construcción** se aclaran los requisitos restantes y se completa el desarrollo del sistema sobre una base estable de la arquitectura. Las fases anteriores sólo dieron una arquitectura básica que es aquí refinada de manera incremental conforme se construye el producto. En esta fase todas las características,

componentes, y requisitos deben ser integrados, implementados, y probados en su totalidad, obteniendo una versión liberada del producto.

Durante la fase de **Explotación Experimental** se convierte la versión liberada del producto en una solución estable, donde se eliminan los errores que surgen durante las pruebas y se obtiene una certificación funcional y de seguridad del producto.

En la fase de **Despliegue** se instala y configura el sistema para un ambiente de producción real, se capacita al personal que usará la aplicación y se continúa dando soporte durante la explotación del sistema, culminando de ser preciso con transferencias tecnológicas.

### *1.7.2 Lenguaje Unificado de Modelado (UML) 2.0*

UML surge como respuesta al problema de contar con un lenguaje estándar para el modelado de software. Está basado en una notación gráfica la cual permite: especificar, construir, visualizar y documentar los objetos de un sistema programado (EVA).

**Características de UML:**

1. Es un lenguaje consolidado, fácil de aprender y permite la documentación de todo el ciclo de desarrollo del sistema.
2. Puede ser usado en todas las etapas de desarrollo del sistema y su representación gráfica puede ser usada para comunicarse con los usuarios o entre el equipo de desarrollo.
3. Se ha venido adoptando en diferentes medios empresariales y académicos como el lenguaje estándar para el análisis y diseño de los sistemas de software.

### *1.7.3 Visual Paradigm 8.0*

Esta herramienta acelera el desarrollo de aplicaciones, ya que sirve de puente visual entre arquitectos, analistas y diseñadores de sistemas de información, haciendo el trabajo más fácil y dinámico. La misma es utilizada hoy día para el modelado de los diagramas UML. Visual Paradigm es una herramienta *CASE* para el modelado UML muy potente, gratuita, fácil de instalar, utilizar y actualizar. Permite dibujar todo tipo de diagramas UML, revertir código fuente a modelos UML, generar código fuente desde los diagramas UML (Cuervo, 2005).

### *1.7.4 Entorno de Desarrollo Integrado (IDE) NetBeans 8.0*

NetBeans es un entorno de desarrollo, es una herramienta de código abierto bajo la licencia pública (GPL). Permite el uso de un amplio rango de tecnologías de desarrollo tanto para escritorio, como aplicaciones Web, o para dispositivos móviles. Da soporte a varias tecnologías como Java, PHP, Groovy, C/C++, HTML5. Entre sus funcionalidades permite escribir, depurar, compilar y ejecutar programas. Además, se destacan

sus ventajas de auto completamiento de código, interfaz para el diseño de GUI y la capacidad para importar clases (NetBeans, 2016).

### *1.7.5 Servidor de mapas MapServer 6.4*

Los servidores de mapas proveen un alto manejo de la información geográfica. Por una parte, el cliente tiene acceso a los datos en su conformación original, de forma tal que es capaz de realizar consultas tan complicadas como las que haría un SIG.

Un servidor de mapas actúa enviando desde un navegador, una sucesión de páginas HTML, con una cartografía relacionada entre sí en formato de imagen (puede ser, una imagen GIF o JPG). Las versiones iniciales de servidores de mapas sólo admitían efectuar funciones elementales de visualización y consultas alfanuméricas simples. En las versiones modernas es posible ejecutar funciones mucho más avanzadas (Kropla, 2005).

### *1.7.6 Servidor de aplicaciones Apache 2.2*

Algunas características importantes del servidor Apache son: (Novell, 2011)

- Puede ser adaptado a diferentes entornos y necesidades, con los diferentes módulos de apoyo que proporciona, y con el API de programación de módulos, para el desarrollo de módulos específicos.
- Incentiva la realimentación de los usuarios, obteniendo nuevas ideas, informes de fallos y parches para la solución de los mismos.
- Se desarrolla de forma abierta.
- Gracias a ser modular, se han desarrollado diversas extensiones entre las que destaca PHP, un lenguaje de programación del lado del servidor.

Las funcionalidades más elementales se encuentran en el módulo base, siendo necesario un módulo multiproceso para manejar las peticiones. Se han diseñado varios módulos multiproceso para cada uno de los sistemas operativos sobre los que se ejecuta el Apache, optimizando el rendimiento y la rapidez del código (Clifton, 2007).

### *1.7.7 Gestor de bases de datos PostgreSQL 9.4 (PostGIS 2.0)*

PostgreSQL, llamado Sistema Gestor de Bases de Datos Relacionales Orientadas a Objetos, es usado para manejar grandes cantidades de información, y está basado en el modelo relacional, aunque incorpora conceptos del modelado orientado a objetos. Es distribuido bajo licencia BSD y con su código fuente disponible libremente, es multiplataforma, soporta múltiples transacciones, integridad de datos, presenta una estabilidad muy alta, gran seguridad de los datos, potencia, robustez, facilidad de administración e implementación de estándares estas son algunas de sus características principales (Martinez, 2013).

**PostGIS:** es una extensión de PostgreSQL que le añade soporte de objetos geográficos, debido a que está construido sobre PostgreSQL. PostGIS hereda automáticamente sus características, así como los estándares abiertos. Incluye también funciones que calculan relaciones espaciales entre diferentes objetos geográficos, como la intersección, distancia, superposición, etc. Mediante estas sentencias SQL adaptadas se podría, por tanto, definir reglas topológicas (Cantero, 2008).

### *1.7.8 PHP 5.3*

PHP es el acrónimo de *Hypertext Prepocessor.* Este lenguaje de programación, se creó para la generación de páginas *Web*, este lenguaje es libre y gratuito, es decir, está amparado bajo el movimiento código abierto (*open source*), tiene extensiones para soportar múltiples bases de datos y además es multiplataforma. Generalmente se ejecuta en un servidor W*eb*, tomando el código en PHP como su entrada y creando páginas *Web* como salida (Hanze, y otros, 2008).

## 1.8 Bibliotecas

### *1.8.1 ExtJS 3.0*

ExtJS es una biblioteca o conjunto de bibliotecas JavaScript para el desarrollo de aplicaciones Web interactivas, usa tecnologías AJAX, DHTML y DOM. La misma permite realizar interfaces de usuario, es generalmente usada en sistemas que requieren altos niveles de interacción con el usuario (Frederick, y otros, 2008).

### *1.8.2 OpenLayers 2.0*

Es una biblioteca JavaScript para componer mapas dinámicos en páginas Web. Forma parte de los proyectos de la *Open Source Geospatial Foundation (OGC)*, está bajo una licencia BSD (Morales, 2015).

**¿Qué podemos hacer con OpenLayers 2.0?**

- Crear mapas interactivos.
- Visualizar información espacial/geográfica.
- Incluir y superponer distintos tipos de capas.
- Editar información espacial.

**Características:**

- Superposición de múltiples capas de mapa en una sola aplicación.
- Representación de elementos vectoriales.
- Reproyección del lado del cliente.

- Elementos de clúster y paginación.

## 1.9 Conclusiones parciales

El estudio de los principales conceptos asociados al objeto de estudio permitió al autor comprender los fundamentos teóricos necesarios para el desarrollo de la investigación. A partir de estos conceptos se encaminó el estado del arte de los sistemas homólogos ArcGIS, QGIS y gvSIG evidenciándose que no resolvían el problema planteado. Aun así, aportaron elementos importantes para el desarrollo del componente de detección de errores. Además se caracterizaron las herramientas y tecnologías que utiliza GeneSIG lo que permitió definir el entorno técnico con el cual se realizará el desarrollo de la solución.

# Capítulo II. Características y diseño del componente comprobador topológico

## 2.1 Introducción

En el presente capítulo se elabora el modelo de dominio y se exponen los conceptos y relaciones que intervienen en el mismo. Se realiza la especificación de los requisitos de software. Se describen los elementos de la arquitectura y los patrones arquitectónicos y de diseño a emplear. Se crea el diagrama correspondiente al modelo de diseño que es de vital importancia para la implementación del sistema.

## 2.2 Modelo de dominio

El objetivo del Modelo de Dominio es comprender y describir las clases más importantes dentro del contexto del sistema. Es el mecanismo fundamental para comprender el dominio del problema y para establecer conceptos comunes (Pressman, 2002).

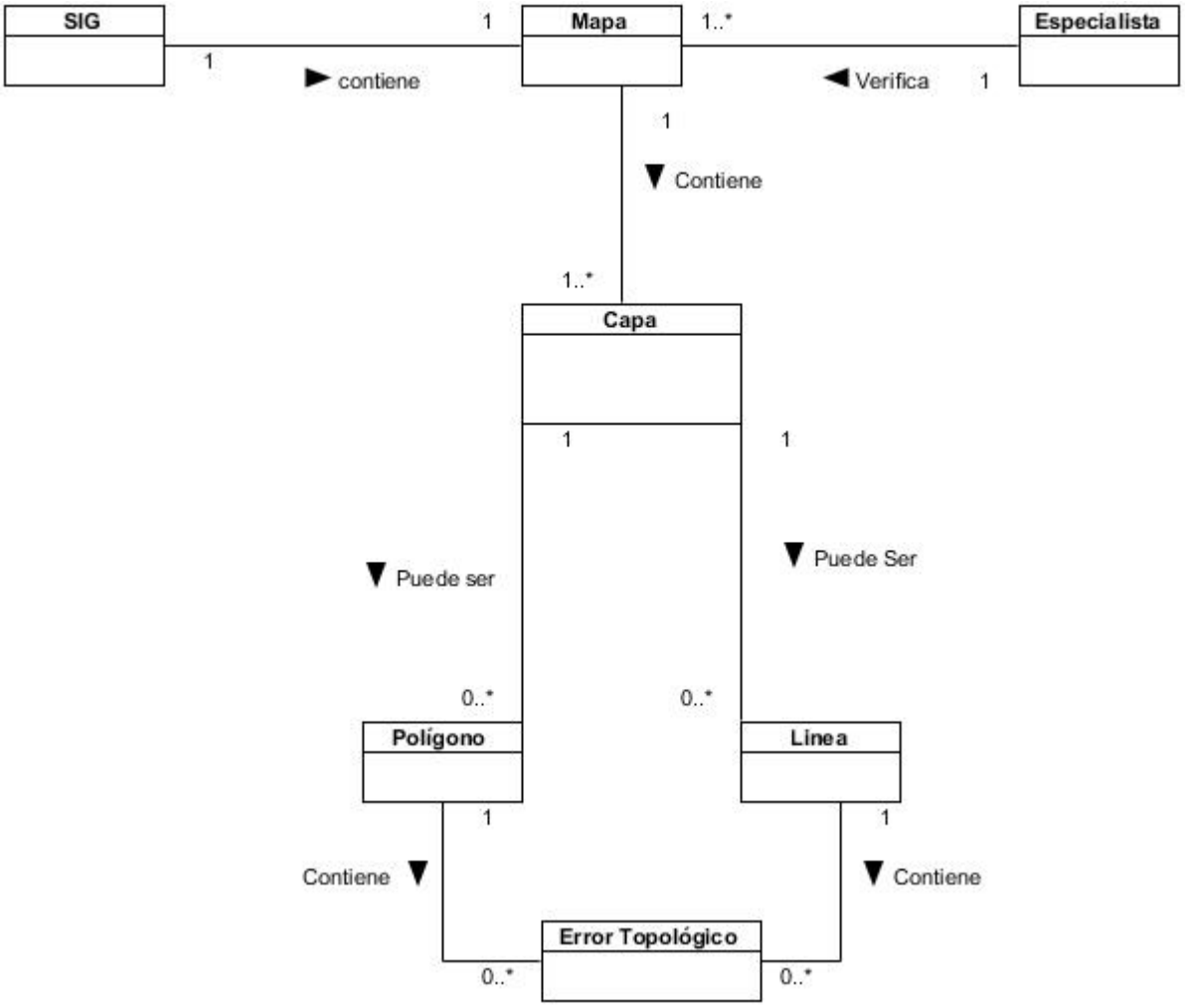



**Figura 4:** Modelo de dominio del sistema.

## 2.3 Descripción de los conceptos relacionados en el modelo de dominio

| Concepto | Descripción |
|---|---|
| Especialista | Es quien consulta la información representada sobre los mapas. |
| Mapa | Es consulta por una persona, contiene las capas. |
| Capa | Está contenida dentro de los mapas, pueden ser de tipo línea o polígono. |
| SIG | Es una integración organizada de hardware, software y dato geográfico diseñado para capturar, almacenar, manipular, analizar y desplegar en todas sus formas la información geográficamente referenciada. |
| Línea | Las líneas se utilizan en pequeñas escalas para representar polígonos, estos elementos lineales pueden medir distancia y son usadas para rasgos lineales: como curvas de nivel, ferrocarriles, ríos, entre otros. |
| Polígono | Los polígonos, tienen gran utilidad ya que representan elementos geográficos que cubren un área particular de la tierra, los mismos transmiten el mayor volumen de información en archivos vectoriales y en ellos se puede medir el perímetro y el área. |
| Error topológico | Rompen la relación entre los elementos, pueden agruparse dependiendo de si los tipos de características de vector son polígonos o líneas. |

**Tabla 1:** Descripción de los conceptos del dominio del negocio.

## 2.4 Requisitos del sistema

El levantamiento de requisitos se realiza con el objetivo de entender y comprender mejor el problema que se necesita solucionar, existen los Requisitos Funcionales (RF) los cuales especifican los detalles más relevantes que la aplicación debe presentar y los Requisitos No Funcionales (RNF) que son las cualidades que la misma debe poseer para un mejor funcionamiento.

### *2.4.1 Requisitos funcionales*

Los requisitos funcionales definen los servicios que un sistema debe proveer, su comportamiento a las diferentes entradas y situaciones. Para el desarrollo del componente se definen los siguientes RF agrupados en un proceso:

**Proceso: Realizar análisis de reglas**

**RF 1 Adicionar regla:** Adiciona una regla a la lista, para detectar un posible error topológico.

**RF 2 Modificar regla:** Modifica los valores de una regla que se encuentre añadida en la lista.

**RF3 Eliminar regla:** Elimina una regla o varias reglas de la lista.

**RF4 Verificar regla:** El sistema debe ser capaz de comprobar errores topológicos para todas las capas que contienen las reglas, en toda su extensión o en una extensión determinada del mapa. Se tienen en cuenta los tipos de reglas por capa, y todas las reglas que existen para cada tipo de capa.

**RF5 Limpiar campos:** Se limpia la lista de reglas, además las capas del mapa que estén representadas con un color luego de presentar un error.

**RF6 Listar reglas:** Se almacenan las reglas adicionadas al sistema.

El requisito número cuatro verificar regla, es el más crítico en el proceso de implementación, en este se implementan las reglas definidas para detectar los errores en las capas que se analizan. A continuación, se mencionan dichas reglas.

Para capas tipo polígono

- ✓ No se deben superponer
- ✓ No deben superponerse con
- ✓ Contiene un punto
- ✓ Contiene anillos

Para capas de tipo líneas

- ✓ Los puntos finales deben estar cubiertos
- ✓ No debe tener extremos sueltos
- ✓ No debe tener pseudos

Estas capas presentan reglas en común es el caso:

- ✓ No debe tener duplicados
- ✓ No debe tener geometrías multiparte
- ✓ No debe tener geometrías no válidas

A continuación, se muestran las especificaciones de los requisitos funcionales:

Especificación del **RF 1.** Adicionar regla:

| **Conceptos tratados** | **Conceptos** | **Atributos** |
|---|---|---|
| | Capa, error topológico. | - |
| **Precondiciones** | **Precondiciones** | **Pre-requisitos** |
| | Se debe haber cargado una capa. | No procede. |
| **Descripción** | 1. Se da clic en el botón Comprobador topológico. | |

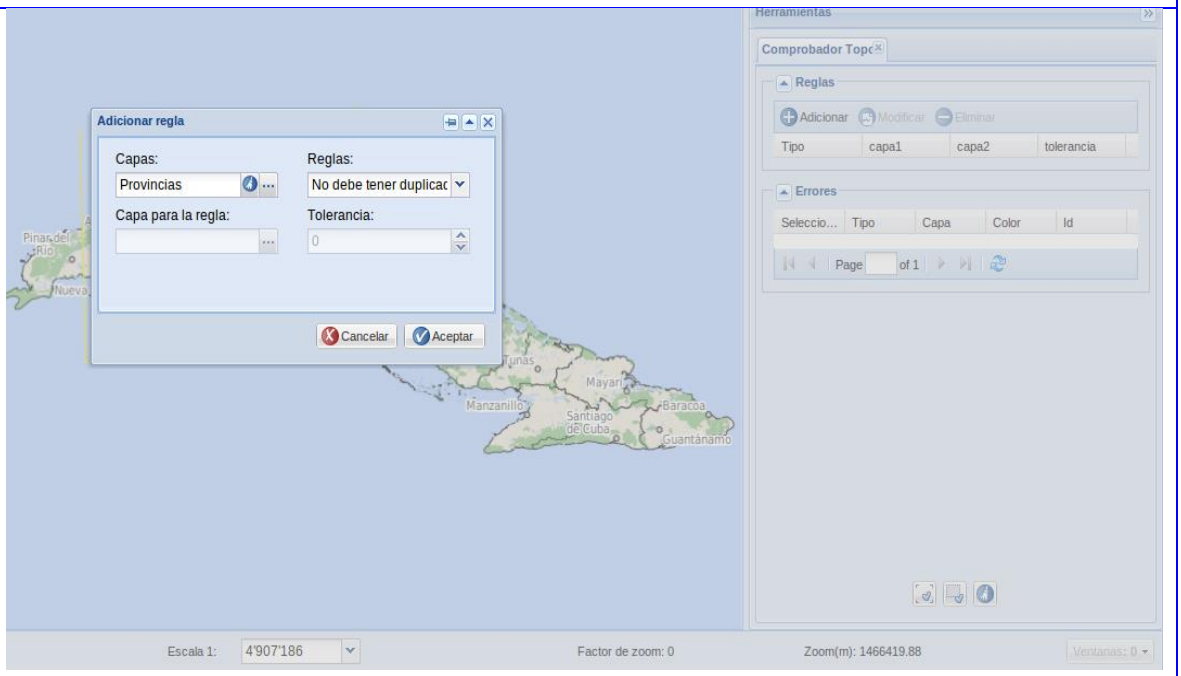
| | |
|---|---|
| | 2. Se da clic en el botón adicionar regla que se encuentra en la barra lateral derecha de herramientas.<br>3. Se selecciona la capa que se quiere cargar dando clic en el botón “capas” y se da clic en el botón aceptar, se selecciona la regla según el tipo de capa ya sea línea, punto o polígono de la lista despegable “reglas”, en caso de que la regla implique dos capas se selecciona una segunda dando clic en el botón “capa para la regla” además se le asigna un valor de tolerancia en el botón “tolerancia” a la regla si esta necesita este último valor.<br>4. Se hace clic sobre el botón aceptar. |
| **Validaciones** | Solo se activará el botón aceptar si se encuentra seleccionada una capa y una regla, además solo se podrá seleccionar una segunda capa si la regla requiere de la misma. |
| **Post-condiciones** | Quedará seleccionada la regla a detectar. |
| **Post-requisitos** | Detecta un posible error topológico. |
| **Prototipo de interfaz** |  |

**Tabla 2:** Especificación del RF 1. Adicionar regla.

Especificación del **RF 2.** Modificar regla:

| **Conceptos tratados** | **Conceptos** | **Atributos** |
|---|---|---|
| | Capa, error topológico | - |
| **Precondiciones** | **Precondiciones** | **Pre-requisitos** |
| | Se debe haber adicionado una regla. | No procede. |

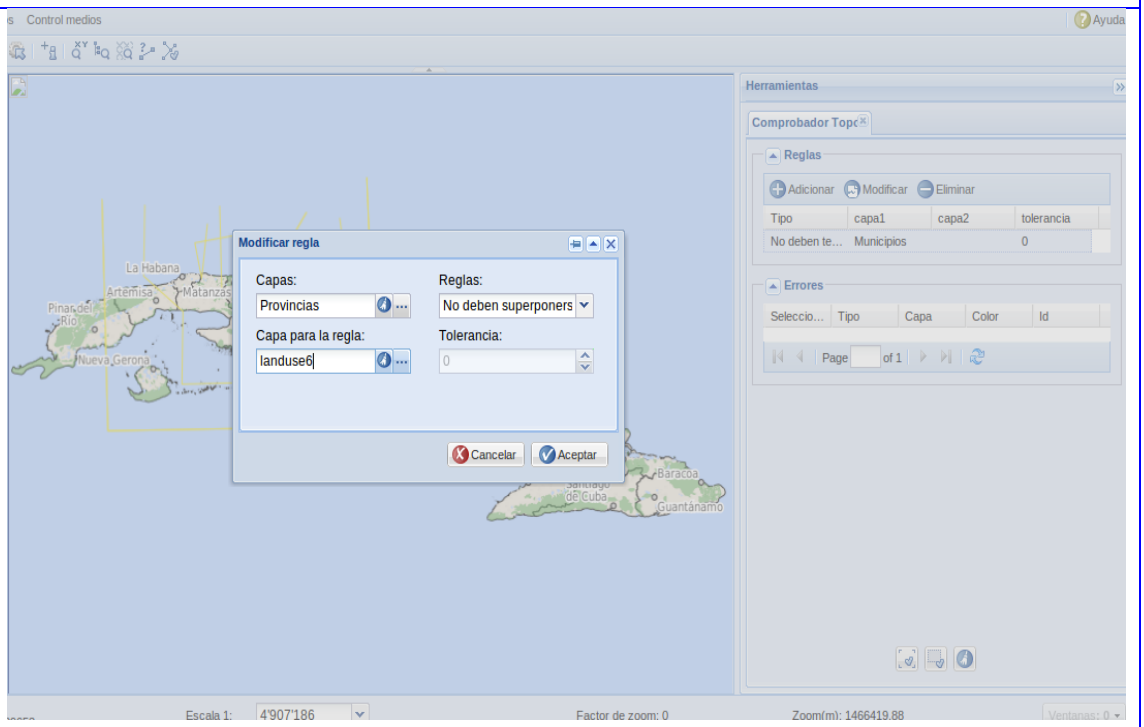
| | |
|---|---|
| **Descripción** | 1. Se da clic en el botón modificar regla que se encuentra en la barra lateral derecha de herramientas.<br>2. Se selecciona la nueva capa que se quiere cargar dando clic en el botón “capas”, se selecciona una nueva regla según el tipo de capa ya sea de tipo línea, punto o polígono de la lista despegable “reglas”, en caso de que la regla implique dos capas se selecciona una segunda del botón “capa para regla” además se le asigna un valor de tolerancia en el botón “tolerancia” a la regla.<br>3. Se hace clic sobre el botón aceptar. |
| **Validaciones** | Solo se activará el botón aceptar si se encuentra seleccionada una capa y una regla, además solo se podrá seleccionar una segunda capa si la regla requiere de la misma. |
| **Post-condiciones** | Quedará modificada la regla a detectar. |
| **Post-requisitos** | Detecta un posible error topológico. |
| **Prototipo de interfaz** |  |

**Tabla 3:** Especificación del RF 2. Modificar regla.

Especificación del **RF 3.** Eliminar regla:

| **Conceptos tratados** | **Conceptos** | **Atributos** |
|---|---|---|
| | Capa, error topológico | - |
| **Precondiciones** | **Precondiciones** | **Pre-requisitos** |

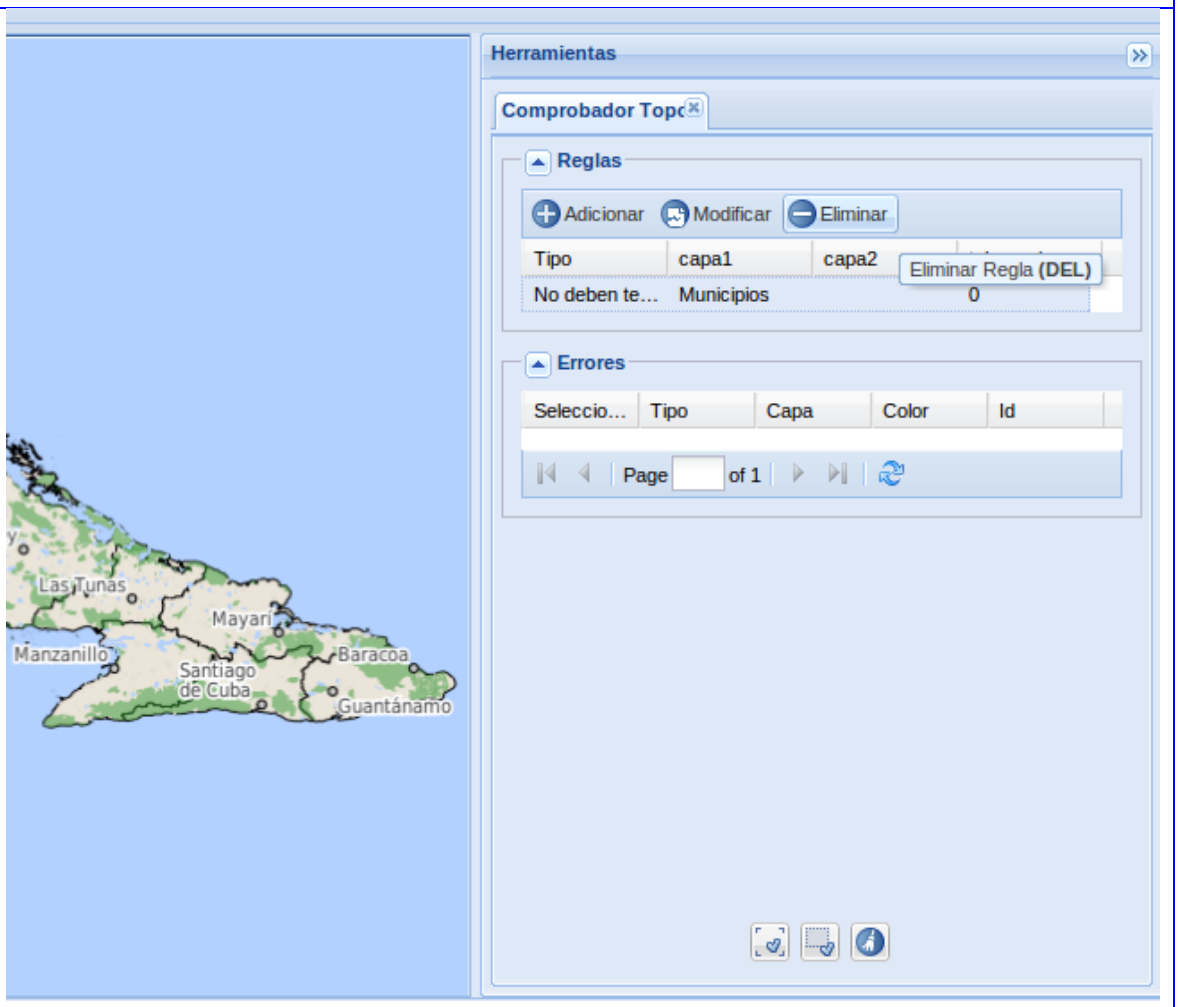
| | | |
|---|---|---|
| | Se debe haber adicionado una regla. | No procede. |
| **Descripción** | 1. Se selecciona una o varias reglas del listado que se encuentra en la barra lateral derecha de herramientas.<br>2. Se da clic en el botón eliminar regla que se encuentra en la barra lateral derecha de herramientas. | |
| **Validaciones** | Solo se activará el botón eliminar si se tiene seleccionada una o varias reglas. | |
| **Post-condiciones** | Se eliminará la regla seleccionada. | |
| **Post-requisitos** | _ | |
| **Prototipo de interfaz** |  | |

**Tabla 4:** Especificación del RF 3. Eliminar regla.

Especificación del **RF 4.** Verificar regla:

| **Conceptos tratados** | **Conceptos** | **Atributos** |
|---|---|---|
| | Capa, error topológico | - |
| **Precondiciones** | **Precondiciones** | **Pre-requisitos** |
| | Se debe haber adicionado una regla. | No procede. |
| **Descripción** | 1. Se da clic sobre el botón verificar todo para comprobar todo el mapa, o se da clic en el botón verificar extensión para comprobar solo la parte que muestra el mapa ambos botones se encuentran en la barra lateral derecha de herramientas.<br>2. Al dar clic en uno de estos botones se llamará a la función correspondiente que verifica la regla. | |
| **Validaciones** | Solo se activará el botón verificar todo o verificar extensión si se encuentra seleccionada una regla. | |
| **Post-condiciones** | Se verificará la regla seleccionada. | |
| **Post-requisitos** | En caso de que la capa presente errores se mostraran los mismos. | |

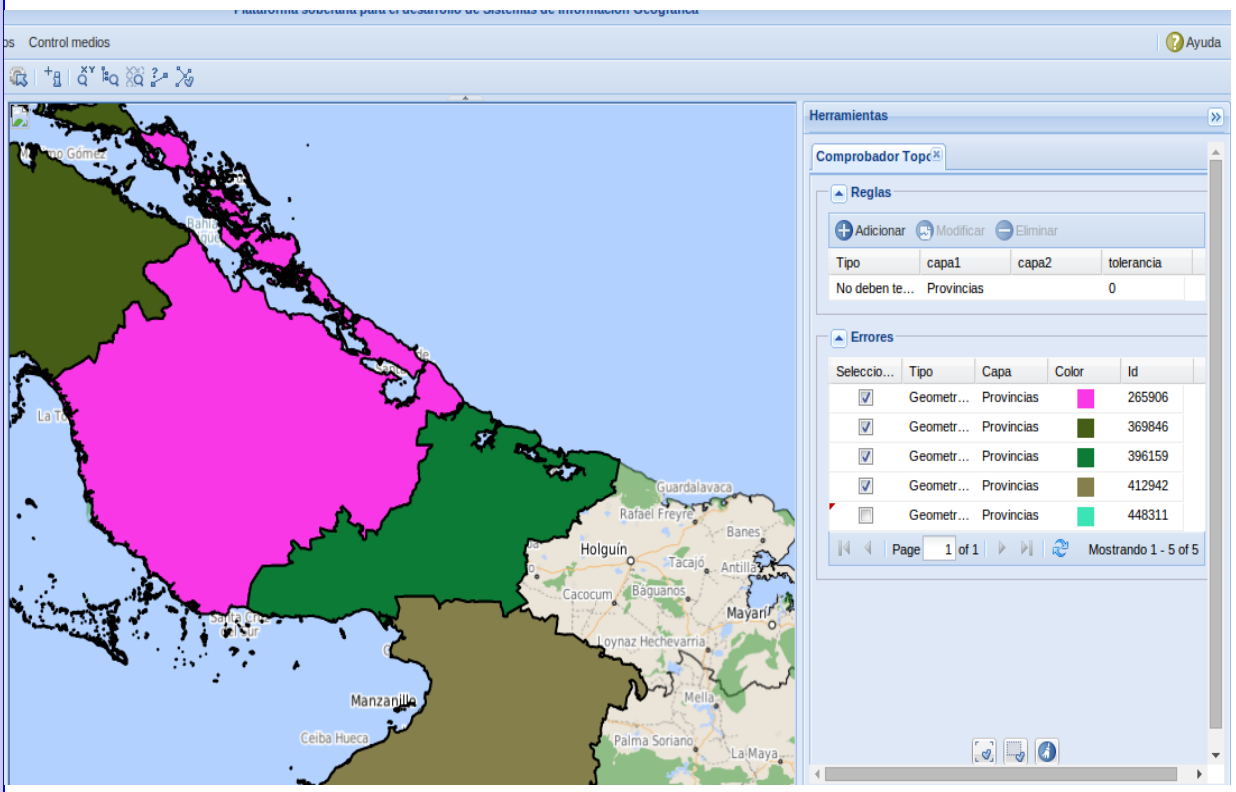
| Prototipo de interfaz |  |
|---|---|

**Tabla 5:** Especificación del RF 4. Verificar regla.

### *2.4.2 Requisitos no funcionales*

- **RNF 1 Interfaz gráfica o apariencia externa:**

  El componente debe poseer una interfaz gráfica uniforme incluyendo menús y opciones, tanto el título del componente, como los mensajes para interactuar con los usuarios, deberán ser en idioma español y tener una apariencia uniforme guiada por los colores blanco y azul claro. Los mensajes definidos deberán ser lo suficientemente informativos para dar a conocer la severidad de los mismos.

- **RNF 2 Software**

  Para las PCs clientes:

  - Un navegador como Mozilla Firefox versión 28.0 o superior u otro navegador que cumpla los estándares de W3C.

  Para las PCs servidores:

  - Sistema operativo GNU/Linux versión 14.04.
  - Servidor Web Apache 2.0 o superior, con módulo PHP 5.3.

- PostgreSQL 9.4 como sistema gestor de base de datos.
- PostGIS 2.0 como extensión de PostgreSQL 9.4 como soporte de datos espaciales.
- MapServer 6.4, con extensión PHP MapScript.

❖ **RNF 3 Hardware**

Para PCs clientes:

- Se requiere tengan tarjeta de red.
- Al menos 1 GB de memoria RAM.
- Procesador 512 MHz mínimo.

Para PCs servidores:

- Se requiere tarjeta de red.
- El servidor de mapas debe tener como mínimo 2 GB de RAM.
- El servidor de base de datos debe tener como mínimo 2 GB de RAM.
- Procesador 3 GHz mínimo.

❖ **RNF 4 Restricciones de diseño e implementación**

- Lenguaje de programación: JavaScript y PHP.
- Como gestor de base de datos: PostgreSQL 9.4 con extensión PostGIS 2.0.

## 2.5 Arquitectura del sistema

La plataforma GeneSIG emplea la arquitectura orientada a objetos y la arquitectura basada en componentes, de este modo son las utilizadas para el desarrollo del componente. A continuación, se realiza una breve descripción de estas:

**Arquitectura Orientada a Objetos:** “nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes del estilo se basan en principios orientados a objetos: encapsulamiento, herencia y polimorfismo. Las interfaces están separadas de las implementaciones. Las representaciones de los datos y las operaciones están encapsuladas en un tipo abstracto de datos u objeto. La comunicación entre los componentes es a través de mensajes” (Reynoso, y otros, 2004).

**Arquitectura Basada en Componentes:** “se centra en el diseño y construcción de sistemas computacionales que utilizan componentes de software reutilizables”. Define la composición de software

como “el proceso de construir aplicaciones mediante la interconexión de componentes de software a través de sus interfaces (de composición)", abogaba por la utilización de componentes prefabricados sin tener que desarrollarlos de nuevo” (Robaina, 2008).

### *2.5.1 Estilo Arquitectónico*

Un estilo arquitectónico: “Expresa componentes y las relaciones entre estos, con las restricciones de su aplicación y la composición asociada, así como también las reglas para su construcción. Se consideran como un tipo particular de estructura fundamental para un sistema de software, conjuntamente con un método asociado que especifica cómo construirlo incluyendo información acerca de cuándo usar la arquitectura que describe, sus invariantes y especializaciones, así como las consecuencias de su aplicación” (Camacho, y otros, 2004).

Los estilos arquitectónicos son un tipo particular de estructura que definen los posibles patrones a usar en la implementación de la aplicación. Los estilos más conocidos son: Flujo de Datos, Centrado en datos y Llamada y Retorno, este último es el que se emplea en este trabajo. Los componentes y las relaciones entre estos se realizan a través de este estilo que: “Permite construir una estructura de programa relativamente fácil de modificar y ajustar. Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Es el estilo más generalizado en sistemas de gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas” (Reynoso, y otros, 2004).

Luego de haber analizado la arquitectura de la plataforma GeneSIG y haber decidido mantener la misma para el componente a desarrollar, se hace necesario entonces definir los patrones de diseño que se pueden emplear en la implementación de la solución.

## 2.6 Diseño del Sistema

Siguiendo las buenas prácticas de la metodología de desarrollo Prodesoft, la disciplina de Diseño y Arquitectura se especifica detalladamente. Por tanto, a continuación, se presentan algunos aspectos importantes que se tuvieron en cuenta asociados a esta disciplina:

### *2.6.1 Patrones de diseño*

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular (Visconti, y otros, 2005). En el presente trabajo se emplean los siguientes Patrones Generales de Asignación de Responsabilidades (GRASP, pos sus siglas en inglés):

**Experto:** se encarga de asignar una responsabilidad al experto en información, es decir, la clase que cuenta con la información necesaria para cumplir la responsabilidad. Este patrón se pone de manifiesto en la clase **ClientTopologic.php** la cual es experta en manejar las peticiones. El uso de este patrón permite que se conserve el encapsulamiento, donde cada objeto contiene sus propios atributos y funcionalidades para cumplir su tarea.

**Creador:** tiene la responsabilidad de identificar quién debe ser el responsable de la creación de nuevos objetos o clases. La nueva instancia deberá ser creada por la clase que tiene la información necesaria para realizar la creación del objeto, usa directamente las instancias creadas del objeto, o maneja varias instancias de la clase. El uso de este patrón se puede apreciar en el diseño de la solución cuando se le solicita información a la clase **ClientTopologic** y esta se encarga de crear instancias de la clase **toplogicRequest**. De igual forma la clase **ServerTopologic** crea instancias de la clase **toplogicResult**.

**Alta Cohesión:** asigna a las clases responsabilidades que trabajen sobre una misma área de la aplicación y que no tengan mucha complejidad, evitando así que una clase sea la única responsable de muchas tareas en áreas funcionales muy heterogéneas.

**Bajo Acoplamiento:** se asignan las responsabilidades de forma tal que cada clase se comunique con el menor número de clases, minimizando el nivel de dependencia.

Los patrones de Grupo de los Cuatro (GoF, por sus siglas en inglés) resultan otra clasificación de los patrones de diseño que pueden ser utilizados en una solución informática. En la solución propuesta se pusieron en práctica los siguientes patrones GoF:

**Singlenton:** aplicar este patrón en el diseño de clases garantiza el acceso único a una clase mediante una única instancia. Por este medio se puede controlar el acceso a las clases. El patrón Singlenton se evidencia al modificar el framework CartoWeb (framework en que está basada la estructura de GeneSIG), donde el objetivo del mismo es crear el objeto “mapa” para que no se cree cada vez que se hace un envío en la aplicación.

**Command:** este patrón permite encapsular las peticiones a través de un objeto, lo que permite realizar operaciones como gestionar las acciones de dicho objeto. Se utiliza para la comunicación a través de las interfaces de usuario, específicamente a través de la clase **AJAXHelper:** que es la encargada de comunicar las interfaces con el servidor. Uno de los aspectos más importantes en el sistema son las interfaces gráficas de usuario, ya que el usuario interactúa constantemente con ellas y por eso principalmente se aplica este patrón GoF a la solución.

### *2.6.2 Descripción del modelo de diseño*

Para una mejor comprensión del propósito de cada clase del diseño de la solución, se describen los elementos del Proceso.

**index.php:** tiene como propósito controlar la realización del RF en sí, recibe las peticiones realizadas por el cliente, gestiona las mismas y manda a construir la ClientPage.

**CartoClient:** integra y recoge todos los datos y funciones realizadas por cada una de los ficheros JavaScript que intervienen en el RF y se definen una serie de variables globales que van a ser utilizadas por la aplicación.

**Client:** contiene todos los archivos específicos de PHP del lado de CartoClient y permite la interacción entre la index.php y la CartoClient.

**PluginManager:** clase que se utiliza para gestionar la base de plugins.

**ClientPlugin:** contiene las interfaces necesarias para los plugins del lado del cliente.

**ServerPlugin:** esta clase proporciona la base de herramientas para el desarrollo de plugins.

**ServerContex:** es la clase contenedora de la información común que ha de ser utilizada por la parte cliente y la servidora, empleando la información seleccionada como un objeto para un fácil manejo de los datos.

**MapObj**: es donde se definen los métodos, funciones, además del lenguaje para el intercambio de datos con el servidor de mapa (mapa MapServer).

**Common:** es la clase encargada de administrar las conexiones a la base de datos para ejecutar las consultas a la misma satisfactoriamente, esto incluye tratamiento de los datos.

**Pgsql:** gestiona desde PHP las funciones de PostgreSQL.

**CwSerializable:** se encarga de serializar todas aquellas clases que pueden ser serializadas, permitiendo la comunicación entre el Client y el Server del plugin.

**AJAX_Helper:** tiene como propósito enviar las respuestas de los plugins “AJAX”, para alimentar a los plugins que responden a las peticiones del usuario.

**Index.html:** página principal encargada de mostrar en el mapa la región localizada.

**TopologicAJAX:** es la encargada de gestionar el pedido y respuesta a las peticiones del usuario por Ajax.

**Realizar Análisis de Reglas:** proceso que abarca los seis requisitos funcionales.

**ServerToplogic:** es la clase servidora que tiene como principal función realizar las operaciones topológicas sobre diferentes capas. Establece la conexión con la base de datos espacial para realizar las consultas requeridas y enviar las respuestas necesarias a la clase **ClientToplogic**.

**ClientToplogic:** es la clase encargada de recibir las peticiones provenientes de los ficheros JavaScript de la aplicación y enviarlas a la clase **ServerTopologic** conformando un objeto de la clase **ToplogicRequest.** Además, es el puente de comunicación entre la clase **ServerTopologic** y la interfaz de la aplicación una vez que se tenga un resultado.

**ToplogicRequest:** es una clase común encargada de transportar los datos recogidos en **ClientToplogic** desde la interfaz y transportarlos a la clase **ServerToplogic**.

**ToplogicResult**: es una clase común encargada de transportar los datos generados en **ServerToplogic** a la clase **ClientToplogic**.

### *2.6.3 Diagrama de clases del diseño*

Representa una abstracción de una o varias clases en la implementación del sistema, dependiendo del lenguaje de programación. Las clases definen los objetos, con los cuales se implementan los casos de uso. Las particularidades del lenguaje de programación influyen en el diseño de las clases (Larman, 2003).

A continuación, se presenta en la Figura 5 el diagrama de clases del diseño del proceso realizar análisis de reglas.

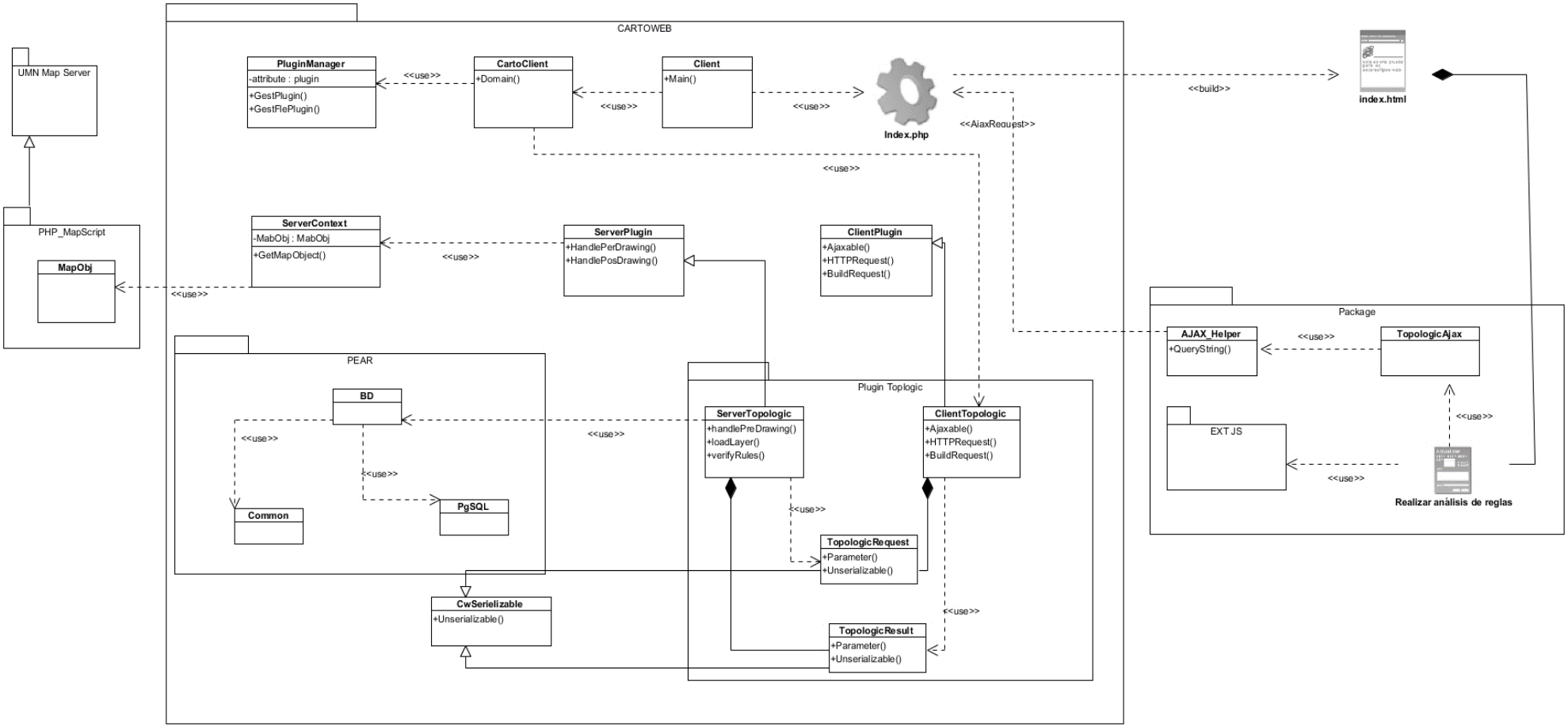



**Figura 5:** Diagrama de Clases del diseño del Proceso. Realizar análisis de reglas.

## 2.7 Conclusiones parciales

En el presente capítulo se generaron los artefactos de la metodología de desarrollo necesarios para la implementación de la solución. Se identificaron seis requisitos funcionales agrupados en el proceso realizar análisis de regla, siendo crítico el de verificar reglas. Además, se determinaron cuatro requisitos no funcionales mediante los cuales el autor estableció las características de hardware y software imprescindibles para el funcionamiento del componente. El empleo de patrones de diseño GRASP y GoF para el desarrollo del componente, permitió que la programación estuviese bien estructurada. Se obtuvo una estructura conocida por todos los programadores lo que proporciona el ahorro de tiempo en la construcción y/o actualización de funcionalidades del software desarrollado.

# Capítulo III. Implementación y prueba del componente comprobador topológico

## 3.1 Introducción

En el presente capítulo se exponen los principales elementos de la implementación del sistema, se realiza el diagrama de componentes, se abordan las estrategias de codificación. Se explica el modelo de despliegue y se le realizan pruebas a la solución para validar y asegurar el correcto funcionamiento del sistema.

## 3.2 Estrategias de codificación

Los estándares de codificación son reglas que se siguen para lograr uniformidad en la escritura del código (Kernighan, y otros, 1999). Su empleo facilita la comprensión del sistema y propicia que diferentes programadores trabajen de forma coordinada. Para el desarrollo del sistema se utiliza el estándar de codificación Camel, el cual posee dos variantes:

- UpperCamelCase: primera letra en mayúscula, al igual que la primera letra de cada palabra interna.
- lowerCamelCase: primera letra en minúscula y la primera letra de cada palabra interna en mayúscula.

Se utiliza UpperCamelCase para el nombre de las clases y lowerCamelCase para el nombre de métodos y variables. En las declaraciones no deben dejarse espacios en blanco entre el nombre de un método y el paréntesis «(» que abre su lista de parámetros.

```php
<?php

class ServerTopologic extends ClientResponderAdapter
{
```

***Figura 6:*** Declaración de una clase empleando UpperCamelCase.

```php
    public function geometriesDuplicate($data, $extent, $geomExtent)
```

**Figura 7:** Declaración de métodos y variables empleando lowerCamelCase.

## 3.3 Diagrama de componentes

Este tipo de diagrama representa la parte modular, desplegable y reemplazable de un sistema que encapsula implementación. También expone un conjunto de interfaces que funcionan como elementos intermediarios de comunicación entre componentes que posibilitan la independencia funcional (ver figura 8).

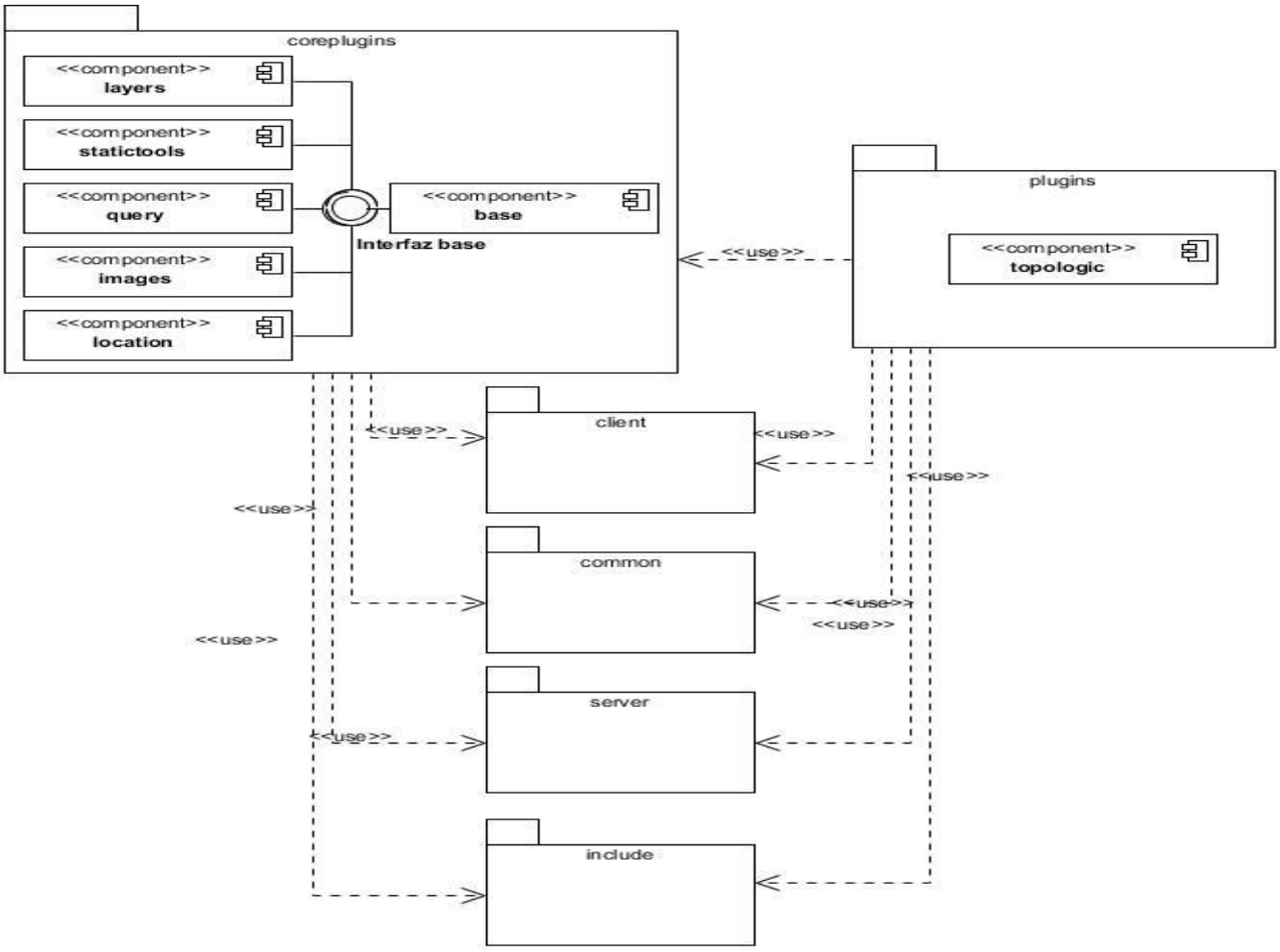



**Figura 8:** Diagrama de Componentes del Proceso Realizar análisis de reglas.

## 3.4 Diagrama de despliegue

En el primer nodo se cuenta con un pc cliente que se comunica por el protocolo HTTP con el servidor de mapas y el servidor de aplicaciones, ambos se encuentran en un único nodo servidor. El servidor de bases de datos se halla instalado en otro nodo servidor, el cual se comunica con el servidor de mapas a través del protocolo TCP/IP. A continuación, se muestra dicho diagrama (ver figura 9).

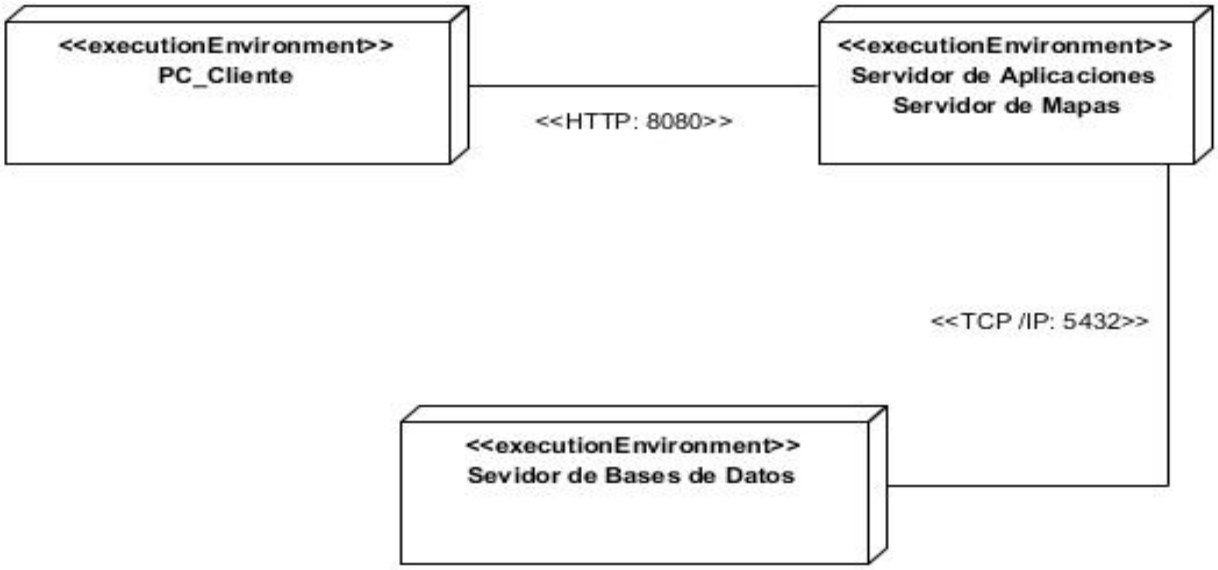



**Figura 9:** Diagrama de despliegue del sistema.

## 3.5 Pruebas

Las pruebas son un proceso de ejecución de un programa que se realiza con la intención de detectar errores. Decir que los casos de prueba se realizan con el propósito de identificar y comunicar las condiciones que se llevarán a cabo en la prueba. Los mismos son necesarios para verificar que la aplicación tenga éxito y que cumpla con los requisitos del producto.

### *3.5.1 Diseño de Casos de Prueba*

Un diseño de caso de prueba (DCP) está compuesto por un conjunto de entradas, respuesta que emite el sistema de acuerdo a esas entradas y el flujo central que indica el camino del escenario descrito. Estos son desarrollados para verificar el cumplimiento total o parcial de un requisito. Las entradas representan las variables que se pueden especificar y las mismas contienen: V, I, o N/A. V indica válido, I indica inválido, y N/A que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante.

Se realizaron los DCP para el requisito del proceso realizar análisis de regla.

A continuación, la tabla 6 muestra los DCP para el requisito adicionar regla.

| Escenario | Descripción | Variables | | | | Respuesta del sistema | Flujo Central |
|---|---|---|---|---|---|---|---|
| | | Capas | Reglas | Capa para la regla | Tolerancia | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| EC 1.1. Adicionar capa. | Seleccionar una capa del árbol. | V<br>Activar botón. | N/A<br>- | N/A<br>- | N/A<br>- | Muestra un árbol con las capas contenidas en el mapa. Estas son de tipo Polígono, Línea y Punto. | Se selecciona el botón “Adicionar” que se encuentra en la barra de herramientas. Se muestra una ventana emergente con el nombre “Adicionar regla”. Se selecciona el botón “Capas” y se elige la capa deseada. |
| EC 1.2. Adicionar regla. | Seleccionar una regla de la lista despegable. | V<br>Activar botón | V<br>Activar radiobut ton | N/A<br>- | N/A<br>- | Muestra las reglas que pueden ser utilizadas según la capa selecciona da. | Se selecciona el botón “Adicionar” que se encuentra en la barra de herramientas. Se muestra una ventana emergente con el nombre “Adicionar regla”. Se selecciona el radiobutton “Reglas” y se elige la regla deseada. Luego se presiona el botón “Aceptar”. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| EC 1.3. Adicionar capa para la regla. | Seleccionar una segunda capa del árbol según la regla seleccionada. | V Activar botón. | V Activar radiobutton. | V Activar botón. | N/A - | Muestra un árbol con las capas que son necesarias para comprobar según la regla. Estas son de tipo Polígono, Línea y Punto. | Se selecciona el botón “Adicionar” que se encuentra en la barra de herramientas. Se muestra una ventana emergente con el nombre “Adicionar regla” con la opción para adicionar una capa. Se selecciona el botón “Capa para la regla” y se escoge la segunda capa deseada. Luego se presiona el botón “Aceptar”. |
| EC 1.4. Asignar tolerancia. | Asigna un valor a la regla que se quiere adicionar. | V Activar botón . | V Activar radiobutton. | V Activar botón. | V Activar spinner. | Crea un valor de tolerancia para la regla que se quiere analizar. | Se selecciona el botón “Adicionar” que se encuentra en la barra de herramientas. Se muestra una ventana emergente con el nombre “Adicionar regla” con la opción para asignar una tolerancia a la regla en caso de la misma necesite o se le aplique tolerancia, se le asigna el valor activando el spinner “Tolerancia”. Luego se presiona el botón “Aceptar”. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| EC 1.5. Adicionar una regla con campos en blanco. | Verifica si se introdujeron todos los datos de la regla a adicionar. | I | I | I | N/A<br>- | El botón "Aceptar", no se activará evidencian do la falta de datos obligatorios de la regla estos, campos se mostraran en rojo. | Se selecciona el botón “Adicionar” que se encuentra en la barra de herramientas. Se muestra una ventana emergente con el nombre “Adicionar regla”. |
| EC 1.6. Cancelar realización de adicionar una regla. | Anular cualquier acción que conlleve adicionar una regla. | N/A<br>- | N/A<br>- | N/A<br>- | N/A<br>- | . | Se selecciona la opción del botón “Cancelar” de la ventana “Adicionar regla”. |

**Tabla 6:** DCP Adicionar regla

| No. | Variable | Valor Nulo | Descripción |
|---|---|---|---|
| 1 | Capas | No | Es un botón que muestra un árbol con todas las capas contenidas en el mapa a las cuales se le puede aplicar una regla deseada. |
| 2 | Reglas | No | Es un radiobutton que permite elegir una regla aplicable a la capa seleccionada. |
| 3 | Capa para la regla | No | Es un botón que muestra un árbol con todas las capas contenidas en el mapa a las cuales se le puede aplicar una regla deseada que tenga como funcionalidad el uso de una segunda capa para su comprobación. |
| 4 | Tolerancia | No | Es un spinner que permita asignar una tolerancia a la regla en caso de que la misma la necesite o se le aplique tolerancia. |

**Tabla 7:** Variables del caso de pruebas Adicionar reglas

A continuación, la tabla 8 muestra los DCP para el requisito Modificar regla.

| Escenario | Descripción | Variables | | | | Respuesta del sistema | Flujo Central |
|---|---|---|---|---|---|---|---|
| | | Capas | Reglas | Capa para la regla | Tolerancia | | |
| EC 2.1 Modificar regla. | Permite modificar una regla que ya se encuentra en la lista de reglas. | V<br>Activar botón | N/A<br>- | N/A<br>- | N/A<br>- | Se modifica la regla, se actualiza la lista de reglas. | Se selecciona el botón “Modificar” que se encuentra en la barra de herramientas. Se muestra una ventana emergente con el nombre “Modificar regla” y se seleccionan los campos que se quieren modificar. |
| EC 2.2 Cancelar operación de modificar una regla. | Se cancela la operación de modificar una regla. | V<br>Activar botón. | V<br>Activar radiobutton. | N/A<br>- | N/A<br>- | Se cancela la operación de modificar una regla. | Se selecciona el botón “Modificar” que se encuentra en la barra de herramientas. Se muestra una ventana emergente con el nombre “Modificar regla” y se presiona el botón “Cancelar”. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| EC 2.3 Modificar una regla con campos en blanco. | Verifica que están todos los datos de la regla. | I | I | I | I | El botón "Aceptar", no se activará evidencian do la falta de datos obligatorios de la regla, estos campos se mostraran en rojo. | Se selecciona el botón “Modificar” que se encuentra en la barra de herramientas. Se muestra una ventana emergente con el nombre “Modificar regla” y se seleccionan los campos que se quieren modificar. |

**Tabla 8:** DCP Modificar regla.

| No. | Variable | Valor Nulo | Descripción |
|---|---|---|---|
| 1 | Capas | No | Es un botón que muestra un árbol con todas las capas contenidas en el mapa a las cuales se le puede aplicar una regla deseada. |
| 2 | Reglas | No | Es un radiobutton que permite elegir una regla aplicable a la capa seleccionada. |
| 3 | Capa para la regla | No | Es un botón que muestra un árbol con todas las capas contenidas en el mapa a las cuales se le puede aplicar una regla deseada, que tenga como funcionalidad el uso de una segunda capa para su comprobación. |
| 4 | Tolerancia | No | Es un spinner que permita asignar una tolerancia a la regla en caso de que la misma la necesite o se le aplique tolerancia. |

**Tabla 9:** Variables del caso de prueba Modificar reglas.

### *3.5.1 Resultados de las pruebas*

A continuación, se presentan los resultados de las pruebas realizadas a la solución informática desarrollada:

Las pruebas de caja negra también denominadas pruebas de comportamiento, se centran en verificar el cumplimiento de los requisitos funcionales del software, se emplean cuando se conoce la función específica

para la que se diseñó la solución y se aplican a la interfaz del software, permitiendo obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales del programa (Pressman, 2005), con el fin de encontrar la mayor cantidad de no conformidades existentes en el producto. Dentro de las técnicas empleadas por las pruebas de caja negra se utilizó, partición de equivalencia, que según define Pressman: “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” (Pressman, 2005).

Una vez diseñados los casos de prueba para cada requisito funcional, se procedió a la ejecución de las pruebas, teniendo en cuenta que la solución tiene características especiales, pues en la mayoría de los casos las entradas son eventos del ratón.
Los resultados se pueden observar en la Figura 10, donde se muestran la cantidad de las no conformidades encontradas clasificadas en altas o bajas, las cuales fueron corregidas. Se realizaron tres iteraciones, luego de las cuales no se encontraron no conformidades, cumpliéndose correctamente los requisitos funcionales.

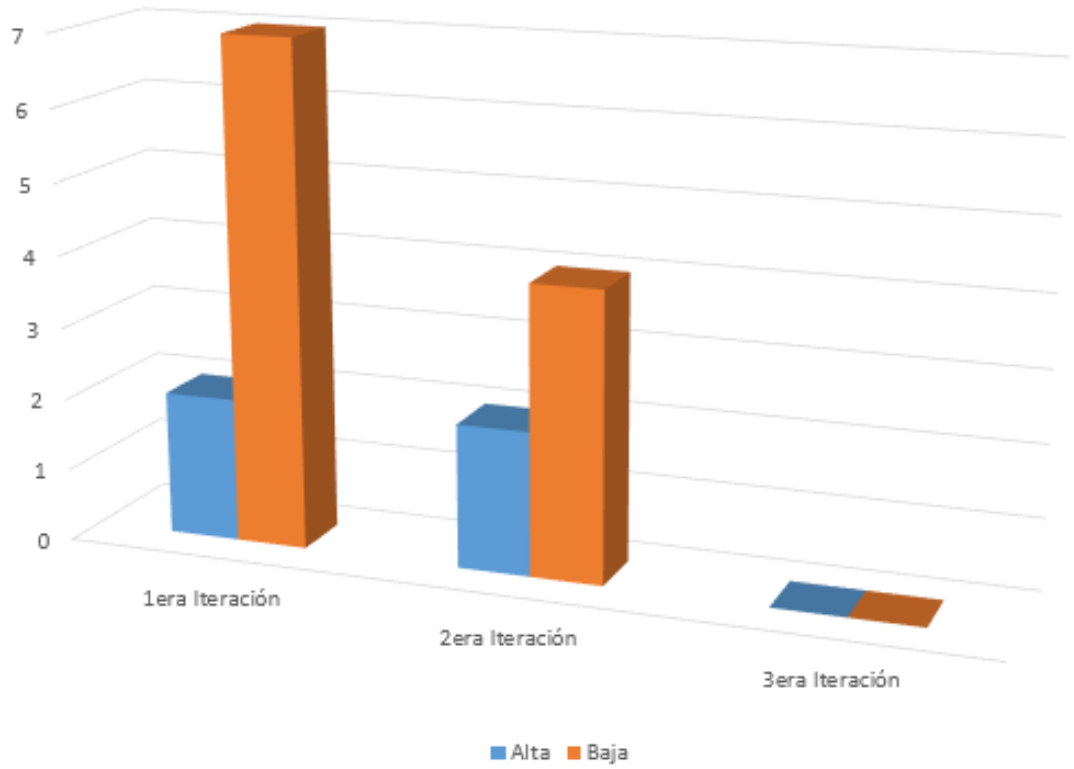



**Figura 10:** Pruebas de Caja Negra. Cantidad de no Conformidades por cada iteración clasificadas en altas y bajas.

Una prueba de unidad es la prueba enfocada a los elementos testeables más pequeños del software. La prueba de unidad siempre está orientada a caja blanca (Oré B, 2009). El diseño de casos de pruebas de una unidad comienza una vez que se ha desarrollado, revisado y verificado en su sintaxis el código a nivel

fuente. Es una forma de probar el correcto funcionamiento de un módulo o componente del código. Esto sirve para asegurar que cada elemento de estos funcione correctamente por separado. Los errores más frecuentes encontrados en este nivel de prueba están relacionados con las interfaces de comunicación entre componentes, interfaces entrada/salida, estructuras de datos locales, cálculos y flujos de control.
Para comprobar el funcionamiento independiente de cada funcionalidad se empleó el método de caja blanca, guiado por la técnica del camino básico.

Salidas de las pruebas de caja blanca aplicadas a la función ***geometriesMultipart***:

```
1  public function geometriesMultipart ($data, $extent, $geomExtent)
2  {
3      try {
4
5          $db = $this->getPluginConnection();
6          $dataLayer = $this->getLayerByName($data->capa1->name);
7          $elements = PostgisDataGenesig::obtenerCuerpoConsulta($dataLayer->data);
8          $geom = $elements['geom'];
9          $geomResults = [];
10
11         if ($data->capa1->connection == 'shape') {
12             $geometries = $this->getFeatures($data->capa1->name, $extent);//geometrias de la capa shape
13             $query = "SELECT ST_GeometryType(ST_GeometryFromText(?))";
14
15             for ($i = 0; $i < count($geometries); $i++) {
16                 $geom = $geometries[$i];
17                 $queryResults = $db->getAll($query, array($geom));
18                 Utils::checkQueryError($queryResults);
19                 if ($queryResults[0][0] == "ST_MultiPolygon" || $queryResults[0][0] == "ST_MultiLineString" || $queryResults[0][0] == "ST_MultiPoint") {
20                     $geomResults[] = $geom;
21                 }
22             }
23         } else {
24             $srid = $this->getSrid($elements);
25             $query = "SELECT ST_AsText(ST_Force2D($geom)) FROM ! WHERE (ST_GeometryType($geom)='ST_MultiPolygon' OR
26             ST_GeometryType($geom)='ST_MultiLineString' OR ST_GeometryType($geom)='ST_MultiPoint' ) AND ST_Intersects($geom ,ST_GeomFromText(?,?))";
27             $queryResults = $db->getAll($query, array($elements['from'], $geomExtent, $srid));
28             Utils::checkQueryError($queryResults);
29             for ($i = 0; $i < count($queryResults); $i++) {
30                 $geomResults[] = $queryResults[$i][0];
31             }
32         }
33         return $geomResults;
34     } catch (Exception $e) {
35         $this->result->msg = $this->msg;
36         return null;
37     }
38 }
```

**Figura 11:** Función geometriesMultipart.

Complejidad Ciclomática:
Aplicada de dos formas.

**V (G)** = *a* - *n* + 2, siendo *a* el número de arcos o aristas del grafo y *n* el número de nodos.
**a=21 n =17 21-17+2=6**

**V (G)** = c + 1, siendo *c* el número de nodos de condición.
**c=5 5+1=6**

Caminos (C) independientes:

**C1** = 1, 2, 3, 4, 5, 6, 7, 8, 9, 4, 16, 17.

**C2** = 1, 2, 3, 4, 5, 6, 15, 17.
**C3** = 1, 2, 3, 4, 16, 17.
**C4** = 1, 2, 10, 11, 12, 13, 14, 13, 16, 17.
**C5** = 1, 2, 10, 11, 12, 15.
**C6** = 1, 2, 10, 11, 12, 13, 16, 17.

Grafo (G) de flujo resultante:

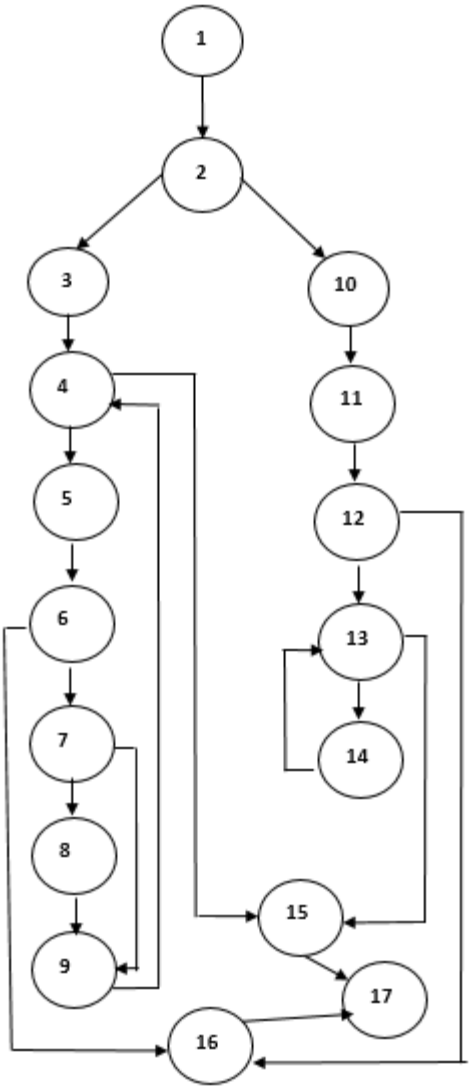



**Figura 12:** Grafo del caso de prueba de caja blanca para el método geometriesMultipart.

Salidas de las pruebas de caja blanca aplicadas a la función ***detached***:

```
public function detached($geometries)
{
    try {
        $db = $this->getPluginConnection();
        $queryVerify = "SELECT ST_IsClosed (ST_GeometryFromText(?))";
        for ($i = 0; $i < count($geometries); $i++) {
            $queryResults = $db->getAll($queryVerify, array($geometries[$i]));
            Utils::checkQueryError($queryResults);
            if ($queryResults[0][0] == 't') {
                $geomResults[] = $geometries[$i];
            }
        }
        return $geomResults;
    } catch (Exception $e) {
        return $queryResults;
    }
}
```

**Figura 13:** Función detached

Complejidad Ciclomática:

Aplicada de dos formas.

**V (G)** = *a* - *n* + 2, siendo *a* el número de arcos o aristas del grafo y *n* el número de nodos.
**a=11 n =10 11-10+2=3**

**V (G)** = c + 1, siendo c el número de nodos de condición.
**c=2 2+1=3**

Caminos (C) independientes:

**C1** = 1, 2, 3, 4, 5, 6, 7, 2, 8, 10.
**C2** = 1, 2, 3, 4, 9, 10.
**C3** = 1, 2, 8, 10.

Grafo (G) de flujo resultante:

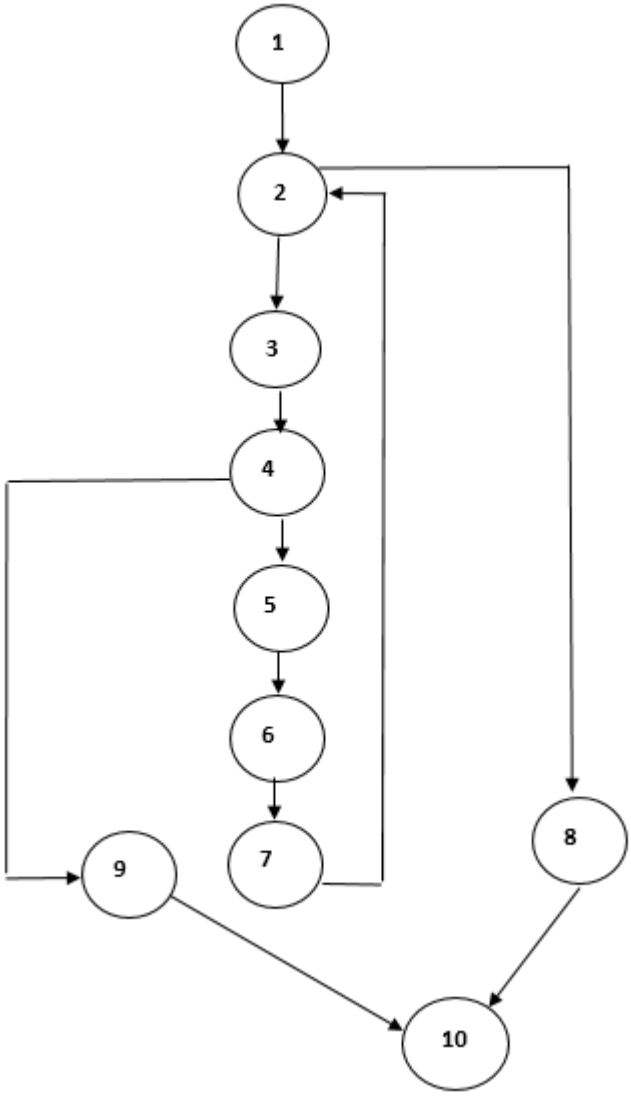



**Figura 14:** Grafo del caso de prueba de caja blanca para el método detached.

Casos de Pruebas de los caminos independientes críticos anteriormente identificados:

| **Proceso:** |
| --- |
| Realizar análisis de reglas. |
| **Caso de Prueba:** |
| Verificar Regla. |
| **Camino independiente:** |
| 1, 2, 10, 11, 12, 13, 14, 13, 16, 17. |

| **Entradas:** |
| --- |
| Presionar el botón “Verificar todo” o “Verificar extensión” se solicitan los datos de la capa de la base de datos, se le realiza la consulta correspondiente a este tipo de función y se devuelven la cantidad de errores encontrados. |
| **Resultados esperados:** |
| Se debe mostrar en el mapa la capa seleccionada dibujada con un color aleatorio representando el error encontrado. |
| **Condiciones de ejecución:** |
| Se debe haber adicionado la regla “**No debe tener geometrías multiparte”.** |

**Tabla 10:** *Caso de Prueba. Verificar regla para el camino básico: 1, 2, 10, 11, 12, 13, 14, 13, 16, 17.*

| **Proceso:** |
| --- |
| Realizar análisis de reglas. |
| **Caso de Prueba:** |
| Verificar Regla. |
| **Camino independiente:** |
| 1, 2, 8, 10. |
| **Entradas:** |
| Presionar el botón “Verificar todo” o “Verificar extensión” se hace llamada al método detached que se le pasa una geometría por parámetros no existen geometrías para recorrer la condicional en este caso el for y se retorna $geomResult en cero. |
| **Resultados esperados:** |
| No se muestran resultados debido a que el método devuelve cero. |
| **Condiciones de ejecución:** |
| Se debe haber adicionado la regla “***looseEnds*”.** |

**Tabla 11:** Caso de Prueba. Verificar regla para el camino básico: 1, 2, 8, 10.

## 3.6 Conclusiones parciales

Se expusieron los artefactos de la metodología Prodesoft necesarios para la implementación y validación de la solución. El empleo de estándares de codificación permitió una mejor legibilidad y coherencia del código facilitando el mantenimiento del mismo. Se realizaron pruebas de caja negra, específicamente la técnica partición equivalente y caja blanca, aplicando la técnica de camino básico para verificar el cumplimiento de los requisitos funcionales definidos para el sistema. A través de estas, se identificaron los errores y las fallas de la herramienta, lo que permitió brindarles especial atención para su posterior solución.

# Conclusiones Generales

- El componente desarrollado permite detectar errores topológicos en capas vectoriales de tipo línea y polígono, contribuyendo a mejorar la cartografía en la plataforma GeneSIG.
- La integración del componente a la plataforma GeneSIG, facilita las tareas del equipo de trabajo, permitiéndoles comprobar la topología, sin tener que acudir a otras herramientas, lo que permitirá ahorrar tiempo y esfuerzo durante el transcurso del desarrollo de soluciones SIG.
- El componente desarrollado cumple con las expectativas propuestas en la investigación ya que permite visualizar los errores detectados en las capas cargadas, mejorando el resultado final en las soluciones SIG que se desarrollan en la plataforma GeneSIG; esto se comprobó mediante pruebas de caja negra y caja blanca.

## Recomendaciones

Una vez vencido el objetivo de la investigación, y teniendo en cuenta la experiencia obtenida, el autor recomienda:

Realizar un estudio de otras reglas que pueden ser añadidas posteriormente al componente. A continuación, se mencionan algunas.

**No debe intersecarse:** Requiere que las entidades de línea desde la misma clase (o subtipo) de entidad no se crucen ni se superpongan entre sí.

**No debe intersecarse con:** Requiere que las entidades de línea de una clase (o subtipo) de entidad no se crucen ni se superpongan las líneas de otra clase (o subtipo) de entidad con otras.

**Debe estar dentro:** Requiere que una línea esté contenida en los límites de una entidad de área.

**Debe estar cubierto por la clase de entidad:** Requiere que un polígono en una clase (o subtipo) de entidad comparta toda su área con los polígonos en otra clase (o subtipo) de entidad.

**Deben cubrirse entre sí:** Requiere que los polígonos en una clase (o subtipo) de entidad compartan toda su área con los polígonos de otra clase (o subtipo) de entidad.

## Glosario de términos

**CartoWeb:** Es una aplicación construida en PHP sobre UMN MapServer que explota AJAX. Su característica más diferenciadora respecto a otros proyectos de clientes Web ligeros sobre MapServer, es que ofrece un framework que ha sido diseñado con una arquitectura bastante modular y escalable.

**W3C:** es un consorcio internacional que produce recomendaciones para la World Wide Web (WWW).

**WFS:** *Web Feature Service*. Es un servicio estándar definido por el OGC que ofrece una interfaz de comunicación que permite interactuar con los mapas como por ejemplo editar la imagen o analizar la imagen siguiendo criterios geográficos.

**WMS:** *Web Map Service*. Es un servicio definido por el OGC que produce mapas de datos referenciados espacialmente, de forma dinámica a partir de información geográfica. Este estándar internacional define un "mapa" como una representación de la información geográfica en forma de un archivo de imagen digital conveniente para la exhibición en una pantalla de ordenador.

**OGC:** *Open Geospatial Consortium*. Entidad dedicada a la creación de estándares entre las diferentes empresas del sector que posibiliten la interoperación de sus sistemas de geoprocesamiento y facilitar el intercambio de la información geográfica en beneficio de los usuarios.

**CASE:** *Computer Aided Software Engineering.* Herramienta que brindan asistencia a los analistas, ingenieros de software y desarrolladores, durante el ciclo de vida de un proyecto.

***GUI:*** *Graphical User Interface.* Es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

**HTML:** *HyperText Markup Language.* Es el lenguaje que se emplea para el desarrollo de páginas de internet. Está compuesto por una serie de etiquetas que el navegador interpreta y da forma en la pantalla. Dispone de etiquetas para imágenes, hipervínculos que nos permiten dirigirnos a otras páginas, saltos de línea, listas, tablas, etc.

**BSD:** *Berkeley Software Distribution.* Es un sistema operativo derivado del sistema Unix nacido a partir de los aportes realizados a ese sistema por la Universidad de California en Berkeley.

**DHTML:** *Dynamic Hypertext Markup Languagees.* Es un término colectivo que sirve para designar una combinación de nuevas etiquetas del lenguaje HTML y nuevas opciones de estilo y programación que

permiten crear páginas web más dinámicas y animadas, a la vez que ofrecen una mayor interactividad por parte del usuario que las versiones previas de HTML.

**DOM:** *Document Objet Model*. Es una interfaz de programación de aplicaciones (API) para documentos HTML y XML. Define la estructura lógica de los documentos y el modo en que se accede y manipula un documento.

**AJAX:** *Asynchronous JavaScript and XML.* Es una técnica que permite la comunicación asíncrona entre un servidor y un navegador en formato XML mediante programas escritos en JavaScript, intercambia información entre el servidor y el cliente sin la necesidad de recargar la página.

# Bibliografía

**Affairs, Department of Land. 2009.** *Understanding topology in vector data.* South Africa : s.n., 2009.

**Alonso, Diego. 2015.** mappingGIS. [En línea] 12 de 3 de 2015. [Citado el: 25 de 11 de 2015.] http://mappinggis.com/2015/03/correccion-de-topologia-en-qgis/.

**Camacho, E y Nuñez, F, Cardeso, G. 2004.** *Arquitecturas de Software, Guías de Estudio.* 2004.

**Cantero, Iñaki Kareaga. 2008.** *Diseño y Desarrollo de herramientas SIG.* Catalunya : s.n., 2008.

**Clifton, Cabello Albino. 2007.** EL MUNDO LINUX. [En línea] 2007. [Citado el: 21 de 11 de 2015.] http://www.elmundolinux.com/apachelinux.php.

**Cuervo, M. C. & Moreno , O.Y. 2005.** *Herramientas Libres para modelar software.* 2005.

**EVA.** EVA. Entorno Virtual de Aprendizaje. [En línea] [Citado el: 25 de 11 de 2015.] http://eva.uci.cu/file.php/161/Documentos/Materiales_complementarios/UD_1_Procesos/Metodologias/RUP/RUP.pdf.

**Frederick, Shea Steve y Ramsay Colin, Blade. 2008.** *Learning ExtJS.* 2008.

**Grupo gvSIG. 2016.** Sitio Oficial de gvSIG. [En línea] 2016. [Citado el: 25 de 11 de 2015.] http://www.gvsig.com.

**Hanze, Xiomara y Heredia Guerrero, Beatriz. 2008.** *Estudio de PHP Y MYSQL para el desarrollo del portal Web para el municipio de Esmeraldas.* Escuela Superior Politécnica de Chimborazo. Esmeraldas : s.n., 2008. Tesis de grado.

**ICDE. 2013.** Infraestructura Colombiana de Datos Espaciales. [En línea] 2013. [Citado el: 25 de 11 de 2015.] http://www.icde.org.co.

**Inc Research Institute ArcGIS. 2016.** ArcGIS for Desktop. [En línea] 2016. [Citado el: 25 de 11 de 2015.] https://desktop.arcgis.com/es/desktop/latest/manage-data/topologies/topology-in-arcgis.htm.

**Kernighan, Brian W' y Pike, Rob. 1999.** *The Practice of programming.* 1999. ISBN 0-201 -6 1586-X.

**Kropla, Bill. 2005.** *Beginning MapServer. Open Source GIS Development.* s.l. : aprees, 2005.

**Larman, Craig. 2003.** *UML y Patrones.* 2003.

**León, Carmen. 2014.** *Trabajo de Cartografía, Instituto Universitario Politécnico "Santiago Mariño".* Venezuela : s.n., 2014.

**Martinez, Guerrero Rafael. 2013.** PostgresSQL. [En línea] 2013. [Citado el: 17 de 4 de 2016.] www.postgresql.org.en.

**Miller, William R. 2012.** *Introducing Geodesign.* 2012.

**Morales, Aurelio. 2015.** mappingGIS. [En línea] 22 de 12 de 2015. [Citado el: 25 de 11 de 2015.] http://mappinggis.com/2012/11/por-que-utilizar-openlayers-y-geoext/.

**NCGIA. 2008.** Copade. [En línea] 2008. [Citado el: 19 de 10 de 2015.] http://www3.neuquen.gov.ar/copade/contenido.aspx?Id=ide-1-5.

**NetBeans. 2016.** NetBeans. [En línea] 2016. [Citado el: 17 de 4 de 2016.] https://netbeans.org/features/.

**Novell. 2011.** openSUSE. [En línea] 2011. [Citado el: 21 de 11 de 2015.] https://es.opensuse.org/Apache.

**Olaya, Victor. 2011.** *Sistemas de Información Geográfica.* s.l. : OpenLibra, 2011.

**Oré B, Alexander. 2009.** Quality Assurance & Software Testing. [En línea] 2009. [Citado el: 3 de mayo de 2015.] http://www.calidadysoftware.com/testing/pruebas_unitarias1.php.

**Pressman, Roger. 2005.** *Ingeniería de Software.* 2005.

**—. 2002.** *Ingeniería del Software: Un enfoque práctico.* España : McGraw-Hill Companies, 2002. 5ta Edición.

**Ramsey, Paul. 2015.** *Manual PostGIS.* 2015. 2.2.0.

**Reynoso, Carlos y Kicillof, Nicolás. 2004.** *Estilos y Patrones en la Estrategia de Arquitectura.* Buenos Aires : s.n., 2004.

**Robaina, I. 2008.** *Propuesta del Diseño Arquitectónico del Simulador de Sistemas Biológicos.* La Habana : s.n., 2008.

**Team Quantum GIS Development. 2016.** Quantum GIS Geographic Information. [En línea] 2016. [Citado el: 24 de 11 de 2015.] http://www.qgis.org/es/site/.

**UCID, Unidad de Compatibilización Integración y Desarrollo De Software para la Defensa. 2012.** *Proceso de Desarrollo y Gestión de Proyectos de Software.* 2012.

**Vargas, Velasquez Giovanni. 2015.** *INVESTIGANDO EN GEOMÁTICA.* Bogotá : s.n., 2015.

**Visconti, Marcello y Astudillo, Hernán. 2005.** *Fundamentos de Ingeniería de Software.* 2005.

# Anexos

**Anexo:** A continuación, se presentan imágenes del componente desarrollado.

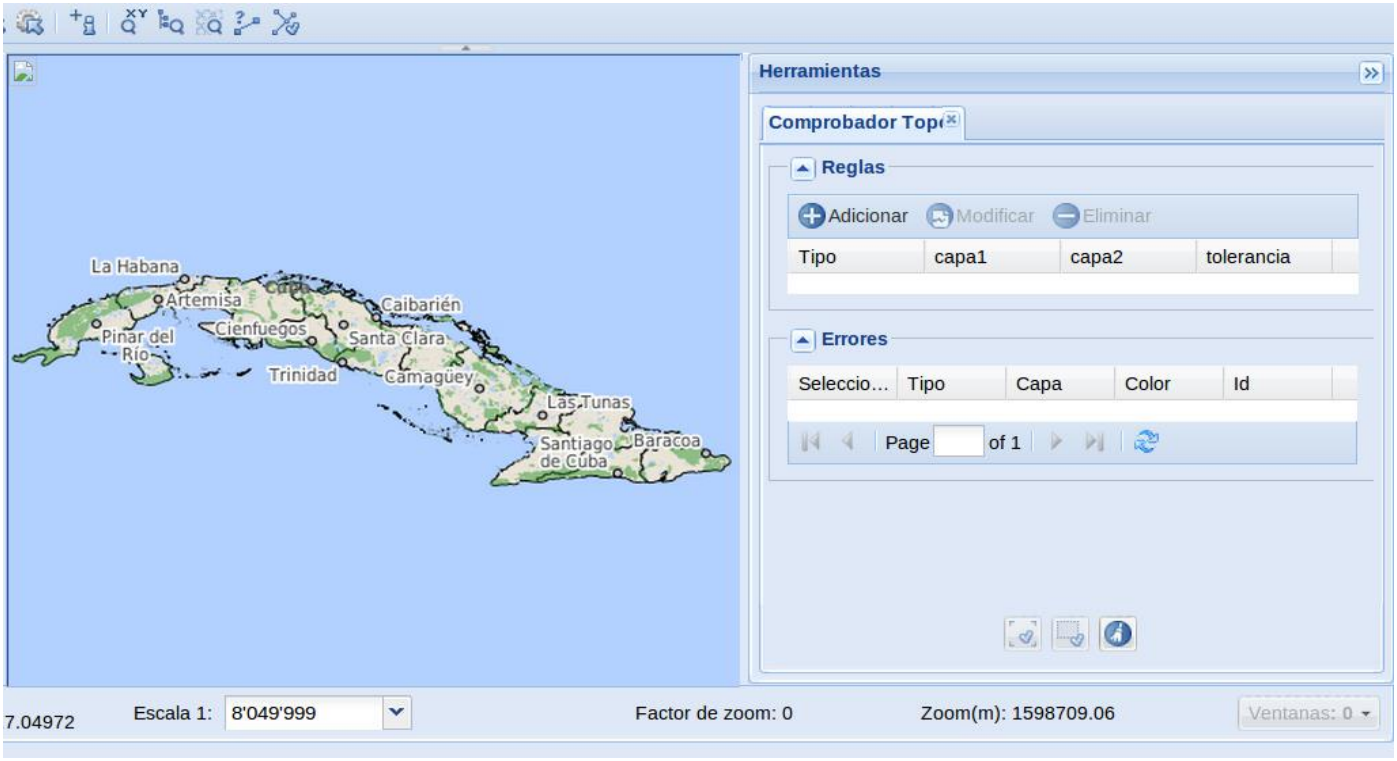



**Figura 15:** Interfaz del componente Comprobador topológico.

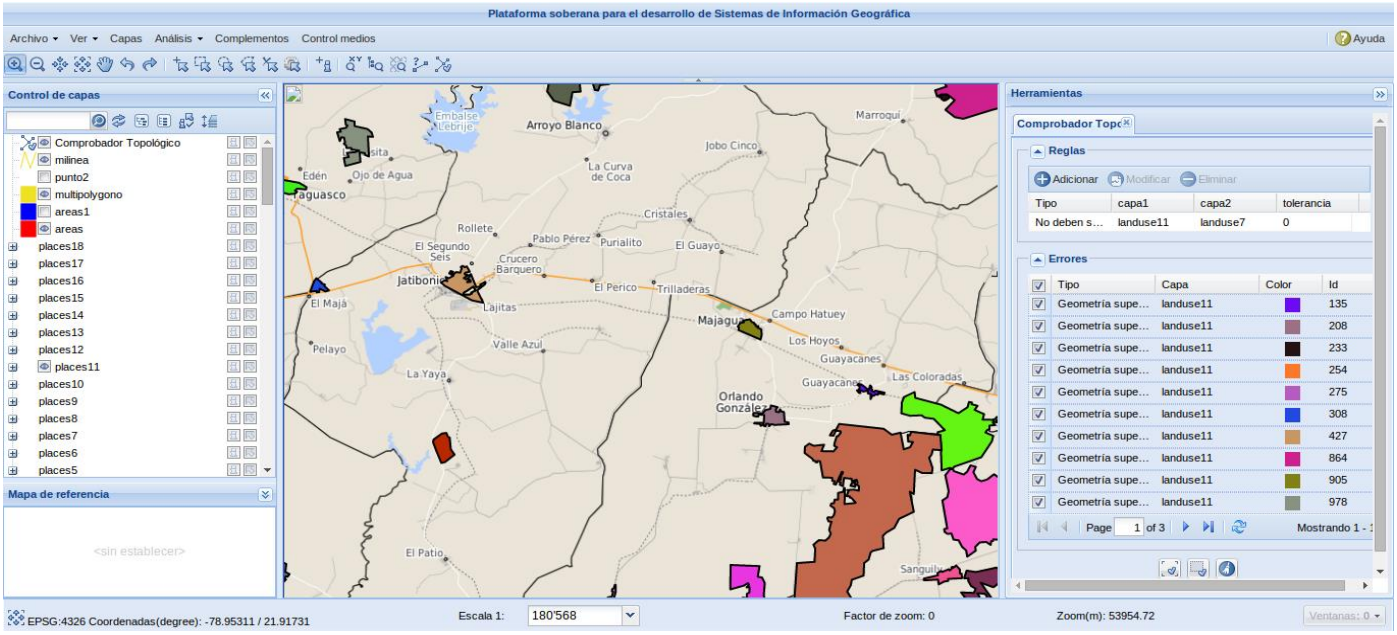



**Figura 16:** Interfaz que muestra los errores de la regla: No se debe superponer con, para capas de tipo polígono.

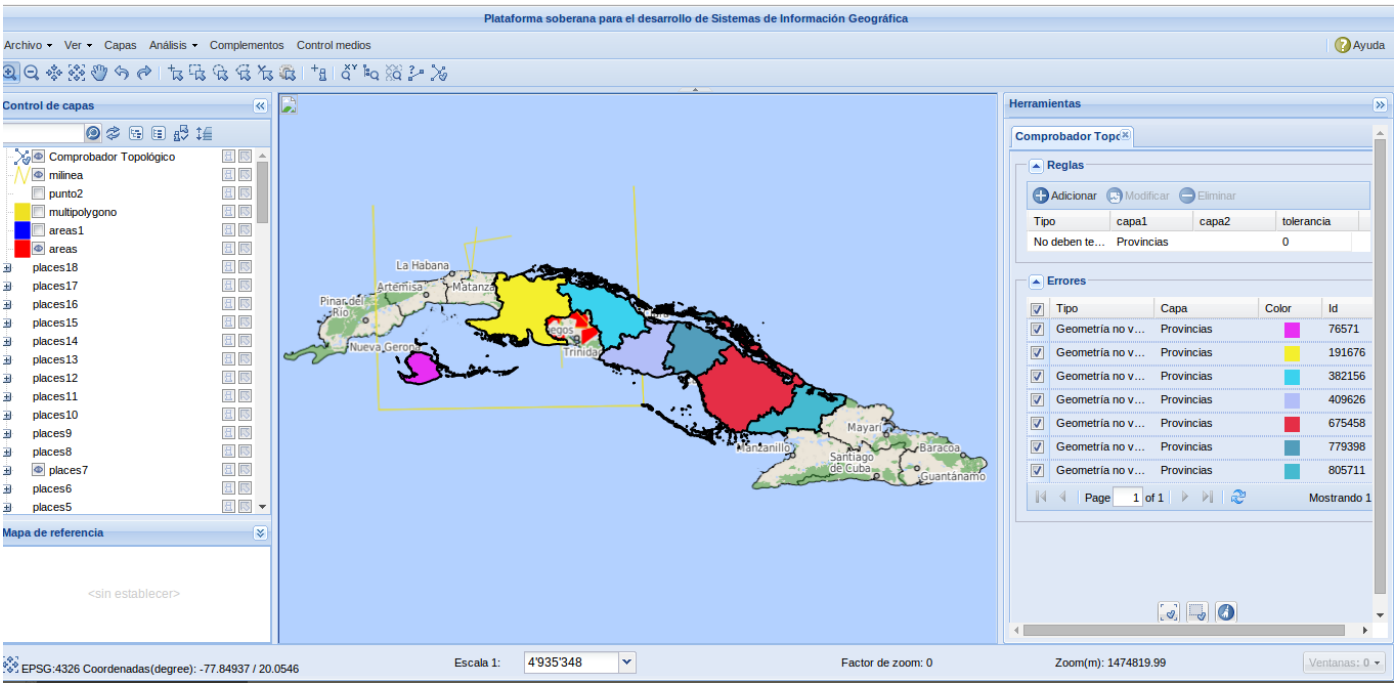



**Figura 17:** Interfaz que muestra los errores de la regla: No debe tener geometrías inválidas, para capas de tipo polígono.

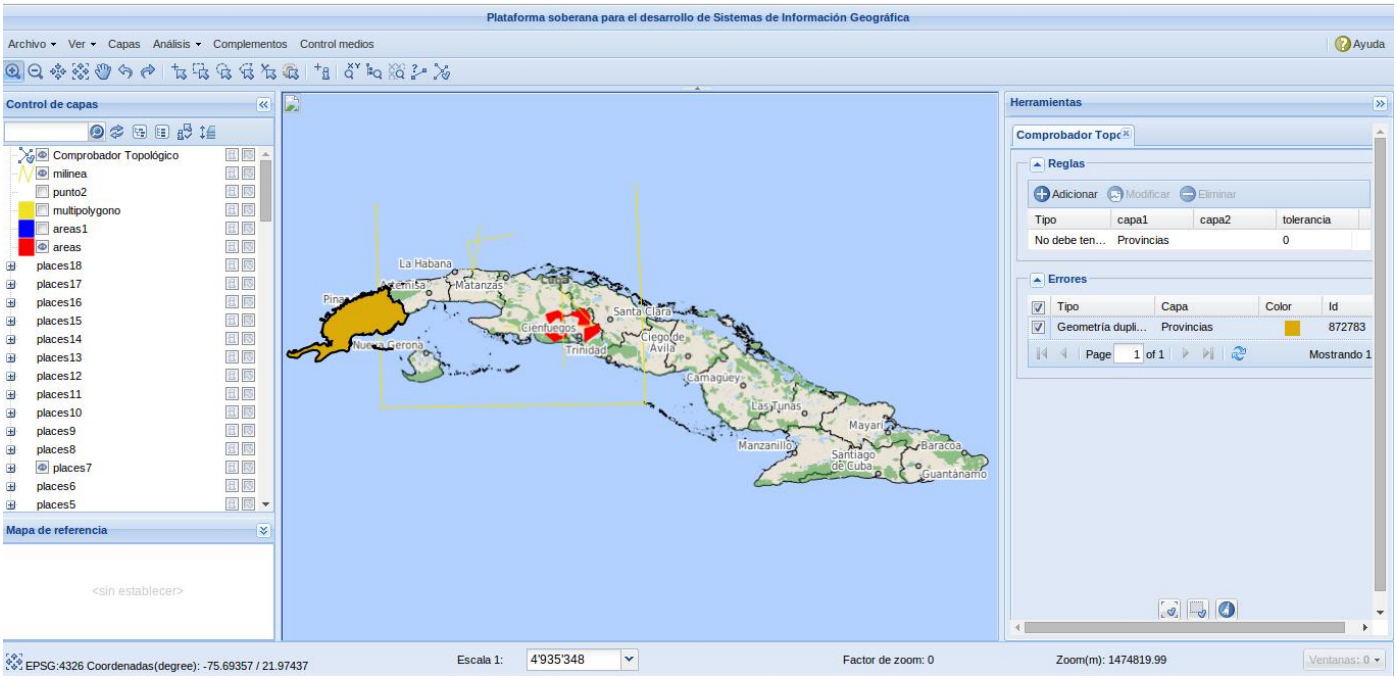



**Figura 18:** Interfaz que muestra los errores de la regla: No debe tener duplicados, para capas de tipo polígono.

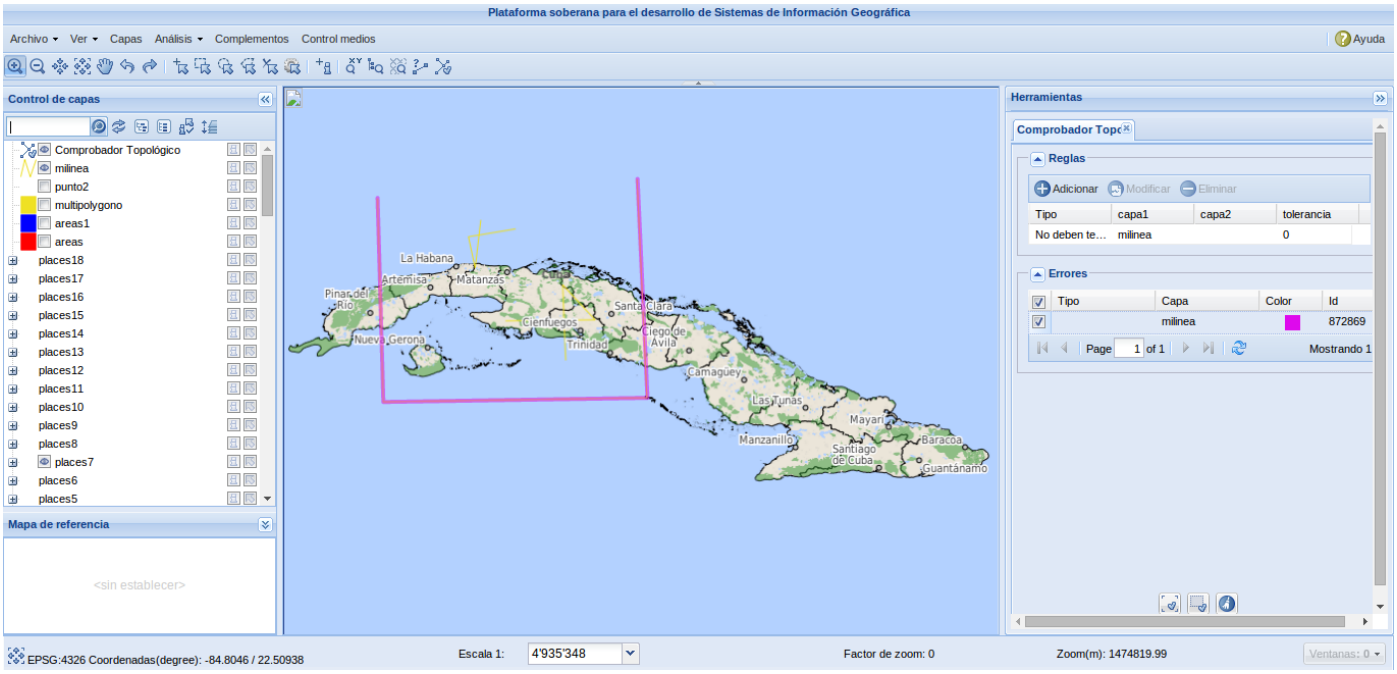



**Figura 19:** Interfaz que muestra los errores de la regla: No debe tener geometrías multiparte, para capas de tipo línea.

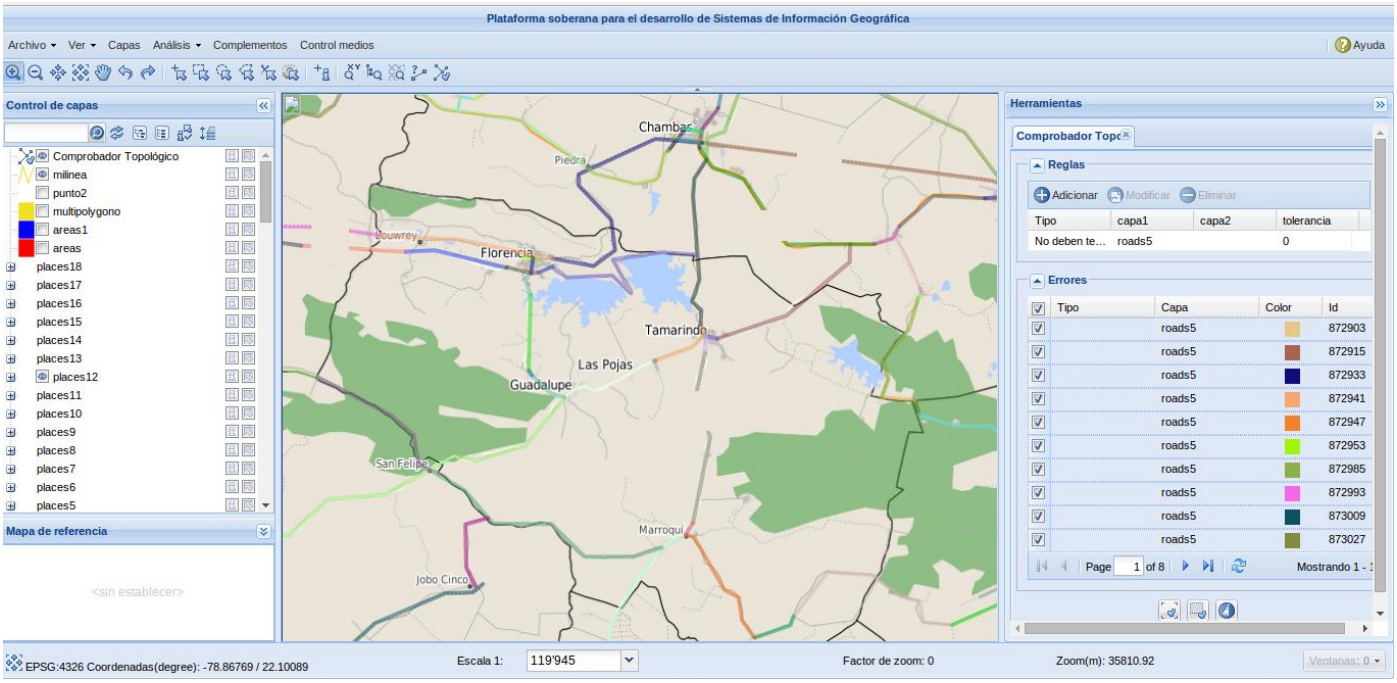



**Figura 20:** Interfaz que muestra los errores de la regla: No debe tener extremos sueltos, para capas de tipo línea.