03063

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

UNIDAD ACADEMICA DE LOS CICLOS PROFESIONAL Y DE POSGRADO DEL COLEGIO DE CIENCIAS Y HUMANIDADES

INSTITUTO DE INVESTIGACIONES EN MATEMATICAS APLICADAS Y SISTEMAS

CONSTRUCCION DE UN SISTEMA DE SOFTWARE PARA LA ELABORACION Y PROCESAMIENTO DE CARTOGRAFIA DIGITAL UTILIZANDO LA METODOLOGIA ORIENTADA A OBJETOS

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS DE LA COMPUTACION

P R E S E N T A :

HERMES ROBLES BERUMEN

DIRECTOR: DRA. HANNA OKTABA

MEXICO, D. F. ABRIL 1997.

TESIS CON FALLA DE ORIGEN

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

PAGINACION VARIA

COMPLETA LA INFORMACION

Dedicatoria

A la memoria de mi padre Luciano, con una especial dedicatoria por que el se dedicaba a la topografía.

A mi madre Socorro por su dedicación y apoyo que siempre nos ha brindado para lograr nuestras metas.

A mis hermanos:
María E. Guadalupe
Ciro
Elena Emma
Ana Isabel
Carlos
Ruth y
Susana
Con la esperanza de que puedan lograr sus anhelos

A mis sobrinos como un ejemplo a seguir por el bien de la familia

A mis Maestros con el afecto del estudiante por haber tenido la oportunidad de recibir su experiencia.

A mis compañeros y amigos por su amistad y ayuda a lo largo del camino.

Agradecimientos

Las deudas de gratitud que e incurrido en la preparación de este trabajo son numerosas son muchas las personas que me han ayudado en diferentes caminos.

Agradezco en especial a la Dra. Hanna Oktaba por su invaluable apoyo en la dirección del presente trabajo.

Al Dr. Román Alvarez Béjar, M. en C. Guadalupe Ibargüengoitia G. y Ing Mario Rodríguez Manzanera por sus sugerencias y haber aceptado ser mis sinodales en el examen de grado.

Al M. en C. Alfredo Cortes por iniciarme en área de los sistemas de información geográfica.

Por último, deseo agradecer a mi familia, su paciencia y apoyo al escribir este trabajo.

Hermes Robles Berumen
México D.F. Abril de 1997

Contenido

Segunda parte desarrollo del sistema

Lista de figuras y tablas

Introducción

La programación de sistemas en sus inicios consistía en escribir programas no muy grandes, debido a las características del hardware con el cual se contaba en ese tiempo. En las décadas de los sesentas y setentas se inicia un cambio dramático en los costos del hardware y capacidades de cómputo. Con el hardware económico y más eficiente se logró atacar otros problemas de mayor complejidad. Los lenguajes de programación de alto nivel se convirtieron rápidamente en herramientas útiles, proporcionando a los desarrolladores la habilidad para construir sistemas cada vez más complejos.

La aparición del lenguaje de programación *Simula* en 1967 bajo el nombre de *Simula 67*, por Ole-Johan Dahl y Krysten Nygaard de la Universidad de Oslo y del Centro de Cómputo de Noruega; introduce importantes avances y da la pauta para la creación de otro paradigma conocido como *orientado a objetos*. Simula es un lenguaje de propósito general y es considerado una extensión del lenguaje Algol 60 pero orientado a objetos (Meyer, 1988).

Durante la década de los sesentas se propusieron varios métodos de análisis y diseño para el manejo de la complejidad creciente del software. En los últimos años las metodologías de modelado orientado a objetos están revolucionando el campo de la ingeniería de software, debido a las ventajas que ofrece.

El objetivo de este trabajo es estudiar la metodología de análisis, diseño y programación orientada a objetos a partir de un caso práctico, el cual consiste en desarrollar un módulo de un sistema de información geográfica, utilizado para capturar la información contenida en los mapas en formato vectorial. Por medio del desarrollo del sistema se observarán las facilidades que ofrece la metodología orientada a objetos en la construcción de sistemas de software.

Las aportaciones de esta tesis se pueden resumir en:

- Una revisión detallada y ordenada del proceso de desarrollo de un sistema orientado a objetos, incorporando comentarios personales.
- Desde el punto de vista de la cartografía, un modelo orientado a objetos para la representación digital de los elementos que forman un mapa.
- Un prototipo, utilizado para elaborar y procesar cartografía digital.

Existen diferentes metodologías de desarrollo de análisis y diseño orientada a objetos; para este trabajo se utilizó la metodología de Booch [1993]; pero también se incorporó notación de Coad y Yourdon [1991]. Se seleccionaron estas por abordar el paradigma con gran detalle. El lenguaje de programación utilizado para la implantación del prototipo propuesto fue C++.

Para una mejor descripción de la tesis esta se encuentra dividida en dos partes principales:

La primera parte esta formada por los capítulos 1 y 2. El capítulo 1 contiene los conceptos básicos sobre los diferentes modelos de representación de la información espaciales en forma digital. El capítulo 2 incluye una introducción a los sistemas de información geográfica. El objetivo de la primera parte es estudiar la representación digital de los mapas, así como los sistemas de software que se utilizan en el análisis de la información espacial, permitiendo de esta manera encontrar una forma de representación digital para ser utilizada en el prototipo propuesto.

La segunda parte se compone de los capítulos del 3 al 8. El capítulo 3 comienza con una breve descripción de la metodología orientada a objetos de Booch. Los capítulos restantes describen en forma práctica y detallada la fase de especificación, análisis, diseño, implantación y mantenimientodel prototipo propuesto en la tesis. Como prueba del prototipo se digitizó una parte de un mapa de Ciudad Universitaria y un de la Ciudad de Zacatecas.

Primera parte

La cartografía digital

1 Representación digital de los mapas

La adquisición de datos espaciales y su análisis es de gran importancia en diversas disciplinas. En las áreas de catastro y planeación urbana se necesita información detallada de la distribución de los recursos en las regiones y ciudades. Los ingenieros civiles utilizan los mapas para obtener datos sobre los caminos y canales para estimar costo de construcción. Los diferentes servicios como el suministro de agua, energía eléctrica, líneas de teléfonos pueden ser planeados y menejados por medio de mapas.

En este capítulo se describen diferentes formas de representación digital de la información contenida en los mapas; con el objetivo de servir de línea base para entender y descubrir los elementos que ayuden a modelar el sistema propuesto en esta tesis.

1.1 Los mapas

Desde las primeras civilizaciones hasta los tiempos modernos, los mapas han sido documentos importantes. El mapa más antiguo conocido en nuestros días, se descubrió en las excavaciones de las ruinas de la ciudad de Gasur a unos 300 Km. al norte de Babilonia; el cual se conserva en el Museo Semítico de la Universidad de Hardvard. Los investigadores encontraron una placa de barro recocido que representa el valle de un río, seguramente el Eufrates con montañas a cada lado, indicadas en forma de escamas de pescado; el río desemboca por un delta de tres brazos en un lago o mar; pero lo más probable es que antes de la elaboración de este mapa la gente ya los utilizaba. Hodgkiss [1981] comenta que originalmente se elaboraban con fines de describir lugares remotos, para ayudar a los navegantes y estrategas militares.

La información de los fenómenos geográficos como: ciudades, ríos, regiones de cultivo y otros es adquirida por navegantes, geógrafos, topógrafos, cartógrafos y agrimensores y su representación gráfica forman los mapas. El área de estudio que se dedica a la construcción de mapas es la *cartografía*, Alonso [1986] la define como una arte o ciencia que se ocupa de la representación gráfica de la superficie curva de la tierra. Dicha representación puede efectuarse en dos dimensiones (planos) o en tres dimensiones (esfera), siendo la primera la más utilizada.

Robinson et al [1978] establece que los mapas son herramientas indispensables, en donde se describen las relaciones espaciales de los objetos del mundo real y comenta que tienen las siguientes características:

- Son reducciones. Son documentos pequeños en proporción a las áreas que representan. En cada mapa se expresan relaciones de lo que existe en el área y lo que se está representando. La relación de escala y la manera de dibujar la información son de primordial importancia para el uso que tendrá.
- Están en una proyección. En todos los mapas con motivo de dibujar una superficie de la tierra sobre un plano se usa una proyección. Existen diferentes tipos de proyecciones, seleccionar una en particular tiene un impacto sobre cómo puede ser utilizado.
- Son abstracciones de la realidad. Son representaciones de regiones de la Tierra por parte de los cartógrafos. La información que está impresa en un mapa, hace que sea utilizado para fines específicos y además se encuentra clasificada y simplificada para proporcionar a los usuarios un fácil manejo.
- Contiene símbolos que representan elementos de la realidad. Son pocos los símbolos que tienen una aceptación universal, pero algunos mapas son usados para estandarizar un conjunto de símbolos.
- Además de los símbolos se pueden utilizar otros elementos para la representación de la información como: diferentes tipos de líneas, colores, texturas y patrones. Generalmente un mapa incluye una *leyenda*, utilizada para ligar los atributos no espaciales a las entidades espaciales.

Los cartógrafos para poder desempeñar su función son hombres de ciencia y artistas al mismo tiempo; deben conocer perfectamente el modelo que han de representar (la Tierra), y además tener la abstracción suficiente para suprimir detalles que no sean relevantes para los fines que se pretenda usar el mapa. La selección acertada de símbolos y colores para dibujar los elementos que van a constituir el mapa, dependerán más del sentido artístico que de la preparación científica del cartógrafo.

Un mapa en su aceptación más elemental es una representación convencional de la superficie terrestre vista desde arriba, al que se le agregan rótulos para identificar los detalles más importantes. Existen diferentes maneras de dibujar la información espacial en un mapa; para su confección se consideran los elementos de escala, el sistema de proyección, los símbolos, el rotulado, el título, los recuadros y los detalles complementarios.

Alonso [1986], describe una clasificación general de los mapas basada en su escala y su contenido:

Mapas generales por escala:

- Mapas topográficos a escala grande con información general.
- Mapas cartográficos que representan grandes regiones, países o continentes a pequeña escala (los atlas pertenecen a esta clase).
- Mapas del mundo entero (mapamundis).

Mapas especiales basados por su contenido:

- Mapas políticos.
- Mapas urbanos.
- Mapas de comunicación.
- Mapas científicos de diferentes clases.
- Mapas económicos y estadísticos.
- Mapas artísticos y de anuncio o reclamo (propaganda).
- Mapas catastrales, dibujados a gran escala, para representar las parcelas de los diferentes propietarios con cultivo.
- Mapas temáticos.
- Cartas para uso en la navegación marítima y aérea.

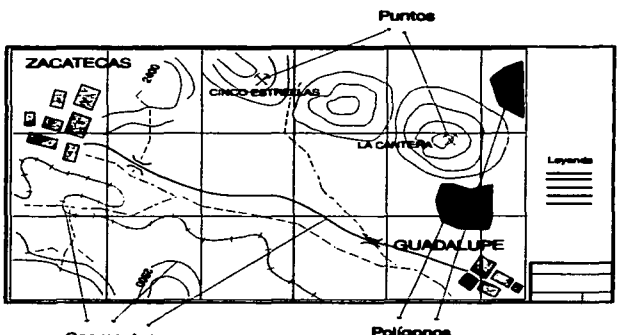


Figura 1.1 Un mapa.

El término de *mapa temático* se usa ampliamente para los mapas que representan diferentes tipos de fenómenos (Fisher, 1978; Hodkiss, 1981), por ejemplo: los que describen propiedades específicas del suelo, como es el caso de la distribución de los valores del pH; la variación de la presión del aire y otras.

El tema de un mapa temático puede ser clasificado como *cualitativo* y/o *cuantitativo*. Un ejemplo de uno cualitativo es cuando se describen las clases de usos del suelo, y uno cuantitativo es cuando se representa la profundidad del nivel freático en una carta hidrológica.

En los temas cualitativos o cuantitativos para la representación de la información se usan áreas que tienen valores iguales, estas reciben el nombre de *isopletas*. Para los temas cuantitativos los datos pueden ser también mapeados en *isolíneas* que son líneas que unen los puntos de igual valor; como los mapas topográficos o de curvas de nivel que consisten de un conjunto de líneas que tienen la misma elevación con respecto al nivel del mar.

Otro tipo de información asociada a los elementos de un mapa temático, son las relaciones topológicas y algunas propiedades geométricas como: distancia, dirección, conectividad y aproximación.

Para el propósito de esta tesis se utilizará la definición de un *mapa* de Burrough [1986], el cual lo define como un conjunto de puntos, líneas y polígonos caracterizados tanto por su localización en el espacio en un sistema de coordenadas, así como por los atributos no espaciales; usualmente se representa en dos dimensiones pero no existe razón alguna para excluirlo de una representación de tres dimensiones, excepto por la dificultad de dibujarlo sobre hojas de papel. Ver Figura 1.1.

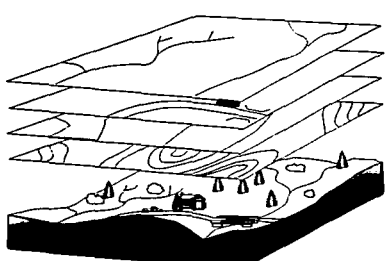


Figura 1.2 El modelo de planos sobrepuestos. El mundo real es clasificado por una serie de capas, donde cada una de ellas cubre un aspecto.

En la cartografía para la representación de los mapas se utiliza el modelo de *planos sobrepuestos*, éste consiste en clasificar los diferentes temas contenidos en un mapa en una serie de capas de información. Por ejemplo: si una región de la Tierra se quiere cartografiar, para facilitar el manejo de la información, se separa por: rasgos topográficos, de uso del suelo, culturales, hidrológicos y otros. Ver Figura 1.2.

Este tipo de modelado tiene la ventaja de adquirir y organizar los datos de diferentes fuentes con mayor facilidad. En el análisis se puede manipular las capas de información en forma separada o combinandolas.

1.2 Modelos de representación

En el mundo real se exhiben fenómenos que pueden ser representados en un mapa; los *datos* son hechos verificables acerca del mundo real. La *información* es un conjunto de datos organizados que revelan un patrón definido y facilitan la búsqueda de los datos (Bonham [1994]). En este trabajo se definió *objeto espacial*, como aquel que puede ser dibujado en un sistema de proyección de coordenadas definido y además tiene asociado atributos temáticos.

La manera en que los datos geográficos son manipulados por las personas es muy diferente al utilizado por las computadoras, debido que para almacenar o recuperar un dato en una computadora se utilizan dispositivos magnéticos que se direccionan de alguna forma especifica.

Por otro lado las estructuras de datos utilizadas para el manejo de la información espacial en una computadora son complejas, debido a que los datos contienen atributos espaciales y relaciones topológicas entre ellos, resultando con frecuencia que la representación de los objetos geográficos no son eficientes.

La manera de definir y organizar consistentemente los datos espaciales en un formato digital es por medio de un *modelo de datos*. La selección de un modelo en particular se basa en cómo los datos son: capturados, manipulados, almacenados e interpretados; los modelos más conocidos para organizar los datos de los mapas son: el modelo *raster* y el *vectorial*.

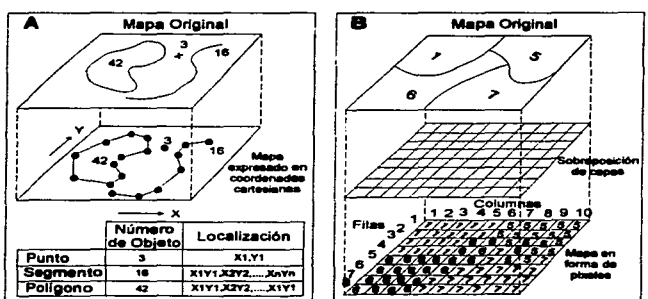
1.2.1 El modelo *raster*

El *modelo de datos raster* consiste de una malla de celdas, donde cada celda se conoce como un *pixel*. Para referenciar a un pixel se usa su número de fila y columna, el cual contiene el valor del atributo que está siendo mapeado. Este tipo de modelo es de fácil manejo para una computadora por medio de una estructura de tipo *arreglo*, utilizada para almacenar y manipular los pixeles. En la representación de un pixel por lo general se usa un byte que equivale a ocho bits u ocho digitos binarios de almacenamiento.

La resolución de una imagen *raster* se mide en lo que cubre un pixel sobre la superficie de la tierra. Una resolución de 100 m. en una área de 100 Km. cuadrados, requiere una arreglo de 1,000 filas por 1,000 columnas, es decir, un millón de pixeles. Para una resolución de 10 m. en la misma área se necesitan 10,000 filas por 10,000 columnas, que equivalen a cien millones de pixeles; esta es una cantidad considerable de espacio para almacenar un mapa en este formato. Debido a que los requerimientos de almacenamiento aumentan geométricamente con el incremento de la resolución, en este modelo se pone mucho énfasis en la compresión de la información.

El modelo de tipo *raster* es ampliamente utilizado para representar superficies en dos dimensiones; es muy adecuado en el manejo de propiedades continuas porque los datos espaciales se pueden subdividir. Otras de las ventajas que tiene el modelo es el manejo de la sobreposición de capas de información para hacer combinaciones de ellas, así como la aplicación de los algoritmos utilizados en el área de *procesamiento de imágenes*. En lo que respecta a las consultas espaciales el modelo es adecuado para implantar búsquedas de objetos adyacentes.

La representación de un punto dentro de un mapa es por medio de un pixel, los segmentos por una cadena de pixeles conectados y las áreas por un conjunto de pixel agrupados con las mismas características. La estimación de las propiedades geométricas como la longitud, el área y otras se realiza con base en las proporciones que existen entre los pixeles y los objetos del mundo real.



	Número de Objeto	Localización
Punto	3	X1,Y1
Segmento	16	X1Y1,X2Y2,.....,XnYn
Polígono	42	X1Y1,X2Y2,.....,X1Y1

Figura 1.3 Bonham [1994], modelos de representación digital de objetos espaciales. A. Modelo vectorial. B. Modelo *raster*.

El almacenamiento de los atributos temáticos en este modelo, se puede usar el valor pixel como un apuntador a una tabla de atributos, en lugar de tener cada uno de ellos en una capa separada de información; esto es particularmente ventajoso para la digitalización de mapas, sobre todo donde el tema puede ser almacenado como una tabla, la desventaja que se tiene es que en algunos casos resulta insuficiente un byte para representar un pixel, debido a que los únicos registros que se pueden direccionar en una tabla son tan sólo 256. Si es requerido mas direccionamiento se deberán usar dos o mas bytes por cada pixel.

1.2.2 El modelo vectorial

El modelo vectorial como su nombre lo dice, se basa en describir los objetos por medio de coordenadas sobre un plano cartesiano o en otra proyección, en lugar de un escalar. La representación de los puntos en un mapa es por un par de coordenadas, los segmentos como una cadena de puntos, y los polígonos como un conjunto de segmentos que describen una área cerrada. Este modelo también se conoce como el modelo de *spaghetti* (Bonham, 1994).

El modelo vectorial es adecuado para la representación de los mapas en forma de puntos, segmentos y polígonos y para la aplicación de operaciones como: escalamientos, cambios de proyección y despliegue de los datos sobre un monitor de video o la impresión en un *plotter*. Además en el despliegue e impresión los puntos pueden ser dibujados con diferentes símbolos, los segmentos pueden tener diferentes colores y anchos y los polígonos pueden ser dibujados con patrones y colores que dependen de sus atributos temáticos.

Al contrario del modelo *raster*, los atributos temáticos, los topológicos y los espaciales de los objetos pueden ser almacenados en un mismo archivo. Pero las estructuras de datos requeridas para su manejo en algunos casos resultan ser más complejas de implementar en una computadora, por que se deben describir explícitamente las relaciones topológicas entre los objetos. Las diferencias entre los modelo *raster* y vectorial se muestran gráficamente en la Figura 1.3.

Los objetos espaciales para su representación en este modelo pueden ser agrupados por su dimensión espacial: un punto no tiene dimensión, una línea tiene una, una área tiene dos y un volumen tres dimensiones. Bonham [1994] comenta que algunas superficies se considera que tienen una dimensión de 2.5 o fraccionaria, por ejemplo: la superficie formada por las curvas de elevación de una superficie de terreno (con excepción de paredes verticales de los acantilados), la superficie inferior descrita por la elevación de un manto de carbón relativamente plegado y otras. La Figura 1.4, se obtuvo de Bonham [1994], la cual describe algunos ejemplos de objetos espaciales agrupados por su dimensión espacial.

Debido que el sistema computacional propuesto en esta tesis se basa en el modelo vectorial, en las siguientes secciones se describe en detalle la representación de los objetos espaciales en este modelo. Por simplificación la representación de los diferentes objetos espaciales se agruparon sólo en: puntos, segmentos y polígonos. En el caso de los objetos que están dentro del grupo de los volúmenes, no serán comprendidos en esta tesis.

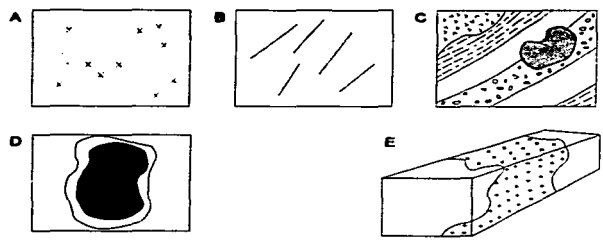


Figura 1.4 Bonham [1994], ejemplo de objetos espaciales geológicos, clasificados de acuerdo con su dimensión espacial. A. Ocurrencia de mineral puede ser mostrada como un punto (0-D) en mapas a pequeña escala. B. Los lineamientos por medio de líneas (1-D). C. Las formaciones sobre un mapa geológico se representan por medio de áreas (2-D). D. Una superficie gravitacional (2.5-D), se dibuja como un mapa de contornos. E. Un yacimiento de mineral (3-D).

1.2.2.1 Puntos

Los puntos son utilizados para representar a todos los objetos geográficos localizados por un par de coordenadas (x,y); estos pueden estar ligados a un segmento, donde reciben el nombre de *vértices*. Un segmento es una secuencia de vértices ordenados. El vértice inicial y terminal de un segmento son dos puntos especiales llamados *nodo inicial* y *nodo final*. Otra información asociada a los puntos es para desplegarlos en los monitores o en las impresoras como: símbolos, etiquetas, tipo de letras de las etiquetas y otras.

1.2.2.2 Segmentos

Los segmentos que están dibujados en un mapa son continuos, pero en el modelo vectorial para poderlos representar se utiliza una secuencia de puntos unidos por medio de líneas. Los segmentos en un mapa representan: caminos, vías de ferrocarril, curvas topográficas y otros objetos. Por lo tanto las estructuras de datos, aparte de la información temática almacenada, también deben contener información gráfica que indica la manera en cómo debe ser dibujado, por ejemplo: con línea continua o línea punteada, ancho de línea y un símbolo para relacionarlo con la leyenda del mapa. Algunos segmentos encontrados en los mapas se muestran en la Figura 1.5.

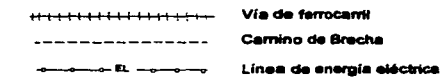


Figura 1.5 Tipos de segmentos utilizados en los mapas.

1.2.2.2.1 Redes de segmentos

Debido a que un segmento es representado como una sucesión de puntos unidos por líneas; esta representación tiene la desventaja de no contener la información necesaria para expresar las relaciones topológicas con otros segmentos, requerida para realizar análisis sobre objetos espaciales que se encuentran conectados. Algunos ejemplo donde los objetos se encuentran unidos son los ríos, los cuales describen un *drenaje*; las caminos entre ciudades y otros. Para poder modelar estos fenómenos geográficos se deberá usar una estructura topológica de tipo *red de segmentos*. La conexión entre segmentos de una red, consiste en que los segmentos estarán compuestos por dos puntos especiales: nodo inicial (es el punto donde inicia) y el nodo final (es el punto donde termina). Los nodos serán compartidos por los segmentos para establecer la conexión entre ellos. La Figura 1.6 ilustra la representación de la estructura red de segmentos.

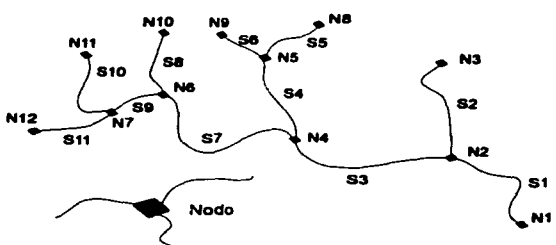


Figura 1.6 Estructura topológica de tipo *red de segmentos*.

1.2.2.3 Polígonos

Existen diferentes estructuras propuestas para la representación de un polígono en forma digital; de los trabajos de Bonham [1994], Burrougt [1986], Cook [1978, 1983], Peuker y Chrisman [1975] y Van Roessel [1987] se recopilaron las estructuras más representativas; una descripción de algunas de ellas se muestra a continuación:

1.2.2.3.1 Polígono simple

La estructura más simple para representar un polígono en forma digital es un arreglo de puntos ordenados, donde los puntos representan los vértices del polígono. Este tipo de estructura tiene la ventaja de ser fácilmente implementada en una computadora, pero tiene dos desventajas. La primera es la redundancia de datos; es decir, si un mapa consiste de un mosaico de polígonos como el que se muestra en la Figura 1.7, cuando son digitalizados los segmentos que describen los límites del polígono, se capturan dos veces, excepto cuando las lados del polígono forman un lado del mapa; esto es debido a que cada línea está formada por dos vértices y es el límite de dos polígonos adyacentes. Para un mapa, que contiene algunos polígonos formados por cientos de vértices, esta representación tiene como consecuencia que el espacio requerido para almacenar la información del mapa aumente considerablemente.

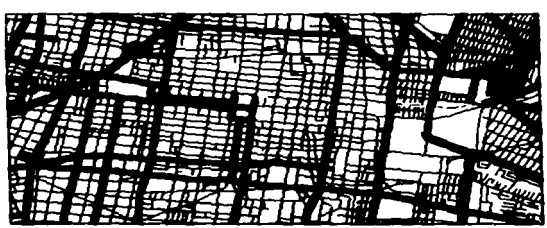


Figura 1.7 Mapa compuesto de un mosaico de polígonos.

Como consecuencia de la doble captura de un segmento, se presenta el problema de *huecos* e *invasiones*. Los huecos e invasiones, ocurren cuando se digitaliza un segmento de un polígono y luego se captura para el otro polígono, encontrándose que los dos segmentos no son iguales. Ver Figura 1.8.

La segunda desventaja de esta estructura es la ausencia de atributos topológicos, relevantes para realizar análisis de la información; por ejemplo, son utilizados en aplicaciones de geología, para encontrar los contactos entre tipos de rocas; también sirven para clasificar las unidades de un mapa a través de unir polígonos adyacentes con características similares. Para obtener la relaciones topológicas entre los objetos espaciales en esta estructura, se deben realizar cálculos que pueden ser ineficientes. Los archivos que contienen datos espaciales sin relaciones topológicas se llaman *inestructurados*.

Las dos desventajas expuestas pueden ser superadas por estructuras topológicas más complejas, así las tareas de búsqueda de los polígonos adyacentes puede ser reducida; en vez de investigar cuáles polígonos tienen vértices iguales, mejor se usan tablas que contienen las relaciones topológicas de esta manera se aumenta la eficiencia. Un ejemplo de una estructura topológica se describe en la sección 1.2.2.4.

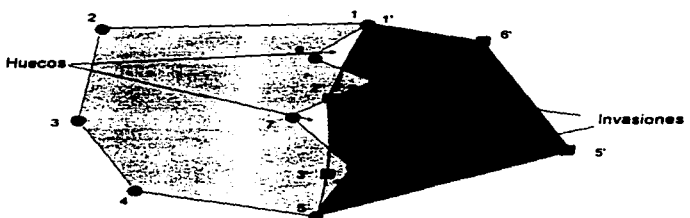


Figura 1.8 Errores que se cometen en la captura de un polígono, usando la representación de *polígono simple*.

1.2.2.3.2 Diccionario de puntos

Si todos los pares de coordenadas son numerados secuencialmente cuando son capturados y además se registran en un diccionario de datos, el cual asocia los puntos a los polígonos, se obtiene la estructura *diccionario de puntos*. Por medio de está se pueden evitar los problemas de huecos e invasiones, es decir, los límites entre los polígonos adyacentes son únicos. La Figura 1.9, muestra la representación de un polígono utilizando un diccionario de puntos.

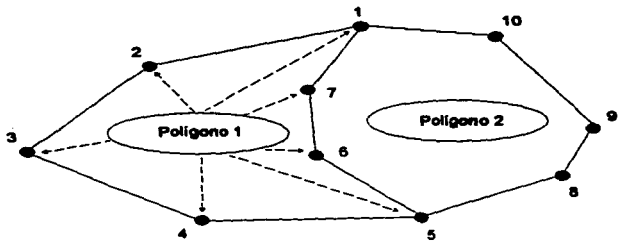


Figura 1.9 Representación de los polígonos utilizando la estructura *diccionario de puntos*.

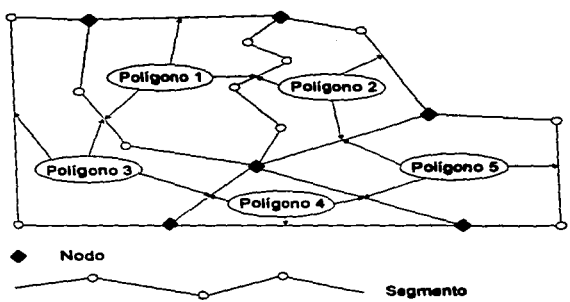


Figura 1.10 Representación de polígonos utilizando la estructura *segmento nodo*.

1.2.2.3.3 Estructura segmento nodo

Otra estructura utilizada en el manejo de los polígonos es la *segmento nodo*, a diferencia de la anterior, utiliza una estructura de datos contenedora por cada polígono, que contendrá las referencias a los segmentos. Los límites entre los polígonos son descritos por segmentos, el cual se define como una secuencia de puntos ordenados, donde los puntos terminales son *nodos*. La redundancia de la información que tiene la estructura polígono simple es evitada. Esta estructura es similar a la de red de segmentos, pero aquí los segmentos forman un ciclo. En la Figura 1.10, se puede observar esta representación.

La estructura segmento nodo, tiene la desventaja en la representación de las *islas*. Se llama isla a un polígono que está incluido dentro de otro, estas son comunes de encontrarlas en algunos mapas. El manejo de las islas se puede hacer por medio de definir explícitamente la relación topológica de *contención* entre los polígonos. En la siguiente sección se describe una estructura más completa para el manejo de las relaciones topológicas incluyendo la representación de islas.

1.2.2.4 Estructura topológica

Esta estructura fue desarrollada por Van Roessel [1987]. Debido a la difusión que tienen las bases de datos relacionales, esta estructura se basa en el modelo relacional y además se encuentra normalizada.

Bonham [1994], comenta sobre la estructura que no es eficiente para uso operacional, pero proporciona una descripción explícita de las relaciones topológicas. Bonham la describe de manera didáctica usando un ejemplo geológico:

Considérese el mapa de la Figura 1.11-A, donde afloran tres tipos de rocas: caliza, arenisca y granito, las cuales ocurren como tres polígonos. Los polígonos están numerados en la Figura 1.11-B, y existe una correspondencia de 1:1 entre los polígonos y los afloramientos. Cada polígono es descrito por medio de un *ciclo externo* y cero a más *ciclos internos*. En las figuras se puede observar que:

1. El polígono 1 es simple, y es definido por el ciclo 2 (un ciclo externo).
2. El polígono 2 no es simple, porque está circunscrito en el ciclo 1 (un ciclo externo) y contiene el ciclo 3 (un ciclo interno) como una isla.
3. El polígono 3 es una isla y es definida sólo por el ciclo 3 (un ciclo externo).

Los polígonos son definidos en términos de ciclos por la tabla topología de polígonos, -ver Tabla 1.1-A-, en la tabla se usa un campo numérico llamado secuencia de ciclos; este campo mantiene la pista de los casos que tienen más de un ciclo por polígono. Si existe más de un ciclo por polígono, significa que tiene un ciclo externo y todos los demás son ciclos internos. La tabla también satisface las reglas de normalización del modelo relacional.

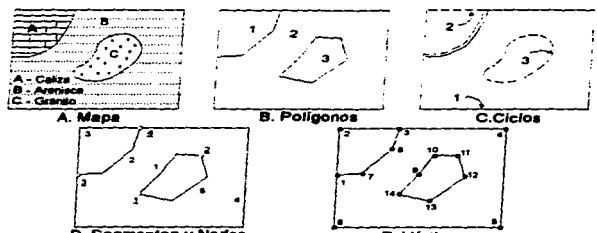


Figura 1.11 Bonham [1994], ilustración de los componentes utilizados en la estructura topológica de Van Roessel. **A. Mapa geológico**, el cual muestra tres tipos de rocas. **B.** Instancias de polígonos por tipo de roca. **C.** Descripción de los polígonos por medio de ciclos, los cuales pueden estar compuestos por un ciclo externo y cero o más ciclos internos. **D.** Los nodos que pertenecen a un segmento y están indicados por el número subrayado. **E.** Los vértices de un segmento.

A. Tabla topología de los polígonos

Polígono	Ciclo	Secuencia de Ciclos
1	2	1
2	1	1
2	3	2
3	3	1

B. Tabla topología de ciclos

Ciclo	Segmento	Secuencia de Ciclos
2	3	1
2	2	2
1	2	1
1	4	2
3	1	1
3	5	2

C. Tabla Topología de Segmentos

Segmento	Nodo Inicio	Nodo Final	Polígono Izquierdo	Polígono Derecho
1	1	2	2	3
2	3	4	1	2
3	4	3	1	0
4	4	3	0	1
5	1	2	3	2

D. Tabla Nodo-Vértice

Nodo	Vértice
1	14
2	11
3	1
4	3

E. Tabla Segmento-Vértices

Segmento	Vértice	Secuencia de Vértice
1	14	1
1	9	2
1	10	3
1	11	4
2	1	1
2	7	2
2	8	3
2	3	4
3	5	1
3	2	2
3	1	3
4	3	1
4	4	2
4	5	3
4	6	4
4	1	5
5	14	1
5	13	2
5	12	3
5	11	4

F. Tabla Coordenadas de los Vértice

Vértice	X	Y
1	x1	y1
2	x2	y2.
..	..	..
..	..	..
14	x14	y14

Tabla 1.1 Bonham [1994], tablas relacionales que definen la estructura topológica del mapa geológico de la Figura 1.11.

La segunda tabla llamada topología de ciclos (Tabla 1.1-B), liga los ciclos a los segmentos (Figuras 1.12-C y D). El ciclo 2 se compone de los segmentos 2 y 3, el ciclo 1 de los segmentos 2 y 4 y el ciclo 3 por los segmentos 1 y 5.

La tercer tabla (Tabla 1.1-C) liga los segmentos a los nodos y polígonos. Así el segmento 1 inicia con el nodo 1 y termina en el nodo 2; además contiene información para saber cuales son los polígonos que están a los lados, por ejemplo: el segmento 1 tiene al polígono 2 sobre el lado izquierdo y el polígono 3 al derecho; esta tabla es utilizada para investigar cuales son los polígonos adyacentes, sin necesidad de realizar una búsqueda exhaustiva sobre las coordenadas de los polígonos. Por ejemplo el problema de encontrar los contactos entre granitos y caliza se reduce a buscar en la tabla de topología de segmentos, las tuplas que satisfagan que el atributo polígono izquierdo y derecho sean igual a granito, caliza o caliza, granito. Resultando que en esta búsqueda no se requiere tener la localización de los vértices, por lo tanto es más rápida y eficiente. También el problema de encontrar los polígonos adyacentes que pertenecen a la misma clase para reclasificarlos, se ha solucionado utilizando la misma tabla, donde los atributos polígono izquierdo y derecho deben ser los mismos.

Las Tablas 1.1-D y E, proporcionan la liga entre los segmentos y sus nodos. La tabla que tiene las coordenadas de los nodos y los vértices es la Tabla 1.1-F. Se puede observar que las coordenadas espaciales están en una sola tabla y completamente separadas de los atributos topológicos.

Este conjunto de seis tablas define completamente las relaciones espaciales y topológicas encontradas en un mapa. Los atributos no espaciales no han sido adicionados, pero se pueden usar tablas adicionales para ligar los objetos espaciales con los atributos temáticos y los geométricos. Por ejemplo, si se requiere describir los atributos geológicos como las formaciones de rocas y los tipos de contactos, se deben ligar los polígonos y segmentos a otras tablas que contengan los atributos temáticos.

En conclusión las ventajas que tiene esta estructura son:

- No existen repeticiones de coordenadas entre los polígono adyacentes, excepto para los nodos.
- La información topológica es almacenada explícitamente y se encuentra separada de la localización de los objetos, facilitando las búsquedas de objetos adyacentes, contenidos y conectados.

Las desventajas de esta topología estructural son:

- Las tablas topológicas deben ser generadas en primera instancia, implicando organizar los datos, lo que puede ser costoso y además requiere espacio para almacenamiento.
- Algunas operaciones simples como el despliegue de la información en forma de mapa sobre el monitor son lentas y pesadas, debido a que se requieren las coordenadas de los objetos y la forma de accesarlas no depende de la topología.

Por ejemplo, para dibujar los límites de todos los polígonos que representan los afloramientos de granito, se deben seleccionar los segmentos de la tabla topología de segmentos, donde los registros tengan el atributo de polígono izquierdo o derecho igual a granito, determinar los nodos de los segmentos, para después obtener el número de vértice que corresponde a cada nodo en la tabla nodo vértice, luego encontrar las coordenadas de los vértices del segmento.

Se puede concluir que escoger una estructura para representar a los objetos espaciales es una de las tareas importantes en la construcción de un sistema de software; la selección dependerá del fin que se busque; por ejemplo si la información es usada para realizar análisis resulta adecuado generar la topología de los datos; si es simplemente para desplegarla en forma de mapa no es conveniente usar una estructura topológica

2 Sistemas de información geográfica

En los últimos años los sistemas de información geográfica se han convertido en herramientas de gran utilidad en las actividades relacionadas con el manejo y procesamiento de la información espacial. Se define un *sistema de información geográfica*, o simplemente SIG, como un sistema computacional que tiene como propósito ayudar a planear y tomar decisiones basadas en los datos espaciales.

En este capítulo se describe en forma general las características de los SIG. Se incluyó el presente con el objeto de descubrir elementos que ayuden en el desarrollo del sistema propuesto en esta tesis; debido que el prototipo funcionará como un módulo de entrada de datos para un SIG.

2.1 Funciones

Bonham [1994] describe que las funciones principales de un SIG son: organizar la información, visualizar, consultar, integrar, analizar y predecir. A continuación se describe cada una de ellas:

Organizar la información

Es de vital importancia tener organizada la información dentro de un SIG. Los datos pueden ser organizados de muchas maneras, pero la más adecuada es aquella que ofrezca una recuperación eficiente. Por lo general en un SIG se organizan los *objetos espaciales* bajo las siguientes características:

- La *posición* en el espacio. La manera principal de organizar los objetos espaciales es por sus coordenadas en un sistema de proyección definido.
- Los atributos que no están relacionados con su posición, también se les conoce como atributos *no espaciales* o *temáticos*.
- Las *relaciones topológicas*, el término topológico se refiere a las propiedades que tiene un objeto espacial con respecto a otros, por ejemplo: la *adyacencia* (es cuando dos objetos están en contacto) y la *contención* (es cuando un objeto forma una isla en otro objeto).

Una tabla de datos que describe los objetos espaciales de un mapa puede mostrar relaciones interesantes entre ellos, pero sin conocer su localización resulta imposible establecer patrones espaciales y relaciones topológicas entre los objetos. Los esquemas para organizar los datos se conocen como *modelos de datos espaciales*. Una descripción detalladas de los diferentes modelos de representación digital se describió en la sección 1.2. Si el SIG no cuenta con una manera eficiente de recuperar la información, esto afectará a sus otras cinco funciones.

Visualizar

Las capacidades gráficas de las computadoras son explotadas por los SIG para visualizar los datos espaciales. Los monitores son utilizados para desplegar la información, pero también se pueden usar otros dispositivos, como impresoras. Los humanos tenemos una extraordinaria habilidad para visualizar y entender complejas relaciones espaciales, pero si esta misma información se presenta en forma de tabla se dificulta reconocer la distribución de los valores altos y bajos en el espacio. Cuando un SIG despliega los datos en forma de mapa, usando colores y símbolos se revelan los patrones de distribución.

Consultar

La visualización es una característica relevante que permite revelar patrones espaciales entre conjuntos de datos organizados, sin embargo por medio de esta no se pueden responder preguntas acerca de los valores de las instancias particulares; por lo tanto la visualización es complementada con consultas espaciales. Los SIG deben proporcionar los medios para realizar dos tipo principales de consultas interactivas: ¿Cuáles son las características de la región? y ¿Donde ocurren estas características?.

Algunas consultas estarán parametrizadas por distancias, orientaciones, adyacencias y contenciones, para estas consultas no sólo se requieren búsquedas eficientes, también se deben derivar relaciones geométricas y topológicas entre los objetos espaciales.

Integrar

La habilidad de manipular, combinar y desplegar un conjunto de datos espaciales de diferentes fuentes, permite entender e interpretar fenómenos geográficos que son imposibles de explicar a partir de capas de información aisladas. El proceso de combinar las capas de información espacial se llama *integración*. Un proceso de integración se puede aplicar de dos formas: la primera es por medio de desplegar un conjunto de capas de información, la segunda es con modelos que efectivamente permitan crear un mapa nuevo formado de dos o más mapas existentes.

Los modelos matemáticos simbólicos de integración usan operaciones aritméticas y lógicas para unir las capas de información en una sola. Las expresiones aritméticas y lógicas que procesan las capas de información se escriben en algún lenguaje de programación, en el contexto de los SIG a estas expresiones se les conoce como *álgebra del mapa*.

Una de las propiedades que hacen a un SIG una herramienta de gran utilidad es la capacidad de juntar expresiones algebraicas de mapas para formar algoritmos mas complejos, de esta manera varios mapas y tablas con atributos de objetos espaciales pueden combinarse en sólo un paso. El proceso de combinar mapas se le conoce como *modelado cartográfico*.

Analizar

El análisis es el proceso de inferir el significado de los datos. En un SIG, el análisis se realiza en base a una serie de medidas obtenidas como resultado de aplicar: cálculos estadísticos, modelado de datos y otras operaciones. Por ejemplo con la estadística se puede resumir las características de una región dibujada en un mapa en una *tabla* o un *histograma*. Las medidas estadísticas de los atributos de los objetos espaciales, como la *media* y la *desviación estándar* sirven para establecer las relaciones que existen entre los objetos. También un modelo de una regresión entre variables espaciales puede ser utilizado para predecir los valores y observar si las variables que describen el medio ambiente están relacionadas.

Debido a que algunas de las herramientas de análisis son de uso específico no están contempladas dentro de un SIG; por lo tanto deben contar con operaciones para exportar los datos a otros sistemas computacionales; así el análisis se puede realizar en base a datos organizados como mapas o como tablas.

Predecir

El propósito de un estudio con un SIG frecuentemente es el de predecir. Por ejemplo la integración de diferentes capas de información puede ayudar a indicar características favorables para un determinado uso del suelo, también por medio de un estudio se pueden deducir localidades favorables que contengan depósitos de minerales económicos. Para hacer las predicciones se aplican modelos basados en conocimiento expresados por un conjunto de reglas, las cuales se obtienen de expertos de diferentes áreas.

2.2 Historia

Los primeros trabajos con matemáticas apropiadas para la solución de problemas espaciales comienzan entre los años treinta y cuarenta con el desarrollo de los métodos estadísticos de análisis de series de tiempo. Pero es hasta la década de los sesentas, con la utilización de las computadoras, cuando comienza el florecimiento de los métodos de análisis espaciales y la posibilidad para trabajar con mapas temáticos en formato digital (Burrough, 1986; Cliff y Ord, 1981; Journel y Huijbregts, 1978; Ripley, 1981; Webster, 1977).

Las áreas del conocimiento como: la topografía, el catastro, la cartografía temática, la ingeniería civil, la geografía, las ciencias edafológicas, la agrimensura, la fotogrametría, la planeación rural y urbana, la percepción remota y el procesamiento de imágenes fueron las promotoras en el uso de computadoras para el manejo de mapas y análisis de información espacial, las cuales dieron la pauta para la creación de los SIG.

Star y Estes [1990] comentan que los primeros trabajos de análisis de información espacial por computadora fueron desarrollos de herramientas analíticas, para ayudar a los investigadores y profesionales en una variedad de aplicaciones. Algunos trabajos descritos por Star y Estes son:

El sistema llamado STORET utilizado para almacenar información espacial acerca de la calidad del agua (Green, 1964) y el MIADS que fue desarrollado por el Servicio Forestal de los E. U. para análisis de alternativas de recreación (Amidon, 1964).

En la comunidad universitaria, durante la década de los sesentas se hicieron importantes contribuciones, como fue el caso del Laboratorio de Graficación por Computadora de Harvard, donde se desarrollaron una serie de programas para el mapeo automático y análisis de la información espacial. La Universidad de Washington en Seattle contribuyó particularmente en las áreas de análisis de transportación y planeación urbana (Gaits, 1969).

El primer sistema que puede ser reconocido como un SIG fue el *Canada Geograhic Information System* o *CGIS* (Peuquet, 1977). Tomlinson [1982] comenta que el diseño y desarrollo del sistema *CGIS* fue realizado específicamente para el programa de *Rehabilitación de la Agriculturas y Desarrollos de Obras* del gobierno de Canadá. El propósito principal del *CGIS* fue para el análisis del Inventario de Datos del Suelo de Canadá. Su implantación se realizó en el año de 1964 (Deuker, 1979), tan sólo un año después de la primera conferencia sobre Sistemas de Información en Planeación Urbana y Programas, que contribuyó al establecimiento de la Asociación de Sistemas de Información Regional y Urbana. Otros de los sistemas implementados por estas fechas son el *New York Landuse* y *Minnesota Land Management Information System*.

Varios programas de computadoras y algunos sistemas fueron desarrollados para el manejo de mapas durante los años setentas, ellos son considerados como aplicaciones de *cartografía asistida por computadora*, particularmente en Canadá y E. U. Con el objeto de apoyar actividades de gobierno y de agencias privadas (Tomlinson et al, 1976; Teicholz y Berry, 1983).

2.3 Componentes de hardware

Los componentes de hardware propuestos por Burrough [1986] para la ejecución de un SIG son: la unidad de proceso central (CPU), un manejador de disco para almacenar la información, un digitizador u otros tipos de dispositivos usados para convertir datos de los mapas a un formato digital, un graficador utilizado para imprimir los resultados de los procesos de análisis y un manejador de cintas para respaldar tanto la información como los programas. Los diferentes dispositivos son manipulados por medio de una terminal. En la Figura 2.1, se muestran los componentes de hardware de un SIG.

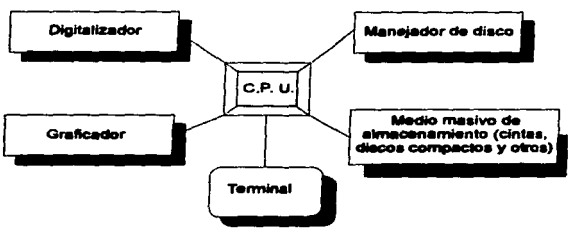


Figura 2.1 Componentes físicos de un sistema de información geográfica.

2.4 Método para un estudio

No existe un método establecido para el desarrollo de un estudio con un SIG, debido a que el fin que se persigue en cada uno de ellos es diferente; pero de los trabajos de Bonham [1994], Burrough [1986] y Star y Estes [1990] se recopiló una serie de fases y actividades que son realizadas en una gran cantidad de estudios. Las fases del método propuesto se muestran en la Figura 2.2, y una descripción breve de cada una de ellas se realiza a continuación:

Adquisición de datos

La primera actividad a realizar es la recopilación de los datos apropiados y necesarios para realizar el estudio. La información es obtenida de los mapas existentes, de las observaciones de campo, de instrumentos de registro y de fotografías aéreas y de satélite.

Construcción de la base de datos

En esta fase se realizan dos tareas principales. La primera es hacer el diseño de la base de datos; para ello se deben considerar las restricciones del problema. La segunda comprende la identificación de las localidades de los objetos relevantes en la información obtenida en la fase anterior, después se deben registrar en la base de datos de una manera sistemática cada uno de ellos; si se requieren las relaciones topológicas se deben generar o capturar.

Procesamiento de datos

Una de las tareas principales del procesamiento de datos es la manipulación y análisis de la información para conseguir respuestas a las preguntas que son formuladas con el motivo de resolver el problema planteado. Estas operaciones se realizan sobre datos espaciales y no espaciales o sobre una combinación de ambos. La manera de realizar el procesamiento es analizar cada una de las capas de información en forma independiente, para derivar los patrones de datos relevantes. Los patrones y la información obtenida de cada capa será combinada en la siguiente fase del proyecto.

Integración

Esta fase se encarga de combinar las evidencias obtenidas de la fase anterior por medio de aplicar modelos de integración. El objetivo de integrar las capas de información es analizar y describir las interacciones entre ellas para hacer predicciones y proporcionar un soporte en la toma de decisiones. En el proceso de integración se combinan las diferentes capas de información con la aplicación de diversos métodos y modelos; Bonham [1994] describe una amplia gama de ellos utilizados en el áreas de ciencias de la tierra.

Generación de productos

El último paso en el desarrollo de un estudio con un SIG, es presentar la información obtenida como resultado de realizar las fases anteriores. La información puede ser mostrada como mapas, gráficas, tablas y reportes textuales en diferentes medios de salida.

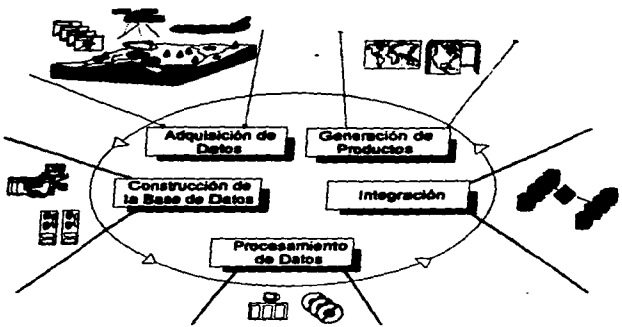


Figura 2.2 Las fases de un estudio con un sistema de información geográfica.

2.5 Aplicaciones

En la actualidad los SIG se han convertido en herramientas de gran utilidad en muchas actividades, debido a que proporcionan los elementos de análisis y modelado de la información espacial para atacar diferentes problemas. Algunas de las áreas donde se utilizan son: catastro, planeación rural y urbana, topografía, agrimensura, fotogrametría, geografía, edafología, geología y otras. En seguida se listan algunos trabajos descritos por Bonham [1994], Jeans y Chistensen [1986], Sorani et al [1993] y Star y Estes [1990] con el objetivo de mostrar la diversidad de usos que tienen actualmente los SIG.

Administración de recursos naturales

La siguiente aplicación que se describe tiene como objeto mostrar que los SIG son útiles en el manejo y administración de recursos naturales. Este estudio consistió en realizar un análisis de irrigación y potencial del agua. *United Nations Food and Agriculture Organization (FAO)* contrató al *Environmental Systems Research Institute* (*ESRI*), mejor conocido por su SIG de tipo vectorial llamado *ARC/INFO*. El contrato consistió en desarrollar una base de datos para la estimación del potencial de irrigación en el continente africano, específicamente fue diseñada para identificar las áreas que tienen suficiente agua para soportar agricultura de riego. El producto final del estudio, fue un mapa que muestra las áreas de irrigación de suelos y las que no necesitan ser irrigadas debido a la lluvia.

Inventario de recursos naturales

El Instituto de Geografía de la Universidad Nacional Autónoma de México y la Secretaría de Agricultura y Recursos Hidráulicos elaboraron un inventario nacional forestal de México. El proyecto consistió de dos fases. En la primera se hizó una clasificación de un mosaico de imagenes Landsat TM con algoritmos de procesamiento digital de imagenes y conjuntamente una clasificación manual. En la segunda fase se validó y corrigió la información obtenida por medio de observaciones en el campo. Los resultadas de la foto-interpretación se pasaron a formato raster y despues se vectorizados para formar un SIG con el fin de hacer una evaluación y planeación de los recursos forestales. Diferentes capas de información se incorporaron al SIG como divisiones políticas, clima, información socieconómica y otras. En primera instancia se buscó con el sistema determinar zonas forestales para su conservación, producción y regeneración. El software usado para este propósito fue ARC/INFO con la ayuda del manejador de bases de datos INGRES.

Prospección de minerales

Existen varios estudios con el objetivo de localizar depósitos de mineral potenciales, los cuales son estudios típicos de geología para tomar decisiones en la realización de exploraciones a detalle. Una de las aplicaciones que describe Bonham [1994] es la evaluación del potencial de mineralización de una pequeña áreas en Manitoba, Canadá. El trabajo fue emprendido como parte de un proyecto de investigación de la *Geological Survey of Canada*; con la innovación de nuevas tecnologías aplicadas a la exploración de depósitos de sulfuros masivos de tipo volcanogenético. El método aplicado consistió de tres fases. En la primera fase se construyó una base de datos, la principal fuente de información fueron los mapas geológicos, mapas de alteraciones de rocas, muestras geoquímicas analizadas por varios elementos metálicos y datos geofísicos. Todos los datos fueron trasladados a un formato digital. En la segunda fase se procesaron las diferentes capas de información para extraer las evidencias críticas (como contactos entre rocas félsicas y volcanoclásticas) para realizar predicciones en zonas favorables de depósitos de minerales. En la tercera fase se combinaron las diferentes capas de información, para obtener un mapa nuevo que mostrara las evidencias de depósitos de sulfuros masivos de tipo volcanogenético.

Planeación para la selección de sitios apropiados

La siguiente aplicación describe un método con la ayuda de SIG de tipo *raster* para la selección de un sitio para el depósito de desperdicios. El problema consistía en el análisis de las propiedades para el uso del suelo. Los pasos del método comienzan con la identificación de las restricciones relevantes para la localización del sitio, seguido de una recopilación de datos en fotografías aéreas, para posteriormente usar fotointerpretación y elaborar un mapa con dicha información, el cual fue digitizado para realizar un análisis del conjunto de datos para detectar las áreas con potencial.

Modelado de producción agrícola

El cuerpo administrativo de *Regione del Veneto* al norte de Italia, tiene un programa de cooperación e investigación con la Universidad de California, cuyo propósito es determinar procedimientos, técnicas y producción de siembras en soporte al manejo de los recursos agrícolas. Se utilizó un SIG para recopilar información del medio ambiente, de la economía y de agronomía requerida para estimar la producción de los cultivos y para construir un modelo económico. Los datos fueron obtenidos de los mapas de clases de cultivo, meteorológicos que incluyen precipitaciones y temperaturas, de uso del suelo y de potenciales agrícolas.

Análisis de hábitat de especies

Varias agencias públicas y grupos privados se propusieron el proyecto de aumentar la población del cóndor de California (*Gymnopyps californianus*). El objetivo de este programa consistía en establecer una población de 100 pájaros. Para cumplir con esto se creo una base de datos con la ayuda de un SIG con cinco capas de información, las cuales describen variables que se relacionan directa e indirectamente con su hábitat, por ejemplo, los sitios donde anida, los sitios de vuelo, los sitios de alimentación, las áreas de recreación, de uso del suelo, de topografía, la red de caminos, las líneas de energía eléctrica y otras. Los resultados del trabajo demostraron que los SIG son herramienta eficientes y efectivas para la recopilación y el análisis de gran cantidad de información de diferente naturaleza, relevante para el estudio de los hábitat de especies de animales.

2.6 Otros sistemas relacionados

Existen otros sistemas relacionados con los SIG, que han jugado importantes papeles en su desarrollo. Los más estrechamente relacionados son: los de *graficación por computadora*, los de *diseño asistido por computadora* y los de *procesamiento digital de imágenes*. Estos sistemas se encargan de manejar datos espaciales en modelos de datos diferentes y además tienen otras funcionalidades.

Aunque en la actualidad los sistemas de graficación por computadora abarcan muchas aplicaciones, desde interfaz gráfica para usuarios hasta aplicaciones en las artes y en animaciones. Los SIG han incorporado técnicas de los sistemas de graficación por computadora sobre todo en las actividades de visualización y manipulación de objetos. Existen sistemas de graficación por computadora para la visualización de datos, encargados de examinar números y símbolos, proporcionando una alternativa de transformar la información al lenguaje visual. Muchos de los datos manipulados por estos sistemas de graficación por computadora son de tipo espacial y pueden ser utilizados como herramientas auxiliares de los SIG.

Los sistemas de diseño asistido por computadora fueron originalmente desarrollados para hacer diseños y dibujos en las áreas de la ingeniería. Estos emplean estructuras de datos de tipo vectorial para la representación de los puntos, líneas y polígonos. Tienen la semejanza con los SIG en que ambos manejan atributos no gráficos para describir a los objetos y además describen relaciones topológicas entre ellos. Las diferencias radican en que en los SIG el volumen y diversidad de información es mucho más grande, y los métodos de análisis de datos son de naturaleza diferente; dichas diferencias implican que un sistema de diseño asistido por computadora no puede ser usado como un SIG ni viceversa.

Los sistemas de procesamiento digital de imágenes fueron desarrollados para manipular y visualizar imágenes digitales en un formato *raster*. El formato *raster* es simplemente un arreglo de pixeles, la localización de un pixel se obtiene implícitamente por la secuencia en el cual fue digitizado. Los sistemas de procesamiento digital de imágenes son herramientas muy poderosas para desplegar y analizar imágenes que pueden contener información espacial, la estructura de tipo *raster* tiene la facilidad para hacer sobreposiciones y combinaciones de capas de información y la aplicación de filtros.

2.7 Consideraciones para el proyecto

El sistema propuesto en esta tesis consiste en un módulo de un SIG, utilizado para capturar la información de los mapas y crear una base de datos espaciales; como el sistema se compone de un dispositivo de tipo tableta para obtener los datos, estará basado en el modelo de representación vectorial. Como se puede observar en este capítulo existen diferentes variantes para organizar y manipular la información contenida en los mapas; en el proyecto los objetos espaciales se agruparán en: puntos, segmentos y polígonos. Los puntos serán utilizados para representar todos los objetos geográficos localizados por un par de coordenadas (x,y). La representación de los segmentos se hará por las estructura topológica *red de segmentos*. Los polígonos se representarán por la estructura segmento nodo por que permite evitar los problemas de doble captura para los mapas que están compuestos por un mosaico de polígonos. Para el manejo de los mapas se utilizará el modelo de *planos sobrepuestos*; para esto primero se construirá un sistema con tres planos de información de puntos, segmentos y polígonos; después se estudiará la manera de extender este modelo para representar cualquier tipo de mapa.

Segunda parte

Desarrollo del sistema

3 Metodología de desarrollo de sistemas orientados a objetos

El presente capítulo tiene como propósito hacer una descripción de la metodología utilizada para modelar el sistema propuesto en esta tesis; también incluye el proceso del ciclo de vida del software orientado a objetos.

Una *metodología* es una colección de métodos con algún enfoque específico, aplicados a lo largo del desarrollo del ciclo de vida del software. La metodología orientada a objetos permite abordar los problemas en una forma más natural de acuerdo a como los humanos observan el mundo real, de esta manera se facilita la construcción de modelos de sistemas de mayor complejidad.

Un *método* es un proceso disciplinado para generar un conjunto de modelos, que describen los aspectos de un sistema de software en desarrollo, para ello se utiliza: una notación que permita expresar el modelo, un proceso que es la actividad que organiza la construcción del modelado del sistema y las herramientas que facilitan el trabajo tedioso.

3.1 El modelo de un sistema

A diferencia de otras formas de modelado como la descomposición algorítmica, la orientada a objetos es una nueva filosofía que trata de resolver los problemas basándose en los *sustantivos* que los describen; por lo tanto la descomposición se hace en base a las abstracciones del dominio del problema, tratando a ellos como agentes autónomos que colaboran para lograr el comportamiento deseado del sistema. Desde esta perspectiva un *objeto* es simplemente una abstracción de una cosa tangible, que exhibe un comportamiento definido. Los objetos realizan servicios y para ser utilizados se comunican entre sí mediante el *envío de mensajes*.

Para realizar una descomposición orientada a objetos de un problema, se utilizan los métodos de análisis y diseño, que tienen como objetivo especificar el sistema en un modelo que refleja la estructura y su comportamiento, en una vista lógica y otra física que a su vez cada una de ellas tiene una vista estática y otra dinámica. La Figura 3.1 muestra el modelo de descomposición de un sistema.

Figura 3.1 Modelo de un sistema orientado a objetos propuesto por Booch [1993].

La vista lógica muestra el significado de las llaves de abstracción y la arquitectura del sistema; la física describe el software y el hardware que se utilizará. Las vistas estática y dinámica capturarán la definición y el comportamiento de los objetos expresado en términos de envío de mensajes. Para ilustrar cada una de las vistas del modelo se utilizan los siguientes diagramas:

Diagramas estáticos:

- Diagramas de clases
- Diagramas de módulos
- Diagramas de procesos

Diagramas dinámicos:

- Diagramas de objetos
- Diagramas de interacción
- Diagramas de transición de estados

3.2 El ciclo de vida del software

En el proceso de desarrollo de un sistema orientado a objetos, se ha observado que la manera de modelar la arquitectura de los sistemas de software tiende a ser iterativa e incremental; se dice que es iterativo en el sentido que envuelve sucesivos refinamientos de la arquitectura orientada a objetos y que es incremental porque cada paso del ciclo a través del análisis, diseño y evolución permite ir gradualmente aumentando la arquitectura del sistema.

Booch [1993] establece que en los desarrollos orientados a objetos no se aplica una descomposición estrictamente de *arriba hacia abajo*, ni estrictamente de *abajo hacia arriba*; en su lugar propone la estrategia que sugiere Druke [1989], que considera que los sistemas complejos deben ser creados a través de un diseño circular (en ingles *round-trip gestalt design*). Este estilo de diseño enfatiza el desarrollo incremental e iterativo por medio de refinamientos sucesivos del modelo lógico del sistema.

3.3 El proceso

El desarrollo de un sistema orientado a objetos se descompone en dos procesos, uno llamado *micro* y otro *macro*. El proceso macro tiene como objetivo controlar las fases del proceso micro. En cada una de las actividades del macro, el micro estará activo y se encargará de ir refinando el modelo del sistema. Una descripción más detallada de ambos se hace a continuación:

3.3.1 El proceso micro

Las actividades principales del proceso micro son:

- Identificación de clases y objetos
- Identificación de la semántica de clases y objetos
- Identificación de las relaciones entre clases y objetos
- Especificación de interfaces e implantación de clases y objetos

En la Figura 3.2 se muestra el orden de las fases del proceso micro. Se puede concluir, que es un proceso incremental e iterativo.

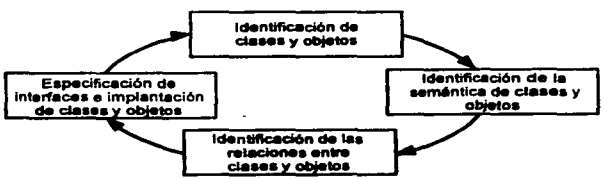


Figura 3.2 Actividades del proceso micro.

3.3.1.1 Identificación de clases y objetos

El propósito de identificar las clases y objetos es establecer los límites del dominio del problema y es la primera actividad en la construcción de un sistema orientado a objetos.

Cuando el proceso macro se encuentra en la fase de análisis se descubren las abstracciones que constituyen la estructura del dominio del problema. En el diseño se inventan nuevas abstracciones que formarán parte de la solución del problema. Finalmente en la implantación se crean las abstracciones de bajo nivel para ser utilizadas en la construcción de abstracciones de alto nivel.

Para descubrir las abstracciones se puede aplicar el *enfoque clásico* de análisis orientado a objetos, este consiste en generar un conjunto de candidatos a clases y objetos a partir de los requerimientos del sistema. El enfoque clásico se utiliza en varias metodologías como: Coad y Yourdon [1990], Shlaer y Mellor [1988], Wirfs-Brock et al [1990].

Rubin y Golberg [1992] establecen otro enfoque que permite completar la identificación de clases y objetos, por medio de las *funciones punto*. Desde la perspectiva de los usuarios finales una función punto representa alguna actividad primaria de un sistema en respuesta a un evento, también son vistas como transformaciones que el sistema hace al medio ambiente. Las funciones punto son utilizadas con dos objetivos. El primero es para especificar todo lo que se observa exteriormente. En el segundo para verificar el comportamiento correcto del sistema. Este enfoque se aplica en la identificación de abstracciones que no se encuentran por medio del enfoque clásico pero están involucrados en el comportamiento del sistema.

En esta fase se construye un diccionario de datos, donde se registran todas las abstracciones encontradas, el cual servirá para documentar el sistema y también para saber cuándo se deberá terminar esta fase, la cual debe concluir con la presencia de un diccionario estable. Debido a que se trata de un proceso iterativo e incremental, es difícil tener la estabilidad del diccionario, por lo tanto es suficiente contar con un conjunto amplio de abstracciones para decidir concluir la primera iteración de la fase.

3.3.1.2 Identificación de la semántica de clases y objetos

Esta fase tiene como objetivo identificar la semántica de las clases y objetos, para ello se deben determinar los atributos y comportamientos de cada una de las abstracciones identificadas en la fase previa. Aquí también se determina cuáles son las clases que dejarán de ser candidatas por medio de la distribución de responsabilidades.

En el análisis, se aplica este paso para asignar responsabilidades, como parte del diseño, para determinar cuáles elementos forman parte de la solución. En la implementación, se refinan las operaciones pasando sus especificaciones no formales a signaturas precisas.

Una de las actividades que permite identificar la semántica de las clases y objetos del sistema, fue la elaboración de un conjunto de escenarios relacionados con las funciones punto.

Esta fase termina cuando se tiene un conjunto de responsabilidades y/o operaciones asignadas a cada abstracción. Al inicio del proceso es suficiente contar con una asignación informal de responsabilidades; pero a medida que se estabiliza el modelo del sistema, se van refinando las responsabilidades de cada una de las clases.

3.3.1.3 Identificación de las relaciones entre clases y objetos

El propósito de identificar las relaciones entre clases y objetos es establecer los límites de colaboración entre cada una de las abstracciones que fueron encontradas en la primera fase del proceso micro; para cumplir con este objetivo también se utilizan los escenarios.

Como parte del análisis, se aplica esta fase para especificar las relaciones de asociaciones, de herencia y de agregación. Para documentar las relaciones entre las abstracciones se usan los diagramas de clases, en ellos se pueden especificar los atributos y operaciones más significantes de cada abstracción.

En el diseño se aplica para agrupar las clases dentro de categorías y los módulos dentro de subsistemas. Los diagramas que se obtuvieron en la parte del análisis servirán para tomar decisiones tácticas y seleccionar los algoritmos en la siguiente fase. Cuando la implantación procede, se refinan las relaciones entre clases y objetos incluyendo las de instanciación y uso.

Esta fase es de importancia porque permite documentar el modelo del sistema a partir de los diagramas de clases, de objetos, de interacción, de módulos y de transición de estados. Los diagramas además de capturar las relaciones entre las abstracciones muestran las diferentes vistas de la arquitectura del sistema.

Esta fase queda terminada cuando se tienen especificadas la semántica y las relaciones entre las abstracciones, para servir de línea base en la implantación del modelo.

3.3.1.4 Especificación de las interfaces de clases e implantación

El objetivo de la especificación de las interfaces de las clases e implantación durante el análisis, es proporcionar un refinamiento de las abstracciones existentes, así como descubrir nuevas clases y objetos para el siguiente nivel de abstracción; permitiendo alimentar la próxima iteración del proceso micro. Para lograr esta meta se implementan las clases y los objetos encontrados hasta el momento. En el diseño, se crean representaciones tangibles de las abstracciones en soporte a los sucesivos refinamientos en la liberación de ejecutables del proceso macro.

Es importante puntualizar que para aplicar esta fase en el análisis y diseño se debe tener un conjunto de *diagramas estables*. Un diagrama estable, es aquel que al aplicar una iteración más del proceso, los cambios realizados en la estructuras y comportamiento de sus clases y objetos es mínimo con respecto a la iteración anterior.

En esta fase también se documenta el modelo del sistema, por medio de diagramas que incluyan la semántica de las abstracciones, tales como los diagramas de transición de estados y de interacción, obteniendo de esta manera una documentación más exacta.

El diccionario de datos se actualiza en este paso, por que se debe incluir las clases y objetos que fueron descubiertos e inventados en la implantación de las abstracciones existentes.

Existen dos actividades principales a realizar en esta fase. La primera es la selección de las estructuras y algoritmos que proporcionan la semántica a las abstracciones identificadas. La segunda es descubrir nuevas abstracciones a las cuales se les delegarán responsabilidades. Las actividades anteriores se pueden desglosar en las siguientes tareas:

- Para cada una de las clases se deben identificar los patrones de uso entre sus clientes, con el fin de determinar cuáles son las operaciones centrales y cuáles pueden ser optimizadas. Cuando procede la implantación se deben afinar las signaturas de todas las operaciones significantes.
- Se deben considerar los objetos a los cuales se les pueden delegar responsabilidades, con el fin de hacer una optimización; esto puede implicar un reajuste pequeño de responsabilidades.
- Seleccionar los algoritmos adecuados para cada operación, incluyendo las más primitivas. Los algoritmos complejos se pueden dividir en menos complicados con el fin de reutilizar partes.

3.3.2 El proceso macro

Para lograr los objetivos del proceso macro, se aplican diferentes medidas con el motivo de obtener varios productos que ayuden al grupo de desarrolladores a asignar riesgos y hacer correcciones al inicio del proceso. Las actividades del proceso macro son realizadas por todos los desarrolladores sobre escalas de tiempo mayor que el proceso micro. El conjunto de actividades de este proceso son:

- Especificación de los requerimientos del sistema.
- Análisis, el cual consiste en desarrollar un modelo del sistema enfocado a determinar el alcance y comportamiento deseado.
- Diseño, tiene como objetivo crear la arquitectura que será utilizada en la implantación.
- Evolución, consiste en la implantación, integración y elaboración de pruebas del sistema.
- Mantenimiento, involucra la administración del sistema después de su liberación.

En la Figura 3.3 se muestra las fases del proceso macro, donde se puede observar que el micro está interactuando en cada una de las fases del macro.

Muchos elementos del proceso macro son simplemente buenas prácticas del manejo del software y se pueden aplicar tanto a sistemas orientados a objetos, como a los no orientados a objetos; algunas de estas son: el aseguramiento de la calidad, la aplicación de métricas, la documentación y otras.

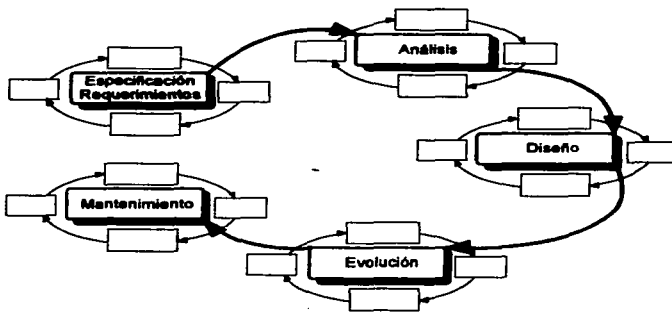


Figura 3.3 Actividades del proceso macro.

3.3.2.1 Especificación de los requerimientos

En la especificación de los requerimientos se investigan los propósitos y características del sistema. Esta actividad es de naturaleza intensamente creativa y no debe ser manejada por reglas rígidas. Las personas que trabajan en esta fase son: solicitantes del producto, vendedores que lo comercializarán, usuarios finales y analistas. Existen diferentes maneras de investigar los requerimientos: una es la realización de entrevistas por parte de los analistas a los expertos del dominio del problema, otro es por medio de estudiar los sistemas que existen para esos fines y en los documentos que describen los propósitos y las características del sistema.

3.3.2.2 Análisis

Para fines prácticos en esta tesis se considera el análisis como las actividades que delimitan el alcance del sistema, el cual proporcionará un modelo global del comportamiento del mismo. Como objetivo particular se descubrirán las clases y objetos del dominio del problema; asignándoles sus atributos, operaciones, responsabilidades y colaboraciones para cada una de ellas.

Una de las actividades es la selección y elaboración de los escenarios relacionados con las *funciones punto* del sistema. Una función punto para el analista, representa una parte del comportamiento del sistema y es usada con dos objetivos. El primero para ilustrar las claves del comportamiento. El segundo para mostrar el comportamiento bajo condiciones excepcionales. La forma de como se elaboraron los escenarios en esta tesis, es la utilizada por Coad y Nicola [1993], que consiste en hacer una descripción de un diálogo entre los objetos que interactuan en una tarea particular.

Otra actividad es el manejo de la evolución del diccionario de datos, este debe incluir todas las clases que fueron identificadas al aplicar el enfoque clásico y las que resultan de los escenarios.

Es importante que los analistas sepan cuáles son sus objetivos, para esto siempre deben tener en mente las siguientes preguntas:

¿Cuál es el alcance del sistema?

¿Cuál es el comportamiento deseado para el sistema ?

¿Cuáles son las responsabilidades de los objetos que participan en el sistema?

Esta fase concluye cuando se tiene un modelo completo y claro del sistema, la documentación de todos los escenarios, por medio de los diagramas de clases, de objetos y de interacción y además validados por expertos del dominio del problema, usuarios, analistas y diseñadores.

3.3.2.3 Diseño

Los límites entre el análisis y diseño orientado a objetos no están bien definidos y es difícil limitar hasta donde llega cada uno de ellos; pero se conoce que estos tienen distintos objetivos. En la parte del análisis como se mencionó, lo que se realiza es encontrar las clases y objetos que modelen el dominio del problema; mientras que en el diseño se inventan las abstracciones y los mecanismos que proporcionen el comportamiento requerido por el modelo.

Una de las actividades durante el diseño, es investigar en las bibliotecas de clases y herramientas de desarrollo, que proporcionan para ser incorporadas al sistema. Generalmente las bibliotecas cuentan con una gran cantidad de clases reutilizables que fueron encontradas durante el análisis.

La incorporación de las clases de las bibliotecas se hace refinando los diagramas de clases, que consiste en sustituir las clases encontradas durante el análisis por las que tengan una estructura y comportamiento igual.

Esta fase termina cuando se tiene validada la arquitectura del sistema y estable para pasar a la evolución del sistema.

3.3.2.4 Evolución

El propósito de la evolución es integrar y mejorar la implantación del sistema, por medio de refinamientos sucesivos. Gran parte del tiempo de esta actividad se dedica a la realización de pruebas para satisfacer un número de restricciones incluyendo funcionalidad, tiempo y espacio.

Los productos que se obtienen son un conjunto de ejecutables, con el objetivo de refinar la arquitectura del sistema. Por otro lado se puede incluir un estudio por medio de prototipos, usados para explorar alternativas de diseño o futuros análisis que ayudarán a esclarecer otras funcionalidades del sistema.

Al inicio de la fase de evolución los códigos ejecutables son liberados para realizar pruebas por otro grupo de desarrolladores, de esta manera se asegura la calidad del software. Las pruebas consisten en verificar si la liberación cumple con lo especificado, generalmente se compara contra los escenarios establecidos durante el análisis; las inconsistencias se corregirán en esta fase para las próximas liberaciones. El proceso de prueba continuará con un grupo de usuarios finales quienes verificarán la calidad del sistema.

Esta fase queda concluida cuando la funcionalidad y calidad del sistema satisface de manera suficiente las necesidades estipuladas en la especificación de los requerimientos.

3.2.3.5 Mantenimiento

La fase de mantenimiento es considerada como una continuación de la evolución, pero se diferencia en que los cambios que se hacen al sistema son debidos a nuevos requerimientos que no fueron considerados en la especificación inicial.

Las actividades del mantenimiento son muy similares a las de la evolución, pero tienen dos actividades diferentes. La primera consiste en agregar nuevas funcionalidades que modifican algún comportamiento existente, mientras que la segunda está encaminada a la eliminación de los defectos y mejoras del sistema, para esto se elaborará una lista de los defectos encontrados por los usuarios finales y de las mejoras por parte de los desarrolladores; a las propuestas se le asignarán una prioridad, para realizar un análisis de costos para posteriores desarrollos y/o próximas evoluciones del sistema.

4 Definición de requerimientos

En este capítulo se describen los requerimientos del sistema propuesto en esta tesis. Por medio del desarrollo del sistema se podrá estudiar en forma detalla y práctica las ventajas que ofrece la metodología orientada a objetos.

4.1 ¿ Qué es un digitizador de mapas (tipo vectorial) ?

El trabajo tedioso de pasar las coordenadas de los objetos de un mapa a un archivo de computadora, puede ser reducido por medio de un sistema llamado *digitizador* que captura los puntos, las líneas y los polígonos que representan los objetos geográficos. El sistema cuenta con una tableta digitizadora que es un dispositivo electromagnético o electrónico que tiene un *lápiz* o *cursor*. El mapa que se quiere digitalizar se pone encima de la tableta como se muestra en la Figura 4.1; para obtener las coordenadas de un objeto espacial representado por un punto se posiciona el lápiz sobre éste y la tableta se encarga de convertir la posición a una señal digital y la trasmite a una computadora; el sistema digitizador se encarga de interpretar la señal para obtener las coordenadas del punto. El trabajo principal de los sistemas digitizadores consiste en convertir un mapa impreso en papel a una representación digital.

El proceso de digitalización descrito anteriormente se conoce como *digitalización manual* (Bonham[1994]), a diferencia del utilizado por dispositivos ópticos conocidos como *scanner*, los cuales generan una matriz de valores digitales en formato de tipo *raster*.

Figura 4.1 El sistema digitizador.

4.2 La especificación

Antes de comenzar el análisis del sistema es conveniente tener una especificación por escrito de los requerimientos describiendo las características del software; para esto es recomendable que aquellas personas que soliciten el sistema la realicen, porque ellas conocen mejor lo que el sistema debe hacer; cuando no se tenga la especificación es apropiado que sea redactada por el analista. La especificación de los requerimientos es de gran utilidad porque permite descubrir las primeras abstracciones del problema, que son la llave del análisis orientado a objetos.

El problema a solucionar en esta tesis, consiste en construir un subsistema de un SIG para la entrada de datos. En detalle se debe hacer el análisis, el diseño y la implantación de un sistema que permita digitalizar la información contenida en los mapas utilizando la metodología orientada a objetos. El sistema recibirá una señal digital de una tableta, y la interpretará como las coordenadas de puntos del mapa y serán manipuladas para crear puntos, segmentos y polígonos, los cuales representarán diferentes tipos de fenómenos geográficos como: ciudades, ríos, carreteras y otros; permitiendo de esta manera construir una representación digital de los mapas.

En cuanto a las características de manejo de errores, será capaz de eliminar los datos redundantes que sean enviados por la tableta y corregir los datos erróneos que sean capturados por el usuario, para la cual contará con algoritmos de depuración y operaciones de edición para verificar los datos digitalizados contra el mapa original.

El sistema permitirá a los usuarios capturar atributos temáticos los cuales son descritos en un mapa por símbolos y textos.

También tendrá una interfaz gráfica amigable, que facilite el aprendizaje y su manejo. Para mayor comodidad de los usuarios se podrán utilizar los botones del lápiz magnético de la tableta para seleccionar opciones del menú, tales como: añadir y eliminar puntos, segmentos y polígonos. El sistema será robusto en cuanto a compatibilidades de diferentes tabletas encontradas en el mercado; para ello contará con operaciones que le permitan integrar diferentes tipos.

Haciendo un resumen de las principales operaciones que debe tener el digitizador son las siguientes:

Sobre el mapa:

- Crear un mapa.
- Recuperar un mapa.
- Añadir elementos al mapa:
  - Añadir polígono.
  - Añadir segmento.
  - Añadir punto.

- Eliminar elementos del mapa:
  - Eliminar polígono.
  - Eliminar segmento.
  - Eliminar punto.
- Realizar cambios de escala sobre el mapa.
- Almacenar mapa.

Sobre polígono:

- Crear polígono (este será creado a partir de los segmentos existentes):
- Desplegar polígono.
- Seleccionar polígono.
- Calcular área.
- Mostrar atributos.
- Modificar polígono:
  - Añadir segmentos.
  - Eliminar segmentos.
- Eliminar polígono.

Sobre Segmento:

- Crear segmento.
- Desplegar segmento.
- Seleccionar segmento.
- Calcular longitud.
- Mostrar atributos.
- Modificar segmento:
  - Añadir puntos.
  - Eliminar puntos.
- Eliminar segmento.

Sobre punto:

- Crear punto.
- Desplegar punto.
- Seleccionar punto.
- Mostrar atributos.
- Eliminar punto.

5 Análisis del sistema

El presente capítulo tiene el objetivo describir cómo se elaboró el análisis del sistema. Este fue desarrollado con la metodología de Booch [1993], pero también se incorporaron elementos de otras metodologías.

La manera que se recomienda para realizar el análisis orientado a objeto de un sistema es dividirlo por los diferentes tipos de clases. En esta tesis primero se desarrolló el análisis de la parte lógica del sistema que corresponde al dominio del problema, y luego se dedicó a la interfaz del sistema.

Coad et al [1995] proponen una clasificación de clases y objetos más detallada para organizar y guiar el desarrollo de un sistema:

- Las clases del Dominio del Problema (DP). Son las que componen la parte lógica del sistema.
- Las de Interacción Humana (IH). Se componen de las abstracciones de la interfaz del sistema y son usadas para mostrar los servicios a los usuarios finales.
- Las de Bases de Datos (BD). Son las abstracciones que se relacionan con las bases de datos, pueden ser orientadas a objetos, relacionales u otras.
- Sistemas de Interacción (SI). Son sistemas independientes al que se está desarrollando, pero que interactuan con él.
- Las clases y objetos que están Fuera de Tiempo (FT). son las que se encuentran fuera del alcance del sistema, pero en posteriores mantenimientos pueden ser incluidas.

5.1 Identificación de clases y objetos

Como se mencionó anteriormente, uno de los objetivos de la fase de análisis es encontrar los límites del problema; para ello se empezó por descubrir los candidatos a clases aplicando el enfoque clásico, estos se buscaron en el documento de la especificación bajo los siguientes conceptos:

- Objetos físicos.
- Conceptos que definan entidades.
- Cosas tangibles.
- Personas.
- Eventos.
- Dispositivos.
- Lugares.
- Unidades organizacionales.

Los candidatos que fueron descubiertos al aplicar el enfoque clásico se incluyeron en el diccionario de datos, como se puede apreciar en la Tabla 5.1. El diccionario que se utilizó consta de dos columnas una para los nombre de las clases y otra para una descripción del papel que juegan.

Clase	Descripción
Digitizador	Sistema utilizado para la captura de mapas.
Mapa	Es un documento que representa una región de la Tierra, el cual describe diferentes fenómenos geográficos, para ello se utilizan puntos, líneas y polígonos que definen sus localizaciones en el espacio en base a una proyección.
Polígono	Se utiliza para la representación de objetos dentro de un mapa, que determinan una superficie cerrada.
Segmento	Representa los objetos dentro de un mapa que pueden ser representados por su longitud.
Punto	Representa todos los objetos geográficos que son localizados por medio de un par de coordenadas (x,y).
Tableta	Dispositivo electromagnético, utilizado para determinar las coordenadas de los objetos dentro de un mapa.
Usuario	Persona que utiliza el sistema.
Texto	Cadena de caracteres usada como un rótulo para describir la información temática.

Tabla 5.1 Diccionario de datos obtenido como resultado de aplicar el enfoque clásico.

Los métodos de análisis y diseño deben contar con una notación para representar el modelo de un sistema. En la metodología de Booch[1993], una clase se representa con un icono en forma de nube amorfa, trazado con líneas punteada y con su nombre dentro del icono. En la Figura 5.1, se observa la representación de algunas clases.

Figura 5.1 Representación gráfica de las clases Mapa, Punto, Segmento, Polígono y Tableta.

5.2 Identificación de la semántica de clases y objetos

La fase de identificación de la semántica es conocida en otras metodología como la asignación de responsabilidades, por ejemplo Wirfs-Brock et al [1990]. Para cada clase que fue identificada en la fase previa, se empezó por investigar qué atributos tiene y cuáles son sus responsabilidades. Para ello se emplearon las preguntas:

¿Qué cosas conoce?

¿Qué sabe hacer?

Los atributos de las clases se obtuvieron por la respuesta a ¿Qué cosas conoce? y las responsabilidades por ¿Qué sabe hacer?. La manera de responderlas se hizo bajo los siguientes criterios:

- Fijándose sólo en las cosas de interés que permitan lograr el comportamiento deseado del sistema. Este es un principio importante dentro muchas actividades del hombre y principalmente en la ingeniería de software.
- Si el nombre de la abstracción es apropiado para describirla, éste puede servir como intuición para determinar *qué sabe* y *qué hace*.
- La respuesta a la pregunta ¿Qué sabe hacer? parte de ella se encontró en la especificación del sistema como verbos y acciones.

Por ejemplo la clase Tableta:

¿Qué conoce la tableta?

- La posición del lápiz magnético.
- El puerto de comunicación donde envía la información.
- La velocidad de envío de los datos.

¿Qué hace la tableta?

- Dar las coordenadas del lápiz magnético.
- Configurar su sistema de coordenadas.

Al inicio del proceso macro fue suficiente contar con una especificación informal de las abstracciones, pero a medida que se avanza en las iteraciones del proceso se fueron refinando y pasando a una especificación formal; para esto se utilizaron los esquemas de clases y finalmente la sintaxis del lenguaje de programación.

5.3 Identificación de las relaciones entre clases y objetos

La fase de identificación de las relaciones entre clases y objetos, también se conoce como *identificación de colaboraciones*; para determinar cuáles son se utilizaron las preguntas:

¿Qué clases conoce?

¿Con quién colabora?

Gran parte de las respuestas se obtuvieron de los *escenarios*. Un escenario es una secuencia de eventos entre objetos que muestran una parte del comportamiento del sistema; visto desde la perspectiva de los usuarios finales, se puede decir que es una descripción de una función punto. Además los escenarios sirvieron para refinar la estructura y comportamiento de las clases descubiertas.

5.3.1 Descripción de escenarios

El principio de Coad y Nicola [1993]. "Servir con una sonrisa. Como un objeto, yo conozco cosas y hago cosas, todo en beneficio directo o indirecto de alguien que usa el sistema".

El principio anterior es de gran utilidad, aun cuando es difícil ver sonreír a un objeto, permite imaginarse a los objetos con vida para ir buscando sus colaboraciones y lograr el comportamiento deseado del sistema; de esta manera, la descripción de los escenarios se hizo en primera persona.

Se comenzó con el escenario cuando una persona desea usar el sistema e interacciona con él.

Yo soy un usuario:

- Inicio la ejecución del sistema digitizador.
- Después puedo seleccionar la opción de crear un mapa nuevo o continuar trabajando con un mapa que ya está capturado.
- Configuro el sistema de coordenadas con la ayuda de la tableta.
- Puedo añadir en el mapa de manera aleatoria puntos, segmentos y polígonos.
- Puedo eliminar puntos, segmentos y polígonos.
- Puedo añadir símbolos y textos.
- Puedo eliminar símbolos y texto.
- Puedo imprimir el mapa.
- También guardar el mapa en algún formato.
- Puedo salir del sistema y regresar en otro momento.

El escenario anterior describe con palabras comunes, como un usuario desea que se comporte el sistema y permite descubrir las abstracciones que participan en una actividad. Analizando la descripción anterior se puede ratificar que las abstracciones: **Mapa**, **Punto**, **Segmento**, **Polígono** y **Tableta** forman parte del modelo del sistema.

La formulación de los escenarios, se hizo de acuerdo a la descripción de las *funciones punto* del sistema. Una función punto para el propósito de este análisis es alguna actividad primaria del sistema en respuesta a algún evento, también se considera como una transformación que el sistema hace al medio ambiente. La manera de desarrollar los escenarios se hizo siguiendo el orden de las funciones punto más generales a las más particulares; de esta manera se fue incrementando y refinando el modelo del sistema.

Descripción del escenario cuando un usuario configura la tableta.

Lo primero que hace un usuario al inicio de la ejecución del sistema es configurar el sistema de coordenadas que debe manejar la tableta:

Yo soy un usuario:
- **Después de tener el mapa que quiero digitalizar sobre la tableta, configuro la tableta para manejar el sistema de coordenadas del mapa.**

Yo soy una tableta:
- **Tengo un sistema de coordenadas locales.**
- **Si quieren que maneje otro sistema de coordenadas, me tienen que indicar cuando menos tres puntos de control .**

Yo soy un usuario:
- **Puedo mover el lápiz magnético de la tableta, para indicar cuando menos tres puntos del mapa que conozco sus coordenadas.**

Yo soy una tableta:
- **Puedo saber las coordenadas de los puntos que el usuario me indique, en mi sistema local de coordenadas.**
- **Teniendo las coordenadas de los puntos en los dos sistemas puedo transformar mis coordenadas al sistema de coordenadas que maneja el mapa.**
- **Puedo dar las coordenadas de mi lápiz magnético en el sistema de coordenadas del mapa.**

En la elaboración de un escenario se debe observar que existen objetos *clientes* y *servidores*, donde los primeros usan los recursos de los segundos. Para caracterizar el comportamiento de un objeto, en los escenarios se considera los servicios que proporciona y con cuáles objetos colabora.

Una de las responsabilidades de la clase **Tableta** es la de configurar el sistema de coordenadas, para esto tendrá que realizar la operación de **configurar**. Otra es dar la posición del **lápiz magnético**, por lo que debe tener un atributo de tipo **Coordenadas** para almacenar la posición. Cuando una clase está compuesta por otros atributos que pertenecen a otra clase se dice que existe una relación de *tiene* entre ellas.

Existen diferentes tipos de relaciones entre clases, una de ellas es la de *tiene*; esta relación significa que cuando se instancie la clase Tableta, se creará un objeto compuesto por otro de tipo Coordenadas. La representación gráfica de la relación de *tiene* se muestra en el *diagrama de clases* de la Figura 5.2, la cual se denota por una línea con un círculo relleno. Si se desea resaltar los atributos y métodos más significantes en un diagrama se pueden incluir dentro de los iconos. Los métodos se diferencian de los atributos por tener paréntesis.

Figura 5.2 Relación entre la clase Tableta y Coordenadas.

En los diagramas de clases, también se puede incluir la cardinalidad de las clases (el número de objetos a instanciar) y el propósito de las relaciones. En la Figura 5.2, se tienen una cardinalidad *uno* a *uno* y el propósito es almacenar la posición del lápiz magnético.

Descripción del escenario para incluir puntos, segmentos y polígonos

Yo soy un usuario:
- Puedo digitalizar un mapa que significa capturar:
  - Puntos.
  - Segmentos.
  - Polígonos.
- Puedo indicar las coordenadas de los puntos, segmentos y polígonos por medio de ir moviendo el lápiz magnético de la tableta sobre el mapa.

Yo soy un mapa:
- Puedo crear una *representación* para los puntos, segmentos y polígonos; pero necesito ayuda para saber sus coordenadas.

Yo soy una tableta:
- Puedo decir en cualquier momento las coordenadas de la posición de mi lápiz magnético.

Yo soy un mapa:
- Estoy compuesto por varios puntos, segmentos y polígonos.
- A mí me pueden añadir puntos, segmentos o polígonos.
- También me pueden eliminar puntos, segmentos o polígonos.

Se puede observar en la descripción del escenario anterior, la existencia de una relación de *tiene* entre la clase **Mapa** y las clases **Punto**, **Segmento** y **Polígono**. Ver Figura 5.3. En el análisis para hacer una simplificación del modelo del sistema se estableció que un mapa está compuesto por puntos, segmentos y polígonos; pero en el diseño se refinó la representación de un mapa utilizando el concepto de *planos sobrepuestos*, donde un mapa esta compuesto de varios planos de información y los planos son los que están compuestos de puntos, segmentos y polígonos.

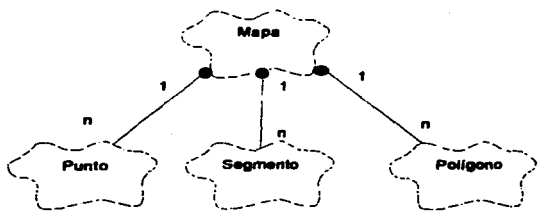


Figura 5.3 Diagrama de clases, la clase Mapa *tiene* puntos, segmentos y polígonos.

Además de las relaciones descritas anteriormente la clase **Mapa** debe tener alguna relación con la clase **Tableta**, por el momento se supondrá que es de tipo *asociación*; ¿por qué?, un mapa debe saber cuál tableta le puede ayudar a determinar las coordenadas de sus puntos, segmentos y polígonos y la tableta debe conocer a qué mapa le envía las coordenadas; ver Figura 5.4. La relación de asociación entre clases se representa en los diagramas de clases uniendo las dos clases con una línea.

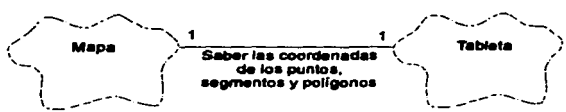


Figura 5.4 Relación de *asociación* entre la clase Mapa y Tableta.

Descripción del escenario para incluir un punto

Yo soy un usuario:
- Necesito incluir un punto en mi mapa.
- Le digo al mapa que añada un punto.

Yo soy un mapa:
- A mí me pueden añadir un punto.
- Si me piden añadir un punto debo obtener su localización, para eso necesito que alguien me ayude a saber las coordenadas del punto.

Yo soy una tableta:
- Puedo ayudar al mapa a saber las coordenadas del punto.

Yo soy un mapa:
- Le pido a la tableta que obtenga las coordenadas del punto.

Yo soy una tableta:
- Espero las coordenadas del punto hasta que el usuario presiona el botón de mi lápiz magnético.
- Puedo dar las coordenadas señaladas por el usuario.

Yo soy un usuario:
- Muevo el lápiz magnético de la tableta a la posición del punto y presiono un botón, en ese momento la tableta sabe las coordenadas del punto.

Yo soy un mapa:
- Recibo las coordenadas y creo al punto con esas coordenadas.
- Pero necesito ayuda para transformar el punto a mi sistema de coordenadas.

Yo soy una tableta:
- Tengo la información suficiente para transformar las coordenadas de la posición de mi lápiz magnético al sistema de coordenadas del mapa.
- Puedo dar las coordenadas transformadas.

Yo soy un mapa:
- Obtengo las coordenadas transformadas.
- Le asigno las coordenadas al punto.
- Almaceno el punto.

Debido a que los escenarios pueden ser considerados como descripciones no formales, los *diagramas de objetos* son utilizados para una especificación más formal, estos se usan para mostrar la existencia de los objetos y sus colaboraciones por medio de *envío de mensajes*. Para cada escenario, cuando fue posible se obtuvo un diagrama de clases y otro de objetos; por ejemplo en la Figura 5.5 y 5.6, se muestra en diagramas el escenario anterior. Un objeto se representa por medio de un icono en forma de nube, pero a diferencia de una clase se dibuja con línea continua, los objetos que se envían mensajes se unen con una línea. El mensaje se describe con una etiqueta y una línea con dirección determina a quién se le envía, el orden de envío se indica numerando los mensajes. El nombre de un objeto se puede definir de tres maneras distintas:

- **N** Nombre del objeto.
- **:C** Nombre de la clase a que pertenece el objeto.
- **N:C** Nombre del objeto y de la clase.

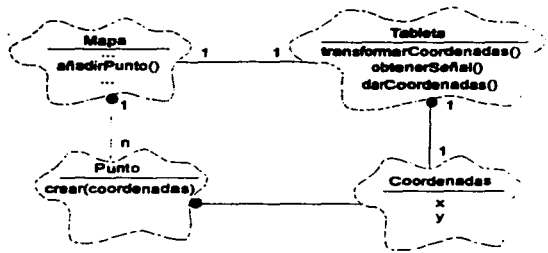


Figura 5.5 Diagrama de clases, que muestra las relaciones entre las clases Mapa, Tableta, Punto y Coordenadas.

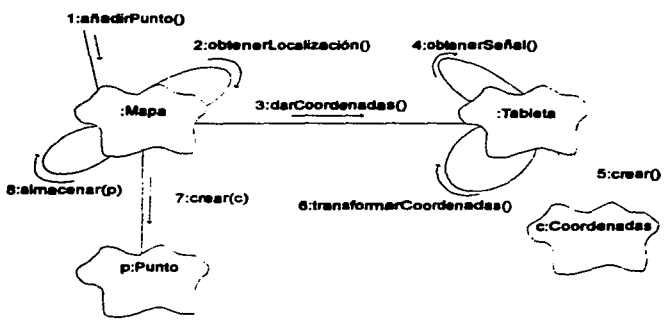


Figura 5.6 Diagrama de objetos, que muestra como se añade un punto a un mapa.

Es recomendable tener un estándar para la asignación de los nombres de las abstracciones. La notación utilizada en esta tesis es la siguiente: los nombres de las clases se escribirán con la primera letra en mayúscula y en singular, excepto para el caso de la clase Coordenadas, que define un par de coordenadas. Los objetos irán en minúsculas y en singular. Los nombres de los métodos deben estar siempre asociados a un verbo y en minúsculas. En el caso de requerir un nombre compuesto por varias palabras para una clase, objeto ó método se usó al terminar cada palabra la siguiente en mayúscula.

También se debe dedicar un poco de tiempo en la selección de los nombres, con el objetivo de que sean lo más apropiados para lo que se quiere representar; esto tiene las ventajas de permitir a otros analistas entender más rápido los diagramas y ayudar en la legibilidad del código de los programas.

Descripción del escenario para incluir un segmento

Un objeto mapa para incluir algún punto, segmento ó polígono se necesita que el objeto tableta le diga las coordenadas, pero se debe considerar que la tableta no puede enviar al mismo tiempo todas las coordenadas de los puntos que forman parte de un segmento.

Yo soy un usuario:
- Necesito incluir un segmento en mi mapa.
- Le digo al mapa que añada un segmento.

Yo soy un mapa:
- Si me piden añadir un segmento, necesito ayuda para saber las coordenadas de los puntos del segmento.

Yo soy un segmento:
- Puedo formar parte de un mapa.
- Estoy compuesto por una lista de puntos.

Yo soy una tableta:
- Espero las coordenadas de los puntos señaladas por el usuario.
- Puedo enviar las coordenadas del segmento al mapa.

Yo soy un usuario:
- Muevo el lápiz magnético de la tableta a donde inicia el segmento que quiero incluir, después presiono el botón de añadir un punto, luego lo muevo al segundo punto del segmento y al mismo tiempo presiono el botón de añadir punto, así continuo hasta presionar el botón de fin de segmento.

Yo soy una tableta:
- Puedo obtener las coordenadas señaladas por el usuario y enviarlas al mapa.
- Puedo decirle al mapa cuándo termina el segmento.

Yo soy un mapa:
- Pregunto a la tableta si el segmento terminó, si la respuesta es no entonces le pido las coordenadas del lápiz magnético.
- Después creo un punto y lo incluyo en el segmento.

Yo soy un segmento:
- Añado los puntos que me están llegando en mi lista de puntos.

Yo soy un mapa:
- Cuando la tableta me dice que el segmento fue capturado lo almaceno.

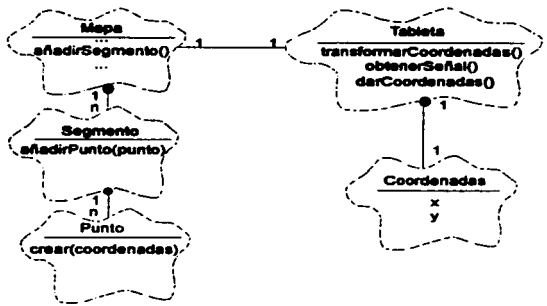


Figura 5.7 Diagrama de clases, que muestra las relaciones entre las clases **Mapa**, **Segmento**, **Punto**, **Tableta** y **Coordenadas**.

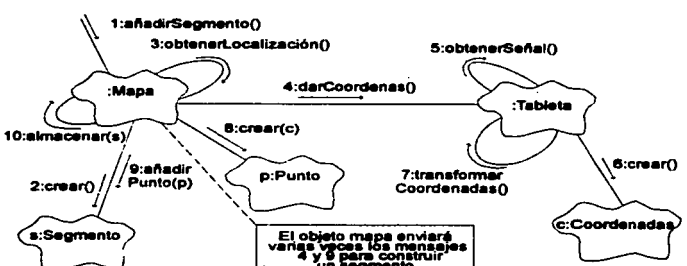


Figura 5.8 Diagrama de objetos para añadir un segmento a un mapa.

La descripción del escenario anterior se muestra en los diagramas de la Figura 5.7 y 5.8, donde el primero corresponde a la vista estática y el segundo a la dinámica.

Otro tipo de diagrama que se utilizó en el análisis fue el *de interacción*, éste sirve para ilustrar el *envío de mensajes* entre objetos de manera dinámica, pero la ventaja que tiene con respecto a los de objetos, es la especificación de la secuencia de envío de mensajes en tiempo relativo.

En un diagrama de interacción en la parte superior se escriben los nombres de los objetos que participan en el escenario. Por cada objeto se dibuja una línea vertical punteada para indicar el transcurso del tiempo. Un mensaje que se envía se representa por medio de una línea horizontal con dirección y una etiqueta para indicar su nombre, dependiendo dónde inicia y termina es quién envía y lo recibe; entre más arriba se encuentren significa que ocurre primero. En la Figura 5.9 se muestra el diagrama de interacción que describe el escenario de añadir un segmento a un mapa.

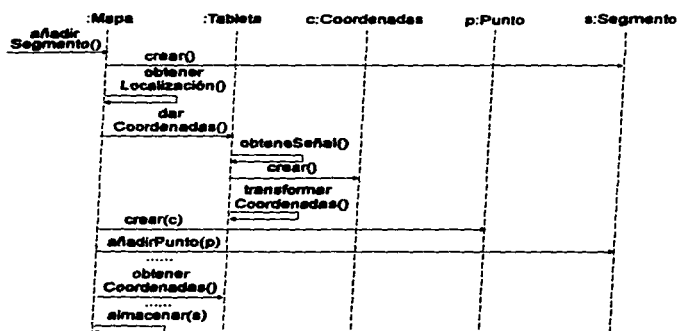


Figura 5.9 Diagrama de interacción, que muestra el envío de mensajes entre los objetos: mapa, tableta, coordenadas, punto y segmento.

El diagrama de Figura 5.9, tiene la desventaja de no especificar el envío de mensajes bajo condiciones o iteraciones; como es el caso del escenario añadir un segmento, donde no se puede describir los mensajes dar Coordenadas, crear y añadir Punto en un ciclo.

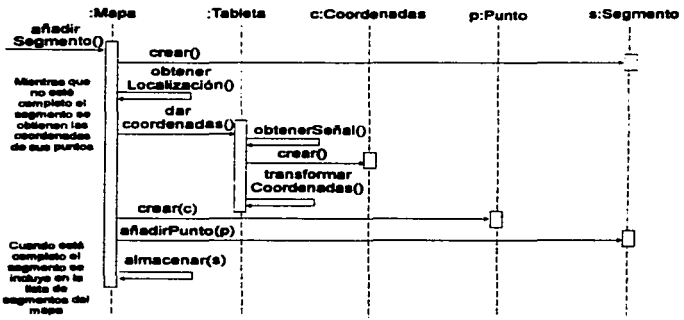


Figura 5.10 Diagrama de interacción que describe la manera de añadir un segmento, el cual contiene la especificación de las condiciones de envío de mensajes y el foco de control.

Booch [1993] propone para especificar el envío de mensajes bajo condiciones y/o ciclos, incluir dos elementos más en los diagramas de interacción. El primero es la descripción de las condiciones y los ciclos, que consiste en agregar al lado izquierdo una especificación en lenguaje natural ó con la sintaxis del lenguaje programación seleccionado, las causas de los envíos de mensajes. El segundo es indicar el *foco de control*, por medio de barras sobre las líneas verticales del tiempo; de esta manera se puede saber cuando un mensaje se envía cuáles se desencadenan. Por ejemplo el mensaje de dar Coordenadas, enviado del objeto mapa a tableta ocasiona que se desencadenen obtener Señal, crear y transformar Coordenadas.

El diagrama de interacción que especifica la manera de añadir un segmento y contempla los dos aspectos anteriores se observa en la Figura 5.10.

Descripción del escenario para eliminar los puntos que fueron mal capturados de un segmento.

Debido que en los requerimientos del sistema se establece que se puedan corregir los datos, los diagramas de las Figuras 5.7, 5.8 y 5.10 se refinaron. La manera propuesta para corregir un segmento es eliminar el último punto que fue capturado, así si se quiere corregir una parte del segmento se eliminarán los siguientes y después se captura de nuevo.

Yo soy un usuario:
- Necesito incluir un segmento en mi mapa.
- También puedo corregir el segmento, eliminando los puntos que están mal capturados.

Yo soy un mapa:
- Para añadir un segmento pido ayuda a la tableta para obtener las coordenadas del segmento, pero además puedo eliminar los puntos del segmento que fueron mal capturados.

Yo soy una tableta:
- Puedo ayudar al mapa a obtener las coordenadas del segmento.
- También puedo saber cuándo un usuario quiere eliminar un punto.

Yo soy un segmento:
- Además de añadir punto, también puedo eliminarlo de mi lista.

Si se desea eliminar el último punto que se añadió, antes de enviar el mensaje de dar Coordenadas (que obtiene las coordenadas del nuevo punto), el objeto mapa debe preguntar a la tableta si se quiere capturar o eliminar un punto.

Refinando la asignación de responsabilidades, se incluyó un método en la clase Tableta llamado dar Evento, este se utilizará para saber si el usuario desea añadir o eliminar un punto.

En los diagramas de interacción que propone Booch [1993], se especifican las iteraciones y condiciones de envío de mensajes entre objetos por medio del lenguaje natural o el de implantación; pero para que sean más explícitos y más fácil de mapear al lenguaje de implantación, se propuso incluir lo siguiente: especificar el foco de control de los mensajes que se envían los objeto a si mismo con otra barra, los mensajes que están dentro de un ciclo o de una condición, se indicará el inicio y fin de la iteración por medio de una línea horizontal sobre el foco de control; también en lugar que la especificación de las condiciones esté al lado izquierdo del diagrama se pueda escribir arriba de las etiquetas de los mensajes. El diagrama de interacción de añadir un segmento a un mapa que incluye lo propuesto se muestra en la Figura 5.11.

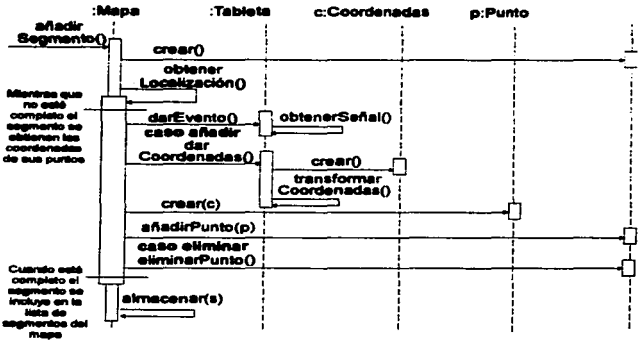


Figura 5.11 Diagrama de interacción que muestra el escenario de añadir un segmento. Incluye la especificación explícita de los mensajes dentro de un ciclo y el foco de control para los mensajes que se envían al mismo objeto.

Descripción del escenario para añadir un polígono

Para la construcción de un polígono se propuso que estos fueran capturados a partir de los segmentos existentes. La descripción del escenario para añadir un polígono a un mapa es la siguiente:

Yo soy un usuario:
- Necesito añadir un polígono al mapa.
- Puedo indicar cuáles segmentos forman parte del polígono.

Yo soy un mapa:
- Si me piden añadir un polígono, puedo buscar los segmentos que lo forman.

Yo soy un polígono:
- Puedo formar parte de un mapa.
- Estoy compuesto por un conjunto de segmentos.
- Puedo añadir y eliminar segmentos en mi lista.

Yo soy una tableta:
- Puedo ayudar al mapa a determinar cuáles son los segmentos del polígono.

Yo soy un mapa:
- Puedo incluir al polígono.

Los diagramas que especifican el escenario para añadir un polígono se muestran en las Figuras 5.12 y 5.13.

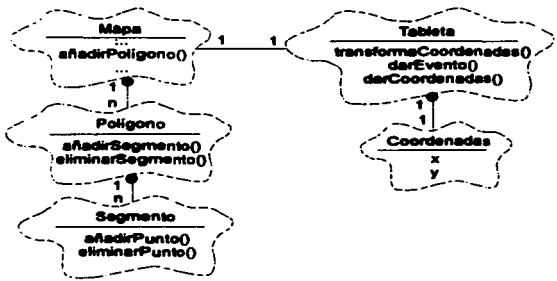


Figura 5.12 Diagrama de clases que muestra las relaciones entre las clases Mapa, Polígono y Segmento.

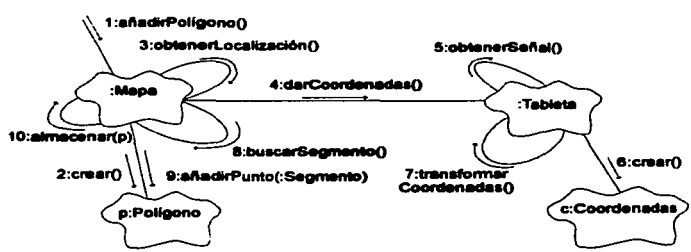


Figura 5.13 Diagrama de objetos que muestra como se añade un polígono a un mapa.

Descripción del escenario para eliminar un punto, segmento ó polígono

Yo soy un usuario:
- Necesito corregir un mapa, para ello puedo modificar o eliminar un punto, segmento o polígono del mapa.

Yo soy un mapa:
- Si me piden eliminar un punto, segmento ó polígono necesito ayuda para saber cuál es.

Yo soy un usuario:
- Puedo indicar cuál elemento deseo eliminar.

Yo soy una tableta:
- Puedo ayudar a indicarle cuál elemento se desea eliminar.
- Espero hasta que el usuario mueva el lápiz magnético para indicar el elemento y presione un botón de seleccionar.

Yo soy un mapa:
- Le pido a la tableta las coordenadas de la posición del elemento, después lo busco en mi lista y lo elimino.

En las Figuras 5.14, se muestra el diagrama de objetos para eliminar un punto del mapa, no se incluyen los diagramas de eliminar un segmento y polígono por ser muy semejantes.

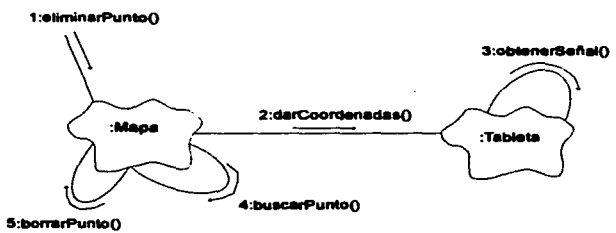


Figura 5.14 Diagrama de objetos que describe el escenario de eliminar un punto del mapa.

5.3.2 Diagramas de transición de estados

Para describir la vista dinámica del modelo de sistema, también se usaron los *diagramas de transición de estados*, estos permiten entender con más precisión el sistema completo y/o el comportamiento de abstracciones individuales. Un diagrama de transición de estado, consiste de un conjunto de estados y eventos. El estado del objeto representa todas las propiedades estáticas de un objeto con sus valores dinámicos asociados en un momento dado. Un evento es una acción externa que actúa sobre el sistema o una acción de un objeto. Este tipo de diagramas se usan para documentar cuáles eventos causan un cambio de estado y las acciones que resultan debido a ese cambio, durante el análisis se usaron para capturar el comportamiento global del sistema y en el diseño para modelar las clases individuales.

El icono utilizado para representar el estado de una abstracción es un rectángulo con las esquinas redondeadas y una etiqueta con su nombre, los eventos se indican con una línea con dirección; por ejemplo en la Figura 5.15, se muestra un diagrama de transición de estados del sistema.

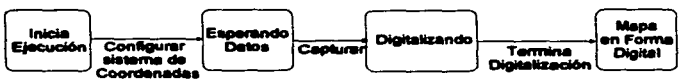


Figura 5.15 Diagrama de transición de estados que describe el comportamiento dinámico del sistema.

5.4 Especificación de las interfaces de clases y objetos

Después de haber obtenido los diagramas de las abstracciones, lo siguiente que se realizó fue la documentación textual para cada una de las clases, objetos y métodos; para ello se utilizaron los *esquemas*.

Los esquemas de clases son de gran utilidad, porque sirven para recopilar los atributos y operaciones de cada clase; tienen la ventaja de ser independientes de la sintaxis de los lenguajes de programación. Para obtener la información de cada abstracción se recurrió a los diagramas de clases, de objetos, de interacción y de transición de estados. En la etapa de implantación se utilizaron los esquemas para mapear las clases al lenguaje de programación. Por ejemplo el esquema de la clase Mapa se muestra a continuación:

Nombre:
Mapa
Responsabilidad:
- La responsabilidad de la clase Mapa es representar a un mapa temático en forma digital.

Operaciones:
- crear
- asignarNombre
- añadirPunto
- añadirSegmento
- añadirPolígono
- eliminarPunto
- eliminarSegmento
- eliminarPolígono

Atributos:
- nombreMapa : Texto
- puntos: Lista[Punto]
- segmentos: Lista[Segmento]
- polígonos: Lista[Polígono]

Cardinalidad: n
Persistencia: persistente

Otra forma alternativa de documentación para los esquemas de clases es utilizar el lenguaje de programación; pero se aconseja realizarlo cuando se tenga muy avanzada o terminada la etapa de análisis, debido a que en fases tempranas del desarrollo se tienen interfaces de clases inestables.

5.5 La interfaz del sistema

Después de haber realizado el análisis de la parte del dominio del problema, y haber descrito los servicios que ofrece el sistema, los cuales se obtuvieron a partir de los escenarios, se pasó al análisis de la interfaz.

En el análisis de la interfaz del sistema, se descubren las abstracciones que ofrecen los servicios a los usuarios finales, es decir, la interfaz es una capa de software que permite comunicar los usuarios con la aplicación; por ejemplo en las Figuras 5.16 y 5.17, se muestra un diagrama de clases y otro de objetos que modelan la captura de un punto pero con los componentes de interacción humana, en estos diagramas se tiene tres abstracciones nuevas. El Menú tiene la responsabilidad de permitir seleccionar una opción, y notificar a la clases correspondiente del dominio del problema para que realice la operación; la Ventana es utilizada para desplegar los datos capturados en forma de mapa; el Cursor indicar la posición del lápiz magnético de la tableta sobre la Ventana.

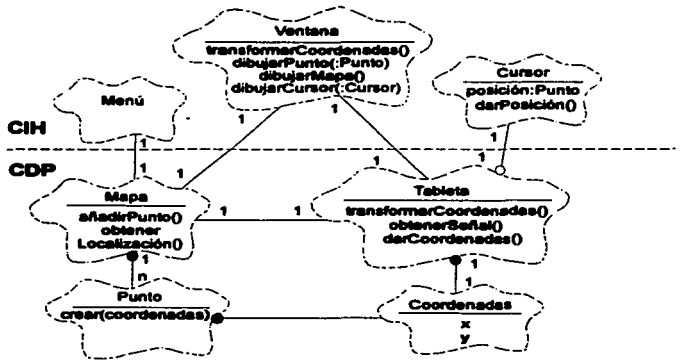


Figura 5.16 Diagrama de clases que muestra las relaciones entre las clases del dominio del problema y las clases de interacción humana.

Coad y Yourdon [1993], en su metodología de análisis y diseño orientado a objetos separan los componentes del dominio del problema y los de interacción humana, para esto encierran las clases de un mismo tipo en un rectángulo; debido a que Booch [1993], en su notación no contempla la distinción de clases, se usó una línea punteada para separarlas y se etiquetaron con CDP (clases del dominio del problema) y CIH (clases de interacción humana); ver Figura 5.16. La ventaja de hacer una distinción entre los dos componentes es tener una mayor claridad en el modelo del sistema, hacer modificaciones con mayor facilidad y además localizar las abstracciones para ser reutilizadas. En los diagramas de objetos se utilizaron las etiquetas ODP (objetos del dominio del problema) y OIH (objetos de interacción humana).

En el diagrama de clases de la Figura 5.16, se muestra otro tipo de relación entre clases conocida como relación de *usa*, esta significa que la clase Tableta para realizar una operación puede utilizar los servicios de la clase Cursor. La diferencia con respecto a la relación de *tiene*, es que una clase no es parte de la otra, sólo le ayuda a completar sus responsabilidades. En los diagramas la relación de *usa*, se representa uniendo las dos clases con un línea y un círculo del lado de la clase que utiliza los servicios de la otra. Esta relación también se puede mostrar implícitamente, como es el caso de los métodos dibujar Punto y dibujar Cursor de la clase Ventana, que significa que esta utilizará las clases Punto y Cursor.

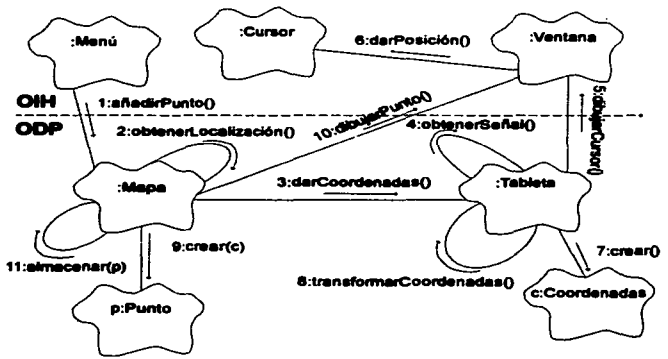


Figura 5.17 Diagrama de objetos que muestra la parte de interacción humana.

6 Diseño del sistema

En esta fase se inventan las abstracciones y los mecanismos para proporcionar el comportamiento que el modelo requiere, además se crea la arquitectura del sistema para que sirva como línea base en la implantación. Al igual que en el análisis es conveniente dividir las tareas del diseño por los diferentes tipos de clases.

6.1 Identificación de clases y objetos

En esta tesis para fines de implementación se decidió utilizar el modelo de *planos sobrepuestos* para representar a los mapas en forma digital; en la sección 1.1, se describió en que consiste este modelo. Se puede observar que cualquier capa de información de un mapa puede ser cartografiada como un plano de puntos, de segmentos o de polígonos; por ejemplo si se quiere digitalizar de una región las ciudades, la red de caminos y las áreas de cultivo; para facilitar la adquisición y el análisis de los datos se usa los siguientes planos de información: uno de puntos, donde un punto representa una ciudad; de segmentos para los caminos y otro de polígonos para las superficies de cultivo.

A partir del concepto de planos sobrepuestos se llegó a la conclusión que la clase **Mapa**, está compuesta de una o varias clases de **Plano de Puntos**, de **Plano de Segmentos** y de **Plano de Polígonos**; el número de planos que tenga la clase **Mapa** dependerá de las capas de información que se quiere capturar; de esta manera las abstracciones **Plano de Puntos**, **de Segmentos** y **de Polígonos**, pasaron a formar parte del modelo del sistema. En la Figura 6.1, se muestra que por cada plano de información que contenga un mapa se utilizará una clase para representarlo.

Por otro lado, otro factor que se debe considerar al representar a los objetos geográficos en un formato digital es la *escala* de los mapas, debido a que no es lo mismo dibujar una ciudad a escala 1:5,000 que a 1:1'000,000. Un mapa con la primera escala, las calles, manzanas y otros detalles se observarán perfectamente; en la segunda escala una ciudad probablemente se observará como un punto. Entonces dependiendo de como estén dibujados los objetos en un mapa será seleccionada su representación.

De lo descrito anteriormente se puede observar que es difícil construir un sistema para obtener una base de datos de cualquier mapa; debido a que no se sabe de cuántas capas de información consta y cuales son los temas de las capas.

Para poder digitalizar un mapa general, se decidió modelar un *mapa simple* compuesto por tres planos: de puntos, de segmentos y de polígonos; debido a que cualquier capa de información puede ser representada dentro de alguno de esos planos.

La desventaja que se encontró en el modelo de un mapa simple, fue la dificultad de poder hacer un análisis posterior de la información; debido a que un plano puede contener más de una capa de información. Para solucionar este problema, en la sección 6.2.1 se propone una manera para digitalizar un mapa específico; la cual permite tener cada capa de información asociada a un plano y además capturar atributos temáticos de los objetos geográficos. Esto fue posible gracias a las bondades del modelo orientado a objetos, por facilitar la reutilización y extensión tanto en el diseño del sistema, como del código de las clase existentes.

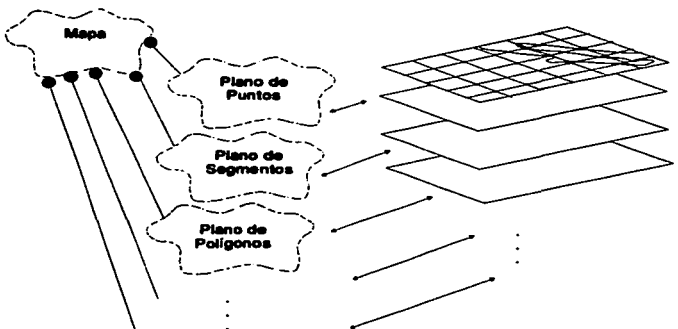


Figura 6.1 Modelo propuesto para representar un mapa en forma digital.

6.1.1 Manejo de la persistencia

Debido a que varias instancias de las clases del modelo del sistema son persistentes, se investigó una manera de implementarlas. Booch [1993] define *persistencia* como la propiedad de un objeto a través del cual su existencia trasciende en el tiempo, (continua su existencia después de que su creador deja de existir) y/o espacio, (los objetos se pueden mover de localidades de memoria donde fueron creados). Son pocos los lenguajes de programación orientados a objetos que proporcionan un mecanismo para el manejo de la persistencia, Smalltalk es una notable excepción. En el caso de C++ la persistencia no es parte del lenguaje; por lo tanto debe implantarse de alguna manera.

No es fácil manejar la persistencia, debido a que implica solucionar varios problemas. Es común dentro de la implementación utilizar los apuntadores o referencias para describir las relaciones entre los objetos; cuando se declara un objeto persistente este debe ser capaz de resolver las referencias que tiene con los otros objetos, lo que implica resolver dos problemas. El primero es cuando se carga a memoria principal un objeto, por que se debe buscar en la memoria a los objetos con los cuales se relaciona y además debe obtener sus referencias. El segundo es la representación de las referencias de manera independiente, cuando un objeto está en memoria tiene una dirección específica, si se almacena y después se recupera es probable que el sistema lo cargue en otra localidad, por tanto debe existir una manera de representar las referencia de manera independiente a las direcciones de memoria.

Otra de las actividades que se hizo durante la fase de diseño, fue investigar en las bibliotecas de clases y las herramientas de desarrollo que proporcionan; con el motivo de reutilizarlas e incorporarlas en el modelo del sistema.

Existen manejadores de bases de datos comerciales que resuelven los problemas de la persistencia, uno de ellos es *POET*. La ventaja de usar un manejador de base de datos existente es que permite a los desarrolladores disminuir la complejidad del problema; debido a que pueden dedicar más tiempo a modelar los aspectos principales del sistema y dejar los aspectos secundarios para ser resueltos por las herramientas.

En el diagrama de clases de la Figura 6.2, se incorporó la clase *PtBase* (clase Base de Datos), ésta es proporcionada por el manejador de base de datos orientada a objetos *POET*, su responsabilidad es almacenar y recuperar objetos en memoria secundaria. En el diagrama también se distingue que la clase pertenece a las de bases de datos (CBD), de acuerdo con la clasificación de Coad et al [1995].

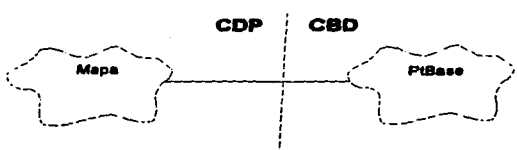


Figura 6.2 Diagrama de clases que muestra la relación entre la clase Mapa y *PtBase*.

6.2 Identificación de la semántica de clases y objetos

En la identificación de la semántica de las abstracciones pero como fase del diseño, se utilizaron los diagramas obtenidos en el análisis y los modelos de representaciones descritos en la sección 1.2, para obtener las estructuras de las clases a nivel de implantación. A continuación se describen las estructuras propuestas para las clases **Mapa**, **Punto**, **Segmento** y **Polígono**:

6.2.1 Representación de mapas

Debido que en las aplicaciones de los SIG se manejan diferentes mapas temáticos, en esta sección se propone una manera de reutilizar el diseño y el código del sistema para usarlo en la captura de diversos mapas.

Como se mencionó anteriormente, se empezó por modelar un sistema para capturar un mapa, formado por tres capas de información, el cual está representado por la clase **Mapa Simple**, que contiene las clases **Plano de Puntos**, **Plano de Segmentos** y **Plano de Polígonos**; esta representación de un mapa digital tiene la ventaja de poder capturar cualquier objeto geográfico, pero tiene la desventaja de que los planos de información pueden contener más de un rasgo, dificultándose el análisis y adquisición de los datos. En la Figura 6.3, se muestra el diagrama para representar un mapa simple. Como se observa se incluye notación para especificar en detalle la relación de *tiene*; en este caso la clase **Mapa Simple** contiene las clases por referencia debido a que el rectángulo está sin colorear, si se encuentra coloreado es una contención física.

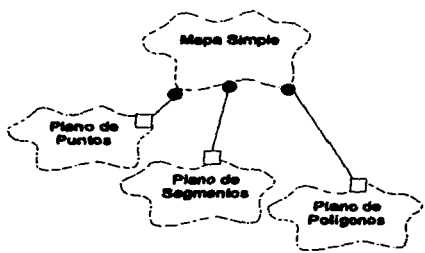


Figura 6.3 Diagrama de clase que ilustrar el modelo de un *mapa simple*.

Para modelar diferentes tipos de mapas en forma digital, se definió la clase *abstracta* **Mapa**, que define las características de un mapa digital genérico. Para representar un mapa digital específico se reutilizará la estructura y comportamiento de la clase **Mapa** por medio de la relación de *herencia*, de esta manera se aprovecharán las operaciones para capturar la posición del los objetos espaciales del mapa específico.

Por ejemplo: en el diagrama de la Figura 6.4, en la parte de CFT (clases fuera de tiempo) se muestran algunas clases que son definidas en futuros requerimientos; si se quiere capturar la información contenida de un mapa urbano conformado por los lotes de los habitantes y las colonias; se definirá una clase **Mapa Urbano** que representa el mapa digital; después se debe identificar las diferentes capas de información que contiene; siguiendo el ejemplo, en este caso sólo contiene dos y están representadas por la clase **Plano de Lotes** y **Plano de Colonias**.

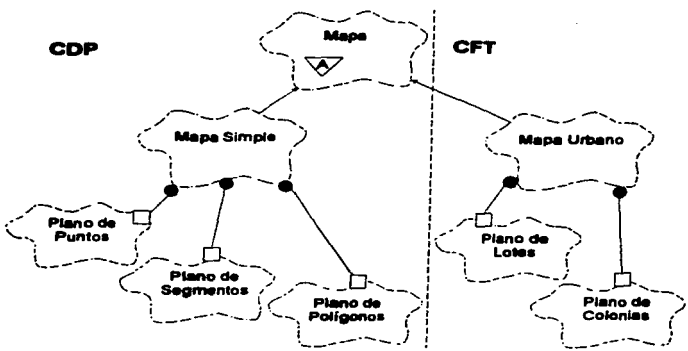


Figura 6.4 Diagrama de clases, utilizado para mostrar el modelo de los mapas.

Como se puede ver en la Figura 6.4, en la representación de los diferentes mapas se utilizó la relación de *herencia*; ésta es una de las características más importantes del paradigma orientado a objetos por que proporciona:

- Una manera de clasificación. Modela los problemas en una forma jerárquica, ésta es una forma de clasificación usada en muchas actividades; facilitando la comprensión de los problemas.
- Reutilizar código. Permite aprovechar la estructura y operaciones para definir otras abstracciones; por ejemplo en el caso de que se quiere representar un mapa de uso de suelos, se define una clase Mapa de Suelos y esta aprovechará la estructura y funcionalidad de la clase Mapa.
- Mantener funcionalidad en las clases. Cuando se quiera extender la funcionalidad de alguna clase no es necesario modificar el comportamiento de *clase base*; mejor se realizan los cambios en la *clase derivada*, garantizándose el correcto funcionamiento del código existente.

La relación de herencia en los diagramas de clases se representa por medio de una línea con dirección, en la clase que inicia la línea se le conoce como *clase derivada* y en la que termina como *clase base*.

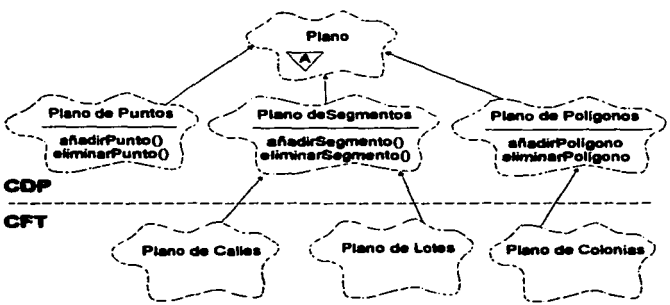


Figura 6.5 Diagrama de clases utilizado para mostrar el modelado de los diferentes tipos de planos.

Para modelar los diferentes planos de información también se utilizó la relación de herencia. Como se mencionó anteriormente la clase **Mapa Simple**, está compuesta de las clases **Plano de Puntos**, **Plano de Segmentos** y **Plano de Polígonos**. En el diagrama se la Figura 6.5, se muestra que la clase abstracta **Plano**, la cual se especializa para definir las clases **Plano de Puntos**, **Plano de Segmentos** y **Plano de Polígonos** y éstos a su vez son utilizados para representar otros planos más específicos. En el ejemplo de la digitalización de un mapa urbano, después de identificar las capas de información, para cada una de ellas se debe investigar que tipo de plano se puede utilizar para representarla; de esta manera se logra separar la información espacial para facilitar su adquisición. Para el caso de la capa de información de los lotes, como un lote es un polígono, la clase **Plano de Lotes** se especializa de la clase **Plano de Polígonos**.

Otra modificación que se hizo en el modelo del sistema fue cambiar las responsabilidades de añadir y eliminar puntos, segmentos y polígonos, de la clase **Mapa** a las clases **Planos**; con el motivo de que los objetos geográficos queden contenidos en su respectiva capa de información.

6.2.2 Representación de puntos

Como se describió en la sección 1.2.2.1, dentro de un mapa pueden existir diferentes tipos de puntos: sin información, con información no espacial y nodos. Para modelarlos se definió una clase **Punto**.

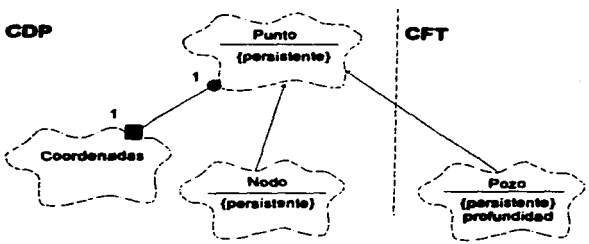


Figura 6.6 Diagrama de clases utilizado para modelar las objetos geográficos que pueden ser representados en un mapa por medio de un punto.

Para los puntos de un mapa que tiene información temática asociada, el sistema deberá permitir capturarla, pero existe el problema de que no se sabe cuál información se desea obtener y de que tipo es. Se puede solucionar este problema extendiendo la clase Punto mediante la relación de herencia, la clase especializada representará el objeto geográfico que incluirá los atributos que interesa capturar. En la Figura 6.6, se describe el modelo para la representación de los objetos geográficos por medio de un punto, este modelo tiene la ventaja de asociar la información no espacial a las abstracciones de una manera fácil y práctica. Por ejemplo, si de un mapa hidrológico se quiere cartografiar todos los pozos y se desea obtener los datos de sus profundidades; entonces se define una clase Pozo, que hereda de la clase Punto y además se le agregará un atributo de profundidad. Ver Figura 6.6.

La desventaja que se encuentra es cuando se trabaja con un lenguaje de programación donde el código debe ser compilado, debido a que se tiene que generar el ejecutable que contenga las nuevas clases lo que puede implicar un trabajo tedioso para los usuarios finales.

6.2.3 Representación de segmentos

Para modelar los objetos geográficos que tienen un patrón lineal en los mapas, se utilizó la clase Segmento, esta consta de una sucesión de puntos ordenados, el primero y último son de tipo Nodo, con el objetivo de utilizarlos en la representación de *modelos de red* (ver sección 1.2.2.2.1). De la misma manera que para los objetos punto, cuando se les asocia información no espacial a los objetos se debe especializar a partir de la clase Segmento.

Otra actividad que se realizó en la fase de identificación de la semántica de las clase, fue transformar las relaciones entre clases de cardinalidad *1* a *n* a una de *1* a *1*, debido a que no se pueden implantarse en forma directa; para mapearlas generalmente se utilizan clases contenedoras. El diagrama de clases de la Figura 5.7, se usó para deducir el de la Figura 6.7, en este último se incluye una clase contenedora de la base de datos *POET* llamada clase *cset* (clase Conjunto), que servirá para almacenar los puntos que pertenecen a un segmento.

La clase *cset* de *POET* es una clase parametrizada, esto significa que denota una familia de clases cuya estructura y comportamiento están definidos independientemente de sus parámetros formales. En el caso de la Figura 6.7, se parametrizó con la clase Punto para obtener la clase Conjunto de Puntos; Booch[1993] llama *instanciación* a esta relación entre clases. El icono para representar una clase parametrizada es de una clase pero con un rectángulo trazado con línea punteada en la parte superior derecha; para representar una clase instanciada, se traza el rectángulo con línea continua y dentro de este el nombre del parámetro actual a instanciar.

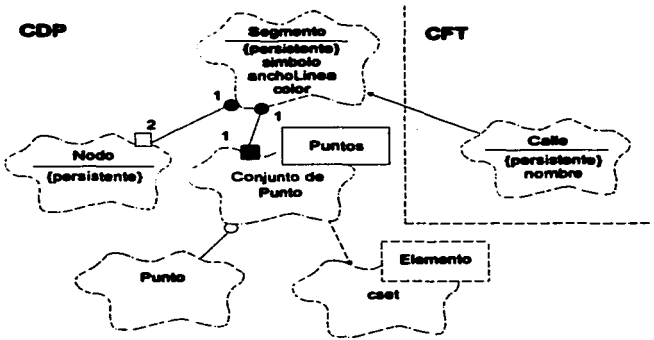


Figura 6.7 Diagrama de clases que muestra la estructura de datos propuesta para los objetos Segmentos.

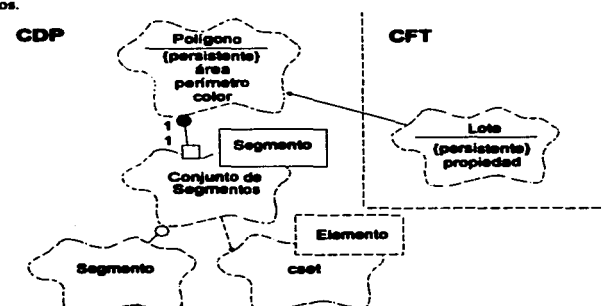


Figura 6.8 Diagrama de clase utilizado para modelar la clase Polígono.

6.2.4 Representación de polígonos

Para representar a los polígonos, en esta tesis se decidió utilizar la representación de *segmento nodo*, con el objetivo de evitar los problemas que tienen los otros modelos. En el diagrama de la Figura 6.8, se puede observar que la clase *caet* está instanciada por la clase Segmento; por tanto se puede concluir que las clases parametrizadas son de gran utilidad para reutilizar el código, por que de una misma clase se está utilizando para generar dos clases: Conjunto de Puntos y de Segmentos.

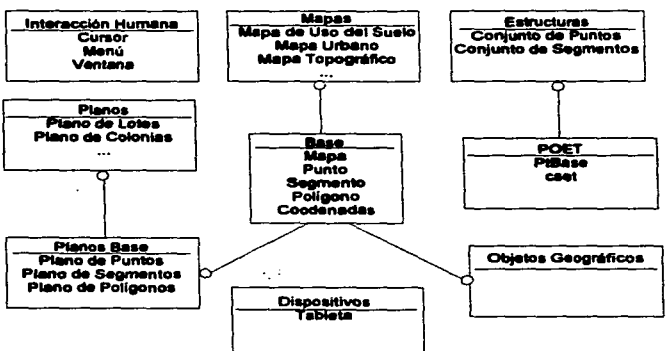


Figura 6.9 Categorías del modelo del sistema.

6.3 Identificación de las relaciones entre clases y objetos

En la fase de identificación de las relaciones entre clases y objetos como parte del diseño se agrupan las clases que tienen características en común dentro de *categorías*. Esta es otra manera de descomponer el modelo lógico de un sistema, pero a un nivel superior de abstracción que el de clases; la descomposición generalmente se realiza cuando el modelo del sistema tiene muchas clases. El icono utilizado para representar un categoría es el de un rectángulo y dentro de este lleva su nombre, además se pueden escribir las clase más representativas que la constituyen. También pueden existir relaciones entre las categorías, por consistencia se define una relación de *uso*, pero se utiliza para mostrar la importancia de conexión entre las categorías. Por ejemplo en la Figura 6.9, se muestran los categorías del modelo del sistema, la relación entre la categorías **Mapas** y **Base** significa que las primeras pueden heredar de las segundas. La manera de como se agrupan las clases fue en respuesta a poderlas reutilizar para capturar diferentes tipos de mapas.

ESTA TESIS NO DEBE SALIR DE LA BIBLIOTECA

7 Evolución e implantación del sistema

Después de haber completado el análisis y diseño del sistema, se pasó a la fase de evolución e implantación. La actividad de programación comienza en el proceso micro en la especificación de interfaces e implantación de clases y objetos, pero es hasta la evolución e implantación donde se completa la codificación del sistema.

En este capítulo se explicará la manera de trasladar el modelo del sistema al lenguaje de programación de C++; no se tiene como objetivo enseñar la sintaxis del lenguaje, para ello se recomienda consultar Stroustrup [1991] y Coplien [1992].

7.1 Definición de las interfaces de clases

En el modelado de un sistema la definición de las interfaces corresponde a la vista estática, estas son especificadas en el análisis y diseño con los esquemas de clases, diagramas de clases y de objetos; lo que resta es traducirlas al lenguaje de programación. La interfaz de una clase es determinada por el conjunto de mensajes que puede responder y se define en un archivo de extensión h, es recomendable que cada una de ellas esté en un sólo archivo, por que es más fácil observar los servicios que proporciona para ser reutilizada en otros sistemas.

La declaración de una clase comienza con la palabra reservada **class**, en seguida se definen sus atributos y métodos, en alguna de las tres partes siguientes: en la **public** (significa que pueden ser accesados por cualquier otras clase), en la **private** (el acceso a los recursos se restringe sólo a la clase) o en la **protected** (es equivalente a **private**, pero se utiliza en la relación de herencia para compartir los atributos y los métodos).

La propiedad que describe la manera de accesar los datos y los métodos se llama *encapsulación*, es decir, los recursos de una clase pueden ser ocultados para otras; esta propiedad permite a los diseñadores poner atención sobre la interacción entre clase, en lugar de los detalles internos que sólo aumentan la complejidad del diseño del sistema; otra disposición que puede ser aprovechada es cuando se tiene una interfaz estable, por que se pueden realizar cambios en los detalles internos de la clase, sin afectar los servicios que proporciona (los servicios seguirán existiendo sólo cambia la manera de como ejecutarlos); en el caso de mejorarlos resultarán ser más eficientes. La propiedad de encapsulación también se le conoce como *ocultamiento de la información*.

A manera de ejemplo, la definición de la interfaz de la clase **Segmento** en el lenguaje C++ se ilustra a continuación:

```
1:   persistent clase Segmento {
2:   public:
3:       // Constructoras y destructoras
4:       Segmento();
5:       ~Segmento() {};
6:       virtual int  Store(PtDepthMode Mode = PtDEEP);
7:       virtual int  Delete(PtDepthMode Mode = PtSHALLOW);
8:       // Implementadoras
9:       void    anadirPunto(Punto*);
10:      void    anadirPunto(Punto*, int)
11:      void    eliminarPunto();
12:      void    eliminarPunto(int);
13:      // Acceso
14:      Punto* darPunto();
15:      Punto* darPunto(int);
16:      Punto* darSiguientePunto();
17:      float   darLongitud()       const;
18:  protected:
19:      // datos miembro
20:      cset<Punto*>  puntos;
21:      float         longitud;
22:  };
```

La numeración del código no es parte del lenguaje, pero se escribió con el objetivo de hacer referencia en las explicaciones. En la definición de la clase Segmento la palabra *persistent* en la línea 1, no es una palabra reservada; esta clase primero debe ser precompilada por el manejador de bases de datos de *POET*. La palabra *persistent* significa que las instancias de la clase podrán ser almacenadas y recuperadas en la memoria secundaria. Eliminando la palabra *persistent* el código restante es reconocido por el compilador.

Los métodos de una clase en el lenguaje C++, se llaman *funciones miembro* y se definen generalmente en la sección public, para exportar los servicios a otras clases. Las funciones miembro son clasificadas por Coad y Nicola[1993] en tres tipos principales:

- *Constructoras y destructoras*. Las constructoras son las encargadas de inicializar los objetos. Debido a que los objetos son entidades que sólo existen en tiempo de ejecución, estos deberán ser creados o instanciados a través de las operaciones constructoras. Simétricamente a las funciones constructoras existen las destructoras que son invocadas implícitamente cuando se abandona el bloque donde fue creado el objeto; garantizándose de esta manera que el objeto haga una serie de acciones antes de que pierda el hilo de control. En la mayoría de los casos una función destructora es usada para liberar la memoria que estaba usando el objeto.

- *Implementadoras*. Son las que implementan las responsabilidades o servicios ofrecidos por la clase.
- *Acceso*. Son las funciones que observan y cambian el estado de un objeto. Se subdividen en *selectoras* y *modificadoras*, dependiendo si sólo ven el estado o lo modifican. En esta tesis se eligió para el nombre de las funciones miembro selectoras el verbo *dar* más el nombre del atributo que se quiere saber su valor, las modificadoras el verbo *asignar* y el nombre del atributo. Generalmente por cada dato miembro debe existir una función miembro selectora y otra modificadora.

Es recomendable que los datos miembro de una clase estén definidos en la parte **private** o **protected**; con el objetivo de mantener la propiedad de encapsulación. Para el caso que se quiera manipular un dato miembro que implica modificar y/o conocer su valor, se debe realizar a través de las funciones miembro de la clase.

Otra propiedad importante desde el punto de vista de ingeniería de software es la modularización, que consiste en dividir los sistemas complejos de software en piezas simples fáciles de comprender, llamados *módulos*. Las clases son la llave en la descomposición orientada a objetos de un problema y pueden ser vistas como pequeños módulos. Existen dos principales beneficios cuando se tiene un sistema dividido en módulos.

El primero es la reutilización del código para construir nuevos sistemas; se pueden poner las clases dentro de bibliotecas y usar estas piezas pequeñas de código para ensamblar otros sistemas.

El segundo es la facilidad para modificar los sistemas, debido a que es más sencillo entender separadamente cada parte que todo el sistema completo; si se necesita reparar una parte, la modularidad ayuda a confinar la búsqueda del código fuente que tiene un funcionamiento no aceptable. La modificación del software como se sabe es una actividad inevitable debido a su naturaleza evolutiva.

Los objetivos que se persiguen en una descomposición modular es que las clases tengan *alta cohesión* y *bajo acoplamiento*. Una clase tiene alta cohesión si todos sus atributos y métodos están fuertemente relacionados, ésta es una característica implícita en un análisis y diseño orientado a objetos *bien* realizados, debido a que cada clase fue creada para cumplir una responsabilidad en particular. El bajo acoplamiento se refiere a que las clases dependan lo menos posibles de otras; entre más grande sea el acoplamiento que exhiban se dificultará la reutilización, el entendimiento y la realización de pruebas del código; debido a que se caería a un esquema donde las piezas de código son de tamaño considerable, por lo tanto es más difícil su manejo.

7.2 Implantación de relaciones entre clases

Una de las razones por la cual se seleccionó el lenguaje de programación orientado a objetos de C++ para la implantación del prototipo, es la facilidad para mapear el modelo del sistema al código fuente. Para cada una de relaciones entre clases, existe un patrón establecido de código en el lenguaje C++. Para no redundar sobre la codificación, se describe un ejemplo de cada una de las relaciones entre clases encontradas en las fases de análisis y diseño.

7.2.1. Relación de Asociación

La relación de *asociación* significa una conexión entre clases, es decir, las instancias de las clases asociadas pueden comunicar y solicitarse servicios mediante el *envío de mensajes*. Booch[1993] establece que esta relación implica una conexión bidireccional, pero en el lenguaje de C++ se puede expresar tanto en las dos direcciones como en una. La relación de asociación entre las clases **Mapa** y **Tableta** descrita en el diagrama de la Figura 5.4, tiene el siguiente patrón:

```
23:    class Tableta;

24:    class Mapa {
25:    public:
26:        Mapa();
27:        ~Mapa();
28:        //...
29:        void asignarTableta(Tableta*);
30:        //...
31:    protected:
32:        //...
33:        Tableta* tableta;
34:        //...
35:    };
```

Debido que la relación de asociación de la Figura 5.4, establece una comunicación bidireccional entre las clases, pero como no se pueden compilar las dos clases al mismo tiempo, se debe declarar una *referencia hacia delante* (línea **23**) que indica que **Tableta** es una clase y después se definirá. Para completar la especificación de la relación en la parte de los datos miembros de la clase **Mapa**, se incluye una variable de tipo apuntador a un objeto **tableta** (línea **33**), esta variable sirve para referenciar al objeto **tableta** que se le enviarán mensajes por parte de un objeto **mapa**. Hasta el momento sólo se tiene comunicación en un sentido, para el otro sentido la definición de la clase **Tableta** tiene una declaración simétrica a ésta; en tiempo de ejecución se completa la relación inicializando los apuntadores a los objetos **tableta** y **mapa**.

7.2.2 Relación de tiene

La relación de *tiene* establece que una clase representa *todo* y otra *parte de*. En el diagrama de clases de la Figura 5.3, se observa que un mapa simple está compuesto de tres planos de información espacial. Para expresar la relación la clase **Mapa Simple** debe tener tres clases: **Plano de Puntos**, **Plano de Segmentos** y **Plano de Polígonos**. La clase **Mapa Simple** es *todo* y las otras clases son *parte*; en el lenguaje C++ se traduce como:

```
36:    class MapaSimple {
37:    public:
38:        MapaSimple();
39:        ~MapaSimple();
40:        //...
41:    protected:
42:        PlanoPuntos       planoPuntos;
43:        PlanoSegmentos    planoSegmentos;
44:        PlanoPoligonos    planoPoligonos;
45:    };
```

La clase **Mapa Simple** en su parte de la declaración de datos miembro se definen los atributos que son *parte de* (las líneas de la **42** a la **44**). A nivel de objetos esta relación significa que cuando se crea una instancia de tipo **Mapa Simple**, también se crean automáticamente tres objetos referenciados por las variables **planoPuntos**, **planoSegmentos** y **planoPoligonos**.

7.2.3 Relación de herencia

La relación de herencia es uno de los pilares dentro del modelo orientado a objetos; por ser un mecanismo que permite la reutilización de código, es decir, se aprovecha la estructura y comportamiento de las clases existentes para crear otras, sin necesidad de copiar el mismo código.

Para mapear la relación de herencia en el lenguaje de C++, primero se declara la clase *base*, a partir de ésta se construirán las clases *derivadas*; a las derivadas se les agregan los atributos y métodos que la especializan. Por ejemplo en el modelo del sistema digitizador, se encontró una clase abstracta **Mapa**, que representa un mapa digital genérico, de esta clase se partió para construir la clase **Mapa Simple** que contiene tres planos de información.

```
46:    class Mapa {
47:    public:
48:        Mapa();
49:        ~Mapa();
50:        void    asignarTableta(Tableta*);
51:        void    asignarVentana(VentanaMapa*);
52:        void    asignarBaseDatos();
53:        void    asignarTamano(const Rectangulo&);
54:        void    asignarTamanoMaximo(const Rectangulo&);
55:        void    desasignarBaseDatos();
56:        void    obtenerTamano();
57:        Rectangulo&   darTamano();
58:        virtual void inicializar()    = 0;
59:        virtual void dibujar()        = 0;
60:    protected:
61:        PtBase*             ptBase;
62:        VentanaMapa*        ventanaMapa;
63:        Tableta*            tableta;
64:        Rectangulo          tamano;
65:    };


66:    class MapaSimple : public Mapa {
67:    public:
68:        MapaSimple();
69:        ~MapaSimple();
70:        PlanoPuntos&        darPlanoPuntos();
71:        PlanoSegmentos&     darPlanoSegmentos();
72:        PlanoPoligonos&     darPlanoPoligonos();
73:        virtual void        inicializar();
74:        virtual void        dibujar();
75:    protected:
76:        PlanoPuntos         planoPuntos;
77:        PlanoSegmentos      planoSegmentos;
78:        PlanoPoligonos      planoPoligonos;
79:    };
```

La línea **66** indica la relación de herencia y se puede leer como *" un Mapa Simple es un tipo de Mapa"*. También en la interfaz de la clase **Mapa** se observa que es una clase *abstracta*, por que tiene definidas funciones virtuales puras. En el lenguaje C++ las funciones miembro que están igualadas a cero significan que son *virtuales puras*.

7.2.4 Relación de usa

La relación de *usa*, se utiliza cuando una clase requiere los servicios de otra para ayudarse a completar sus responsabilidades, para ello una clase puede crea un objeto en el bloque de un método o se le pasa como parámetro en un mensaje para ayudarle a realizar sus tareas. Por ejemplo en diagrama de clases de la Figura 5.16, se definió la clase Cursor, que es usada por la clase Ventana Mapa para indicar la posición del lápiz magnético de la tableta en el monitor de la computadora. En las líneas 94 y 95 un objeto de tipo Cursor se pasa como parámetro el cual indica la relación de *usa*. Para el caso cuando se crea un objeto dentro de un bloque de un método, que es otra manera de expresar la relación, ésta no se puede observar en la interfaz de la clase.

```
89:   class Cursor {
90:   public:
91:       Cursor() : posicion(0.0,0.0) { }
92:       Cursor(Coordenadas& c)   : posicion(c) { }
93:       ~Cursor() { }
94:       void asignarPosicion(Coordenadas&);
95:       Coordenadas darPosicion() const;
96:   protected:
97:       Coordenadas posicion;
98:   };

99:   class VentanaMapa : public TFrameWindow {
100:  public:
101:      VentanaMapa(TWindow* parent, char* titulo, Mapa*);
102:      ~VentanaMapa() { }
103:      void mostrarCursor(Cursor&);
104:      void moverCursor(Cursor&, Coordenadas&);
105:      void dibujarPunto(Punto&);
106:      void dibujarSegmento(Segmento&);
107:      void dibujarPoligono(Poligono&);
108:      //...
109:  private:
110:      Mapa*          mapa;
111:      Transforma     transforma;
112:      //...
113:  };
```

7.2.5 Relación de instanciación

Otras de las características más relevantes dentro del modelo orientado a objetos es la relación de instanciación, por medio de ésta es posible definir diferentes clases e instancias de objetos; aprovechando la estructura y comportamiento de una clase *genérica*. Una clase genérica es equivalente a usar el termino de clase *template* o *parametrizada*.

Una clase genérica en palabras comunes se puede ver como un patrón para hacer ropa, que aplicándolo a diferentes tipos de telas se obtienen una gran variedad de prendas de vestir pero todas ellas con la misma forma.

En los diagramas de clases de las Figuras 6.7 y 6.8, se aprovechó esta relación para modelar la representación de los segmentos y polígonos en forma digital; la implantación de los diagramas se muestran a continuación:

```
106:  persistent class Segmento {
107:  public:
108:      Segmento();
109:      ~Segmento() {};
110:      //...
111:  protected:
112:      cset<Punto*>        puntos;
113:      //...
114:  };

115:  persistent class Polígono {
116:  public:
117:      Polígono();
118:      ~Polígono() { }
119:      //...
120:  protected:
121:      cset<Segmento*>  segmentos;
122:  };
```

Como se puede observar en los diagramas de clases un segmento tiene un conjunto de puntos ordenados secuencialmente; un polígono está formado por un conjunto de segmentos. Para crear estos dos tipos de clases se usó la clase genérica *cset* de *POET*. Las líneas 111 y 120 muestran la relación de *instanciación*.

La clase *cset* describe una familia de clases debido a que se encuentra parametrizada. Una clase que se encuentra parametrizada no puede ser instanciada a menos que primero se definan los parámetros formales con algún tipo definido o con una clase e incluso por ella misma. En el diseño es común encontrar clases que son muy semejantes, que ofrecen las mismas funcionalidades pero sólo se diferencian en el tipo de atributos que manipulan; en estos casos es productivo utilizar las clases parametrizadas, sobre todo en estructuras de datos contenedoras como pila, lista, conjuntos, diccionarios, arreglos y otras.

Las clases que están parametrizadas son también *polimórficas*. El concepto de *polimorfismo* es otro de los pilares dentro del modelo orientado objeto y se refiere a la disposición de un objeto para que asuma diferentes tipos o se comporte de diferentes maneras; dentro de sus características están permitir reusar el mismo código y ayudar a realizar modificación del software con pocos cambios. En el caso de los puntos terminales de un segmento conocidos como nodos; debido a que la clase Nodo está heredando de la clase Punto, los objetos nodos son un tipo Punto (por esta razón no están definidos explícitamente en las clase Segmento); el polimorfismo facilita la manipulación de los nodos, es decir, debido a que los lenguajes orientados a objetos manejan el concepto de *enlace dinámico*, lo que significa que en el código no se necesitan estructuras de control para identificar si se trata de objetos puntos o nodos, en tiempo de ejecución se liga automáticamente con los métodos que puede realizar un objeto, sin necesidad de preguntar por su tipo, logrando de esta manera hacer más claro el código.

7.3 Implantación de métodos

Después de haber definido las interfaces de las clases se pasó a implantar sus operaciones. El procedimiento que se utilizó en la implantación de la parte dinámica del sistema, consistió en las siguientes tres actividades:

- La primera actividad fue buscar en los diagramas de objetos y de interacción los mensajes que son enviados por parte de los usuarios (mensajes externos), para este fin los escenarios también se utilizaron.
- La segunda actividad consistió en realizar un recorrido de la secuencia de envío de mensajes, iniciando en los mensajes externos, y se siguió la numeración de los mensajes en los diagramas de objetos y de interacción, al mismo tiempo se paso lo especificado al lenguaje de programación.
- La última actividad consistió en agrupar los métodos codificación por clases en un archivo con extensión cpp.

Como una ilustración se describe cómo se realizó la codificación de los diagramas de las Figuras 5.8 y 5.11. Debido a la representación que se optó en el diseño para los mapas, en los diagramas se sustituyó el objeto mapa por plano Segmentos. En el diagrama de la Figura 5.8, el primer mensaje es añadir Segmento; en las fases del análisis y diseño de la interfaz, se inventó la clase Diálogo, con el motivo de comunicar los objetos del dominio del problema con los usuarios finales. Lo primero que se codificó se muestra de las líneas 122 a 125.

```
122:   void Dialogo::CmAnadir()
123:   {
124:       planoSegmentos->anadirSegmento( );
125:   }
```

Una observación interesante es que el *mensaje* de **añadir Segmento**, es enviado al objeto plano **Segmento** por un objeto de la clase **Diálogo**, es decir, por la instancia de la clase a la cual pertenece el método donde se pide el servicio; por lo tanto en la línea **124** se indica quién envía, quién lo recibe y cuál mensaje es; es por esta razón que el código en los lenguajes orientados a objetos es muy legible, por ejemplo en el lenguaje español la línea **125** se lee "*el objeto diálogo pide el servicio de añadir un segmento al plano de segmentos*".

En el diagrama de *iteración* también se especifica que el mensaje de **añadir Segmento** desencadena tres *mensajes*. El primero es **crear** un segmento declarado en la línea **128**; en el lenguaje C++ al declarar un objeto, implícitamente se invoca la *constructora* de la clase en tiempo de ejecución. El segundo mensaje es **obtener Localización** enviado por el objeto plano **Segmentos** a sí mismo, para obtener las coordenadas del objeto espacial (línea **130**). El tercer mensaje es **almacenar**, el cual se especifica en las líneas **131** y **132**.

```
128:   void PlanoSegmentos::anadirSegmento()
127:   {
128:       Segmento s;
129:
130:       this->obtenerLocalizacion( s );
131:       s.Assign( ptBase );
132:       s.Store( PtFLAT );
133:   }


134:   void PlanoSegmentos::obtenerLocalizacion(Segmento& s)
135:   {
136:       while ( tableta->darEvento() != BOTON_TERMINAR ) {
137:           switch ( tableta->darEvento() ) {
138:               case BOTON_ASIGNAR:
139:                   punto = new Punto( tableta->darCoordenadas() );
140:                   s.anadirPunto( punto );
141:               break;
142:               case BOTON_ELIMINAR:
143:                   s.eliminarPunto();
144:               break;
145:           }
146:       }
147:   }
```

En el diagrama de interacción de la Figura 5.11, cuando se envía el mensaje de **obtener Localización**, se especifica un ciclo que termina hasta que se complete el segmento que está siendo digitalizando (línea **138** a **146**); dentro del ciclo se encuentran los casos de añadir un punto (**139** y **140**) o eliminarlo (**143**).

7.3.1. Manejo de excepciones

Una de las mejoras incorporadas al lenguaje C++ es el manejo de excepciones. Esta característica es de gran utilidad para conducir situaciones anormales presentadas en tiempo de ejecución.

Cuando se envía un mensaje a un objeto para que éste realice satisfactoriamente el método involucrado, antes se deben cumplir ciertas condiciones; en el caso que no se cumplan, ¿cómo se deben realizar los servicios para los solicitantes?. Por ejemplo cuando se envía el mensaje:

```
planoSegmentos->añadirSegmento( );
```

Si se presenta la situación de un disco saturado para almacenar los objetos; el problema se puede manejar de alguna de las siguientes maneras:

1. En el manejo de esta situación, la más simple es que no se realice ninguna acción al invocar el método **añadir Segmento** y el sistema no envíe ningún mensaje de error.
2. Otra es que el sistema despliegue un mensaje de error explicando el problema y luego termine su ejecución.
3. Otra es notificar en una variable de éxito la terminación de la operación; la variable éxito debe ser pasada al método como un argumento extra.

En el caso de las soluciones 1 y 2, Katrib [1994] las describe respectivamente como *retorno silencioso* e *histeria*. En la solución 3, el método invocado debe asignar un valor de **TRUE** a la variable éxito si termina exitosamente, de lo contrario **FALSE**. En este último caso el sistema resulta ser mas robusto que las dos primeras; el inconveniente es que siempre se tiene que definir una variable extra para cada método; haciendo menos legible al código y además se debe estar revisando el estado de la variable cada vez que se envía el mensaje.

El lenguaje C++ proporciona mecanismo explícito para manejar las excepciones, el cual consiste en que cuando un objeto detecta una falla o problema en una función miembro y no pueda resolverlo *eleve* o *dispare* una excepción, con la esperanza de que quien envío el mensaje la capture y trate de solucionar el problema, si no la captura será trasmitida automáticamente al siguiente objeto quien pidió el servicio mayor; el proceso continuará hasta que alguien capture la excepción, si nadie la obtiene el sistema deberá reportar algún error definitivo.

Esta característica del lenguaje se aprovechó para hacer más robusto el prototipo; para ésto algunas clases del modelo del sistema se modificaron con el objetivo de manejar los problemas en tiempo de ejecución. En seguida se describe como se implementa este recuso.

Las excepciones se definen como una clase dentro de la clase que contiene los métodos que pueden elevar una excepción; la clase excepción puede ser vacía, para el caso que no se requiera pasar información adicional sobre el problema presentado; si se desea describir el problema se definen atributos en la clase. En la abstracción de **Segmento** se puede observar la definición de las clases excepciones **Almacenar Segmento** y **Eliminar Segmento** (líneas **160** a **169**).

En el lenguaje C++, no especifica una sintaxis explícita para definir el lugar de los componentes de una clase. La manera que se optó en esta tesis fue: primero los elementos que son accesados por otras abstracciones (**public**), en seguida los privados y por último la definición de las excepciones en otra parte **public**.

```
148:  persistent class Segmento {
149:  public:
150:      Segmento();
151:      ~Segmento() { };
152:      virtual int Store(PtDepthMode Mode = PtDEEP);
153:      void    anadirPunto(Punto*);
154:      //...
155:  protected:
156:      cast<Punto*> puntos;
157:      //...
158:  // Excepciones
159:  public:
160:      class AnadirPunto {
161:      public:
162:          AnadirPunto(int aError)  : error(aError) { }
163:          int error;
164:      };
165:      class AlmacenarSegmento {
166:      public:
167:          AlmacenarSegmento(int aError) : error(aError) { }
168:          int error;
169:      };
170:  };
```

Cuando se detecta algún problema en la ejecución de un método se eleva una excepción por medio de **throw**, como se muestra en el método de Store (línea **178**).

```
173:  int Segmento::Store(PtDepthMode Mode)
174:  {
175:      int error;
176:
177:      if ( (error = PtObject::Store( Mode )) != 0)
178:          throw AlmacenarSegmento( error );
179:
180:      return error;
181:  }
```

La instrucción **throw**, causa que la ejecución de la función miembro **Store** termine y además el constructor de la clase excepción es utilizado explícitamente para crear un objeto anónimo, (es aquel que no está asociado a ningún nombre de una variable). En el ejemplo, el objeto creado en la línea **178** pertenece a la clase definida en **165**.

Esta no es una terminación normal y el hilo de control no se pasa a la siguiente instrucción de donde se envío el mensaje, sino a un *manejador de excepción*; el manejador comienza con la palabra reservada **catch**, seguido por un bloque de acciones. En el ejemplo si se disparó la excepción 178, se pasará el control al manejador de excepciones de la línea 192.

```
182:    void PianoSegmentos::anadirSegmento(Segmento& s)
183:    {
184:        try {
185:            this->obtenerCoordenadas( segmento );
186:            segmento.Assign( pBase );
187:            segmento.Store( PtFLAT );
188:        }
189:        catch(Segmento::AnadirPunto excepcion) {
190:            throw;
191:        }
192:        catch(Segmento::AlmacenarSegmento excepcion) {
193:            throw;
194:        }
195:    }
```

La construcción **catch** debe ir inmediatamente después de un bloque prefijado con la palabra reservada **try**. Para capturar la excepción se especifica el nombre de la clases donde fue definida y el nombre de la clase excepción (**Segmento::AnadirPunto**, **Segmento::Almacenar Segmento**).

7.4 Prueba e integración del sistema

Las pruebas del funcionamiento correcto de software es una de las actividades criticas dentro del proceso de desarrollo; en esta tesis no se abordo a profundidad por ser un tema muy amplio. La manera de realizar las pruebas como la integración del prototipo se aplicó a tres niveles:

- **Unidades.** La aplicación de pruebas e integración a nivel de unidades se hizo sobre las clases del modelo del sistema.
- **Categorías.** Envuelve las pruebas e integración del conjunto de clases que pertenecen a una categoría.
- **Sistema.** Se realizaron pruebas sobre el sistema completo.

Para la aplicación de pruebas a unidades, se usó una combinación de los enfoques de *cajas blancas* y *negras*.

En el enfoque de caja negras, se trata a las clases como piezas de software que operan sin tener el conocimiento de la manera que fueron diseñadas y codificadas. El conjunto de pruebas en este caso se desarrolla con base a los servicios especificados en la *interfaz* de la clase.

Por el contrario, en las pruebas donde las clases se ven como cajas blancas, se usa la información acerca de la estructura interna de las abstracciones y su especificación es ignorada, es decir, las pruebas se hacen con los métodos de las clases en forma separada y efectuando un recorrido en la secuencia de envíos de mensajes entre los objetos, para ello se toma como base los diagramas de objetos e interacción. Una descripción más detallada de estos enfoques se encuentra en Ghezzi et al [1991].

El proceso de integración es de tipo incremental, el cual consiste en probar el correcto funcionamiento de las clases individuales e incorporarlas con otras, y todas estas a su vez en su respectiva categoría.

Como prueba del sistema a gran escala se aplicó un enfoque de cajas negras; para esto se reportaba las anomalías, después se buscaba en cuál categoría estaba el problema del mal funcionamiento y cuál de sus clases era la responsable. Para esta prueba se digitalizó una parte de Ciudad Universitaria, los datos se tomaron del mapa elaborado por el Instituto de Geografía, a escala 1:5000. Algunas de las funcionalidades del prototipo se muestran en la Figura 7.1.

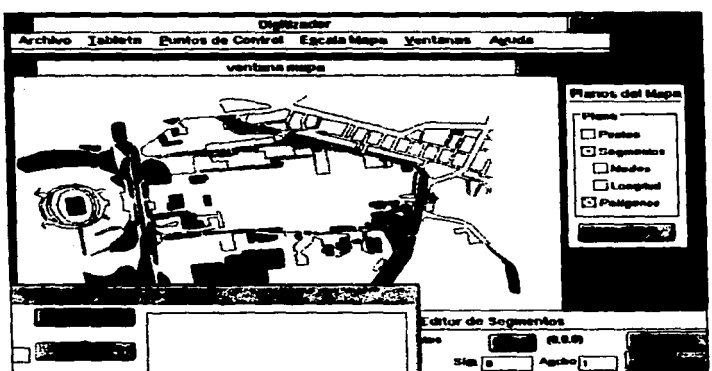


Figura 7.1 El sistema Digitizador.

8 Mantenimiento del sistema

La fase de mantenimiento del sistema consiste en el manejo del sistema después de su liberación. Como un ejercicio práctico, se incluyeron otros requerimientos especificadas en las clases CFT para observar las facilidades que proporciona el paradigma orientado a objetos en el reuso y en la extensión de los productos obtenidos en el análisis, diseño y codificación.

Desde el punto de vista de la representación de los objetos espaciales, se comprobó que las estructuras propuestas pueden ser utilizadas para representar diferentes tipos de mapas temáticos.

8.1 Definición de nuevos requerimientos

A partir del sistema que captura los puntos, segmentos y polígonos se requiere digitizar mapas urbanos mencionados sección 6.2.1 en la parte CFT de los diagramas de clases. La información que se desea capturar a parte de la posiciones de los objetos en el espacio, son cinco capas de información:

1. **Colonias**
   Nombre
2. **Manzanas**
   Número de manzana
   La colonia donde se encuentra la manzana
3. **Lotes**
   Número de lote
   La manzana donde se encuentra el lote
4. **Calles**
   Nombre
   La colonia donde se encuentra la calle
5. **Topografía**
   Alturas de las curvas

8.2 Agregación de otras funcionalidades

El modelo que se creó en las fases análisis y diseño de este trabajo, se aprovechó para extenderlo e incluir otros requerimientos. La manera de representar los diferentes tipos de objetos espaciales se describió en detalle de la sección 6.2.1 a la 6.2.4, la cual consiste en aprovechar los beneficios paradigma orientado a objetos para modelar otros sistemas a partir de componente reusables.

El sistema inicial requirió un tiempo de desarrollo de 5 meses, los beneficios del paradigma orientado a objetos se notaron al agregar las nuevas funcionalidades dado que tan sólo fue de 4 días, concluyéndose que gran parte del código fue reutilizado. El éxito en gran media se debe: a la descomposición modular del sistema, al mecanismo de herencia del modelo orientado a objetos y al principio de *anticipación al cambio*, el cual permite pensar en posibles requerimientos y modelar las abstracciones en forma genérica.

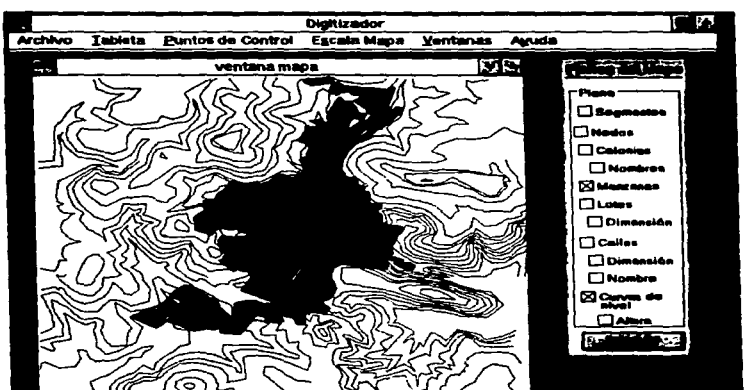


Figura 8.1 El sistema Digitizador, el cual muestra las capas de información de topografía y de manzanas de un mapa urbano.

Desde el punto de vista de la representación digital de la información espacial, se comprobó que la manera de asociar los atributos temáticos a las abstracciones punto, segmentos y polígonos para representar fenómenos geográficos es más natural de acuerdo a como se perciben los mapas. Este modelo de representación también permite implantar estructuras topológicas, que inclusive tengan un manejo explícito de *islas* y consultas de las relaciones topológicas entre los objetos; por ejemplo la estructura topológica propuesta por Van Roessel[1987] se puede representar aprovechado las clases existentes, al derivar una clase **Polígono Topológico** que incluya un conjunto de referencias a polígonos que son *islas*, una clase **Segmento Topológico** con dos atributos para indicar el polígono que se encuentra al lado derecho y izquierdo.

En las Figura 8.1 y 8.2, se muestra el prototipo que incluye los nuevos requerimientos; para esto se digitalizó una parte de la Ciudad de Zacatecas; los datos fueron tomados del mapa urbano de Zacatecas, Zac, hojas 1/4 y 2/4, editado por la INEGI, escala 1:5000, proyección Universal Transversa de Mercator y edición de 1975.

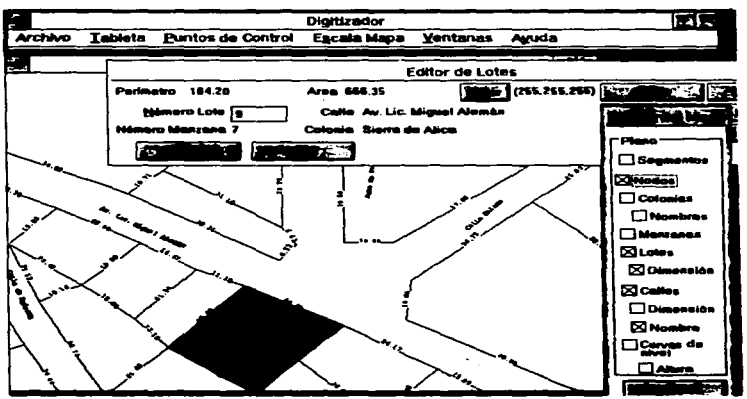


Figura 8.2 El sistema Digitizador, el cual muestra un acercamiento del mapa urbano, donde se observan las capas de información de lotes y calles; se seleccionó un lote para ver sus atributos.

9 Conclusiones

Los sistemas de información geográfica se han convertido en herramientas indispensables para las actividades relacionadas con el manejo y procesamiento de la información espacial; éste tipo de sistemas tiene como propósito ayudar a tomar decisiones, manejar inventarios de recursos, hacer planeaciones y otros.

En esta tesis se utilizó la metodología orientada a objetos para construir un prototipo de un módulo de entrada de datos para un sistema de información geográfica, los resultados obtenidos en el modelado de la información espacial son aceptables y se considera la metodología sin lugar a dudas una excelente alternativa para el desarrollo de sistemas.

9.1 Acerca de la metodología

A lo largo del proceso de desarrollo se encontraron algunas características interesantes sobre la metodología las cuales se describen en seguida:

Una de las ventajas que se le observó es el alto grado de confianza en el funcionamiento correcto del software, debido a que el proceso de desarrollo del ciclo de vida del software es de tipo incremental, donde el crecimiento del modelo se sustenta sobre bases estables que reducen grandemente el riesgo en la construcción de sistemas complejos.

La metodología tiene como objetivo el reuso del software por medio de los conceptos de herencia, polimorfismo y parametrización:

La relación de herencia permite aprovechar el código existente para construir nuevos sistemas; otra característica importante de éste mecanismo es el modelado de los problemas en forma jerárquica, este tipo de clasificación se utiliza ampliamente en varias actividades, de esta manera se facilita el modelado de sistemas complejos.

El polimorfismo es otro concepto importante del modelo orientado a objetos y se refiere a la disposición para que un objeto asuma diferentes tipos o se comporte de diferentes maneras; dentro de sus características están permitir reusar código cuando es de tipo paramétrico y ayudar a modificar el software con pocos cambios.

La parametrización es de gran ayuda para representar estructuras de datos contenedoras, es decir, por medio una clase es posible definir diferentes clases e instancias de objetos, aprovechando la estructura y comportamiento de clases *genéricas* o *templates*.

El modelado orientado a objetos tiene como clave la descomposición de los problemas en abstracciones conocidas como clases, lo que equivale a diseñar un sistema modular; esta es una característica deseable desde el punto de vista de ingeniería de software.

Por otro lado el lenguaje de programación orientado a objeto C++, permitió implantar el modelo del sistema sin dificultades, debido a que existen patrones definidos de código que equivalen a lo especificado en los diagramas obtenidos en el análisis y diseño. Si se asignan pesos en función del trabajo a las fases del proceso de desarrollo, la evolución e implantación es la que tiene un menor peso debido a las facilidades del lenguaje. El lenguaje además incorpora elementos que permiten el conducir situaciones anormales en tiempo de ejecución, de esta manera se construyó un prototipo más robusto al incluir el manejo de excepciones.

Mucho se dice que los cambios de requerimientos o la reestructuración de la arquitectura de un sistema orientado a objetos no afectan; en esta tesis se realizaron diferentes tipos de cambios, donde algunos de ellos mostraron un costo mínimo, mientras que en otros el costo es de consideración. Los cambio encontrados se resumen en los siguientes puntos:

- Adicionar nuevas clases. Se pueden adicionar nuevas clases como resultado de incrementar el modelo que se esta construyendo en el análisis y diseño o con el motivo de mejorar el sistema. En el primer caso el costo no es relevante y es común realizarlo en las primeras iteraciones del proceso micro, es por eso que se pide en las primeras iteraciones no se tenga una especificación formal de las abstracciones. En el segundo caso, si se producen cambios en las interfaces de las clases sobre todo cuando no fueron considerados implica otros costos.
- Cambiar la interfaz de una clase. La modificación de distribución de responsabilidades es común que se realice en los primeros estados del análisis y diseño, estos si son costosos cuando se encuentra el proceso en estados avanzados, debido que un cambio en la interfaz de una abstracción puede afectar a otras que dependen de ella; es por esto que siempre se desea tener un modelo con *alta cohesión* y *bajo acoplamiento*. Otro cambio en la interfaz de una clase es debido a adicionar una operación para ser utilizada por algún cliente, esto no es costoso excepto por la recompilación, este no es un evento costoso en pequeños sistemas, por que hacer el código objeto toma sólo unos minutos; sin embargo para sistemas grandes puede llevar mucho tiempo; en esta tesis cuando se compilaba todo el sistema se necesitaba un tiempo de 25 minutos, esto indica que no es de extrañarse que ésta tarea puede durar más de medio día para un sistema grande. En extremo la recompilación puede ser costosa cuando los cambios nuevos no han sido probados individualmente.
- Modificar la estructura interna de una clase. Las modificaciones en los métodos no son costosos, especialmente cuando se tienen los atributos de la clase encapsulados y una interfaz estable. Por otro lado si la representación de las clases no está encapsulada entonces un cambio en la representación es costosa, por que los clientes pueden tener dependencias.

Algunos puntos en contra del paradigma orientando a objetos y de la metodología se describen a continuación:

La notación utilizada para expresar el modelo del sistema tiene algunos puntos incompletos; por ejemplo la especificación del manejo de excepciones se definió en las ultimas fases del desarrollo, pero un uso correcto de este mecanismo se requiere una estrategia para identificar y especificar los problemas que pueden ocurrir en tiempo de ejecución, desde la etapa inicial del diseño.

En el análisis y diseño se siguió la recomendación de dividir las actividades por diferentes tipos de clases. Debido a que la notación utilizada no contempla esto, se agregaron algunos elementos para hacer una distinción entre las clases; los diagramas de interacción también se modificaron para obtener una especificación más puntual. No es buena práctica cambiar la notación sobre todo cuando se trabaja con estándares.

Un inconveniente en el paradigma orientada a objetos es la implicación respecto al cambio en la forma de pensar, este paso es aparentemente pequeño debido a que se basa en una filosofía de acuerdo a como los humanos observan el mundo real; pero cuando se inicia y se tiene poca experiencia construir correctamente un modelo orientado a objetos requiere de un esfuerzo grande. Uno de los elementos que ayudan a pensar en *objetos* sobre todo cuando se inicia es la elaboración de los *escenarios* relacionados con la funciones punto.

Otro inconveniente es la falta de madurez de algunas metodologías y herramientas de desarrollos, debido a que no existe un enfoque común, por el contrario cada autor tiene su punto de vista sobre el modelo orientado a objetos.

En lo que respecta a la construcción de un sistema por un equipo grande de personas, no se tienen comentarios, porque este sistema fue elaborado por una sola persona; por lo tanto no se determinó si la metodología es adecuada para usarla por un grupo de desarrolladores.

9.2 Representación digital de mapas

En la sección 6.1, se describió que es difícil construir un sistema que permita capturar la información no especial de cualquier tipo de mapa; debido a que no se sabe de cuantas capas de información consta y cuáles son sus tipos. En esta tesis se propuso una manera para modelar diferentes fenómenos geográficos en forma digital.

Lo primero que se propuso fue un modelo de un mapa compuesto por tres planos de información: de puntos, de segmentos y de polígonos debido a que cualquier capa de información especial puede ser agrupada dentro de alguno de estos planos. La desventaja que se encontró fue la dificultad al organizar los datos especiales, debido a que un plano puede contener más de una capa de información del mapa.

Una segunda propuesta para modelar diferentes tipos de mapas, consistió en construir un conjunto de clases reusables que sirvan de base para representar mapas digitales específicos. Para esto se aprovecharon las bondades del paradigma orientado a objetos.

El modelo de representación propuesto se sintetiza en las siguientes ventajas y desventajas:

Ventajas:

- Se modelan los mapas de modo más natural, debido a que esta representación se basa en *objetos* que es una manera común de percibir los mapas.
- Se puede aprovechar el código existente para capturar la información de cualquier mapa temático.
- Maneja los atributos no espaciales y topológicos de los diferentes fenómenos geográficos.
- Es posible representar estructuras topológicas sin modificar las clases existentes, sólo se agregan otras para representar la topología ente los objetos espaciales. Por ejemplo la propuesta por Van Roessel [1987], la cual utiliza el modelo relacional, se puede implantar con el modelo orientado a objetos.

Desventajas:

- Para capturar un mapa que no se encuentra representado por las abstracciones que se tienen definidas, se necesita codificarlas y generar un ejecutable; lo cual puede resultar ser una tarea tediosa.

9.3 Mejoras al prototipo

Todavía falta camino por recorrer para convertir el prototipo en un verdadero sistema. Se propone para futuros requerimientos aumentar su eficiencia, por medio de utilizar algoritmos y estructuras más sofisticadas que permitirán buscar objetos espaciales en las bases de datos con tiempos de respuesta mejores; una de las estructuras que se pretende implantar es la de *árboles de búsqueda binaria multidimensionales*, descrita por Bentley [1975].

No se obtuvo un sistema que permite capturar cualquier tipo de mapa temático, en su lugar se construyó un conjunto de clases reusables que son fácilmente aprovechadas para representar cualquier objeto espacial en forma digital. Para futuros trabajos se pretende hacer más amigable el sistema, para ello se propone construir un generador de código; los usuarios no tendrán que escribir el código para representar las capas información del mapa y ni los atributos espaciales de los objetos. El generador de código es un proyecto viable, debido a que es muy semejante el esquema de código del sistema para digitizador mapas simples y urbanos, y se pueden aplicar los conceptos innovadores del paradigma orientado a objetos de *patrones*. Este generador ayudaría en la fase de construcción de la base de datos para un estudio con un sistema de información geográfica.

Otras operaciones que se agregarán al prototipo son las de exportación de información a otros formatos como: **dxf**, **gif**, **bmp**, **wks** y **dbf**; por medio de estas operaciones la información capturada se podrá procesar en otros sistemas.

Clase Alphabetical List

Cadena
Char
Chotesky
Coordenadas
Cast<>
Cast<Punto>
Cast<Segmento>
Cursor
Cursor de Trazo
Diálogo Configurar Tableta
Diálogo Editor
Diálogo Editor de Polígonos
Diálogo Editor de Puntos
Diálogo Editor de Puntos de Control
Diálogo Editor Segmentos
Diálogo Mapa Simple
float
int
Línea
Lista<Puntos de Control>
long
Mapa
Mapa Simple
Matriz
Nodo
Plano
Plano de Polígonos
Plano de Puntos
Plano de Segmentos
Polígono
PtBase
Puerto Serial
Punto
Punto de Control
Rectángulo
Segmento
Tableta
TArray<>
TCheckBox
TColor
TComboBox
TDialog
TEdit

104
TFrameWindow
TGroupBox
TListBox
Transforma
TStatic
Vector
Ventana del Mapa
Ventana Menú

Modelo del sistema Digitizador

OOwin/CRC Card

Cadena	
Responsibility	**Collaboration**
Manejo de cadenas de caracteres	Char

Responsibility Detail

Manejo de cadenas de caracteres

Operation Detail	Visibility
localizaCadena	Public
~Cadena	Public
darTamaño	Public
copiar	Public
darComoEntero	Public
darComoReal	Public
leerHastaDelimitador	Public
Cadena	Public
contieneSubcadena	Public
operator <<	Public
copiarSecuencia	Public

Modelo del sistema Digitizador

opetator +	Public
operator +=	Public
operator =	Public
operator <	Public
operator >	Public
operator ==	Public
operator *	Public
quitarHastaDelimitador	Public

Property Detail

Property Class	*Label*	*Quantity*	*Relationship*
Char	caracteres	1 or More	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Cholesky	
Responsibility	**Collaboration**
Resolver sistemas de ecuaciones lineales	Matriz
	Vector
	Mapa

Responsibility Detail

Resolver sistemas de ecuaciones lineales

Operation Detail

				Visibility
resolverSistemaEcuaciones				Public
Parameter Class	Label		Quantity	Type
Vector	b		1	UnSpecified
Return Class		Label		Quantity
Vector	x			1
Uses Class		Label		Quantity
Matriz	gT			1

transformarAgpT	Public
asignarMatriz	Public
~Cholesky	Public
Cholesky	Public

Property Detail

Property Class	Label	Quantity	Relationship
Matriz	g	1	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Coordenadas	
Responsibility	**Collaboration**
Posición del los puntos en el plano	int
	float
	Coordenadas

Responsibility Detail

Posición del los puntos en el plano

Operation Detail

operator ==				Visibility: Public
Parameter Class	Label	Quantity	Type	
Coordenadas	c	1	input	
Return Class	Label		Quantity	
int			1	

~Coordenadas	Public

asignarCoordenadas	Public

asignarCoordenadasX	Public

asignarCoordenadasY	Public

Coordenadas	Public

operator +=				Public
Parameter Class	Label	Quantity	Type	
Coordenadas	c	1	input	
Return Class	Label		Quantity	
Coordenadas			1	

darCoordenadasY	Public

Modelo del sistema Digitizador

operator !=	Public

operator +	Public

operator -	Public

operator *	Public
Producto cruzado de coordenadas utilizado para calcular el área de un polígono	

Parameter Class	Label	Quantity	Type
Coordenadas	c	1	Input

Return Class	Label	Quantity
Coordenadas		1

operator <	Public
Regresa las coordenadas más a la izquierada	

Parameter Class	Label	Quantity	Type
Coordenadas	c	1	Input

Return Class	Label	Quantity
Coordenadas		1

operator >	Public
Regresa las coordenadas que están más hacia la derecha	

Parameter Class	Label	Quantity	Type
Coordenadas	c	1	Input

Return Class	Label	Quantity
Coordenadas		1

darCoordenadasX	Public

operator =	Public

Parameter Class	Label	Quantity	Type
Coordenadas	c	1	UnSpecified

Return Class	Label	Quantity
Coordenadas		1

Modelo del sistema Digitizador

Property Detail

Property Class	*Label*	*Quantity*	*Relationship*
float	y	1	Contains
float	x	1	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Cset<>	
Responsibility	
Estructura contenedora con manejo persistente	

Responsibility Detail

Estructura contenedora con manejo persistente

Operation Detail	Visibility
Unlock	Public
Lock	Public
Unget	Public
Seek	Public
Put	Public
Insert	Public
GetNum	Public
Get	Public
Find	Public
Delete	Public
Clear	Public

Assign	Public

Append	Public

Modelo del sistema Digitizador

OOwin/CRC Card

Cursor	
Responsibility	**Collaboration**
Saber la posición del lápiz magnetico de la tableta	Coordenadas

Responsibility Detail

Saber la posición del lápiz magnetico de la tableta

Operation Detail	Visibility
darPosición	Public
asignarPosición	Public
~Cursor	Public
Cursor	Public

Property Detail

Property Class	Label	Quantity	Relationship
Coordenadas	posición	1	Contains

OOwin/CRC Card

Cursor de Trazo	
Responsibility	***Collaboration***
Saber la posición del lápiz magnético de la tableta	Coordenadas

Responsibility Detail

Saber la posición del lápiz magnético de la tableta

Operation Detail	*Visibility*
darPosiciónFinal	Public
darPosiciónInicial	Public
asignarPosiciónFinal	Public
asignarPosiciónInicial	Public
~CursorTrazo	Public
CursorTrazo	Public

Property Detail

Property Class	*Label*	*Quantity*	*Relationship*
Coordenadas	posicionFinal	1	Contains
Coordenadas	posicionInicial	1	Contains

OOwin/CRC Card

Diálogo Configurar Tableta	
Superclass	
TDialog	
Responsibility	**Collaboration**
Obtener y desplegar la configuración de la tableta	TComboBox
	Tableta
	TEdit

Responsibility Detail

Obtener y desplegar la configuración de la tableta

Operation Detail	Visibility
CmAceptar	Public
~DiálogoConfiguraciónTableta	Public
DiálogoConfiguraciónTableta	Public

Event Detail

Petición del usuario

Property Detail

Property Class	Label	Quantity	Relationship
TEdit	eDistanciaEntrePuntos	1	Private
TEdit	eTiempoEnvioPuntos	1	Private
TComboBox	cBParidad	1	Private
TComboBox	cBBitsDato	1	Private
TComboBox	cBBitsParo	1	Private
TComboBox	cBBaudios	1	Private
TComboBox	cBDirecciónPuerto	1	Private
Tableta	tableta	1	Private

20-Apr-97

Modelo del sistema Digitizador

OOwin/CRC Card

Diálogo Editor	
Superclass	
TDialog	
Subclass	
Diálogo Editor de Polígonos	
Diálogo Editor Segmentos	
Diálogo Editor de Puntos	
Responsibility	**Collaboration**
Obtener la información de los elementos a digitizar	Ventana del Mapa

Responsibility Detail

Obtener la información de los elementos a digitizar

Operation Detail	Visibility
CmInicializaCampos = 0	Public
CmEliminar = 0	Public
CmSeleccionar = 0	Public
CmAlmacenar = 0	Public
CmAñadir = 0	Public
SetupWindow	Public
~DiálogoEditor	Public
DiálogEditor	Public

Modelo del sistema Digitizador

OOwin/CRC Card

Diálogo Editor de Polígonos	
Superclass	
Diálogo Editor	
Responsibility	**Collaboration**
Obtener y desplegar la información de los polígonos	TBitstic
	TColor
	Polígono
	Plano de Polígonos

Responsibility Detail

Obtener y desplegar la información de los polígonos

Operation Detail	Visibility
CmObtenerColor	Public
desplegarDatos	Public
leerDatos	Public
asignarPlano	Public
~DiálogoEditorPolígonos	Public
DiálogoEditorPolígonos	Public

Event Detail

Petición del usuario

Property Detail

Property Class	Label	Quantity	Relationship

20-Apr-97

Modelo del sistema Digitizador

TStatic	pColor	1	Protected
TStatic	pArea	1	Protected
TStatic	pPerimetro	1	Protected
TColor	sColor	1	Protected
Poligono	poligonoSeleccionado	1	Private
Plano de Poligonos	planoPoligonos	1	Private

OOwin/CRC Card

Diálogo Editor de Puntos	
Superclass	
Diálogo Editor	
Responsibility	**Collaboration**
Obtener y desplegar la información de los puntos	TEdit
	Punto
	Plano de Puntos

Responsibility Detail

Obtener y desplegar la información de los puntos

Operation Detail	Visibility
desplegarDatos	Public
leerDatos	Public
asignarPlanoPuntos	Public
~DiálogoEditorPuntos	Public
DiálogoEditorPuntos	Public

Event Detail

Petición del usuario

Property Detail

Property Class	Label	Quantity	Relationship
TEdit	eCoordenadasY	1	Protected
TEdit	eCoordenadasX	1	Protected
Punto	puntoSeleccionado	1	Private

Modelo del sistema Digitizador

Plano de Puntos	planoPuntos	1	Private

Modelo del sistema Digitizador

OOwin/CRC Card

Diálogo Editor de Puntos de Control	
Superclass	
TDialog	
Responsibility	**Collaboration**
Obtener y desplegar los puntos de control	TListBox
	TEdit
	TStatic
	TableIa

Responsibility Detail

Obtener y desplegar los puntos de control

Operation Detail	Visibility
darListaPuntosControl	Public
elimiarPuntoControl	Public
insertarPuntoControl	Public
EventoTiempo	Public
~DiálogoEditorPuntosControl	Public
DiálogoEditorPuntosControl	Public

Event Detail

Petición del usuario

Property Detail

Property Class	Label	Quantity	Relationship

20-Apr-97

Modelo del sistema *Digitizador*

TListBox	listaPuntosControl	1	Private
TEdit	aCoordenadaRealY	1	Private
TEdit	aCoordenadaRealx	1	Private
TStatic	aCoordenadaTabletaY	1	Private
TStatic	aCoordenadaTabletaX	1	Private
TStatic	aNúmeroPuntos	1	Private
Tableta	tableta	1	Private

Modelo del sistema Digitizador

OOwin/CRC Card

Diálogo Editor Segmentos	
Superclass	
Diálogo Editor	
Responsibility	**Collaboration**
Obtener y desplegar la información de los segment	TStatic
	TEdit
	TColor
	Segmento
	Plano de Segmentos

Responsibility Detail

Obtener y desplegar la información de los segmentos

Operation Detail	Visibility
CmObtenerColor	Public
desplegarDatos	Public
leerDatos	Public
asignarPlano	Public
~DiálogoEditorSegmentos	Public
DiálogoEditorSegmentos	Public

Event Detail

Petición del usuario

Property Detail

Property Class	Label	Quantity	Relationship

20-Apr-97

Modelo del sistema Digitizador

Property Class	Label	Quantity	Relationship
TStatic	eColor	1	Protected
TStatic	eNumeroPuntos	1	Protected
TStatic	eLongitud	1	Protected
TEdit	eAnchoLinea	1	Protected
TEdit	eSimboloLinea	1	Protected
TColor	tColor	1	Protected
Segmento	segmentoSeleccionado	1	Private
Plano de Segmentos	planoSegmentos	1	Private

OOwin/CRC Card

Diálogo Mapa Simple	
Superclass	
TDialog	
Responsibility	**Collaboration**
Decir cuáles son las capas de información a dibujar	TCheckBox
	TGroupBox
	Mapa Simple
	Plano de Polígonos

Responsibility Detail

Decir cuáles son las capas de información a dibujar

Operation Detail	Visibility
seDibujaNodos	Public
seDibujaPlanoPoligonos	Public
seDibujaPlanoSegmentos	Public
seDibujaPlanoPuntos	Public
~DigMapaSimple	Public
DigMapaSimple	Public

Property Detail

Property Class	Label	Quantity	Relationship
TCheckBox	cbPlanoNodos	1	Contains
TCheckBox	cbPlanoPoligonos	1	Contains
TCheckBox	cbPlanoSegmentos	1	Contains
TCheckBox	cbPlanoPuntos	1	Contains
TGroupBox	gbPlanos	1	Contains
Mapa Simple	mapaSimple	1	Contains

20-Apr-97

Modelo del sistema Digitizador

OOwin/CRC Card

Mapa	
Subclass	
Mapa Simple	
Responsibility	***Collaboration***
Representar fenómenos geográficos en forma digital	Rectángulo
	Ventana del Mapa
	PtBase

Responsibility Detail

Representar fenómenos geográficos en forma digital

Operation Detail	*Visibility*
dibujar = 0	Public
darTamañoMaximo	Public
darTamaño	Public
obtenerRegión	Public
desasignarPtBase	Public
asignarTamañoMáximo	Public
asignarTamaño	Public
asignarPtBase	Public
asignarVentana	Public
asignarTableta	Public

Modelo del sistema Digitizador

-Mapa	Public

Mapa	Public

Property Detail

Property Class	*Label*	*Quantity*	*Relationship*
Rectángulo	tamañoMáximo	1	Contains
Rectángulo	tamaño	1	Contains
Ventana del Mapa	ventanaMapa	1	Contains
PtBase	ptBase	1	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Mapa Simple	
Superclass	
Mapa	
Responsibility	**Collaboration**
Representar en forma digital un mapa	Plano de Polígonos
	Plano de Segmentos
	Plano de Puntos

Responsibility Detail

Representar en forma digital un mapa

Operation Detail	Visibility
dibujar	Public
darPlanoPolígonos	Public
darPlanoSegmentos	Public
darPlanoPuntos	Public
~MapaSimple	Public
MapaSimple	Public

Property Detail

Property Class	Label	Quantity	Relationship
Plano de Polígonos	planoPolígonos	1	Contains
Plano de Segmentos	planoSegmentos	1	Contains
Plano de Puntos	planoPuntos	1	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Matriz	
Responsibility	**Collaboration**
Manejo de matrices algebráicas	int

Responsibility Detail

Manejo de matrices algebráicas

Operation Detail	Visibility
intercambiaFilas	Public
esSimétrica	Public
darNúmeroColumnas	Public
darNúmeroFilas	Public
operator *	Public
operator -	Public
operator +	Public
operator ==	Public
operator =	Public
~Matriz	Public
Matriz	Public

Modelo del sistema Digitizador

Property Detail

Property Class	*Label*	*Quantity*	*Relationship*
int	columnas	1	Contains
int	filas	1	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Nodo	
Superclass	
Punto	
Responsibility	**Collaboration**
Manejar la topología entre los segmentos	int

Responsibility Detail

Manejar la topología entre los segmentos

Operation Detail	Visibility
darAsociación	Public
asignarAsociación	Public
Delete	Public
Store	Public
~Nodo	Public
Nodo	Public

Property Detail

Property Class	Label	Quantity	Relationship
int	asociación	1	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Plano	
Subclases	
Plano de Puntos	
Responsibility	**Collaboration**
Representar capas de información	PtBase Tableta

Responsibility Detail

Representar capas de información

Operation Detail	Visibility
dibujar = 0	Public
asignarVentanaMapa	Public
asignarTableta	Public
asignarPtBase	Public
~Plano	Public
Plano	Public

Property Detail

Property Class	Label	Quantity	Relationship
Ventana del Mapa	ventanaMapa	1	Contains
PtBase	ptBase	1	Contains
Tableta	tableta	1	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Plano de Polígonos	
Responsibility	**Collaboration**
Representa capas de información del mapa	Punto

Responsibility Detail

Representa capas de información del mapa

Operation Detail	Visibility
dibujar	Public
eliminarPolígono	Public
seleccionarPolígono	Public
almacenarPolígono	Public
añadirPolígono	Public
obtenerLocalización	Public
~PlanoPolígonos	Public
PlanoPolígonos	Public

Modelo del sistema Digitizador

OOwin/CRC Card

Plano de Puntos	
Superclase	
Plano	
Responsibility	**Collaboration**
Representar capas de información de un mapa	Ventana del Mapa

Responsibility Detail

Representar capas de información de un mapa

Operation Detail	Visibility
dibujar	Public
eliminarPunto	Public
seleccionarPunto	Public
almacenarPunto	Public
añadirPunto	Public
obtenerLocalizaciónPunto	Public
~PlanoPuntos	Public
PlanoPuntos	Public

Modelo del sistema Digitizador

OOwin/CRC Card

Plano de Segmentos	
Responsibility	**Collaboration**
Representar capas de información de un mapa	Segmento
	Tableta
	Ventana del Mapa

Responsibility Detail

Representar capas de información de un mapa

Operation Detail	Visibility
dibujar	Public
desplegarLongitudes	Public
dibujarNodos	Public
darNodo	Public
eliminarSegmento	Public
seleccionarSegmento	Public
almacenarSegmento	Public
añadirSegmento	Public

Parameter Class	Label	Quantity	Type
Segmento	segmento	1	Input
Event Triggered			
Petición del usuario			

20-Apr-97

Modelo del sistema Digitizador

obtenerLocalización			Public
Obtiene la localización de un segmento			
Parameter Class	*Label*	*Quantity*	*Type*
Segmento		1	Input

Uses Class	*Label*	*Quantity*
Tableta	tableta	1
Ventana del Mapa	ventanaMapa	1
Event Triggered		
Añadir un segmento		

~PlanoSegmentos()	Public

PlanoSegmentos()	Public

Event Detail

Petición del usuario
Operation That Triggers
añadirSegmento

Añadir un segmento
Operation That Triggers
obtenerLocalización

OOwin/CRC Card

Polígono	
Responsibility	**Collaboration**
Manejo de persistencia	float
Saber relaciones espaciales y topológicas	long
Representar objetos geográficos	Cset<Segmento>

Responsibility Detail

Saber relaciones espaciales y topológicas

Operation Detail	Visibility
calcularArea	Public
calcularPerimetro	Public
darNúmeroPuntos	Public
darNúmeroSegmento()	Public
darArea	Public
darPerimetro	Public
estaDentroPunto	Public

Property Detail

Property Class	Label	Quantity	Relationship
float	fPerimetro	1	Contains
float	fArea	1	Contains

Modelo del sistema Digitizador

Responsibility Detail

Representar objetos geográficos

Operation Detail	Visibility
eliminarSegmento	Public
añadirSegmento	Public
asignarColor	Public
~Poligono	Public
Poligono	Public

Property Detail

Property Class	Label	Quantity	Relationship
long	iColor	1	Contains
Cast<Segmento>	segmentos	1	Contains

Responsibility Detail

Manejo de persistencia	
Operation Detail	***Visibility***
Delete	Public
Store	Public

Modelo del sistema Digitizador

OOwin/CRC Card

Puerto Serial	
Responsibility	**Collaboration**
Comunicar la tableta con el sistema	jni

Responsibility Detail

Comunicar la tableta con el sistema

Operation Detail	Visibility
darBitsParo	Public
darParidad	Public
darBitsDatos	Public
darPuerto	Public
asignarPuerto	Public
estaListo	Public
obtenerDato	Public
escribirDato	Public
abrir	Public
asignarBitsParo	Public
asignarParidad	Public

Modelo del sistema Digitizador

asignarBitsDatos	Public
asignarBaudios	Public
~PuertoSerial	Public
PuertoSerial	Public

Property Detail

Property Class	*Label*	*Quantity*	*Relationship*
int	bitsParo	1	Contains
int	paridad	1	Contains
int	bitsDatos	1	Contains
int	baudios	1	Contains
int	puerto	1	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Punto	
Subclass	
Nodo	
Responsibility	**Collaboration**
Manejo de persistencia	Coordenadas
Saber relaciones especiales	
Representar objetos geográficos	

Responsibility Detail

Manejo de persistencia

Responsibility Detail

Saber relaciones especiales

Operation Detail	*Visibility*
darDistancia	Public
darDistancia2	Public
producto	Public
transladar	Public
operator >	Public
operador <	Public

Responsibility Detail

Representar objetos geográficos
Representar todos los objetos geográficos que son localizados por medio de un par de coordenadas(x,y)

Operation Detail	*Visibility*
darCoordenadas	Public
asignarCoordenadas	Public
operator =	Public
operator !=	Public
operator ==	Public
~Punto	Public
Punto	Public

Property Detail

Property Class	*Label*	*Quantity*	*Relationship*
Coordenadas		1	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Punto de Control	
Responsibility	**Collaboration**
Saber las coordenadas de un punto en dos sistemas	Coordenadas

Responsibility Detail

Saber las coordenadas de un punto en dos sistemas

Mantiene las coordenadas de puntos en dos sistema, estos puntos se utilizan para crear un sistema de ecuaciones para determinar los vectores de transformación en los dos sistemas

Operation Detail	Visibility
darCoordenadas2	Public
darCoordenadas1	Public
operator ==	Public
operator =	Public
asignarCoordenadas2	Public
asignarCoordenadas1	Public
~PuntoControl	Public
PuntoControl	Public

Property Detail

Property Class	Label	Quantity	Relationship
Coordenadas	c2	1	Contains
Coordenadas	c1	1	Contains

20-Apr-97

Modelo del sistema Digitizador

OOwin/CRC Card

Rectángulo	
Responsibility	**Collaboration**
Manejo del tamaño del mapas	Punto

Responsibility Detail

Manejo del tamaño del mapas

Operation Detail	Visibility
darArea	Public
darPerímetro	Public
darAncho	Public
darAlto	Public
darPuntoSuperiorDerecho	Public
darPuntoInferiorIzquierdo	Public
darPuntoInferiorDerecho	Public
darPuntoSuperiorIzquierdo	Public
estaDentroPunto	Public
asignarMáximoRectángulo	Public
ordenarPuntos	Private

asignarPuntoInferiorDerecho	Public
asignarPuntoSuperiorIzquierdo	Public
~Rectángulo	Public
Rectángulo	Public

Property Detail

Property Class	*Label*	*Quantity*	*Relationship*
Punto	puntoInferiorDerecho	1	Contains
Punto	puntoSuperiorIzquierdo	1	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Segmento	
Responsibility	**Collaboration**
Manejo de persistencia	iorg
Saber relaciones espaciales y topológicas	ini
Representar objetos geográficos	Cpet<Punto>

Responsibility Detail

Manejo de persistencia

Operation Detail	Visibility
Delete	Public
Store	Public

Modelo del sistema Digitizador

Responsibility Detail

Representar objetos geográficos
Representar los objetos dentro de un mapa que puede ser indicados por su longitud

Operation Detail	***Visibility***
darColor	Public
darAnchoLínea	Public
darSímbolo	Public
darPunto	Public
eliminarPunto	Public
añadirPunto	Public
asignarColor	Public
asignarAnchoLínea	Public
asignarSímbolo	Public
Delete	Public
Store	Public
~Segmento	Public
Segmento	Public

Modelo del sistema Digitizador

Property Detail

Property Class	*Label*	*Quantity*	*Relationship*
long	iColor	1	Contains
int	iAnchoLinea	1	Contains
int	iSimbolo	1	Contains
Cast<Punto>	puntos	1	Contains

Responsibility Detail

Saber relaciones especiales y topológicas

Operation Detail	***Visibility***
darNumeroPuntos	Public
calcularProducto	Public
calcularLongitud	Public
darLongitud	Public
insertarPunto	Public

Modelo del sistema Digitizador

OOwin/CRC Card

Tableta	
Responsibility	**Collaboration**
Dar coordenadas del lápiz magnético	Lista<Puntos de Control>
	Puerto Serial
	Cgordenadas
	Host

Responsibility Detail

Dar coordenadas del lápiz magnético

Operation Detail	Visibility
darErrorConfiguración	Public
darPuntosControl	Public
darPuertoSerial	Public
darDistanciaEntrePuntos	Public
darTiempoEnvioPuntos	Public
darCoordenadas	Public
obtenerSeñal	Public
configurarSistemasCoordenadas	Public
asignarPuertoSerial	Public
asignarDistanciaEntrePuntos	Public

Modelo del sistema Digitizador

asignarTiempoEnvioPuntos	Public
~Tableta	Public
Tableta	Public

Property Detail

Property Class	*Label*	*Quantity*	*Relationship*
Lista<Puntos de Control>	listaPuntosControl	1	Contains
Puerto Serial	puertoSerial	1	Contains
Coordenadas	posiciónLápizMagnético	1	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Transforma	
Responsibility	**Collaboration**
Transformar sistemas de coordenadas a otro	Cholesky
	Matriz
	Lista<Puntos de Control>
	Vector

Responsibility Detail

Transformar sistemas de coordenadas a otro

Trasforma las coordenadas del lápiz magnético de la tableta al del mapa que se esta digitalizado.
Tranforma las coordenadas del mapa a la ventana donde se despliega.

Operation Detail	Visibility
darCoordenadas	Public

darErrorConfiguración	Public

calcularVectores			Public
Parameter Class	**Label**	**Quantity**	**Type**
Lista<Puntos de Control>	lPuntosControl	1	Input
Uses Class	**Label**		**Quantity**
Cholesky	cholesky		1
Vector	b2		1
Vector	b1		1
Matriz	a		1
Event Triggered			
Configurar tableta			
Cambio de escala del mapa			

~Transforma	Public

Transforma	Public

Event Detail

Modelo del sistema Digitizador

OOwin/CRC Card

Vector	
Responsibility	**Collaboration**
Manejo de vectores	float Matriz

Responsibility Detail

Manejo de vectores

Operation Detail	Visibility
darDimensión	Public
intercambiaElelentos	Public
operator ()	Public
operator *	Public
operator -	Public
operator +	Public
operator =	Public
~Vector	Public
Vector	Public

Property Detail

Property Class	Label	Quantity	Relationship
float	elementos	1 or More	Contains

Modelo del sistema Digitizador

Configurar tableta
Operation That Triggers
calcularVectores

Cambio de escala del mapa
Operation That Triggers
calcularVectores

Property Detail

Property Class	*Label*	*Quantity*	*Relationship*
Vector	vTransformaciónY	1	Contains
Vector	vTransformaciónX	1	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Ventana del Mapa	
Superclass	
TFrameWindow	
Responsibility	**Collaboration**
Desplegar el mapa que se está digitizando	Transforma
	Mapa

Responsibility Detail

Desplegar el mapa que se está digitizando

Operation Detail	Visibility
dibujarPunto	Public
~VentanaMapa	Public
EventoSize	Public
EventoPaint	Public
cambiarEscala	Public
dibujarCursor	Public
dibujarCursorTrazo	Public
VentanaMapa	Public
moverCursorTrazo	Public
desplegarTexto	Public

Modelo del sistema Digitizador

dibujarPuntoSeleccionado	Public
borrarPunto	Public
dibujarSegmento	Public
dibujarSegmentoSeleccionado	Public
borrarSegmento	Public
dibujarPolígono	Public
dibujarPolígonoSeleccionado	Public
borrarPolígono	Public
moverCursor	Public

Property Detail

Property Class	*Label*	*Quantity*	*Relationship*
Transforma	transforma	1	Contains
Mapa	mapa	1	Contains

Modelo del sistema Digitizador

OOwin/CRC Card

Ventana Menú	
Superclass	
TFrameWindow	
Responsibility	**Collaboration**
Presentar los servicios del sistema al usuario	Ventana del Mapa
	Puerto Serial
	Tableta
	Mapa

Responsibility Detail

Presentar los servicios del sistema al usuario

Operation Detail	Visibility
CmCanClose	Public
CmAcerca	Public
CmAlejamientoSinReferenciaMapa	Public
CmAcercamientoSinReferenciaMapa	Public
CmAcercamientoReferenciaMapa	Public
CmDesplegarErrorConfiguraciónTransformaciónCoordenadas	Public
CmEditarPuntosControl	Public
CmConfigurarTableta	Public
CmArchivo	Public
inicializar	Public

Modelo del sistema Digitizador

~MenúVentana	Public

MenúVentana	Public

Property Detail

Property Class	*Label*	*Quantity*	*Relationship*
Ventana del Mapa	ventanaMapa	1	Protected
Puerto Serial	puertoSerial	1	Protected
Tableta	tableta	1	Protected
Mapa	mapa	1	Private

Class Alphabetical List

Calle
Colonia
Curva de Nivel
Diálogo Editor de Calles
Diálogo Editor de Colonias
Diálogo Editor de Curvas de Nivel
Diálogo Editor de Lotes
Diálogo Editor de Manzanas
Diálogo Mapa Urbano
Lote
Manzana
Mapa Urbano
Plano de Calles
Plano de Colonias
Plano de Curvas de Nivel
Plano de Lotes
Plano de Manzanas

OOwin/CRC Card

Calle	
Superclass	
Segmento	
Responsibility	**Collaboration**
Representar vías de circulación	Cadena

Responsibility Detail

Representar vías de circulación

Operation Detail	Visibility
darNombre	Public
asignarNombre	Public
~Calle	Public
Calle	Public

Property Detail

Property Class	Label	Quantity	Relationship
Cadena	nombre	1	Contains

OOwin/CRC Card

Colonia	
Superclases	
Polígono	
Responsibility	**Collaboration**
Representar conjuntos de manzanas urbanas	Cadena
	int

Responsibility Detail

Representar conjuntos de manzanas urbanas

Operation Detail	Visibility
~Colonia	Public
Colonia	Public
darNombre	Public
darNúmero	Public
asignarNombre	Public
asignarNúmero	Public

Property Detail

Property Class	Label	Quantity	Relationship
Cadena	nombre	1	Contains
int	número	1	Contains

OOwin/CRC Card

Curva de Nivel	
Superclass	
Segmento	
Responsibility	**Collaboration**
Representar curvas con igual elevación con respec	float

Responsibility Detail

Representar curvas con igual elevación con respecto nivel del mar

Operation Detail	Visibility
darAltura	Public
asignarAltura	Public
~CurvaNivel	Public
CurvaNivel	Public

Property Detail

Property Class	Label	Quantity	Relationship
float	altura	1	Contains

OOwinVCRC Card

Lote	
Superclass	
Polígono	
Responsibility	**Collaboration**
Representar predios urbanos	Calle
	int

Responsibility Detail

Representar predios urbanos

Operation Detail	Visibility
darManzana	Public
darCalle	Public
darNúmero	Public
asignarManzana	Public
asignarCalle	Public
asignarNúmero	Public
~Lote	Public
Lote	Public

Property Detail

Property Class	Label	Quantity	Relationship
Manzana	manzana	1	Contains
Calle	calle	1	Contains
int	número	1	Contains

20-Apr-97

Modelo del sistema Digitizador de Mapas Urbanos

OOwin/CRC Card

Manzana	
Superclass	
Polígono	
Responsibility	**Collaboration**
Representar un conjunto de predios urbanos unido	Colonia
	int
	Calle

Responsibility Detail

Representar un conjunto de predios urbanos unidos

Operation Detail	Visibility
derColonia	Public
derNúmero	Public
asignarColonia	Public
asignarNúmero	Public
~Manzana	Public
Manzana	Public

Property Detail

Property Class	Label	Quantity	Relationship
Colonia	colonia	1	Contains
int	número	1	Contains

Modelo del sistema Digitizador de Mapas Urbanos

OOwin/CRC Card

Mapa Urbano	
Responsibility	**Collaboration**
Representar una región urbana en forma digital	Diálogo Mapa Urbano
	Plano de Colonias
	Plano de Manzanas
	Plano de Lotes
	Plano de Calles
	Plano de Curvas de Nivel
	Manzana

Responsibility Detail

Representar una región urbana en forma digital

Operation Detail	Visibility
dibujar	Public
darPlanoColonias	Public
darPlanoManzanas	Public
darPlanoLotes	Public
darPlanoCalles	Public
darPlanoCurvasNivel	Public
~MapaUrbano	Public
MapaUrbano	Public

Property Detail

Property Class	Label	Quantity	Relationship
Diálogo Mapa Urbano	dlgMapaUrbano	1	Contains

20-Apr-97

Modelo del sistema Digitizador de Mapas Urbanos

Plano de Colonias	planoColonias	1	Contains
Plano de Manzanas	planoManzanas	1	Contains
Plano de Lotes	planoLotes	1	Contains
Plano de Calles	planoCalles	1	Contains
Plano de Curvas de Nivel	planoCurvasNivel	1	Contains

20-Apr-97

OOwin/CRC Card

Plano de Calles	
Superclase	
Plano de Segmentos	
Responsibility	**Collaboration**
Representar la capa de información de calles de u	Calle Manzana

Responsibility Detail

Representar la capa de información de calles de un mapa urbano

Operation Detail	Visibility
dibujar	Public

seleccionarSegmento		Public
Return Class	**Label**	**Quantity**
Calle	calle	1
Uses Class	**Label**	**Quantity**
Calle	calle	1

~PlanoCalles	Public

PlanoCalles	Public

OOwin/CRC Card

Plano de Colonias	
Superclass	
Plano de Poligonos	
Responsibility	**Collaboration**
Representar la capa de información de colonias de un mapa urbano	Colonia

Responsibility Detail

Representar la capa de información de colonias de un mapa urbano

Operation Detail	Visibility
dibujar	Public

seleccionarPolígono			Public
Parameter Class	**Label**	**Quantity**	**Type**
Colonia	colonia	1	Unspecified
Return Class	**Label**		**Quantity**
Colonia	colonia		1

~PlanoColonias	Public

PlanoColonias	Public

OOwin/CRC Card

Plano de Curvas de Nivel	
Superclass	
Plano de Segmentos	
Responsibility	**Collaboration**
Representar la capa de información de la topografí	Curva de Nivel
	Lote

Responsibility Detail

Representar la capa de información de la topografía

Operation Detail	Visibility
dibujar	Public

seleccionarSegmento			Public
Parameter Class	**Label**	**Quantity**	**Type**
Curva de Nivel	curvaNivel	1	UnSpecified
Return Class	**Label**		**Quantity**
Curva de Nivel	curvaNivel		1

~PlanoCurvasNivel	Public

PlanoCurvasNivel	Public

Modelo del sistema Digitizador de Mapas Urbanos

OOwin/CRC Card

Plano de Manzanas	
Superclase	
Plano de Poligonos	
Responsibility	**Collaboration**
Representar la capa de información de manzanas	Manzana
	Curva de Nivel

Responsibility Detail

Representar la capa de información de manzanas de un mapa urbano

Operation Detail	Visibility
dibujar	Public

seleccionarPoligono			Public
Parameter Class	**Label**	**Quantity**	**Type**
Manzana	manzana	1	UnSpecified
Return Class	**Label**		**Quantity**
Manzana	manzana		1

~PlanoManzanas	Public

PlanoManzanas	Public

OOwin/CRC Card

Plano de Lotes	
Superclass	
Plano de Poligonos	
Responsibility	**Collaboration**
Representar la capa de información de predios de	Lote
	Colonia

Responsibility Detail

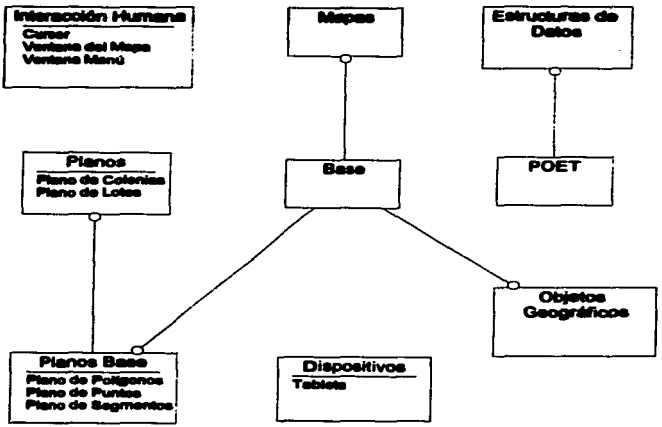
Representar la capa de información de predios de un mapa urbano

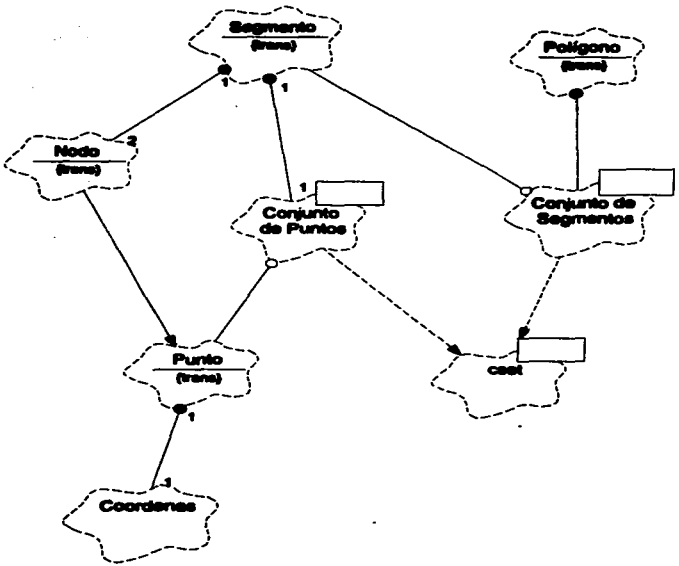
Operation Detail	Visibility
dibujar	Public

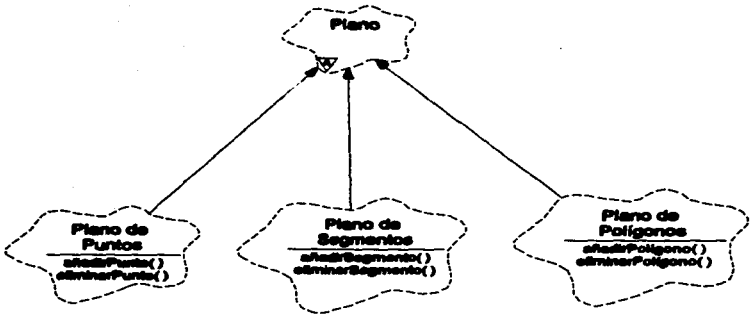
seleccionarPoligono				Public
Parameter Class	**Label**		**Quantity**	**Type**
Lote	lote		1	Unspecified
Return Class	**Label**			**Quantity**
Lote	lote			1

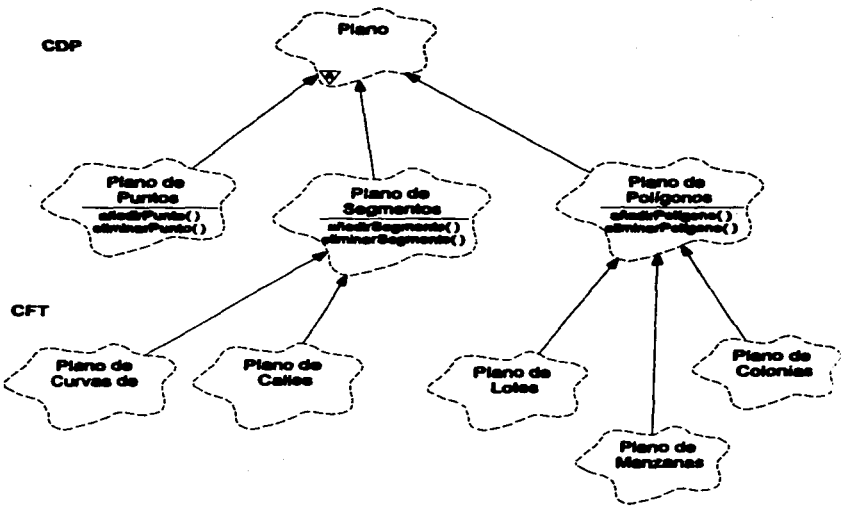
~PlanoLotes	Public

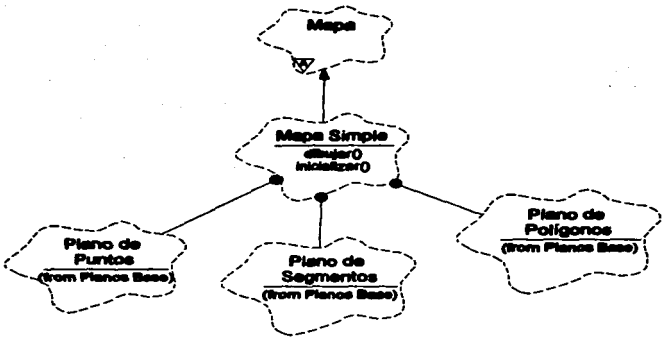
PlanoLotes	Public

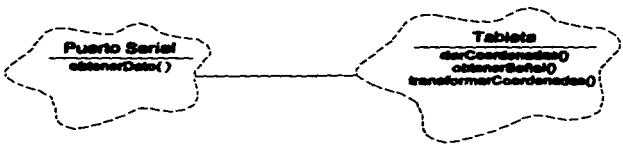

Interacción Humana
Cursor
Ventana del Mapa
Ventana Menú
Mapas
Estructuras de Datos
Planos
Plano de Colonias
Plano de Lotes
Base
POET
Objetos Geográficos
Planos Base
Plano de Polígonos
Plano de Puntos
Plano de Segmentos
Dispositivos
Tablets

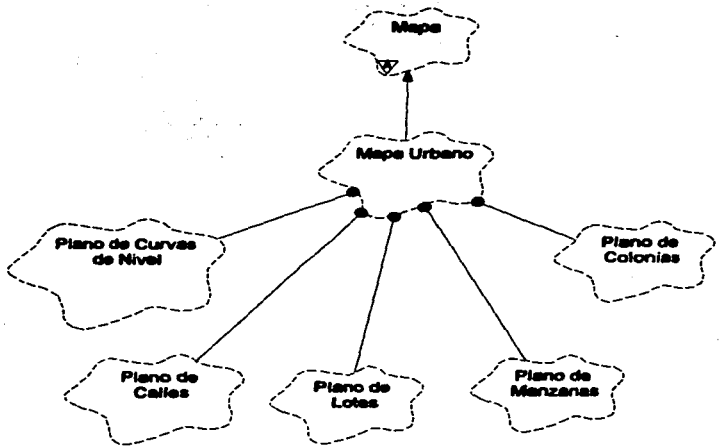

Segmento
(trans)
Polígono
(trans)
Nodo
(trans)
Conjunto de Puntos
Conjunto de Segmentos
Punto
(trans)
cast
Coordenas
1
2

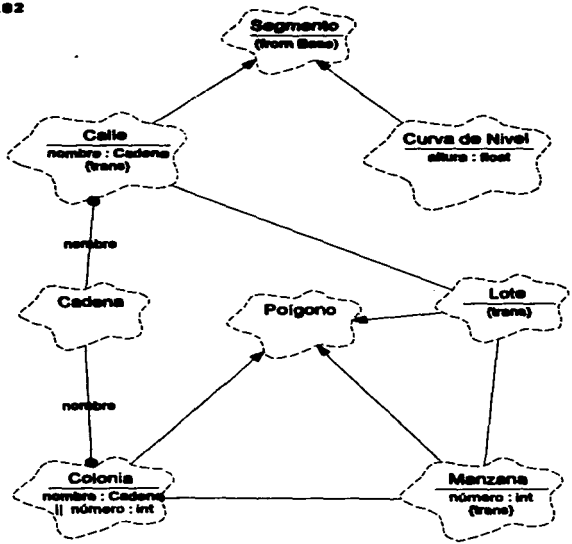

Plano
Plano de Puntos
añadirPunto()
eliminarPunto()
Plano de Segmentos
añadirSegmento()
eliminarSegmento()
Plano de Polígonos
añadirPolígono()
eliminarPolígono()

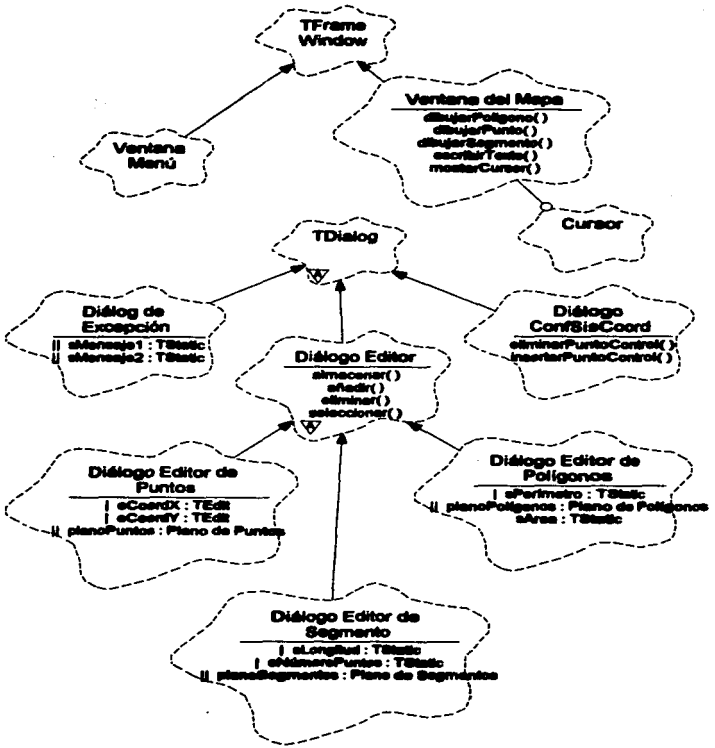

CDP
Plano
Plano de Puntos
eñadirPunto()
eliminarPunto()
Plano de Segmentos
eñadirSegmento()
eliminarSegmento()
Plano de Polígonos
eñadirPolígono()
eliminarPolígono()
CFT
Plano de Curvas de
Plano de Calles
Plano de Lotes
Plano de Manzanas
Plano de Colonias

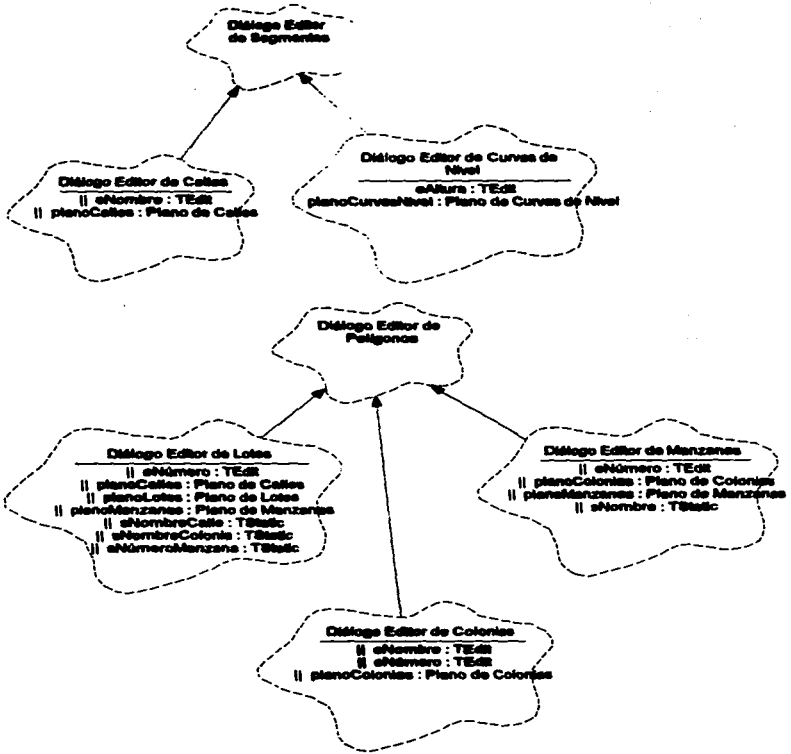

Mapa
Mapa Simple
dibujar()
inicializar()
Plano de Puntos
(from Planos Base)
Plano de Segmentos
(from Planos Base)
Plano de Poligonos
(from Planos Base)


Puerto Serial
obtenerData()
Tableta
darCoordenadas()
obtenerSeñal()
transformarCoordenadas()

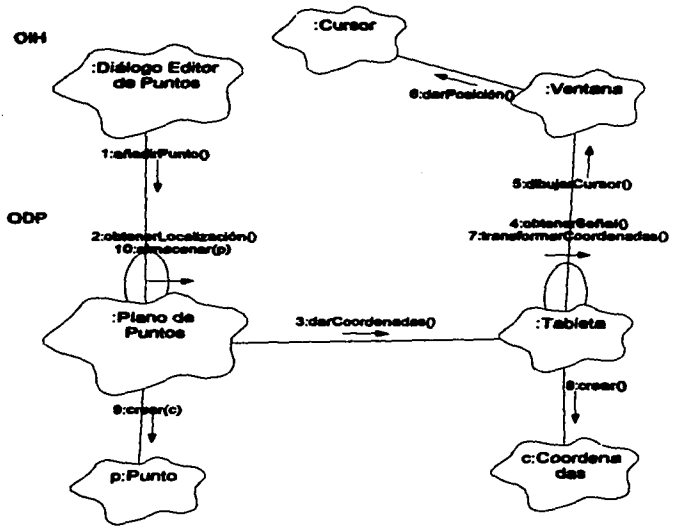

Mapa
Mapa Urbano
Plano de Curvas de Nivel
Plano de Colonias
Plano de Calles
Plano de Lotes
Plano de Manzanas


Segmento
(from Base)
Calle
nombre : Cadena
{trans}
Curva de Nivel
altura : float
nombre
Cadena
Polgono
Lote
{trans}
nombre
Colonia
nombre : Cadena
|| número : int
Manzana
número : int
{trans}

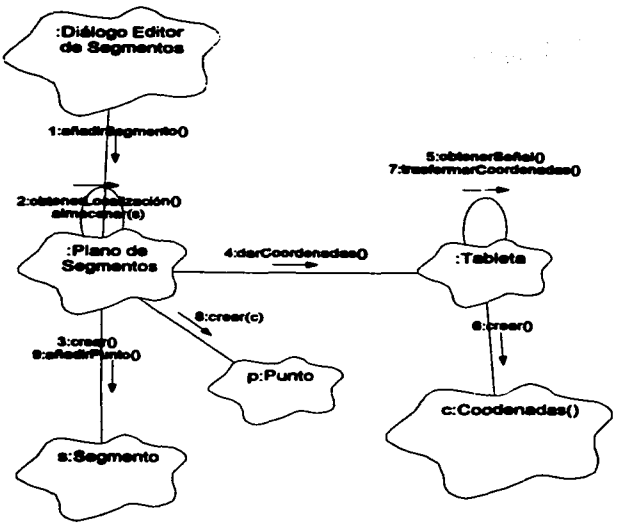

TFrame Window
Ventana Menú
Ventana del Mapa
dibujarPolígono( )
dibujarPunto( )
dibujarSegmento( )
escribirTexto( )
mostrarCursor( )
Cursor
TDialog
Diálog de Excepción
sMensaje1 : TStatic
sMensaje2 : TStatic
Diálogo ConfSisCoord
eliminarPuntoControl( )
insertarPuntoControl( )
Diálogo Editor
almacenar( )
añadir( )
eliminar( )
seleccionar( )
Diálogo Editor de Puntos
eCoordX : TEdit
eCoordY : TEdit
planoPuntos : Plano de Puntos
Diálogo Editor de Polígonos
sPerímetro : TStatic
planoPolígonos : Plano de Polígonos
sArea : TStatic
Diálogo Editor de Segmento
sLongitud : TStatic
eNúmeroPuntos : TStatic
planoSegmentos : Plano de Segmentos



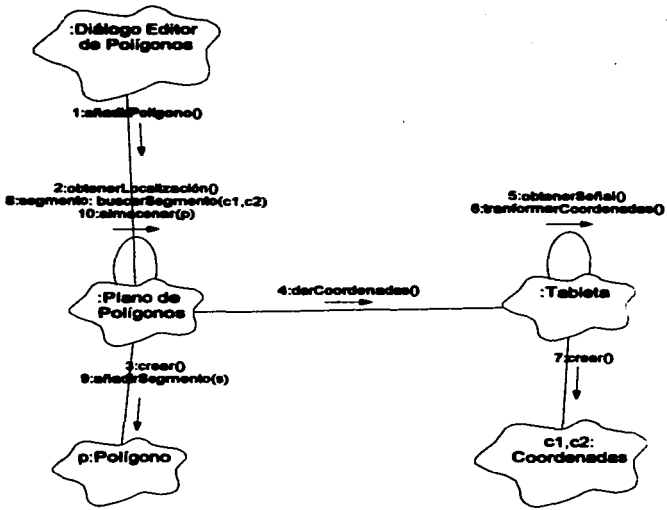
Fits: Sat Apr 19 19:18:59 1997 Class Diagram: Interacción Humana / Modelo del sistema Digitalizador de Mapas Urbanos

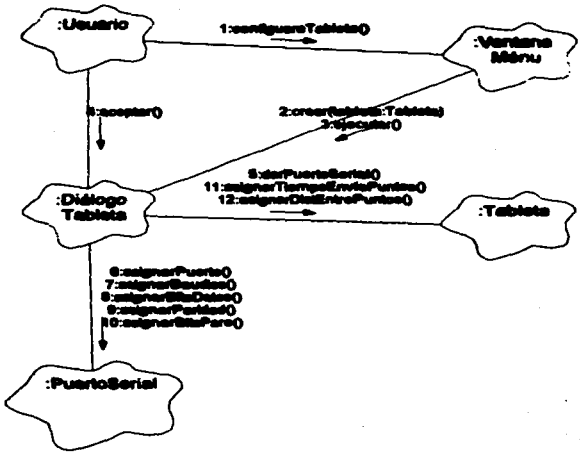


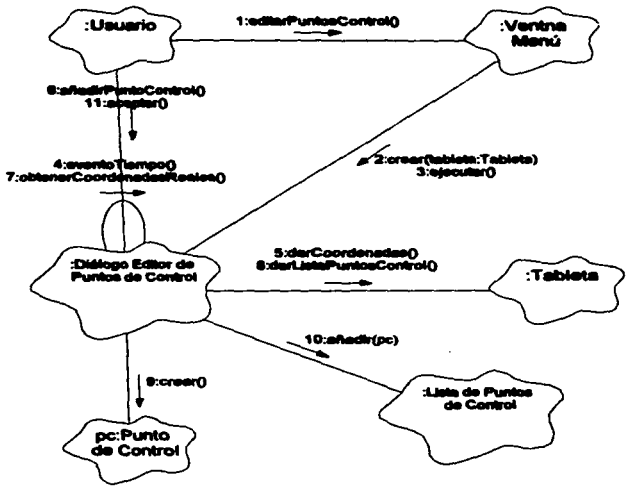
File: Sat Apr 18 20:03:47 1997 Object Diagram: Planos Base / Añadir Punto

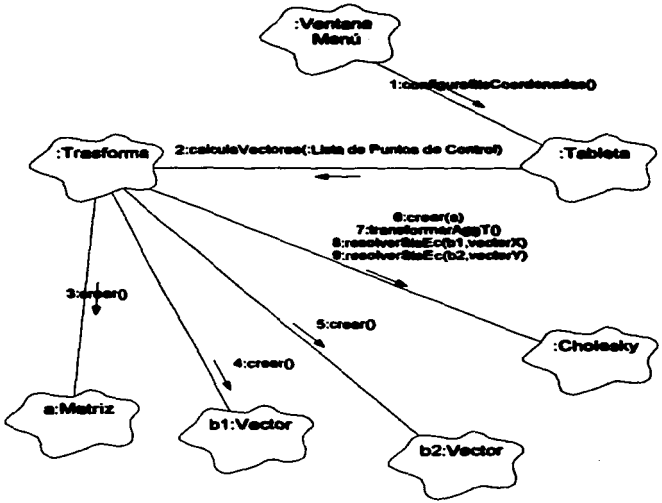


File: Sat Apr 19 20:08:39 1997 Object Diagram: Plano Base / Añadir Segmento

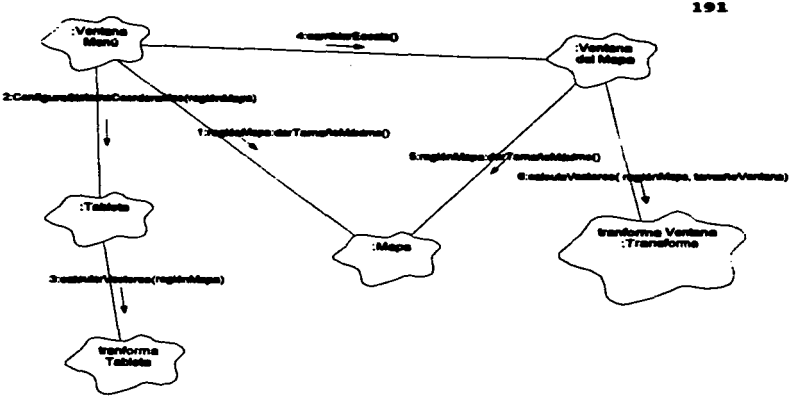

:Diálogo Editor de Polígonos
1:añadirPolígono()
2:obtenerLocalización()
8:segmento: buscarSegmento(c1,c2)
10:almacenar(p)
5:obtenerSeñal()
6:tranformarCoordenadas()
:Plano de Polígonos
4:darCoordenadas()
:Tableta
3:crear()
9:añadirSegmento(s)
7:crear()
p:Polígono
c1,c2: Coordenadas


:Usuario
1:configurarTableta()
:Ventana Menu
4:aceptar()
2:crear(tableta:Tableta)
3:ejecutar()
5:darPuertoSerial()
11:asignarTiempoEnvioPuntos()
12:asignarDistEntrePuntos()
:Dialogo Tableta
:Tableta
6:asignarPuerto()
7:asignarBaudios()
8:asignarBitsDatos()
9:asignarParidad()
10:asignarBitsPare()
:PuertoSerial

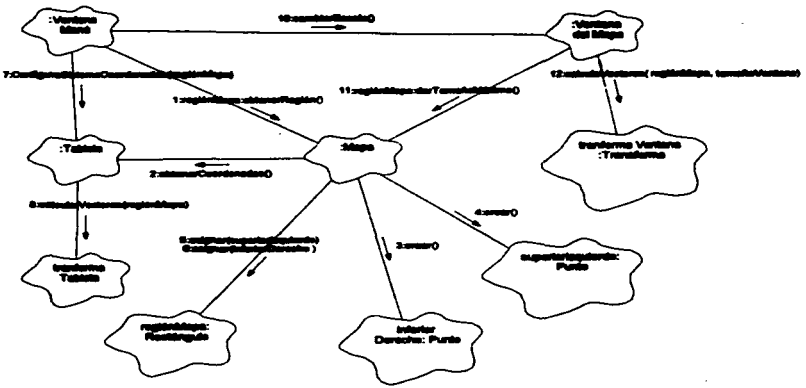

:Usuario
1:editarPuntosControl()
:Ventna Menú
8:añadirPuntoControl()
11:ejecutar()
4:eventoTiempo()
7:obtenerCoordenadasReales()
2:crear(tableta:Tableta)
3:ejecutar()
:Diálogo Editor de Puntos de Control
5:derCoordenadas()
6:darListaPuntosControl()
:Tableta
10:añadir(pc)
9:crear()
:Lista de Puntos de Control
pc:Punto de Control



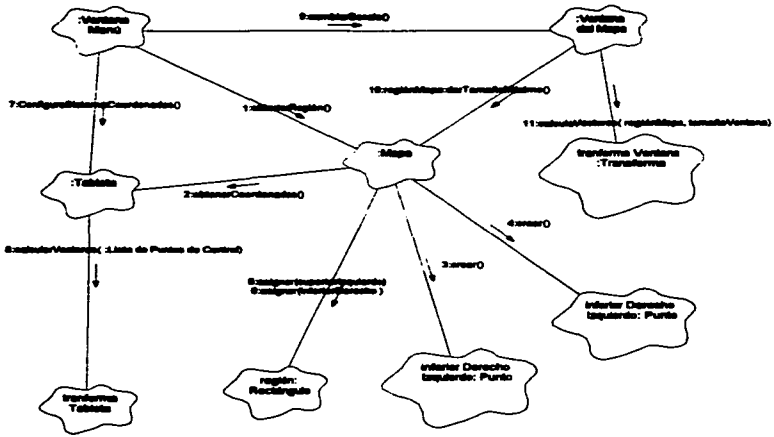
File: Sat Apr 19 19:32:37 1997 Object Diagram: Dispositivo / Configura Sistema de Coordenadas

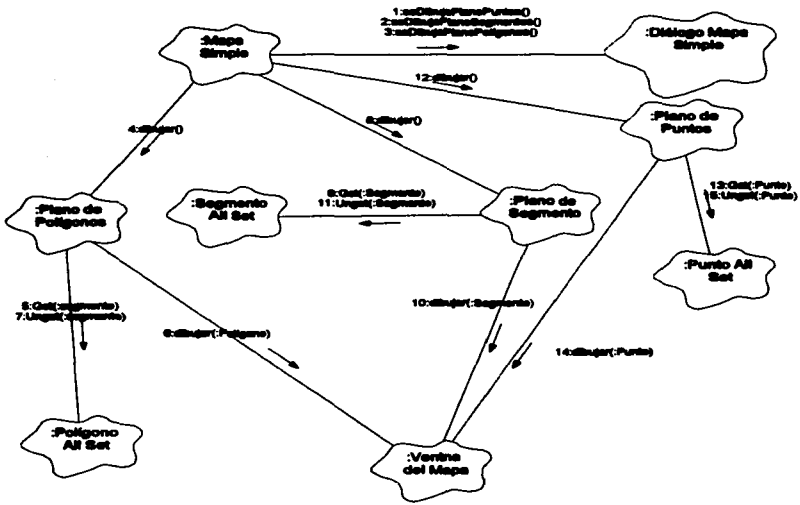


File: Sat Apr 19 19:48:52 1997 Object Diagram: Mapas / Alejandando del Mapa sin Referencia



File: Sat Apr 19 19:40:46 1997 Object Diagram: Mapas / Acercamiento al Mapa sin Referencia


:Ventana Mundo
:Ventana del Mapa
:Mapa
:Tablilla
tranforma Ventana :Transforma
Inferior Derecho Izquierdo: Punto
region: Rectángulo
Inferior Derecho Izquierdo: Punto
tranforma Tablilla


:Mapa Simple
:Dialogo Mapa Simple
:Plano de Puntos
:Plano de Poligonos
:Segmento All Set
:Plano de Segmento
:Punto All Set
:Poligono All Set
:Ventna del Mapa
4:dibujar()
8:dibujar()
9:Get(:Segmento)
11:Urget(:Segmento)
13:Get(:Punto)
15:Urget(:Punto)
10:dibujar(:Segmento)
6:dibujar(:Poligono)
14:dibujar(:Punto)

Bibliografía

Referencias bibliográficas sobre cartografía y sistemas de información geográfica:

Alonso, F. 1986. *Apuntes de Cartografía*. Facultad de Ingeniería. Universidad Nacional Autónoma de México.

Amidon, E. L. 1964. *A Computer-Oriented System for Assembling and Displaying Land Management Information*. U.S. Forest Servic Research Paper PSW-17, Berkeley, California.

Bentley, J. L. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*. Vol 18, Num 19, Sept. 1975, pp. 509-517.

Bonham-Carter, G. F. 1994. *Geographic Information Systems for Geoscientists: Modelling with GIS*. Pergamon/Elsevier Science Publications.

Burrough, P.A. 1986. *Principles of Geographical Information Systems for Earth Resources Assessment*. Oxford: Clarendon Press.

Cliff, A. D. and Ord, J. K. 1981. *Spatial processes: models and applications*. Pion, London.

Cook, B. G. 1978. The structural and algorithmic basis of a geographic data base. *Harvard papers on geographic information systems: First International Advanced Study Symposium on Topological Data Structures for Geographic Information systems* (ed. G. Dutton) Vol. 4. Laboratory for Computer Graphics and Spatial Analysis. Graduate School of Design. Harvard University.

Cook, B. G.1983. An introduction to the design of geographic database. *In Proc. Workshop on Databases in the Natural Sciences*, CSIRO Division of Computing Research 7-9 Sept. 1983, pp. 175-186. Cunningham Laboratory, Brisbane, Queensland.

Deuker, K. J. 1979. Land Resource Information Systems: Spatial and Attribute Resolution Issues. *Proceedings, Int. Symposium on Cartography and Computing:* Auto-Car IV, Vol. 2, pp. 328-336

Fisher, H. T. 1978. Thematic cartography-what is and what is different about it. *Harvard paper in theoretical cartography*. Laboratory for Computer Graphics and Spatial Analysis. Harvard.
Geits, G. M. 1969. Thematic Mapping by Computer. *Cartographic Journal*. Vol. 6, No. 1, pp. 50-68.

Green, R.1964. *The Storage and Retrieval of Data for Water Quality Control*. Public Health Service Publication No. 1263, U.S. Department of Health, Education, and Wealfare, Public Health Service. Washington, D.C.

Hodgkiss. A. G. 1981. *Understanding maps*. UK: Dawson Folkestone..

Journel, A. G. and Huijbregts, Ch. J. 1978. *Mining geostatistics*. London: Academic Press.

Peuker, T. K. and Chrisman, N. 1975. Cartographic data structures. *American Cartographer*, Vol. 2, No. 1, pp. 55-69.

Peuquet, D. J. 1977. *Raster Data Handling in Geographic Information Systems*. Buffalo, New York: Geographic Information Systems Laboratory, State University of New York.

Ripley, B. D. 1981, *Spatial Statistics*. New York: John Wiley and Sons.

Robinson, A., Sale, R. and Morrison, J. 1978. *Elements of Cartography*. 4th edition. New York: John Wiley and Sons.

Sorani, V., Alvarez, R., Baca, J. C., and Varela, S. 1993. The Forest Inventory of México, Phase1: Classification with TM Imagery. *Presented at the 25th International Symposium, Remote Sensing and Global Environmental Change, Graz, Austria*, Vol. 2, pp. 423-434.

Star, J. L. and Ester, J. E.. 1990. *Geographic Information Systems*. Englewood Cliffs, New Jersey: Prentice-Hall.

Teicholz, E. and Berry, B. J. L. 1983. *Computer Graphics and Environmental Planning*. Englewood Cliffs, New Jersey: Prentice-Hall.

Tomlinson, R. F., Calkins, H. W., and Marble, D. F. 1976. Computer Handling of Geographic Data. UNESCO. Geneva.

Tomlinson R. F. 1982. Panel Discussion: Technology Alternatives and Technology Tranfer. *In Computer Assisted Cartography and Geographic Information Processing, Hope, and Realism*, Douglas and Boyle, eds. Canadian Cartographic Association, Dept. of Geography, University of Ottawa, pp.65-71.

Van Roessel, J.W., 1987. Design of a Spatial Data Structure Usign Relational Normal Form. *International Journal of Geographical Information Systems*, Vol. 1, No. 11, p. 33-50.

Webster, R.1977. *Quantitative and Numerical Methods for Soil Survey and Classification*. Oxford: Oxford University Press.

Referencias bibliográficas sobre sistemas orientados a objetos:

Booch, G. 1993. *Object-Oriented Analysis and Design with Applications*. The Benjamin/Cummings Publishing Company, Inc.

Coad, P. and Yourdon, E. 1990. *Object-Oriented Analysis*. Englewood Cliffs, New Jersey: Prentice-Hall.

Coad, P. and Nicola, J. 1993. *Object-Oriented Programming*. Englewood Cliffs, New Jersey: Yourdon Press.

Coad, P., North D., and Mayfield, M. 1995. *Object Models Strategies, Patterns, & Applications*. Englewood Cliffs, NJ: Prentice-Hall.

Coplien, J. 1992. *Advanced C++ Programming Styles and Idioms*. Reading, Massachusetts: Addison-Wesley Publishing Company.

Druke, M. 1989. Comunicación Privada a Grady Booch en 1993.

Ghezzi, C., Jazayeri, M. and Mandrioli, D. 1991. Fundamentals of Software Engineering. Englewood Cliffs, NJ: Prentice-Hall.

Katrib, M. 1994. *Programación Orientada a Objetos en C++*. México D.F.: Infosys.

Meyer, B. 1988. *Object-Oriented Software Construction*. New York, NY: Prentice Hall.

Rubin, K. and Golberg, A. 1992. *Object Behavior Analysis*. Communication of the ACM vol. 35(9).

Shlaer, S. and Mellor, S. 1988. *Object-Oriented Systems Analysis: Modeling the World in Data*. Englewood Cliffs, NJ: Yourdon Press.

Stroustrup, B. 1991. *The C++ Programming Language*, Second Edition. Reading, Massachusetts: Addison-Wesley Publishing Company.

Wirfs-Brock, R., Wilkerson, B., and Wiener, L. 1990. *Designing Object-Oriented Software*. Englewood Cliffs, New Jersey : Prentice Hall.